Approaches to Efficient Resource-Constrained Project Rescheduling

Jürgen KUSTER [a,1] and Dietmar JANNACH [a]
[a] *Department of Business Informatics and Application Systems, University of Klagenfurt, Austria*

Abstract. The intrinsic and pervasive uncertainty of the real world causes unforeseen disturbances during the execution of plans. The continuous adaptation of existing schedules is necessary in response to the corresponding disruptions. Affected Operations Rescheduling (AOR) and Matchup Scheduling (MUP) have proven to be efficient techniques for this purpose, as they take only a subset of future activities into account. However, these approaches have mainly been investigated for the job shop scheduling problem, which is too restrictive for many practical scheduling problems outside the shop floor. Thus, this paper analyzes and describes how the respective concepts can be extended for the more generic problem class of the Resource-Constrained Project Scheduling Problem (RCPSP). The conducted evaluation reveals that particularly our generalized version of AOR (GAOR) can yield significant performance improvements in comparison to the strategy of rescheduling all future activities.

Keywords. Resource-Constrained Project Scheduling Problem, Rescheduling, Disruption Management, Decision Support Systems

1. Introduction

Most work on project scheduling is based on the assumption of complete information and a static and fully deterministic environment [17]. However, a schedule is typically subject to the intrinsic and pervasive uncertainty of the real world, as soon as it is released for execution: Disruptions occur and elaborated plans become obsolete. The continuous adaptation and reparation of schedules is thus essential for efficient operation.

Rescheduling (or reactive scheduling) is the process of updating an existing schedule in response to disruptions or other changes [16]. Basically, the options of (1) simply shifting all subsequent activities, (2) rescheduling only a subset of affected operations or (3) generating an entirely new schedule for all of the remaining activities can be distinguished. The first method is computationally inexpensive but can lead to poor results [9] whereas the last one maximizes schedule quality, typically requiring high computational effort and imposing a huge number of schedule modifications [17]. The option of partial rescheduling represents sort of a tradeoff: It aims at the identification of a schedule which provides the optimal combination of schedule efficiency and schedule stability at reasonable computational costs.

[1]Corresponding Author: Jürgen Kuster, Universitätsstraße 65-67, 6020 Klagenfurt, Austria. E-mail: jkuster@ifit.uni-klu.ac.at.

The idea of considering only the operations directly or indirectly affected by a disruption has first been articulated by Li et al. [14]: All activities succeeding a delayed operation within the regarded job or on the processing machine are recursively added to a binary tree if not sufficient slack time is available to compensate the disruption. The time effects associated with the nodes describe imposed schedule modifications. Abumaizar and Svestka [1] illustrate the superior performance of this approach – also known as Affected Operations Rescheduling (AOR) – if compared to the right-shift rescheduling and full rescheduling methods; Huang et al. [10] recently introduced a distributed version of AOR. The matchup scheduling procedure (MUP) as proposed by Bean et al. [3] represents another form of partial rescheduling: Instead of identifying the set of actually affected operations, Bean et al. propose to reschedule all activities executed before a heuristically determined matchup point. If rescheduling is possible with respect to the given constraints, all operations after the matchup point can be executed according to the original schedule - otherwise, the matchup point is postponed or jobs are reassigned to different machines. Akturk and Gorgulu [2] apply this procedure to flow shop scheduling problems where machine breakdowns occur.

Partial rescheduling has mainly been studied for problems with unary and non-exchangeable resources: Particularly the production-specific job-shop scheduling problem has been investigated. More generic problem classes and other practical domains have almost completely been disregarded so far, even though the goal of returning to an already existing schedule as early as possible may be of equal importance. Consider for example the management of supply chains, traffic flows or airport operations: The respective domains are characterized by frequent disruptions, the requirement of fast intervention and high numbers of dependencies, which implies that single actors can not easily alter large portions of the collective schedule. Since triggering modifications is usually associated with penalties or costs, involved participants try to identify quick, minimal and local forms of problem resolution in the process of disruption management.

This paper discusses the complexity of partial rescheduling in scenarios, in which discrete resource capacities and arbitrarily linked activities are important. It introduces and evaluates generalized approaches to Affected Operations Rescheduling and Matchup Scheduling for the generic Resource-Constrained Project Scheduling Problem (RCPSP) and compares them to the strategy of rescheduling all future activities. The remainder of this document is structured as follows: In Section 2 the conceptual framework of the RCPSP is presented before the proposed extensions of the considered rescheduling techniques are described in detail. In Section 3 the results of a thorough analysis and comparison are discussed. Section 4 provides an outlook on future work and summarizes the contributions of this paper.

