Extending the RCPSP for Modeling and Solving Disruption Management Problems

Jürgen Kuster, Dietmar Jannach, and Gerhard Friedrich [1]

Abstract. This paper introduces an extension to the well-established Resource-Constrained Project Scheduling Problem for the comprehensive description of disruption management problems. This conceptual framework employs the concept of alternative activities to consider both the temporal shift of activities or the reallocation of resources and switches from one valid process variant to another one. Activities can be serialized or parallelized, process steps can be inserted or removed and durations as well as resource requirements can be modified dynamically during optimization. Focusing on the domain of the aircraft turnaround as the most important airport ground process, we illustrate how the Extended RCPSP (*x-RCPSP*) can be applied for decision support. A specific evolutionary algorithm is presented that identifies good-quality solutions to relatively large disruption management problems within only a few seconds. The results of the evaluation illustrate fast convergence on good or optimal schedules and serve as a basis for the development of future problem solving algorithms.

1 Introduction

Airport ground processes are a popular example of operations that are particularly susceptible to disruption or failure. The high levels of time and resource dependencies make such complex processes prone to unforeseen events and disruptions. Managing their execution and responding to the problems that occurred during planned and scheduled operations is a difficult task. The dependencies between parallel and interrelated activities have to be considered in context, and decisions must usually be made within short time. Human operators who are responsible for the management of disruptions base their decisions on the qualitative assessment of the current situation according to their individual experience and domain knowledge. Accordingly, two elements are crucial for the quality of decisions: the amount and precision of available information that

describes the current situation and the ability of the human operator to analyze the given data. Subsequently both of these aspects must be considered if the quality of decisions is to be improved.

With respect to the former condition, the quality and availability of information may be addressed by various initiatives directed at intensified information sharing and increased information awareness. Initiated by the Federal Aviation Administration (FAA) and adopted by EUROCONTROL, the concept of Collaborative Decision Making (CDM, see [1, 2] for example) is among the most popular and widespread techniques. By making relevant data accessible to all actors involved in the air traffic processes, conditions for the coordination of decisions and dispositions, as well as *common situational awareness*, is improved. CDM therefore focuses on the availability of information. The need for improvement in data quality is mainly addressed through the implementation and realization of appropriate information systems. ALLEGRO (see [3, 4] for instance), which was implemented with the goal of tracking all events occurring during the execution of airport ground processes as well as gathering the respective time stamps in a comprehensive database at Deutsche Lufthansa AG, can be mentioned as a successful example.

With regard to the latter aspect – the analysis of available data – the application of Artificial Intelligence (AI) provides promising opportunities. Intelligent systems can be employed to preprocess available information, to analyze the current situation and to propose (ideally optimal) interventative actions to the human operator. However, although recent efforts to increase information awareness have led to the emergence of integrated databases, and despite the fact that increases in computational power and algorithmic efficiency continue to aid the development of real-time systems, current Decision Support Systems (DSS) often suffer from a lack of expressive power and are thus only consider basic forms of intervention. This is mainly due to the fact that (to the best of our knowledge) there is currently no conceptual framework available for the formal description of disruption management problems which takes both temporal shifts and resource reallocations as well as dynamic process execution paths into account in cases of unexpected disruptions.

This work intends to overcome this deficiency by proposing an approach for modelling disruption management problems in a comprehensive way. Furthermore, an algorithm is presented that can identify good-quality solutions in a

short period of time (usually in real-time or after several seconds). The respective research activity was motivated by the insights and findings of a study conducted in collaboration with Deutsche Lufthansa AG. While evaluating the potential use of intelligent systems in the Hub Control Center (HCC) at Frankfurt International Airport, the elementary requirements of turnaround-related decision support systems were analyzed in a practical context.

The paper is structured as follows. In Section 2 Disruption Management (DM) is defined and an overview on related work is provided. Section 3 describes the exemplary Turnaround Process as the most typical airport ground process. In Section 4, the *x-RCPSP* is introduced as a conceptual framework for the comprehensive description of disruption management problems. By extending the well-established Resource-Constrained Project Scheduling Problem (RCPSP) with the notion of alternative activities, it is possible to formulate a wide variety of potential schedule modifications that exceeding simply shifting activities temporally or reallocating resources. In Section 5, we illustrate how the modeling framework provided by the *x-RCPSP* can be applied for decision support. Here, a specialized Genetic Algorithm (GA) with specific crossover and mutation operators is presented for this purpose together with a performance evaluation that reveals its fast convergence on good or optimal solutions. Section 6 summarizes the contributions of this paper.

2 Disruption Management

This section starts by informally defining disruption, intervention, disruption management and the DM problem. The second part of the section provides a brief overview of the most relevant literature in the research area of disruption management.

2.1 Definitions

Definition 1 (Disruption). *A disruption is an unforeseen event that occurs during the execution of planned operations and triggers a deviation from an existing schedule.*

Such deviations are associated with (unanticipated, not necessarily monetary) costs whenever predetermined plans are optimized according to some spe-

cific criteria. In many practical situations, manifold options allow the system to dynamically adapt to the modified situation and to repair broken schedules.

Definition 2 (Intervention). *An intervention is a modification of a current or future schedule, aimed at minimizing the negative impacts associated with a disruption.*

Practically important forms of intervention include (1) the *temporal shift* of one or several activities, (2) a change in the *allocation of resources* and (3) switches from one valid *process variant* to another one. The latter form includes options such as the modification of activity durations, the insertion or removal of operations, the serialization or parallelization of process steps, and so forth.

Definition 3 (Disruption Management). *Disruption management (DM) is the process of selecting an appropriate set of interventions after the occurrence of a disruption.*

Typically, the central aim of disruption management is to find a way to get back on track [5] and to optimize the relation between planned and real processes by updating future schedules in a way that minimizes the negative impact of the disruption.

Definition 4 (Disruption Management Problem). *A disruption management problem consists of (1) a given plan for the future, (2) a particular disruption, (3) various potential interventions and (4) an objective for the adaptation of plans. It is solved through the process of disruption management.*

2.2 Related Work

Since the 1980s, the problem of responding to unforeseen schedule disruptions has started to attract attention. Particularly the development of the popular factory scheduling systems ISIS [6] and its successor OPIS (Opportunistic Intelligent Scheduler, see [7]) helped to define and separate the concepts of *scheduling* and *rescheduling*. While the former scheduler focused generating plans using *predictive scheduling* (or *baseline scheduling*), the latter one was intended to be able to react to disruptions by implementing *reactive scheduling*. OPIS, for instance, heuristically selects an appropriate rescheduling method, such as rescheduling

tasks dependent on congested resources, changing the previous resource allocation decisions or postponing a subset of operations until the disrupted schedule is free of conflicts.

Another important step in the consideration of schedule disruptions was made with the GERRY scheduling and rescheduling component [8] of the Ground Processing Scheduling System (GPSS, see [9]) developed by the NASA Kennedy Space Center. While planning the process of preparing space shuttles for launch (which basically involves inspection, repair, refurbishment, etc.), an existing (complete but infeasible) schedule is taken as a starting point and, using a constraint-directed approach which is similar to the *min-conflicts heuristic* [10], iteratively optimized to increase the quality of the solutions . In order to avoid early convergence on a poor local optimum, simulated annealing (see [11]) is also applied during this process.

More recently, the distinction between *predictive-reactive* and *proactive-reactive scheduling* has been introduced. The former strategy focuses on mere repairing the original schedule by taking an unexpected disruption into account (see [12], for example). In the latter strategy, schedules are also updated with the intention of being *robust*: safety or slack time is incorporated to absorb anticipated disturbances, meaning reactive procedures need only be reapplied in the case of severe or unexpected situations (see [13], for example).

A comprehensive overview of rescheduling approaches in the context of manufacturing systems can be found in [14]. The authors define rescheduling as the process of responding to schedule disruptions and propose a framework of strategies, policies and methods for the classification of the approaches presented in literature. Furthermore, they discuss aspects of the rescheduling environment as well as typical performance measures. Aytug et al. [15] provide another and more recent review of the extensive area of rescheduling, discussing the purpose of scheduling and defining a taxonomy for different types of uncertainty.

Originally, responding to disruptions was primarily regarded in terms of scheduling and rescheduling. It was only recently that the concept of *Disruption Management* emerged. The primary difference between DM and rescheduling is that rescheduling focuses on the identification of a schedule which is optimal in terms of the original objective function whereas disruption management also aims to minimize the deviation of the updated schedule from the original one [5]. The Descartes (Decision Support for Integrated Crew and Aircraft Re-

covery) project, a cooperative project of British Airways, Carmen Systems and the Technical University of Denmark, represented an important step forward in the formation of the new research area and provided the basis for much of the literature on DM (see [16–18], for example).

Beside air traffic and airline-related scheduling, the domains of machine scheduling and production planning have been at the center of research in disruption management [19]. Bean et al. [20] were among the first to consider deviation costs in their approach to matchup scheduling, which is based on the idea of identifying an updated schedule that converges with the original one at some early point in the future. While Clausen et al. [16] discuss disruption management in the execution of shipbuilding processes, Xia et al. [21] investigate DM in the context of a two-stage production and inventory system, evaluating solutions for fixed and flexible setup epochs as well as different forms of penalty functions. Yang et al. [19] consider cost and demand disruptions occurring on a single-product manufacturing plant and propose a pseudo-polynomial dynamic programming procedure for the general cost case and present advanced solution procedures for specific forms of cost functions. Additional information and comprehensive overviews of DM in the context of production planning can be found in [5] and [22]. Apart from the areas of application mentioned above, disruption management plays a crucial role in the context of many other real-world processes. Research, for example, has been conducted in the domains of telecommunication [16], project management [22, 23, 12], supply chain coordination [22, 24, 25] and logistics management [22].

3 The Aircraft Turnaround Process

In the domain of air traffic, disruptions occur frequently due to dependencies between many different service providers, shared resources and external factors such as weather conditions or legal constraints. Even worse, delays resulting from disruptions are typically propagated throughout the entire air traffic network, causing further schedule deviations far from and long after the original disturbance. The high costs associated with delays (according to EUROCONTROL [26] airlines have to bear delay costs of 72 Euros per minute for delays of more than 15 minutes, see [27] for additional figures) have led to particu-

lar interest in DM in the practical context of air traffic and airline operations management, as discussed in Subsection 2.2.

Aircraft turnaround is one major source of delay in the domain of air traffic [28]. Combining all activities related to the postprocessing of incoming and preprocessing of outgoing flights, turnaround can be regarded as the process of preparing an aircraft for a flight to another destination airport while on ground. It is characterized by a high number of simultaneously executed operations [29] and the involvement of many service providers [30]. Supplemented by the fact that a turnaround can be executed with many different variations, the process is extremely complex to manage and thus interesting for further consideration.

