INFORMS TutORials in Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

# Generating Robust Project Baseline Schedules

Willy Herroelen

To cite this entry: Willy Herroelen. Generating Robust Project Baseline Schedules. *In* INFORMS TutORials in Operations Research. Published online: 14 Oct 2014; 124-144.
https://doi.org/10.1287/educ.1073.0038

TUTORIALS IN
OPERATIONS RESEARCH
INFORMS 2007

 | ISBN 13 978-1-877640-22-3
DOI 10.1287/educ.1073.0038



# Generating Robust Project Baseline Schedules

***Willy Herroelen***
Research Center for Operations Management, Katholieke Universiteit Leuven, B-3000 Leuven, Belgium, willy.herroelen@econ.kuleuven.be

**Abstract** Most research efforts in resource-constrained project scheduling assume a static and deterministic environment within which the precomputed baseline schedule will be executed. Project activities, however, may be subject to considerable uncertainty, which may lead to numerous schedule disruptions during project execution. In this tutorial, we discuss proactive project scheduling procedures for generating robust baseline schedules that are sufficiently protected against anticipated time and/or resource disruptions in combination with reactive policies that may be deployed to repair the baseline schedule during project execution.

**Keywords** project scheduling; uncertainty; stability; buffers

## 1. Introduction

In this tutorial we focus on proactive procedures for generating project baseline schedules that are sufficiently protected against disruptions that may be caused by uncertainties in the activity durations and/or resource availabilities. The proactive procedures can be used in combination with reactive procedures to be deployed when the baseline schedule, despite its protection, breaks.

### 1.1. The Project Baseline Scheduling Problem

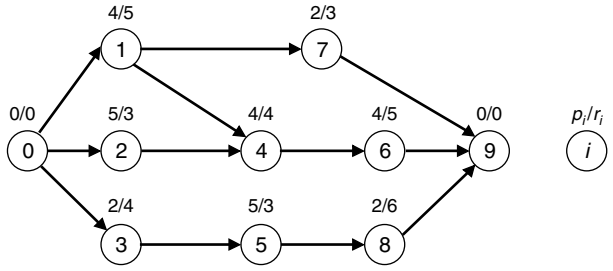
The vast majority of the project scheduling research efforts over the past several years has concentrated on the development of workable baseline schedules, assuming complete information and a static and deterministic environment. Such a *baseline schedule* (*preschedule*, *predictive schedule*) is traditionally constructed by solving the so-called *resource-constrained project scheduling problem* (RCPSP).

The RCPSP (problem $m,1|cpm|C_{\max}$ in the notation of Herroelen et al. [15]) involves the determination of activity starting times that satisfy both the zero-lag finish-start precedence constraints and the renewable resource constraints under the objective of minimizing the project duration (for reviews, we refer to Brucker et al. [7], Demeulemeester and Herroelen [9], Herroelen et al. [14], Kolisch and Padman [20]).

The deterministic RCPSP can be stated as follows. Consider a project network $G(N,A)$ represented in activity-on-the-node representation format with dummy start node 0 and dummy end node $n$. All nondummy project activities have durations $p_i$ and are subject to zero-lag finish-start precedence constraints on the elements of $A$: We require $s_j \geq s_i + p_i$ if $(i,j) \in A$, with $s_i$ the planned start time of activity $i$. Nondummy activities require an integer per period amount $r_{ik}$ of one or more renewable resource types $k$, $k = 1,\ldots,q$, during their execution. All resource types $k$ have a per-period capacity $a_k$. The objective is to derive a precedence and resource feasible baseline schedule $\mathcal{S}^B = (s_0, s_1, \ldots, s_n)$ of activity start times that minimizes the duration of the project.

A project network example is shown in Figure 1 (Van de Vonder [37]). For each activity, the duration and per period requirement for a single renewable resource are shown above

FIGURE 1. Project network example.




the corresponding node. The single renewable resource type is assumed to have a per period availability $a = 10$.

Conceptually, the RCPSP can be formulated as follows:

$$\text{minimize} \quad s_n \tag{1}$$

$$\text{subject to} \quad s_i + p_i \leqslant s_j \quad \forall (i,j) \in A \tag{2}$$

$$s_0 = 0 \tag{3}$$

$$\sum_{i:\, i \in \mathcal{A}_t} r_{ik} \leqslant a_k \quad \text{for } k = 1, \ldots, q \text{ and } t = 1, \ldots, s_n. \tag{4}$$

The set $\mathcal{A}_t$ that is used in Equation (4) denotes the set of activities that are in progress at time $t$. The objective function Equation (1) minimizes the start time of the dummy end activity and hence the duration of the project. Equation (2) expresses the finish-start zero-lag precedence relations, whereas Equation (3) forces the dummy start activity to start at time 0. Finally, Equation (4) expresses that at no time instant during the project horizon the resource availability may be violated. The formulation is conceptual: The linear program cannot be solved directly because there is no easy way to translate the set $\mathcal{A}_t$ into a linear programming formulation. We refer to Demeulemeester and Herroelen [9] for a detailed discussion of mathematical programming formulations for the RCPSP. The deterministic RCPSP has been shown to be strongly $\mathcal{NP}$-hard (Blazewicz et al. [6]).

A minimum makespan schedule for the project network of Figure 1 is shown in Figure 2. The critical sequence, which determines the 15-period project duration is the chain $\langle 0, 2, 4, 6, 8, 9 \rangle$. Figure 3 shows the corresponding resource profile. A baseline schedule $\mathcal{S}^B$, such as the one given in Figure 2, serves a number of important functions (Aytug et al. [4], Mehta and Uzsoy [31], Wu et al. [47]). One of these is to provide internal visibility within the organization of the planned activity execution periods reflecting the requirements for the key staff, equipment, and other resources. The baseline schedule is also the starting point for communication and coordination with external entities in the company's inbound and outbound supply chain: It constitutes the basis for agreements with suppliers and subcontractors (e.g., for planning external activities such as material procurement and preventive

FIGURE 2. Minimum makespan schedule.

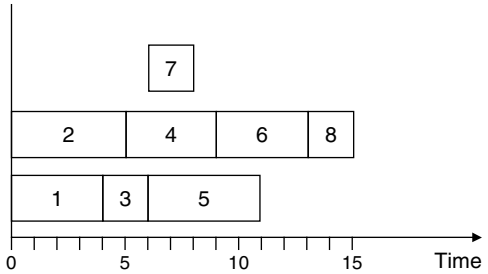
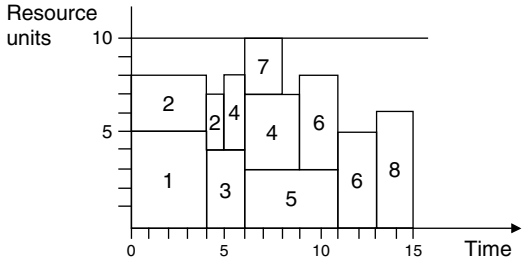


FIGURE 3. Resource profile for the minimum makespan schedule.




maintenance), as well as for commitments to customers (delivery dates). During execution, however, a project may be subject to considerable uncertainty, which may lead to numerous disruptions in the baseline schedule.

Many types of disruptions have been identified in the literature (we refer to Yu and Qi [48], Wang [46], and Zhu et al. [49]). Activities can take longer or shorter than expected, resource requirements or availability may vary, ready times and due dates may change, new activities may have to be inserted in the schedule, the project may have to be interrupted for a certain time, etc. In this tutorial we focus on schedule disruptions that may be caused by uncertainty in the duration of the project activities and/or in the availability of the renewable resources.

*Proactive/reactive project scheduling* procedures try to cope with schedule disruptions that may occur during project execution through the combination of a *proactive scheduling procedure* for generating a robust baseline schedule with a *reactive procedure* that is invoked when a schedule breakage occurs during project execution and the schedule needs to be repaired. Research in proactive/reactive project scheduling is growing steadily (for surveys in machine scheduling and project scheduling environments, see Aytug et al. [4], Herroelen and Leus [12, 13], and Vierra et al. [45]). The objective of this tutorial is to discuss promising proactive/reactive project scheduling procedures that may be deployed at the occurrence of disruptions that are caused by uncertain activity durations or uncertain resource availabilities.

The chapter is organized as follows. Section 2 distinguishes between the concepts of solution and quality robustness and defines a number of robustness measures. Section 3 focuses on exact and suboptimal proactive/reactive scheduling procedures under the assumption that the uncertainty stems from the activity durations. Both resource allocation and time-buffering proactive procedures are discussed that can be used in conjunction with exact and suboptimal reactive strategies that may be deployed during project execution upon schedule breakage. Section 4 concentrates on solution robust scheduling under resource availability uncertainty. We discuss an integrated proactive scheduling procedure to be used in combination with exact or suboptimal reactive scheduling methods. The last section presents some overall conclusions.

