Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)

Local Rescheduling – A Novel Approach for Efficient Response to Schedule Disruptions

Jürgen Kuster, Dietmar Jannach, and Gerhard Friedrich

Abstract— **Whenever an unforeseen disturbance occurs during the execution of scheduled operations, rescheduling might be necessary: Beside *temporal shifts* and the *allocation of alternative resources*, also potential *switches from one process variant to another one* have typically to be considered. In realistic scenarios of operational disruption management (DM) the high number of potential options makes the provision of online decision support complex. It is thus necessary to significantly reduce the size of the regarded (search) problems which can for instance be achieved by applying methods of partial rescheduling. However, existing approaches such as Affected Operations Rescheduling (AOR) or Matchup Scheduling (MUP) focus on production-specific problems and can not be applied to more generic problem classes. To overcome this limitation, we introduce a novel approach to partial rescheduling in this paper: Local Rescheduling (LRS) is based on the incremental extension of a time window which is regarded for potential schedule modifications. We discuss how this time window can be initialized, extended and used for rescheduling. Moreover, we illustrate the superior performance of LRS in comparison with full rescheduling and MUP.**

I. Introduction

As an intrinsic and pervasive aspect of the real world, uncertainty typically leads to deviations from predetermined plans. The process of responding to unforeseen disturbances during the execution of planned and scheduled operations by ways of rescheduling is called disruption management (DM, see [1], [2]): Central aim is to select appropriate repair actions, directed at the minimization of the negative impact typically associated with a disruption. Practically relevant options are especially (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.

It is particularly the latter form of intervention that makes DM more than just an application of scheduling methods: Rather than only identifying valid starting times and resource allocations for a fixed set of activities (in the *scheduling* part of the problem), the structure of the executed operations itself has also to be reconsidered (in the *planning* part of the problem). The space of relevant options is thus even larger than in classical schedule optimization. As the complexity associated with analyzing all potential combinations of repair actions coincides with the typical requirement of responding to a disruption as quickly as possible, the use of exact and deterministic algorithms is usually inappropriate for the provision of online decision support in realistic scenarios of operational DM. However, it is not sufficient to simply implement a metaheuristic procedure either: Recent and extensive performance evaluations (see [3], for example) reveal that even by using the most powerful heuristic procedures only up to about 100-120 activities can be scheduled efficiently in reasonable time. Beside the fact that these studies only apply to the scheduling part of the DM problem, it has to be stated that this number of activities is relatively small in relation to many real-world problems.

In order to cope with realistic problem sizes, other strategies than simply optimizing all future operations (in a procedure equal or similar to the one applied during the generation of the initial schedule) have therefore to be considered. A good idea is to take the existing schedule (the *baseline* schedule) into account and to actually do *re*scheduling. This is basically what has motivated the emergence of partial rescheduling approaches (see [4] for an overview): Instead of generating a *new* schedule for all future activities (such Full Rescheduling, FRS, is usually done through the application of existing scheduling algorithms), only a subset of pending operations is considered for *modification*. This way, an optimal combination of schedule efficiency and schedule stability shall be attained.

Existing approaches to partial rescheduling have mainly focused on production-specific scheduling problems. Affected Operations Rescheduling (AOR, see [5], [6], [7]) can only be applied to job shop problems, as the underlying binary tree allows only for one job and one resource successor per activity. Similarly, Matchup Scheduling (MUP, see [8], [9]) is restricted to problems where activities require no more than one resource. Motivated by the fact that many practical problems can not be mapped to these specific kinds of problems, we herein present a novel, more generic approach to partial rescheduling: Local Rescheduling (LRS) is based on the incremental extension of a time window regarded for the reparation of a disrupted schedule.

The remainder of this document is structured as follows: In Section II the Extended Resource-Constrained Project Scheduling Problem (*x-RCPSP*) is introduced as a conceptual framework for the formal description of disruption management problems. In Section III the LRS method is presented in detail: In particular, an exemplary way of initializing the time window after the occurrence of a disruption as well as potential time window extension schemes are discussed. Section IV evaluates Local Rescheduling in comparison with Full Rescheduling and a generalized version of Matchup

Manuscript received October 31, 2006. The work presented herein was partially funded by grants from (1) FFF Austria, Project *cdm@airports*, and (2) the European Union, Project *WS-Diamond*, Contract 516933.

J. Kuster, D. Jannach and G. Friedrich are with the Institute of Business Informatics and Application Systems at the University of Klagenfurt, Universitätsstraße 65-67, 9020 Klagenfurt, Austria (e-mail: {jkuster, dietmar, gerhard}@ifit.uni-klu.ac.at)

Scheduling (which is basically a specific form of LRS): It is shown that the performance of an evolutionary algorithm can be increased drastically through the consideration of LRS and that even large problems can be tackled in reasonable time by use of this partial rescheduling method.

II. The Extended RCPSP

The Resource-Constrained Project Scheduling Problem (RCPSP, see [10], [11]) provides a framework for the formalization of scheduling problems. We take it as a starting point for the description of DM problems due to several reasons:

- Unlike the production-specific job shop, flow shop and open shop problems, the more generic RCPSP imposes no restrictions on the number of resource entities, the structure of processes or the associated requirements.
- Highly efficient algorithms are available for the generation of schedules based on the RCPSP: A recent survey and evaluation can be found in [3], for example.
- Activities, precedence constraints, resources and respective requirements form the fundamental modeling constructs of an RCPSP: They make it possible to describe entities and relationships on a reasonably high level of abstraction.