2. Rescheduling Approaches for the RCPSP

The Resource-Constrained Project Scheduling Problem [4,5] represents a generalization of various forms of production scheduling problems such as the job-shop, the flow-shop and the open-shop problem. It is all about scheduling the activities of a project according to some optimization criterion (the minimization of the project duration, for example), such that all precedence constraints are respected and resource requirements never exceed respective availabilities.

A project in the RCPSP is usually defined by a set of activities $\mathcal{A} = \{0, 1, ..., a, a + 1\}$, where 0 and $a + 1$ denote abstract start and end activities with a duration of 0 and no resource requirements associated. Execution is based on a set of resource types $\mathcal{R} = \{1, ...r\}$: Of each type $k \in \mathcal{R}$, a constant amount of c_k units is available. The following constructs can be used for the description of time and resource dependencies.

- *Duration Value.* For each activity i, a duration d_i describes how long its execution lasts. Note that the RCPSP typically consists of non-preemptive operations.
- *Precedence Constraints.* The order of activities can be defined by precedence constraints: The existence of $p_{i,j}$ in the respective set $\mathcal{P}$ states that activity i has to be finished at or before the start of activity j.
- *Resource Requirements.* The relationships between resource types and activities are defined in the set of resource requirements $\mathcal{Q}$: Activity i requires $q_{i,k} \in \mathcal{Q}$ units of type $k \in \mathcal{R}$ throughout its execution.

The goal of the scheduling process is the identification of a vector of scheduled starting times $(\beta_1, ..., \beta_a)$, for which $\beta_i + d_i \leq \beta_j$, $\forall p_{i,j} \in \mathcal{P}$ and $\sum_{i \in \mathcal{A}_t} q_{i,k} \leq c_k$ for any resource type $k \in \mathcal{R}$ at any time t, where $\mathcal{A}_t$ corresponds to all activities concurrently executed at t. For the subsequent discussion of rescheduling approaches the following additional concepts are required: ρ_i is the planned starting time of activity i, forming the lower bound for β_i. δ_i is the due date associated with an $i \in \mathcal{A}$, defining when an activity shall be considered late. $\mathcal{U} = \{1, ..., u\}$ groups all periods of non-shiftable resource unavailability, which are used to describe fixed resource reservations, for example. Each $l \in \mathcal{U}$ is defined by an amount $q_{l,k}$ required on a resource type k, a starting time β_l and a duration d_l. $\mathcal{U}_k$ denotes the subset of unavailabilities of a single resource type $k \in \mathcal{R}$.

The vast majority of research in the domain of the RCPSP has focused on the development of efficient algorithms for the generation of baseline schedules (i.e. schedules which do not consider another previously generated plan). Although rescheduling has been investigated for the specific problems of machine scheduling, the characteristics of the more general RCPSP formulation have almost completely been disregarded so far: It is particularly the potential existence of resources with capacities greater than one that makes existing techniques like AOR or MUP inapplicable for project scheduling.

Since these approaches of partial rescheduling have proven to be highly efficient in the domain of machine scheduling [1,2], their extension for the RCPSP is proposed in this section: After the summary of the regarded problem and a brief description of full rescheduling, generalized versions of AOR and MUP are presented in detail.

2.1. Regarded Rescheduling Problem

Prior to the discussion of potential rescheduling strategies, let us describe the regarded problem more precisely: In the following, we consider a schedule during the execution of which a disruption occurs. Such a disruption might be of one of the following types, according to the classification scheme proposed by Zhu et al. [19] (cf. [15]): Additional activity, additional precedence constraint, duration modification, requirements modification, resource capacity reduction or milestone modification. Each type has a specific set of parameters associated, the only value all of them have in common is the time of disruption detection t_d. The objective of rescheduling is to identify a schedule which (1) considers the modified circumstances, (2) is as close as possible to the original one and (3) is optimal according to some predefined criterion.

2.2. Full Rescheduling

Full rescheduling is concerned with the generation of a new schedule for *all* activities with future starting points. For this purpose, a sub-problem of the original scheduling problem is generated according to the following rules:

- *Past Activities.* All activities which are already finished when the disruption occurs ($\beta_i + d_i \leq t_d$, $i \in \mathcal{A}$) are ignored. Note that also the precedence relations which make these activities predecessors of future elements can be omitted.
- *Running Activities.* Since the activities running at the time the disruption is detected ($\beta_i \leq t_d < \beta_i + d_i$, $i \in \mathcal{A}$) must not be interrupted, it has to be made sure that they are positioned at their past and actual starting time within the new schedule. For this purpose, it is necessary to set the planned to the actual starting time and to replace all associated resource requirements ($q_{i,k} \in \mathcal{Q}$, $\forall k \in \mathcal{R}$) by corresponding resource unavailabilities: The former imposes a lower bound on the newly assigned schedule time, the latter expresses the fact that resources are actually blocked and can not be reallocated anymore. Since the modified version of the activity requires no resources and is not precedence dependent on any past operation, it will definitively be scheduled at its real starting time.
- *Future Activities.* All activities starting after the detection of the disruption ($t_d < \beta_i$, $i \in \mathcal{A}$) can be considered in their original version.
- *Resources.* All resources required by running or future activities have to be regarded in the rescheduling sub-problem. As regards associated unavailabilities, merely future periods have to be considered.