## 2. Proactive/Reactive Project Scheduling

Proactive/reactive scheduling involves a proactive and a reactive phase. During the proactive phase, a baseline schedule $\mathcal{S}^B$ is constructed that accounts for statistical knowledge of uncertainty and anticipates disruptions. The underlying idea is to protect the schedule as much as possible against the distortions that may occur during the execution of the project. When disruptions occur during actual project execution, it may be necessary to call upon reactive scheduling procedures to modify the baseline schedule in response to these disruptions. The schedule actually executed after these modifications is called the *realized schedule* $\mathcal{S}^R$ (Aytug et al. [4]). In general terms, a baseline schedule that is rather "insensitive" to

disruptions that may occur during project execution is called *robust*. The robustness concept has been used in many disciplines (see e.g., Kouvelis and Yu [21], Roy [34], and Billaut et al. [5]). Many different types of robustness have been identified in the literature.

In the next section we introduce proper definitions and measures.

## 2.1. Robustness Types and Measures

The robustness measure used can be single or composite. Two often used types of *single robustness measures* have been distinguished: quality robustness and solution robustness (Herroelen and Leus [13], Van de Vonder et al. [43]; for other typologies we refer to Sanlaville [35]).

**2.1.1. Solution Robustness or Schedule Stability.** Solution robustness or schedule stability refers to the *difference* between the baseline schedule $\mathcal{S}^B$ and the realized schedule $\mathcal{S}^R$. This difference or *distance* $\Delta(\mathcal{S}^B, \mathcal{S}^R)$ for a given execution scenario can be measured in a number of ways: the number of disrupted activities, the difference between the planned and realized activity start times, etc. Sanlaville [35] suggests to measure solution robustness as

$$\max_I \Delta(\mathcal{S}^B, \mathcal{S}^R), \tag{5}$$

the maximum difference between the baseline schedule $\mathcal{S}^B$ and the realized schedule $\mathcal{S}^R$ over the set of execution scenarios $I$. The objective of the proactive/reactive scheduling procedure then is to minimize the maximum distance between the baseline and the realized schedule.

Leus and Herroelen [29] suggest to measure the difference by the weighted sum of the absolute difference between the planned and realized activity start times, i.e.

$$\Delta(\mathcal{S}^B, \mathcal{S}^R) = \sum_{i \in N} w_i |S_i - s_i|, \tag{6}$$

where $s_i$ denotes the planned starting time of activity $i \in N$ in the baseline schedule $\mathcal{S}^B$, $S_i$ is a random variable denoting the actual starting time of activity $i$ in the realized schedule $\mathcal{S}^R$, and the weights $w_i$ represent the activity disruption cost per time unit, i.e., the nonnegative cost per unit time overrun or underrun on the start time of activity $i$. This disruption cost reflects either the difficulty in shifting the booked time window on the required resources (*internal stability*, or the difficulty in obtaining the required resources) or the importance of on-time performance of the activity (*external stability*). In practice, these penalties may be considerable. For example, the penalty of not meeting the delivery date of the renovated Berlaymont Building, housing the European Commission in Brussels (Belgium), was set to EUR 221,000 per month of delay (Kinnock [19]).

The objective of the proactive/reactive scheduling procedure is then to minimize $\sum_{i \in N} w_i E|S_i - s_i|$ with $E$ denoting the expectation operator; i.e., to minimize the weighted sum of the expected absolute difference between the planned and realized activity start times. It should be observed that the exact determination of the expected value of a function of the activity durations is unrealistic (Möhring [32], Leus and Herroelen [29]). Hagstrom [11] has shown for projects without resource constraints that even when every stochastic activity duration $P_i$ can take only two discrete values, then computing the expected project duration and computing the probability that the project is finished by a given time instant, assuming an early-start schedule, is $\#\mathcal{P}$ complete. For $\mathcal{NP}$-hardness proofs of several cases of the scheduling problem for stability for projects subject to a deadline and discrete disturbance scenario, we refer to Leus and Herroelen [30]. The objective function is usually determined using simulation (Igelmund and Radermacher [17], Leus and Herroelen [29], Stork [36]). The obtained objective function values will then be dependent on the simulated disruptions and

on the reactive procedure applied during the simulated project execution in order to repair the schedule upon breakage (Leon et al. [26]).

As will be illustrated later in this tutorial, solution robustness may also be evaluated using *surrogate* objective functions that are easier to compute (see also Aloulou and Portmann [1], Policella et al. [33], Deblaere et al. [8], Lambrechts et al. [22]–[25]).



**2.1.2. Quality Robustness.** Quality robustness refers to the insensitivity of the solution value of the baseline schedule to distortions. The ultimate objective of a proactive/reactive scheduling procedure is to construct a baseline schedule $\mathcal{S}^B$ for which the solution value does not deteriorate when disruptions occur. The quality robustness is measured in terms of the value of the objective function $z$. In a project setting, commonly used objective functions are project duration (makespan), project earliness and tardiness, project cost, net present value, etc.

When stochastic data are available, quality robustness can be measured by considering the *expected value of the objective function*, such as the expected makespan $E[C_{\max}]$, the classical objective function used in stochastic resource-constrained project scheduling (Stork [36]).

It is logical to use the *service level* as a quality robustness measure, i.e., to maximize $P(\mathbf{z} \leq z)$, the probability that the solution value of the realized schedule stays within a certain threshold. As a result, we want to maximize the probability that the project completion time does not exceed the project due date $d_n$, i.e., $P(S_n \leq d_n)$, where $S_n$ denotes the actual starting time of the dummy end activity. Van de Vonder et al. [41] refer to this measure as the *timely project completion probability* (TPCP). It should be observed that even the analytic evaluation of this measure for a given schedule and in the presence of ample resource availability is very cumbersome, the PERT problem being $\#\mathcal{P}$ complete (Hagstrom [11]).

Quality robustness can also be measured by comparing the solution value $\mathbf{z}$ of the realized schedule obtained by the proactive/reactive scheduling procedure and the optimal solution value $\mathbf{z}^*$ computed ex post by applying an exact procedure on the basis of the actually realized activity durations. Leus and Herroelen [28], for example, have used the percentage deviation of $S_n$, the project duration of the realized schedule, from the optimal makespan, computed by applying a branch-and-bound procedure on the basis of the actually realized activity durations as a measure of quality robustness.

**2.1.3. Composite Robustness Measures.** The robustness measures described above are all single measures. Also composite robustness objectives may be used (Hoogeveen [16]). Van de Vonder et al. [39] use the bicriteria objective $F(P(S_n \leq d_n), \sum w_i E|S_i - s_i|)$ of maximizing the timely project completion probability and minimizing the weighted sum of the expected absolute deviation in activity starting times. The authors assume that the composite objective function $F(.,.)$ is not known a priori, that the relative importance of the two criteria is not known from the outset, and that no clear linear combination is known that would reflect the preference of the decision maker. The analytic evaluation of the composite objective function is very cumbersome (as mentioned before, the PERT problem is $\#\mathcal{P}$ complete (Hagstrom [11]) and the scheduling problem for stability is $\mathcal{NP}$-hard in the ordinary sense (Leus and Herroelen [30])). A natural way out is to evaluate the composite objective function through simulation.

## 3. Solution Robust Project Scheduling Under Activity Duration Uncertainty

### 3.1. The Proactive/Reactive Scheduling Problem

We consider a project network $G(N, A)$ represented in activity-on-the-node representation with dummy start node 0 and dummy end node $n$. All nondummy project activities are now assumed to have *stochastic activity durations* $P_i$ and are subject to zero-lag finish-start precedence constraints on the elements of $A$: We require $S_j \geq S_i + P_i$ if $(i, j) \in A$,

with $S_i$ the activity start time of activity $i$ realized during project execution. Nondummy activities require an integer per period amount $r_{ik}$ of one or more renewable resource types $k$, $k = 1, \ldots, q$, during their execution. All resource types $k$ have a per-period capacity $a_k$. As before, the activity weight $w_i$ denotes the marginal cost of a deviation between the realized start time $S_i$ of activity $i$ and its planned start time $s_i$ in the baseline schedule. As mentioned earlier, these weights may include unforeseen storage costs, extra organizational costs, costs related to agreements with subcontractors, or just a cost that expresses the dissatisfaction of employees with schedule changes. We assume that $w_0 = 0$, whereas $w_n$ denotes the cost of delaying the project completion beyond a predetermined deterministic due date $d_n$.

The proactive scheduling objective is to build a *solution robust* or *stable* precedence and resource feasible baseline schedule, the activity starting times of which are denoted by $s_i$. We assume in this chapter that stability is strived for by minimizing the stability function $\sum_{i \in N} w_i E|S_i - s_i|$, defined earlier in Equation (6).

## 3.2. Proactive Scheduling

We assume that a *two-phase procedure* is used for generating stable schedules. In the first phase, an input schedule is generated that is both precedence and resource feasible but is not intentionally protected against anticipated disruptions. Such a schedule can be generated by solving the underlying resource-constrained project scheduling problem (problem RCPSP or problem $m, 1|cpm|C_{\max}$), using (deterministic) single-point estimates $p_i$ for each $P_i$.

Two strategies may then be used in the second phase to increase the stability of the input schedule. One way is to aim at a robust resource allocation, i.e., to decide on a clever way in which the various resource units are transferred between the activities of the schedule. Another is to insert time buffers that should prevent as much as possible the propagation of distortions throughout the schedule.