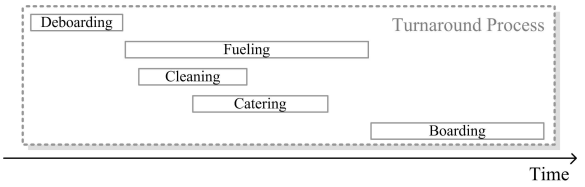

Fig. 1. Simplified version of the turnaround process

In order to prevent the examples from being overly complex, a simplified version of turnaround is presented in this paper. With a focus on passenger-related core processes [3], its structure can be described as follows. After the plane reaches its destination gate or position, the incoming passengers leave the aircraft before it is fueled, cleaned and restocked (catering) simultaneously. When these activities are finished, the outgoing passengers enter the plane which then departs for the runway. Figure 1 illustrates an exemplary instance of this process.

Let us consider typical disruptions and their effects by examining two different scenarios. In the first one, we assume that some sort of disturbance occurs prior to deboarding. The most typical example for such a disruption is an arrival delay. If the resulting schedule deviation is large enough, the process start is delayed to the extent that it has effects on the entire turnaround, eventually causing a corresponding departure delay. This might cause another arrival delay and additional problems at the next destination of the aircraft. This process is illustrated in Figure 2 where the thunderbolt symbolizes the disruption, grey

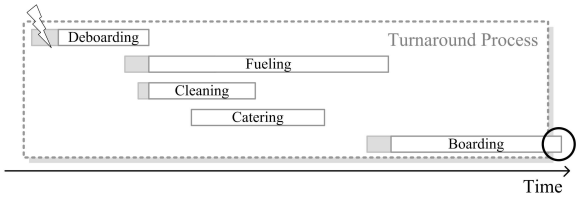

Fig. 2. Disrupted instance of the turnaround process

bars represent the original plan and white bars indicate the modified plan. The circle shows where the actual problem lies.

As the airline responsible for the turnaround typically faces penalties for delays, it is usually interested in taking some form of interventative action. In this case, it would be desirable to accelerate the process at some point. A good idea might be to assign additional resources along the critical path. An additional tanker could reduce the time required for fueling or additional buses could accelerate disembarking and boarding if the airplane is positioned on the tarmac. Another option might be to reallocate resources. Relocating the aircraft to a gate position is often a good way to speed up the processes of passengers leaving and entering the plane. Lastly, it might be helpful to modify the structure of the executed process. One of the options applied in practice is to refuel the aircraft while passengers are boarding under the supervision of the fire brigade.

The effects of a disruption do not necessarily appear in the disrupted process itself. In the second scenario we assume that there is sufficient slack time available to compensate for an irregularity occurring prior to the start of the turnaround. However, this time the turnaround is coupled to another one. Passengers arriving on the incoming flight need to depart on an outgoing flight. Such a dependency causes the disruption to have noticeable effects. As the passengers arrive late they either miss their flight or the outgoing plane has to wait for them and thus also experiences delay. Figure 3 illustrates this scenario, with the passenger transfer representing the linking process step.

Importantly, both a delay for the outgoing aircraft and passengers missing their flights cause costs. Consider, for instance, the necessary expenses for alternative bookings and overnight stays as well as the costs arising due to losses in customer satisfaction. Therefore, a form of intervention is required, accelerating either disembarking or the passenger transfer. This could be achieved through

the assignment of additional buses which carry transfer passengers from the incoming flight directly to the outgoing aircraft. Another option is to place the affected planes close to one another. Finally, if no other option is available or the delay is too large, it has to be evaluated if it is better to wait and delay the outgoing flight or to proceed as planned and rebook the affected passengers.

For the examples presented in the remainder of this paper, it is assumed that three forms of intervention are available apart from the temporal shift of activities and the reallocation of resources. First, it is possible to accelerate disembarking by assigning additional buses. Second, it is possible to shorten cleaning if the cabin is then inspected by the cabin crew prior to the passengers boarding. Finally, it is possible to parallelize fueling and boarding if the fire brigade is available for supervision.

Note that these options describe only those forms of intervention that were inspired by the observations made during the field study conducted in cooperation with Deutsche Lufthansa AG. Depending on the given situation, other kinds of repair actions might be possible in practice. We decided to focus on this set of interventions as they are common on the one hand and extensive enough to illustrate the shortcomings of current approaches with respect to modeling (see Subsection 4.1) and the expressive power of the proposed framework (see Subsection 4.5) on the other hand.

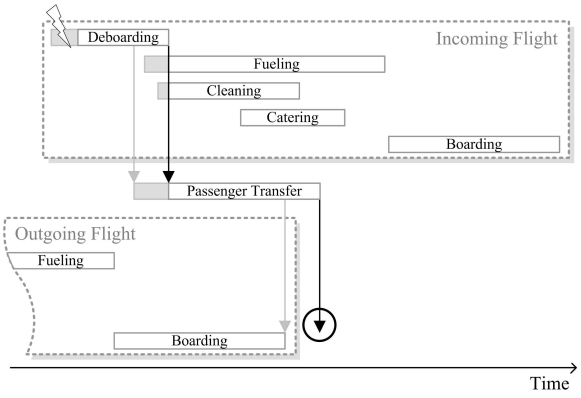

Fig. 3. Another disrupted instance of the turnaround process

We now turn our attention to how the example turnaround process can be modeled with all its variability, before finally demonstrating how the associated disruption management problems can be effectively solved.

4 Modeling Disruption Management Problems

This section introduces methods for formally describing DM problems. The conceptual framework for Resource-Constrained Project Scheduling Problems is presented and its applicability to practical disruption management is evaluated. Motivated by a lack of possibilities to describe potential process variations, the Extended RCPSP (*x-RCPSP*) is introduced, illustrating the expressive power of of alternative activities through several modeling patterns. Finally a formal description of the simplified turnaround process is presented.

4.1 The Resource-Constrained Project Scheduling Problem

The Resource-Constrained Project Scheduling Problem (RCPSP, see [31, 32]) generalizes various forms of production-specific problems (such as the job shop, the flow shop and the open shop problem) and provides a powerful framework for the formal description of scheduling problems. The process of solving an RCPSP aims to schedule arbitrarily linked activities, which are processed on arbitrary resource types, according to some predefined optimization criteria (such as the minimization of the overall project duration).

Formally, a RCPSP can be described as follows. A project consists of a set of activities $\mathcal{A} = \{0, 1, ..., a, a+1\}$: The first and the last element represent abstract start and end activities, having a duration of 0 and no resource requirements associated. Each remaining $i \in \mathcal{A}$ has a non-negative duration d_i. Activities are ordered using precedence constraints. The existence of an element $p_{i,j}$ in the set $\mathcal{P}$ states that activity j must not start before the execution of activity i is finished. The execution of activities is based on a set of renewable resource types $\mathcal{R} = \{1, ..., r\}$, where for each type $k \in \mathcal{R}$ a constant amount of c_k units is available. A set of resource requirements defines the relation between resources and activities. Activity i requires $q_{i,k} \in \mathcal{Q}$ units of resource type k throughout its execution. The scheduling process should produce a vector of starting times $(\beta_1, ..., \beta_a)$ for which (1) $\beta_i \geq 0, i \in \mathcal{A}$, (2) $\beta_i + d_i \leq \beta_j, p_{i,j} \in \mathcal{P}$ and (3) $\sum_{i \in \mathcal{A}(t)} q_{i,k} \leq c_k$ for any resource type $k \in \mathcal{R}$ at any time t, where $\mathcal{A}(t)$

corresponds to all activities simultaneously executed at t. Typically, some form of optimization criterion must also be considered when searching for a solution.

The RCPSP is used as a starting point for the formal description of disruption management problems for the following reasons:

- Defining an RCPSP is easy and intuitive. The modeling constructs mentioned above make it possible to describe entities and corresponding relationships with a high level of abstraction.
- Unlike the production-specific job shop, flow shop or open shop problems, it imposes no restrictions on the number of resource entities, the structure of the processes or their associated resource requirements.
- Many highly efficient algorithms are available for problem solution and optimization. Meta-heuristic approaches, for example, can provide good results within a short time. Local search, tabu search, evolutionary and ant colony optimization have all been applied in the context of the RCPSP. A comprehensive survey and evaluation of the performance of these techniques can be found, for instance, in [33].

Despite its strengths, however, the Resource-Constrained Project Scheduling Problem is not powerful enough to cope with the typical requirements of disruption management. Although it can often identify optimal activity starting times and optimal resource allocations, the RCPSP provides no opportunity for the reconsideration of previous choices with respect to activity execution, preventing the chosen process execution paths from varying dynamically. When considering the types of intervention mentioned in Subsection 2.1, it is not possible to perform a switch from one valid process variant to another.

4.2 Extending the RCPSP for Disruption Management

To overcome this limitation, we propose extending the RCPSP with new constructs. To the best of our knowledge, the only other approach that allows for additional flexibility during optimization is Multi-Mode RCPSP (MRCPSP, see [34]). By permitting the dynamic alternation of activity execution modes, it is possible to consider potential duration and resource requirements associated with changes. Still, it is clear that the concept of mode alternation is not sufficient for the description of the entire range of potentially relevant process variations mentioned above.

When surveying previous work on DM in the context of the RCPSP, it is obvious that the vast majority of research has focused on responding to disruptions by shifting activity start times and changing the allocation of resources. Artigues et al. [35] discuss adding an unexpected activity to the current schedule. They consider the arrival of each such activity as a disruption. Elkhyari et al. [36] use explanations to handle over-constrained networks in dynamic scheduling problems. Zhu et al. [23] present a classification scheme for disruptions and describe a hybrid mixed integer programming/constraint propagation approach to handle them. In contrast to these works, we focus on the extension of existing methods to formally describe and consider additional forms of intervention during optimization.

Disruption management as defined in Subsection 2.1 can be regarded as a combination of planning and scheduling. The consideration of potential changes in process execution paths corresponds to the former, whereas the consideration of temporal shifts and resource reallocations corresponds to the latter part of the problem. As the scheduling-specific options can be tackled using the constructs provided by the classical Resource-Constrained Project Scheduling Problem, it is particularly important to focus on the planning part of the problem when adopting the RCPSP for DM. More specifically, it has to be extended by introducing constructs which make the formal description of switches from one process variant to another one possible.

We base our approach on the concept of *alternative activities* and *activity dependencies*. Each activity might have one or more substitutes that can be used during optimization. As the execution of one activity is often dependent on the execution of others, various forms of dependencies might be associated with the execution of replacements. We combine the planning and the scheduling part of the DM problem into one single conceptual model by describing potential modifications directly within the process.

Up to the present, little research has focused on considering alternative activities in scheduling problems. One of the most important approaches can be found in the area of constraint-directed scheduling. Beck and Fox [37] introduce 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, our goal is to incorporate alternative activities into the conceptual framework of the RCPSP. We propose

a compact and intuitive way of modeling variable process execution paths which is entirely independent of the underlying search procedure.

