Handling Alternative Activities in Resource-Constrained Project Scheduling Problems

Planning and Scheduling,
Genetic Algorithms,
Applications

Abstract

In the context of operative disruption management, decision support systems have to evaluate the typically manifold options of responding to disturbances: The *temporal shift* of activities and the *allocation of alternative resources* can be assessed by the application of generic scheduling frameworks such as the Resource-Constrained Project Scheduling Problem (RCPSP). However, *switches from one process variant to another one* are usually not supported by the corresponding models, even though they represent a common way of repairing broken schedules in many practical domains. In this paper, we thus show how the RCPSP can be extended by the concept of alternative activities, making it possible to model and search within alternative process execution paths. Beside a formal description of the conceptual extension, we show how such generalized rescheduling problems can be solved by a novel genetic algorithm and summarize the promising results of a detailed evaluation.

1 Introduction

Disruption management (DM, see [Yu and Qi, 2004; Clausen *et al.*, 2001]) is the process of responding to an unforeseen disturbance occurring during the execution of planned and scheduled operations. It is aimed at the selection of appropriate repair actions which help to get back on track and to minimize the negative impact typically associated with a disruption. Two forms of interventions can be distinguished:

- *Rescheduling* corresponds to a shift of scheduled starting times or a change in the allocation of resource entities to activities. It is particularly relevant in domains where the set of operations is fixed and no process variability is given: Production processes form a typical example.
- A *process variation* corresponds to the switch from a previously chosen process variant to another one. This form of modification is applied in many practical domains to insert or remove activities, to change execution orders, etc.: For example project scheduling and supply-chain management are typically characterized by the existence of various predefined process variants.

The Resource-Constrained Project Scheduling Problem (RCPSP, see [Błazewicz *et al.*, 1983; Brucker *et al.*, 1999]) provides a generic and well-established framework for the formal description of scheduling problems: Generalizing the production-specific job shop, flow shop and open shop problems, it imposes no restrictions on either the number of entities per resource type, the way activities are connected or the characteristics of resource requirements. Moreover, the existence of various highly efficient algorithms for its solution (see [Kolisch and Hartmann, 2005]) proves that the RCPSP has been and still is an area of intensive research.

The rescheduling part of DM problems can be solved within the framework provided by the RCPSP. As regards potential process variations, however, there exist no possibilities to describe alternative execution paths within the RCPSP: It is thus impossible to overcome a previously made decision on a certain process variant during optimization. To the best of our knowledge, the only approach to more flexibility has been made in the Multi-Mode RCPSP (MRCPSP, see [Hartmann, 2001] for example): Through the definition of various *executions modes* per activity, it is possible to consider changes in their durations and associated resource requirements. But it is obvious that the concept of mode alternations is not sufficient for the description of the entire range of potential variations.

Research on DM in the context of the RCPSP has mainly focused on responding to disruptions by way of rescheduling: Artigues et al. [2003] propose an approach to insert an unexpected activity into a given schedule; Elkhyari et al. [2004] use explanations to cope with over-constrained networks in dynamic scheduling problems; Zhu et al. [2005] present a hybrid mixed integer programming/constraint propagation approach to handling various classes of disruptions. In contrast to these works, which all focus on mere *rescheduling*, we herein extend the scope of available options by making *switches between different process variants* possible.

For this purpose, we propose to extend the RCPSP by the concept of alternative activities: In Section 2, the formal description of the conceptual extension and a modeling example from the domain of air traffic management are provided. In Section 3, we propose a novel genetic algorithm (GA) for the solution of the problem: The specific crossover and mutation operators are discussed as well as the promising results of a detailed evaluation. Finally, Section 4 summarizes the contributions of this paper.