Leus and Herroelen [30] have shown that under the assumption that a single activity may deviate from its preschedule duration (without knowing which) and a single disruption scenario per activity, the machine scheduling problem for stability is strongly $\mathcal{NP}$-hard for a single machine with unequal ready times or precedence constraints and for the case of a free number of parallel machines; the single-machine problem without ready times and precedence constraints is still ordinarily $\mathcal{NP}$-hard.

**3.2.1. Robust Resource Allocation.** Leus [27] and Leus and Herroelen [29] study the problem of generating a *robust resource allocation* for a *given* feasible baseline schedule $\mathcal{S}^B$. They explore the fact that the search for an optimal resource allocation reduces to the search for a so-called *resource flow network* (which describes the routing of resources across the activities in the schedule) that exhibits desirable robustness characteristics. A branch-and-bound algorithm is developed that solves the robust resource allocation problem in exact and approximate formulations for the case of a *single renewable resource type* and exponential activity duration disruption lengths.

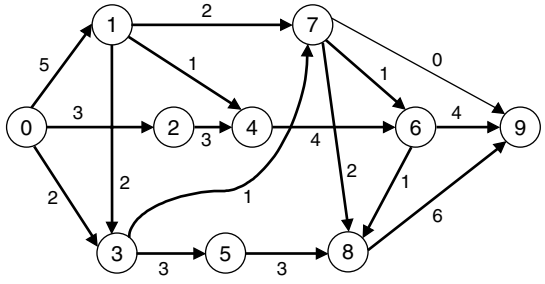
***Resource Flow Network.*** Artigues and Roubellat [2] introduce the concept of a resource flow network in which the flow describes the resource units transferred among the activities. Let $u_i = r_i$, $\forall i \in N \backslash \{0, n\}$ and $u_0 = u_n = a$. A *resource flow* $f$ associates with each activity pair $(i, j) \in N \times N$ a value $f_{ij} = f(i, j) \in \mathbb{N}$. The flow values must respect the following constraints, which impose the conservation of flow in a node as well as a lower and upper bound on the flow

$$f(i, N) = u_i \quad \forall i \in N \backslash \{n\} \tag{7}$$

$$f(N, i) = u_i \quad \forall i \in N \backslash \{n\}, \tag{8}$$

$f_{ij}$ denotes the number of units of the single resource type transferred from activity $i$ to activity $j$. For a flow $f$, let $\Phi(f) = \{(i, j) \in N \times N \mid f_{ij} > 0\}$ denote the set of arcs carrying

FIGURE 4. Resource flow network for the example project.




nonzero flow. Let $TA$ denote the transitive closure of $A$, meaning that $(i,j) \in TA$ if a path exists from $i$ to $j$ in $G(N,A)$. The arcs $\mathcal{X}(f) = \Phi(f) \backslash TA$ denote the flow carrying arcs that do not represent technological precedence constraints. In other words, $\mathcal{X}(f)$ is a set of additional arcs inducing additional precedence constraints. For all $X \subset N \times N$, $G_X$ denotes the graph $G(N, TA \cup X)$. The flow $f$ is a *feasible flow* if $G_{\mathcal{X}(f)}$ is acyclic: The additional precedence constraints implied by $\mathcal{X}(f)$ do not prohibit the realization of the project.

For the project network of Figure 1 $u_0 = u_9 = 10$, $u_1 = 5$, $u_2 = 3$, $u_3 = 4$, $u_4 = 4$, $u_5 = 3$, $u_6 = 5$, $u_7 = 3$, and $u_8 = 6$. A possible resource flow is shown in Figure 4.

Activity 8, for example, has a per period resource requirement of six units ($u_8 = 6$). It uses three resource units released by its predecessor activity 5, two units passed on by activity 7 and one unit released by activity 6.

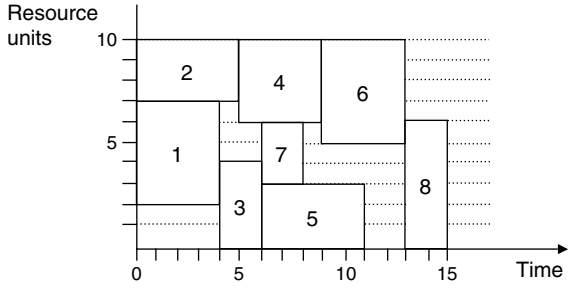
The set of arcs carrying nonzero flow, $\Phi(f) = \{(0,1),(0,2),(0,3),(1,3),(1,4),(1,7),(2,4),(3,5),(3,7),(4,6),(5,8),(6,8),(6,9),(7,6),(7,8),(8,9)\}$ are shown in bold. The set of arcs $\mathcal{X}(f) = \{(1,3),(3,7),(6,8),(7,6),(7,8)\}$ represent extra precedence relations that were not present in the original project network. The arc $(7,9)$ does not carry flow.

Let $\theta(X)$, $X \subseteq N \times N$, be the earliest start schedule in which the starting time of the dummy activity $s_0 = 0$ and each other activity $i$ starts at time $s_i = \max_{j \in \pi_{A \cup X}(i)} \{s_j + p_j\}$, with $\pi_{A \cup X}(i)$ the immediate predecessors of activity $i$ in the acyclic graph $G(N, A \cup X)$. A solution to an RCPSP instance can be obtained by finding a feasible flow $f$ that minimizes $s_n(\theta(A \cup \mathcal{X}(f)))$.

The resource flows in Figure 4 may be represented in the resource profile shown in Figure 5, where the use of the 10 resource units is now shown along the 10 horizontal bands.

Leus and Herroelen [29] have shown that for any realizable flow $f$, $X = \mathcal{X}(f)$ defines a realizable *early start policy* (*ES*-policy). In order to obtain a feasible schedule for a given scenario of activity durations, an *ES*-policy simply computes earliest activity start times in a graph by performing a forward CPM (longest path) pass (Stork [36]). The idea behind an *ES*-policy (Igelmund and Radermacher [17]) is to extend the partially ordered set $G(N,A)$ to

FIGURE 5. Resource profile showing the resource transfers.



a partially ordered set $G(N, A \cup X)$ such that no so-called *forbidden sets*[1] remain precedence unrelated and can thereby not be scheduled in parallel. The feasibility condition for the policy is that $G(N, A \cup X)$ still be acyclic. The arc $(1,3)$ in Figure 4, for example, guarantees that the three activities of the forbidden set $\{1,2,3\}$ cannot be scheduled in parallel.

***A Branch-and-Bound Algorithm.*** Leus and Herroelen [29] impose the restriction that a resource allocation be compatible with a precomputed baseline schedule $\mathcal{S}^B$. Let $s_i(\mathcal{S}^B)$ denote the planned start time of activity $i$ in the baseline schedule $\mathcal{S}^B$ so that $C_i(\mathcal{S}^B) = s_i(\mathcal{S}^B) + p_i$ denotes its corresponding planned completion time. Let $\Lambda(\mathcal{S}^B) = \{(i,j) \in N \times N \mid (i,j) \notin TA, i \neq j, C_i(\mathcal{S}^B) \leq s_j(\mathcal{S}^B)\}$. A feasible flow $f$ is said to be compatible with a feasible baseline schedule $\mathcal{S}^B$, written $f \sim \mathcal{S}^B$, if $\forall (i,j) \in TA \cup \mathcal{X}(f), C_i(\mathcal{S}^B) \leq s_j(\mathcal{S}^B)$, or in other words $\mathcal{X}(f) \subseteq \Lambda(\mathcal{S}^B)$. One should attempt to respect the baseline schedule as much as possible: During project execution, activities are started at the maximum of the finish times of their predecessors and their baseline starting time (often referred to as *railway scheduling*). As such, the actual starting time of activity $i$ is a stochastic variable $S_i(P, \mathcal{X}(f), \mathcal{S}^B) = \max\{s_i(\mathcal{S}^B), \max_{j \in \pi_{TA \cup \mathcal{X}(f)}(i)}\{S_j(P, \mathcal{X}(f), \mathcal{S}^B) + P_j\}\}$, with $s_0(\mathcal{S}^B) = 0$. Following the logic of Equation (6), the authors then aim at generating a feasible flow $f$ with $\mathcal{X}(f) \subseteq \Lambda(\mathcal{S}^B)$ such that

$$E\left[\sum_{i \in N} w_i \times (S_i(P, \mathcal{X}(f), \mathcal{S}^B) - s_i(\mathcal{S}^B))\right] \equiv g(\mathcal{X}(f)) \tag{9}$$

is minimized. Minimizing the expected makespan is the special case $w_i = 0$, $i \neq n$, and $w_n \neq 0$.

The set of decision variables is the set of flows $f_{ij}$ with $(i,j) \in F = TA \cup \Lambda(\mathcal{S}^B)$. For $(i,j) \in F$, denote by $B_{ij}$ the domain initially associated with $f_{ij}$. The objective is to minimize the expression in Equation (9) subject to the constraints given by Equations (7)–(8) and the requirement that $f \sim \mathcal{S}^B$. Also, $G_{\mathcal{X}(f)}$ is acyclic because arc $(i,j) \in TA \cup \Lambda(\mathcal{S}^B)$ has $C_i(\mathcal{S}^B) \leq s_j(\mathcal{S}^B) \leq C_j(\mathcal{S}^B)$ because the baseline schedule $\mathcal{S}^B$ is feasible. For $f_{ij} \in F$, $B_{ij}$ can be represented by its minimal value $LB_{ij}$ and its maximal value $UB_{ij}$. The domains are represented as intervals. The branch-and-bound algorithm implicitly evaluates all the valid flow values and relies on constraint propagation in order to reduce the search space. Leus and Herroelen [29] find an optimal resource allocation for a schedule $\mathcal{S}^B$ by considering all subsets $\Omega \subseteq \Lambda(\mathcal{S}^B)$ that allow a feasible flow in network $TA \cup \Omega$. One such subset corresponds with at least one and mostly multiple feasible $f$, with $\mathcal{X}(f) \subseteq \Omega$. They iteratively add arcs of $\Lambda(\mathcal{S}^B)$ to $\Omega$ until a feasible flow is attainable.

