Proactive and Reactive Strategies for Resource-Constrained Project Scheduling with Uncertain Resource Availabilities*

Olivier Lambrechts Erik Demeulemeester

Willy Herroelen†

Department of Decision Sciences and Information Management

Research Center for Operations Management

Faculty of Economics and Applied Economics

Katholieke Universiteit Leuven (Belgium)

Abstract

Research concerning project planning under uncertainty has primarily focused on the stochastic resource-constrained project scheduling problem (stochastic RCPSP), an extension of the basic RCPSP, in which the assumption of deterministic activity durations is dropped. In this paper, we introduce a new variant of the RCPSP for which the uncertainty is modeled by means of resource availabilities that are subject to unforeseen breakdowns. Our objective is to build a robust schedule that meets the project deadline and minimizes the schedule instability cost, defined as the expected weighted sum of the absolute deviations between the planned and the actually realized activity starting times during project execution. We describe how stochastic resource breakdowns can be modeled, which reaction is recommended when a resource infeasibility occurs due to a breakdown and how one can protect the initial schedule from the adverse effects of potential breakdowns. An extensive computational experiment

*This research has been supported by project OT/03/14 of the Research Fund of K.U.Leuven, project G.0109.04 of the Research Programme of the Fund for Scientific Research - Flanders (Belgium) (F.W.O.-Vlaanderen) and project NB/06/06 of the National Bank of Belgium.

†Corresponding author. Tel: +32-16-326970; fax: +32-16-326732; e-mail: *willy.herroelen@econ.kuleuven.be*

is used to show the relative performance of the proposed proactive and reactive strategies. It is shown that protection of the baseline schedule coupled with intelligent schedule recovery yields significant performance gains over the use of deterministic scheduling approaches in a stochastic setting.

1 Introduction

Most of the research in project scheduling deals with the generation of an *initial project schedule* (*baseline schedule*) in a static and deterministic environment with complete information. Traditional objective functions include minimizing the project makespan, leveling the resource usage over time, minimizing the total cost of acquiring the necessary resources, maximizing the project net present value and minimizing weighted earliness-tardiness penalty costs. For an extensive overview we refer to Brucker et al. (1999), Herroelen et al. (1998) and Demeulemeester and Herroelen (2002).

In practice, however, these assumptions will hardly, if ever, be satisfied. As Aytug et al. (2005) indicate, it is often assumed that 'a system that works in a deterministic environment can be engineered to work under at least certain stochastic conditions'. Whereas for some problems this will indeed be the case (e.g. Pinedo (1995) shows that the WSPT rule minimizes the average weighted flow time for the single machine problem in the deterministic case and likewise the WSEPT rule minimizes the expected average weighted flow time in the stochastic case in the class of nonpreemptive static list policies and nonpreemptive dynamic policies), for others it will not. For an overview of stochastic scheduling we refer the interested reader to Scholl (2001) and Nikulin (2004).

We have to protect the initial baseline schedule from the adverse effects of possible disruptions. This protection is necessary because often project activities are subcontracted or executed by resources that are not exclusively reserved for the current project. A change in the starting times of such activities could lead to infeasibilities at the organizational level (in a multi-project context) or penalties in the form of higher subcontracting costs. A possible measure for the deviation between the initial schedule and the realized schedule is the *weighted instability cost*. It can be defined as the expected weighted absolute deviation between the planned and the actually realized activity starting times. The weight w_i, assigned to each activity i, reflects that activity's importance of starting it at its planned starting time in the initial schedule. More specifically,

w_i denotes the marginal cost of deviating from the planned starting time of activity i during project execution. This marginal cost can be seen as an extra cost for having subcontractors start later than originally agreed or as an inventory cost for storing raw materials between the delivery time and the starting time of the activity. We assume that activities are never started before their planned starting times s_i ('railroad scheduling') and that the total deviation cost is a linear function of the absolute deviation. We assume there are no interaction effects between individual deviation costs.

Minimizing instability then means that we are looking for a schedule which is able to accommodate disruptions without too much change in activity starting times, i.e. a *robust schedule* that satisfies the precedence and resource constraints and does not exceed the deadline set by the project's client. Meeting this deadline during project execution is encouraged by giving a higher instability weight to the activity that signals the end of the project. Recent research by Leus (2003) and Van de Vonder et al. (2005a) considers this objective function for the case of project scheduling with stochastic activity durations. Other possible causes for uncertainty in project execution might be, amongst others, inaccurate time estimates, bad weather conditions or unavailability of resources. In this paper we study the last of these possible causes.

In machine scheduling, the problem of machines randomly breaking down has been reasonably well studied for the single machine (Mehta and Uzsoy, 1999) and the job shop case (Mehta and Uzsoy, 1998). However, except for Drezet (2005) we are not aware of any existing research in project scheduling dealing with the stochastic resource availability case. Drezet (2005) considers the problem of project planning with human resource constraints such as competences, a limit on the number of hours an employee works per day, vacation periods and unavailability of employees. A mathematical model as well as several algorithms are presented for building a robust schedule and for repairing a disrupted schedule.

2 Problem Statement

Aytug et al. (2005) stress the importance of taking potential disruptions into account when building and executing production schedules. The authors distinguish between predictive and reactive scheduling. Predictive (proactive) scheduling approaches try to accommodate uncertainties in advance whereas

reactive approaches react after the fact.

Purely reactive project scheduling forgoes the construction of a baseline schedule and solely relies on the use of *scheduling policies* (Stork, 2001) to decide on-line which activities are to be started at random decision points t that occur serially through time. These random decision points correspond with the completion times of the activities and the decision to start a precedence and resource feasible set of activities at time t can only be based on the information that has become available up to that time (non-anticipativity assumption).

Contrary to scheduling policies, *proactive scheduling* is based on the construction of a *baseline schedule*. This baseline schedule will guide schedule execution by providing for each activity its planned periods of execution as well as the resource units to be reserved during these execution periods. A baseline schedule is indispensable to coordinate resource allocation between multiple projects in a multi-project environment and to coordinate outsourced activities with subcontractors. The arguments Aytug et al. (2005) use to underline the importance of developing a production schedule in machine scheduling also apply to project scheduling. Some of the motivations they cite are: verifying the feasibility of executing the given tasks within a certain timeframe, providing visibility of future actions for internal and external parties, offering degrees of freedom for reactive scheduling, evaluating performance and avoiding further problems.

The aim of proactive scheduling is to build a *robust baseline schedule*: a schedule that is protected as much as possible against disruptions during schedule execution. Of course, it is still possible that the robust baseline schedule, despite the built-in protection, becomes infeasible during project execution (Davenport and Beck, 2002) due to the occurrence of one or more resource breakdowns and the resulting resource shortage. Therefore, we need a *reactive policy* $\mathcal{R}$ that dictates how to revert to a feasible schedule that deviates as little as possible from the original baseline. *Proactive-reactive project scheduling* thus implies a combination of a proactive strategy for generating a protected baseline schedule S with a reactive strategy $\mathcal{R}$ to resolve the schedule infeasibilities caused by the disturbances that occur during schedule execution.

The proactive baseline scheduling problem is an extension of the classical deterministic *resource-constrained project scheduling problem* (RCPSP or problem $m, 1|cpm|C_{max}$ in the notation of Herroelen et al. (2000)) for which a conceptual

formulation can be written as follows:

$$\text{minimize } s_n \tag{2.1}$$

subject to

$$s_i + d_i \leqslant s_j \qquad \forall (i,j) \in A \tag{2.2}$$

$$\sum_{i:i\in\mathcal{S}_t} r_{ik} \leqslant a_k \qquad \forall t, \forall k \tag{2.3}$$

A project is commonly represented using the activity-on-node representation by means of a digraph $G = (N, A)$ where the set of nodes N represents the activities and the set of directed arcs A the finish-start, zero-lag precedence relations. When $(i, j) \in A$ we say that activity i (i : 1 to n) is an immediate predecessor of activity j, implying that activity j cannot start before activity i has finished. Precedence feasibility is enforced by constraints (2.2) where d_i is the deterministic duration of activity i and s_i the planned starting time of activity i. Constraints (2.3) enforce the renewable resource constraints. When executing the project, a deterministic amount a_k is available of each resource type k (k : 1 to K). A feasible baseline schedule will always have to respect this maximal resource availability. The constraints imply that there does not exist a time period t and a resource type k for which the cumulative resource requirements of the activities that are in progress during period t exceed the deterministic per-period availability a_k for the considered resource type. Here r_{ik} denotes the number of units of renewable resource type k required per period by activity i and $\mathcal{S}_t$ is the set of activities that are in progress at time period t. The objective (2.1) of the deterministic RCPSP is to minimize the project duration s_n.

