Applying Local Rescheduling in Response to Schedule Disruptions

Jürgen Kuster[1], Dietmar Jannach[2 *], and Gerhard Friedrich[1]

[1]Institute for Applied Informatics,
University Klagenfurt, 9020 Klagenfurt, Austria
[2]Department of Computer Science
Dortmund University of Technology, 44221 Dortmund, Germany

[*] Corresponding author: Dietmar Jannach, Technische Universität Dortmund, 44221 Dortmund, Germany. Phone: +49 231 755 7272, E-Mail: dietmar.jannach@tu-dortmund.de

Abstract. In realistic scenarios of disruption management the high number of potential options makes the provision of decision support – on how to get back on track – complex. It is thus desirable to reduce the size of the regarded problems by applying methods of partial rescheduling. As existing approaches (such as Affected Operations Rescheduling or Matchup Scheduling) mainly focus on production-specific problems, we propose Local Rescheduling (LRS) as a generic approach to partial rescheduling in this paper. It integrates previous research on partial rescheduling and local search in the context of complex project scheduling problems. LRS is based on the bidirectional incremental extension of a time window regarded for potential schedule modifications. Experiments show that LRS outperforms previous approaches. †

Keywords. Local Optimization, Partial Rescheduling, Reactive Scheduling, Disruption Management, Extended Resource-Constrained Project Scheduling Problem.

† This is a revised and extended version of the paper originally published in the Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling, Honolulu, USA.

1 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 Yu and Qi (2004) as well as Clausen et al. (2001)). The central aim of DM is to get back on track by selecting appropriate repair actions, directed at the minimization of the negative impact associated with a disruption. Practically relevant options of intervention typically 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.

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 set of operations to be executed has also to be reconsidered (in the *planning* part of the problem). It might be necessary to dynamically insert/remove activities, to change the order of their execution, to serialize/parallelize process steps and so forth. The space of relevant options is thus even larger than in classical schedule optimization. The complexity associated with the task of analyzing all potential combinations of repair actions on the one hand and the typical requirement of responding to a disruption as quickly as possible on the other hand make the use of exact and deterministic algorithms usually inappropriate for the provision of online decision support in realistic scenarios of operational DM. However, it is not sufficient to simply implement a meta-heuristic procedure either. Recent and extensive performance evaluations (see Kolisch and Hartmann (2006), 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

such a number of activities is relatively small when compared with many real-world problems.

In the current literature on project scheduling, different strategies are proposed to cope with uncertainty in scheduling problems (see Herroelen and Leus (2004) as well as Herroelen and Leus (2005) for comprehensive overviews). *Robust* or *proactive scheduling* is intended to develop schedules that anticipate a certain degree of variability. In *dynamic scheduling*, no fixed schedule is generated in advance; instead, the decisions on which activity to execute at what point in time are delayed until the time of execution. They are then dynamically taken based on a given scheduling policy.

Another approach to dealing with uncertainty in dynamic environments is called *predictive-reactive* scheduling, which is based on the idea of developing a predictive (or so-called *baseline*) schedule and reacting to disruptions as they occur. In that context, different strategies can be distinguished for the reparation of broken schedules:

- One typical option is to apply *schedule repair actions*, which correspond to relatively simple methods aimed at the quick reparation of the disrupted schedule. The most popular example of such a repair action is the right-shift rule (see Smith (1994), for example) which simply delays all activities affected by a disruption.
- Another option is to perform *full rescheduling* (FRS). In case of a disruption the remaining part of the schedule is discarded and an entirely new schedule is generated for the remaining activities.

Both of these approaches have their drawbacks. Schedule repair actions often lead to a poor quality of the resulting schedule, i.e. to poor schedule efficiency. Full rescheduling implies large computational efforts and may result in poor schedule stability. This is basically what has motivated the emergence of *partial rescheduling* approaches (see Vieira, Herrmann and Lin (2003) for an overview). Partial rescheduling means to re-schedule only a subset of all future operations, and to thereby attain an optimal combination of schedule efficiency and schedule stability.

When considering existing *partial rescheduling* techniques, previous approaches have mainly focused on production-specific scheduling problems. *Affected Operations Rescheduling* (AOR, see Li, Shyu and Adiga (1993), Abumaizar and Svestka (1997) as well as Huang et al. (2006)) can only be applied to job shop problems as the underlying binary tree allows only for one successor on the job and one successor on the resource associated with an activity. Similarly, *Matchup Scheduling* (MUP, see Bean et al. (1991) as well as Akturk and Gorgulu (1999)) is restricted to problems where activities require no more than one resource. Jain and Elmaraghy (1997) investigated how to apply Genetic Algorithms to the manufacturing scheduling problem and only take a subset of all pending tasks into account when performing rescheduling. Wang and Liu (2006) presented a rolling partial rescheduling procedure for the single-machine large-scale rescheduling problem.