Even though the RCPSP can thus be applied for modeling and solving the scheduling part of DM problems, it unfortunately provides no support for the description of alternative process execution paths: It is not possible to consider options such as the dynamic insertion or removal of activities, changes in the order of activity execution or the serialization/parallelization of process steps. The only approach to additional structural flexibility has been made in the Multi-Mode RCPSP (MRCPSP, see [12]), which allows the definition of different *execution modes* for each activity: Thereby it is possible to consider different combinations of *durations* and *resource requirements* in the process of schedule optimization. However, this concept is not sufficient for the description of more complex process variations.

In the context of the RCPSP, previous research on DM (see [13], [14], [15], for example) has mainly focused on the scheduling part of respective problems. To make the consideration of additionally relevant *process variations* possible, we have proposed to extend the classical RCPSP by the concept of alternative activities: The Extended Resource Constrained Project Scheduling Problem (*x-RCPSP*, see [16]) is based on a distinction between *active* and *inactive* activities and the idea of considering only *active* elements in the generation of the final schedule. By activating and deactivating activities according to predefined substitution rules and various forms of constraints, the activation state of an *x-RCPSP* instance can be modified continuously and different process variants can be considered during optimization.

Formally, the structure of the *x-RCPSP* can be summarized as follows [16]: A process is described by a set of potential activities $\mathcal{A}^+ = \{0, 1, ..., a, a+1\}$. The first and the last element correspond to abstract start and end activities, having a duration of 0 and no resource requirements associated. All remaining elements $i \in \mathcal{A}^+$ have an arbitrary non-negative duration d_i assigned. $\mathcal{A} \subseteq \mathcal{A}^+$ represents the set of *active* activities, implying that the set difference $\mathcal{A}^+ \backslash \mathcal{A}$ groups all *inactive* operations. $\mathcal{R} = \{1, ..., r\}$ defines a set of renewable resource types: For each $k \in \mathcal{R}$, a constant amount of c_k units is available. As regards the description of activity relations, the following modeling constructs are available:

- The elements in $\mathcal{A}^+$ can be ordered by use of *precedence constraints*. $\mathcal{P}^+$ contains all potentially relevant constraints: $p_{i,j} \in \mathcal{P}^+$ states that an activity $i \in \mathcal{A}^+$ has to be finished before or at the start of $j \in \mathcal{A}^+$. Corresponding to the distinction between *active* and *inactive* elements, $\mathcal{P} \subseteq \mathcal{P}^+$ contains only those $p_{i,j}$ for which $i, j \in \mathcal{A}$.
- *Resource requirements* describe the relation between activities and resource types: $\mathcal{Q}^+$ is a two-dimensional array combining all $i \in \mathcal{A}^+$ with all $k \in \mathcal{R}$. The element $q_{i,k} \in \mathcal{Q}^+$ then describes how many units of resource type k are required throughout the execution of i.

Valid activation state modifications are defined by use of the following additional constructs:

- *Activity substitution rules* describe legal forms of deliberate activation state modifications. The existence of the element $x_{i,j}$ in the respective set of potential substitutions $\mathcal{X}^+$ indicates that the activation of $j \in \mathcal{A}^+ \backslash \mathcal{A}$ in exchange for the deactivation of $i \in \mathcal{A}$ represents a valid form of process variation.
- The activation or deactivation of an activity might have an impact on the state of one or several other activities. *Activity dependencies/constraints* therefore describe obligatory activation state modifications associated with the application of activity substitution rules. Elements of the following types are contained in the corresponding set $\mathcal{M}^+$: $m^{\oplus}_{i,j}$ states that activity $j \in \mathcal{A}^+$ has to be activated upon the activation of activity $i \in \mathcal{A}^+$; $m^{\ominus}_{i,j}$ states that j has to be deactivated upon the deactivation of i; $m^{\oplus}_{i,j}$ states that j has to be deactivated upon the activation of i; and $m^{\ominus}_{i,j}$ states that j has to be activated upon the deactivation of i.

Each combination of activation state and corresponding activity starting times represents a solution of the *x-RCPSP*: A *schedule* S thus combines the planned starting times β_i of all activities $i \in \mathcal{A}$ in a vector $(\beta_1, \beta_2, ..., \beta_n)$ with $n = |\mathcal{A}|$. S is valid if both of the following criteria are fulfilled:

- *Activation State Validity.* The activation state of the *x-RCPSP* is represented by the set of active activities $\mathcal{A}$. It is considered valid if $\mathcal{A}$ can be derived from an original (valid) activation state through the application of the substitution rules defined in $\mathcal{X}^+$, taking all constraints of $\mathcal{M}^+$ into account.
- *Starting Times Validity.* If the set of activities running at a time t is denoted as $\mathcal{A}_{[t]} = \{i \in \mathcal{A} | \beta_i \leqslant t < \beta_i + d_i\}$, the following criteria can be defined for starting times validity: (1) $\beta_i \geqslant 0$ for any $i \in \mathcal{A}$, (2) $\beta_i + d_i \leqslant \beta_j$ for any $p_{i,j} \in \mathcal{P}$ and (3) $\sum_{i \in \mathcal{A}_{[t]}} q_{i,k} \leqslant c_k$ for any $k \in \mathcal{R}$ at any t. Note that these criteria alone define schedule validity in the context of the classical RCPSP [17].