4.3 The x-RCPSP

Model. In the Extended Resource-Constrained Project Scheduling Problem (*x-RCPSP*), the implementation of the concept of alternative activities is based on distinguishing between *active* and *inactive* elements. Only the former are actually considered during the generation and optimization of the schedule. Activities can be activated and deactivated dynamically according to predefined rules. Note that each such change of the activation state results in a new instance of the classical RCPSP. For its solution, well-established approaches can be applied (see [33], for example).

Formally, the *x-RCPSP* can be described as follows (see Subsection 4.1 for the definition of the underlying RCPSP). A process is defined by a set of *potential* activities $\mathcal{A}^+ = \{0, 1, ..., a, a+1\}$. The first and the last element represent *mandatory* abstract start and end activities with a duration of 0 and no associated resource requirements. Each remaining $i \in \mathcal{A}^+$ has an associated non-negative duration d_i. *Active activities* form the subset $\mathcal{A} \subseteq \mathcal{A}^+$ and inactive activities are grouped in the corresponding set difference $\mathcal{A}^+ \setminus \mathcal{A}$. The *initial* activation state and process execution path are described by another subset $\mathcal{A}_0 \subseteq \mathcal{A}$. The execution of activities is based on a set of renewable resource types $\mathcal{R} = \{1, ..., r\}$ where a constant amount of c_k units is available of each type $k \in \mathcal{R}$. The following constructs can be used for the description of activity/resource dependencies:

- *Precedence Constraints.* The order of activities is described by use of precedence constraints. The existence of an element $p_{i,j}$ in the set of potentially relevant dependencies $\mathcal{P}^+$ states that activity $j \in \mathcal{A}^+$ must not start before activity $i \in \mathcal{A}^+$ is finished. The subset $\mathcal{P} = \{p_{i,j} \in \mathcal{P}^+ | i, j \in \mathcal{A}\}$ groups all precedence constraints for which both the predecessor and the successor are active.
- *Resource Requirements.* The relationship between activities and resource types is defined by use of resource requirements. In a two-dimensional array $\mathcal{Q}^+$, all activities $i \in \mathcal{A}^+$ are combined with all resources $k \in \mathcal{R}$. A constant amount of $q_{i,k}$ units of a resource type $k \in \mathcal{R}$ is required throughout the execution of activity $i \in \mathcal{A}^+$. The subset $\mathcal{Q} = \{q_{i,k} \in \mathcal{Q}^+ | i \in \mathcal{A}\}$ contains only those elements for which the associated activity is active.

The following constructs are available for the formal description of activity relations and dependencies:

- *Activity Substitutions.* The existence of an element $x_{i,j}$ in the set of potential activity substitutions $\mathcal{X}^+$ states that the replacement of activity $i \in \mathcal{A}$ with activity $j \in \mathcal{A}^+ \backslash \mathcal{A}$ (i.e. the deactivation of i for the activation of j) represents a legal form of process variation.
- *Activity Dependencies.* Activating or deactivating an activity might have an impact on the state of other activities. $\mathcal{M}^+$ therefore defines the mandatory implications of a deliberate activity state modification. The existence of $m^{\oplus}_{i,j}$ ($m^{\ominus}_{i,j}$) $\in \mathcal{M}^+$ indicates that activity $j \in \mathcal{A}^+$ has to be (de)activated upon the (de)activation of $i \in \mathcal{A}^+$; the existence of $m^{\sqsubset}_{i,j} \in \mathcal{M}^+$ indicates that j has to be deactivated upon the activation of i; finally, the existence of $m^{\sqsupset}_{i,j} \in \mathcal{M}^+$ indicates that j has to be activated upon the deactivation of i.

The *x-RCPSP* represents a generalization of the classical Resource-Constrained Project Scheduling Problem. Any instance of an *x-RCPSP* with $\mathcal{X}^+ = \mathcal{M}^+ = \emptyset$ can be converted into an equivalent RCPSP. Both problem classes share a reasonably high level of abstraction at which problem instances are formulated. Therefore, defining an *x-RCPSP* is essentially as easy and intuitive as defining a classical RCPSP. It has to be stated, however, that creating and maintaining the respective model might be more difficult if large numbers of potentially relevant options have to be described. This issue can be addressed through the provision of appropriate (potentially graphically supported) modeling tools and constructs in practical scenarios.

Consistency Criteria. The following section describes the two types of consistency criteria for activity substitutions that have to be fulfilled in any *x-RCPSP*.

- *Dependency Consistency.* An activity must never be activated and deactivated at the same time during the a substitution. More formally, let us consider $\mathcal{A}^{\oplus}_i$ as the set of all activities activated upon the activation of activity i, $\mathcal{A}^{\sqsupset}_i$ as the set of activities activated upon its deactivation, $\mathcal{A}^{\sqsubset}_i$ as the set of activities deactivated upon its activation and $\mathcal{A}^{\ominus}_i$ as the set of activities deactivated upon its deactivation:

$$\mathcal{A}^{\oplus}_i = i \cup \mathcal{A}^{\oplus}_j, m^{\oplus}_{i,j} \in \mathcal{M}^+ \tag{1}$$

$$\mathcal{A}_i^{\sqsupset} = \mathcal{A}_j^{\oplus}, m_{i,j}^{\sqsupset} \in \mathcal{M}^+ \tag{2}$$

$$\mathcal{A}_i^{\sqsubset} = \mathcal{A}_j^{\ominus}, m_{i,j}^{\sqsubset} \in \mathcal{M}^+ \tag{3}$$

$$\mathcal{A}_i^{\ominus} = i \cup \mathcal{A}_j^{\ominus}, m_{i,j}^{\ominus} \in \mathcal{M}^+ \tag{4}$$

Based on these sets, we now can define $\mathcal{A}^{\boxplus}(x_{i,j}) = \mathcal{A}_i^{\sqsupset} \cup \mathcal{A}_j^{\oplus}$ as the set of all activities activated upon the application of the substitution $x_{i,j} \in \mathcal{X}^+$. Accordingly, $\mathcal{A}^{\boxminus}(x_{i,j}) = \mathcal{A}_i^{\ominus} \cup \mathcal{A}_j^{\sqsubset}$ describes which activities are deactivated upon the application of $x_{i,j}$. The activity dependencies can now be defined as consistent if $\mathcal{A}^{\boxplus}(x_{i,j}) \cap \mathcal{A}^{\boxminus}(x_{i,j}) = \emptyset$ for all $x_{i,j} \in \mathcal{X}^+$. In that case, no activity is ever allowed to be activated and deactivated at the same time.

– *Precedence Consistency.* As no activity can precede *and* succeed another activity at the same time, there must be no cyclic dependencies in the set of active precedence constraints $\mathcal{P}$. Given the application of $x_{i,j} \in \mathcal{X}^+$ on a set of active activities $\mathcal{A}_0$, the resulting set of active activities can be defined as $\mathcal{A}_1 = \mathcal{A}_0 \cup \mathcal{A}^{\boxplus}(x_{i,j}) \setminus \mathcal{A}^{\boxminus}(x_{i,j})$. An *x-RCPSP* is considered precedence consistent if no cycles exist within the set of precedence constraints associated with any potential $\mathcal{A}_1$. Note, however, that the set of *potentially* relevant precedence constraints might very well contain cyclic dependencies. This is, for example, the case whenever options for changing the execution order of activities are available. Consider a simple process consisting of three activities a, b and c which can be executed either in alphabetical (a, b, c) or in reverse order (c, b, a). To describe both potential execution paths using alternative activities, it is sufficient to replace b with two variants b_1 and b_2, of which (1) the former is a successor of a and a predecessor of c and (2) the latter is a successor of c and a predecessor of a (see Subsection 4.4 for further details). The precedence graph describing all activities in context is depicted in Figure 4, where dashed lines illustrate optional or alternative process execution paths. It can easily be observed that the activities form a circle in terms of precedence relations. This is, however, perfectly okay if at any time only b_1 *or* b_2 is activated. In that case, the respective set $\mathcal{P}$ is free of any cyclic dependencies.

Apart from these substitution-specific aspects, *requirement consistency* is another fundamental criterion for the consistency and solvability of the *x-RCPSP*. An activity must never require more resource entities than there are actually

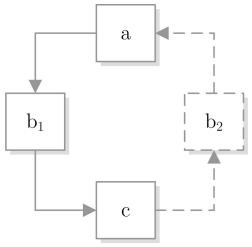

Fig. 4. Cyclic precedence relations in the *x-RCPSP*

available. Therefore, $q_{i,k} \in \mathcal{Q}^+$ has to be lower than or equal to c_k for any activity $i \in \mathcal{A}^+$ and any resource type $k \in \mathcal{R}$.

x-RCPSP Solutions. The process of solving an *x-RCPSP* is based on the identification of an optimal combination of activation state and activity starting times. Both elements are represented in a *schedule*, which essentially corresponds to a vector of starting times for all active activities. Let β_i denote the scheduled starting time for an activity $i \in \mathcal{A}$ and $\mathcal{A}(t)$ again describe the set of activities executed at a time t. A schedule can be considered valid if and only if

(1) $\beta_i \geq 0$ for any $i \in \mathcal{A}$,

(2) $\beta_i + d_i \leq \beta_j$ for any $p_{i,j} \in \mathcal{P}$,

(3) $\sum_{i \in \mathcal{A}(t)} q_{i,k} \leq c_k$ for any $k \in \mathcal{R}$ at any t and

(4) the activation state described by S can be derived from an original (valid) activation state through the application of substitutions defined in $\mathcal{X}^+$, taking the dependencies described in $\mathcal{M}^+$ into account.

Identifying the optimal solution of an RCPSP and its generalizations (such as the *x-RCPSP*) represents an NP-hard problem. This can be deduced from the fact that even very special cases such as the job shop scheduling problem are NP-hard in the strong sense [32].

4.4 Modeling Patterns for the x-RCPSP

This subsection discusses how several typical options of process variation can be modeled by use of alternative activities. For better readability we use a simplified notation in the subsequent descriptions:

(1) $i \rightarrow j$ stands for $p_{i,j} \in \mathcal{P}^+$,

Table 1. Mode Alternation in the *x-RCPSP*

	Original Process	Extended Process
$\mathcal{A}^+$	a, b, c	a, b_1, b_2, b_3, c
$\mathcal{P}^+$	$a \to b,$	$a \to b_1, a \to b_2, a \to b_3,$
	$b \to c$	$b_1 \to c, b_2 \to c, b_3 \to c$
$\mathcal{X}^+$	$\emptyset$	$b_1 \leftrightsquigarrow b_2, b_1 \leftrightsquigarrow b_3, b_2 \leftrightsquigarrow b_3$
$\mathcal{M}^+$	$\emptyset$	$\emptyset$

(2) $i \rhd n \times k$ for $q_{i,k} = n \in \mathcal{Q}^+$,

(3) $i \rightsquigarrow j$ for $x_{i,j} \in \mathcal{X}^+$,

(4) $i \leftrightsquigarrow j$ for $\{x_{i,j} \cup x_{j,i}\} \subseteq \mathcal{X}^+$,

(5) $i \oplus j$ for $m_{i,j}^{\oplus} \in \mathcal{M}^+$, etc.