Motivated by the idea of integrating such partial rescheduling procedures with previous work on local search in the context of project scheduling problems (see Wang (2005), for example), we herein present a generic approach to partial rescheduling named Local Rescheduling (LRS). LRS is (1) based on the incremental extension of a time window, in which the algorithm tries to locally repair the broken schedule and (2) can be applied to significantly improve the performance of schedule optimization in the context of complex process disruption management problems.

The remainder of this document is structured as follows: In Section 2, 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 3, the LRS method is presented in detail. In particular, we discuss how the size of the initial time window can be chosen after the occurrence of a disruption and what schemes for window extension can be applied. Section 4 evaluates Local Rescheduling both in comparison with Full Rescheduling and a generalized version of Matchup Scheduling (which is basically a specific form of LRS). The conducted performance evaluation shows that the performance of schedule optimization algorithms can be drastically increased when using LRS. By use of this partial rescheduling method even large problems can be tackled in reasonable time.

2 The Extended RCPSP

The Resource-Constrained Project Scheduling Problem (RCPSP, see Błazewicz, Lenstra and Kan (1983) as well as Brucker et al. (1999)) provides a framework for the formalization of scheduling problems. We take it as a starting point for the description of DM problems due to the following 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, for example, be found in Kolisch and Hartmann (2006).
- 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 therefore 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 for instance not possible to consider repair options such as the dynamic insertion or removal of activities, changes in the order of activity execution or the serialization/parallelization of process steps. Although in recent years methods have been proposed that aim at the integration of planning and scheduling in the solution process - see for example Laborie (2003) who proposes constraint propagation techniques for the generation of partially ordered solutions instead of fully instantiated ones - the literature on corresponding *conceptual extensions* of the underlying models toward Disruption Management is rather sparse. In the context of the RCPSP, to the best of our knowledge, the only approach to additional structural flexibility has been made in the Multi-Mode RCPSP (MRCPSP, see Hartmann (2001)), which allows the definition of different *execution modes* for each activity. These execution modes allow us 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 for instance Artigues, Michelon and Reusser (2003) as well as Elkhyari, Guéret and Jussien (2004)) has mainly focused on the *scheduling part* of respective problems, meaning that only activity starting times and resource allocation can be modified. The approach that covers the widest range of potential repair interventions is the one presented by Zhu, Bard and Yu (2005). Apart from rescheduling they also take mode alternations and resource availability increases into account. Based on these extensions, it is possible to model intervention alternatives such as "subcontracting" and "activity cancellation". In order to make it possible to model even more general *process variations*, we have proposed to extend the classical RCPSP by the concept of alternative activities in a previous work. The Extended Resource Constrained Project Scheduling Problem (*x-RCPSP*, see Kuster, Jannach and Friedrich (2007)) 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 (Kuster, Jannach and Friedrich (2007)). A process is described by a set of potential activities $A^+ = \{0, 1, ..., a, a+1\}$. The first and the last element correspond to abstract start and end activities, having duration of *0* and no resource requirements associated. All remaining elements $i \in A^+$ have an arbitrary non-negative duration d_i assigned. $A \subseteq A^+$ represents the set of *active* activities, implying that the set difference $A^+ \setminus A$ corresponds to the set of *inactive* operations. $R = \{1, ..., r\}$ defines a set of renewable resource types. For each $k \in R$, a constant amount of c_k units is available. With respect to the description of activity relations, the following modeling constructs are available.

- The elements of A^+ can be ordered with the help of *precedence constraints*. P^+ contains all potentially relevant constraints: $p_{i,j} \in P^+$ states that an activity $i \in A^+$

has to be finished before or at the start of $j \in A^+$. Corresponding to the distinction between *active* and *inactive* elements, $P \subseteq P^+$ contains only those $p_{i,j}$ for which $i,j \in A$.

- *Resource requirements* describe the relation between activities and resource types. Q^+ is a two-dimensional array combining all $i \in A^+$ with all $k \in R$. The element $q_{i,k} \in Q^+$ 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 X^+ indicates that the activation of $j \in A^+ \setminus A$ in exchange for the deactivation of $i \in A$ (i.e. i is substituted by j) 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 can be contained in the corresponding set M^+: $m^{\oplus}_{i,j}$ states that activity $j \in A^+$ has to be activated upon the activation of activity $i \in A^+$; $m^{\circ}_{i,j}$ states that j has to be deactivated upon the deactivation of i; $m^{\subset}_{i,j}$ states that j has to be deactivated upon the activation of i; and $m^{\supset}_{i,j}$ states that j has to be activated upon the deactivation of i.