For the regarded context, we define a disruption management problem as a combination of the following elements:

- *Execution State.* The state of a disrupted system is described by an existing schedule S and the current point in time t_c (i.e. the time of disruption detection). I is the *x-RCPSP* describing relevant constraints as well as valid forms of process modifications, and forming the basis of S.
- *Disruption.* A disruption D is defined by its type and a set of corresponding parameters: Potential rescheduling factors are described by Li et al. [5] or Vieira et al. [4] for example. However, due to the fact that the classification and definition of Zhu et al. [15] is both more topical and closer related to the RCPSP, we focus on the types of disruptions distinguished there in the following: We take the New Activity, the Precedence, the Activity Duration, the Activity Resource and the Resource Disruption into consideration.
- *Optimization Goal.* The preferred quality of the resulting schedule is defined by an evaluation function $\varphi : S \rightarrow \mathbb{R}$ which is used to convert a schedule into a corresponding cost value. Whereas classical schedule optimization focuses mainly on the minimization of the overall execution time (the so-called *makespan*), more complex goals are usually given in DM problems: Common objectives (or at least part of composite objectives) are the minimization of costs for earliness, tardiness, the application of interventions or deviations from the original schedule. Note that apart from cost minimization of course also quality maximization can be defined as a goal for optimization.

The solution process aims at the identification of an optimal schedule: $S^\star$ is a modification of S (and a legal solution of I), taking the occurrence of D at t_c into account and being better than all potential alternatives in terms of φ. In practical scenarios, the difference between S and $S^\star$ is typically interpreted as the set of interventions to apply. In the following, we use the subsequent two-step approach for the identification of $S^\star$:

1) I is combined with the modifications implied by D in a disrupted problem instance I^D. Accordingly, S^D represents a modified version of S taking D into account and describing what would happen if no intervention was taken. For the identification of S^D any method that returns a feasible (yet still suboptimal) solution within minimal time can be applied: We use a simple right-shift procedure.
2) According to φ optimization is performed on S^D until a predefined breaking condition is fulfilled: In operational DM, it is a common approach to spend a certain amount of time searching a solution as good as possible. The result of optimization is $S^\star$, which can therefore be considered an optimized version of S^D.

LRS provides a possibility to take the information associated with a disruption into account during *optimization*: Thus we focus on the discussion of the latter step in the following.

III. Local Rescheduling

A. Overview

An unforeseen disturbance typically changes some part of the future. Local Rescheduling is based on the idea of responding to a disruption right there where it takes effect: Instead of trying to optimize the entire future immediately, it is first attempted to resolve problems *locally*. Optimization therefore starts within a relatively small time window which is iteratively extended until finally the entire future is regarded: This way, it is made sure that the global optimum is definitively not excluded from the search space.

The use of LRS is encouraged whenever (1) the baseline schedule is (nearly) optimal and (2) there exist possibilities to cope with disruptions on a local level: In any other case, it might be ineffective to focus on time windows smaller than the entire future. However, we claim that in most practical applications of DM the criteria for LRS to unfold its full potential are fulfilled. Firstly, there is usually sufficient time available for the optimization of the initial schedule prior to execution: It is thus typically (at least close to) optimal. Secondly, means for *local* schedule reparation are common: Otherwise responding to disruptions would be even more complex and almost impossible within reasonable time. After all, human process managers who are responsible for operational DM usually also start with the consideration of a relatively small time window and regard more complex options only if sufficient time remains. Note that this is also what makes the approach of LRS both natural and intuitive.

The basic LRS procedure is summarized in Algorithm 1: The regarded time window is defined by a lower bound l_i and an upper bound u_i in each iteration $i = \{1, ..., n\}$. The amount of time by which the regarded period shall be extended is described in m_i and p_i: The former element defines how to reduce the lower bound, the latter element defines how to increment the upper bound. t_c denotes the current time (i.e. the time the disruption is detected) and t_h represents the end of the regarded time horizon. In the presented algorithm, first (line 1) the initial time window is initialized. In an iterative procedure (lines 2-7), the regarded period is then continuously extended: The step sizes are determined (lines 3-4) before the boundaries are updated (line 5) and rescheduling is performed within the modified time window (line 6). Note that $l_0 - \sum_i m_i = t_c$ and $u_0 + \sum_i p_i = t_h$ have to be true if it shall be ensured that the entire future is regarded in the final iteration.

Algorithm 1 LRS (Basic Scheme)
Input Disrupted Schedule S^D, Disruption D, Number of Iterations n, Current Time t_c, End of Regarded Horizon t_h

1: $(l_0, u_0) \leftarrow$ InitializeTimeWindow (S^D, D)
2: **for each** i **in** 1 **to** n **do**
3: $\quad m_i \leftarrow$ GetDownwardStepsize (i, n, l_0, t_c)
4: $\quad p_i \leftarrow$ GetUpwardStepsize (i, n, u_0, t_h)
5: $\quad l_i \leftarrow l_{i-1} - m_i$, $u_i \leftarrow u_{i-1} + p_i$
6: $\quad$ Reschedule (S^D, l_i, u_i)
7: **next**

TABLE I
POTENTIAL STRATEGY FOR TIME WINDOW INITIALIZATION

Type of disruption	l_0	u_0
(1) New Activity	planned starting time of the inserted activity	planned ending time of the inserted activity
(2) Precedence	original starting time of the successor	new planned ending time of the successor
(3) Activity Duration	original ending time of the affected activity	new planned ending time of affected activity
(4) Activity Resource	planned starting time of the affected activity	planned ending time of the affected activity
(5) Resource	start of the period of reduced capacity	end of the period of reduced capacity

The concrete realization of the unspecified methods characterizes a particular implementation of the LRS procedure: (1) INITIALIZETIMEWINDOW defines the way the time window is initialized, (2) RESCHEDULE defines the way rescheduling is conducted for a certain period of interest and (3) GETDOWNWARDSTEPSIZE as well as GETUPWARDSTEPSIZE define the way the time window is extended in each iteration. All of these aspects are discussed in more detail in the following.