At each node $k$ in the search tree the set $F = TA \cup \Lambda(\mathcal{S}^B)$ is partitioned into three disjoint subsets: $F = \alpha_k \cup \nu_k \cup \omega_k$, with $\alpha_k = \{(i,j) \in F, LB_{ij} > 0\}$ the set of included arcs, $\nu_k = \{(i,j) \in F, UB_{ij} = 0\}$ the set of forbidden arcs, and $\omega_k = \{(i,j) \in F, LB_{ij} = 0 \text{ and } UB_{ij} > 0\}$ the set of undecided arcs. Bounds $LB_{ij}$ and $UB_{ij}$ are established through constraint propagation in conjunction with branching conditions. All arcs in $\alpha_k \backslash TA$ are added to $\Omega_k$ which results in the partial network $G_k = G_{\Omega_k}$. If a feasible flow cannot be obtained in $G_k$, no further branching is needed, otherwise no further arcs must be added to the network and the procedure *backtracks* after having checked whether the objective function value corresponding with the current feasible solution improves on the objective function value of the best known feasible solution. The branching decision entails the selection of an undecided arc $(i,j) \in \Lambda(\mathcal{S}^B) \cap \omega_k$: the left branch is to set $LB_{ij} = 1$, so to include $(i,j)$ in the partial network $G_k$. The right branch is to impose $UB_{ij} = 0$, so to forbid any flow across $(i,j)$, and prohibit inclusion of $(i,j)$ in $\Omega$ by placing the arc into set $\nu_k$. The result of the addition of a constraint is to split up the flow domain into two disjoint subsets, one of which is singleton $\{0\}$.

[1] A forbidden set is defined as a set of precedence unrelated activities, which cannot be scheduled together due to the resource constraints.

The authors report on promising computational results on randomly generated instances up to 61 activities (79% of the 61 activity instances are solved to optimality in an average CPU time of 45.5 seconds on a 800 MHz PC). Extension of the algorithm to multiple resource types would require a revision of the branching decisions taken by the branch-and-bound procedure and the consistency tests involved in the constraint propagation.

***Suboptimal Algorithms.*** For the general *multiresource-type case*, Deblaere et al. [8] derive lower bounds on scheduling stability and develop and validate three integer programming-based resource-allocation heuristics and one constructive procedure against the flow generation algorithm of Artigues et al. [3] and three algorithms developed by Policella et al. [33]. Overall excellent results have been obtained using the constructive procedure MABO (myopic activity-based optimization). The procedure is myopic because the authors do not look at other activities while deciding on the best possible resource allocation for an activity. MABO consists of three steps which have to be executed for each activity $j$. Step 1 examines whether the current predecessors of activity $j$ may release sufficient resource units to satisfy the resource requirements of activity $j$. If not, extra predecessors are added in Step 2 with a minimal impact on stability. Step 3 then defines resource flows $f_{ijk}$ from predecessor activities $i$ to activity $j$ for renewable resource type $k$. The detailed steps of the procedure can be written as follows:

Initialize: $A_R = A_U$ and $\forall k$: $alloc_{0k} = a_k$
Sort the project activities by increasing $s_j$ (tie break: decreasing $w_j$)
  Take next activity $j$ from list
    1. Calculate $Avail_{jk}(A \cup A_R) = \sum_{\forall i:\,(i,j)\in A\cup A_R} alloc_{ik}$ for each $k$
    2. If $\exists k$: $Avail_{jk}(A \cup A_R) < r_{jk}$
        2.1 Define the set of arcs $H_j$
            with $(h,j) \in H_j \iff$
            $(h,j) \notin A \cup A_R$
            $s_h + p_h \leq s_j$
            $\exists k$: $alloc_{hk} > 0$
        2.2 Find a subset $H_j^*$ of $H_j$
            such that $\forall k$: $Avail_{jk}(A \cup A_R \cup H_j^*) >= r_{jk}$
              and $Stability\_cost(A \cup A_R \cup H_j^*)$ is minimized
        2.3 Add $H_j^*$ to $A_R$
    3. Allocate resource flows $f(i,j,k)$ to the arcs $(i,j) \in (A \cup A_R)$:
        For each resource type $k$:
          3.1 Sort predecessors $i$ of $j$ by:
          Increasing number of successors $l$ of $i$
            with $s_l > s_j$ and $r_{lk} > 0$
          Tie break 1: Decreasing finish times $s_i + p_i$
          Tie break 2: Decreasing variance $\sigma_i^2$ of $P_i$
          Exception: Activity 0 is always put last in the list
        3.2 While $alloc_{jk} < r_{jk}$
            Take next activity $i$ from the list
              $f(i,j,k) = \min(alloc_{ik}, r_{jk} - alloc_{jk})$
              Add $f(i,j,k)$ to $alloc_{jk}$
              Subtract $f(i,j,k)$ from $alloc_{ik}$

Let us discuss the various steps of the algorithm in more detail. Let $G = (N, A \cup A_R)$ denote the resource flow network, where $A$ denotes the set of arcs representing the original precedence relations and the resource arcs $A_R$ are connecting two nodes $i$ and $j$ if there is a resource flow $f(i,j,k)$ of any resource type $k$ between the corresponding activities $i$ and $j$. In the initialization step, the set of resource arcs $A_R$ is initialized to the set of unavoidable

arcs $A_U \subset A_R$. Two activities $i$ and $j$ must be connected by an *unavoidable resource arc* in the resource flow network for a given input schedule, if the schedule causes an unavoidable strictly positive amount of resource units $f(i,j,k)$ of some resource type $k$ to be sent from activity $i$ to activity $j$. The conditions to be satisfied by activities $i$ and $j$ can be formally specified as follows:

$$\forall i \in N; \forall j \in N \text{ with } s_j \geq s_i + p_i\text{:}$$
$$(i,j) \in A_U \iff \exists k\text{: } a_k - \sum_{l \in \mathcal{A}_{s_j}} r_{lk} - \max\left(0, r_{ik} - \sum_{z \in Z} r_{zk}\right) < r_{jk} \tag{10}$$

in which $\mathcal{A}_{s_j}$ is the set of the activities that are in progress at time $s_j$ and $Z$ is the set of activities that have a baseline starting time $s_z$: $s_i + p_i \leq s_z < s_j$. The left-hand side of Equation (10) identifies the number of resource units of type $k$ that can be maximally supplied to activity $j$ at time $s_j$ from other activities than activity $i$. If this number is smaller than $r_{jk}$, there is an unavoidable resource flow between $i$ and $j$. The exact amount and resource type of the flows on the unavoidable resource arc are irrelevant at this time. We are only interested in the fact that an arc $(i,j)$ must be added to the set of unavoidable resource arcs $A_U$.

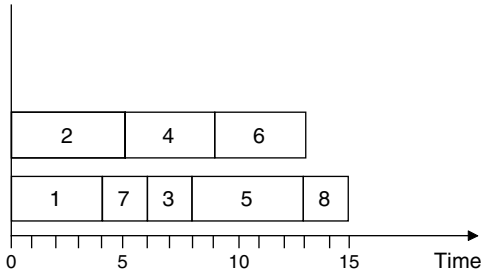
Figure 6 shows an alternative minimum duration schedule for the project example of Figure 1 (Van de Vonder [37]). This schedule requires an unavoidable resource arc from activity 7 to activity 3. At time $s_3 = 6$ only activity 4 is in progress with $r_4 = 4$. Because activity 3 starts when activity 7 ends, $Z$ is obviously void. This results in the left-hand side of Equation (10) being equal to $10 - 4 - \max(0, 3 - 0) = 3$, which is smaller than $r_3 = 4$. Arc $(7,3)$ should thus be added to $A_U$. The complete set of unavoidable arcs for the schedule in Figure 6 is $A_U = \{(0,1),(0,2),(1,7),(7,3),(3,5),(4,6),(6,8),(8,9)\}$.

For each resource type $k$, the number of resource units $alloc_{0k}$ that may be transferred from the dummy start activity 0 is initialized to the resource availability $a_k$. The project activities are placed on a list in increasing order of their planned starting times using decreasing activity weight as tie-break rule.

Step 1 computes the amount of resource units $Avail_{jk}(A \cup A_R)$ currently allocated to the predecessors of activity $j$ in $A \cup A_R$.