Mode Alternation. A *mode* is a fixed combination of duration and resource requirements associated with the execution of an activity [34]. Activities for which more than one potential execution mode is defined are considered *multi-mode*. A *mode alternation* corresponds to the switch from a previously chosen mode to another one. It has already been mentioned that the Multi-Mode RCPSP (see Subsection 4.2) is a generalization of the classical RCPSP that considers such mode alternations (in addition to activity starting times and resource allocations) during optimization. Here we illustrate how potential switches between various execution modes can be described using the *x-RCPSP*, which itself generalizes the MRCPSP.

Let us consider a simple process consisting of three sequentially executed activities a, b and c where the execution mode of b is variable. The first mode has the original time and resource requirements. In the second mode, the execution of b takes longer but requires less resources, whereas the third mode executes more quickly but is more resource-intensive. As an *x-RCPSP*, each of these modes can be represented as a separate alternative activity. Instead of b, three options b_1, b_2 and b_3 are therefore modeled. They precede and succeed the same activities as b and differ only in time and resource requirements. A mode alternation corresponds to the activation of one alternative in exchange for the deactivation of another one. In $\mathcal{X}^+$, this indicates that each mode can be replaced by any other one. Table 1 summarizes the alterations to the original process model that make dynamic mode alternations possible.

Table 2. Resource Alternation in the *x-RCPSP*

	Original Process	Extended Process
$\mathcal{R}$	r_1, r_2	r_1, r_2, r_2'
$\mathcal{A}^+$	a, b, c	a, b_1, b_2, c_1, c_2
$\mathcal{P}^+$	$a \rightarrow b,$	$a \rightarrow b_1, a \rightarrow b_2,$
	$b \rightarrow c$	$b_1 \rightarrow c_1, b_2 \rightarrow c_1, b_1 \rightarrow c_2, b_2 \rightarrow c_2$
$\mathcal{X}^+$	$\emptyset$	$b_1 \nleftrightarrow b_2, c_1 \nleftrightarrow c_2$
$\mathcal{M}^+$	$\emptyset$	$b_1 \oplus c_1, b_1 \sqsubset c_2, b_2 \oplus c_2, b_2 \sqsubset c_1$
		$c_1 \oplus b_1, c_1 \sqsubset b_2, c_2 \oplus b_2, c_2 \sqsubset b_1$
$\mathcal{Q}^+$	$b \rhd 2 \times r_1, b \rhd 1 \times r_2,$	$b_1 \rhd 2 \times r_1, b_1 \rhd 1 \times r_2, b_2 \rhd 2 \times r_1, b_2 \rhd 1 \times r_2',$
	$c \rhd 3 \times r_2$	$c_1 \rhd 3 \times r_2, c_2 \rhd 3 \times r_2'$

Resource Alternation and Capacity Change. Apart from changes in activity execution and the associated resource requirements, the concept of mode alternation can be used to model additional forms of intervention. Both switches between alternative resource types and modifications to available resources can be accurately described using separate activity execution modes.

Regarding the former, additional (and alternative) resource types must first be introduced into the model. The model can then express that activities requiring the original resource type might also be executed using the potential substitute. Thus two alternative activities represented by separate execution modes with identical durations but different resource requirements replace the original activity. This approach provides extensive flexibility in the definition of possible alternatives, describing their effects on costs, durations and resource requirements on the activity level. Let us consider a process consisting of three activities a, b and c, sequentially executed using two resource types r_1 and r_2. b requires two units of r_1 and one unit of r_2, c requires three units of r_2. If the possibility to replace r_2 with an alternative resource type exists, an additional variant is introduced for each activity requiring r_2. Thus b_1 and c_1 are executed using r_2 whereas b_2 and c_2 are executed using the alternative r_2'. The exchange of a resource type corresponds to the synchronous switch of all related alternatives. The constraints in $\mathcal{M}^+$ ensure that at any time the same variants of all involved activities are executed. Table 2 summarizes the differences between the original and this extended process model.

Changes of resource capacities can be described in a similar fashion. Consider for example temporary increases of the number of available resource units.

Table 3. Changes of capacity in the *x-RCPSP*

	Original Process	Extended Process
$\mathcal{R}$	r_1	r_1, r_1^+
$\mathcal{A}^+$	a, b, c	a, b, c_1, c_2
$\mathcal{P}^+$	$a \rightarrow b$,	$a \rightarrow b$,
	$a \rightarrow c$	$a \rightarrow c_1, a \rightarrow c_2$
$\mathcal{X}^+$	$\emptyset$	$c_1 \nleftrightarrow c_2$
$\mathcal{M}^+$	$\emptyset$	$\emptyset$
$\mathcal{Q}^+$	$b \rhd 2 \times r_1$	$b \rhd 2 \times r_1$
	$c \rhd 2 \times r_1$	$c_1 \rhd 2 \times r_1, c_2 \rhd 2 \times r_1^+$

Instead of directly changing the c_k of $k \in \mathcal{R}$, an additional resource type representing the available standby units is inserted into the model. An alternative mode requiring resources of this type is defined for all activities which might trigger a (temporary) expansion of capacities. This approach is appropriate for the description of many realistic scenarios in which resource capacities are changed for the execution of specific activities. Furthermore, it is reasonable to consider optional resource entities on an activity level as common practice to keep the periods of reserve usage as small as possible. Let us consider an example process consisting of three activities a, b and c and in which a has to precede b and c. Furthermore, assume that activities b and c both require all the available units of resource type r_1. In this original scenario, b and c are never executed simultaneously even though it would be possible taking only the precedence constraints into account. The possibility of increasing r_1 is modeled by introducing an alternative (i.e. a standby) resource type r_1^+ and distinguishing between different variants of activities that require r_1. One variant (representing the original activity) is executed using r_1 only, while other variants might be (partly or fully) executed using r_1^+. Table 3 shows how the original model can be extended to reflect the possibility of a capacity change which allows b and c to be executed in parallel.

Activity Insertion/Removal. Next, we examine how the insertion and removal of activities as another important form of process variation can be modeled using the *x-RCPSP*.

Table 4. Activity insertion and removal in the *x-RCPSP*

	Original Process	Extended Process
$\mathcal{A}^+$	a, b, c	a, b_1, b_2, c, e
$\mathcal{P}^+$	$a \rightarrow b,$	$a \rightarrow b_1, a \rightarrow b_2,$
	$b \rightarrow c$	$b_1 \rightarrow c, b_2 \rightarrow e, e \rightarrow c$
$\mathcal{X}^+$	$\emptyset$	$b_1 \nleftrightarrow b_2$
$\mathcal{M}^+$	$\emptyset$	$b_2 \oplus e, b_2 \ominus e$

Consider a simple process consisting of three sequential activities a, b and c in which an additional activity e could be inserted (or removed) between b and c. In order to consider such optional process steps, appropriate *activity dependencies* have to be modeled within the *x-RCPSP*. In addition, one activity of the original model is replaced by two alternative versions, one including the optional activity, the other not.

In the example process consisting of a, b and c, b is replaced by two variants b_1 and b_2. Activity e is omitted whenever b_1 is executed and activated whenever the alternative activity b_2 is activated. Both variants of b take the place of the original activity and depend on activity a. With respect to their successors, b_1 has to be executed prior to c whereas b_2 is modeled as predecessor of e which itself precedes c. The insertion or removal of the optional activity corresponds to a switch from one alternative to another. The mutual exchangeability of b_1 and b_2 is defined in $\mathcal{X}^+$. In the set of dependencies $\mathcal{M}^+$, activity e is linked to b_1 and b_2 appropriately. Table 4 summarizes the differences between the original process and the extended model which supports the dynamic insertion and removal of process steps.

Order Change. Alternative activities can also be used to model flexibility in the execution order of process steps.

Let us consider a process consisting of a strict sequence of four activities a, b, c and d, which should be modified such that the positions of b and d can be swapped. In the *x-RCPSP*, the possibility of reordering the activities (and thus changing the precedence constraints associated with some activities) can be described by introducing alternate versions of all process steps for each potential position of an activity. In the extended process, a distinction is made between the two different versions of b and d. The first one describes the original,where b_1 and d_1 precede and succeed the same activities as the original activities b and d. The second variant describes the optional position where b_2 and d_2 are associate

Table 5. Order change in the *x-RCPSP*

	Original Process	Extended Process
$\mathcal{A}^+$	a, b, c, d	a, b_1, b_2, c, d_1, d_2
$\mathcal{P}^+$	$a \rightarrow b,$	$a \rightarrow b_1, a \rightarrow d_2,$
	$b \rightarrow c,$	$b_1 \rightarrow c, d_2 \rightarrow c,$
	$c \rightarrow d$	$c \rightarrow d_1, c \rightarrow b_2$
$\mathcal{X}^+$	$\emptyset$	$b_1 \leftrightsquigarrow b_2, d_1 \leftrightsquigarrow d_2$
$\mathcal{M}^+$	$\emptyset$	$b_1 \oplus d_1, b_1 \sqsubset d_2,$
		$b_2 \oplus d_2, b_2 \sqsubset d_1,$
		$d_1 \oplus b_1, d_1 \sqsubset b_2,$
		$d_2 \oplus b_2, d_2 \sqsubset b_1$

with the precedence relations of the respective counterpart. Changing the order of activities corresponds to switching from one alternative to another as defined in $\mathcal{X}^+$. Since the same variants of all involved activities must be activated in the respective process variation, mutual dependencies between the alternative versions of b and d are defined in $\mathcal{M}^+$. Table 5 summarizes the modifications to the original process to support the option of reordering activities.

Serialization/Parallelization. The serialization of activities marked for parallel execution and vice versa represents another common form of process variation. Such variations can be formally modeled using the *x-RCPSP* as follows.

Consider six activities, a, b, c, d, e and f, scheduled for sequential execution. Let us examine the possibility of parallelizing the sequence b, c, d with the execution of e. As an initial step, we distinguish between two versions of the *first activity* of the latter sequence (with respect to the serial execution). The first variant corresponds to the serial variant, whereas the second is scheduled for parallel execution. In this case activity e is replaced by two exchangeable alternatives: e_1 replaces the original e and e_2 is used to describe the option of parallelization. With respect to precedence relations, e_2 is (1) a successor of all predecessors of the first activity of the former sequence b, c, d and (2) a predecessor of all successors of the original activity e. In addition, the last activity of the preceding sequence must be linked directly with the successors of the succeeding sequence. Serialization or parallelization may be achieved by allowing to switch between the alternative activities in $\mathcal{X}^+$. Table 6 summarizes the differences between the models describing strictly serial and optional parallel execution modes.