2 A Conceptual Extension of the RCPSP

2.1 Alternative Activities

We base our approach to defining alternative execution paths on the concept of *alternative activities* and *activity dependencies*. The expressive power of these constructs is illustrated by several modeling examples of typical process variations:

- A *mode alternation* corresponds to a change of the duration and the resource requirements associated with an activity. Basically, each execution mode can be represented by a separate alternative activity. If, for example, in a sequence a, b and c three modes of b shall be distinguished, the alternative activities b_1, b_2 and b_3 can be considered instead of b.
- For the dynamic *insertion/removal* of activities, the possibility to describe activity dependencies is required: An activity i is considered *dependent* on another activity j, if j has to be executed whenever i is scheduled. If, for example, it shall be possible to dynamically insert an activity e into a sequence a, b and c, it is sufficient to consider two alternatives of an arbitrary scheduled operation (activity b, for example): One variant (b_1) is dependent on the execution of e whereas the other one (b_2) is not.
- A potential *order change* can be described by alternative versions of all involved process steps: One variant represents the original, the other one the optional position of the activity. If, for example, it shall be possible to swap b and d in a sequence a, b, c and d, b_1 and d_1 correspond to the original, b_2 and d_2 to the alternative positions.
- The *serialization/parallelization* of operations is based on two alternative activities with different precedence relations associated: One variant is a predecessor of another activity whereas the other one is not. If, for example, it shall be possible to parallelize b and c in a sequence a, b, c and d, it suffices to consider two versions of b: b_1 is a predecessor of c and represents the option of serial execution, whereas b_2 is not necessarily scheduled before c and represents the option of parallel execution.

By applying the concept of alternative activities, both parts of the DM problem – the selection *and* scheduling of operations – are combined in one single conceptual model. This implies that (1) no detached process definitions need to be synchronized and that (2) particularly efficient algorithms can be applied for problem solving [Barták, 1999].

2.2 The Extended Resource-Constrained Project Scheduling Problem

As regards the consideration of alternative activities in scheduling problems, little research has been done so far. One of the most important approaches can be found in the area of constraint-directed scheduling: Beck and Fox [2000] base their work on the introduction of XOR-nodes as well as PEX (Probability of Existence) variables into the constraint graph, and propose specific, PEX-based propagators and heuristics for solving such extended problems.

In contrast to their work, which aims at extending the scope of constraint-directed scheduling approaches, our goal is to incorporate the concept of alternative activities into the more general framework of the RCPSP. Consequently, we also propose a compact way of modeling process execution variants which is independent from the underlying search procedure: In the Extended Resource-Constrained Project Scheduling Problem (*x-RCPSP*), the concept of alternative activities is implemented through the distinction between *active* and *inactive* activities: Only the former ones are actually considered in the resulting schedule. By activating and deactivating activities it is possible to change the *activation state* of the model. Note that each potential combination of active elements forms an individual instance of the classical RCPSP: It is thus possible to apply well-established methods for activity sequence optimization.

Formally, the *x-RCPSP* can be described as follows. A process is composed of a set of potential activities $\mathcal{A}^+ = \{0, 1, ..., a, a+1\}$ where the first and the last element represent abstract start and end activities: They have a duration of 0 and do not require any resources. Each remaining $i \in \mathcal{A}^+$ has a non-negative duration d_i assigned. Active activities are grouped in a subset $\mathcal{A} \subseteq \mathcal{A}^+$, inactive activities are contained in the set difference $\mathcal{A}^+ \backslash \mathcal{A}$. For the execution of activities, a set of renewable resource types $\mathcal{R} = \{1, ..., r\}$ is available: For each type $k \in \mathcal{R}$, there exists a constant amount of c_k units. Various forms of activity dependency can be described by use of the following constructs:

- *Precedence Constraints.* The order of activities is described by use of precedence constraints: The existence of $p_{i,j}$ states that activity $i \in \mathcal{A}^+$ has to be finished at or before the start of $j \in \mathcal{A}^+$. In accordance with the distinction between active and inactive activities, $\mathcal{P}^+$ contains all potentially relevant constraints whereas $\mathcal{P} = \{p_{i,j} \in \mathcal{P}^+ | i, j \in \mathcal{A}\}$ groups only relations in which predecessor and successor are currently active.
- *Resource Requirements.* Resource requirements describe the relation between activities and resources. Activity $i \in \mathcal{A}^+$ requires a constant amount of $q_{i,k}$ units of resource type $k \in \mathcal{R}$ throughout its execution. Again, two sets are distinguished: $\mathcal{Q}^+$ contains all potentially relevant dependencies whereas $\mathcal{Q} = \{q_{i,k} \in \mathcal{Q}^+ | i \in \mathcal{A}\}$ groups only the requirements of active activities.