If this amount of available resource units is not sufficient for any resource type $k$, new precedence constraints have to be added to $A_R$ in Step 2. The set $H_j$ is defined as the set of all possible arcs between a possible resource supplier $h$ of the current activity $j$ and $j$ itself. By solving a small recursion problem, the subset $H_j^*$ of $H_j$ is found that accounts for the missing resource requirements of $j$ for any resource type $k$ at a minimum stability cost $Stability\_cost(A \cup A_R \cup H_j^*)$. The stability cost $Stability\_cost(A \cup A_R \cup H_j^*)$ is the average stability cost $\sum_{j \in N} w_j E|S_j - s_j|$, computed through simulation of sufficient executions of the (partial) schedule, keeping the resource flows fixed, and respecting the additional precedence constraints $A_R \cup H_j^*$ that were not present in the original project network diagram. The set

FIGURE 6. Alternative minimum duration schedule for the project of Figure 1.



of arcs $H_j^*$ is added to $A_R$ such that the updated $Avail_{jk}(A \cup A_R) \geq r_{jk}$ and the resource-allocation problem for the current activity is solved in a myopic way.

In Step 3, the actual resource flows $f(i,j,k)$ are allocated to the predecessors of $j$ in $A \cup A_R$ and $alloc_{ik}$, the number of resource items allocated to each activity, is updated. If $Avail_{jk}(A \cup A_R) > r_{jk}$ for resource type $k$, it has to be decided which predecessors account for the resource flows. The predecessors $i$ are sorted by increasing number of their not yet started successors $l$ with $r_{lk} > 0$, because these successors might count on these resources to be available. Two tie-break rules are used: decreasing finish times and decreasing activity duration variances. The principle is that the predecessors earlier in the sorted list normally have a higher probability to disrupt future activities. It is advisable to consume all the resource units they release as much as possible such that their possible high impact on later activities is neutralized. This allocation procedure is redone for every resource type $k$ independently. After all this, the three-step procedure is restarted for the next activity in the list until a complete feasible resource allocation is obtained at the end of the list. The procedure uses an optimal recursion algorithm for each activity, but is not necessarily optimal over all activities.

As an illustration, we run MABO on the minimum duration schedule of Figure 6. The activity weights are $w_0 = 0$, $w_1 = 2$, $w_2 = 7$, $w_3 = 4$, $w_4 = 5$, $w_5 = 3$, $w_6 = 7$, $w_7 = 5$, $w_8 = 5$, and $w_9 = 38$. Because the problem instance has a single resource type, we omit the index $k$. We start by ordering the activities, yielding the list $(0,2,1,7,4,3,5,6,8,9)$. All available resource units are allocated to the dummy start activity ($alloc_0 = 10$).

Activity 2 is next on the list. It has dummy activity 0 as single predecessor, so that $Avail_2 = alloc_0 = 10$. As $Avail_2 > r_2$ $(10 > 3)$, no extra precedence relations have to be added and we can proceed to Step 3. We set $f(0,2) = \min(alloc_0, r_2 - alloc_2) = 3$, $alloc_0 = 7$, and $alloc_2 = 3$.

Also activity 1 poses no problems because its only predecessor (activity 0) still has 7 transferrable resource units and $r_1 = 5$. Thus, $f(0,1) = 5$, $alloc_0 = 2$, $alloc_1 = 5$.

Activity 7 is the next activity on the list and we calculate in Step 1 that $Avail_7((1,7)) = alloc_1 = 5$ while $r_7 = 3$. Step 2 can thus again be skipped and the algorithm decides in Step 3 that $f(1,7) = \min(alloc_1, r_7 - alloc_7) = 3$, $alloc_1 = 2$, and $alloc_7 = 3$.

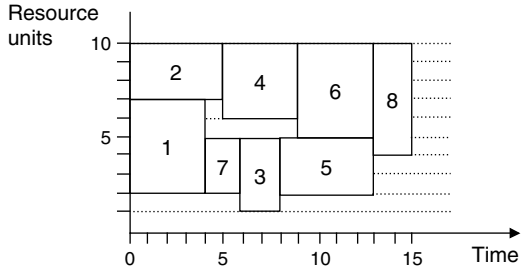
Activity 4 is next. $Avail_4((1,4),(2,4)) = 2 + 3 = 5$ while $r_4 = 4$. Step 3 gives priority to activity 2, because neither activity 1 nor activity 2 has any not yet started predecessor left, but activity 2 ends later in the baseline schedule and is thus a greater stability threat for activities further down the list. This results in $f(2,4) = \min(alloc_2, r_4 - alloc_4) = 3$ and $alloc_4 = 3$ and $alloc_2 = 0$. Then $f(1,4) = \min(alloc_1, r_4 - alloc_4) = 1$, $alloc_1 = 1$, and $alloc_4 = 4$. Activities 3 and 5 are processed in a similar way.

When activity 6 is looked at, the current situation is $alloc_0 = 1$, $alloc_1 = 1$, $alloc_3 = 1$, $alloc_4 = 4$, and $alloc_5 = 3$, resulting in $Avail_6(4,6) = alloc_4 = 4$, which is smaller than $r_6 = 5$. Thus, for the first time, an extra precedence relationship has to be added to supply one resource unit from $H_6 = \{(0,6),(1,6),(3,6)\}$. Two subsets of $H_6$, namely $(0,6)$ and $(1,6)$ can resolve the resource-allocation problem for activity 6 without extra cost. This is no surprise because both 0 and 1 are already transitive predecessors of 6. Activity 1 is selected to supply the missing resource unit and thus $(1,6)$ is added to $A_R$. The procedure then moves on, until a complete feasible resource allocation is found.

Figure 7 shows the resource profile for the obtained robust resource allocation. The horizontal bands again define the resource flow between the activities.

**3.2.2. Buffer Insertion.** Van de Vonder [37] and Van de Vonder et al. [38]–[43] have developed several exact and suboptimal procedures for inserting time buffers in an input schedule under the objective of minimizing the stability cost function $\sum_{i \in N} w_i E|S_i - s_i|$, defined earlier in Equation (6). The included time buffers are idle periods (gaps) in the schedule that should act as cushions to prevent propagation of a disruption throughout the schedule.

FIGURE 7. MABO resource allocation for the schedule in Figure 6.




***Exact Algorithm.*** Van de Vonder [37] has developed a depth-first branch-and-bound algorithm. The algorithm accepts as input an unbuffered schedule and accompanying resource allocation that will be preserved during schedule execution. An upper bound $UB$ on the stability cost can be computed using a heuristic, for example the heuristic described in the next section. During the preprocessing phase, the activities are listed in decreasing order of their starting time $s_j^U$ in the unbuffered input schedule $\mathcal{S}^U$ (decreasing activity number as tie break). For the input schedule of Figure 2, this would yield the list $L = (l_{[1]}, l_{[2]}, \ldots, l_{[n]}) = (9, 8, 6, 7, 5, 4, 3, 2, 1, 0)$. The set $N_{j-}$ denotes the set of activities preceding activity $j$ in the list, whereas the set $N_{j+}$ denotes the set of its successor activities in the list. Let $es_j = s_j^U$ be the earliest allowable start time of activity $j$. The project due date $d_n$ defines the latest allowable start time $ls_n$ for the dummy end activity $n$. Backward calculations in the network $G = (N, A \cup R)$ then yield the latest allowable start time $ls_j$ for each activity $j$. The total schedule float of activity $j$, the maximum amount of time by which activity $j$ may be delayed without violating the due date $d_n$, is then computed as $TF_j = ls_j - es_j$. A lower bound $stab_j^{\min}(FF_j)$ on the stability cost induced by activity $j$ when activity $j$ is preceded by a time buffer of $FF_j$ time units, is then computed by running the following algorithm:

$\forall j \in N$:
  **for** $FF_j = 0$ **to** $(ls_j - es_j)$ **do**
    Simulate sufficient scenarios of $G(N_{j+}, A \cup A_R)$ with:
      $\forall i \in N_{j+}$: $s_i = es_i$
      $s_j = es_j + FF_j$
    Calculate $stab_j^{\min}(FF_j) = w_j E|S_j - s_j|$.

The depth first search considers the activities in the order dictated by the list $L = (l_{[1]}, l_{[2]}, \ldots, l_{[n]})$. The activity under consideration is called the *current activity*. The first activity $l_{[1]}$ on the list is the dummy end activity $n$. The root node at level 0 of the search tree is generated with $s'_n = d_n$. Moving to the second current activity $c = l_{[2]}$ on the list, the left most node at level 1 of the search tree is generated with $s'_c = s_c$. The lower bound on the stability cost induced by this current activity $c$ is computed using simulation as

$$LB1_c = \sum_{l \in N_{c-}} w_l E|S_l - s_l|. \tag{11}$$

If $LB1_c \geq UB$, then the node can be fathomed, as it is not necessary to evaluate the starting times $s_i > es_i$ for any $i \in N_{c+}$, given that $s'_l$ has been fixed for each $l \in N_{c-}$. Also, it is not necessary to evaluate other starting times for $c$. Hence, the procedure *backtracks* to the previous activity in the list.

Backtracking is done by moving one level up in the search tree and investigating the next position $s'_{(c')} + 1$ for the current activity $c'$ explored at that level. If there exists no subsequent starting time of $c'$ such that $s_{c'} + p_{c'} \leq \max_{\forall m:(c',m) \in (A \cup A_R)} s'(m)$, then backtracking continues until an activity is met with feasible subsequent starting times. When backtracking reaches the root of the search tree, the algorithm stops.