When developing an x-RCPSP model in practice, it is first of all necessary to identify all activity execution modes, replacements and order changes that are possible in the domain. These process variants have then to be modeled by means of activity alternatives. In fact, many practically relevant forms of interventions can be modeled by activity substitutions alone, as has been shown in Kuster, Jannach and Friedrich (2007). The formal x-RCPSP model finally consists of this set of potentially relevant activities A^+ as well as the sets X^+ and M^+ that describe the set of activity

substitutions and activity dependencies respectively. For further details on how to model practical DM problems and for more examples see Kuster, Jannach and Friedrich (2008).

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 $i \in A$ in a vector $(\beta_1, \beta_2, ..., \beta_n)$ with $n=|A|$. *S* is valid if both of the following criteria are fulfilled.

- *Activation State Validity.* The "activation state" of an *x-RCPSP* corresponds to the set of active activities A. An activation state is considered valid if A can be derived from an original (valid) activation state through the application of the substitution rules defined in X^+, taking all constraints of M^+ into account.
- *Starting Times Validity.* If the set of activities running at a time *t* is denoted as $A_{(t)} = \{ i \in A \mid \beta_i \leq t < \beta_i + d_i \}$, the following criteria can be defined for starting times validity:

  (1) $\beta_i \geq 0$ for any $i \in A$

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

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

  Note that these criteria alone define schedule validity in the context of the classical RCPSP (see Kolisch and Hartmann (1999)).

In the context of this work, *disruption management problems* have to be solved in the context of the following characteristics.

- *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). *J* is the *x-RCPSP* describing the constraints and dependencies of the domain as well as the valid forms of process modifications. The *x-RCPSP* model is the basis of the existing schedule *S*.
- *Disruption.* A disruption *D* is defined by its type and a set of corresponding parameters. Typical forms of disruptions (so-called *rescheduling factors*) are for instance described by Li, Shyu and Adiga (1993) or Vieira, Herrmann and Lin

(2003). A more topical set of definitions and classifications of disruptions in the context of the RCPSP are given by Zhu, Bard and Yu (2005). We will focus our subsequent discussion on the types of disruptions described in this more recent paper. In particular, we will consider the "New Activity", the "Precedence", the "Activity Duration", the "Activity Resource" and the "Resource Disruption".

- *Optimization Goal.* The preferred quality of the resulting schedule is defined by an evaluation function φ: $S \rightarrow \mathrm{R}$ which is used to convert a schedule into a corresponding cost value. While 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 $S^{\star}$, which is a modification of S (and a legal solution of J), taking the occurrence of D at t_c into account and being better than all potential alternatives with respect to the evaluation function 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 an optimal schedule.

- J 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 suboptimal) solution within minimal time can be applied. In our experiments we use a simple right-shift procedure.
- *Optimization* is performed on S^D according to φ until a predefined termination condition is fulfilled. In operational DM, it is a common approach to set a fixed time limit that can be used for optimizing a solution. The result of the optimization phase is S^*, which corresponds to an optimized version of S^D.

In summary, the x-RCPSP can be used to formulate a wide range of practical disruption management problems which can be solved with the help of the above-described procedure. In the following, Local Rescheduling (LRS) is presented as a meta-level approach which makes it possible to handle even large problems of realistic size in reasonable time. LRS takes the additional information associated with an occurring disruption into account and can thus be used to significantly improve the performance of underlying schedule optimization algorithms.

3 Local Rescheduling

In this section, Local Rescheduling is presented as a generic and efficient technique for solving DM problems. First, an overview on the general LRS procedure is given before technical aspects of Local Rescheduling are discussed in detail.

3.1 Overview

An unforeseen disturbance typically changes some part of the future. Local Rescheduling is based on the idea of responding to a disruption right where it takes effect. Instead of trying to optimize the entire future immediately, it is first attempted to resolve problems *locally*. Optimization (in the LRS approach) 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 not excluded from the search space.

Intuitively, the use of LRS may be encouraged whenever (1) the baseline schedule is (nearly) optimal and (2) possibilities to cope with disruptions on a local level exist in the domain. 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, in real-world scenarios, sufficient time is available for the optimization of the initial schedule prior to execution. Schedules are 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 look at 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 iterative LRS procedure is summarized in Algorithm 1. In each iteration $i = \{1, ..., n\}$, a time window is defined by a lower bound l_i and an upper bound u_i. The amount of time by which the regarded period shall be extended is described in Δl_i and Δu_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 time window size is initialized. In an iterative procedure (lines 2-7) the period of time taken into consideration for repair 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 $l_0 - \sum_i \Delta l_i = t_c$ and $u_0 + \sum_i \Delta u_i = t_h$ have to be true in order to ensure that the entire future is regarded in the final iteration.

