Scheduling resource-constrained projects with a flexible project structure

Carolin Kellenbrink, Stefan Helber[a]

Department of Production Management
Leibniz Universität Hannover
Königsworther Platz 1, D-30167 Hannover, Germany
carolin.kellenbrink@prod.uni-hannover.de, +49 511 7628002
stefan.helber@prod.uni-hannover.de, +49 511 7625650

[a] *Corresponding author*

Abstract

In projects with a flexible project structure, the activities that have to be scheduled are not completely known beforehand. Instead, scheduling such a project includes the decision whether to carry out particular activities at all. This also effects precedence constraints between the finally implemented activities. However, established model formulations and solution approaches for the resource-constrained project scheduling problem (RCPSP) assume that the project structure is given in advance. In this paper, the traditional RCPSP is hence extended by a highly general model-endogenous decision on this flexible project structure. This is illustrated by the example of the aircraft turnaround process at airports. We present a genetic algorithm to solve this type of scheduling problem and evaluate it in an extensive numerical study.

Keywords: Project scheduling, Genetic algorithms, RCPSP, Flexible projects

1. Introduction

In the classical resource-constrained project scheduling problem (RCPSP), the project structure is given exogenously, i.e., all activities as well as all precedence constraints are known and all activities have to be implemented. In this paper, the RCPSP is extended by a model-endogenous decision on the project structure. In case of projects with a flexible project structure, scheduling includes the decisions whether to implement specific optional activities at all and to impose the related precedence constraints.

Due to these constraints, any predecessor activity has to be finished before a directly succeeding activity can be started. In addition, resource constraints must be considered for renewable and/or non-renewable resources. Renewable resources, e.g., machines or human resources, are available with a given quantity in each period. In contrast, non-renewable resources can be consumed once over the entire planning horizon. An example of this latter kind of resource is a budget that can be spent for the whole project. The typical aim of the RCPSP is to create a schedule which minimizes the total makespan of the project, i.e., the completion time of the last activity. The established multi-mode extension of the RCPSP (MRCPSP, cf., e.g., Talbot (1982)), can be interpreted as a special case of the problem studied in this paper. In the MRCPSP, each activity can be performed in one or more alternative modes out of which one has to be chosen while all the precedence constraints have to be respected irrespective of the chosen activity modes. In our approach, we would introduce a specific activity corresponding to each mode of the MRCPSP and impose the same set of precedence constraints. For this reason, the MRCPSP is included in the problem studied in this paper. However, as will be seen below, the modeling flexibility achieved by our approach notably exceeds those of the MRCPSP.

The remainder of this paper is organized as follows: In Section 2 the assumptions for the resource-constrained project scheduling problem with model-endogenous decision on the project structure (RCPSP-PS) are stated and a practical example is given. In Section 3, we develop a mathematical model for the RCPSP-PS. The genetic algorithm to solve the problem is presented in Section 4. We report numerical results in Section 5. The paper ends with some conclusions and suggestions for further research in Section 6.

2. Problem statement

2.1. Projects with a flexible project structure

In projects with a flexible project structure, decisions have to be made whether to implement specific activities and to impose specific precedence constraints. This leads to the question on how to model this flexibility of the project structure.

Even in flexible projects, some activities as well as the precedence constraints between those activities may be mandatory, i.e., they have to be implemented in any case, as in a classical RCPSP. In addition, in order to develop the RCPSP-PS, we assume that

(i) choices between alternative activities have to be made, (eventually)

(ii) causing the (non-)implementation of further activities, and/or

(iii) activating further choices.

Note that due to assumption (ii) and (iii), such a RCPSP-PS with a flexible structure differs from the multi-mode MRCPSP with its rigid project structure. In the RCPSP-PS, a set of *potential* precedence constraints is defined in the same manner as the precedence constraints in the RCPSP. However, such a potential precedence constraint is only enforced if both the preceding and the succeeding activity connected via this constraint are actually implemented in a schedule. In this schedule, the starting and finishing times of the implemented activities are determined. Thus, the timing of the implemented activities and the decision on the project structure depend on each other. In addition, the decision about the project structure addressing topics (i) to (iii) can itself have a combinatorial character. Therefore, it is not reasonable to separate these two planning steps from each other. Instead, the decision about the project structure as well as the timing of activities should be made model-endogenously and simultaneously.

Such flexible projects can describe, e.g., the passenger aircraft turnaround process at an airport, which is explained in the next subsection in order to illustrate the problem from a practical perspective. However, flexible projects can also be found in other fields like the often highly individual regeneration of complex capital goods such as aircraft engines. For a given wear pattern, alternative regeneration processes may be legally possible within the regulations of the airworthiness authorities and the engine manual of the engine producer. This leads to a flexible project structure in which alternative activities reflect alternative ways to regenerate the aircraft engine.

2.2. Practical example: The aircraft turnaround process

The aircraft turnaround process consists of the steps a passenger aircraft passes through between its arrival at an airport and its next departure. This turnaround process can be interpreted as a (small) project and it can be organized in different ways, cf. Kuster et al. (2009).

Table 1 presents a strongly simplified version of the flexible turnaround process. Some activities are mandatory, e.g., cleaning the aircraft, catering and boarding. There is a choice between alternative arrival options, which affects the (de-)boarding of the aircraft. The aircraft can arrive either at the apron of the airfield or at the terminal. If the aircraft arrives at the terminal, this choice activates two other activities: First, the deboarding of passengers has to be carried out by a bridge. Second, a push-back operation has to be performed by a tow vehicle in order to transport the aircraft back to the airfield. If, by contrast, the aircraft arrives at the apron, this causes another choice related to the deboarding: After leaving the aircraft via stairs, the passengers can (in principle)

Modeling aspect	Example(s)
Mandatory implementation of some activities	Cleaning the aircraft
	Catering
	Boarding
Choices between alternative activities	Arrival at the apron of the airfield **or** at the terminal
	Deboarding by foot **or** by bus
	Fueling with **or** without fire fighters
Activities caused by choices made	Arrival at the terminal **causes** deboarding by bridge
	Arrival at the terminal **causes** push-back
Choices caused by choices made	Arrival at the apron of the airfield **causes** choice on deboarding mode
Precedence constraints caused by optional activities	Fueling without supervision by fire fighters must be completed before boarding can start

Table 1: Modeling aspects of the flexible project structure for a turnaround

either walk to the terminal building or be transported by a bus. Finally, there is a choice concerning the fueling. If fire fighters supervise the fueling, passengers may board the aircraft while it is being fueled. Without the supervision of fire fighters, fueling has to be completed before passengers may board the aircraft. (To simplify the example, we intentionally abstract from the different boarding operations or possible aircraft relocation operations.) This tiny and simplified example of a flexible project already contains all the problem aspects introduced in Subsection 2.1. We will come back to the example in Subsection 3.1.

2.3. Related literature

A broad body of literature on resource-constrained project scheduling exists. An extensive literature survey is given by Hartmann & Briskorn (2010) as well as Kolisch & Padman (2001) and Brucker et al. (1999). Thus, only the most important research publications regarding alternative modes of implementation of activities are addressed here.

In the multi-mode extension of the RCPSP, the MRCPSP (cf., e.g., Talbot (1982)), different modes of implementation can be available for an activity. While the capacity load and the duration of the activity vary over these different modes, each activity still has to be implemented. Thereby, the precedence relations are fixed as well as the set of activities to be implemented, i.e., the project structure itself does not vary in the MRCPSP. Another difference to the RCPSP-PS studied in this paper is the fact that the

modes of the MRCPSP are chosen independently, i.e., a mode chosen for one activity does not imply a specific mode for another activity.

Tiwari et al. (2009) extended the MRCPSP by rework activities. Rework is necessary if the original activity is implemented in a specific predefined mode. In this approach, rework always consists of only a single activity, which is a direct successor of the original activity causing the rework activity. That makes the project structure vary indeed, but only to a very limited extent as only single rework activities can be activated.

Belhe & Kusiak (1995) present a so-called "design activity network" in order to include logical dependencies among some activities. In case of a logical "or"-dependency, a decision has to be made which branch in the network is implemented. Čapek et al. (2012) also examine alternative process plans. In contrast to the RCPSP-PS, these approaches assume that a logical dependency between activities always goes along with a precedence constraint between these activities. By the means of their design activity network, Belhe & Kusiak (1995) generate all possible precedence networks and analyze them separately via a (potentially time-consuming) full enumeration.

The notion of the project structure not being known in advance has also be treated in the context of stochastic project networks, cf. Neumann (1990). There, and in contrast to this paper, the decision on the implementation of activities depends on random exogenous influences. Therefore, the developed methods for stochastic project networks consider a problem class different from the RCPSP-PS treated in this paper.

Kuster et al. (2009) and Kuster et al. (2010) address disruption management problems at airports with alternative process implementation paths. The basic assumptions are similar to the RCPSP-PS. However, their approach only treats the rescheduling problem, but does not address the question of how to create a new schedule in the first place, given the flexibility of the project structure. Furthermore, the authors do not present a mathematical model to precisely describe their problem setting.