If $LB1_c < UB$, a tighter lower bound $LB2_c$ can be computed. Latest starting times $ls'_i$ for $i \in N_{c+}$ are calculated given that $\forall l \in (N_{c-} \cup \{c\})$: $ls'_l = s'_l$. For any activity $i \in (N_{c+} \cup \{c\})$, the expression $w_i E|S_i - s_i| \geq stab_i^{\min}(ls'_i - es_i)$ holds for any schedule that would be generated in lower branches of the search tree.
Aggregation yields

$$\sum_{i \in (N_{c+} \cup \{c\})} w_i E|S_i - s_i| \geq \sum_{i \in (N_{c+} \cup \{c\})} stab_i^{\min}(ls'_i - es_i). \tag{12}$$

We have shown above that $\sum_{i \in N_{c-}} w_i E|S_i - s_i| \geq LB1$ holds in the current branch of the search tree. It results that

$$LB2 = LB1 + \sum_{i \in (N_{c+} \cup \{c\})} stab_i^{\min}(ls'_i - es_i) \leq \sum_{i \in N} w_i E|S_i - s_i|. \tag{13}$$

$LB2$ is thus a lower bound on the stability cost for all schedules with $s_j = s'_j$ and $\forall l \in N_{j-}$: $s_l = s'_l$. The computations made in the preprocessing stage make this bound rather easy to compute. If $LB2 \geq UB$, the current node can be fathomed and the search continues by evaluating the next larger start time for current activity $c$. If $LB2 < UB$, the current branch needs to be further explored by branching on the next activity in the ordered list. When branching has been done for all activities $i$ with $es_i > 0$ and $LB2 < UB$, a new best solution has been found with stability cost $LB2$.

Running the optimal buffer insertion algorithm on the input schedule given in Figure 2 with $d_n = 20$, activity weights $w_0 = 0$, $w_1 = 2$, $w_2 = 7$, $w_3 = 4$, $w_4 = 5$, $w_5 = 3$, $w_6 = 7$, $w_7 = 5$, $w_8 = 5$, and $w_9 = 38$, and expected activity durations $E(D_1) = 4$, $E(D_2) = 5$, $E(D_3) = 2$, $E(D_4) = 4$, $E(D_5) = 5$, $E(D_6) = 4$, $E(D_7) = 2$, and $E(D_8) = 2$, yields the robust project schedule of Figure 8.

***A Heuristic Procedure.*** Several heuristic procedures have been developed for generating stable buffered schedules under the $\sum_{i \in N} w_i E|S_i - s_i|$ objective (Van de Vonder et al. [39]). Despite the simplicity of its underlying assumptions and structure, the *starting time criticality heuristic* (STC) obtained excellent results.

The STC exploits information about both the weights of the activities and the variance structure of the activity durations. The basic idea is to start from a minimum duration schedule and iteratively create intermediate schedules by adding a one-unit time buffer in front of that activity that needs it the most in the current intermediate schedule, until adding more safety would no longer improve stability. We thus need a measure to quantify for each activity how critical its current starting time is in the current intermediate baseline schedule. The *starting time criticality* for activity $j$ is defined as $stc(j) = P(S_j > s_j) \times w_j = \gamma_j \times w_j$, where $\gamma_j$ denotes the probability that activity $j$ cannot be started at its scheduled starting time $s_j$.

The iterative procedure runs as follows. At each iteration step (see Figure 9), the buffer sizes of the current intermediate schedule are updated as follows. The activities are listed

FIGURE 8. Buffered schedule produced by branch-and-bound.

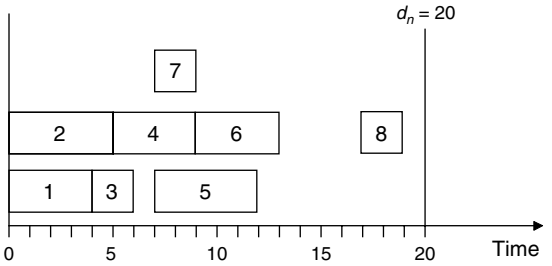


FIGURE 9. Iteration step of STC heuristic.

```
Calculate all stc(j)
Sort activities in decreasing order of the stc(j)
While no improvement found do
  Take next activity j from list
  If stc(j) := 0 then procedure terminates
  Else add buffer in front of j
    Update schedule
    If improvement and feasible then
      Store schedule
      Goto next iteration step
    Else
      Remove buffer in front of j
      Restore schedule
```

in decreasing order of the $stc(j)$. The list is scanned and the size of the buffer to be placed in front of the currently selected activity from the list is augmented by one time unit such that the starting times of the activity itself and of the direct and transitive successors of the activity in $G(N, A \cup R)$ are increased by one time unit. If this new schedule has a feasible project completion ($s_n < d_n$) and results in a lower approximated stability cost ($\sum_{j \in N} stc(j)$), the schedule serves as the input schedule for the next iteration step. If not, the next activity in the list is considered. Whenever an activity $j$ is encountered for which $stc(j) = 0$ (all activities $j$ with $s_j = 0$ are by definition in this case) and no feasible improvement is found, a local optimum is obtained and the procedure terminates.

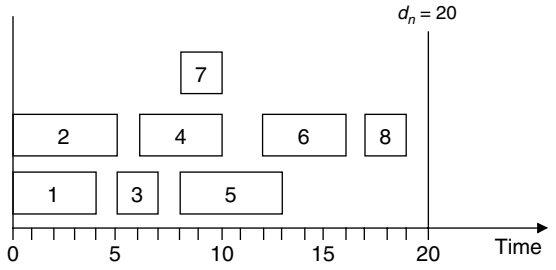
Regrettably, the probabilities $\gamma_j$ are not easy to compute. The authors define $k(i, j)$ as the event that predecessor $i$ disturbs the planned starting time of activity $j$. The probability that this event occurs can be expressed as $P(k(i, j)) = P(S_i + P_i + LPL(i, j)) > s_j)$, in which $LPL(i, j)$ is the sum of the durations of all activities $h$ on the longest path between activity $i$ and activity $j$ in the graph $G(N, A \cup R)$. We can now calculate $\gamma_j = P(\bigcup_i k(i, j))$ for $\forall i$: $(i, j) \in T(A \cup R)$.

STC makes two assumptions in approximating $\gamma_j$: (a) only one activity at a time disturbs the starting time of activity $j$, and (b) the predecessor $i$ of activity $j$ starts at its originally planned starting time. Assumption (a) means that $P(\bigcup_i k(i, j))$ is estimated by $\sum_i P(k(i, j))$, i.e., it is assumed that $P(k(i1, j) \cap k(i2, j) = 0$ for each $i1$, $i2$. Assumption (b) boils down to setting $S_i = s_i$. Combining both assumptions yields $\gamma'_j = \sum_i P(P_i > s_j - s_i - LPL(i, j))$ such that $stc(j) = \gamma'_j \times w_j$. Because $s_i$, $s_j$, $LPL(i, j)$ and the distribution of the $P_i$ are all known, the values of $\gamma'_j$ and $stc(j)$ can be easily computed for every activity $j$.

Assuming activity weights $w_0 = 0$, $w_1 = 2$, $w_2 = 7$, $w_3 = 4$, $w_4 = 5$, $w_5 = 3$, $w_6 = 7$, $w_7 = 5$, $w_8 = 5$, and $w_9 = 38$, and expected activity durations $E(D_1) = 4$, $E(D_2) = 5$, $E(D_3) = 2$, $E(D_4) = 4$, $E(D_5) = 5$, $E(D_6) = 4$, $E(D_7) = 2$, and $E(D_8) = 2$, the application of the STC algorithm to the input schedule of Figure 2 runs as follows. We illustrate the calculation of the $LPL(i, j)$ for $LPL(1, 3)$, $LPL(1, 5)$, and $LPL(1, 8)$. Activity 3 is immediately preceded by activity 1 in the schedule, hence $LPL(1, 3) = 0$. The resource flow network of Figure 4 shows a unique path $\langle 1, 3, 5 \rangle$ leading from activity 1 to activity 5, yielding $LPL(1, 5) = E(D_3) = 2$. Multiple paths exist between activity 1 and activity 8, namely $\langle 1, 3, 5, 8 \rangle$, $\langle 1, 3, 7, 8 \rangle$, $\langle 1, 3, 7, 6, 8 \rangle$, $\langle 1, 7, 8 \rangle$, $\langle 1, 7, 6, 8 \rangle$, and $\langle 1, 4, 6, 8 \rangle$, with corresponding path length of $7, 4, 8, 4, 2, 6$, and 8 respectively. Thus, $LPL(1, 8) = 8$.