Potential activation state modifications and associated dependencies are described by use of the following constructs:

- *Activity Substitutions.* If $x_{i,j}$ is contained within the set of potential activity substitutions $\mathcal{X}^+$, it represents a legal form of process variation to activate $j \in \mathcal{A}^+ \backslash \mathcal{A}$ for the deactivation of $i \in \mathcal{A}$: j replaces i within $\mathcal{A}$.
- *Activity Dependencies/Constraints.* Since the activation or deactivation of an activity might have an impact on the state of other activities, $\mathcal{M}^+$ describes dependencies between the elements of $\mathcal{A}^+$: $m^{\oplus}_{i,j}$ ($m^{\ominus}_{i,j}$) $\in \mathcal{M}^+$ indicates that activity $j \in \mathcal{A}^+$ has to be (de)activated upon the (de)activation of $i \in \mathcal{A}^+$; $m^{\oplus}_{i,j} \in \mathcal{M}^+$ indicates that j has to be deactivated upon the activation of i, and $m^{\ominus}_{i,j} \in \mathcal{M}^+$ indicates that j has to be activated upon the deactivation of i.

The presented *x-RCPSP* is a generalization of the MRCPSP, which itself generalizes the classical RCPSP [Hartmann, 2001]: Any MRCPSP can be formulated as an *x-RCPSP*. As regards the formulation of an *x-RCPSP* as an MRCPSP, the limitations of MRCPSPs must be observed. Such a formulation which preserves the activities, precedence constraints, and resource constraints of the *x-RCPSP* is possible if the following properties hold:

$$x_{i,j} \in \mathcal{X}^+ \Rightarrow x_{j,i} \in \mathcal{X}^+ \quad (1)$$

$$x_{i,j}, x_{j,k} \in \mathcal{X}^+ \Rightarrow x_{i,k} \in \mathcal{X}^+ \quad (2)$$

$$\begin{aligned} x_{i,j} \in \mathcal{X}^+, p_{i,k} \in \mathcal{P}^+ \Rightarrow p_{j,k} \in \mathcal{P}^+ \\ x_{i,j} \in \mathcal{X}^+, p_{k,i} \in \mathcal{P}^+ \Rightarrow p_{k,j} \in \mathcal{P}^+ \end{aligned} \quad (3)$$

$$\mathcal{M}^+ = \varnothing \quad (4)$$

(1) It is not possible to describe one-directional mode substitutions. (2) Any activity execution mode has always to be a direct substitute of all other ones. (3) Activity replacements are only possible at exactly the same position: No changes of the activity execution order can thus be defined. (4) No activity dependencies can be described for different execution modes: It is therefore not possible to dynamically insert or remove activities.

The result of solving the *x-RCPSP* is a combination of model activation state and sequence of all active activities: A *schedule* S is represented as a vector of starting times $(\beta_1, \beta_2, ..., \beta_n)$ for the set of active activities $\mathcal{A}$ with $|\mathcal{A}| = n$. S is considered *valid* if the following criteria are fulfilled:

- *Activation State Validity.* The activation state $\mathcal{A}$ associated with a schedule S is valid, if and only if it can be derived from an original valid activation state through the application of the substitutions defined in $\mathcal{X}^+$, satisfying all constraints defined in $\mathcal{M}^+$.
- *Starting Times Validity.* Let $\mathcal{A}(t)$ be the set of activities carried out at a time t. The staring times $\beta_1, ..., \beta_n$ are valid if (1) $\beta_i \geqslant 0$ for any $i \in \mathcal{A}$, (2) $\beta_i + d_i \leqslant \beta_j$ for any $p_{i,j} \in \mathcal{P}$ and (3) $\sum_{i \in \mathcal{A}(t)} q_{i,k} \leqslant c_k$ for any $k \in \mathcal{R}$ at any t. Note that these criteria correspond to the ones defining schedule validity in the context of the RCPSP.