The respective scheduling problem can be solved by applying the same method as used for the generation of the baseline schedule. It is important to note that full rescheduling does not necessarily mean that scheduling is performed from scratch: Instead, the original (and disrupted) schedule can be regarded as a starting point for incremental optimization based on appropriate heuristics (such as genetic algorithms, for example). This approach implicitly makes sure that closely related solutions are evaluated prior to completely different ones.

2.3. Affected Operations Rescheduling

The concept of Affected Operations Rescheduling is motivated by the idea of preserving as much as possible of the original schedule and maintaining a maximum level of stability [16]. AOR is therefore concerned with the identification of the set of minimal changes, which are necessary to adapt the existing schedule to a new situation.

It has already been mentioned that in problems where resource capacities are restricted to one and where each activity has at most one successor associated, a binary tree can be applied for this purpose [1,14]. However, this approach is not feasible for the more generic RCPSP: On the one hand an activity might have more than one successor, and on the other hand the set of actually affected resource successors can not be identified unambiguously anymore. This is mainly due to the fact that several activities can be executed on one single resource simultaneously. Consider the simple example where an activity a requires 2 units of of a resource r with $c_r = 3$ and is immediately succeeded by the activities b, c and d in the original schedule. If no precedence constraints link the single activities and if each successor of a requires exactly one unit of r (i.e. $q_{i,r} = 1$,

Algorithm 1 GAOR
Input Schedule S with $\mathcal{A}$ and $\mathcal{U}$; modifications $\mathcal{D}$; time of last regarded conflict t_x
Return A set of minimally modified schedules $\mathcal{S}$

1: $S' \leftarrow$ APPLYMODIFICATIONS (S, $\mathcal{D}$)
2: $S' \leftarrow$ MAKEPRECEDENCEFEASIBLE (S')
3: $t_c \leftarrow min(t | \sum_{i \in \mathcal{A}'_t} q_{i,k} > c_k,\ k \in \mathcal{R}, t \in \{\beta_i, i \in \mathcal{A}' \cup \mathcal{U}'\})$
4: **if** t_c is undefined **then**
5: $\quad \mathcal{S} \leftarrow \{S'\}$
6: **else**
7: $\quad$ **if** $t_c \neq t_x$ **then** $\mathcal{A}_c \leftarrow \mathcal{A}'_{t_c} \setminus S_{\mathcal{A}_{mod}}$ **else** $\mathcal{A}_c \leftarrow \mathcal{A}'_{t_c}$
8: $\quad$ **for all** $k \in \mathcal{R} \mid \sum_{i \in \mathcal{A}_{t_c}} q_{i,k} - c_k > 0$ **do**
9: $\quad\quad \mathcal{R}_c \leftarrow \mathcal{R}_c \cup k$
10: $\quad\quad \mathcal{B} \leftarrow \mathcal{B} \otimes$ GETMINIMALSUBSETS ($\mathcal{A}_c$, k, $\sum_{i \in \mathcal{A}_{t_c}} q_{i,k} - c_k$)
11: $\quad \mathcal{B} \leftarrow \{\mathcal{A}_i \in \mathcal{B} \mid \nexists \mathcal{A}_j \subset \mathcal{A}_i : \sum_{l \in \mathcal{A}_j} q_{l,k} \geq \sum_{m \in \mathcal{A}_{t_c}} q_{m,k} - c_k, \forall k \in \mathcal{R}_c\}$
12: $\quad$ **for all** $\mathcal{A} \in \mathcal{B}$ **do**
13: $\quad\quad$ **for all** $i \in \mathcal{A}$ **do**
14: $\quad\quad\quad t_t \leftarrow min(t \mid t > t_c, t \in \{\beta_j + d_j \mid q_{j,k} > 0,\ k \in \mathcal{R}_c,\ j \in \mathcal{U}' \cup \mathcal{A}' \setminus i\})$
15: $\quad\quad\quad \mathcal{D}' \leftarrow \mathcal{D}' \cup$ GETACTIVITYSHIFT $(i, t_t - \beta_i)$
16: $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup$ GAOR $(S', \mathcal{D}', t_c)$
17: **end if**

$i \in \{b, c, d\}$), a delay of a causes the shift of exactly 2 of its successors: either $\{b, c\}$, $\{b, d\}$ or $\{c, d\}$ might be affected by the disruption and each of them represents the basis for a set of minimal schedule modifications.