B. Time Window Initialization

When deciding on a time window (l_i, u_i), a tradeoff between two contradicting requirements has to be found: On the one hand, the search space shall be kept as small as possible to make optimization simple. On the other hand, it has to be large enough to stand a reasonable chance of containing an adequate solution to the disruption management problem. To cope with these requirements, the proposed Local Rescheduling procedure is based on the approach of extending the initial time window already in the first iteration: As (l_0, u_0) is thus never actually considered for rescheduling but rather defines the starting point for continuous extension, it can be initialized with an interval as tight as possible.

Basically, this means that the initial time window shall only cover the period in which the effects of the occurring disruption unfold. The proper choice of l_0 and u_0 thus depends on the type of disruption to deal with: Table I summarizes a potential strategy for the initialization of the time window given any of the regarded kinds of disruptions. (1) If an additional activity has to be scheduled, resource or precedence conflicts might arise during its execution: Therefore, l_0 is set to the planned starting and u_0 is set to the planned ending time of the inserted activity. (2) If an additional precedence constraint is inserted, Local Rescheduling shall start with the period between the original start and the updated end of the new successor. (3) If the duration of an activity is extended, conflicts are likely to arise within the period of activity extension: l_0 is set to the original, u_0 is set to the new planned ending time of the activity. (4) If an activity requires more resources than originally intended, its entire execution period shall be regarded initially. (5) If the capacity of a resource type decreases during a certain time frame, this entire period of reduced availability shall describe the initial time window.

C. Rescheduling

The actual rescheduling procedure consists of three separate steps: (1) A partial schedule optimization problem is created for all elements within the regarded time window. (2) Optimization is performed on this subproblem. (3) The result of optimization is merged with the original schedule.

As regards the first step, let $I^D = \{\mathcal{R}, \mathcal{A}^+, \mathcal{P}^+, \mathcal{Q}^+, \mathcal{X}^+, \mathcal{M}^+\}$ denote the instance of the *x-RCPSP* that provided the basis for the disrupted schedule S^D. A subproblem $I_s^D = \{\mathcal{R}_s, \mathcal{A}_s^+, \mathcal{P}_s^+, \mathcal{Q}_s^+, \mathcal{X}_s^+, \mathcal{M}_s^+\}$ for the time window (l, u) can be created according to the following strategy, for example:

- *Activities.* Let $\mathcal{A}_{[t]} = \{i \in \mathcal{A} | \beta_i \leqslant t < \beta_i + d_i\}$ again denote the subset of activities scheduled to be running at a time t and $\mathcal{A}_{[l,u]} = \{i \in \mathcal{A} | l < \beta_i, \beta_i + d_i \leqslant u\}$ be the subset of activities scheduled to be running within the time window (l, u). The subset of relevant activities $\mathcal{A}_s^+ \leftarrow \{\mathcal{A}_{[l,u]} \cup (\mathcal{A}_{[l]} \cup \mathcal{A}_{[u]}) \cup (\mathcal{A}^+ \backslash \mathcal{A})\}$ consists of three parts: (1) $\mathcal{A}_{[l,u]}$ denotes the set of operations starting and ending within the time window: They represent the elements which are actually modified during optimization. (2) $\mathcal{A}_{[l]} \cup \mathcal{A}_{[u]}$ combines the activities running at the start and the end of the regarded period: As they are (partly) lying outside the time window of interest, they are marked as *unmodifiable* (neither shifts of starting times nor modifications of activation states are allowed). They still have to be regarded to make sure that associated constraints and requirements are actually considered. (3) $\mathcal{A}^+ \backslash \mathcal{A}$ represents the set of *inactive* elements, which are not part of S^D: All of them are considered in order to preserve all options of applying activity substitutions. Note that this is unproblematic in terms of complexity as only the number of elements in $\mathcal{X}_s^+$ defines the size of the regarded search space.
- *Activity substitution rules.* The set of activities activated/deactivated upon the execution of $x_{i,j} \in \mathcal{X}^+$ can be determined unambiguously for valid instances of the *x-RCPSP*. Let $\mathcal{A}^{\boxplus}(x_{i,j})$ denote the set of all operations activated upon the application of $x_{i,j} \in \mathcal{X}^+$, combining j with all activities activated due to the constraints in $\mathcal{M}^+$. Let $\mathcal{A}^{\boxminus}(x_{i,j})$ denote the set of deactivations associated with $x_{i,j}$, correspondingly. $\mathcal{X}_s^+$ combines all $x_{i,j} \in \mathcal{X}^+$ for which the following conditions are fulfilled: (1) The replaced activity is contained in the subproblem: $i \in \mathcal{A}_s^+$. It is obvious that a substitution rule is never applicable if this first criterion is not true. (2) The deactivations associated with the substitution rule concern only operations scheduled within the period of interest: $\mathcal{A}^{\boxminus}(x_{i,j}) \subseteq \mathcal{A}_{[l,u]}$. If activities outside the regarded time window were deactivated, the effects of the respective substitution would not be traceable. (3) As regards the activations associated with $x_{i,j}$, all additional activities have to be scheduled within (l, u): Only in that case the effect of applying a substitution rule can be evaluated correctly. No element in $\mathcal{A}^{\boxplus}(x_{i,j})$ must therefore have a successor starting before or a predecessor ending after the regarded time window.