Table 6. Parallelization/serialization in the *x-RCPSP*.

	Original	Modified Network
$\mathcal{A}^+$	a, b, c, d, e, f	a, b, c, d, e_1, e_2, f
$\mathcal{P}^+$	$a \to b,$	$a \to b, a \to e_2,$
	$b \to c,$	$b \to c,$
	$c \to d,$	$c \to d,$
	$d \to e,$	$d \to e_1, d \to f$
	$e \to f$	$e_1 \to f, e_2 \to f$
$\mathcal{X}^+$	$\emptyset$	$e_1 \nleftrightarrow e_2$
$\mathcal{M}^+$	$\emptyset$	$\emptyset$

4.5 Modeling the Turnaround Process

We now explore how different execution variants of the simplified turnaround process (as introduced in Section 3) can be modeled using the previously presented patterns.

In the flowchart of the turnaround process shown in Figure 5, each alternative activity is associated with an *xor*-node that expresses the possibility to chose between alternative execution paths. The angular nodes of the graph correspond to activities, the round nodes represent *xor*-nodes and the directed arcs between these nodes visualize the precedence constraints. Solid lines outline the activities of the "default" process execution path, whereas dashed lines depict potential process alternatives that can, for instance, be exploited to handle disruptions.

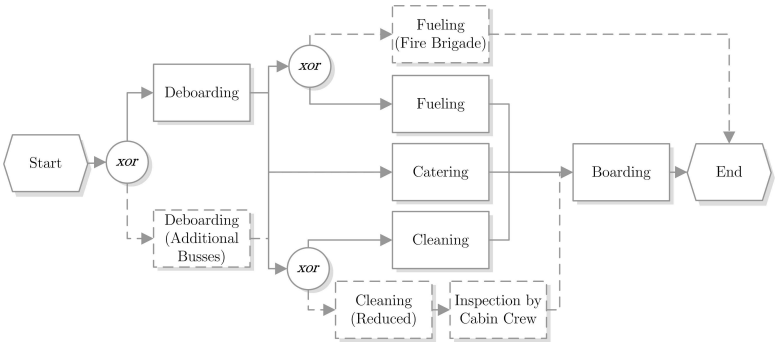

Fig. 5. Turnaround process with modification possibilities

The three concrete forms of intervention which were sketched for aircraft turnaround in Section 3 are related to the abstract modeling patterns presented

in Subsection 4.4 as follows. The potential acceleration of boarding through the assignment of additional resources corresponds to a *mode alternation*. The option of shortening cleaning at the cost of introducing an additional cabin inspection step can be modeled as a combination of *mode alternation and activity insertion*. Finally, the synchronous execution of fueling and boarding corresponds to a applying the *activity parallelization* pattern.

The complete *x-RCPSP* model that captures this variable process is shown in Table 7. Note that in a practical setting it is advisable to use appropriate graphical modeling tools which offer the automatic derivation of formal *x-RCPSP* model definitions. The definition of such a (possibly domain-specific) graphical modeling language and the corresponding set of precise transformation rules is however beyond the scope of this paper.

Table 7. Formal description of aircraft turnaround with execution alternatives

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

In Table 7, process steps are represented by the first three letters of the associated activity names. Deb^B is the alternative of Deb which is characterized by reduced execution time and additional resource requirements. Cle^R is the reduced version of Cle which takes less time but is dependent on the execution on the optional process step of cabin inspection Ins. Fue^P is an alternate version of Fue which is not necessarily executed prior to boarding but has additional resource requirements.

5 Applying the x-RCPSP in Disruption Management

This section illustrates how the *x-RCPSP* modeling framework can be applied to practical disruption management. After sketching the overall approach, we propose an evolutionary algorithm for the incremental optimization of solutions, present the results of a detailed performance evaluation and discuss possible algorithmic enhancements.

5.1 Overview

In line with the general definitions of subsection 2.1, a disruption management problem can be formulated based on the *x-RCPSP* as follows:

- Disruption management problems are solved in the context of an initial schedule S_0 where the current point in time, t_0, typically corresponds to the point at which the disruption is detected. The initial schedule reflects the choices that have been made and thus describes planned process execution paths and activity starting times. The current point in time divides the schedule into a past (unmodifiable) and a future (modifiable) part, i.e., the current *plan for the future*.
- The *disruption* D_0 is characterized by its type and a set of corresponding parameters. The most common forms of disruptions affect activity and resource attributes and lead to changes in durations, starting times, availabilities and so forth. Comprehensive overviews and classifications of different types of disruptions can be found in [14], [23] or [38].
- The *potential forms of intervention* are implicitly defined within the given *x-RCPSP* model of the disruption management problem. This model forms the basis of the initial schedule S_0 and specifies the time, resource and precedence constraints as well as options for the potential restructure of the future part of the schedule.
- The *objective for the adaptation of schedules* is defined through an evaluation function $\varphi : S \rightarrow \mathbb{R}$ describing the costs associated with the execution of a schedule S. Cost can combine various aspects such as the costs of violated constraints, earliness or lateness penalties, the costs of applying interventions as well as the costs associated with deviations from the original schedule or other measures of "schedule quality". These costs should naturally be minimized during optimization.

The central aim of disruption management is to identify the optimal response to a given disruption. In the context of this work, we propose the following strategy:

1. Update S_0 to S_1 to reflect the effects of a disruption D_0. The past is left unmodified, however current and future parts of the schedule are updated to take the effects associated with the occurrence of D_0 into account. S_1 therefore describes what will happen if no form of intervention occurs. The simplest method of generating a "disrupted version" of the schedule is to right-shift all affected activities. However, any other method that reflects the operational strategy and leads to a feasible (but not necessarily good) solution quickly can also be applied.
2. Optimize the future part of S_1 according to φ until a predefined stopping criterion is fulfilled. In operational DM, a common approach is to set a time limit for the identification of a good solution [39]. Note that optimization might be necessary even if S_1 is equal to S_0 (i.e. the disruption has no direct impact on the feasibility of the schedule) as the disruption may have altered the problem such that S_1 is no longer optimal.
3. Derive the set of interventions to from the optimized schedule $S_1^\star$, which describes the best known plan for future process execution. The interventions are given by the difference between $S_1^\star$ and the original schedule S_0.

In the following, we will present a schedule optimization method for Extended Resource-Constrained Project Scheduling Problems which aims to identify an optimal combination of activation state and activity starting times. In general, the practical relevance of the proximity of the optimized to the original schedule suggests the use of an incremental local search algorithm. However, as performance usually represents a crucial factor in the operative process of disruption management and as genetic algorithms perform particularly well for the RCPSP [33], we propose extending RCPSP-specific evolutionary algorithms to take the particularities of the *x-RCPSP* into account.

5.2 An Evolutionary Algorithm

In an evolutionary algorithm, optimization is based on the continuous evolution of a population of solutions. Each generation is composed of the fittest individuals of the previous population and their children, which are the result of

recombination and mutation. The main ideas behind this concept are that (1) a combination of good solutions might result in or be at least close to even better ones and that (2) slight variations in the generated solutions help to avoid local minima. The evolution process, i.e., the actual optimization, is continued until a predefined stopping criterion is reached. The central questions that have to be addressed when designing an evolutionary algorithm are (cf. [40]):

- *Representation.* How are solutions represented in the algorithm?
- *Initialization.* How is the initial population generated?
- *Fitness and Selection.* How are the fittest individuals selected?
- *Crossover.* How can solutions be combined?
- *Mutation.* How can a solution be modified slightly?

The following discusses these aspects with respect to a *x-RCPSP*-specific algorithm.

Representation. Due to the inherent complexity associated with the direct modification of time values, abstract forms of solution representation are commonly used during schedule optimization [33, 41]. Such representations are composed of easily describable and modifiable elements which can be unambiguously converted into corresponding schedules. There are several representations that have been developed in the context of the classical RCPSP [42]. In our approach, we decided to use *activity lists* as they best fit the requirements of the *x-RCPSP*, where not only the order of execution but also the set of selected activities characterizes a solution. Such an approach implicitly addresses both aspects and thus this representation can be adopted for the *x-RCPSP* without modification.

An activity list λ corresponds to a precedence-feasible list of all $i \in \mathcal{A}$ and describes the order in which *active* activities should be considered for the generation of a schedule. An activity list can be converted into a corresponding timetable based on sequential activity insertion (cf. [41, 42], for example). Each operation is scheduled to be executed in the order prescribed by λ at the earliest possible time with respect to the precedence constraints and resource requirements. As *inactive* operations are not contained in λ, they are not part of a solution. Note that even though different lists might result in the same schedule, the unambiguity requirement mentioned above is fulfilled by the Serial Schedule Generation Scheme (SGS) as each λ has exactly one associated schedule.

Initialization. Two scenarios exist for the generation of the initial population of solutions:

- In disruption management, an initial valid schedule typically exists which serves as a starting point for optimization. Considering the descriptions above, the updated schedule S_1 is considered for the generation of the initial population. The corresponding activity list λ_1 can then be determined by sorting all future activities by their starting times. In DM scenarios this initial list corresponds to the option of taking no intervention at all.
- Although the focus of this paper is on disruption management, we also briefly describe a method for generating an initial solution in scenarios where no previous schedule exists. For the creation of a valid activity list λ_1, appropriate choices have to be made about the activation state and the sequence of activities . With respect to the former, $\mathcal{A}_0$ provides a starting point. All activities $i \in \mathcal{A}_0$ are activated and all remaining elements are deactivated. To identify a valid sequence of activities, Algorithm 1 can be used to generate λ from a given set of activities $\mathcal{A}$ and a set of precedence constraints $\mathcal{P}$ (making it possible to generate λ_1 from $\mathcal{A}_0$ and the associated set $\mathcal{P}^+$). In the iteration described in lines 2-6, all $i \in \mathcal{A}$ are sequentially appended to the (initially empty) list of activities. For this purpose, the subset of currently schedulable activities $\mathcal{A}^s$ is determined (line 3). $\mathcal{P}(i) = \{j \in \mathcal{A} | p_{j,i} \in \mathcal{P}\}$ corresponds to the set of all predecessors of an activity $i \in \mathcal{A}$ and we thus combine all activities that (1) have not been scheduled before and (2) do not have any scheduled predecessors or only have scheduled predecessors. If $\mathcal{A}^s$ is empty before all elements of $\mathcal{A}$ have been added (line 6), an inconsistency is detected and no valid solution can be returned (line 4). Otherwise, an

Algorithm 1 Generate Activity List $(\mathcal{A}, \mathcal{P})$

1: $\lambda \leftarrow \emptyset$
2: **repeat**
3: $\quad \mathcal{A}^s \leftarrow \{i \in \mathcal{A} | i \notin \lambda \wedge (\mathcal{P}(i) = \emptyset \vee \mathcal{P}(i) \subseteq \lambda)\}$
4: $\quad$ **if** $\mathcal{A}^s = \emptyset$ **then** return *false*
5: $\quad$ **else** add an arbitrary element of $\mathcal{A}^s$ at the end of λ
6: **until** $|\lambda| = |\mathcal{A}|$
7: return λ

arbitrary element from the set of schedulable activities is appended to the generated list (line 5).