For the identification of the corresponding set of minimally modified schedules in the context of the RCPSP we propose GAOR as a generalized version of AOR [14]. Even though it is motivated by the same underlying idea, the respective technique is significantly more complex than the existing approaches for the job-shop scheduling problem. Algorithm 1 describes GAOR formally: Given the original schedule S and the modifications $\mathcal{D}$ implied by a disruption, first a potential new schedule S' is generated by incorporating all $\mathcal{D}$ into S. The precedence conflicts potentially associated with the application of the schedule modifications are then resolved by shifting all of the respective activities to the end of their last predecessor. It is then checked, if any resource conflicts remain in the new schedule S': The time of the first conflict t_c corresponds to the minimal t among the starting times of all activities and resource unavailabilities within S' for which requirements exceed availabilities on a resource type $k \in \mathcal{R}$. Note that the consideration of starting times is sufficient since activities and unavailabilities always block a constant amount of resource entities. If no conflict was found, S' represents a feasible modification of S and is returned. Otherwise, the recursive reparation process is started:

- First (line 7) a set of activities to consider $\mathcal{A}_c$ is generated: It usually consists of all activities running at the time of the conflict less the elements that have been modified earlier. Only if t_c equals the previously regarded t_x (i.e. the conflict could not be resolved in the previous step), also these already altered activities shall be considered: This distinction is necessary to make sure that neither one single activity is shifted to the end of the schedule, nor fixed resource unavailabilities can block the reparation process.

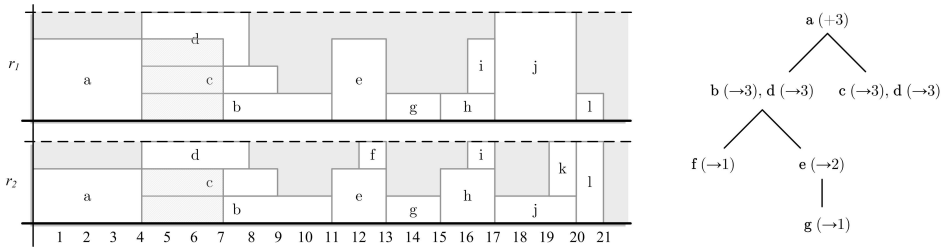

Figure 1. Exemplary application of GAOR: Original Schedule and Search Tree

- The next step (line 8 to 11) is the identification of the minimal sets of activities, the shift of which is sufficient to resolve all resource conflicts at t_c. First, all involved resources are stored in $\mathcal{R}_c$. For each of them, all subsets of $\mathcal{A}_c$ which can cover the excessive requirements and which are minimal (i.e. there exists no subset which is able to set enough units free by itself) are merged into $\mathcal{B}$ – a set of activity sets: The specific $\otimes$-operator, which joins each set $\mathcal{A}_i \in \mathcal{B}_u$ with each set $\mathcal{A}_j \in \mathcal{B}_v$, is used to make sure that at any time the sets in $\mathcal{B}$ do consider *all* of the resource types regarded so far. Since non-minimal activity sets might result from this merge, they are removed from $\mathcal{B}$ in the final step.
- In the last iteration (line 12 to 16) all candidate activity sets are evaluated: For each $\mathcal{A} \in \mathcal{B}$ a set of associated modifications $\mathcal{D}'$ is generated. For this purpose, first the time to shift an activity $i \in \mathcal{A}$ to is identified: t_t is the first point in time after t_c, at which some units of any of the resources causing the conflict are set free (by ending unavailabilities or by other activities). The according modification, which describes the requirement of shifting i by $t_t - \beta_i$ time units, is added to the set $\mathcal{D}'$. Finally, these modifications are evaluated by calling GAOR for the intermediary schedule S', the set of modifications $\mathcal{D}'$ and t_c as the last regarded conflict.

An example for the application of GAOR is depicted in Figure 1: On the left-hand side, the resource requirements associated with an original schedule are illustrated: r_1 and r_2 are required by the activities a to l, which are not linked by any precedence constraints. The dashed area visualizes a pending extension of activity a's duration by 3 time units. Given this disruption, GAOR first tries to resolve the resource conflict at 4 by shifting d and either c or b. In the former case, no further conflicts exist and a minimal set of modifications has been identified. In the latter case, however, resource requirements exceed the availabilities of r_1 at 12 and either f or e has to be shifted. This procedure is continued until feasible schedules have been identified on each branch of the thereby spanned search tree, which is illustrated on the right-hand side: Note that plus stands for the extension of durations and the arrow symbolizes the temporal shift of an activity.