- *Activity dependencies.* Following the above argumentation, only constraints for which both the left- and the right-hand side activities are contained in the subproblem are considered. Thus, $\mathcal{M}_s^+$ contains all $m_{i,j}^{\oplus}$, $m_{i,j}^{\ominus}$, $m_{i,j}^{\oplus}$ and $m_{i,j}^{\bigcirc}$ for which $i, j \in \mathcal{A}_s^+$.
- *Precedence constraints.* According to the fact that activities lying (completely) outside the regarded time window are not considered at all in the generated subproblem, all precedence constraints associated with such elements are also omitted. As therefore only constraints for which both the predecessor and the successor are contained in $\mathcal{A}_s^+$ are of interest, the subset of constraints can be defined as $\mathcal{P}_s^+ \leftarrow \{p_{i,j} \in \mathcal{P}^+ | i, j \in \mathcal{A}_s^+\}$.
- *Resource requirements.* In the generated subproblem it is sufficient to consider the requirements associated with activities in $\mathcal{A}_s^+$. Therefore, $\mathcal{Q}_s^+ \leftarrow \{q_{i,k} \in \mathcal{Q}^+ | i \in \mathcal{A}_s^+\}$.
- *Resources.* Only those resource types of which at least one entity is required by the contained activities have to be considered. The subset of resources can therefore be defined as $\mathcal{R}_s \leftarrow \{k \in \mathcal{R} | \sum_{i \in \mathcal{A}_s^+} q_{i,k} > 0,\ q_{i,k} \in \mathcal{Q}_s^+\}$.

As regards the second step, basically the same optimization approaches as used for the creation of the original schedule can be applied. The only aspects that have to be considered are (1) the existence of an initial solution (the part of S^D describing the starting times of elements in $\mathcal{A}_s^+$) as well as (2) the fact that no activity of the resulting (partial) schedule must lie outside the regarded time window.

Regarding the limit of optimization, various strategies can be distinguished for the division of the overall available time among the LRS iterations: One approach is to portion time *linearly* (i.e. an equal amount of time is spent in each iteration), another one to divide it *proportionally* to the number of activities contained in the respective subproblem (i.e. more/less time is spent on subproblems containing more activities). The proper choice depends mainly on the average problem complexity: If the difficulty of finding a solution increases drastically with the number of activities, it might be better to focus on narrow time windows and to therefore apply an indirect proportional division of the available time.

After the predefined breaking condition has been reached, the optimized solution of the subproblem – the subschedule $S_s^\star$ – is merged with the original schedule S^D in the third and last step of the rescheduling procedure: The information on all elements within the time window (l, u) is taken from $S_s^\star$ and used to update S^D accordingly.

D. Extension Scheme

The GETDOWNWARDSTEPSIZE and the GETUPWARDSTEPSIZE methods define the way the time window is extended in each iteration of the LRS procedure. From the many possible forms of decreasing the lower bound from l_0 to t_c and increasing the upper bound from u_0 to t_h, we focus on three approaches in the following:

- *Linear Extension:* The amount by which the time window is expanded is equal in each iteration. The downward step size is thus $m_i = \frac{l_0 - t_c}{n}$ for each $i \in \{1, ..., n\}$ and the upward step size is $u_i = \frac{t_h - u_0}{n}$.

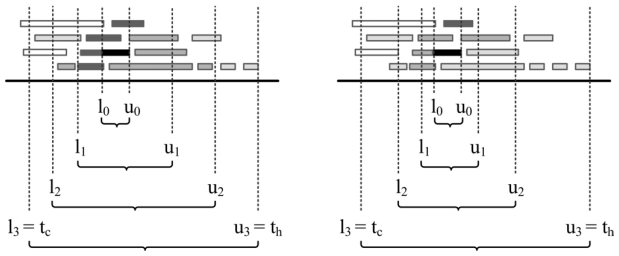

Fig. 1. Linear and Exponential Extension Scheme

- *Exponential Extension:* The amount by which the time window is expanded increases exponentially in each iteration. If $l_0 - t_c > 0$, $t_h - u_0 > 0$ and $n > 1$, the downward and upward step sizes can, for instance, be defined as follows for each $i \in \{1, ..., n\}$:

$$m_i = i^{\frac{\log(l_0 - t_c)}{\log(n)}} \tag{1}$$

$$u_i = i^{\frac{\log(t_h - u_0)}{\log(n)}} \tag{2}$$

  By use of an additional parameter $k \in \mathbb{R}^+$ it is easily possible to vary the step size relations. Consider the following modified version of the above functions, which are valid if $l_0 - t_c > 0$, $t_h - u_0 > 0$ and $n > 0$:

$$m_i = (i + k)^{\frac{\log(l_0 - t_c)}{\log(n + k)}} \tag{3}$$

$$u_i = (i + k)^{\frac{\log(t_h - u_0)}{\log(n + k)}} \tag{4}$$

- *Logarithmic Extension:* The amount by which the time window is expanded increases logarithmically in each iteration. If $n > 0$ and $k \in \mathbb{R}^+$, step size functions can be defined as follows, for example:

$$m_i = \log(i + k) * \frac{l_0 - t_c}{\log(n + k)} \tag{5}$$

$$u_i = \log(i + k) * \frac{t_h - u_0}{\log(n + k)} \tag{6}$$

The selected extension scheme has significant impact on the development of the size of the search space and the complexity of the optimization problem: Whereas a linear form of extension typically implies an increase of the number of considered activities at a constant rate, the exponential (logarithmic) extension scheme rather focuses on periods close to (far from) the original time window. Note again that also other functions can be used to define the way the time window is extended.