2.3 Modeling an Exemplary Process

In this section it is illustrated how the framework of the *x-RCPSP* can be applied for the formal description of realistic processes. For this purpose, the *aircraft turnaround* – as the process aircrafts typically go through at an airport between touchdown and takeoff – is considered in a simplified version (corresponding to the combination of core processes as defined by [Carr, 2004]): After the plane reaches its final position, incoming passengers leave the aircraft (*deboarding*). It is then prepared for the next flight during *fueling*, *cleaning* and *catering*, which may be executed simultaneously. Outgoing passengers enter the aircraft (*boarding*) before the the plane finally leaves its position heading for the runway.

Apart from inherent options of rescheduling, three forms of variations are assumed to be available for the adaptation of the process in response to disruptions: First, deboarding can be accelerated through the assignment of additional buses. Second, cleaning can be shortened if in exchange the cabin is inspected by the cabin crew prior to boarding. And third, the process can be accelerated by parallelizing fueling and boarding if the fire brigade is present for supervision.

Table 1: *x-RCPSP* Description of the Aircraft Turnaround

Set	Content
$\mathcal{R}$	$Bus, Firebrigade$
$\mathcal{A}^+$	$Start, Deb, Deb^B, Fue, Fue^P, Cat, Cle, Cle^R, Ins, Boa, End$
$\mathcal{P}^+$	$Start \rightarrow Deb, Start \rightarrow Deb^B, Deb \rightarrow Fue, Deb \rightarrow Fue^P, Deb \rightarrow Cat, Deb \rightarrow Cle, Deb \rightarrow Cle^R, Deb^B \rightarrow Fue, Deb^B \rightarrow Fue^P, Deb^B \rightarrow Cat, Deb^B \rightarrow Cle, Deb^B \rightarrow Cle^R, Fue \rightarrow Boa, Fue^P \rightarrow End, Cat \rightarrow Boa, Cle \rightarrow Boa, Cle^R \rightarrow Ins, Ins \rightarrow Boa, Boa \rightarrow End$
$\mathcal{Q}^+$	$Deb \rhd 1 \times Bus, Deb^B \rhd 2 \times Bus, Fue^P \rhd 1 \times Firebrigade$
$\mathcal{X}^+$	$Deb \leftrightsquigarrow Deb^B, Fue \leftrightsquigarrow Fue^P, Cle \leftrightsquigarrow Cle^R$
$\mathcal{M}^+$	$Cle^R \oplus Ins, Cle^R \bigcirc Ins$

Table 1 summarizes the central elements of the corresponding *x-RCPSP* (particularly activity durations and resource capacities are omitted) based on a simplified form of notation: (1) $i \rightarrow j$ stands for $p_{i,j}$, (2) $i \rhd n \times k$ for $q_{i,k} = n$, (3) $i \rightsquigarrow j$ for $x_{i,j}$, (4) $i \leftrightsquigarrow j$ for $x_{i,j} \cup x_{j,i}$, (5) $i \oplus j$ for $m^{\oplus}_{i,j}$, etc. Process steps are represented by the first three letters of the associated activity names. Potential process variations are expressed by use of alternative activities. The first of the available options corresponds to a mode alternation (as also possible in an MRCPSP): Deb^B is characterized by reduced time and additional resource requirements. The second option corresponds to a combination of mode alternation and activity insertion: Cle^R takes less time but is connected to the optional activity Ins (for cabin inspection). The third option corresponds to the parallelization of two process steps: Fue^P is not necessarily executed prior to boarding but has additional resource requirements associated.

3 Solving the x-RCPSP

The application of the *x-RCPSP* to DM is based on a comprehensive model describing the baseline schedule and all valid process variants. The following steps are performed upon the occurrence of a disruption: (1) The schedule is updated accordingly. (2) Based on a schedule evaluation function $f(S)$ and the corresponding objective (minimization or maximization), optimization is performed. Unlike in scheduling problems, where mainly the minimization of the total process execution time (the so-called *makespan*) is of interest, common goals in DM are the minimization of costs for earliness, tardiness, interventions and the deviation from the original schedule: Typically, several of these aspects are combined. (3) Finally, the difference between the original and the optimized schedule is interpreted as the set of interventions to apply.