The main difference with the model dealt with in this paper is that we now incorporate uncertainty by dropping the assumption of deterministic resource availabilities. Each of the a_k units of resource type k, allocated at the start of the project, is subject to wear and tear and can therefore break down during execution, resulting in a real availability of resource type k at time point t of $\mathbf{a}_{kt}$. Furthermore, in order not to deviate too much from the initial baseline schedule, the objective function becomes:

$$\text{minimize } \sum_{i\in N} w_i |E(\mathbf{s}_i) - s_i| \tag{2.4}$$

The decision variables s_i represent the planned starting times for each activity

i and have to respect the sets of constraints (2.2) and (2.3). Together, they define the baseline schedule which is represented by the vector $S = (s_1, s_2, \ldots, s_n)$. Because of the stochastic nature of the problem we cannot always stick to this baseline schedule. The real starting times are consequently stochastic variables that are represented by the stochastic vector $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n)$ and that depend on the realization of the stochastic resource availabilities ($\mathbf{a}_{kt}$), on the planned starting times (s_i) and on the reactive policy $\mathcal{R}$ that is used to repair a disrupted schedule. We assume that a 'railroad scheduling' approach is used. This means that activities are never started before their planned starting time ($\mathbf{s}_i \geqslant s_i$), implying that the objective function can be rewritten as $\sum_{i \in N} w_i(E(\mathbf{s}_i) - s_i)$. Finally, we add a deadline constraint:

$$s_n \leqslant \delta \tag{2.5}$$

to prevent the construction of an unreasonably long protected baseline schedule.

Using the classification scheme of Herroelen et al. (2000), our problem can be classified as $m, 1, \mathbf{va}|cpm, \delta| \sum w_i(E(\mathbf{s}_i) - s_i)$. The first field specifies the resource characteristics: $(m, 1, \mathbf{va})$ refers to an arbitrary number of renewable resource types, each with a stochastic availability that varies over time. The second field indicates the use of finish-start, zero-lag precedence relationships and a deterministic project deadline. Finally, the last field shows the objective function, here the expected weighted instability cost.

The deterministic resource-constrained project scheduling problem is known to be strongly NP-hard. Allowing for stochastic resource availabilities complicates the problem. Moreover, the analytic evaluation of the objective function (2.4) is very cumbersome, so that one usually relies on simulation. For NP-hardness proofs of several cases of the scheduling problem for stability subject to a deadline and discrete disturbance scenario, we refer to Leus and Herroelen (2005).

We introduce the example network in figure 1 to illustrate the various proactive and reactive strategies we present in this paper. This graph represents a project consisting of 10 activities. Above each activity node, we indicate its planned duration, its resource requirement of a single renewable resource type with a deterministic per period availability of 8 units (each subject to breakdowns) and its instability weight. Note that activities 1 and 10 are dummy activities with a duration and a resource usage of 0. Activity 1 indicates the start of the project whereas activity 10 signals the end. The instability weight

for activity 10 is much larger than the other instability weights in order to reflect the fact that in practice meeting the project deadline is often deemed more important than meeting planned activity starting times. In this example we assume a project deadline of 18. The baseline starting time of the dummy start activity is then set to the release date of the project (time period 0) whereas the dummy end activity is assumed to end at the project deadline. Note that for ease of notation and illustration only one resource type is considered, but the examples as well as the algorithms presented in this paper are easily extensible to and will be tested for the multi-resource case.

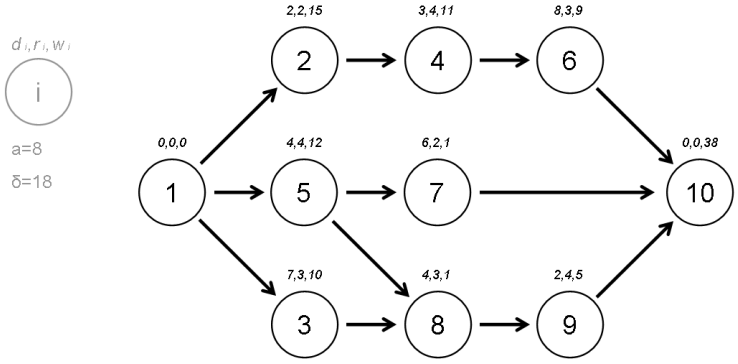

Figure 1: Example project network

The paper is structured as follows. In section 3 the proactive strategies will be treated in detail. The reactive policies are then presented in section 4, where we introduce two list scheduling policies and a tabu search procedure. Section 5 reports on extensive computational results obtained by testing the proactive-reactive procedures on a set of randomly generated test instances. Finally, section 6 presents our overall conclusions.

3 Proactive Strategies

We propose an approach for proactive scheduling in which three choices need to be made. Each choice consists of two options, giving us a total of 2^3 different strategies as can be seen in the decision tree depicted in figure 2. First of all, the schedule can be built using the optimal solution for the RCPSP as a starting schedule or alternatively using a schedule in which activities that have a high impact on instability are scheduled as early as possible so that the

probability that they get disrupted is lower. Secondly, it needs to be decided whether to allow for resource slack or instead use the deterministic, maximum availabilities. Allowing for resource slack means that one plans the project considering a resource availability that is lower than the amount a_k that is allocated at the start of the project, so that a certain margin exists for absorbing resource breakdowns. Finally, it is either possible to protect individual activity starting times by inserting a time buffer of one or more time units in front of them or alternatively not to explicitly buffer activities at all.

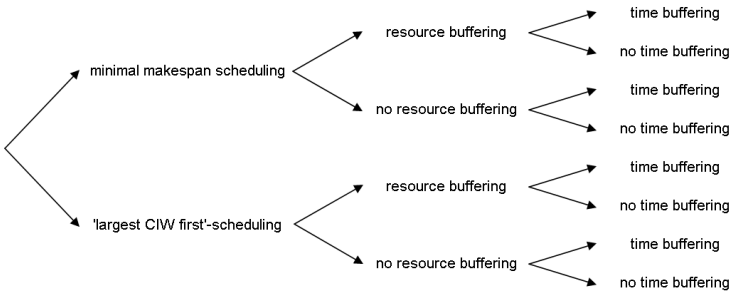

Figure 2: Proactive strategies

3.1 Optimal solution for the RCPSP

The problem under study is an extension of the RCPSP. Since the objective of the deterministic RCPSP is to minimize the project makespan, the associated schedule will usually be very dense. This means that activities are scheduled compactly with as little resource and time slack as possible. In such a schedule even a minor disruption in the resource availabilities during a scheduling period will have a major impact on the starting times of all activities that are scheduled in subsequent periods. Therefore, it can be expected that such a schedule will perform very badly for the weighted instability cost objective. The optimal solution for the RCPSP associated with the project instance of figure 1 is given in figure 3. As expected, little free slack exists in this schedule. Free slack has been proposed by Al-Fawzan and Haouari (2005) as a metric for measuring the robustness of a schedule. We can use it here as an ex-ante surrogate measure for giving an indication of how well the schedule can be expected to perform for the weighted instability objective function (Lambrechts et al., 2006). The free slack of an activity is defined as the amount of time the activity can be delayed beyond its planned starting time without forcing any other activities in the schedule to

be postponed. In contrast to the traditional free float metric, this measure does not only take precedence but also resource constraints into account. In our example the total free slack is equal to 6. Because of the resource constraints, no activity can be delayed without compromising the execution of subsequent activities except for activities 6 and 9. Both activities can be delayed with at most 3 time units while respecting the project deadline ($\delta = 18$). This yields a total free slack of 6.