GAOR ends recursion only when the modified schedules are sound: It does never return invalid or infeasible solutions. As regards its completeness, the following theorem holds for problems based on monotonously increasing cost functions (i.e. additional tardiness/earliness or additional modifications always cause additional costs):

Theorem 1. *The solution which is optimal according to some specified cost function is always contained in the set of schedules generated by* GAOR*, if no activity must start earlier than defined in the originally optimal preschedule ($\rho_i = \beta_i, \forall i \in \mathcal{A}$).*

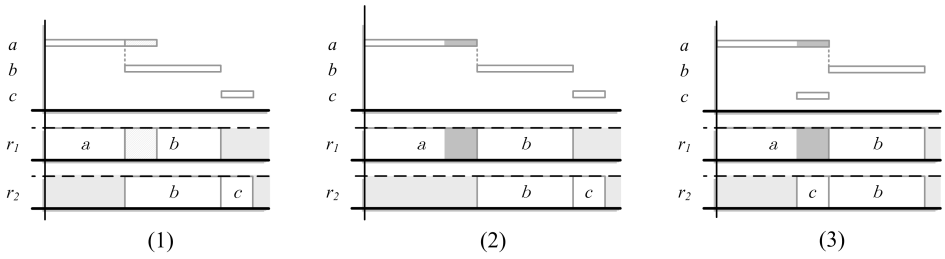

Figure 2. GAOR Suboptimality: Original Situation, GAOR Schedule and Actual Optimum

Proof (Sketch). Since the baseline schedule is assumed to be optimal, any temporal shift of activities is associated with costs. Given a monotonously increasing cost function, the respective value increases with the deviation from the original plan. For this reason, all adaptations have to be minimal in the optimal solution. The constraint that no activity must start earlier than defined in the original schedule makes sure that only the postponement of activities represents a feasible reaction to a disruption.

In case of a shortening of process execution times, the existing schedule has therefore not be modified at all. In case of an extension of process execution times, however, a minimal set of activities has to be shifted to the right-hand side by a minimal amount of time: The content of this set of postponed elements depends on the activity-related earliness/tardiness costs. The proposed generalized version of Affected Operations Rescheduling evaluates *all* existing possibilities for shifting activities to the right-hand side: All subsets of all activities potentially causing a conflict are analyzed. By considering available slack and idle times, it makes sure that only minimal modifications are made to the existing schedule. The algorithm therefore definitively identifies the previously characterized optimal solution. □

The precondition $\rho_i = \beta_i, \forall i \in \mathcal{A}$ implies that only shifts to the right-hand side are considered, which is often a practical and natural approach to efficient rescheduling: In many real world domains the number of processes and dependencies is so high, that the operative managers are not able to consider more complex shifts and reallocations within the short time available for intervention. Take, for example, the management of airport ground processes, which is characterized by short execution times, the involvement of many actors with tight individual schedules as well as the availability of only little time to react to disruptions [13,18]. However, it has to be stated that in such complex scenarios (where already modified schedules have to be rescheduled) the actual optimum might lie outside the set of solutions considered by GAOR. Take the exemplary situation depicted in Figure 2(1), where the upper plane visualizes precedence relations between the activities a, b and c as dashed lines and the two lower planes illustrate the resource usage of types r_1 and r_2: The dashed area shows a pending extension of a's duration. We assume that both b and c may start at the end of activity a at the earliest ($\rho_b = \rho_c = \beta_a + d_a$) and that the aim of optimization is a minimal makespan. If GAOR is applied in this situation, schedule (2) is generated, which is obviously worse than the optimal schedule (3).

To overcome this limitation, we suggest to adapt the strategy of human process managers, who first try to find efficient solutions by shifting activities to the right-hand side before they spend the additionally available time in considering more extensive modifications. GAOR shall therefore particularly be considered in combination with a second

step of full rescheduling, based on incremental heuristic optimization. Since the evaluation of all potentially relevant combinations of schedule modifications soon becomes intractable for realistic problem sizes and short response times, we propose and analyze a hybrid rescheduling strategy, in which only a certain amount of time is spent in the constructive search for an initial set of (good) solutions, based on the generalized version of AOR. The respective schedules represent the starting point for the subsequent step of further optimization, which is based on some form of heuristics: Considering also the remaining solutions, the employed algorithm has to make sure that the theoretical optimum is identified at least within an infinite time horizon.