To the best of our knowledge, the RCPSP-PS has not be treated in the literature before. However, it appears to contain several of the problem settings described above as special cases. A prelimenary introduction to this scheduling problem has been presented in Kellenbrink (2013), but without a formal model or the details of the algorithm as well as the numerical analysis presented in this paper.

3. Formal description of the RCPSP-PS

3.1. Basic assumptions and elements of the modeling approach

The resource constrained project scheduling problem with a flexible project structure (RCPSP-PS) generalizes the established modeling approach of the RCPSP (cf., e.g., Pritsker et al. (1969)). The RCPSP as well as our RCPSP-PS is modeled as an activity-on-node network and built around a central binary variable x_{jt} for activity $j \in \mathcal{J}$ and period $t \in \mathcal{T}$. This variable indicates whether activity j is finished in period t, i.e., $x_{jt} = 1$,

or not, i.e., $x_{jt} = 0$. Using this established variable, the structural decision whether or not to implement an activity at all can also be described implicitly: If an activity j is not implemented at all, the correspondent binary variable x_{jt} is 0 for each period t. For this reason, we can expand the modeling flexibility of the RCPSP without introducing new decision variables. Instead, we use indices and sets of indices to model the project structure flexibility introduced in Subsection 2.1 and exemplified in Table 1 as follows:

- We denote with $e \in \mathcal{E}$ all *choices* that may have to be made.
- The activities $j, i \in \mathcal{J}$ are ordered topologically such that for any pair of activities i and j with $i < j$, there must not be a constraint enforcing activity j to precede activity i.
- The subset $\mathcal{V} \subseteq \mathcal{J}$ contains all activities that are mandatory, i.e., that have to be implemented in any case. The other activities are non-mandatory or optional.
- Each choice e consists of the selection of exactly one activity j out of several optional activities $\mathcal{W}_e \subseteq \mathcal{J} \setminus \mathcal{V}$. Each optional activity can belong to at most one set $\mathcal{W}_e$.
- Each choice e is connected to a triggering activity $a(e)$. If the triggering activity $a(e)$ is mandatory (as is the dummy 'start' activity $i = 1$), choice e is also mandatory, i.e., $a(e) = 1$, and one optional activity $j \in \mathcal{W}_e$ has to be implemented. However, the triggering activity $a(e)$ may itself be optional. In this case, it has to be among the optional activities related to exactly one (earlier) choice e', i.e., $a(e) \in \mathcal{W}_{e'}$ for one e'. In this latter case, it is possible that choice e is not triggered and hence none of the optional activities $j \in \mathcal{W}_e$ is implemented. Each optional activity $j \in \mathcal{W}_e$ can trigger at most one other decision.
- An optional activity $j \in \mathcal{W}_e$ may have a set of (dependent) optional activities $\mathcal{B}_j \subseteq \mathcal{J} \setminus \mathcal{V}$ all of which have to be implemented if the optional activity j itself is implemented. Note that $k \in \mathcal{B}_j$ implies $j < k$, i.e., the dependent activity cannot precede its own trigger activity j.

  Each optional activity either belongs to one set $\mathcal{W}_e$ or to one set $\mathcal{B}_j$ with $j \in \mathcal{W}_e$.
- Any choice e can only be triggered by an activity $a(e)$ which must not have to succeed the optional activities $j \in \mathcal{W}_e$ in order to avoid cycles of cause and effect. In this case, the topological ordering of the activities can be such that the condition $a(e) < j$ holds for each choice e and each optional activity $j \in \mathcal{W}_e$.
- Like the activities, the choices e are also ordered topologically such that for any pair of choices e' and e with $e' < e$, choice e must not trigger choice e'.
- A precedence constraint is implemented whenever it connects two implemented activities.

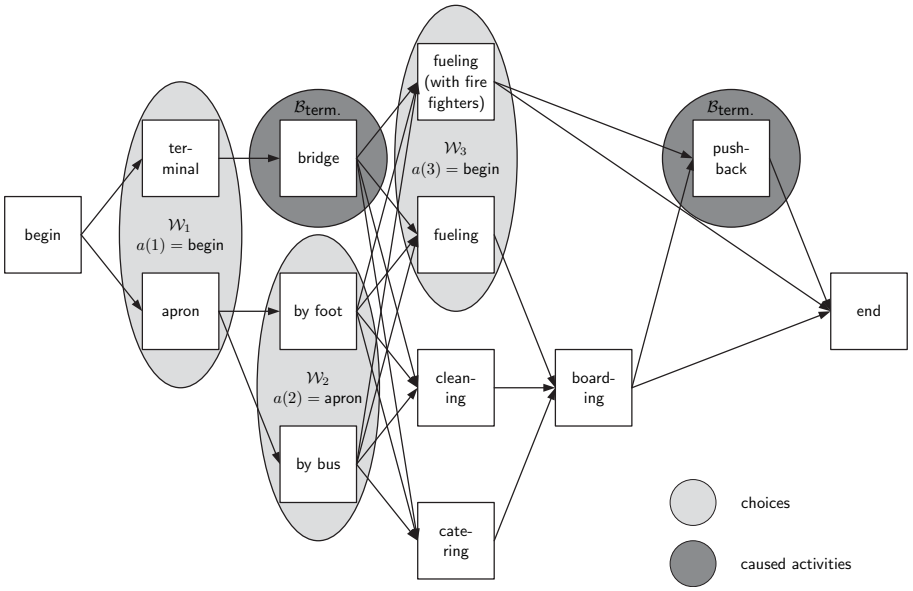

Figure 1: Project network for a simplified turnaround at an airport

We use the example introduced in Subsection 2.2 to show the use of these sets and indices. To this end, we also introduce a specific graphical representation based on an activity-on-node network. Each choice e is represented as an oval around its optional activities $j \in \mathcal{W}_e$. Activities caused by other activities are specifically marked. Note that the established graphical representations of project networks are not suitable to represent projects with a flexible project structure, cf. Kuster et al. (2009), p. 244. Even the graphical evaluation and review technique (GERT, cf., e.g., Neumann (1990), pp. 17-19) used for stochastic project networks is not appropriate, because it is not possible to represent logical dependencies and precedence constraints independently. In contrast to the RCPSP-PS, a logical dependency between activities in a GERT network always goes along with a precedence constraint between these activities.

The project network for the turnaround example is presented in Figure 1. The set of mandatory activities is $\mathcal{V} = \{\text{begin}, \text{cleaning}, \text{catering}, \text{boarding}, \text{end}\})$. Choice 1 is the mandatory choice between alternative activities concerning the arrival ($\mathcal{W}_1 = \{\text{terminal}, \text{apron}\}$). As it is mandatory, choice $e = 1$ is triggered by the beginning activity, i.e., $a(1) =$ 'begin'. The arrival at the terminal causes deboarding by bridge and the push-back ($\mathcal{B}_{\text{terminal}} = \{\text{bridge}, \text{push-back}\}$). The optional choice 2 is caused by the decision to arrive at the apron $a(2) =$ 'apron' and determines how to transport deboarding passengers from the apron to the terminal. The optional activities for $e = 2$ are $\mathcal{W}_2 = \{\text{by foot}, \text{by bus}\}$. Finally, there is a mandatory choice $e = 3$ with $a(3) =$ 'begin' concerning the fueling ($\mathcal{W}_3 = \{\text{fueling(with fire fighters)}, \text{fueling}\}$).

Indices and (ordered) sets	
$a(e)$	single activity causing choice e
$i \in \mathcal{B}_j$	activities caused by activity j, with $i, j \notin \mathcal{V}$
$e \in \mathcal{E}$	choices, with $e = 1, ..., E$
$j, i \in \mathcal{J}$	topologically ordered activities, with $j, i = 1, ..., J$
$r \in \mathcal{N}$	non-renewable resources
$i \in \mathcal{P}_j$	predecessors of activity j
$r \in \mathcal{R}$	renewable resources
$t, \tau \in \mathcal{T}$	periods, with $t, \tau = 1, ..., T$
$j \in \mathcal{V}$	mandatory activities, including (dummy) activities 1 and J
$j \in \mathcal{W}_e$	optional activities of choice e

Parameters	
d_j	duration of activity j
k_{jr}	capacity consumption of resource r by activity j
K_r	capacity of resource r

Decision variable	
x_{jt}	$\begin{cases} 1, & \text{if activity } j \text{ is finished in period } t \\ 0, & \text{otherwise} \end{cases}$

Table 2: Notation of the RCPSP-PS

3.2. Mathematical model

The presented modeling approach of introducing new sets $\mathcal{E}$, $\mathcal{V}$, $\mathcal{W}_e$ and $\mathcal{B}_j$ as well as indices without changing the decision variable x_{jt} has the effect that only limited modifications of the RCPSP are necessary to formally establish the RCPSP-PS, cf. Klein (2000), pp. 79-80. It is presented below using the notation in Table 2.