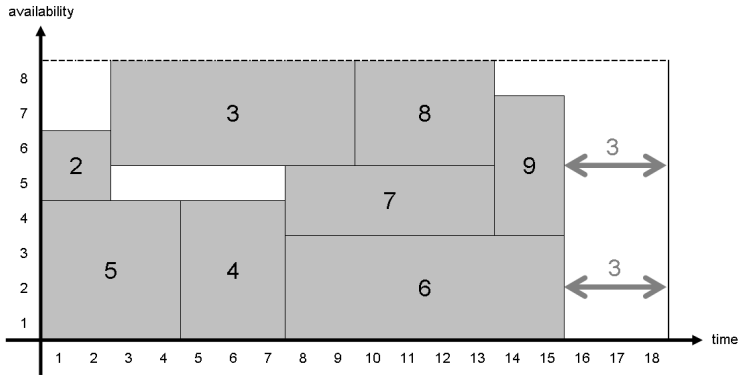

Figure 3: Minimal makespan schedule

3.2 Highest cumulative instability weight first

One way to improve schedule robustness is to schedule the activities in decreasing order of their cumulative instability weight. We define the *cumulative instability weight* of activity i as follows:

$$CIW_i = w_i + \sum_{j:j \in SUCC_i^*} w_j \tag{3.1}$$

with $SUCC_i^*$ the set of direct and indirect successors of activity i. Because disruptions propagate throughout the schedule, activities for which a change in starting time would have a high impact on instability are now less likely to get severely disrupted than activities with a lower impact since the former are scheduled earlier in time and are thus less prone to disruptions. The schedule is constructed in two phases. In the first phase, a precedence feasible priority list is constructed with the activities in non-increasing order of their CIW_i (tie-breaker is lowest activity number) so that no activity enters the list in front of one of its predecessors. In the second phase, this priority list is transformed into

a precedence and resource feasible schedule using the serial schedule generation scheme that was first introduced by Kelley (1963). The *serial schedule generation scheme* sequentially adds activities to the schedule until a feasible complete schedule is obtained. In each step, the next activity in the priority list is selected and for that activity the first precedence and resource feasible starting time is chosen. If we apply this heuristic to our example network, we obtain the vector of cumulative instability weights: $CIW = (102, 73, 54, 58, 57, 47, 39, 44, 43, 38)$. This vector corresponds to a priority list $L = (1, 2, 4, 5, 3, 6, 8, 9, 7, 10)$ that yields the schedule depicted in figure 4 when decoded using the serial schedule generation scheme. As expected, this schedule has a higher total free slack than the minimal makespan schedule (13 versus only 6): the free slack amounts to 5 time units for activity 6, 7 time units for activity 7 and 1 time unit for activity 9.

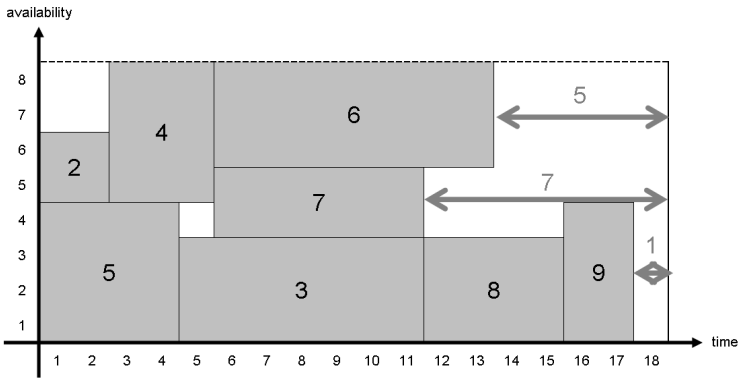

Figure 4: 'Highest CIW first' schedule

3.3 Protection by means of resource slack

The number of renewable resource units of type k actually available in period t is a random variable $\mathbf{a}_{kt}$ instead of a deterministic value. That means that each of the a_k resource units originally allocated to the project is subject to breakdowns characterized by a certain failure time and repair time distribution. Baseline schedules can be protected against disruptions by including resource slack. This means that the project is planned using a resource availability a_k^* that is lower than the maximum, deterministic availability a_k. In this case, a breakdown of one or more resource units will not always lead to a disruption of the schedule. This principle is inspired by the well-known result from factory physics that the lead time increases in a strongly non-linear fashion with increasing resource

utilization and that therefore excess capacity is important (Hopp and Spearman, 2001). The required size of the resource buffer will depend on the probability distribution of the resource availabilities. This probability distribution can be obtained using the theory of birth-death processes or renewal theory. We opted for the last approach as this allows us to avoid the assumption of exponentially distributed failure and repair times. It can be shown (Ross, 1983) that a single resource unit of resource type k with independent and identically distributed failure times $\mathbf{X}_k$ and independent and identically distributed repair times $\mathbf{Y}_k$ has a stationary availability (the probability that the resource is active at a time in the future):

$$A_k = \frac{E(\mathbf{X}_k)}{E(\mathbf{X}_k) + E(\mathbf{Y}_k)} \tag{3.2}$$

As is commonly done in literature dealing with machine breakdowns, we write $E(\mathbf{X}_k) = \frac{1}{\lambda_k} = MTTF_k$ (*mean time to failure*), $E(\mathbf{Y}_k) = \frac{1}{\mu_k} = MTTR_k$ (*mean time to repair*) and $\rho_k = \frac{MTTR_k}{MTTF_k}$ so that:

$$A_k = \frac{1}{1 + \rho_k} \tag{3.3}$$

We can now write:

$$Pr(\mathbf{a}_k = j) = \binom{a_k}{j} {A_k}^j (1 - A_k)^{a_k - j} = \binom{a_k}{j} \frac{\rho_k^{a_k - j}}{(1 + \rho_k)^{a_k}} \tag{3.4}$$

Since we now know the discrete probability distribution of these steady state availabilities, we can determine the expected value (taking breakdowns into account) of the resource availability in the steady state for resource type k as $E(\mathbf{a}_k) = \lfloor \sum_{m=0}^{a_k} mPr(\mathbf{a}_k = m) \rfloor$ and use it as the buffered availability a_k^*. In case this buffered availability is smaller than $\max_{i \in N} r_{ik}$, we increase it until the activity with the highest resource demand for resource type k can be executed. The schedule is then built using the exact RCPSP method or the 'highest CIW first'-method, but now with these adapted, buffered availabilities. Note, however, that it is possible that the obtained resource buffered schedule exceeds the deadline. Therefore, we need to add a mechanism that limits the maximal amount of resource buffering so that the deadline constraint (2.5) is not violated. If the resource buffered schedule turns out to be deadline-infeasible, we determine the most constraining resource type, and progressively increase its availability up to the maximum (original) availability and re-execute the RCPSP or 'highest

CIW first' procedure until the deadline is met. We define the most constraining resource type as the resource type that leads to the highest decrease in schedule makespan when its buffered availability is increased by one unit. As a tie-breaker we choose the resource type with the smallest deviation between the expected resource availability and the adjusted buffered availability.