2.4. Matchup Scheduling

Matchup Scheduling as proposed by Bean et al. is motivated by the idea of identifying a schedule that seeks to match up with an existing preschedule as early as possible[3]. If compared to AOR, the main difference lies in the fact that the point in time until which the disruption shall be compensated – the so-called matchup point – is determined heuristically. Scheduling is performed in a stepwise approach, where in each iteration the regarded time frame is extended until finally the entire future is considered.

For the application of MUP to the Resource-Constrained Project Scheduling Problem, we introduce a generalized version named GMUP. A matchup point t_m is identified by some form of heuristics, the proper choice of which depends mainly on the structure of activities and dependencies in the relevant problems. The further proceeding is similar to the method applied in full rescheduling (see Section 2.2): A sub-problem of the original scheduling problem is generated, starting at t_d and ending at t_m. Its creation is based on the rules discussed above, only modified in the following one:

- *Future Activities.* All activities starting after the detection of the disruption and ending before the matchup point ($t_d < \beta_i, \beta_i + d_i \leq t_m, i \in \mathcal{A}$) can be considered in their original version. Future activities ending after the matchup point ($t_d \leq \beta_i$, $t_m < \beta_i + d_i$, $i \in \mathcal{A}$) are handled similarly to the ones running at t_d: They are planned at their scheduled time, associated resource requirements are eliminated and replaced by unavailabilities: It shall thereby be guaranteed that their starting time is not modified. Moreover, their duration is shortened to $t_m - \beta_i$: This way it is made sure that none of them ends after the matchup point, which makes it possible to distinguish between valid and invalid solutions (see below). All remaining future activities starting only after the matchup point ($t_m < \beta_i$, $i \in \mathcal{A}$) are omitted along with associated precedence constraints and resource requirements.

The sub-schedules resulting from the resolution of this problem are valid if no contained activity ends after the matchup point. In that case, updating the starting times defined in the baseline schedule according to the GMUP timetable provides a feasible way to handle the given irregularity. As regards the quality of the respective solution, however, it is only made sure that disruptions are compensated as early as possible. Even though rescheduling problems can be resolved quickly based on Matchup Scheduling, it has thus to be considered that the first identified schedules might be of bad quality in terms of more complex objective functions. For this reason, we propose to continue the extension of the regarded search space (until finally all future activities are considered) instead of terminating after the identification of the first feasible solution: Only this way the theoretical optimum can be found.

3. Performance Evaluation

This section describes the results of the analysis and comparison of the discussed approaches: After some remarks on the general characteristics of the regarded problem classes and the experimental setup, the generation of testsets, the considered performance indicators and the results of the performed measurements are discussed.

3.1. Characteristics and Setup of the Evaluation

The rescheduling problems used for the evaluation of the proposed approaches are multiple-project tardiness problems [6]: They consist of a baseline schedule and a single disruption of type activity duration extension. Note, however, that any of the irregularities summarized in Section 2.1 could be handled by use of the above methods. The availability of only a limited amount of time for rescheduling is assumed: We are particularly interested in the potential application of the discussed approaches to (near) real-time decision support systems for operative process disruption management (cf. Section 4).

For the evaluation of the proposed strategies, a generic rescheduling engine has been implemented in Java, in which schedule optimization is based on the genetic algorithm proposed by Hartmann [7]. As regards the implementation of GAOR and GMUP, the following annotations can be made:

- According to the above argumentation, GAOR has been implemented and evaluated as a preprocessing technique of full rescheduling: In a heuristic approach, portions of the available time were assigned to the two steps the way that both of them can make significant contribution to the identification of an optimal solution. For the regarded problem sizes, a ratio of one to three turned out to (1) permit the identification of an appropriate number of minimally modified schedules by GAOR and to (2) leave sufficient time for further optimization.
- The methods and parameters applied for the identification of matchup points in GMUP were also determined heuristically, targeting at the maximum effectiveness for the given problem structures (see Section 3.2). In a three-step approach, matchup is first tried at the time where twice the additional resource requirements are covered. The second matchup point is the point at which enough idle time has passed to cover four times the requirements imposed by the disruption. Finally, a third step performs full rescheduling. To make sure that each of these steps can contribute appropriately, one half of the overall available time was reserved for the last step and the other half was split among the two former steps in proportion to the number of activities contained in the respective sub-problems.