The *stc*-values are first calculated for the initial minimum duration schedule of Figure 2 with due date $d_n = 20$. For example, $stc(6)$ is calculated as $w_6 \times (P(k(1, 6)) + P(k(2, 6)) + P(k(3, 6)) + P(k(4, 6)) + P(k(7, 6)))$, with $P(k(1, 6) = P(P_1 > s_6 - s_1 - LPL(1, 6)) = P(P_1 > 5) = 0.11$, $P(k(2, 6)) = P(P_2 > s_6 - s_2 - LPL(2, 6)) = P(P_2 > 5) = 0.23$, $P(k(3, 6)) = P(P_3 > s_6 - s_3 - LPL(3, 6)) = P(P_3 > 3) = 0.01$, $P(k(4, 6)) = P(P_4 > s_6 - s_4 - LPL(4, 6)) = P(P_4 > 4) = 0.34$, and $P(k(7, 6)) = P(P_7 > s_6 - s_7 - LPL(7, 6)) = P(P_7 > 5) = 0.05$. As a

FIGURE 10. STC schedule for the example problem.




result, $stc(6) = 7 \times (0.11 + 0.23 + 0.01 + 0.34 + 0.05) = 5.18$. Computing the other *stc*-values yields $\sum stc(j) = 0 + 0 + 1.20 + 1.71 + 1.47 + 5.18 + 2.45 + 4.88 + 0.04 = 16.93$ as the total schedule cost. Ordering the activities by decreasing *stc* gives the list $(6, 8, 7, 4, 5, 3, 9, 1, 2)$. Adding a one-time period buffer in front of activity 6 yields updated start times $s_6 = 10$ and $s_8 = 14$. The newly inserted buffer in front of activity 6 requires a recalculation of its *stc*-value and the *stc*-values of its successor activities 8 and 9. Activity 7 now has the largest *stc*-value, yielding the ordered list $(7, 8, 4, 6, 5, 3, 9, 1, 2, 0)$. Delaying the starting time of activity 7 is feasible and leads to a reduction in the total schedule cost $\sum stc(j) = 0 + 0 + 1.20 + 1.71 + 1.47 + 1.69 + 2.45 + 2.22 + 0.04 = 10.78$. Subsequently, the procedure will insert a time buffer in front of activity 8, activity 6, activity 4 (leading to a one-period delay in activity 6 and 8), activity 5, activity 3 (leading to a one-period delay of activities 5 and 7). The procedure then continues by examining a possible delay of activity 6. Delaying activity 6, however, would lead to an increase in the total schedule cost. Therefore, activity 6 is not delayed. In a similar fashion, delaying activity 3 or either of the following activities in the list would yield no cost improvement. The procedure terminates with the schedule shown in Figure 10 (Van de Vonder [37]).

### 3.3. Reactive Scheduling

Proactive/reactive scheduling implies that the buffered baseline schedules generated by the proactive procedures should be combined with reactive scheduling procedures that are deployed during project execution when disruptions occur that cannot be absorbed by the baseline schedule.

The literature concerning reactive project scheduling is virtually void. Yu and Qi [48] describe an ILP model for the multimode RCPSP (problem $m, 1T|cpm, disc, mu|C_{\max}$ in the notation of Herroelen et al. [15]) and report on computational results obtained by a hybrid mixed integer programming/constraint propagation approach for minimizing the schedule deviation caused by a single disruption induced by a known increase in the duration of a single activity. Van de Vonder et al. [40] developed and extensively evaluated a number of exact and heuristic reactive procedures.

The reactive scheduling problem at decision point $t$ when the baseline schedule $\mathcal{S}^B$ breaks, can be viewed as a resource-constrained project scheduling problem with weighted earliness tardiness costs (problem $m, 1|cpm|early/tardy$ in the notation of Herroelen et al. [15]). Due dates are set equal to the activity completion times $s_i + p_i$ in the predictive schedule. The earliness and tardiness costs may be assumed to be symmetrical and chosen as the weights $w_i$ in the stability objective function, with a possible exception for the earliness cost of the dummy end activity, which can be set equal to zero.

Efficient exact procedures for solving problem $m, 1|cpm|early/tardy$ have been proposed in the scheduling literature (see e.g., Vanhoucke et al. [44] and Kéri and Kis [18]). However, Van de Vonder et al. [40] found that calling an exact weighted earliness-tardiness procedure at each schedule breakage point becomes computationally infeasible already for small networks. The authors obtained excellent computational results with a sampling approach.

The basic *sampling approach* by Van de Vonder et al. [40] relies on different priority lists in combination with different schedule generation schemes. It tries to make a suitable decision at each decision time $t$ as follows ($\mathcal{S}^0$ is the baseline schedule):

**for** $t = 0, \ldots, T$ **do**
Step 1: Check for new scheduling information.
Step 2: **If** no new information **then** $\mathcal{S}^t = \mathcal{S}^{t-1}$ and **goto** period $t+1$
**else** goto step 3
Step 3: **For** list $\lambda_l = \lambda_0, \ldots, \lambda_L$ **do**
Construct $\mathcal{S}^t_{\lambda_l, RP}$ and calculate $\Delta(\mathcal{S}^0, \mathcal{S}^t_{\lambda_l, RP})$
Construct $\mathcal{S}^t_{\lambda_l, RS}$ and calculate $\Delta(\mathcal{S}^0, \mathcal{S}^t_{\lambda_l, RS})$
Construct $\mathcal{S}^t_{\lambda_l, P}$ and calculate $\Delta(\mathcal{S}^0, \mathcal{S}^t_{\lambda_l, P})$
Construct $\mathcal{S}^t_{\lambda_l, S}$ and calculate $\Delta(\mathcal{S}^0, \mathcal{S}^t_{\lambda_l, S})$
Store the schedule $\mathcal{S}^t$ that minimizes $\Delta(\mathcal{S}^0, \mathcal{S}^t)$
Step 4: Start all activities $i$ with $s_i^t = t$.

*Step* 1 checks for new information becoming available at time $t$. If at time $t$, no activity finishes and no activity was projected to finish ($s_i^{t-1} = t$), then no new information became available since the previous decision point $t-1$. The previous projected schedule $\mathcal{S}^{t-1}$, generated at time $t-1$, remains valid (*Step* 2).

Instead of using one priority list in combination with one schedule generation scheme (SGS), *Step* 3 uses multiple lists $\lambda_l = \lambda_0, \ldots, \lambda_L$ at time $t$ in combination with several SGSs.

The authors evaluate different priority lists: EBST (earliest start time in the baseline schedule), LST (latest starting time), LW (largest activity weight), LAN (lowest activity number), RND (random), EPST (earliest starting time in the schedule generated at the last decision point), and MC (lowest current stability cost).

For each of these priority lists $\lambda_l$, a complete schedule is generated using four schedule generation schemes. The *parallel schedule generation scheme* ($\mathcal{S}^t_{\lambda_l, P}$) iterates over time and starts at each decision point, in the order dictated by the priority list, as many unscheduled activities as possible in accordance with the precedence and resource constraints. The *robust parallel schedule generation scheme* ($\mathcal{S}^t_{\lambda_l, RP}$) is similar to the parallel scheme, but considers at each decision time $t$ only the activities for which the current decision time $t$ is greater than or equal to their planned starting time in the baseline schedule. The *serial schedule generation scheme* ($\mathcal{S}^t_{\lambda_l, S}$) schedules at each decision point $t$ the next activity from the priority list. The *robust serial schedule generation scheme* ($\mathcal{S}^t_{\lambda_l, RS}$) considers the activities in the order dictated by the priority list and starts them at a feasible time as close as possible to their planned starting time in the baseline schedule.

In this way, a total of $4 \times L$ candidate schedules are generated and the schedule $\mathcal{S}^t_{\lambda_l, .}$ yielding the smallest stability cost deviation $\Delta(\mathcal{S}^0, \mathcal{S}^t_{\lambda_l, .})$ from the baseline schedule $\mathcal{S}^0$ is stored. The procedure then continues in *Step* 4 by starting the activities for which the planned starting time in the schedule equals $t$.

## 4. Solution Robust Scheduling Under Resource Availability Uncertainty

The literature on proactive/reactive project scheduling under resource availability uncertainties is virtually void. Drezet [10] considers the problem of project planning subject to human-resource constraints, which have to do with job competences, working hour limits, vacation periods, and unavailability of employees. She presents a mathematical model as well as dedicated algorithms for robust schedule generation and schedule repair. Yu and Qi [48], in their above-mentioned ILP model for the multimode problem, allow for (known) decreases in the resource availabilities in certain planning periods.

### 4.1. The Problem

Contrary to §3, activity durations are now assumed to be deterministic; uncertainty originates from the stochastic nature of renewable resource availability $A_k$ $(k = 1, \ldots, q)$. This means that during schedule execution infeasibilities may occur due to renewable resource breakdowns, so that the schedule needs to be repaired. The proactive project scheduling problem then consists of generating a proactive schedule that is as well as possible protected from such disruptions, subject to a project deadline, finish-start, zero-lag precedence constraints and renewable resource constraints.

A maximum resource availability $a_k$ is considered for each renewable resource type $k$. Each of these $a_k$ resource units initially allocated to the project is subject to breakdowns; some of the proposed models presuppose knowledge of the mean time to failure and the mean time to repair. The objective function to be minimized is again $\sum_{i \in N} w_i E(|S_i - s_i|)$, the weighted expected deviation between the planned and actually realized activity start times.

Lambrechts et al. [22] develop and evaluate eight proactive and three reactive scheduling procedures. A tabu search procedure for generating robust baseline schedules is presented in Lambrechts et al. [23]. In Lambrechts et al. [24], the authors analytically determine the impact of unexpected resource breakdowns on activity durations and develop effective and efficient algorithms for inserting explicit idle time into an initial unbuffered input schedule. Reactive strategies are discussed in Lambrechts et al. [25].