Model RCPSP-PS

$$\min Z = \sum_{t=1}^{T} t \cdot x_{Jt} \tag{1}$$

subject to

$$\sum_{t=1}^{T} x_{jt} = 1 \qquad j \in \mathcal{V} \tag{2}$$

$$\sum_{i \in \mathcal{W}_e} \sum_{t=1}^{T} x_{it} = \sum_{t=1}^{T} x_{a(e),t} \qquad e \in \mathcal{E} \tag{3}$$

$$\sum_{t=1}^{T} x_{it} = \sum_{t=1}^{T} x_{jt} \qquad e \in \mathcal{E}; \quad j \in \mathcal{W}_e; \quad i \in \mathcal{B}_j \tag{4}$$

$$\sum_{t=1}^{T} t \cdot x_{it} \leq \sum_{t=1}^{T} (t - d_j) \cdot x_{jt} + T \cdot (1 - \sum_{t=1}^{T} x_{jt}) \qquad j \in \mathcal{J}; \quad i \in \mathcal{P}_j \qquad (5)$$

$$\sum_{j=1}^{J} k_{jr} \cdot \sum_{\tau=t}^{t+d_j-1} x_{j\tau} \leq K_r \qquad r \in \mathcal{R}; \quad t \in \mathcal{T} \qquad (6)$$

$$\sum_{j=1}^{J} k_{jr} \cdot \sum_{t=1}^{T} x_{jt} \leq K_r \qquad r \in \mathcal{N} \qquad (7)$$

In the objective function (1), the completion time of the last activity J and hence the makespan of the project is minimized. Constraints (2) - (5) model the flexible project structure. Equation (2) declares that each mandatory activity $j \in \mathcal{V}$ has to be implemented once.

Equation (3) reflects the choices $e \in \mathcal{E}$ on the project structure. If choice e is triggered by the implementation of an activity $a(e)$, exactly one activity $i \in \mathcal{W}_e$ has to be chosen for implementation. Note that none of the optional activities $i \in \mathcal{W}_e$ is implemented if choice e is not triggered. Equation (4) treats the implemented non-mandatory activities caused by other implemented non-mandatory activities. If activity $j \in \mathcal{W}_e$ of choice e is implemented, each activity $i \in \mathcal{B}_j$ has to be implemented as well.

Constraint (5) states that a predecessor $i \in \mathcal{P}_j$ has to be finished before the implementation of the successive activity j can be started. Due to the second summand on the right-hand side of the constraint, this constraint can only be binding if both predecessor i as well as successor j are implemented.

Constraint (6) ensures that the capacity consumption k_{jr} of all activities that affect period t does not exceed the available capacity K_r of any renewable resource $r \in \mathcal{R}$. In addition, constraint (7) states the capacity constraints of the non-renewable resources $r \in \mathcal{N}$.

Note that in the standard RCPSP with its exogenous project structure, it is possible to determine earliest and latest starting and ending times of all activities via simple forward and backward recursions by ignoring capacity constraints. In the context of our RCPSP-PS, this is not easily possible as the project structure is not known beforehand. For this reason, the summations, e.g., in the objective function (1), run over *all* periods $t = 1, ...T$. If the set $\mathcal{E}$ of choices is empty, constraints (3) and (4) disappear and the classical RCPSP results.

3.3. Structural characteristics of the RCPSP-PS

We introduce in Figure 2 a simple project consisting of 10 activities. In this project, a mandatory choice has to be made between the implementation of activity 4 and activity 5. If (and only if) activity 4 is chosen, activity 9 is implemented. If (and only if) activity 5 is chosen, decision 2 is triggered. The optional choice 2 is about the implementation of activity 7 or activity 8.

Three different project structures with specific sets of implemented activities $\mathcal{A} = \{1, 2, 3, \mathbf{4}, 6, \mathbf{9}, 10\}$, $\mathcal{B} = \{1, 2, 3, \mathbf{5}, 6, \mathbf{7}, 10\}$, and $\mathcal{C} = \{1, 2, 3, \mathbf{5}, 6, \mathbf{8}, 10\}$ exist for this project. In addition to the mandatory activities 1, 2, 3, 6, and 10, in project structure A the optional activities 4 and 9 are implemented. In project structures B and C the optional activity 5 is chosen in the mandatory decision 1 and thereby decision 2 is triggered. In this decision, in project structure B, activity 7 is implemented and in project structure C, activity 8 is chosen.

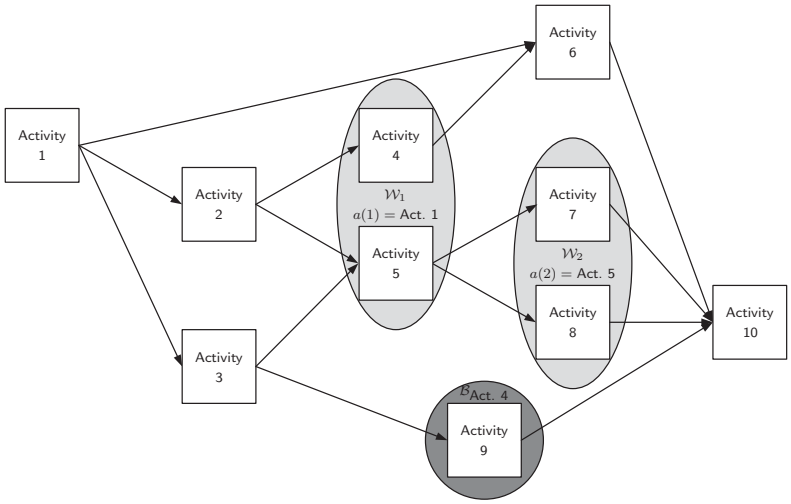

Figure 2: Example of a flexible project

j	1	2	3	4	5	6	7	8	9	10
d_j	0	3	4	3	5	6	4	2	2	0
k_{j1}	0	3	7	5	2	8	6	5	4	0

Table 3: Data for the example project

In Table 3, the duration and resource consumption of a single renewable resource is given. Non-renewable resources do not exist.

We present in Figure 3 the variation of the optimal makespan of this project depending on the capacity of the renewable resource K_1. An increase of K_1 leads to a decrease of the makespan as well as to a change of the optimal project structure. Similar effects can be shown for non-renewable resources.

These characteristics show that the decision on the project structure should be treated together with the scheduling decision since the capacity restrictions affect the selection of the optimal project structure.

3.4. RCPSP-PS vs. full enumeration over multiple RCPSP

For each *given* decision on the project structure, the RCPSP-PS reduces to the RCPSP, so that this problem can in principle be solved via a full enumeration over all possible

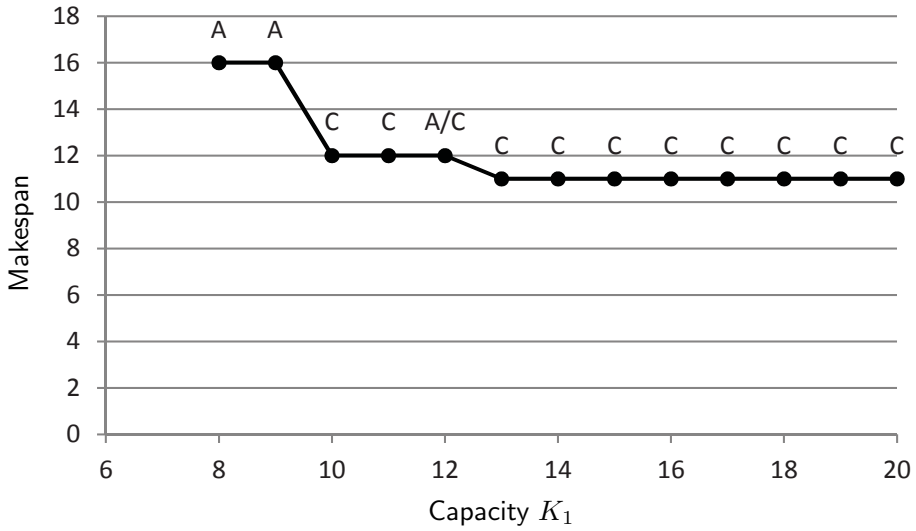

Figure 3: Optimal makespan depending on the capacity of the renewable resource

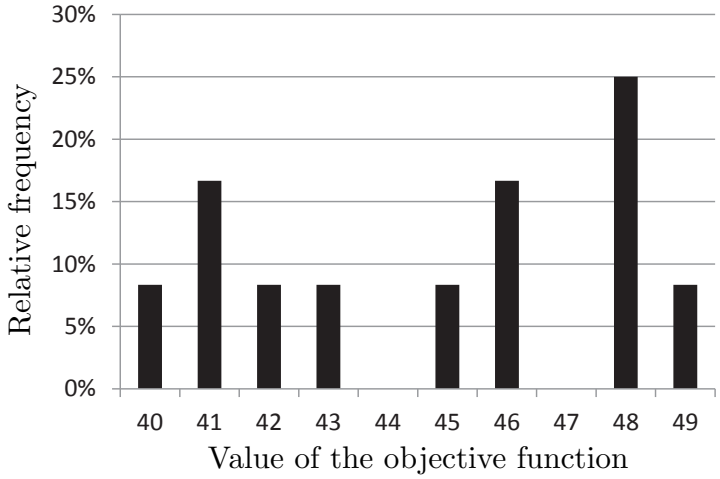

Figure 4: Relative frequency of the values of the objective function for all possible project structures

project structures. However, this does not appear to be a very efficient approach. We experimented with a somewhat larger but still small example with only 30 activities, but already 12 alternative project structures due to a few endogenous choices.