<< Algorithm 1 >>

The concrete realization of the unspecified methods in the Algorithm 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 time window, 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 sections.

Note that the notion of time windows has been used for the reduction of the regarded search space before. Zhu, Bard and Yu (2005) for instance apply a so-called *recovery time window* to limit the size of a deviation and to restrict the extent by which an existing schedule may be modified. The main difference between their approach and the LRS procedure presented herein is the fact Zhu, Bard and Yu regard

the time window as *given*, i.e. as a fixed characteristic of the problem that describes the bounds for a certain type of intervention. In contrast, in our work time windows are dynamically initialized and modified in LRS. They represent the dynamic part of an iterative solution procedure aimed at an intelligent reduction of the problem size.

3.2 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. Since (l_0, u_0) is therefore 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 1 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.

Note that of course other strategies of time window initialization can be applied. The proper choice mainly depends on the structure of the problem as well as on the characteristics of the given baseline schedule. Based on our experiments, it can for instance be stated that tight initial time windows are better for scenarios in which many short activities are executed in parallel. In such situations, even small time windows yield subproblems of significant size, which offer a good chance of containing good-quality solutions.

3.3 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 subschedule resulting from optimization is merged with the original schedule.

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

- *Activities*. Let $A_{(t)} = \{ i \in A \mid \beta_i \leq t < \beta_i + d_i \}$ again denote the subset of activities scheduled to be running at a time *t* and $A_{(l,u)} = \{ i \in A \mid l < \beta_i, \beta_i + d_i \leq u \}$ be the subset of activities scheduled to be running within the time window *(l, u)*. The subset of relevant activities $A^+_s \leftarrow \{ A_{(l,u)} \cup (A_{(l)} \cup A_{(u)}) \cup (A^+ \setminus A) \}$ consists of three parts: (1) $A_{(l,u)}$ is the set of operations starting and ending within the time window. They represent the elements which are actually modified during optimization. (2) $A_{(l)} \cup A_{(u)}$ 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) $A^+ \setminus A$ represents the set of *inactive* elements, which are not part of S^D: All of them are taken into consideration in order to preserve all options of applying activity

substitutions. Note that this is unproblematic in terms of problem complexity as only the number of elements in X^+_s defines the size of the regarded search space.

- *Activity substitution rules.* The set of activities activated/deactivated upon the execution of $x_{i,j} \in X^+$ can be determined unambiguously for valid instances of the *x-RCPSP*. Let $A^{\oplus}(x_{i,j})$ denote the set of all operations activated upon the application of $x_{i,j} \in X^+$. It comprises j as well as all activities activated due to the constraints in M^+. Let $A^{\Theta}(x_{i,j})$ denote the set of deactivations associated with $x_{i,j}$, correspondingly. X^+_s combines all $x_{i,j} \in X^+$ for which the following conditions are fulfilled: (1) The replaced activity is contained in the subproblem: $i \in A^+_s$. 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: $A^{\Theta}(x_{i,j}) \subseteq A_{(l,u)}$. If activities outside the regarded time window were deactivated, the effects of the respective substitution would not be traceable. (3) With respect to 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 $A^{\oplus}(x_{i,j})$ 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, M^+_s contains all $m^{\oplus}_{i,j}$, $m^{\circ}_{i,j}$, $m^{\subset}_{i,j}$ and $m^{\supset}_{i,j}$ for which $i,j \in A^+_s$.
- *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 A^+_s are of interest, the subset of constraints can be defined as $P^+_s \leftarrow \{ p_{i,j} \in P^+ \mid i,j \in A^+_s \}$.
- *Resource requirements.* In the generated subproblem it is sufficient to take the requirements associated with activities in A^+_s into consideration. Therefore, $Q^+_s \leftarrow \{ q_{i,k} \in Q^+ \mid i \in A^+_s \}$.

- *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 $R_s \leftarrow \{ k \in R \mid \sum_{i \in A+s} q_{i,k} > 0, q_{i,k} \in Q^+_s \}$.

Regarding 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 A^+_s) as well as (2) the fact that no activity of the resulting (partial) schedule must lie outside the regarded time window.

With respect to the limit of optimization, various strategies can be distinguished for distributing the overall available time among the separate LRS iterations. One approach is to partition the 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 current subproblem (i.e. more/less time is spent on subproblems containing more activities). The proper choice mainly depends on the 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 – is merged with the original schedule S^D in the third and last step of the rescheduling procedure. This can easily be done by updating S^D according to the information provided by S^*_s, i.e. the starting times of all activities scheduled within the time window *(l, u)* are set to the values defined in S^*_s.