The initial population comprises of the corresponding initial solution and a certain number of associated fellows. These fellows are deduced from λ_1 through the (simple or repeated) application of the mutation operator discussed below. For our example of the turnaround process, a potential initial solution is (Deb, Fue, Cle, Cat, Boa) from which other solutions such as (Deb, Cle, Fue, Cat, Boa) or (Deb, Cle, Cat, $Boa\ Fue^P$) can be generated.

Fitness and Selection. In each iteration of a genetic algorithm the best solutions of the current population are chosen and used to form the basis of the next generation. Thus, it is necessary to assess the quality of the activity lists and compare potential candidates. The first step of this process is to convert the activity list into its corresponding schedule using the schedule generation scheme associated with the chosen representation (see also the discussion above). The costs of the resulting timetable are then calculated with the help of the schedule evaluation function φ (see Subsection 5.1). As this function should take all decision-relevant aspects into consideration, the values obtained can be used directly for the quantitative comparison of solutions. In particular, the activity lists that result in the schedules with the lowest costs should be taken into consideration for the evolution of the current generation.

Crossover. The main difficulty in performing a crossover operation in the context of the *x-RCPSP* results from the fact that the two activity lists to be combined might describe different activation states and thus contain different sets of operations. RCPSP-specific procedures can only be applied in the special case in which the contents of both lists are the same. In this subsection we therefore propose a technique for the combination of two parent solutions λ_a and λ_b which is based on the idea that one parent prescribes the activation state and the other prescribes the order of the process steps. The first parent thus defines which process variations are chosen in the child solution λ whereas the second one influences the position of activities in the resulting list.

Algorithm 2 describes a suitable crossover operator in more detail. The set of activities contained in λ_i is denoted as $\mathcal{A}_i$. The subset $\mathcal{X}_j \subseteq \mathcal{X}^+$ describes the substitutions that have led from the original activation state $\mathcal{A}_0$ to $\mathcal{A}_j$.

Algorithm 2 Crossover (λ_a, λ_b)

```
1: if A_a = A_b then
2:     generate λ through the application of an 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) ∪ (X_b \ X_a)| then
6:         return incompatible
7:     else
8:         λ ← λ_b
9:         do
10:            changed ← false
11:            if i ∈ λ, x_{i,j} ∈ T then
12:                replace i with j in λ, taking M^+ into consideration
13:                changed ← true
14:            end if
15:        while changed
16:    end if
17: end if
18: return λ
```

Whenever the contents of the parent activity lists λ_a and λ_b are equal, one of the established RCPSP-specific crossover operators (see [41, 43], for example) is applied (lines 1-2). If, however, $\mathcal{A}_a \neq \mathcal{A}_b$, a *transition set* $\mathcal{T} \subseteq \mathcal{X}^+$ is used to cope with the different list contents. The transition set describes which substitutions should be applied to convert λ_b to λ_a. Given the substitution sets $\mathcal{X}_a$ and $\mathcal{X}_b$ associated with the activity lists, $\mathcal{T}$ combines (1) all substitutions exclusive to λ_a and (2) the inversions of all substitutions exclusive to λ_b (line 4). If one of the substitutions exclusive to λ_b is not invertible, the transition set is incomplete, which means that λ_b can not be directly converted into λ_a. If the size of the symmetric difference between $\mathcal{X}_a$ and $\mathcal{X}_b$ is larger than the number of elements in $\mathcal{T}$ (line 5), the activity lists are thus considered *incompatible* for crossover and no valid combination of the parent activity lists can be performed. The algorithm returns (without result) and a different selection of parents has to be made (line 6). Otherwise, a valid child is generated as follows. First, the new solution λ is initialized as a clone of λ_b (line 8). Then, in a repetitive procedure (lines 9-15), activity substitutions are applied until λ contains no more replaceable

elements. Note that all precedence and activity dependency constraints have to be considered when modifying the contents of λ (line 12, see also the discussion on where and how to place newly inserted activities in the following subsection). The result of this replacement procedure is a new, valid activity list λ containing the elements of λ_a and following the order prescribed by λ_b.

Consider the two parent solutions $\lambda_1 = (Deb^B, Fue, Cle, Cat, Boa)$ and $\lambda_2 = (Deb^B, Cat, Cle^R, Ins, Boa, Fue^P)$ of our turnaround process. If $\lambda_a \leftarrow \lambda_1$ and $\lambda_b \leftarrow \lambda_2$, the transition set $\mathcal{T} = \{Cle^R \rightsquigarrow Cle, Fue^P \rightsquigarrow Fue\}$ results from the combination of $\mathcal{X}_a = \{Deb \rightsquigarrow Deb^B\}$ and $\mathcal{X}_b = \{Deb \rightsquigarrow Deb^B, Cle \rightsquigarrow Cle^R, Fue \rightsquigarrow Fue^P\}$. Based on these elements, the list $\lambda = (Deb^B, Cat, Cle, Fue, Boa)$ is generated. If, alternatively, $\lambda_a \leftarrow \lambda_2$ and $\lambda_b \leftarrow \lambda_1$, the sequence $\lambda = (Deb^B, Fue^P, Cle^R, Cat, Boa)$ can be generated.

Mutation. As mentioned above, mutation is used in evolutionary algorithms to avoid convergence on poor local minima. The basic idea is to randomly and slightly modify some part of the survivors and the newly generated children. For the *x-RCPSP*, mutating (all or only several randomly chosen) activity lists corresponds to making some slight schedule modifications. In the *x-RCPSP*-specific mutation operator described below (Algorithm 3), each of the interventions modeled in $\mathcal{X}^+$ is applied with a certain probability.

Algorithm 3 Mutate (λ)

1: **if** a randomly generated value $\in [0, 1] \leq \theta$ **then**
2: rearrange λ through the application of an RCPSP-specific mutation operator
3: **else**
4: select an arbitrary $x_{i,j} \in \mathcal{X}^+ | \ i \in \lambda$
5: replace i with j in λ, taking all dependencies defined in $\mathcal{M}^+$ into consideration
6: **end if**
7: return λ

Algorithm 3 performs a simple rearrangement of the activities with a certain probability $\theta \in [0, 1]$ (line 2). For the purpose of merely changing the order of activities in λ (and therefore leaving the activation state unmodified), any existing RCPSP-specific operator (see [43, 41], for example) can be applied. Accordingly, the chosen process execution path is only modified with a probability of $1 - \theta$.

In that case, an arbitrary substitution is selected (line 4) and then applied on λ, taking all related dependencies and constraints into consideration (line 5). In the context of such process modifications, the question of where and how to place newly inserted activities into λ may arise. Precedence constraints can make certain positions unavailable and the rearrangement of other activities necessary. One strategy might be (1) to put replaced activities as close as possible to the former position of the replaced activity, (2) to insert the operations associated with the substitution through $\mathcal{M}^+$ at the earliest possible position and (3) to ensure precedence feasibility by shifting affected activities to the right-hand side of the respective predecessors. However, the fact that other strategies could potentially be applied (and perhaps even randomly varied) provides mutation with additional flexibility.

Let us consider an activity list $\lambda = (Deb^B, Cat, Cle, Fue, Boa)$ in our turnaround process. Temporal shifts and/or resource reallocations can for instance be associated with the rearrangement of the operations into a sequence $(Deb^B, Cle, Cat, Fue, Boa)$ or $(Deb^B, Fue, Cle, Cat, Boa)$. A process variation is performed if λ is mutated into $(Deb^B, Cat, Cle, Fue^P, Boa)$ through the application of $Fue \rightsquigarrow Fue^P$ or into $(Deb^B, Cat, Cle^R, Ins, Fue, Boa)$ through the application of $Cle \rightsquigarrow Cle^R$.

5.3 Computational Experiments

In the following, we discuss the results of an experimental evaluation that shows that the proposed genetic algorithm quickly converges on good-quality solutions. We describe the experimental setup and test cases before summarizing the obtained results. Note that the intention of this evaluation was to show the general applicability of the presented procedures rather than to illustrate the performance of the specific operators used in the experiments. Thus, the proposed algorithm and the corresponding evaluation results should be viewed as a starting point for further improvements and extensions.

Experimental Setup and Problem Instances. The proposed algorithm and the operators were evaluated in a Java-based framework for the generation and solution of disruption management problems. Furthermore, the RCPSP-specific operators were implemented using the highly efficient [33, 42] algorithm proposed

by Hartmann [43]. The following parameter settings and probabilities were used in the described experiments:

- Each generation consists of 10 solutions.
- The best solution of a population is added to the succeeding generation without modification (i.e. the rate of survival is set to 0.1).
- For the creation of the remaining solutions, two parents are combined using the crossover operator described in Algorithm 2. In accordance with the given survival rate, the probability for crossover is thus $P = 0.9$. Parent solutions are selected with a probability that is directly proportional to their fitness value (i.e. fitness proportionate selection).
- For the newly generated child solutions, mutation is *attempted* with a probability of $P = 0.5$. That is, an attempt is made to either change the position of two randomly chosen activities or to perform a replacement according to the above algorithm. Note that such an attempt to mutate the activity list does not necessarily result in a modified schedule as on the one hand precedence constraints often prohibit permutations of the list while on the other hand a mutated list might represent the same schedule as the original λ.
- With a continuously decreasing probability $P = \frac{ln(2)}{ln(i)}$ the original schedule is reinserted into the ith generation. This ensures that the focus remains on solutions close to the original schedule.

All of the above values were chosen and adjusted based on several test runs that were conducted on various artificially generated DM problem instances. Note that the parameters were not optimized for domain-specific problems and were not dynamically adjusted during optimization, leaving room for additional performance improvements and further research which is, however, beyond the scope of this paper.

As no publicly available data sets or generators for instances of reactive scheduling and disruption management problems [44] existed at the time, we decided to implement a highly configurable framework for the automatic generation of appropriate problem instances. The framework not only supports the parametrization of network complexity, resource factor and resource strength (see [45]) but also allows us to control the structure of the original (the so-called baseline) schedule and the associated disruptions and the potential forms of interventions.

Throughout the experiments, the following parameter settings were used for generating the test cases and subsequently lead to 16 different problem classes.

- *Low/High Process Complexity.* This parameter determines the number of precedence constraints among activities. Low process complexity indicates that there are few, whereas high process complexity means that there are many precedence relations defined.
- *Low/High Resource Complexity.* This setting combines both resource requirements and resource availability. Low resource complexity means that there are many resource units available to cover low requirements, whereas high resource complexity means that there are only a few resource entities available to satisfy high requirements.
- *With/Without Left-shifts.* A left-shift corresponds to scheduling an activity to start earlier than it did in the baseline schedule. This setting controls whether such starting time modifications are permitted.
- *Tight/Wide Baseline Schedule.* This setting controls the distribution of activity starting times and the amount of slack time incorporated into the schedule to compensate for disruptions. In a tight schedule, activities start at the earliest possible point in time and tend to be executed in parallel. In a wide schedule, activities start only after a certain amount of slack time has passed and are therefore less likely to be executed simultaneously.