The computation time to solve the problem RCPSP-PS to optimality was only about 9% of the time required to optimally solve all 12 RCPSPs with given project structure. Furthermore, Figure 4 presents the distribution of the optimal objective function values over those 12 different project structures, determined using CPLEX. The variance is apparently large, so it is important find the optimal project structure.

4. Genetic algorithm

4.1. Concept and representation of solutions

The RCPSP-PS includes as a special case the RCPSP, which is already known to be $\mathcal{NP}$-hard, see Blazewicz et al. (1983), p. 21. For this reason, we have to resort to heuristic approaches in order to solve large problem instances. Among the most powerful algorithms for RCPSPs are genetic algorithms and we hence present a genetic algorithm for this new problem class.

Genetic algorithms mimic evolutionary processes in biology. Evolution is seen as the attempt of any species to adapt to its environment by letting members of its population mate with one another in order to produce potentially "fitter" offspring with modified characteristics and better chances of survival. This process is interpreted as a general population-based approach to solve an optimization problem that can be mimicked to solve mathematical optimization problems. In this solution approach, each individual within the population represents a solution of the underlying problem. The genetic algorithm starts with the initialization of a start population. Afterwards, over the course of several generations of populations, new individuals are generated by crossover, mutation and selection, cf. Goldberg (1989).

The genetic algorithm for the RCPSP-PS is a modification and generalization of the genetic algorithm for the MRCPSP presented in Hartmann (2001). In order to extend that approach by the model-endogenous decision on the project structure, the representation of the individuals, the initialization of the population, the crossover and the mutation as proposed in Hartmann (2001) had to be generalized and modified. As it is the basis for all further modifications, we start with the explanation of the representation of individual solutions of the RCPSP-PS.

Any solution of the RCPSP-PS consists of two components. The first one addresses the chosen *structure* of the implemented network of activities, i.e., the ensemble of actually implemented optional activities $j \in \mathcal{W}_e$ for all the choices $e \in \mathcal{E}$ as well as the mandatory activities $j \in \mathcal{V}$ and the related precedence constraints. The second component eventually determines the actual *schedule* of those activities that are actually implemented based on those former choices on the structure. To represent a solution, we hence use a combination of a *choice list* α to reflect the chosen structure and an *activity list* λ supplemented by an *implementation list* ν to (indirectly) determine the actual schedule. The general structure for an individual I

$$I = \left(\begin{array}{c|c} \text{Choice list } \alpha & \text{Activity list } \lambda \\ \text{Auxiliary information} & \text{Implementation list } \nu \end{array} \right)$$

is detailed as follows:

$$I = \left(\begin{array}{cccc|cccc} \alpha_1 & \alpha_2 & ... & \alpha_E & \lambda_1 & \lambda_2 & ... & \lambda_J \\ a(1); \mathcal{W}_1 & a(2); \mathcal{W}_2 & ... & a(E); \mathcal{W}_E & \nu_{\lambda_1} & \nu_{\lambda_2} & ... & \nu_{\lambda_J} \end{array} \right)$$

The choice list $\alpha = (\alpha_1, \alpha_2, ..., \alpha_E)$ contains for each of the choices e the selected optional activity $j \in \mathcal{W}_e$, if any (in case of optional choices). The auxiliary information below the choice list only serves expository purposes in this paper to explain the examples, it is not actually implemented in our algorithm. For each choice e, it contains the triggering activity $a(e)$ as well as the set of optional activities $j \in \mathcal{W}_e$. (Remember that any mandatory choice is triggered by the start activity.)

The activity list $\lambda = (\lambda_1, \lambda_2, ..., \lambda_J)$ contains all activities in the sequence of their treatment in the process of creating a schedule by the means of the serial schedule generation scheme, cf. Kelley (1963), pp. 352-353 or Kolisch & Hartmann (1999), p. 152. Note, however, that due to the choices on the project structure, some optional activities on the activity list are not implemented at all. For this reason, the activity list is supplemented by an implementation list $\nu = (\nu_{\lambda_1}, \nu_{\lambda_2}, ..., \nu_{\lambda_J})$. The elements of the implementation list are coded binary. If the associated activity on the activity list is implemented, the element of the implementation list equals 1, otherwise 0. Note that there is a redundancy of information between the choice list α and the implementation list ν which we deliberately use here to facilitate the explanation of the algorithm.

An activity list is feasible if the sequence of activities on the activity list does not contradict any of the *active* precedence constraints of the project. If an implemented activity i has to precede another implemented activity j due to the precedence constraints, this activity i also has to precede activity j in the activity list. A feasible activity list can be decoded to a unique schedule. In the serial schedule generation scheme, each active element of the activity list is scheduled as early as possible considering the available capacity of renewable resources and the precedence constraints. This method creates a so-called active schedule in which no activity can be started earlier without delaying another activity, cf. Kolisch (1996), p. 325.

In order to explain all central elements of the genetic algorithm, we refer to the example of a flexible project introduced in Subsection 3.3. We consider the second possible project structure with the set $\mathcal{B} = \{1, 2, 3, \mathbf{5}, 6, \mathbf{7}, 10\}$ of implemented activities, which is also shown in Figure 5 (using solid arrows and boxes for implemented activities and precedence constraints). One possible solution to the remaining scheduling problem for this structure is represented by the following individual denoted as I^M:

$$I^M = \left(\begin{array}{cc|cccccccccc} 5 & 7 & 1 & 3 & 6 & 4 & 2 & 8 & 5 & 7 & 9 & 10 \\ 1; \{4,5\} & 5; \{7,8\} & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{array} \right)$$

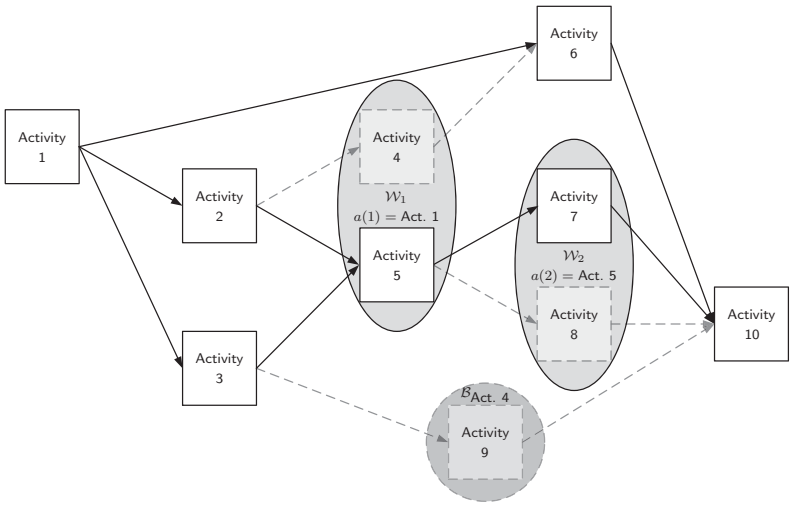

Figure 5: Example of a flexible project with a chosen project structure

The choice list indicates that in the mandatory choice 1, activity 5 is selected. As the optional choice 2 is triggered by activity 5, either activity 7 or 8 has to be implemented and activity 7 is chosen. The implementation list contains the (derived) information that activities 4, 8, and 9 are not implemented. The activity list shows that the implemented activities are treated (within the scheduling process) in the sequence $1, 3, 6, 2, 5, 7$, and 10.

It should be noted that the feasibility of an activity list depends on the content of the choice list and the corresponding implementation list. Assume that in choice 1, the activity 4 had been chosen instead of activity 5. In this case, it would have been necessary to consider an (indirect) precedence relation between activities 2 and 6, cf. Figure 5. The current activity list would violate this constraint and hence be infeasible.

4.2. Fitness computation

Fitness values are computed only for individuals with a feasible activity list. If (according the implementation list) the individual is also feasible with respect to the non-renewable resources, its fitness value is simply the makespan of the project. The lower this fitness value is, the better is the individual. However, if the implemented activities exceed the available capacity of the non-renewable resources, the individual does not represent a capacity-feasible schedule, irrespective of its makespan. In this case, the fitness value is computed as the sum of the length of the planning horizon $T = \sum_{j=1}^{J} d_j$ plus the additionally required capacity of the non-renewable resource(s), as in Hartmann (2001). Therefore, individuals representing infeasible schedules always have a higher and hence worse fitness value than feasible individuals.

To illustrate the fitness computation, we come back to the example introduced in Subsection 3.3. We assume that no non-renewable resource and only one renewable resource with a capacity $K_1 = 10$ is needed and use the additional data in Table 3. The scheduling sequence $1, 3, 6, 2, 5, 7, 10$ from the activity list lead to the schedule in Figure 6.