Due to the central relevance of the second step, we will focus on the optimization of the *x-RCPSP* in this section. As regards the choice of an approach to the identification of the optimal combination of activation state and activity starting

times, the mentioned practical relevance of proximity to the original schedule suggests the use of incremental local search algorithms. Since, moreover, performance usually represents a crucial factor in the operative process of DM and since particularly genetic algorithms perform well for the RCPSP [Kolisch and Hartmann, 2005] we herein present and evaluate an evolutionary algorithm for the solution of the *x-RCPSP*.

3.1 An Evolutionary Algorithm

Representation

Due to the complexity associated with the direct modification of time values, it is a common approach to use some sort of abstract solution representation during optimization [Kolisch and Hartmann, 2005; Hindi *et al.*, 2002]. We decided on the use of *activity lists*: λ is a precedence feasible list of all elements in $\mathcal{A}$, describing the order in which active activities shall be added to the schedule. In the associated serial schedule generation scheme, each operation is scheduled at the earliest possible time: This way, each valid λ can be converted into a valid schedule unambiguously.

Initialization

The original schedule can be converted into a corresponding activity list easily: λ_0 is obtained by simply sorting all active activities by their scheduled starting times. It represents the legal option of performing no intervention at all and is therefore considered the first element of the first generation. All other solutions of the initial population are generated by the application of the mutation operator (see below) on λ_0.

Crossover

If the elements contained in two parent activity lists λ_a and λ_b are equal, one of the well-elaborated RCPSP-specific crossover operators can be applied (see [Hartmann, 1998; Hindi *et al.*, 2002] for examples or [Kolisch and Hartmann, 2005] for a comprehensive overview). If, however, the contents of λ_a and λ_b differ, list combination is more difficult: Algorithm 1 summarizes a crossover operator, which is based on the idea that λ_a steers the selection of elements contained in the child whereas λ_b determines the respective order, and which guarantees that only valid activity lists are generated.

Algorithm 1 Crossover (λ_a, λ_b)

```
 1: if A_a = A_b then
 2:     apply RCPSP-specific crossover operator
 3: else
 4:     T ← (X_a \ X_b) ∪ {x_{i,j} ∈ X^+ | x_{j,i} ∈ (X_b \ X_a)}
 5:     if |T| < |(X_a △ X_b)| then return incompatible
 6:     λ ← λ_b
 7:     do
 8:         changed ← false
 9:         if i ∈ λ, x_{i,j} ∈ T then
10:             replace i with j in λ
11:             changed ← true
12:         end if
13:     while changed
14: end if
15: return λ
```

Algorithm 2 Mutate (λ)

```
1: if a randomly generated value ∈ [0, 1] ⩽ θ then
2:     rearrange λ by applying RCPSP-specific mutation
3: else
4:     select an arbitrary x_{i,j} ∈ X^+ | i ∈ λ
5:     replace i with j in λ
6: end if
7: return λ
```

Denoting the set of activities in λ_i as $\mathcal{A}_i \subseteq \mathcal{A}^+$, it is first checked whether an RCPSP-specific operator can be applied (line 1). If this is not possible due to $\mathcal{A}_a \neq \mathcal{A}_b$, a *transition set* $\mathcal{T} \subseteq \mathcal{X}^+$ is initialized for coping with different list contents: $\mathcal{T}$ describes which substitutions have to be applied for the conversion of λ_b to λ_a. Let $\mathcal{X}_i \subseteq \mathcal{X}^+$ be the set of substitutions that has led from an original set $\mathcal{A}_0$ to $\mathcal{A}_i$: $\mathcal{T}$ can then be defined as the combination of all substitutions exclusive to λ_a and the inversion of all substitutions exclusive to λ_b (line 4). Since, however, substitutions are not necessarily directly reversible, it might be the case that λ_b can not be (directly) transformed into λ_a: If the size of $\mathcal{T}$ is unequal to the size of the symmetric difference $\mathcal{X}_a \triangle \mathcal{X}_b$, the activity lists are considered *incompatible* for crossover and a different selection of parent lists has to be made (line 5). Otherwise, the child activity list λ is initialized as a clone of λ_b before a repetitive replacement procedure is started (line 7 to 13): As long as λ contains replaceable elements in terms of $\mathcal{T}$, the respective substitutions are applied: Note that dependencies in $\mathcal{M}^+$ have to be observed and that precedence feasibility has to be secured by shifting successors to the right-hand side of their predecessors. If the application of the substitutions in $\mathcal{T}$ results in an inconsistency (i.e. activities have to be activated and deactivated at the same time due to contradicting dependencies in $\mathcal{M}^+$), the replacement operator (called in line 10) fails: λ_a and λ_b are considered *incompatible* for crossover and different parent activity lists have to be selected.