Note that similar approaches to identifying, solving and extending subproblems have been proposed for the RCPSP. Palpant, Artigues and Michelon (2004) for instance, propose to decompose instances of the RCPSP in a *large neighborhood* search procedure named LSSPER. The criteria by which activities are selected for inclusion in the subproblem are, however, different to our approach. LSSPER is based on activity relations whereas LRS is based on time windows. Another difference between the two approaches is that LSSPER defines a concrete solution technique for the RCPSP whereas LRS represents a general method for iteratively solving large-scale scheduling problems which is independent of the underlying problem solving

method. Finally, also the fact that LSSPER is applied to pure RCPSP problems whereas LRS is applied to the more generic x-RCPSP herein represents a major difference between our work and the approach described in Palpant, Artigues and Michelon (2004). Other approaches to decomposing the RCPSP can be found in Mausser and Lawrence (1997), Sprecher (2002) as well as Debels and Vanhoucke (2007).

3.4 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. For each $i \in \{1, ..., n\}$ the downward and the upward step size is thus defined as follows:

(1) $$\Delta l_i = \frac{l_0 - t_c}{n}$$

(2) $$\Delta u_i = \frac{t_h - u_0}{n}$$

- *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 $i \in \{1, ..., n\}$:

(3) $$\Delta l_i = i^{\frac{\log(l_0 - t_c)}{\log(n)}}$$

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

It is easily possible to vary the relation between the sizes of the individual steps with an additional parameter k. Consider for example the following modified version of the above functions, which are valid if $l_0 - t_c > 0$, $t_h - u_0 > 0$ and $n > 0$:

(5) $$\Delta l_i = (i+k)^{\frac{\log(l_0 - t_c)}{\log(n+k)}}$$

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

- *Logarithmic Extension:* The amount by which the time window is expanded increases logarithmically in each iteration. For $n > 0$ and the step size relation parameter k, step size functions can for instance be defined as follows.

(7) $$\Delta l_i = \log(i+k) * \frac{l_0 - t_c}{\log(n+k)}$$

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

The selected extension scheme has a significant impact on the development of the size of the search space and the complexity of the optimization problem. While a linear form of extension typically implies an increase of the number of activities to be considered 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.

<< Figure 1 >>

4 Performance Evaluation

In this section, the performance evaluation that has been conducted in order to compare LRS with the approaches of full rescheduling and matchup scheduling is presented. First, the experimental setup and the generated problem instances are discussed, before the regarded rescheduling methods are summarized. Finally, the results of the evaluation are presented and analyzed.

4.1 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. Both the generation of the baseline schedules as well as the rescheduling step of the LRS procedure are based on the genetic algorithm described in Kuster, Jannach and Friedrich (2007), which itself is an extension of the RCPSP-specific algorithm proposed by Hartmann (1998). Note that also other algorithms for solving the x-RCPSP can be applied, as LRS is a meta-level procedure which can be implemented separately from the underlying optimization approach.

As there are currently neither generators for nor instances of reactive rescheduling problems available (Policella and Rasconi (2005)), we have decided to implement an own test case generator. This generator has been used to create various combinations of baseline schedules and associated sets of disruptions by random but according to various parameters settings. Examples of parameters, which can be adjusted for the experiments, are network complexity, resource factor or resource strength (as defined by Kolisch, Sprecher and Drexl (1995)). Additional parameters make it possible to control the number of inter-process dependencies, the amount of slack time incorporated into the baseline schedule, the amount and characteristics of occurring disruptions, or the structure of activity alternatives and process variants. Based on

these 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. Note that restricting an activity to start only at or after its original starting time is also known as *railroad scheduling assumption* (see Lambrechts, Demeulemeester and Herroelen (2007)).
- *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. In a wide schedule, activities start only after a certain amount of slack time has passed and they are therefore less likely to be executed simultaneously. Note that this problem class might also be referred to as *time buffering approach* (see Lambrechts, Demeulemeester and Herroelen (2007)).

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; large-sized problems of 50 processes containing 20 activities. Instances of 100, 300 and 1000 activities (executed on three different resource types) have thus been considered for evaluation. Note that these problems are much larger than the ones typically regarded in classical schedule optimization, see Kolisch, Schwindt and Sprecher (1999).

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 has been assigned with a probability of $P = 0.05$.

- *Longer duration/fewer requirements.* The alternative activity requires less resources but its duration d_i is longer.
- *Shorter duration/more requirements.* 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, which links the original activity to one of its successors.

In order to make sure that the generated baseline schedule is optimal (as was defined as a requirement for LRS to unfold its full potential in Section 3.1) from all given activity alternatives one was chosen to represent the optimal form of process execution. Basically, all other available options were made less attractive through the assignment of appropriate activity execution costs. The chosen optimum had the smallest costs associated and has been activated in the original schedule.