3.2. Testset Generation and Problem Classes

Due to the currently existing lack of publicly available instances of reactive scheduling problems [15], a corresponding testset generator has been implemented: Relevant parameters include normalized versions of network complexity, resource factor and resource strength as proposed by Kolisch et al. [11] as well as specific extensions for the description of inter-process relations and exogenous events. Since the structure of the baseline schedule has significant impact on the results of rescheduling [1,9] it can also be parameterized. Accordingly, the following configurations have been used to generate eight different classes of problems:

- *Low/High Process Complexity.* Process complexity defines the density of the activity network in terms of precedence relations. Low complexity means that few precedence relations exist, high complexity that the activities are strongly linked.
- *Low/High Resource Complexity.* The resource complexity parameter combines the aspects of resource requirements and resource availability: Low complexity means that many entities are available to cover few requirements, high complexity means that only small amounts are available to cover high requirements.
- *Tight/Wide Baseline Schedule.* The tightness of the baseline schedule depends on two elements: The amount of incorporated slack time and the distribution of planned activity starting times. In a tight schedule, activities start immediately at the earliest possible time and many processes are executed synchronously. In a wide schedule, some amount of time is available between the end of a predecessor and the start of its successor and only few processes are executed in parallel.

For each thereby defined class, 100 cases were generated by random, forming a total of 800 problem instances. Each case consists of 10 processes (i.e. projects) containing 10 activities, which are executed on up to 3 different kinds of resources[1].

3.3. Used Performance Indicators and Objective Function

As regards performance indicators for rescheduling approaches, particularly effectiveness and schedule stability can be distinguished [10]. The former describes the quality of the applied modifications and can be measured by evaluating the new schedule in terms of costs: Effectiveness corresponds to the relation between the respective value and the theoretically possible optimum. The latter describes the nervousness of the schedule and can be measured by comparing the modified schedule with the original one: Schedule stability corresponds to the portion of activities with unmodified starting times.

For the performed evaluation, we have combined both performance indicators into one single cost function: The sum of all activities' tardiness $\sum_{i \in \mathcal{A}} max(0, \beta_i + d_i - \delta_i)$ is used as the measure of effectiveness. It is related to schedule stability the way that each schedule modification causes three times the costs of one time unit of tardiness.

3.4. Results

For each problem instance, rescheduling was performed 10 times with each of the discussed strategies. The execution time was limited to only 3 seconds, which can be considered a pretty hard setting. Since it is not easily possible to determine the definite optimum for problems of the regarded size (cf. [12]), the respective results were compared to the best value identified within all of the performed runs instead. The figures listed in Table 1 summarize how much of the thereby defined optimization potential could be tapped by the regarded strategies[2]: If, for example, the disrupted schedule causes costs of 10 before full rescheduling, GAOR and GMUP bring this value down to 5, 2 or 3, respectively, the table would show $\frac{10-5}{10-2}$ = 62.5% for the first, $\frac{10-2}{10-2}$ = 100.0% for the second and $\frac{10-3}{10-2}$ = 87.5% for the third approach. The fact that the listed values range from about 75 to 90% illustrates some sort of stability: Since only few outliers exist, it is quite likely that in many of the regarded cases the global optimum could be identified.

[1] The XML description of the instances can be downloaded from http://rcpsp.serverside.at/rescheduling.html

[2] On http://rcpsp.serverside.at/rescheduling.html also the detailed results of this evaluation are provided.

Table 1. Cumulated Performance Values for Full Rescheduling, GAOR and GMUP

	Process Complexity		Resource Complexity		Baseline Schedule		Overall
	low	high	low	high	tight	wide	
Full	82.16%	77.32%	82.00%	77.48%	83.55%	75.93%	79.74%
GAOR	86.04%	85.04%	90.76%	80.32%	86.61%	84.47%	85.54%
GMUP	84.82%	78.06%	84.04%	78.85%	84.32%	78.57%	81.44%

Overall, the solutions identified by the proposed techniques of partial rescheduling are consistently and significantly better than the ones determined by full rescheduling. The best results are achieved by GAOR, which performs particularly well on wide baseline schedules. For the positive effects associated with the different levels of process and resource complexity, mainly two aspects are crucial: On the one hand, higher complexities cause a wider distribution of activities and work the same way as low schedule tightness does. On the other hand, low complexities ensure that more solutions can be analyzed within the available time and that it therefore is likely to identify the optimum already during preprocessing (i.e. the constructive search). Since process complexity has a rather direct influence on the wideness of the baseline schedule, and since particularly resource complexity extends the number of possibilities to consider by GAOR, best performance improvements can be achieved in scenarios with high process and low resource complexity. As regards GMUP, it can be observed that all of the respective performance values are about 1 to 2 % above full rescheduling: Again, significantly better results can be obtained for wide baseline schedules (with large amounts of slack times or high numbers of precedence constraints). A low level of resource complexity has smaller impact since GMUP is less dependent on the respectively defined combinatorial possibilities.