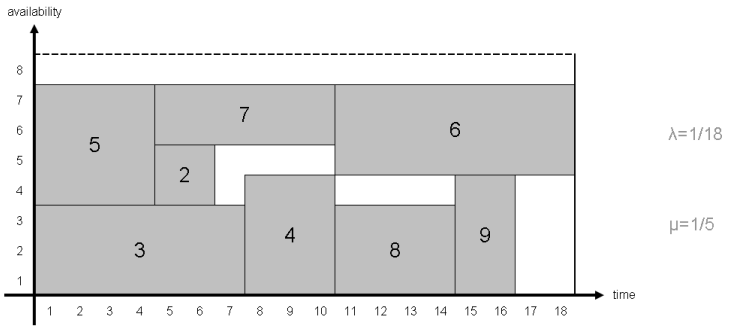

Figure 5: Resource buffering applied to a minimal makespan schedule

In our example, adding resource buffering to the minimal makespan schedule would yield the schedule depicted in figure 5. For this project using a $MTTF$ of 18 and a $MTTR$ of 5, the average availability is equal to $\lfloor 6.26 \rfloor = 6$. Clearly, it is impossible to construct a feasible schedule for this availability since $\sum_i (d_i r_i) > a_k^* \delta$. Increasing a_k^* with one unit and calculating the minimal makespan schedule for $a_k^* = 7$ gives the schedule in figure 5.

3.4 Time buffering

Instead of or in addition to resource buffering, another form of schedule protection can be used. Time buffering boils down to the inclusion of slack time in front of activities in order to absorb potential disruptions caused by earlier resource breakdowns and the resulting activity shifts. We start from a feasible baseline schedule to which protection is added by iteratively right-shifting activities with the aim of protecting the activity starting times as well as possible. Our objective is to insert a time buffer of size b_i in front of the starting times s_i of each activity i so that the expected instability is minimized while not exceeding the deadline.

In order to set correct buffer sizes b_i for each activity i, we need to have a rough idea of the impact of the disruption (I_i) we can expect for that activity

given a certain baseline schedule S. Disruptions, forcing the activity to start at a later point in time than originally planned, can occur amongst others when one or more of the activity's predecessors finish at a later time than expected because of resource breakdowns. First of all, we will describe how the duration increase ($\mathbf{\Delta}_i$) of an activity due to resource breakdowns can be approximated. We will then show how we can use these expected duration increases to determine which activities in the schedule will be affected. Finally, we will introduce a heuristic that selects the activities to be buffered and determines the proper buffer size, based on the expected impact of resource breakdowns on the activities constituting the project.

In a preempt-repeat environment, it is assumed that whenever an activity is disrupted because of a breakdown of one of the resources it is being executed on, it has to be restarted from scratch after repair of the affected resource. This means that in reality the time it will take to complete activity i is not d_i but $d_i + \mathbf{\Delta}_i$. Part of the duration extension of i consists of repair time, the other part represents the subsequent re-executions due to the preempt-repeat setting. We will now show how $E(\mathbf{\Delta}_i)$ can be calculated using three assumptions. First of all, we assume that the times to failure ($\mathbf{X}_k$) and the repair times ($\mathbf{Y}_k$) for each resource unit of each resource type k are exponentially distributed with respective parameters $\lambda_k = \frac{1}{MTTF_k}$ and $\mu_k = \frac{1}{MTTR_k}$. The assumption of exponentially distributed interfailure times can be motivated as follows: resources, whether they are humans, complex machinery or tools, can fail for a wide variety of reasons. We can therefore consider each resource unit to be composed of different components, each associated with a possible failure cause, with different times to failure. Let $\mathbf{N}(t)$ be the total amount of breakdowns up to time t, split up by components 1 to m as follows: $\mathbf{N}(t) = \mathbf{N}_1(t) + \mathbf{N}_2(t) + ... + \mathbf{N}_m(t)$. If m is large enough and the times between counts for each breakdown cause are independent and identically distributed stochastic variables, then the resulting counting process $\mathbf{N}(t)$ will follow a Poisson distribution (Hopp and Spearman, 2001). Because a Poisson counting process corresponds to an exponential distribution of interarrival times (Girault, 1959), the times between failures will be exponentially distributed. Unfortunately, this reasoning cannot be so easily applied to the repair times. However, it is analytically interesting but also practically acceptable to assume that these times are also exponentially distributed. The second assumption is that of fixed resource allocations. This means that in case a resource unit used by activity i breaks down, activity i will be preempted and repeated after the repair of this resource unit. This implies that it is not

possible to prevent the preemption of i by assigning it to other resource units. Finally, resource units are only subject to wear and tear when they are in use. Therefore, it is only possible for an active unit to break down.

Let us consider resource type k. The duration increase of activity i due to breakdowns on this resource type can be split up into a part that corresponds to aggregated repair times and a part that corresponds to aggregated re-execution times:

$$\boldsymbol{\Delta}_{ik} = \boldsymbol{\Delta}_{ik}^{repair} + \boldsymbol{\Delta}_{ik}^{repeat} \tag{3.5}$$

Both $\boldsymbol{\Delta}_{ik}^{repair}$ and $\boldsymbol{\Delta}_{ik}^{repeat}$ will depend of the number of times activity i is preempted. Let the stochastic variable $\boldsymbol{\mathcal{B}}_{ik}$ represent the number of interruptions experienced by activity i due to breakdowns of resource type k before completion. During each execution of activity i on resource type k the activity will either be completed or preempted. Let π_{ik} be the probability that activity i is preempted due to breakdown of k:

$$\begin{aligned}
\pi_{ik} &= 1 - [Pr(\mathbf{X}_k > d_i)]^{r_{ik}} && (3.6)\\
&= 1 - [\int_{d_i}^{\infty} \lambda_k e^{-\lambda_k x} dx]^{r_{ik}} && (3.7)\\
&= 1 - e^{-\lambda_k d_i r_{ik}} && (3.8)\\
& && (3.9)
\end{aligned}$$

This allows us to write:

$$Pr(\boldsymbol{\mathcal{B}}_{ik} = \mathcal{B}_{ik}) = (1 - \pi_{ik})\pi_{ik}^{\mathcal{B}_{ik}} \tag{3.10}$$

We can now write:

$$\begin{aligned}
\boldsymbol{\Delta}_{ik} &= \boldsymbol{\Delta}_{ik}^{repair} + \boldsymbol{\Delta}_{ik}^{repeat} && (3.11)\\
&= \sum_{\mathcal{B}_{ik}=0}^{\infty} [E(\boldsymbol{\Delta}_{ik}^{repair} | \boldsymbol{\mathcal{B}}_{ik} = \mathcal{B}_{ik}) + E(\boldsymbol{\Delta}_{ik}^{repeat} | \boldsymbol{\mathcal{B}}_{ik} = \mathcal{B}_{ik})] && (3.12)
\end{aligned}$$

For $\boldsymbol{\mathcal{B}}_{ik} = \mathcal{B}_{ik}$ we have $\mathcal{B}_{ik}$ repairs and $\mathcal{B}_{ik}$ re-executions. The expected duration of a repair is equal to the expected repair duration for a single resource unit (μ_k) since the activity breaks down as soon as one resource unit breaks down and resource units can only break down during the execution of an activity. The expected duration of a re-execution corresponds to the smallest time to failure over all resource units used by activity i. The minimum of r_{ik} exponentially dis-

tributed random variables with parameter λ_k can be shown to be exponentially distributed with parameter $r_{ik}\lambda_k$. Therefore we can write:

$$\begin{aligned}
\boldsymbol{\Delta}_{ik} &= \sum_{\mathcal{B}_{ik}=0}^{\infty} (1-\pi_{ik})\pi_{ik}^{\mathcal{B}_{ik}}(\mathcal{B}_{ik}E[\mathbf{Y}_k] + \mathcal{B}_{ik}E[min(\mathbf{X}_{k1},...,\mathbf{X}_{kr_{ik}})]) && (3.13)\\
&= \sum_{\mathcal{B}_{ik}=0}^{\infty} (1-\pi_{ik})\pi_{ik}^{\mathcal{B}_{ik}}\mathcal{B}_{ik}(\frac{1}{\mu_k} + \frac{1}{r_{ik}\lambda_k}) && (3.14)\\
&= (\frac{1}{\mu_k} + \frac{1}{r_{ik}\lambda_k})\frac{\pi_{ik}}{1-\pi_{ik}} && (3.15)
\end{aligned}$$

Let us now assume that:

$$\boldsymbol{\Delta}_i = \lceil max_k \boldsymbol{\Delta}_{ik} \rceil \tag{3.16}$$

Of course this is a simplification of reality, but taking the interaction effects between resource types into account would greatly complicate the results because of the different parameters of the exponential distributions.