With respect to the disruption occurring during the execution of the baseline schedule, in each test 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 of optimization is expressed by the schedule evaluation function φ. In real world situations this function is typically complex and highly dynamic as it might be necessary to consider multiple, potentially even conflicting goals, see for example Viana and de Sousa (2000) as well as Cowling et al. (2006). For illustrative purposes, however, it is also possible to apply a relatively simple function as the process of

evaluating the quality of a schedule can be regarded independent from the actual optimization procedure. The following remarks shall be made on the function that had to be minimized in the context of our experiments. We assume that the goal of optimization is to minimize a weighted sum of (1) overall process tardiness, (2) activity execution costs and (3) the number of schedule modifications. For the determination of process tardiness we define δ_i as the due date assigned to an activity $i \in A^+$: Each time unit the abstract process end activity is scheduled to finish after a predefined δ_i was assumed to cause costs of *1* monetary unit. For the determination of activity execution costs, ε_i is used to denote the costs of executing $i \in A$. The values were chosen in a way that made the baseline schedule optimal (see above). In order to simulate a cost situation which is typical for disruption management scenarios, it was furthermore assumed that a schedule modification (like a temporal shift of an activity or a process variation) is rather expensive and induces costs of 3 monetary units. If Δ denotes the number of such modifications, the objective can be defined as follows:

$$\min z = 3 * \Delta + \sum_{i \in A} \varepsilon_i + \sum_{i \in A} \max(0, \beta_i + d_i - \delta_i) \quad (9)$$

4.2 Evaluated Rescheduling Methods

In order to reduce 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 of the disrupted plan. Full Rescheduling can be considered a specific version of LRS. Only $n = 1$ iteration is made, 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.* As described above, the original form of MUP (see Bean et al. (1991)) 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. In such a version, the lower bound of the time window is set to $l_0 = l_i = t_c$ for each iteration $i \in \{1, ..., n\}$. In $n = 3$ steps the upper bound is extended according to the linear function defined in Section 3.4. The number of iterations has been chosen in accordance with the value applied in Local Rescheduling (see below). 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 descriptions. The extension schemes proposed in Section 3.4 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 (5-8) with $k = 1$. After the experimental evaluation of various exemplary test cases, which aimed at the identification of optimal testing parameters, the number of performed iterations was set to $n = 3$.

Having our focus 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 - Intel Pentium M with 1400 MHz and 768 MB RAM) 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 performed.

4.3 Results

Even by use of the most powerful methods (see Laborie (2005) 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 FRS run limited to 10 minutes was taken as a reference instead. If S^D denotes the disrupted and S^* denotes the best identified solution, the known optimization potential can be defined as $\varphi(S^D) - \varphi(S^*)$.

Accordingly, $\pi = \frac{\varphi(S^D) - \varphi(S)}{\varphi(S^D) - \varphi(S^*)}$ describes the portion of the known potential that could be tapped by an arbitrary schedule *S*. The figures listed in Table 2 and summarized in Figure 2 correspond to the average value of π over all solutions identified in the respective category. 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.

- 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, as many constraints have to be considered and the size of the regarded

search space is large. 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.

<< Figure 2 >>

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. Both aspects are typically given in realistic applications of operational disruption management.

5 Conclusion

In this paper, we presented an efficient approach to disruption management in realistic scenarios. First we showed how the generic framework of the RCPSP can be extended to the *x-RCPSP* to make the consideration of practically relevant forms of interventions possible. The *x-RCPSP* is based on the concepts of alternative activities and activity dependencies, and allows us to describe process variations – apart from the options of temporally shifting activities or reallocating resource entities.

Second, the Local Rescheduling (LRS) was introduced as a generic method for partial rescheduling. Inspired by the typical approach of human process managers, LRS first tries to resolve problems locally; 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 can be identified. Aspects of how the relevant time window can be initialized and extended have been comprehensively discussed.

Finally, a performance evaluation was conducted in order to compare the performance of LRS with other approaches. The comparison of three different variants of LRS with a generic version of Matchup-Scheduling and Full Rescheduling techniques 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 indicates its practical applicability for realistic disruption management problems.

References

Abumaizar, Riyad J., and Joseph A. Svestka. (1997). "Rescheduling Job Shops under Random Disruptions", *International Journal of Production Research,* 35(7), 2065-2082.

Akturk, M. Selim, and Elif Gorgulu. (1999). "Match-up Scheduling under a Machine Breakdown", *European Journal of Operational Research*, 112(1), 81-97.

Artigues, Christian, Philippe Michelon, and Stéphane Reusser. (2003). "Insertion Techniques for Static and Dynamic Resource Constrained Project Scheduling", *European Journal of Operational Research*, 149, 249-267.

Bean, James C., John R. Birge, John Mittenthal, and Charles E. Noon. (1991). "Matchup Scheduling with Multiple Resources, Release Dates and Disruptions", *Operations Research,* 39(3), 470-483.