4. Conclusions and Future Work

This paper described how two well-established techniques for partial rescheduling in the domain of machine scheduling can be adapted to and extended for the more generic problem classes of project scheduling. As generalized versions of Affected Operations Rescheduling and Matchup Scheduling, GAOR and GMUP were proposed as approaches to efficient rescheduling in the context of the RCPSP. Their analysis and comparison to the strategy of rescheduling all future activities revealed that they can yield significant performance improvements. Particularly the use of GAOR in domains with sparse and wide baseline schedules can be suggested.

It has therefore been illustrated how it is possible to perform efficient *rescheduling* in case of schedule disruptions: Adaptation and reparation focused on the mere temporal shift of activities. For efficient disruption management in real-world applications, however, also potential *structural modifications* have to be considered when it is possible to extend and shorten durations deliberately, to exchange or modify the planned order of operations or to parallelize or serialize activity execution. The only form in which at least some of the respective flexibility can be modeled in the domain of project scheduling is provided by the Multi-Mode RCPSP (MRCPSP, see [8]) which allows the alternation of activity execution modes. We have therefore proposed the *x-RCPSP* as a conceptual extension of the RCPSP, targeting at the support of more comprehensive forms of disruption management [13]. Future work will be directed at the adaptation of the techniques proposed in this paper for such an extended framework.

Acknowledgements

This work is part of the *cdm@airports* project, which is carried out in cooperation with FREQUENTIS GmbH (Austria) and is partly funded by grants from FFF (Austria).

References

[1] R.J. Abumaizar and J.A. Svestka, Rescheduling job shops under random disruptions, *International Journal of Production Research* **35** (1997), 2065–2082.
[2] M.S. Akturk and E. Gorgulu, Match-up scheduling under a machine breakdown, *European Journal of Operational Research* **112** (1999), 81–97.
[3] J.C. Bean, J.R. Birge, J. Mittenthal and C.E. Noon, Matchup Scheduling with Multiple Resources, Release Dates and Disruptions, *Operations Research* **39** (1991), 470–483.
[4] J. Błazewicz, J.K. Lenstra and A.H.G. Rinnooy Kan, Scheduling Projects to Resource Constraints: Classification and Complexity, *Discrete Applied Mathematics* **5** (1983), 11–24.
[5] P. Brucker, A. Drexl, R. Möhring, K. Neumann and E. Pesch, Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research* **112** (1999), 3–41.
[6] E.L. Demeulemeester, W.S. Herroelen, *Project Scheduling: A Research Handbook*, Kluwer Academic Publishers, Boston, 2002.
[7] S. Hartmann, A competitive genetic algorithm for resource-constrained project scheduling, *Naval Research Logistics* **45** (1998), 733–750.
[8] S. Hartmann, Project Scheduling with Multiple Modes: A Genetic Algorithm, *Annals of Operations Research* **102** (2001), 111–135.
[9] W. Herroelen and R. Leus, Robust and reactive project scheduling: a review and classification of procedures, *International Journal of Production Research* **42** (2004), 1599–1620.
[10] G.Q. Huang, J.S.K. Lau, K.L. Mak, L. Liang, Distributed supply-chain project rescheduling: part II - distributed affected operations rescheduling algorithm, *International Journal of Production Research* **44** (2006), 1–25.
[11] R. Kolisch, A. Sprecher, A. Drexl, Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems, *Management Science* **41** (1995), 1693–1703.
[12] R. Kolisch, A. Sprecher, PSPLIB - A project scheduling library, *European Journal of Operational Research* **96** (1996), 205–216.
[13] J. Kuster, D. Jannach, Extending the Resource-Constrained Project Scheduling Problem for Disruption Management, *IEEE Conference On Intelligent Systems* (2006), to appear.
[14] R.K. Li, Y.T. Shyu and A. Sadashiv, A heuristic rescheduling algorithm for computer-based production scheduling systems, *International Journal of Production Research* **31** (1993), 1815–1826.
[15] N. Policella and R. Rasconi, Testsets Generation for Reactive Scheduling, *Workshop on Experimental Analysis and Benchmarks for AI Algorithms* (2005).
[16] G.E. Vieira, J.W. Herrmann and E. Lin, Rescheduling manufacturing systems: a framework of strategies, policies, and methods, *Journal of Scheduling* **6** (2003), 39–62.
[17] S. van de Vonder, E. Demeulemeester and W. Herroelen, An investigation of efficient and effective predictive-reactive project scheduling procedures, *Journal of Scheduling* (2006).
[18] C.L. Wu, R.E. Caves, Modelling and Optimization of Aircraft Turnaround Time at an airport, *Transportation Planning & Technology* **27** (2004), 47–66.
[19] G. Zhu, J.F. Bard and G. Yu, Disruption management for resource-constrained project scheduling, *Journal of the Operational Research Society* **56** (2005), 365–381.