The expected duration increases can now be used to approximate the impact of resource breakdowns on the given schedule S. For each non-dummy activity i we determine its direct and transitive predecessors $j \in PRED_i^*$. In case the predicted finish time of the considered predecessor $(s_j + d_j + E(\boldsymbol{\Delta}_j))$ exceeds s_i, the planned starting time of activity i, it can be expected that there is a reasonable chance that activity i will be disrupted. The impact I_i on the objective function that can be attributed to the disruption of activity i caused by its predecessors j can be estimated as:

$$I_i = \sum_{j \in PRED_i^*} w_i max\{0, s_j + d_j + E(\boldsymbol{\Delta}_j) - s_i\} \tag{3.17}$$

The non-dummy project activities are placed in a list Q in non-increasing impact order with the lowest activity number as a tie-breaker. The first activity in list Q is selected and right-shifted with one time unit. Affected activities are likewise right-shifted with one time unit in order to keep the schedule precedence and resource feasible. In case the resulting schedule also respects the deadline constraint, we can move to the next iteration by recalculating the expected impacts for each activity for the new schedule S' and building a new list Q'. In case the new schedule is not deadline feasible, we revert the move and select the next activity in Q; if no such activity can be found, the procedure is terminated. The pseudocode for this pseudo-polynomial approach is given in algorithm 1.

Algorithm 1 Time buffering heuristic

1: **for** $i := 2$ to $n-1$ **do**
2: $\quad b_i := 0$
3: $\quad E(\mathbf{\Delta}_j) := \lceil max_k[(\frac{1}{\mu_k} + \frac{1}{r_{ik}\lambda_k})\frac{\pi_{ik}}{1-\pi_{ik}}]\rceil$
4: **end for**
5: **for** $i := 2$ to $n-1$ **do**
6: $\quad I_i := \sum_{j \in PRED_i^*} w_i max\{0, s_j + d_j + E(\mathbf{\Delta}_j) - s_i\}$
7: **end for**
8: $Q := N \setminus \{1, n\}$
9: sort Q in non-increasing order of I_i (tie-breaker is lowest activity number)
10: $b_{Q_{(1)}} := b_{Q_{(1)}} + 1$
11: determine S'
12: **if** $s'_n \leqslant \delta$ **then**
13: $\quad S := S'$
14: $\quad$ go to line 5
15: **else**
16: $\quad b_{Q_{(1)}} := b_{Q_{(1)}} - 1$
17: $\quad Q \setminus \{Q_{(1)}\}$
18: $\quad$ **if** $Q = \emptyset$ **then** exit **else** go to line 10
19: **end if**

We illustrate the time buffering heuristic by describing some of the steps applied to our example network. Using the procedure to calculate the expected activity duration increases detailed above, we obtain the duration extension vector $E(\mathbf{\Delta}) = (0, 4, 25, 10, 14, 31, 14, 11, 6, 0)$. We start from the 'highest CIW first'-baseline schedule shown as the top schedule in Figure 6. Calculating the expected impacts for this schedule yields the disruption impact vector $I = (0, 0, 0, 44, 0, 99, 13, 32, 175, 2432)$. Activity 10 clearly has the highest impact value but is not considered for buffering because doing so would violate the deadline of 18 since activity 10 is assumed to start and end at the project deadline. Therefore activity 9 is selected and buffered with one time unit, yielding the second schedule in figure 6. The impact values are indicated between parentheses. The activity that is selected for buffering is marked in black. Continuing the algorithm eventually yields the third schedule of figure 6. The corresponding vector of buffer sizes is now $B = (0, 0, 0, 0, 0, 5, 4, 0, 1, 0)$.

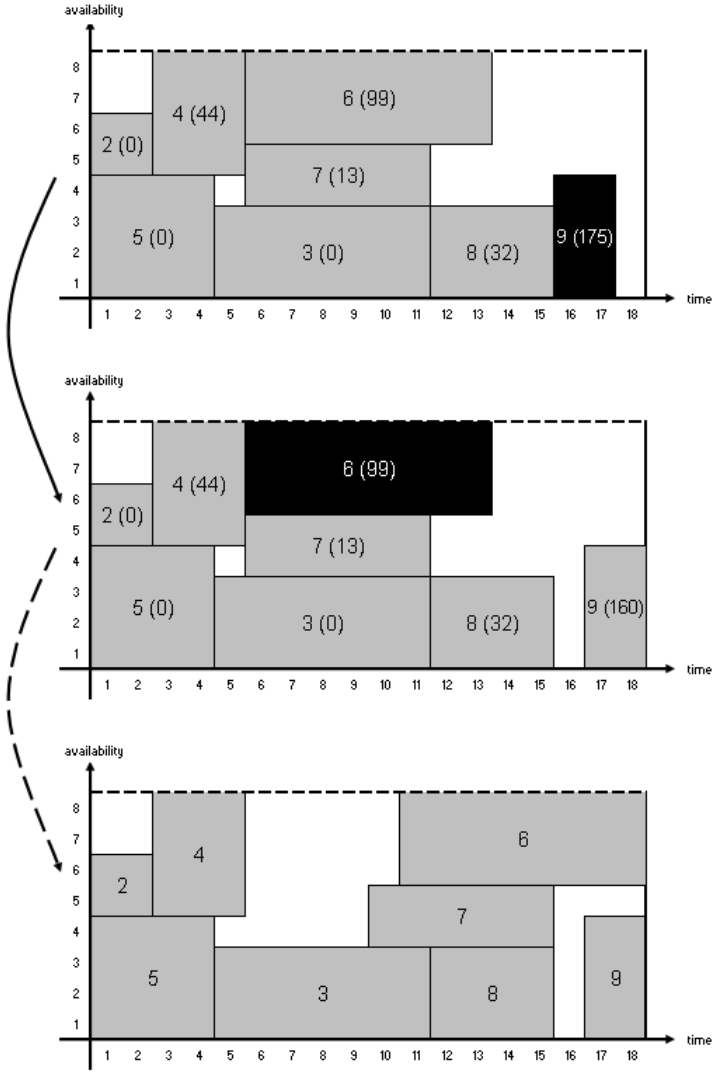

Figure 6: Time buffering applied to a 'highest CIW first' schedule

4 Reactive Strategies

After the baseline schedule has been determined, project execution can start. However, no matter how much we try to protect the predictive schedule against possible disruptions, we can never totally eliminate their occurrence. The execution of the baseline schedule continues either until the completion of the

dummy end activity, signaling the end of the project, or until a resource conflict is encountered resulting from the breakdown of one or more resources. When a resource conflict occurs, this conflict will have to be resolved by postponing one or more activities in order to restore schedule feasibility. We assume that preemption is not allowed unless a resource infeasibility due to a resource breakdown is resolved by interrupting the execution of an activity that was in progress at the time of the breakdown. Furthermore, we assume that this interrupted activity then has to be restarted from scratch (*preempt-repeat*). An extension to a *preempt-resume* setting would be an interesting topic for further research.

4.1 List scheduling

A good reactive strategy restores schedule feasibility while minimizing the deviation from the baseline schedule and preventing future disruptions from occurring. A simple reactive strategy could rely on *list scheduling*.

A *random* precedence feasible priority list can serve as a benchmark. As an alternative, we rely on a *scheduled order list* that allows us to reschedule the activities in the order dictated by the schedule, while taking into account the new, reduced resource availabilities. More specifically, when a disruption occurs at time t^*, we create a priority list L including the activities that are not yet completed, ordered in non-decreasing order of their baseline starting times.