The difference between two of the above approaches is illustrated in Figure 1, where several activities are depicted as bars along the (horizontal) time line. The black area visualizes an activity duration disruption, serving as a starting point for the determination of the initial time window (l_0, u_0): The subfigure on the left-hand side shows the linear, the subfigure on the right-hand side shows the exponential extension scheme. Dark-grey activities are considered in the first, grey activities in the second and light-grey activities in the third iteration. It can easily be observed that the size of the regarded subproblems increases much faster based on the linear time window extension scheme.

IV. Performance Evaluation

A. Experimental Setup and Instance Generation

For the evaluation of LRS in comparison with other rescheduling strategies, a generic framework for the solution of disruption management problems has been implemented in Java: Schedule optimization is based on the genetic algorithm described in [16], which itself represents an extension of the RCPSP-specific algorithm proposed by Hartmann [18].

As there are currently neither generators for nor instances of reactive rescheduling problems available [19], we have also decided on the realization of a corresponding test case generator. It can be used to create combinations of baseline schedules and associated sets of disruptions by random but according to various parameters: For example network complexity, resource factor and resource strength are available as defined by Kolisch et al. [20]. Additional settings make it possible to control the number of inter-process relations, the amount of slack time incorporated into the baseline schedule, the number and properties of occurring disruptions, the given structure of activity alternatives, etc. Based on the respective parameters, 16 different problem classes have been defined according to the following configurations:

- *Low/High Process Complexity.* This setting controls the number of precedence constraints linking activities and processes: Low complexity means few, high complexity means many precedence relations.
- *Low/High Resource Complexity.* The aspects of resource requirements and resource availability are combined in this setting: Low complexity means that many resource entities are available to cover few and relatively small requirements, high complexity indicates the opposite.
- *With/Without Left-Shifts.* Scheduling an activity to start earlier than in the original schedule is considered a left-shift. If doing so represents a legal form of modification, more options of rescheduling are available and finding the optimal solution is thus more difficult.
- *Tight/Wide Baseline Schedule.* The distribution of activity starting times and the amount of incorporated slack time control the tightness of a schedule: A schedule is considered tight if activities start at the earliest possible point and tend to be executed in parallel.

Of each defined class, ten problems of three different sizes have been generated (giving an overall of 480 instances): Small-sized problems consist of 10 processes containing 10 activities; medium-sized problems of 30 processes containing 10 activities; and large-sized problems of 50 processes containing 20 activities. Instances of 100, 300 and 1000 activities (executed on three different resource types) were thus considered for evaluation. Note that these problems are much larger than the ones typically regarded in classical schedule optimization (see [21]).

From zero up to five alternatives were associated with each activity, making it possible to vary the process execution path and to therefore minimize the negative impact associated with a disruption. Basically, each of the following forms of alternation was assigned with a probability of $P = 0.05$:

- *Longer duration/less requirements.* The alternative activity requires less resources but its d_i is larger.
- *Shorter duration/more requirements.* The other way round, the alternative activity requires additional resource entities but the associated d_i is smaller.
- *Shorter duration/additional activity.* The alternative lasts shorter but is dependent on an optional activity: If it is activated, an additional activity has to be scheduled.
- *Activity insertion.* The alternative is entirely equal to the original activity, except for the fact that it depends on the execution of an optional (additional) activity.
- *Parallelization.* The alternative activity differs from the original one only in lacking one precedence constraint linking the original activity to one of its successors.

From all given alternatives, one was chosen to be active in the original schedule: All other ones were made less attractive through the assignment of appropriate activity execution costs. This way it is made sure that the baseline schedule is optimal (as was defined as a requirement for LRS to unfold its full potential in Section III-A): The originally scheduled process variant has the smallest costs associated.

As regards the disturbance occurring during the execution of the baseline schedule, in each case the duration of exactly one activity is doubled: This Activity Duration Disruption is assumed to be detected immediately at the start of execution. The goal is then to minimize a weighted sum of overall process tardiness, activity execution costs and the number of schedule modifications: As regards the former element, let δ_i denote the due date assigned to an activity $i \in \mathcal{A}^+$: each time unit the abstract process end activity is scheduled to finish after a predefined δ_i causes costs of 1; as regards the latter element, each temporal shift or application of a process variation causes costs of 3. If Δ denotes the number of such modifications and ϵ_i corresponds to the cost of executing $i \in \mathcal{A}$, the objective can be defined as follows:

$$\min z = 3 * \Delta + \sum_{i \in \mathcal{A}} \epsilon_i + \sum_{i \in \mathcal{A}} \max\left(0, \beta_i + d_i - \delta_i\right) \quad (7)$$

B. Regarded Rescheduling Methods

In order to eliminate the effects of randomness, optimization was performed ten times for each generated instance based on each of the following five rescheduling methods:

- *Full Rescheduling.* In the strategy denoted as FRS, all the available time is spent regarding the entire future. Full Rescheduling can be considered a specific version of LRS: Only $n = 1$ iteration is conducted, the time window is initialized with $l_0 = t_c$ and $u_0 = t_h$ in any case and the size of the steps to take is set to 0.
- *Matchup Scheduling.* It has already been mentioned that the original form of MUP [8] can not easily be applied to more generic problem classes such as the RCPSP or the *x-RCPSP*. However, the basic idea behind Matchup Scheduling – trying to reschedule everything before a matchup point which is incrementally extended until a valid solution is identified – can be adapted to define another specific version of LRS: The lower

bound of the time window is set to $l_0 = l_i = t_c$ for each iteration $i \in \{1, ..., n\}$. In $n = 3$ iterations, the upper bound is extended according to the linear function defined in Section III-D. Note that of course any other extension scheme could also be applied to increase the upper bound in an arbitrary number of iterations: What characterizes Matchup Scheduling is the fact, that the time window is only extended into one direction.