The evaluated problem instances were split into two groups depending on their size. Small problems consist of one process with 10 activities and large problems consist of one process containing 100 activities.

Furthermore, 10 different problem instances were generated for each size and each complexity class, forming a total of 320 test cases. Each of the baseline schedules was disrupted by doubling the duration of fifty percent of the activities immediately after the start of execution. Various forms of intervention were available in the test cases to respond to this disruption. In addition to temporal shifts and resource reallocations, process execution path modifications were also available. The test case generator assigned one of the execution alternatives described in Subsection 4.4 to each activity with a probability of $P = 0.1$

In order to evaluate a schedule, various quantitative and qualitative characteristics can in principle be taken into account (see Subsection 5.1). In real world situations it might be necessary to consider multiple and potentially conflicting

objectives, typically resulting in the combined schedule evaluation function φ being complex and highly dynamic (see [46] and [47] for the discussion of related approaches).

In our experiments, however, we decided to use a relatively simple objective function for determining the fitness of a schedule as the main goal of the evaluation was to assess how quick the algorithm converges on suitable solutions. The rationale of the cost function used in the experiments is as follows. Every $i \in \mathcal{A}^+$ has a due date δ_i. Costs of 1 monetary unit are assumed for each time unit of delay per activity. In order to simulate a typical cost situation, it is assumed that a schedule modification (such as the temporal shift of an activity or a process variation) is comparably expensive and induces costs of 3 monetary units. The schedules were thus optimized according to the following function, in which Δ denotes the number of modifications and $\mathcal{A}$ corresponds to the set of active elements in S.

$$\varphi(S) = 3 * \Delta + \sum_{i \in \mathcal{A}} \max\left(0, \beta_i + d_i - \delta_i\right)$$

Results. To the best of our knowledge, no public benchmark data exists for the particular disruption management problems addressed in this work. A comparison of our algorithm with performance of existing methods is thus not possible. The only theoretical option would be to compare the presented *x-RCPSP*-specific algorithm with classical schedule optimization techniques that, however, only take temporal shifts and resource reallocations into account. It would therefore be inappropriate to draw conclusions from a comparison between an algorithm which is capable of considering alternative and a method that cannot.

Thus, instead of comparing our GA to an algorithm that actually solves a different class of problems, we illustrate that the proposed methods facilitate the identification of good-quality solutions to relatively large problem instances in very short time. This fast convergence on optimal or at least good schedules is thus the main criterion for the applicability of the GA in realistic scenarios of operational disruption management. Furthermore, publishing the benchmark problems and the results of this evaluation provides a basis for future comparisons and thereby stimulates further research in this area.

Table 8 illustrates the optimization potential that could be obtained within a limited period of time[2]. For *small* problem instances, the exact optimum $S_1^{\bigstar}$ was

Table 8. Portion of the optimization potential tapped within limited time

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

initially identified using a simple deterministic procedure. All potential schedules that were generated using the possible activity list permutations and substitutions were then evaluated using the given cost function. The *optimization potential* is defined as the difference between the costs associated with the disrupted and unmodified schedule S_1 and $\varphi(S_1^{\bigstar})$. The genetic algorithm was evaluated over 10 separate test runs for each of the generated test cases. The execution time was limited to 5 or 15 seconds respectively on a standard desktop PC[3]. Note that these tight time limits should correspond to the near-real-time requirements of practical disruption management. In each test run, the optimization procedure obtained a certain amount of the theoretically available potential. If $S_1^{\star}$ denotes the schedule resulting from 5 or 15 seconds of optimization respectively, the obtained portion corresponds to $\frac{\varphi(S_1)-\varphi(S_1^{\star})}{\varphi(S_1)-\varphi(S_1^{\bigstar})}$. The figures shown in Table 8 represent average values over all test runs.

Even when using recent *exact methods* (see [48], for example) it is not possible to solve hard scheduling problems of the regarded sizes in reasonable time [43, 49]. Therefore, we followed a different strategy for the evaluation of *large* problem instances. Instead of determining the exact optimum, the best solution that could be identified during (1) all conducted GA runs as well as (2) an additional 10-minute test run was taken as the reference solution $S_1^{\bigstar}$. The values in Table 8 correspond to the average *known* optimization potential that could be obtained

within a limited period of time. Note also that an additional test lasting 45 seconds was conducted in order to illustrate the development of optimization over time.

The following observations can be made based on an analysis of the figures listed in Table 8 and the detailed performance evaluation published online.

- The assumption that the proposed genetic algorithm converges on optimal or at least good-quality solutions is plausible. In almost all small cases the exact optimum could be identified within the first 5 seconds of optimization. For large instances, the fact that approximately 75% of the known potential (identified using more than 20 minutes of optimization) could be obtained within 15 seconds is promising.
- There is a high probability of obtaining a good-quality solution within a limited amount of time. This is illustrated in Figure 6, which summarizes the distribution of the obtained optimization potential for all 15-second test runs on large problem instances.

  The x-axis defines 5% ranges for the obtained optimization potential. The y-axis shows the percentage of the identified solutions that fell into a particular optimization range. The rightmost point in the figure for instance states that about 33% of all identified solutions could obtain between 95 and 100% of the known optimization potential. The thick line links the actual observed values, whereas the thin dashed line provides a corresponding exponential approximation.

  Figure 6 highlights that almost half of the test runs resulted in schedules which could obtain at least 75% of the known optimization potential. The probability of identifying a solution that obtains more than 50% of the potential is higher than 80%.
- Most of the parameters discussed for varying the test cases have an actual impact on the complexity of the problem and the performance of the algorithm.
  - A high number of precedence constraints narrows the search space. The more restrictions that exist, the less sequences that are valid and have to be evaluated. Thus the genetic algorithm converges faster given a high process complexity.
  - Resource complexity has a considerable impact on the time required for the conversion of an activity list into the corresponding schedule. The

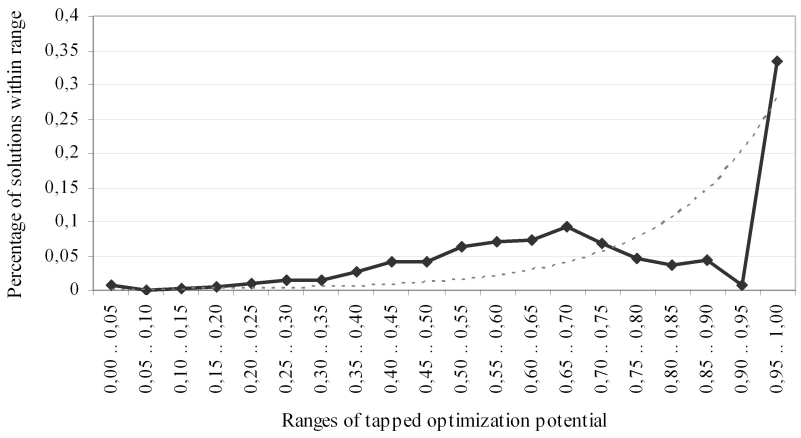

Fig. 6. Distribution of the tapped optimization potential for large problem instances

higher the requirements and the less units that are available, the more effort that is required to identify the first possible execution slot for an activity. Low resource complexity thus means that more schedules can be evaluated and the genetic algorithm will converge faster.

- Restricting temporal shifts to activity postponements means that the space of available options is narrowed. The genetic algorithm therefore converges faster if no left-shifts of activity starting times are allowed.
- The existence of slack time has no considerable impact on the speed of convergence in the analyzed scenarios.

– Large improvements can be made within the first few seconds of optimization. This is illustrated in Figure 7, where the typical development of the costs associated with the best known schedule is depicted throughout the evolution of 2000 generations of 20 solutions each. Note that it is this fast convergence that makes the proposed algorithm applicable in realistic scenarios of operational disruption management where decisions typically have to be made in near-real-time.

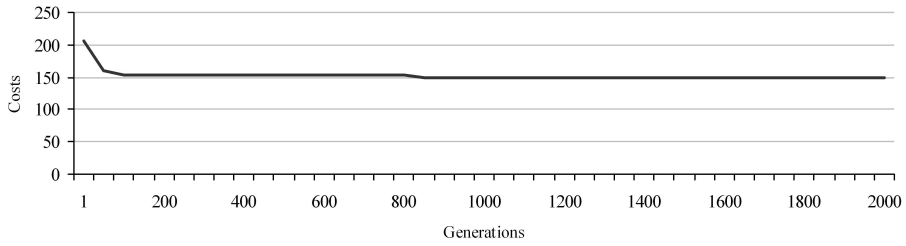

Fig. 7. Reduction of schedule costs during optimization

6 Summary and Conclusion

This paper proposes a comprehensive approach to modeling and solving disruption management problems in a practical context. Based on examples from the domain of aircraft turnaround management it was shown that standard rescheduling techniques (such as the temporal shift of activities or the reallocation of resources) does not suffice in many real-world disruption management problems and that it is also necessary to consider alternative process execution paths in order to optimally recover from a disrupted schedule.

Motivated by the lack of possibilities to model these forms of intervention in existing scheduling frameworks, the well-established Resource-Constrained Project Scheduling Problem (RCPSP) was extended to meet the demands of such advanced forms of disruption management. We subsequently introduced the concept of alternative activities and corresponding activation states into the *x-RCPSP*, which facilitate the considerations of such variations in the process execution path and to therefore combine planning and scheduling during optimization. Beside this basic extension we also provided several modeling patterns which can be used as a guideline when formulating disruption management problems.

An evolutionary algorithm was proposed for incremental schedule optimization based on the fact that in many practical situations disruption management problems have to be dealt with in near-real-time and that Genetic Algorithms perform well in such situations . Subsequently a Genetic Algorithm was developed for the *x-RCPSP* and appropriate initialization, crossover, and mutation operators were developed.

The new method was analyzed in an experimental evaluation using example problems have of various size, resource complexity, tightness and so forth. Importantly, it became obvious that even for relatively large problems the genetic algorithm converges on optimal or good-quality solutions within only a few seconds. This behavior of early convergence indicates that the presented approach is suitable for dealing with practical disruption management problems. We therefore propose that the presented algorithm forms an ideal starting point for further extensions and improvements. One potential approach is to make it applicable to even larger disruption management problems using the recently developed Local Rescheduling technique described in [50].

Finally, a comprehensive set of publicly available test cases and benchmark results for disruption management problems were provided which we hope will stimulate further research in the area.