The priority list is decoded into a feasible schedule using a *modified serial schedule generation scheme* and taking into account the known resource availabilities up to the current time period. The modification of the serial schedule generation scheme has to do with the case where the current activity taken from the list is in progress but not yet completed when the infeasibility occurs. This activity can be left unchanged, or it can be interrupted and repeated. The pseudocode for this procedure is given in algorithm 2. The new activity starting times are denoted as s'_i, the currently known availabilities as a'_{kt} and the set of direct predecessors of activity i as $PRED_i$. M is a sufficiently large integer so that the project ends before time period M.

Activities selected from the list are scheduled as early as possible. For activities that are in execution during the time of disruption t^*, this means that the procedure first tries the current scheduled starting time. If this turns out to be infeasible, the procedure searches for feasibility by starting the activity in the next time period (t^*+1) and subsequent time periods if necessary. For activities that did not start yet, it is only necessary to consider the earliest precedence

Algorithm 2 Modified Serial Schedule Generation Scheme

1: $t^* :=$ time period with resource infeasibility
2: $L :=$ precedence feasible ordered list with activities $i : s_i + d_i > t^*$
3: **for** $k := 1$ to K, $t := 1$ to M **do**
4: **if** $t \leqslant t^*$ **then** $a'_{kt} := a_{kt}$
5: **else** $a'_{kt} := a_k$
6: **end for**
7: **for** $i := 1$ to n **do**
8: **if** $i \notin L$ **then** $s'_i := s_i$
9: **end for**
10: **for** $p := 1$ to $|L|$ **do**
11: **if** $s_{L_{(p)}} < t^*$ **then** $s'_{L_{(p)}} := s_{L_{(p)}}$
12: **else** $s'_{L_{(p)}} := max\{t^* + 1, max_{i \in PRED_{L_{(p)}}}(s'_i + d_i)\}$
13: **while** $\exists k, t : \sum_{i:i \in \mathcal{S}_t} r_{ik} > a'_{kt}$ **do**
14: $s'_{L_{(p)}} := t + 1$
15: **end while**
16: **end for**

feasible starting time. Note that, as we stated in section 2, we never allow an activity to start before its baseline starting time s_i.

4.2 Tabu search

Solutions may be improved by superimposing a tabusearch based improvement heuristic (Glover and Laguna, 1993) on the priority list rule. This procedure will try to improve the starting solution by iteratively executing the best precedence feasible adjacent interchange of two activities in the priority list that does not lead to a state included in the tabu list. The objective is to find a precedence feasible ordering of activities that corresponds to a feasible schedule that deviates as little as possible from the baseline schedule. This deviation is measured by calculating the weighted sum of the absolute deviations between the baseline and the reactive schedule starting times. The advantage of tabu search is that by using a tabu list (a list of moves or states that are forbidden for a number of iterations) the procedure can also choose non-improving moves so that it avoids getting stuck in local optima like traditional local search approaches. The procedure is explained in algorithm 3.

Our implementation considers a maximum number of iterations $MAXITER$ that has to be executed before the procedure ends and includes a frequency based penalty function to further prevent cycling. The length of the tabu list

Algorithm 3 Tabu-search based reactive procedure

set $L^* := L$, $O^* := \sum_{i \in N} w_i(s'_i - s_i)$, $T := \lfloor \sqrt{|L|} \rfloor$, $iter := 0$
while (iter < MAXITER) **do**
  $O_0 := 999999$, $i^* := 0$
  for $i := 2$ to $n-2$ **do**
    if $(L_{(i)}, L_{(i+1)}) \notin A$ **then**
      exchange $L_{(i)}$ and $L_{(i+1)}$
      generate S by applying the modified serial schedule generation scheme to list L
      $O := \sum_{i \in N} w_i(s'_i - s_i)$
      if $s'_n \leqslant \delta$ and $O + freq_{L_{(i)}, s'_{L_{(i)}}} + freq_{L_{(i+1)}, s'_{L_{(i+1)}}} < O_0$ **then**
        if $(iter > tabu_{L_{(i)}, s'_{L_{(i)}}}$ AND $iter > tabu_{L_{(i+1)}, s'_{L_{(i+1)}}})$ OR $O < O^*$ **then**
          store $i \rightarrow i^*$
          $O_0 := O$
        end if
      end if
      exchange $L_{(i)}$ and $L_{(i+1)}$
    end if
  end for
  if $i^* \neq 0$ **then**
    $freq_{L_{(i)}, s'_{L_{(i)}}} + 10$, $freq_{L_{(i+1)}, s'_{L_{(i+1)}}} + 10$
    $tabu_{L_{(i)}, s'_{L_{(i)}}} := iter + T$, $tabu_{L_{(i+1)}, s'_{L_{(i+1)}}} := iter + T$
    exchange $L_{(i^*)}$ and $L_{(i^*+1)}$
    if $O = \sum_{i \in N} w_i(s'_i - s_i) < O^*$ **then**
      $O^* := O$ AND $L^* := L$
    end if
  end if
  $iter := iter + 1$
end while
generate S by applying the modified serial schedule generation scheme to list L^*

is set to $\lfloor\sqrt{|L|}\rfloor$. The best solution that is found so far is stored in L^* and has an objective function value equal to O^*. O_0 then is the objective function value of the best adjacent interchange found so far in the current iteration. The frequency based penalties are stored per pair (i, s_i) in the variables $freq_{i,s_i}$. A value of 10 was chosen because this yielded the best overall results and because a sufficiently large penalty is necessary to significantly change the usually relatively large objective function value. Likewise, the tabu status is stored in the variables $tabu_{i,s_i}$. Observe that an aspiration criterion is included: in case a tabu solution L has an objective function that outperforms the best solution value that was found so far ($O < O^*$), then the tabu status will be overridden and the solution will be stored anyway.

In order to illustrate the reactive procedure we include the example in figure 8. The schedule shown in figure 7 is disrupted in period 10 due to the breakdown of two resource units.

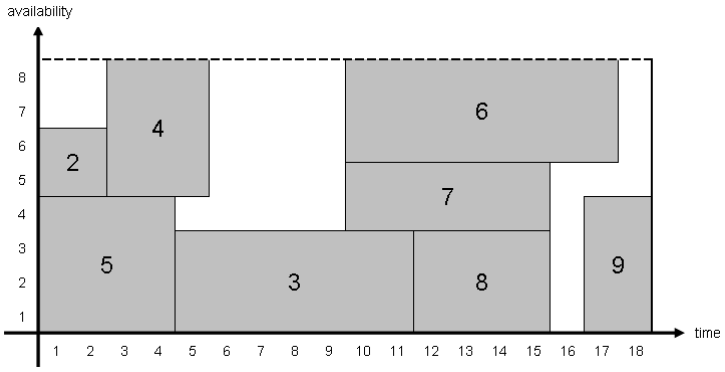

Figure 7: Disturbed schedule: 2 units down in period 10

Keeping the original schedule order, the partial priority list $(3, 7, 6, 8, 9, 10)$ is obtained, which yields the repaired schedule depicted in the top part of figure 8. However, improvement is possible. If we preempt and postpone activity 7 instead of activity 6, we obtain the partial priority list $(3, 6, 7, 8, 9, 10)$ and the schedule depicted in the bottom part of figure 8. The former schedule corresponds to a weighted schedule deviation cost of 9, whereas the latter schedule yields a weighted deviation cost of only 2.