Błazewicz, Jacek, Jan Karel Lenstra, and Alexander Rinnooy Kan. (1983). "Scheduling Projects to Resource Constraints: Classification and Complexity", *Discrete Applied Mathematics*, 5, 11-24.

Brucker, Peter, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. (1999). "Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods", *European Journal of Operational Research*, 112, 3-41.

Clausen, Jens, Jesper Hansen, Jesper Larsen, and Allan Larsen. (2001). "Disruption Management", *ORMS Today*, 28, 40-43.

Cowling, Peter, Nic Colledge, Keshav Dahal and Stephen Remde. (2006). "The Trade Off Between Diversity and Quality for Multi-objective Workforce Scheduling". In Jens Gottlieb and Günther Raidl (eds.), *Evolutionary Computation in Combinatorial Optimization*, LNCS 3906, 13-24.

Debels, Dieter, and Mario Vanhoucke. (2007). "A Decomposition-Based Genetic Algorithm for the Resource-Constrained Project-Scheduling Problem", *Operations Research*, 55(3), 457-469.

Elkhyari, Abdallah, Christelle Guéret, and Narendra Jussien. (2004). "Constraint Programming for Dynamic Scheduling Problems". In H. Kise (ed.), *ISS'04 International Scheduling Symposium*. Awaji, Japan: Japan Society of Mechanical Engineers.

Hartmann, Sönke (1998). "A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling", *Naval Research Logistics*, 45, 733-750.

Hartmann, Sönke. (2001). “Project Scheduling with Multiple Modes: A Genetic Algorithm”, *Annals of Operations Research*, 102, 111-135.

Herroelen, Willy, and Roel Leus. (2004). “Robust and Reactive Project Scheduling: A Review and Classification of Procedures”, *International Journal of Production Research*, 42(8), 1599-1620.

Herroelen, Willy, and Roel Leus. (2005). “Project Scheduling under Uncertainties: Survey and Research Potentials”, *European Journal of Operational Research*, 165(2), 289-306.

Huang, George Q., Jason S. K. Lau, Kwok-ken L. Mak, and Liang Liang. (2006). “Distributed Supply-Chain Project Rescheduling: Part II - Distributed Affected Operations Rescheduling Algorithm”, *International Journal of Production Research,* 44, 1-25.

Jain, A.K., and H. A Elmaraghy. (1997). “Production Scheduling/Recheduling in Flexible Manufacturing”, *International Journal of Production Research*, 35(1), 281-309.

Kolisch, Rainer, Arno Sprecher, and Andreas Drexl. (1995). “Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems”, *Management Science*, 41, 1693-1703.

Kolisch, Rainer, Christoph Schwindt, and Arno Sprecher. (1999). “Benchmark Instances for Project Scheduling Problems”. In: J. Weglarz (ed.), *Handbook on Recent Advances in Project Scheduling*.

Kolisch, Rainer, and Sönke Hartmann. (1999). “Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis”. In: J. Weglarz (ed.), *Project Scheduling: Recent Models, Algorithms, and Applications*.

Kolisch, Rainer, and Sönke Hartmann. (2006). “Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update”, *European Journal of Operational Research,* 174(1), 23-37.

Kuster, Jürgen, Dietmar Jannach, and Gerhard Friedrich. (2007). “Handling Alternative Activities in Resource-Constrained Project Scheduling Problems”. In: M. Veloso (ed.), *IJCAI-07, Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*.

Kuster, Jürgen, Dietmar Jannach, and Gerhard Friedrich. (2008). “Extending the RCPSP for Modeling and Solving Disruption Management Problems”, *Applied Intelligence*, to appear.

Laborie, Philippe. (2003). “Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results”, *Artificial Intelligence*, 143(2), 151-188.

Laborie, Philippe. (2005). “Complete MCS-Based Search: Application to Resource Constrained Project Scheduling”. In: L. Kaelbling and A. Saffiotti (eds.), *IJCAI-05, Proceedings of the 19th International Joint Conference on Artificial Intelligence*.

Lambrechts, Olivier, Erik Demeulemeester, and Willy Herroelen. (2007). “Proactive and Reactive Strategies for Resource-Constrained Project Scheduling with Uncertain Resource Availabilities”, *Journal of Scheduling*, to appear.

Li, Rong-Kwei, Yu-Tang Shyu, and Sdash Adiga. (1993). “A Heuristic Rescheduling Algorithm for Computer-based Production Scheduling Systems”, *International Journal of Production Research,* 31(8), 1815-1826.

Mausser, Helmut E., and Stephen R. Lawrence. (1997). “Exploiting Block Structure to Improve Resource-Constrained Project Schedules”. In: I.H. Osman and J.P. Kelly (eds.), *Meta-Heuristics: Theory and Applications*, 203-218.