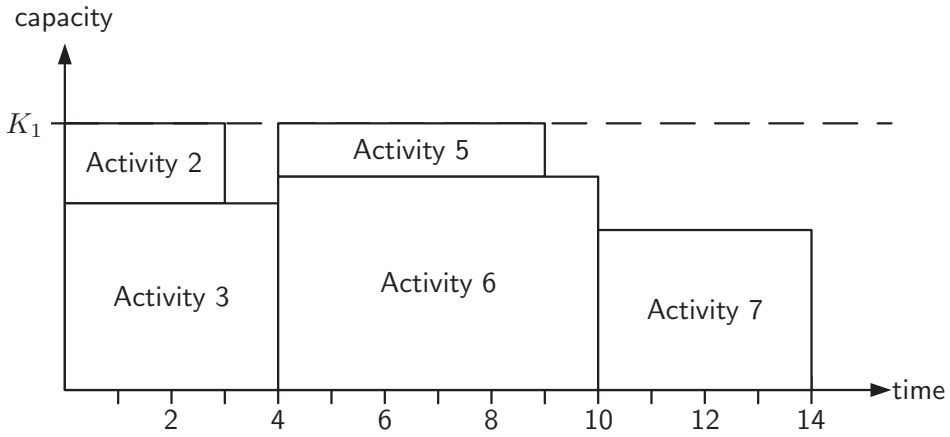

Figure 6: Schedule for I^M

(Note that activities 1 and 10 are "dummy" activities with zero duration and capacity consumption.) Activity 3 is scheduled first starting at time 0 and consumes 7 units of the renewable resource. Due to the limitation of the renewable resource to 10 units, it is not possible to start activity 6 with a consumption of 8 units prior to the completion of activity 3 at time 4. Afterwards, activity 2 is considered. The capacity constraints as well as the precedence constraints allow to start this activity at time 0. The precedence relation between activities 3 and 5 allows to start activity 5 after the end of activity 3 at time 4. The precedence constraints furthermore allow activity 7 to start after the completion of activity 5. However, at this time, there are not enough resource units available and the start of activity 7 has to be postponed until time 10. The makespan of the project and hence the fitness value $F(I^M)$ is 14.

4.3. Generation of the initial population

At the beginning of the genetic algorithm, N^I individuals are generated as a start population. For each individual, a choice list, an activity list as well as an implementation list are determined randomly but consistently. The procedure is shown in Algorithm 1. Due to the fact that the feasibility of the activity list depends on the choice list and, hence, the implementation list, the initialization of the individuals begins with the decision on the structure of the network, i.e., the choice of the optional activities.

First of all, all mandatory activities are activated ($\nu_j := 1,\ j \in \mathcal{V}$). All other activities are—at least initially—deactivated ($\nu_j := 0,\ j \notin \mathcal{V}$). Next, each choice $e \in \mathcal{E}$ is considered in the sequence of the topological ordering of the choices. If choice e is triggered by the mandatory start activity or an already selected optional activity ($\nu_{a(e)} = 1$), one activity $j \in \mathcal{W}_e$ is randomly selected ($\alpha_e := j$) and activated ($\nu_j := 1$). Eventually, all those activities i which are triggered by activated optional activities are activated as well ($\nu_i := 1,\ i \in \mathcal{B}_j$ with $\nu_j = 1$).

The procedure to determine the choice list and, hence, the implemented activities (in Part I of the algorithm) is demonstrated below using the example project introduced in

/* Part I: Determine activities to be implemented */
repeat
  Activate all mandatory activities ($\nu_j := 1,\ j \in \mathcal{V}$)
  Deactivate all optional activities ($\nu_j := 0,\ j \notin \mathcal{V}$)
  for *each choice* $e \in \mathcal{E}$ **do**
    if *choice* e *is triggered* ($\nu_{a(e)} = 1$) **then**
      Randomly select one activity $j \in \mathcal{W}_e$ and set $\alpha_e := j$
      Activate the selected activity ($\nu_j := 1$)
    end
  end
  for *each active activity* j ($\nu_j = 1$) **do**
    Activate all triggered activities i ($\nu_i := 1,\ i \in \mathcal{B}_j$)
  end
until *Capacity constraint is kept for each* $r \in \mathcal{N}$ *or* $Trial_{max}$ *is reached*

/* Part II: Determine sequence of activities */
Place dummy activity 1 at the first position of the activity list
for *positions* 2 *to* $J-1$ *of the activity list* λ **do**
  for *each not yet included (active and inactive) activity except the dummy activity* J **do**
    if *all active predecessors are already included on the activity list* **then**
      Add the activity to the set of eligible activities
    end
  end
  Randomly select one eligible activity considering the latest start times
  Place the selected activity at the current position of the activity list
end
Place dummy activity J at the last position of the activity list

Algorithm 1: Initialization of a new individual

Subsection 3.3. Now a new individual I^F is generated. Firstly, the mandatory activities 1, 2, 3, 6, and 10 are activated ($\nu_1^F = \nu_2^F = \nu_3^F = \nu_6^F = \nu_{10}^F := 1$) and all optional activities are deactivated ($\nu_4^F = \nu_5^F = \nu_7^F = \nu_8^F = \nu_9^F := 0$). In the mandatory decision 1, activity 4 is chosen randomly ($\alpha_1^F := 4;\ \nu_4^F := 1$). Therefore, choice 2 is not triggered ($\nu_5^F = 0$) and none of the optional activities 7 and 8 is activated. Eventually, activity 9 is triggered by the activation of activity 4 ($\nu_9^F := 1$).

Given the information of the activities to be implemented, the capacity consumption of the non-renewable resources can now be computed. If the available capacity is exceeded for any non-renewable resource, a new project structure has to be determined. If the number of trials to compute a feasible selection of activated optional activities exceeds Trial_{max}, the procedure continues with the infeasible selection of activated optional activities (and a prohibitively poor fitness value).

In Part II of the algorithm, the activity list is finally determined. For each position on the activity list, the set of those activities is determined that are eligible to be placed at this position. In this set of eligible activities, implemented as well as not implemented

activities are considered alike. An activity is eligible if it has not yet been placed on the activity list and if all its active predecessors have already been placed on the activity list. A probabilistic dispatching procedure, see Baker (1974), p. 204, as well as Appendix Appendix A, is used to randomly choose one activity from this set for the current position on the activity list.

One possible combination of choice list, activity list and implementation list for the choices described above leads to the following individual I^F:

$$I^F = \left(\begin{array}{cc|cccccccccc} 4 & - & 1 & 2 & 3 & 7 & 4 & 5 & 6 & 8 & 9 & 10 \\ 1;\{4,5\} & 5;\{7,8\} & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array} \right)$$

The corresponding schedule (not shown here) has a makespan of 13 time units.

4.4. Crossover

In the crossover operation of the genetic algorithm, two parent individuals I^M ("mother") and I^F ("father") are combined in order to create two new child individuals I^D and I^S, such that each new individual possesses traits of both parent individuals. The new individuals always combine elements of two distinct individuals from the previous generation. Each individual from the parent generation is chosen exactly once as a mother or a father for the two children. In Algorithm 2, we show how a daughter I^D inherits from (mother) I^M and (father) I^F. A son I^S is created analogously. Only the roles of the mother and the father in the algorithm are exchanged.

As in the initial population, all mandatory activities of the daughter are initially activated ($\nu_j^D := 1,\ j \in \mathcal{V}$) and all optional activities are—initially—deactivated ($\nu_j^D := 0,\ j \notin \mathcal{V}$). A crossover parameter c^α is determined randomly ($1 \leq c^\alpha \leq |\mathcal{E}|$). The first c^α choices are inherited from I^M, which affects both the choice list α^D and the implementation parameter ($\nu_j^D := \nu_j^M,\ j \in \mathcal{W}_e$). The remaining choices are inherited from I^F. Note that a choice e inherited from the mother ($e \leq c^\alpha$) can trigger another choice e' with $c^\alpha < e'$ to be inherited from the father. If that decision e' is not (also) triggered for I^F, the choice $\alpha_{e'}$ and the implementation state is inherited from the mother I^M as well ($\nu_j^D := \nu_j^M,\ j \in \mathcal{W}_{e'}$). Finally, all possibly triggered activities are activated if the triggering activity is implemented ($\nu_i^D := 1,\ i \in \mathcal{B}_j$ if $\nu_j^D = 1$).

The crossover of the activity list λ is taken from Hartmann (2001). Firstly, the crossover parameter for the activity list c^λ is randomly determined ($1 \leq c^\lambda \leq J$). The c^λ first activities are inherited from the mother's activity list λ^M. The sequence of the remaining activities is inherited from the father's list λ^F.