If, for example, $\lambda_a = (Deb^B, Fue, Cle, Cat, Boa)$ and $\lambda_b = (Deb^B, Cat, Cle^R, Ins, Boa, Fue^P)$ for the process modeled above, the transition set $\mathcal{T} = \{Cle^R \rightsquigarrow Cle, Fue^P \rightsquigarrow Fue\}$ can be deduced from $\mathcal{X}_a = \{Deb \rightsquigarrow Deb^B\}$ and $\mathcal{X}_b = \{Deb \rightsquigarrow Deb^B, Cle \rightsquigarrow Cle^R, Fue \rightsquigarrow Fue^P\}$. Crossover thus results in $\lambda = (Deb^B, Cat, Cle, Fue, Boa)$.

Mutation

An *x-RCPSP*-specific version of the mutation operator is described in Algorithm 2: Apart from changes in the order of activities, also potential process variations are considered.

With a certain probability θ, the activation state is left unmodified and any RCPSP-specific mutation operator can be applied for the random modification of λ (see the references mentioned above for crossover). Otherwise, an arbitrary replacement is selected for an element contained in the activity list: As regards its application, again all associated dependencies and precedence constraints have to be observed.

Fitness and Selection

The quality of an activity list is evaluated by converting it into the corresponding schedule and analyzing the associated costs by use of the predefined schedule evaluation function.

Table 2: Portion of the identified optimization potential that could be tapped by the genetic algorithm within limited time

	Limit	Process Complexity		Resource Complexity		Left-Shifts		Baseline Schedule		Overall
		low	high	low	high	yes	no	tight	wide	
Small	5 sec	99.15%	100.00%	100.00%	99.15%	99.37%	99.78%	99.56%	99.59%	99.58%
	15 sec	99.63%	100.00%	100.00%	99.63%	99.69%	99.93%	99.81%	99.81%	99.81%
Large	5 sec	55.63%	65.01%	82.38%	38.27%	53.27%	67.38%	60.06%	60.58%	60.32%
	15 sec	70.99%	76.48%	89.21%	58.26%	67.16%	80.31%	74.64%	72.83%	73.74%
	45 sec	81.80%	85.65%	92.58%	74.87%	77.99%	89.46%	84.01%	83.43%	83.72%

This way, activity lists are made comparable. In each step of evolution, a new generation is derived from the previous one by combining the best individuals (the survivors) with new ones (their children) generated through the application of crossover and mutation to the fittest activity lists. In order to avoid the convergence to a local optimum, we randomly replace individual members of the generated population with the initial activity list λ_0: The probability for such replacements is decreasing with the continuing progress of evolution. Due to the significant role of randomness in the selection and mutation of solutions, it is made sure that the exact optimum is identified by the GA at least within an infinite time horizon.

3.2 Computational Experiments

The above algorithm has been implemented in a Java-based rescheduling engine: The realization of the RCPSP-specific operators is based on the GA proposed by Hartmann [1998]. As there are currently neither instances of nor testset generators for reactive scheduling problems available [Policella and Rasconi, 2005], a framework for the parameterized generation of DM problems has also been developed: Normalized versions of network complexity, resource factor and resource strength (as proposed by Kolisch et al. [1995]) as well as parameters describing the characteristics of baseline schedules and disruptions can be used to define the problem structure.