Palpant, Mireille, Christian Artigues, and Philippe Michelon. (2004). “LSSPER: Solving the Resource-Constrained Project Scheduling Problem with Large Neighbourhood Search”, *Annals of Operations Research*, 131(1-4), 237-257.

Policella, Nicola, and Riccardo Rasconi. (2005). “Testsets Generation for Reactive Scheduling”, *Workshop on Experimental Analysis and Benchmarks for AI Algorithms,* Ferrara, Italy.

Smith, Stephen F. (1994). “Reactive Scheduling Systems”. In: D. Brown and W. Scherer (eds.), *Intelligent Scheduling Systems*.

Sprecher, Arno. (2002). “Network Decomposition Techniques for Resource-Constrained Project Scheduling”, *Journal of the Operational Research Society*, 53 (4), 405-414.

Viana, Ana, and Jorge Pinho de Sousa. (2000). “Using Metaheuristics in Multiobjective Resource Constrained Project Scheduling”, *European Journal of Operational Research*, 120(2), 359-374.

Vieira, Guilherme E., Jeffrey W. Herrmann, and Edward Lin. (2003). “Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods”, *Journal of Scheduling*, 6(1), 39-62.

Wang, Bing, and Tao Liu. (2006). “Rolling Partial Rescheduling with Efficiency and Stability Based on Local Search Algorithm”. In: De-Shuang Huang, Kang Li, and George W. Irwin (eds.), *Intelligent Computing*, LNCS 4113, 937-942.

Wang, Juite. (2005). “Constraint-Based Schedule Repair for Product Development Projects with Time-Limited Constraints”, *International Journal of Production Economics*, 95(3), 399-414.

Yu, Gang, and Xiangtong Qi. (2004). *Disruption Management: Framework, Models and Applications*. Singapore: World Scientific Publishing.

Zhu, Guidong, Jonathan F. Bard, and Gang Yu. (2005). "Disruption Management for Resource-Constrained Project Scheduling", *Journal of the Operational Research Society*, 56, 365-381.

Algorithms and Tables

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: $\Delta l_i \leftarrow$ GETDOWNWARDSTEPSIZE (i, n, l_0, t_c)

4: $\Delta u_i \leftarrow$ GETUPWARDSTEPSIZE (i, n, u_0, t_h)

5: $l_i \leftarrow l_{i-1} - \Delta l_i$, $u_i \leftarrow u_{i-1} + \Delta u_i$

6: RESCHEDULE (S^D, l_i, u_i)

7: **next**

Table 1. Potential Strategy for LRS 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 the 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

Table 2. 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%

Figures

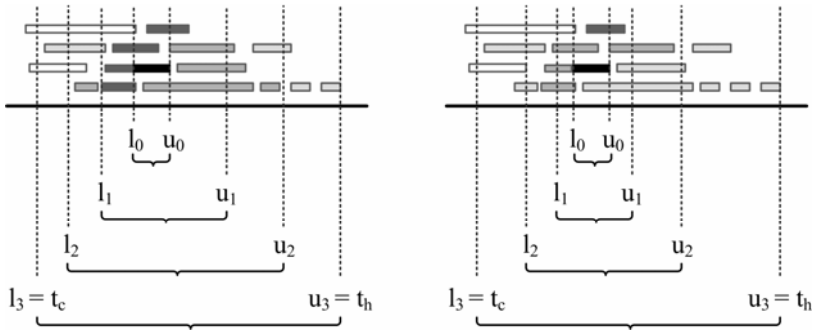

Figure 1. Linear and Exponential LRS Extension Scheme.

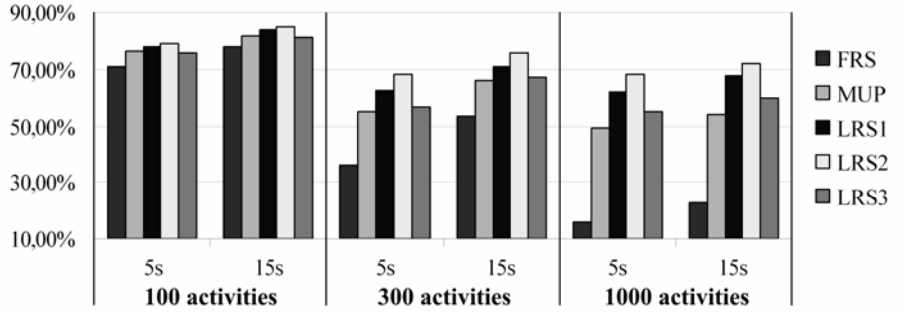

Figure 2. Overall Performance of the Regarded Rescheduling Strategies.