- *Local Rescheduling.* LRS has been implemented according to the above description. The extension schemes proposed in Section III-D form the basis for three different variants of Local Rescheduling: LRS1 employs the linear function, LRS2 is based on the exponential and LRS3 on the logarithmic extension scheme as described in the equations (3-6) with $k = 1$. The number of iterations was set to $n = 3$.

Focusing on *operational* DM, where it is necessary to select interventions in near real-time, we decided on tight time limits: In two different scenarios, the time available for optimization (on a standard desktop PC) was limited to 5 or 15 seconds, respectively. For each of the 160 generated cases, an overall of 10 (runs) $*$ 5 (strategies) $*$ 2 (limits) $= 100$ separate test runs has therefore been conducted.

C. Results

Even by use of the most powerful methods (see [22] for a recent approach) it is not possible to determine the exact optimum for problem instances of the regarded sizes in reasonable time. Therefore, the best value that could be identified during (1) all 100 test runs and (2) an additional

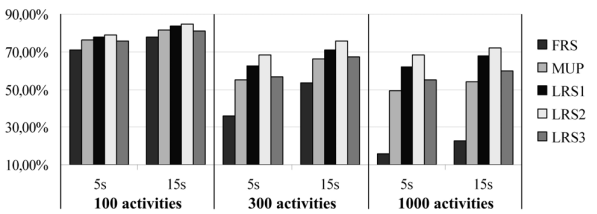

Fig. 2. Overall Performance of the Regarded Rescheduling Strategies

FRS run limited to 10 minutes was taken as a reference instead: If S^D denotes the disrupted and $S^\star$ denotes the best identified solution, the *known* optimization potential can be defined as $\varphi(S^D) - \varphi(S^\star)$: Accordingly, $\pi = \frac{\varphi(S^D)-\varphi(S)}{\varphi(S^D)-\varphi(S^\star)}$ describes the portion of the known potential that could be tapped by an arbitrary schedule S. The figures listed in Table II and summarized in Figure 2 correspond to the average value of π over all solutions identified in the respective category[1]: If, for example, a disrupted schedule caused costs of 10 monetary units and three optimization runs resulted in solutions with costs of 5, 2 and 3 associated, the table would show 83.33% as the average of $\frac{10-5}{10-2} = 62.5\%$, $\frac{10-2}{10-2} = 100.0\%$ and $\frac{10-3}{10-2} = 87.5\%$. Each value in the problem-class specific columns thus aggregates the information on 80 problem instances, for each of which 100 test runs were performed. Analyzing the figures in context, the following observations can be made:

[1]All 480 problem instances and detailed results can be obtained from http://rcpsp.serverside.at/ssci-07.html: They may serve as a starting point for further evaluations and comparisons of related procedures.

TABLE II

PORTION OF THE IDENTIFIED OPTIMIZATION POTENTIAL THAT COULD BE TAPPED BY DIFFERENT RESCHEDULING METHODS ON AVERAGE

Activities	Limit	Method	Process Complexity		Resource Complexity		Left-Shifts		Baseline Schedule		Overall
			low	high	low	high	yes	no	tight	wide	
100	5 sec	FRS	70,01%	71,43%	76,69%	64,75%	55,51%	85,94%	72,38%	69,07%	70,72%
		MUP	73,03%	79,88%	79,28%	73,63%	66,27%	86,64%	79,44%	73,47%	76,46%
		LRS1	74,58%	81,47%	81,37%	74,68%	68,15%	87,90%	80,12%	75,93%	78,02%
		LRS2	76,04%	81,93%	81,68%	76,29%	69,53%	88,44%	80,56%	77,40%	78,98%
		LRS3	73,33%	78,33%	78,37%	73,29%	63,34%	88,32%	77,15%	74,51%	75,83%
	15 sec	FRS	77,40%	78,09%	81,74%	73,75%	63,78%	91,71%	78,90%	76,59%	77,75%
		MUP	77,69%	85,08%	82,41%	80,36%	71,39%	91,37%	83,96%	78,80%	81,38%
		LRS1	80,69%	86,61%	85,44%	81,86%	74,16%	93,14%	86,17%	81,13%	83,65%
		LRS2	81,10%	88,37%	86,49%	82,98%	75,96%	93,51%	85,62%	83,85%	84,74%
		LRS3	79,41%	83,10%	82,72%	79,79%	69,01%	93,50%	82,87%	79,64%	81,26%
300	5 sec	FRS	34,89%	37,41%	42,87%	29,42%	27,94%	44,36%	35,05%	37,25%	36,15%
		MUP	56,22%	54,36%	60,11%	50,47%	50,42%	60,15%	51,82%	58,75%	55,29%
		LRS1	65,06%	59,77%	66,78%	58,05%	57,99%	66,83%	59,95%	64,88%	62,41%
		LRS2	71,22%	65,84%	69,16%	67,90%	61,53%	75,54%	66,27%	70,79%	68,53%
		LRS3	58,23%	55,08%	60,13%	53,18%	50,57%	62,74%	52,36%	60,94%	56,65%
	15 sec	FRS	51,96%	54,61%	62,70%	43,87%	41,16%	65,40%	52,22%	54,34%	53,28%
		MUP	67,60%	65,11%	71,66%	61,05%	59,29%	73,42%	64,32%	68,39%	66,35%
		LRS1	74,42%	67,52%	75,76%	66,19%	64,68%	77,27%	68,97%	72,98%	70,97%
		LRS2	78,03%	73,62%	77,86%	73,79%	68,66%	82,99%	74,14%	77,52%	75,83%
		LRS3	70,11%	64,26%	72,04%	62,33%	58,40%	75,98%	64,15%	70,22%	67,19%
1000	5 sec	FRS	15,90%	15,48%	24,36%	7,02%	5,31%	26,07%	17,94%	13,45%	15,69%
		MUP	49,85%	48,95%	57,12%	41,67%	43,18%	55,61%	47,21%	51,59%	49,40%
		LRS1	60,80%	63,57%	68,31%	56,06%	61,98%	62,39%	57,45%	66,92%	62,18%
		LRS2	69,06%	67,84%	76,74%	60,16%	62,07%	74,83%	62,34%	74,56%	68,45%
		LRS3	56,82%	52,74%	58,67%	50,89%	51,93%	57,63%	49,17%	60,39%	54,78%
	15 sec	FRS	21,85%	23,80%	34,70%	10,94%	9,14%	36,51%	24,38%	21,27%	22,82%
		MUP	51,47%	56,57%	62,28%	45,76%	45,34%	62,70%	51,22%	56,82%	54,02%
		LRS1	64,88%	70,90%	75,56%	60,22%	65,25%	70,52%	63,52%	72,26%	67,89%
		LRS2	71,10%	73,11%	79,87%	64,34%	64,96%	79,25%	67,06%	77,16%	72,11%
		LRS3	58,52%	60,81%	64,35%	54,98%	55,49%	63,84%	54,35%	64,98%	59,67%