The generated instances consist of a baseline schedule and several disruptions occurring during its execution. Based on the possibility to assign a due date δ_i to an activity $i \in \mathcal{A}^+$, the goal of optimization is to minimize the sum of the overall process tardiness $\sum_{i \in \mathcal{A}} \max(0, \beta_i + d_i - \delta_i)$ and the number of schedule modifications: Each modification is assumed to cause three times the costs of one time unit of tardiness.

As regards complexity aspects, the following configurations have been used to generate 16 different problem classes:

- *Low/High Process Complexity.* Based on this parameter it can be specified whether the scheduled activities are linked by few or by many precedence constraints.
- *Low/High Resource Complexity.* The aspects of resource requirements and resource availability are combined in this parameter: Low resource complexity corresponds to the existence of few requirements and the availability of many resource entities, high complexity to the opposite.
- *With/Without Left-Shifts.* If an activity is scheduled to start earlier than it did in the original schedule, this is considered a left-shift. Whether such modifications are valid or not can be defined by the assignment of appropriate lower bounds to the activity starting times.
- *Tight/Wide Baseline Schedule.* The distribution of starting times and the amount of incorporated slack time are combined in this parameter: In a tight (wide) schedule, activities (do not) start at the earliest possible point and many (only few) processes are executed simultaneously.

As regards the size of the generated instances, small and large cases were distinguished: The former consist of one process containing 10, the latter of one process containing 100 activities. For each problem class, 10 instances of both sizes were generated by random; with a probability of $P = 0.1$, for each activity one of the execution alternatives mentioned in Section 2.1 was inserted by the generator. In each case, a disruption was injected immediately at the start of execution by doubling the durations of half the activities.

As there is currently no benchmark data available for DM problems, it was not possible to compare the proposed GA to existing approaches. Instead, the fast convergence of schedule quality towards the optimal or at least a good solution is illustrated. Table 2 summarizes how much of the identified optimization potential could be tapped within limited time[1].

For each of the *small instances*, first the exact optimum was identified based on a deterministic procedure: The difference between the costs associated with the disrupted schedule S^Δ and the *optimal* schedule $S^\star$ defines the optimization potential. For the evaluation of the GA, 10 runs were conducted for each of the generated cases and each of the regarded time limits: Let S° denote the schedule resulting from such limited optimization. The figures listed in Table 2 thus correspond to the average value of $\frac{f(S^\Delta)-f(S^\circ)}{f(S^\Delta)-f(S^\star)}$: Basically, they reveal that in almost any case the optimum could be identified already within the first 5 or 15 seconds of optimization.

As even by use of the most powerful exact methods (see [Laborie, 2005] for a recent approach) hard scheduling problems of the considered size are not tractable in reasonable time [Hartmann, 1998; Alcaraz *et al.*, 2003], an alternative procedure has been chosen for the *large instances*. Instead of identifying the actual optimum, the best solution that could be found during (1) all GA runs and (2) an additional run limited to 10 minutes was taken as a reference. This approach was motivated by the observation that large improvements can mainly be made within the first generations: The typical development of the costs associated with the best known schedule throughout 10 minutes is depicted in Figure 1. Correspondingly, the figures listed in Table 2 show how close the GA could get to the *best known* solution within 5, 15 and 45

[1]The used problem instances and more detailed evaluation results can be obtained from http://anonymous.for.review/evaluation.html

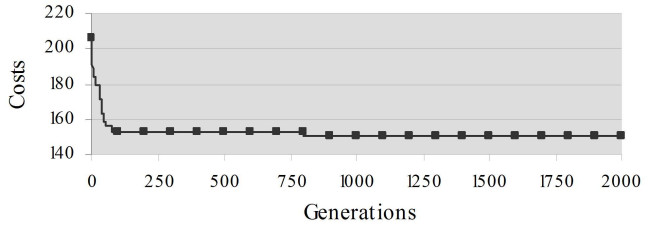

Figure 1: Reduction of Schedule Costs during Optimization

seconds. The following observations can be made:

- Due to the fact that the existence of many precedence constraints narrows the search space, the GA converges faster for problems with high process complexity.
- Resource complexity has considerable impact on the time required for the conversion of activity lists into schedules. A low complexity value thus means that more schedules can be analyzed within the available time and that the GA converges faster, therefore.
- The possibility of left-shifts extends the space of available rescheduling options: The genetic algorithm performs better if only activity postponements are valid.
- The aspects of slack time and process simultaneity have no significant impact on the speed of convergence.