Acknowledgement

This research presented in this paper was partly funded by grants from (1) FFF Austria, Project *cdm@airports*, and (2) the European Union, Project *WS-Diamond*, Contract 516933.
The authors would like to thank the reviewers for their invaluable comments on the paper and their detailed and constructive suggestions for improvement. We would also like to thank Arthur Pitman for his stylistic feedback. This paper combines and continues previous research presented in [51–53].

Notes

[1]Affiliation of authors: Institute of Applied Informatics, University of Klagenfurt, Universitätsstraße 65-67, 9020 Klagenfurt, Austria.
Contact author: Dietmar Jannach, dietmar@ifit.uni-klu.ac.at.

[2]All generated problem instances as well as detailed test results can be obtained from http://rcpsp.serverside.at/applied-intelligence-07.html. They may be regarded as a starting point for further evaluations and algorithm improvements.

[3]Intel Pentium M, 1400 MHz, 768 MB RAM, Windows XP SP2

References

1. M. Wambsganss, "Collaborative decision making through dynamic information transfer," *Air Traffic Control Quarterly*, vol. 4, pp. 107–123, 1997.
2. R. Hoffman, M. Ball, A. Odoni, W. Hall, and M. Wambsganss, "Collaborative decision making in air traffic flow management," UC Berkeley, Tech. Rep., 1999.
3. F. R. Carr, "Robust decision support tools for airport surface traffic," Ph.D. dissertation, Massachusetts Institute of Technology, 2004.
4. F. Carr, G. Theis, J.-P. Clarke, and E. Feron, "Evaluation of improved pushback forecasts derived from airline ground operations data," *Journal of Aerospace Computing, Information and Communication*, vol. 2, no. 1, pp. 25–43, 2005.
5. X. Qi, J. F. Bard, and G. Yu, "Disruption management for machine scheduling: The case of spt schedules," *International Journal of Production Economics*, vol. 103, no. 1, pp. 166–184, 2006.
6. M. S. Fox and S. Smith, "Isis: A knowledge based system for factory scheduling," *Expert Systems*, vol. 1, no. 1, 1984.
7. S. F. Smith, N. Muscettola, D. C. Matthys, P. S. Ow, and J.-Y. Potvin, "Opis: An opportunistic factory scheduling system," in *IEA/AIE '90: Proc. 3rd International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*. New York, NY, USA: ACM Press, 1990, pp. 268–274.
8. M. Zweben, E. Davis, B. Daun, and M. J. Deale, "Scheduling and rescheduling with iterative repair," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 6, pp. 1588–1596, 1993.
9. M. Deale, M. Yvanovich, D. Schnitzuius, D. Kautz, M. Carpenter, M. Zweben, G. Davis, and B. Daun, "The space shuttle ground processing scheduling system," in *Intelligent Scheduling*, M. Zweben and M. Fox, Eds. Morgan Kaufmann, 1994, pp. 423–449.
10. S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, "Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems," *Artificial Intelligence*, vol. 58, no. 1-3, pp. 161–205, 1992.
11. S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4589, pp. 671–680, 1983.
12. W. Herroelen and R. Leus, "Project scheduling under uncertainty: Survey and research potentials," *European Journal of Operational Research*, vol. 165, pp. 289–306, 2005.

13. S. Van de Vonder, E. Demeulemeester, and W. Herroelen, "A classification of predictive-reactive project scheduling procedures," *Journal of Scheduling*, vol. 10, no. 3, pp. 195–207, 2007.
14. G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing systems: a framework of strategies, policies, and methods," *Journal of Scheduling*, vol. 6, no. 1, pp. 39–62, 2003.
15. H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, "Executing production schedules in the face of uncertainties: A review and some future directions," *European Journal of Operational Research*, vol. 161, no. 1, pp. 86–110, 2005.
16. J. Clausen, J. Hansen, J. Larsen, and A. Larsen, "Disruption management," *ORMS Today*, vol. 28, pp. 40–43, 2001.
17. J. Larsen, M. Løve, K. R. Sørensen, and J. Clausen, "Disruption management for an airline - rescheduling of aircraft," in *Applications of Evolutionary Computing*, ser. Lecture Notes in Computer Science, S. Cagnoni, J. Gottlied, E. Hart, M. Middendorf, and G. R. Raidl, Eds., vol. 2279. Springer, April 2002, pp. 315–324.
18. M. Løve, K. R. Sørensen, J. Larsen, and J. Clausen, "Using heuristics to solve the dedicated aircraft recovery problem," *Central European Journal of Operations Research*, vol. 13, no. 2, pp. 189–207, 2005.
19. J. Yang, X. Qi, and G. Yu, "Disruption management in production planning," *Naval Research Logistics*, vol. 52, no. 5, pp. 420–442, 2005.
20. J. Bean, J. Birge, J. Mittenthal, and C. Noon, "Matchup scheduling with multiple resources, release dates and disruptions," *Operations Research*, vol. 39, pp. 470–483, 1991.
21. Y. Xia, M.-H. Yang, B. Golany, S. M. Gilbert, and G. Yu, "Real-time disruption management in a two-stage production and inventory system," *IIE Transactions*, vol. 36, no. 2, pp. 111–125, 2004.
22. G. Yu and X. Qi, *Disruption Management: Framework, Models and Applications*. Singapore: World Scientific Publishing, 2004.
23. G. Zhu, J. F. Bard, and G. Yu, "Disruption management for resource-constrained project scheduling," *Journal of the Operational Research Society*, vol. 56, pp. 365–381, 2005.
24. M. Xu, X. Qi, G. Yu, H. Zhang, and C. Gao, "The demand disruption management problem for a supply chain system with nonlinear demand functions," *Journal of Systems Science and Systems Engineering*, vol. 12, no. 1, pp. 82–97, 2003.
25. X. Qi, J. F. Bard, and G. Yu, "Supply chain coordination with demand disruptions," *Omega*, vol. 32, pp. 301–312, 2004.
26. EUROCONTROL Performance Review Commission, "Evaluating the true cost to airlines of one minute of airborne or ground delay," 2004. [Online]. Available: http://www.eurocontrol.int/prc

27. Z. A. Shavell, "The effects of schedule disruptions on the economics of airline operations," in *Air Transportation Systems Engineering*, G. L. Donohue and A. Zellweger, Eds. AIAA Press, 2001.
28. S. Adeleye and C. Chung, "A simulation based approach for contingency planning for aircraft turnaround operation system activities in airline hubs," *Journal of Air Transportation*, vol. 11, no. 2, pp. 140–155, 2006.
29. C.-L. Wu and R. E. Caves, "Modelling and simulation of aircraft turnaround operations at airports," *Transportation Planning & Technology*, vol. 27, no. 1, pp. 25–46, 2004.
30. N. Ashford, H. M. Stanton, and C. A. Moore, *Airport Operations*, 2nd ed. McGraw-Hill, 1997.
31. J. Błazewicz, J. K. Lenstra, and A. Rinnooy Kan, "Scheduling projects to resource constraints: Classification and complexity," *Discrete Applied Mathematics*, vol. 5, pp. 11–24, 1983.
32. 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*, vol. 112, pp. 3–41, 1999.
33. R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: An update," *European Journal of Operational Research*, vol. 174, pp. 23–37, 2006.
34. S. Hartmann, "Project scheduling with multiple modes: A genetic algorithm," *Annals of Operations Research*, vol. 102, no. 1-4, pp. 111–135, 2001.
35. C. Artigues, P. Michelon, and S. Reusser, "Insertion techniques for static and dynamic resource constrained project scheduling," *European Journal of Operational Research*, vol. 149, pp. 249–267, 2003.
36. A. Elkhyari, C. Guéret, and N. Jussien, "Constraint programming for dynamic scheduling problems," in *ISS'04 International Scheduling Symposium*, H. Kise, Ed., Awaji, Hyogo, Japan, 2004, pp. 84–89.
37. J. Beck and M. Fox, "Constraint directed techniques for scheduling with alternative activities," *Artificial Intelligence*, vol. 121, pp. 211–250, 2000.
38. R. Li, Y. Shyu, and A. Sadashiv, "A heuristic rescheduling algorithm for computer-based production scheduling systems," *International Journal of Production Research*, vol. 31, no. 8, pp. 1815–1826, 1993.
39. J. G. Herrero, A. Berlanga, and J. M. Molina, "Methods for operations planning in airport decision support systems," *Applied Intelligence*, vol. 3, no. 22, pp. 183–206, 2005.
40. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing, 1989.

41. K. Hindi, H. Yang, and K. Fleszar, "An evolutionary algorithm for resource-constrained project scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 512–518, 2002.
42. R. Kolisch and S. Hartmann, "Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis," in *Project scheduling: Recent models, algorithms, and applications*, J. Weglarz, Ed. Kluwer Academic Publishers, 1999, pp. 147–178.
43. S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Naval Research Logistics*, vol. 45, pp. 733–750, 1998.
44. N. Policella and R. Rasconi, "Testsets generation for reactive scheduling," in *Workshop on Experimental Analysis and Benchmarks for AI Algorithms*, Ferrara, Italy, 2005.
45. R. Kolisch, A. Sprecher, and A. Drexl, "Characterization and generation of a general class of resource-constrained project scheduling problems," *Management Science*, vol. 41, pp. 1693–1703, 1995.
46. A. Viana and J. P. de Sousa, "Using metaheuristics in multiobjective resource constrained project scheduling," *European Journal of Operational Research*, vol. 120, no. 2, pp. 359–374, 2000.
47. P. Cowling, N. Colledge, K. Dahal, and S. Remde, "The trade off between diversity and quality for multi-objective workforce scheduling," pp. 13–24, 2006.
48. P. Laborie, "Complete MCS-based search: Application to resource constrained project scheduling," in *IJCAI-05 Proceedings of Nineteenth International Joint Conference on Artificial Intelligence*, L. P. Kaelbling and A. Saffiotti, Eds., Edinburgh, Scotland, 2005, pp. 181–186.
49. 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*, vol. 54, no. 6, pp. 614–626, 2003.
50. J. Kuster, D. Jannach, and G. Friedrich, "Local rescheduling - a novel approach for efficient response to schedule disruptions," in *IEEE Symposium on Computational Intelligence in Scheduling*, Honolulu, USA, 2007, pp. 79–86.
51. J. Kuster and D. Jannach, "Handling airport ground processes based on resource-constrained project scheduling," in *Advances in Applied Artificial Intelligence*, ser. LNCS 4031, M. Ali and R. Dapoigny, Eds. Springer, 2006, pp. 166–176.
52. ——, "Extending the resource-constrained project scheduling problem for disruption management," in *Proceedings of the 3rd IEEE Conference on Intelligent Systems*. London, UK: IEEE Press, 2006, pp. 95–102.
53. J. Kuster, D. Jannach, and G. Friedrich, "Handling alternative activities in resource-constrained project scheduling problems," in *IJCAI-07, Proceedings of*

the Twentieth International Joint Conference on Artificial Intelligence, Hyderabad, India, 2007, pp. 1960–1965.