- The developed methods of partial rescheduling provide consistently and significantly better results than FRS. Spending the longest time within narrow time windows, LRS2 could achieve the best overall performance: In any case much more than 65% of the known potential could be tapped within only 5 seconds of optimization.
- The relative benefit associated with the application of partial rescheduling increases along with the problem size. As FRS is directly affected by additional activities, it identifies fewer solutions for larger instances within limited time. In contrast, the size of the regarded subproblems increments relatively slowly in LRS.
- If the temporal limit of optimization is extended, the relative benefit associated with the application of partial rescheduling decreases: The more time is spent optimizing, the closer the solver gets to the optimal (or the best known) solution and the harder it gets to identify possibilities of further improvements.
- The detailed evaluation results reveal that the benefit of partial rescheduling is particularly high whenever (1) process and resource complexity are high, (2) left-shifts have to be considered or (3) the baseline schedule is wide. The former two aspects make problems complex and difficult to solve in the full rescheduling approach: They imply the relevance of many constraints and a large size of the search space. The latter aspect supports the characteristics of Local Rescheduling: Subproblems are smaller (i.e. contain fewer activities) if the baseline schedule is wide, which means that many solutions can be identified and evaluated by LRS.

To sum it up, it can be stated that LRS performs particularly well if the considered problem is complex and if there is only little time available for the identification of a good solution: As both of these aspects are typically given in realistic applications of operational disruption management, the use of Local Rescheduling approaches can be suggested.

V. Conclusion

Presenting an efficient approach to disruption management in realistic scenarios, this paper first described how the generic framework of the RCPSP can be extended to make the consideration of practically relevant forms of interventions possible: Based on the concept of alternative activities, the *x-RCPSP* allows to describe also options of process variation – apart from the options of temporally shifting activities or reallocating resource entities.

Second, the strategy of Local Rescheduling was introduced: Inspired by the typical approach of human process managers, it is first tried to resolve problems locally in this method of partial rescheduling. The remaining time is then used to incrementally extend the regarded search space: This way, more far-reaching and more complex forms of schedule reparation shall be identified. How the relevant time window can be initialized, extended and applied for rescheduling has been discussed comprehensively in this paper.

Finally, the conducted performance evaluation was described and respective results were discussed. The comparison of a generic version of MUP and three variants of LRS with FRS revealed a significantly better performance of the proposed partial rescheduling methods. The fact that LRS performs particularly well on large and complex problems where only little time is available for optimization suggests its use in practical applications of disruption management.

References

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

[2] J. Clausen, J. Hansen, J. Larsen, and A. Larsen, "Disruption management," *ORMS Today*, vol. 28, pp. 40–43, 2001.

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

[4] 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.

[5] 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.

[6] R. J. Abumaizar and J. A. Svestka, "Rescheduling job shops under random disruptions," *International Journal of Production Research*, vol. 35, no. 7, pp. 2065–2082, 1997.

[7] G. Huang, J. Lau, K. Mak, and L. Liang, "Distributed supply-chain project rescheduling: part II - distributed affected operations rescheduling algorithm," *International Journal of Production Research*, vol. 44, pp. 1–25, 2006.

[8] 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.

[9] M. S. Akturk and E. Gorgulu, "Match-up scheduling under a machine breakdown," *European Journal of Operational Research*, vol. 112, pp. 81–97, 1999.

[10] 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.

[11] 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, 1999.

[12] S. Hartmann, "Project scheduling with multiple modes: A genetic algorithm," *Annals of Operations Research*, vol. 102, 2001.

[13] 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.

[14] 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.

[15] 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.

[16] 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*, 2007, to appear.

[17] 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*, pp. 147–178. 1999.

[18] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Naval Research Logistics*, vol. 45, 1998.

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

[20] 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.

[21] R. Kolisch and C. S. und Arno Sprecher, "Benchmark instances for project scheduling problems," in *Handbook on recent advances in project scheduling*, J. Weglarz, Ed., pp. 197–212. Kluwer, 1999.

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