The overall values indicate that the assumption of fast convergence to a good schedule quality holds: Particularly the result that about 75% of the optimization potential identified in more than 20 minutes could be tapped within the first 15 seconds is promising. Regarding the fact that the current implementation does not consider any domain-specific knowledge and might be optimized further, this indicates that the proposed algorithm might well be applied to operative DM in realistic scenarios.

4 Conclusions and Future Work

This paper described how the generic framework of the Resource-Constrained Project Scheduling Problem can be extended by the concept of alternative activities, to make its application in realistic disruption management problems possible: The *x-RCPSP* is based on a distinction between active and inactive activities as well as the definition of valid activity substitutions and associated constraints. Beside the modeling framework, we presented a novel genetic algorithm for the solution of the proposed generalization of the RCPSP. Its evaluation proved the fast convergence of schedule quality towards the optimal or at least a good solution.

References

[Alcaraz *et al.*, 2003] J. Alcaraz, C. Maroto, and R. Ruiz. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. *Journal of the Operational Research Society*, 54(6):614–626, 2003.

[Artigues *et al.*, 2003] C. Artigues, P. Michelon, and S. Reusser. Insertion techniques for static and dynamic resource constrained project scheduling. *European Journal of Operational Research*, 149:249–267, 2003.

[Barták, 1999] R. Barták. Conceptual models for combined planning and scheduling. In *Proceedings of CP99 Workshop on Large Scale Combinatorial Optimisation and Constraints*, pages 2–14, 1999.

[Beck and Fox, 2000] J.C. Beck and M.S. Fox. Constraint directed techniques for scheduling with alternative activities. *Artificial Intelligence*, 121:211–250, 2000.

[Błazewicz *et al.*, 1983] J. Błazewicz, J. K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling projects to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[Brucker *et al.*, 1999] 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, 1999.

[Carr, 2004] F. R. Carr. *Robust Decision Support Tools for Airport Surface Traffic*. PhD thesis, 2004.

[Clausen *et al.*, 2001] J. Clausen, J. Hansen, J. Larsen, and A. Larsen. Disruption management. *ORMS Today*, 2001.

[Elkhyari *et al.*, 2004] A. Elkhyari, C. Guret, and N. Jussien. Constraint programming for dynamic scheduling problems. In Hiroshi Kise, editor, *ISS'04 International Scheduling Symposium*, pages 84–89, Japan, 2004.

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

[Hartmann, 2001] S. Hartmann. Project scheduling with multiple modes: A genetic algorithm. *Annals of Operations Research*, 102(1-4):111–135, 2001.

[Hindi *et al.*, 2002] K.S. Hindi, H. Yang, and K. Fleszar. An evolutionary algorithm for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(5):512–518, 2002.

[Kolisch and Hartmann, 2005] R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 2005.

[Kolisch *et al.*, 1995] 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.

[Laborie, 2005] P. Laborie. Complete MCS-based search: Application to resource constrained project scheduling. In *IJCAI-05*, pages 181–186, 2005.

[Policella and Rasconi, 2005] N. Policella and R. Rasconi. Testsets generation for reactive scheduling. In *Workshop on Experimental Analysis and Benchmarks for AI Algorithms*, 2005.

[Yu and Qi, 2004] G. Yu and X. Qi. *Disruption Management: Framework, Models and Applications*. World Scientific Publishing, Singapore, 2004.

[Zhu *et al.*, 2005] G. Zhu, J.F. Bard, and G. Yu. Disruption management for resource-constrained project scheduling. *Journal of the Operational Research Society*, 56, 2005.