We apply the crossover to the example from Subsection 3.3 with assumed crossover parameters $c^\alpha = 1$ and $c^\lambda = 4$ and create the daughter I^D. The first choice e is inherited from I^M, i.e., activity 5 is activated ($\alpha_1^D := \alpha_1^M = 5$ and $\nu_5^D := 1$). The second decision

/* Part I: Determine activities to be implemented */
Activate all mandatory activities ($\nu_j^D := 1,\ j \in \mathcal{V}$)
Deactivate all optional activities ($\nu_j^D := 0,\ j \notin \mathcal{V}$)
Determine a random crossover position $1 \leq c^\alpha \leq |\mathcal{E}|$
for *choices* $e = 1, ..., c^\alpha$ **do**
  /* Inherit from the mother */
  $\alpha_e^D := \alpha_e^M$
  for *all* $j \in \mathcal{W}_e$ **do**
    $\nu_j^D := \nu_j^M$
  end
end
for *the remaining decisions* $e = c^\alpha + 1, ..., |\mathcal{E}|$ **do**
  if *choice* e *is triggered for the daughter* ($\nu_{a(e)}^D = 1$) **then**
    if *choice* e *is triggered for the father* ($\nu_{a(e)}^F = 1$) **then**
      /* Inherit from the father */
      $\alpha_e^D := \alpha_e^F$
      for *all* $j \in \mathcal{W}_e$ **do**
        $\nu_j^D := \nu_j^F$
      end
    else
      /* Inherit from the mother */
      $\alpha_e^D := \alpha_e^M$
      for *all* $j \in \mathcal{W}_e$ **do**
        $\nu_j^D := \nu_j^M$
      end
    end
  end
end
for *each active activity* j ($\nu_j^D = 1$) **do**
  Activate all triggered activities i ($\nu_i^D := 1,\ i \in \mathcal{B}_j$)
end

/* Part II: Determine sequence of activities */
Determine a random crossover position $1 \leq c^\lambda \leq J$
Inherit the c^λ first activities from the mother's activity list λ^M
for *each activity* j *in the order of the father's activity list* λ^F **do**
  if j *is not yet placed on the daughter's activity list* **then**
    Append activity j to the daughter's activity list λ^D
  end
end

Algorithm 2: Crossover of I^M and I^F for the generation of I^D

should (in principle) be inherited from I^F. However, this is not possible because it is not triggered for the father, while it is triggered for the daughter. Therefore, the choice as well as the implementation state is taken from the mother, i.e., activity 7 is chosen ($\alpha_2^D := \alpha_2^M = 7$ and $\nu_7^D := 1$). No further activities are triggered. In the activity list, the four ($c^\alpha = 4$) first activities (1, 3, 6, 4) are inherited from the mother λ^M and the order

of the remaining activities (2, 7, 5, 8, 9, 10) from the father λ^F.

For the son I^S the first decision is inherited from I^F, i.e., activity 4 is activated ($\alpha_1^S := 4$ and $\nu_4^S := 1$). The second decision is not triggered for the son. Therefore, no value is assigned to α_2^S. The choice to implement activity 4 triggers the implementation of activity 9 ($\nu_9^S := 1$). In the activity list, the first four activities (1, 2, 3, 7) are inherited from λ^F and the order of the remaining activities (6, 4, 8, 5, 9, 10) stems from λ^M.

$$
I^M = \left(\begin{array}{cc|cccccccccc} 5 & 7 & 1 & 3 & 6 & 4 & 2 & 8 & 5 & 7 & 9 & 10 \\ 1;\{4,5\} & 5;\{7,8\} & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{array}\right)
$$

$$
I^F = \left(\begin{array}{cc|cccccccccc} 4 & - & 1 & 2 & 3 & 7 & 4 & 5 & 6 & 8 & 9 & 10 \\ 1;\{4,5\} & 5;\{7,8\} & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array}\right)
$$

$$
I^D = \left(\begin{array}{cc|cccccccccc} 5 & 7 & 1 & 3 & 6 & 4 & 2 & 7 & 5 & 8 & 9 & 10 \\ 1;\{4,5\} & 5;\{7,8\} & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right)
$$

$$
I^S = \left(\begin{array}{cc|cccccccccc} 4 & - & 1 & 2 & 3 & 7 & 6 & 4 & 8 & 5 & 9 & 10 \\ 1;\{4,5\} & 5;\{7,8\} & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{array}\right)
$$

Note that the implementation list ν of the newly created daughter I^D and son I^S reflects its respective choice list α^D or α^S. The activity list λ^D or λ^S of the new individual may be inconsistent with the possibly new project structure and may require a repair. This problem is handled in the next subsection along with the mutation operation.

4.5. Mutation, repair and selection

The mutation operation aims at creating some random diversity in the population of solutions. The objective is to make such spaces of the solution space accessible that cannot be reached by combining elements of already known solutions via the crossover. To this end, individual elements of the solution are occasionally changed (or mutated) at random. In our context, this affects on the one hand the structure of the project (as expressed via the choice list α and the corresponding implementation list ν) and on the other hand the schedule (as determined via the activity list λ). Systematic repair operations may be necessary to ensure that the content of the different lists is consistent and feasible. Note that the crossover operation may have already led to an activity list λ that is infeasible. For this reason, we use a three-step approach to mutate and to repair. In the first step, mutation is applied to the choice list α. In the second step, the activity list λ which may now be infeasible is repaired. This leads to a consistent and feasible combination of choice list α, activity list λ and implementation list ν. In the final third step, the activity list λ is mutated in such a way that only feasible solutions result. The procedure is shown in Algorithm 3.

For the mutation of the choice list α and implementation list ν (in the first of the three steps), each decision is considered. If a (due to the crossover) now triggered choice e

/* Part 1: Mutate choice list */
for *choices* $e = 1, ..., E$ **do**
  if *choice* e *is currently triggered* ($\nu_{a(e)} = 1$) **then**
    if *choice* e *was previously not triggered* ($\alpha_e = n/a$) **then**
      Randomly select one activity $j \in \mathcal{W}_e$
      $\nu_j := 1$; $\alpha_e := j$
    else
      Determine a realization m of a $U(0;1)$-distributed random variable
      if $m \leq m^{\alpha}$ **then**
        Randomly select one activity $j \in \mathcal{W}_e$
        if $j \neq \alpha_e$ **then**
          $\nu_{\alpha_e} := 0$; $\nu_j := 1$; $\alpha_e := j$
        end
      end
    end
  else
    if *choice* e *was previously triggered* ($\alpha_e \neq n/a$) **then**
      $\nu_{\alpha_e} := 0$; $\alpha_e := n/a$
    end
  end
end
for *each choice* e **do**
  Update triggered activities i ($\nu_i := \nu_j, j \in \mathcal{W}_e, i \in \mathcal{B}_j$)
end

/* Part 2: Repair the activity list */
if *the activity list* λ *violates the precedence constraints* **then**
  Create a copy λ^* of the infeasible activity list ($\lambda^* := \lambda$)
  Delete all elements from the activity list λ
  for *each position of the activity list* λ **do**
    Select the first activity j in λ^* which is not included in λ yet and for which all active predecessors are already included in λ
    Append the selected activity j to the activity list λ
  end
end

/* Part 3: Mutate the activity list */
for *each activity* j *in the activity list* λ *which is implemented* ($\nu_j = 1$) **do**
  Determine a realization m of a $U(0;1)$-distributed random variable
  if $m \leq m^{\lambda}$ **then**
    Determine the next implemented activity i in the activity list λ
    if j *is no predecessor of* i ($j \notin \mathcal{P}_i$) **then**
      Exchange the positions of j and i in the activity list λ
    end
  end
end

Algorithm 3: Mutation and repair

was previously not triggered, one activity $j \in \mathcal{W}_e$ is selected at random and choice list α as well as implementation list ν are updated. Otherwise, with a low mutation probability m^α, an activity $j \in \mathcal{W}_e$ is selected at random to replace the previous activity α_e if it differs from j. If choice e is currently not triggered, but was previously triggered, it has to be deactivated. Finally, all dependent activities $i \in \mathcal{B}_j$ for each $j \in \mathcal{W}_e$ are updated ($\nu_i := \nu_j$).

We use the individuals I^D and I^S from Subsection 4.4 to demonstrate the mutation of the implementation list. For I^D, we assume that a mutation occurs in the second choice $e = 2$. Instead of activity 7, activity 8 is activated ($\alpha_2 := 8; \nu_7^D := 0,\ \nu_8^D := 1$). No further adjustments are necessary for I^D.

In the case of I^S, we assume that due to a mutation of the first choice $e = 1$, activity 5 is implemented instead of activity 4 ($\alpha_1 := 5; \nu_4^S := 0,\ \nu_5^S := 1$). Thereby, the second decision is triggered. Activity 7 is chosen randomly from $\mathcal{W}_2$ ($\nu_7^S := 1$). Due to the deactivation of activity 4, activity 9 is not triggered any longer and has to be deactivated ($\nu_9^S := 0$).

$$I^D = \left(\begin{array}{cc|cccccccccc} 5 & 8 & 1 & 3 & 6 & 4 & 2 & 7 & 5 & 8 & 9 & 10 \\ 1;\{4,5\} & 5;\{7,8\} & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{array}\right)$$

$$I^S = \left(\begin{array}{cc|cccccccccc} 5 & 7 & 1 & 2 & 3 & 7 & 6 & 4 & 8 & 5 & 9 & 10 \\ 1;\{4,5\} & 5;\{7,8\} & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right)$$

The second step of the algorithm is only performed if the activity list λ is infeasible because a pair (i, j) of activities exists that are both active ($\nu_i = \nu_j = 1$) and exhibit a precedence constraint $i \in \mathcal{P}_j$, while activity j precedes activity i on the activity list λ. The activity list is re-sequenced by placing activity j behind i on the activity list for all such conflicting pairs.