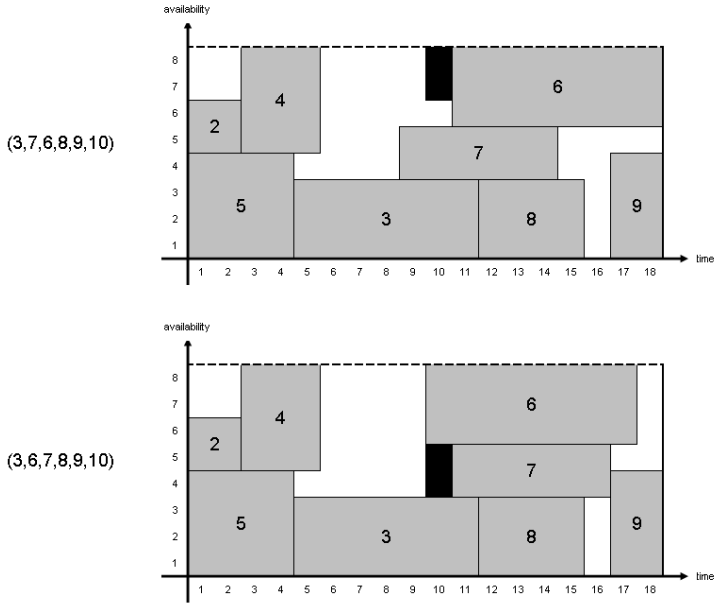

Figure 8: Reactive improvement procedure: scheduled versus improved ordering

5 Results

5.1 Experiment

The above algorithms were coded in Microsoft Visual C++ 6.0 and executed on a Dell Optiplex GX270 workstation. In order to evaluate the instability objective we use simulation. For determining the optimal solution to the deterministic RCPSP, we use the branch-and-bound algorithm developed by Demeulemeester and Herroelen (1992,1997).

The instability weights w_i for all non-dummy activities are drawn from a discrete, triangularly shaped distribution between 1 and 10 with $P(\mathbf{w}_i = x) = 0.21 - 0.02x$. Corresponding to what can be expected in real-life projects, most activities will have a low instability weight whereas only a minority are more heavily penalized for being started later than planned. The instability weight of the dummy end activity represents the importance of meeting the projected deadline and is set equal to β times the average of the instability weight distribution function, which is 3.85 for $P(\mathbf{w}_i = x)$. Because meeting the project deadline is usually deemed more important than starting each activity at the

planned starting time, we set $\beta = 10$ for our experiment.

The project deadline is derived from the minimal makespan schedule. In a static and deterministic environment, the lower bound on the makespan, C_{max}^{RCPSP}, corresponds to the makespan of the schedule obtained when optimally solving the RCPSP. It seems reasonable to assume that the project manager will prefer a makespan that does not deviate too much from this lower bound. Therefore, we set the deadline of the robust schedule at $C_{max}^{RCPSP}(1+\alpha)$, where the deadline factor α is a parameter chosen by the project manager that constitutes the trade-off between project stability and project duration (Van de Vonder et al., 2005b).

As mentioned in section 3.4, it can be shown that resource breakdowns can be modeled using exponential distributions with the parameters $MTTF_k$ and $MTTR_k$. We draw the $MTTR_k$ values from a uniform discrete distribution between 1 and 5. The values for $MTTF_k$ are drawn from a uniform discrete distribution between 50% and 150% of C_{max}^{RCPSP}.

As a test set for assessing the effectiveness of the proactive-reactive strategies, we use the 480 30-activity RCPSP instances of the well-known PSPLIB set of test problems (Kolisch and Sprecher, 1997). Each combination of a proactive policy and a reactive policy was tested using 10 replications for each problem instance, each having different $MTTF_k$'s and $MTTR_k$'s. Furthermore, we used 50 iteration steps for the reactive tabu search.

5.2 Computational Results

The computational results are shown in tables 1, 2 and 3. The results shown in table 1 were obtained for a tight deadline setting $\alpha = 15\%$, those in table 2 for $\alpha = 30\%$, and those in table 3 for an ample deadline setting of $\alpha = 45\%$. We list the median values of the weighted instability costs over all projects and MTTF-MTTR scenarios for the eight proactive scheduling combinations (time buffering or not, resource buffering or not, in combination with a minimum makespan schedule or a schedule obtained using 'highest CIW first') in combination with the three reactive procedures (random list scheduling, scheduled order list scheduling and tabu search). The numbers shown in italic in the last column give the average weighted instability cost value for each of the proactive scheduling rules, the italic numbers in the bottom row represent the average instability cost value for each of the reactive procedures.

Let us first have a look at the results for the proactive procedures. As could

be expected, a proactive scheduling procedure using time buffering always seems to outperform procedures that do not. Of course, the possible improvement of time buffering directly depends on the degree of freedom offered to the time buffering algorithm for inserting buffers in the schedule. Therefore, whereas for the $\alpha = 30\%$ scenario, an average improvement of 41.27% can be observed for a minimal makespan schedule, this decreases to 32.18% for the 'highest CIW first'-schedule and only 25.24% for a resource buffered 'highest CIW first'-schedule. Similar results hold when varying the deadline factor. For minimal makespan schedules, a 54.86% improvement seems to be possible when $\alpha = 45\%$ compared to only 21.92% when $\alpha = 15\%$.

Resource buffering performs quite well, offering average improvements of 60.19% for the minimal makespan case. Again, the improvement potential decreases as the deadline factor is decreased. Taking into consideration the promising required computation time, resource buffering based strategies cannot be neglected. In case of minimal makespan scheduling, for 83 of the 480 test instances the calculated buffered availability turned out to be too low. However, for 'highest CIW first' scheduling the picture is slightly different. For the networks for which 'highest CIW first' scheduling respected the project deadline, the average availability turned out to be insufficient to obtain a feasible schedule for 14.79% of the cases when $\alpha = 15\%$ and for 17.29% when $\alpha = 30\%$ and when $\alpha = 45\%$.

For most cases, 'highest CIW first' performs better than minimal makespan scheduling. This is not surprising since we actively try to improve the objective function value of the minimal makespan schedule. However, for $\alpha = 30\%$ the use of time buffering actually seemed to favour minimal makespan scheduling. A possible reason might be that minimal makespan scheduling creates a shorter schedule in which more opportunities for time buffering exist. On the other hand, similar results were not found for $\alpha = 45\%$. For $\alpha = 45\%$ the combinations based on 'highest CIW first' performed slightly better than the ones based on minimal makespan. Also note that for $\alpha = 15\%, 30\%, 45\%$, determining the 'highest CIW first'-schedule turned out to be impossible in respectively 26.46%, 3.13% and 0.00% of the instances, because the corresponding schedule exceeded the deadline.

The results for the reactive strategies are as expected. Random list scheduling performs worst. Scheduled order list scheduling offers a significant performance increase. Tabu search allows further improvements upon scheduled order list scheduling.

Table 1: Median of weighted instability for $\alpha = 15\%$

			random list	scheduled order	tabu search	
no time buf	**no res buf**	**min Cmax**	1348.55	325.55	276.60	*650.23*
		max CIW	1004.60	225.60	177.90	*469.37*
	res buf	**min Cmax**	612.55	122.50	112.20	*282.42*
		max CIW	524.10	104.00	76.50	*234.87*
time buf	**no res buf**	**min Cmax**	1080.30	234.50	208.35	*507.72*
		max CIW	746.20	152.70	140.10	*346.33*
	res buf	**min Cmax**	541.10	107.35	96.50	*248.32*
		max CIW	376.10	80.70	72.60	*176.47*
			779.19	*169.11*	*145.09*	

Table 2: Median of weighted instability for $\alpha = 30\%$

			random list	scheduled order	tabu search	
no time buf	**no res buf**	**min Cmax**	1117.40	296.50	239.30	*551.07*
		max CIW	1011.30	243.90	198.80	*484.67*
	res buf	**min Cmax**	477.40	96.75	80.80	*218.32*
		max CIW	433.40	82.90	67.80	*194.70*
time buf	**no res buf**	**min Cmax**	707.85	141.65	121.35	*323.62*
		max CIW	711.40	145.90	128.80	*328.70*
	res buf	**min Cmax**	293.15	53.65	49.10	*131.97*
		max CIW	333.30	55.40	48.00	*145.57*
			635.65	*139.58*	*116.74*	