For the individual I^D of our example, no repair of the activity list λ is necessary: All precedence constraints are met. For the individual I^S, however, the precedence relation between activity 5 and activity 7 is violated. This requires the postponement of activity 7 until activity 5 has been placed on the activity list. The sequence of the remaining activities does not have to be changed. This leads to the following (preliminary) result:

$$I^S = \left(\begin{array}{cc|cccccccccc} 5 & 7 & 1 & 2 & 3 & 6 & 4 & 8 & 5 & 7 & 9 & 10 \\ 1;\{4,5\} & 5;\{7,8\} & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right)$$

The mutation of the activity list in the third step is implemented analogously to Hartmann (2001). With a low probability m^λ, an activity j is selected to be exchanged with the next implemented activity i in the activity list. This exchange is only feasible and therefore implemented if activity j is not a predecessor of activity i ($j \notin \mathcal{P}_i$).

In our example, a mutation of activity $\lambda_7 = 5$ is tried for I^D. The next active activity is activity $\lambda_8 = 8$. Due to the precedence relation between these activities, the mutation is not feasible. For I^S, an exchange of activity $\lambda_4 = 6$ with the next implemented activity ($\lambda_7 = 5$) is intended. This exchange is feasible and therefore the positions of these activities are exchanged ($\lambda_7 := 6,\ \lambda_4 := 5$), with the following result:

$$I^S = \left(\begin{array}{cc|cccccccccc} 5 & 7 & 1 & 2 & 3 & 5 & 4 & 8 & 6 & 7 & 9 & 10 \\ 1;\{4,5\} & 5;\{7,8\} & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right)$$

After the mutation, the generation of new individuals is finished. For each individual, the fitness can be computed and the best N^I individuals out of the parent and the children generation are selected as the parent generation for the next iteration of the genetic algorithm. The procedures crossover, mutation and selection of the genetic algorithm are applied again until N^G generations are created and evaluated. Afterwards, the fittest individual and its corresponding best schedule of the last generation is chosen as a solution of the underlying problem.

For the presented example, the fitness F of the individuals is $F(I^S) = 14$, $F(I^M) = 14$, $F(I^F) = 13$ and $F(I^D) = 12$, as can be seen if schedules for the individuals I^F, I^D, and I^S are generated similar to I^M in Figure 6. The individuals I^D and I^F would be selected as parent generation for the next iteration of the genetic algorithm if a population size of 2 were used (which would be too small in a realistic attempt to use a genetic algorithm).

5. Numerical analysis

5.1. Test design

In order to evaluate the genetic algorithm with respect to speed and solution quality, it was necessary to generate adequate test instances. We enhanced the instance generator ProGen (cf. Kolisch et al. (1995)) to create problem instances with a flexible project structure. Four different problem classes with 30, 60, 90, and 120 non-dummy activities were generated, cf. Appendix Appendix B. Each class consisted of 1536 test instances out of which some were infeasible due to limited non-renewable resources.

The genetic algorithm was implemented in Delphi XE and executed on a 2.66 GHz Intel Core2 Quad machine with 4 GB of RAM using a single thread. Each generation of the genetic algorithm consisted of $N^I = 80$ individuals (and hence schedules). In order to show the improvement of the solutions over the generations, we present results for 13, 63, and 125 generations N^G, i.e., after generating 1,040, 5,040, and 10,000 schedules. The mutation parameters were set to $m^\alpha = 3\%$ and $m^\lambda = 10\%$, respectively.

We used CPLEX 12 on a 2.00 GHz Intel Xeon machine with 110 GB of RAM and four threads within a central compute cluster at Leibniz Universität Hannover, cf. `http://www.rrzn.uni-hannover.de/clustersystem.html` to determine reference values. The

reference values represent optimal solutions (for the first problem class with 30 non-dummy activities) or upper bounds (for the problem classes with 60 or more non-dummy activities). It was not possible to solve the test instances with 60 activities or more to optimality within reasonable time. Therefore, we limited the CPLEX CPU time per instance to one hour. Those test instances required a maximum of 25 GB of RAM. The new genetic algorithm found a feasible solution for each instance for which CPLEX either found at least a feasible solution or was unable to decide on (in)feasibility of the instance within a given time limit. Out of the 1536 test instance of each class, CPLEX found optimal or at least feasible solutions for 1166, 1072, 1046, and 1040 instances, respectively. Those instances were used evaluate the genetic algorithm.

In order to compute additional reference values, we systematically enumerated over *all* the possible project structures per instance for all the instances with 30, 60, and 90 non-dummy activities. To solve all the resulting RCPSPs (with fixed structure) in this *structure enumeration approach*, we again used a genetic algorithm, but with a lower number of individuals or schedules for each possible project structure ($N^I = 50$) and only 10 generations ($N^G = 10$), i.e., 500 schedules per project structure. Due to the large number of possible project structures and the substantial computational effort, we were unable to determine reference values with this method for problem class 4 with the largest number of non-dummy activities.

5.2. Results

We present the numerical results in Tables 4 - 7. They show consistently that the presented genetic algorithm is very fast. Even for the problem class with the largest number of 120 non-dummy activities, it requires only about one second to generate 10,000 different schedules, see Table 7. Furthermore, the solution quality appears to be very high. For the first problem class with 30 non-dummy activities, the average deviation of the makespan from the optimal values is only 0.24%, see Table 4. For the remaining three classes we report the deviations from the best known solutions. The genetic algorithm always outperforms both CPLEX and the structure enumeration approach both with respect to accuracy and speed. It usually finds better solutions than the other approaches, in particular for larger instances and a larger number of generations. We conclude that the presented algorithm is a powerful instrument to schedule flexible projects with resource constraints.

6. Conclusions and outlook

In this paper we analyzed the problem to schedule projects with a flexible structure. We introduced the concepts of both mandatory and optional choices, of optional activities, and of dependent activities. These concepts substantially increase the power and flexibility to model real-world resource-constrained project scheduling projects. We furthermore

	Genetic Algorithm			CPLEX	Structure enumeration approach
	1,040 schedules	5,040 schedules	10,000 schedules	optimal solutions	
deviation	0.42 %	0.26 %	0.24 %	0 %	0.31 %
optimal	88.4 %	92.2 %	92.6 %	100 %	90.5 %
time	0.02 s	0.07 s	0.13 s	1214.01 s	0.37 s

Table 4: Results for the test instances with 30 activities

	Genetic Algorithm			CPLEX	Structure enumeration approach
	1,040 schedules	5,040 schedules	10,000 schedules	upper bounds	
deviation	1.19 %	0.50 %	0.44 %	1.17 %	0.73 %
best known	69.9 %	84.0 %	86.7 %	89.1 %	77.2 %
time	0.04 s	0.18 s	0.34 s	609.70 s	14.51 s

Table 5: Results for the test instances with 60 activities

	Genetic Algorithm			CPLEX	Structure enumeration approach
	1,040 schedules	5,040 schedules	10,000 schedules	upper bounds	
deviation	1.39 %	0.36 %	0.26 %	3.83 %	0.71 %
best known	69.4 %	86.2 %	92.3 %	80.1 %	76.2 %
time	0.09 s	0.34 s	0.64 s	914.25 s	378.75 s

Table 6: Results for the test instances with 90 activities

	Genetic Algorithm			CPLEX	Structure enumeration approach
	1,040 schedules	5,040 schedules	10,000 schedules	upper bounds	
deviation	1.31 %	0.24 %	0.11 %	10.40 %	n/a
best known	69.0 %	86.3 %	96.5 %	73.8 %	n/a
time	0.14 s	0.53 s	1.01 s	1,099.80 s	n/a

Table 7: Results for the test instances with 120 activities

presented a formal decision model and developed a genetic algorithm to solve the model for a given problem instance. The model itself is a generalization of the well-established RCPSP. It also covers the multi-mode RCPSP. A comprehensive numerical study showed that the presented algorithm is both fast and accurate.

In application fields such as the regeneration of complex capital goods such as jet engines, alternative project structures often go along with varying quality characteristics or functional features. The different project structures typically affect both cost and revenues of the project, which can have a major effect on the eventually chosen project structure and schedule. The presently discussed version of the problem, however, attempts to minimize the makespan of the project and does not consider those aspects. Future research should address these topics. If both cost and revenues additionally depend on time, the problem structure changes substantially and it may be economically efficient to deliberately delay an activity, for example, to avoid overtime costs. This will require substantial modifications of the algorithm, but also open important new fields of application.

Acknowledgments

The authors thank the German Research Foundation (DFG) for the financial support of this research project in the CRC 871 'Regeneration of complex durable goods'.

References

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. New York: John Wiley & Sons.

Belhe, U., & Kusiak, A. (1995). Resource Constrained Scheduling of Hierarchically Structured Design Activity Networks. *IEEE Transactions on Engineering Management*, *42*, 150–158.

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

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

Čapek, R., Šůcha, P., & Hanzálek, Z. (2012). Production scheduling with alternative process plans. *European Journal of Operational Research*, *217*, 300–311.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Massachusetts: Addison-Wesley Publishing Company.

Hartmann, S. (2001). Project Scheduling with Multiple Modes: A Genetic Algorithm. *Annals of Operations Research*, *102*, 111–135.

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

Kellenbrink, C. (2013). First results on resource-constrained project scheduling with model-endogenous decision on the project structure. In *Operations Research Proceedings 2012*. Springer Verlag. In press.

Kelley, J. E. (1963). The Critical-path Method: Resources Planning and Scheduling. In J. F. Muth, & G. L. Thompson (Eds.), *Industrial Scheduling* (pp. 347–365). Englewood Cliffs: Prentice-Hall.

Klein, R. (2000). *Scheduling of Resource-Constrained Projects*. Boston: Kluwer Acadamic Publishers.

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

Kolisch, R., & Hartmann, S. (1999). Heuristic algorithms for the resource-constrained project scheduling problem: classification and computational analysis. In J. Węglarz (Ed.), *Project Scheduling: Recent Models, Algorithms and Applications* (pp. 147–178). Boston: Kluwer Academic Publishers.

Kolisch, R., & Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, *29*, 249–272.

Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. *Management Science*, *41*, 1693–1703.

Kuster, J., Jannach, D., & Friedrich, G. (2009). Extending the RCPSP for modeling and solving disruption management problems. *Applied Intelligence*, *31*, 234–253.

Kuster, J., Jannach, D., & Friedrich, G. (2010). Applying Local Rescheduling in response to schedule disruptions. *Annals of Operations Research*, *180*, 265–282.

Neumann, K. (1990). *Stochastic Project Networks*. Berlin: Springer-Verlag.

Pascoe, T. L. (1966). Allocation of resources C.P.M. *Revue française de recherche operationelle*, *38*, 31–38.

Pritsker, A. A. B., Watters, L. J., & Wolfe, P. M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, *16*, 93–108.

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

Tiwari, V., Patterson, J. H., & Mabert, V. A. (2009). Scheduling projects with heterogeneous resources to meet time and quality objectives. *European Journal of Operational Research*, *193*, 780–790.

Appendix A. Probabilistic dispatching procedure

Following Hartmann (2001), the choice of one out of the set $\mathcal{J}^*$ of eligible activities to be placed on the activity list λ takes the latest start times of the activities into account. The idea is that the probability g_i to select an activity $i \in \mathcal{J}^*$ for the next position on the activity list should be high if it has to be started early, i.e., has a small latest start time LST_i. To this end, we calculate the probabilities g_i as follows:

$$g_i = \frac{\frac{1}{LST_i}}{\sum\limits_{j \in \mathcal{J}^*} \frac{1}{LST_j}} \qquad i \in \mathcal{J}^* \tag{A.1}$$

However, it is not clear how to assign a "latest start time" LST_i to an activity that is not implemented (for the currently considered project structure). In our approach, we first determine a potentially weak upper bound UB on the makespan as follows:

$$UB = \sum_{j \in \mathcal{J}} d_j \tag{A.2}$$

This leads to a "raw" latest start time $\overline{LST}_J = UB - d_J$ of the final activity J. In a backward recursion we then determine "raw" latest start times

$$\overline{LST}_i = \min\left\{\overline{LST}_j | i \in \mathcal{P}_j \wedge \nu_j = 1\right\} - d_i \qquad i \in \mathcal{J} \tag{A.3}$$

for each activity i, considering only precedence relations to currently implemented activities j (with $\nu_j = 1$). We finally determine fictive latest start times $LST_i > 0$ from these raw values as follows:

$$LST_i = \overline{LST}_i - \min\left\{\overline{LST}_j | j \in \mathcal{J}\right\} + 1 \tag{A.4}$$

These values are then used to determine dispatching probabilities g_i in Equation (A.1).

Problem class	**Non-dummy activities**	N^E	N^C
1	30	$\{2;4\}$	$\{1;2\}$
2	60	$\{3;6\}$	$\{2;4\}$
3	90	$\{4;8\}$	$\{3;6\}$
4	120	$\{5;10\}$	$\{4;8\}$

Table B.8: Parameters for the 4 different classes of test instances

Parameter	#	**Values**			
$\lvert\mathcal{N}\rvert$	1	2			
$\lvert\mathcal{R}\rvert$	1	2			
N^E	2	*see Table B.8*			
N_e^W	2	2	4		
N^C	2	*see Table B.8*			
N_j^B	2	1	3		
RF_R	2	0.50	1.00		
RS_R	4	0.25	0.50	0.75	1.00
RF_N	2	0.50	1.00		
RS_N	2	0.75	1.00		
NC	3	1.5	1.8	2.1	

Table B.9: Parameters for each class of test instances

Appendix B. Test design for the numerical analysis

Four different classes of test instance with different numbers of non-dummy activities were generated, cf. Table B.8. In each class, the parameters related to the flexibility of the project structure were varied in order to create instances of different complexity. In problem class 1 with 30 activities, the number of choices N^E equals either 2 or 4. The number of activities N^C which may cause further activities is either 1 or 2. Note that problem classes 2 - 4 not only contain more non-dummy activities, but also more choices and dependent activities, and hence a higher flexibility of the project structure.

In addition to these values, further parameters were systematically varied within each class, cf. Table B.9, to create a full-factorial test bed with $2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 4 \cdot 2 \cdot 2 \cdot 3 = 1536$ instances for each class. However, not all test instances were feasible due to the limited non-renewable resources.

In each test instance two non-renewable and two renewable resources are considered

($|\mathcal{N}| = |\mathcal{R}| = 2$). The number of optional activities per choice e is N_e^W while N_j^B is the number of dependent activities that are caused by such an optional activity.

The other parameters are defined according to Kolisch et al. (1995). The resource factor RF_R (cf. Pascoe (1966), p. 35 and Kolisch et al. (1995), p. 1697) is computed as follows:

$$RF_R = \frac{1}{J}\frac{1}{|\mathcal{R}|}\sum_{j=1}^{J}\sum_{r\in\mathcal{R}}\begin{cases} 1, & \text{if } k_{jr} > 0 \\ 0, & \text{else} \end{cases} \tag{B.1}$$

A value $RF_R = 1$ means that each activity uses all types of renewable resources and for a value of $RF_R = 0$ no renewable resources are used, i.e., the problem is unconstrained with respect to renewable resources, cf. Kolisch et al. (1995), p. 1697. The resource factor for the non-renewable resources RF_N is defined analogously.

The resource strength RS_R for the renewable resources RS_R is used in order to determine the resource availability, cf. Kolisch et al. (1995), pp. 1698-1699. It is computed as a convex combination of the minimal and the maximal required capacity per period with RS_R as a scaling parameter:

$$K_r = K_r^{min} + round\left(RS_R \cdot \left(K_r^{max} - K_r^{min}\right)\right) \qquad r \in \mathcal{R} \tag{B.2}$$

The maximal capacity demanded by a single activity K_r^{min} is determined as follows:

$$K_r^{min} = \max\left\{k_{jr} \mid j = 2, ..., J-1\right\} \qquad r \in \mathcal{R} \tag{B.3}$$

Denote with $\mathcal{JC}_t = \{j|ST_j + 1 \leq t \leq CT_j\}$ the set of activities being performed at time t in a particular schedule and assume that an earliest start schedule is given for a related problem in which there are i) no capacity constraints and in which ii) all activities are implemented and all precedence constraints are respected which leads to these times ST_j and CT_j. Then the maximal period capacity K_r^{max} can be computed as follows:

$$K_r^{max} = \max\left\{ \sum_{j\in\mathcal{JC}_t} k_{jr} \;\middle|\; t = 1, ..., T \right\} \qquad r \in \mathcal{R} \tag{B.4}$$

We used a similar approach to define the resource strength for the non-renewable resources RS_N based on the assumption that all activities were implemented to determine the capacity K_r of the non-renewable resources:

$$K_r = RS_N \cdot \sum_{j=1}^{J} k_{jr} \qquad r \in \mathcal{N} \tag{B.5}$$

The value $RS_N = 1$ means that the problem is not constrained concerning the non-renewable resources.

The network complexity NC is defined as the average number of arcs per node, cf. Pascoe (1966), p. 34 and Kolisch et al. (1995), p. 1696:

$$NC = \frac{\text{Number of Arcs}}{\text{Number of activities (incl. dummy activities)}} \tag{B.6}$$

This ratio is computed based on all the potentially implemented activities and precedence relations. Due to the choice on the structure, some activities as well as related precedence relations will eventually be dropped while additional precedence constraints will be added to make sure that each implemented activity is either directly or indirectly connected to the dummy start and end activity.