Table 3: Median of weighted instability for $\alpha = 45\%$

			random list	scheduled order	tabu search	
no time buf	**no res buf**	**min Cmax**	1025.15	296.50	232.30	*517.98*
		max CIW	866.30	234.40	188.00	*429.57*
	res buf	**min Cmax**	393.05	93.85	78.65	*188.52*
		max CIW	322.60	68.50	55.10	*148.73*
time buf	**no res buf**	**min Cmax**	501.20	104.35	95.90	*233.82*
		max CIW	486.40	97.10	87.60	*223.70*
	res buf	**min Cmax**	173.15	39.85	36.10	*83.03*
		max CIW	173.90	35.60	29.20	*79.57*
			492.72	*121.27*	*100.36*	

Table 4: Average CPU time in seconds

				$\alpha = 15\%$	$\alpha = 30\%$	$\alpha = 45\%$
proactive	**no time buf**	**no res buf**	**min Cmax**	0.066	0.066	0.068
			max CIW	0.000	0.000	0.000
		res buf	**min Cmax**	0.173	0.076	0.078
			max CIW	0.001	0.001	0.000
	time buf	**no res buf**	**min Cmax**	0.149	0.188	0.234
			max CIW	0.172	0.189	0.236
		res buf	**min Cmax**	0.124	0.162	0.209
			max CIW	0.149	0.163	0.208
reactive	**random list**			0.000	0.000	0.000
	scheduled order			0.000	0.000	0.000
	tabu search			0.069	0.070	0.073

Table 4 lists the average required CPU times in seconds. Proactive policies based on 'highest CIW first' are computationally very cheap. This is not surprising given the simple schedule construction procedures based on the serial schedule generation scheme. Minimal makespan scheduling is slower because of the exact branch-and-bound approach. Especially when looking at the case of $\alpha = 15\%$ we see that the minimal makespan procedure combined with resource buffering is rather slow. This is probably due to the fact that given the restrictive deadlinefactor the procedure has to be executed a number of times until the buffered availability allows for the creation of a deadline feasible schedule. The time buffering procedure is also quite demanding with increasing computation times as the deadlinefactor increases. The simple reactive policies are very fast, tabu search, however, is computationally slightly more expensive.

6 Conclusion

In this paper we gave an overview of the challenges a project manager has to deal with in an environment characterized by uncertain resource availabilities. We gave an overview of the literature on scheduling under uncertainty and underlined the necessity of building a proactive baseline schedule for minimizing weighted instability. For the generation of a robust baseline schedule we proposed eight strategies. First of all, a starting schedule can be built using for example minimal makespan scheduling or 'highest CIW first' scheduling. This schedule can then be protected against the effects of disruptions by using the average resource availabilities, obtained from steady-state probability calculations, instead of the given deterministic availabilities. Alternatively or additionally, protection can be added in the form of explicitly inserted idle time. To determine where and in what amount to insert these so-called time buffers, we developed a time buffer allocation heuristic based on the estimation of the expected impacts of activity duration prolongations due to resource breakdowns. The advantages of 'highest CIW first' scheduling, combined with resource buffering and time buffering, immediately become apparent when comparing the weighted instability results with those from a minimal makespan strategy without buffering. From a simulation experiment using the PSPLIB set of test problems we were able to observe an average improvement of 77%.

Unfortunately, no matter how much one tries to protect the initial schedule, the occurrence of disruptions during project execution can never be totally

avoided. Therefore reactive policies, indicating how to restore schedule feasibility after the occurrence of a resource breakdown, are required. We proposed three reactive policies to resolve infeasibilities resulting from schedule disruptions. A random order rule was included for benchmarking purposes. The scheduled order rule seems to perform reasonably well in comparison but can still be improved by tabu search at the cost of an increase in computation time. Here the computational experiment also immediately showed the advantages of using an intelligent reactive strategy. Using scheduled order instead of random order allowed us to obtain results that were on average 77% better. A further improvement of about 16% was possible when superimposing a tabusearch improvement procedure on the original order rule.

In conclusion, we were able to show that a combination of 'highest CIW first' scheduling, resource buffering, time buffering and a reactive strategy based on an improvement of the scheduled order rule, allows for an improvement of the objective function of 96% compared to minimal makespan scheduling without any buffering using a naive random order strategy.

References

Al-Fawzan, M. A. and Haouari, M. (2005). A bi-objective model for robust resource-constrained project scheduling. *International Journal of Production Economics*, 96, pp 175–187.

Aytug, H., Lawley, M., McKay, K., Mohan, S. and Uzoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161, pp 86–110.

Brucker, P., Drexl, A., Möhring, R., Neumann, K. and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models and methods. *European Journal of Operational Research*, 112, pp 3–41.

Davenport, A. and Beck, J. (2002). A survey of techniques for scheduling with uncertainty. *Unpublished manuscript.*

Demeulemeester, E. and Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38, pp 1803–1818.

Demeulemeester, E. and Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43, pp 1485–1492.

Demeulemeester, E. and Herroelen, W. (2002). *Project scheduling - A research handbook*. Vol. 49 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Boston.

Drezet, L.-E. (2005). *Résolution d'un problème de gestion de projets sous contraintes de ressources humaines: De l'approche prédictive à l'approche réactive*. PhD thesis. Université François Rabelais Tours, France.

Girault, M. (1959). *Initiation aux Processus Aléatoires*. Dunod, Paris.

Glover, F. and Laguna, M. (1993). *Tabu Search*. Blackwell Scientific, Oxford. pp 70–141. In C. Reeves (Editor): Modern Heuristic Techniques for Combinatorial Problems.

Herroelen, W., De Reyck, B. and Demeulemeester, E. (1998). Resource-constrained scheduling: A survey of recent developments. *Computers and Operations Research*, 25, pp 279–302.

Herroelen, W., De Reyck, B. and Demeulemeester, E. (2000). On the paper "Resource-constrained project scheduling: Notation, classification, models and methods" by Brucker et al.. *European Journal of Operational Research*, 128(3), pp 221–230.

Hopp, W. and Spearman, M. (2001). *Factory physics: Foundations of manufacturing management*. McGraw-Hill.

Kolisch, R. and Sprecher, A. (1997). PSPLIB - A project scheduling library. *European Journal of Operational Research*, 96, pp 205–216.

Lambrechts, O., Demeulemeester, E. and Herroelen, W. (2006). A tabu search procedure for generating robust project baseline schedules under stochastic resource availabilities. *Research Report KBI 0604*. FETEW, Katholieke Universiteit Leuven, Belgium.

Leus, R. (2003). *The generation of stable project plans*. PhD thesis. Department of applied economics, Katholieke Universiteit Leuven, Belgium.

Leus, R. and Herroelen, W. (2005). The complexity of machine scheduling for stability with a single disrupted job. *Operations Research Letters*, 33, pp 151–156.

Mehta, S. and Uzsoy, R. (1998). Predictive scheduling of a job shop subject to breakdowns. *IEEE Transactions on Robotics and Automation*, 14, pp 365–378.

Mehta, S. and Uzsoy, R. (1999). Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, 12, pp 15–38.

Nikulin, Y. (2004). Robustness in combinatorial optimization and scheduling theory: An annotated bibliography. *Research Report 583*. Christian-Albrechts University in Kiel, Institute of Production and Logistics.

Pinedo, M. (1995). *Scheduling - Theory, Algorithms and Systems*. Prentice Hall, Englewood Cliffs, New Jersey.

Ross, S. (1983). *Stochastic Processes*. John Wiley, New York.

Scholl, A. (2001). *Robuste Planung und Optimierung: Grundlagen, Konzepte und Methoden, Experimentelle Untersuchungen*. Physica-Verlag, Heidelberg.

Stork, F. (2001). *Stochastic Resource-Constrained Project Scheduling*. PhD thesis. Technical University of Berlin, School of Mathematics and Natural Sciences.

Van de Vonder, S., Demeulemeester, E., Herroelen, W. and Leus, R. (2005a). The trade-off between stability and makespan in resource-constrained project scheduling. *International Journal of Production Research*, 44(2), pp 215–236.

Van de Vonder, S., Demeulemeester, E., Herroelen, W. and Leus, R. (2005b). The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics*, 97, pp 227–240.