**4.1.1. Proactive Strategies.** The two-step proactive scheduling procedure (Lambrechts et al. [24]) for generating stable baseline schedules may take as input an unbuffered schedule generated by an exact or heuristic procedure for solving the deterministic RCPSP, or by a procedure that places activities that can be expected to have a high impact on the total project stability earliest in time (*largest CIW first*). Furthermore, it can be decided to include resource slack (*resource buffering*).

Resource buffering boils down to planning the project subject to a deterministic nominal resource availability that is strictly below the maximum deterministic resource availability $a_k$. More precisely, the nominal availabilities are set equal to $E(A_k) = \sum_{m=0}^{a_k} (a_k - m)\pi_m$, where $\pi_m$ denotes the steady state probability that $m$ resource units of resource type $k$ are inactive. When necessary, this value is increased to $\max_{i \in N} r_{ik}$ to allow for the activity with the highest resource demand for resource type $k$ to be scheduled.

The largest-CIW-first rule schedules the activities $i$ in nonincreasing order of their cumulative instability weight $CIW_i = w_i + \sum_{j \in Succ_i} w_j$, where $Succ_i$ denotes the set of direct and indirect successors of activity $i$. In a first step a precedence feasible priority list is constructed in which precedence-unrelated activity pairs appear in nonincreasing order of $CIW_i$ (lowest activity number as tie breaker). Afterwards this priority list is transformed into a precedence and resource feasible schedule using a serial schedule generation scheme that 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.

Time buffers can then be inserted into the unbuffered schedule in order to increase its stability. *Time buffering* implies that time buffers are inserted in front of activities in order to absorb potential disruptions caused by earlier resource breakdowns and the resulting activity shifts. The input schedule may be iteratively buffered using a simulation-based steepest descent procedure (Lambrechts et al. [24]). In each iteration, every activity (except the dummy start) is considered for buffering. The selected activity is then 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. The activity leading to the greatest stability cost reduction (determined by simulation) that yields a schedule respecting the deadline is buffered. If no such activity can be found, the procedure terminates.

Such a simulation-based procedure is very time consuming. Surrogate measures can be used to estimate the instability costs. Lambrechts et al. [24] conclude that for the preempt-repeat case, in which an interrupted activity must be restarted later, the best results are obtained by computing the surrogate measure as $\sum_{j\in N}\sum_{i\in Pred_j} w_j \max(0, s_i+p_i+LPL_{ij}+E[\sigma_i]-s_j)$, where $Pred_j$ denotes the immediate and transitive predecessors of activity $j$, $LPL_j$ represents the length of the longest path between activities $i$ and $j$ in $G(N, A\cup R)$, and $E[\sigma_i]$ denotes the expected duration extension of activity $i$ caused by the resource breakdown.

Lambrechts et al. [24] conclude from their computational experiment, assuming exponential or uniform repair time distributions, and combining the proactive procedure with the scheduled order reactive procedure (§4.1.2), that simulation-based time buffering always outperforms the time-buffering approaches that use surrogate stability cost estimates. However, its computational requirements are prohibitive. In the preempt-resume case, when interrupted activities may be resumed on repair of the broken resource(s), and in the preempt-setup case, when a setup time is needed when activities are resumed, best results are obtained using the STC heuristic described earlier. In the absence of resource and time buffering, the largest CIW scheduling rule outperforms the use of a minimum makespan input schedule. Resource buffering always pays off.

**4.1.2. Reactive Strategies.** When the baseline schedule breaks, i.e., when activities have to be interrupted because of a resource breakdown, the schedule needs to be repaired using a reactive procedure. Lambrechts et al. [22] investigate a preempt-repeat setting (interrupted activities have to be restarted anew); they generate a list $L$ containing all activities that are not yet completed at the time of interruption, ordered in nondecreasing order of the baseline starting times. This list is then decoded into a feasible schedule using a serial schedule generation scheme that tries to schedule the interrupted activities as early as possible; a tabu search algorithm to improve the generated reactive schedule is also proposed in the same source.

Lambrechts et al. [25] study exact and suboptimal procedures to restore schedule feasibility under the objective of minimizing the weighted sum of deviations between the repaired schedule and the baseline schedule, under the assumption that the encountered disruption is the last disruption until project completion. The exact algorithm relies on the (truncated) branch-and-bound algorithm of Vanhoucke et al. [44] for solving the resulting resource-constrained project scheduling problem with weighted earliness tardiness costs (problem $m,1|cpm|early/tardy$ in the notation of Herroelen et al. [15]). They also present a *scheduled order list scheduling heuristic* that allows to reschedule the activities in the order dictated by the baseline schedule (the lowest activity number being the tie breaker) although taking into account the new, reduced resource availabilities. They obtain improved solutions by imposing a tabu search procedure on the priority list rule.

Lambrechts et al. [25] extend the tabu search procedure allowing it, when a disruption occurs, not only to generate a repaired baseline schedule that does not deviate too much from the original baseline, but a repaired schedule that is also protected against the occurrence of future disruptions. They use a surrogate robustness measure based on the expected duration increase of an activity due to resource breakdowns.

## 5. Conclusions

Real-life projects are typically subject to considerable uncertainty. This chapter has addressed proactive/reactive project scheduling procedures that may be deployed when the uncertainty pertains to the duration of activities or to the availability of renewable resources. *Proactive* procedures have been described to generate a robust baseline schedule that is appropriately protected against distortions that may occur during project execution. The term "robustness" in this context refers to solution robustness or stability. Our aim has

been to generate proactive precedence and resource feasible baseline schedules that minimize one particular stability cost function, namely the weighted sum of the expected deviation between the actually realized activity start times during project execution and the planned activity start times in the baseline. When distortions during project execution cause the baseline schedule to become infeasible, a *reactive* policy needs to be invoked to repair the schedule.

For variable activity durations, Van de Vonder et al. [37–43] have developed exact and heuristic proactive time-buffer-insertion strategies that can be combined with effective reactive policies for (optimally or heuristically) solving the underlying weighted earliness-tardiness problem. These research efforts allow to draw interesting and reassuring conclusions. It appears that the combination of proactive and reactive scheduling techniques leads to significant stability improvements (reduction in the planning nervousness), with only moderate (hence acceptable) increases in schedule makespan.

The unavailability of resources is a second potential but very realistic cause of substantial deviations from the baseline schedule. Consequently, the development of proactive/reactive scheduling procedures under stochastic resource availability is relevant from a theoretical and a practical point of view. Research in this area is just emerging. Lambrechts et al. [22–25] have obtained excellent results with their proactive/reactive procedures to cope with resource breakdowns.

These promising results justify the engagement in additional research. The development of effective and efficient single-step (monolithic) proactive scheduling procedures for the generation of stable (solution robust) baseline schedules with acceptable makespan performance, which can be easily combined with effective reactive scheduling policies that are able to operate under various types of schedule distortions, deserves priority. The exploration of robustness measures other than the weighted activity starting time deviations, which was explored in this chapter, constitutes another interesting area of future research.

## Acknowledgments

This research has been supported by Project OT/03/14 of the Research Fund of Katholieke Universiteit Leuven, Project G.0109.04 of the Research Foundation – Flanders (FWO-Vlaanderen) and Project NB06163 supported by the National Bank of Belgium. The author is very much indebted to Filip Deblaere, Erik Demeulemeester, Olivier Lambrechts, Roel Leus, and Stijn Van de Vonder. Their research results, obtained over the last few years, were indispensable for the preparation of this tutorial.

## References

[1] M. Aloulou and M.-C. Portmann. An efficient proactive scheduling approach to hedge against shop floor disturbances. *Proceedings of the First Multidisciplinary Conference on Scheduling: Theory and Applications*, Nancy, France, 337–362, 2003.

[2] C. Artigues and F. Roubellat. A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research* 127(2):297–316, 2000.

[3] C. Artigues, P. Michelon, and S. Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research* 149(2):249–267, 2003.

[4] H. Aytug, M. A. Lawley, K. McKay, S. Moan, and R. Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research* 161:86–110, 2005.

[5] J.-C. Billaut, A. Moukrim, and E. Sanlaville. *Flexibilité et robustesse en ordonnancement. Traité IC2, Série Informatique et systèmes d'information*. Hermes Science Publications, Paris, France, 2005.

[6] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling subject to resource constraints—classification and complexity. *Discrete Applied Mathematics* 5(1):11–24, 1983.

[7] 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(1):3–41, 1999.

[8] F. Deblaere, E. Demeulemeester, W. Herroelen, and S. Van de Vonder. Robust resource-allocation decisions in resource-constrained projects. *Decision Sciences* 38(1):5–37, 2007.

[9] E. Demeulemeester and W. Herroelen. *Project Scheduling—A Research Handbook. International Series in Operations Research & Management Science*, Vol. 49. Springer, Heidelberg, Germany, 2002.

[10] L.-E. Drezet. Résolution d'un problème de gestion de projets sous contraintes de ressources humaines: De l'approche prédictive à l'approche réactive. Ph.D. thesis, Université François Rabelais, Tours, France, 2005.

[11] J. N. Hagstrom. Computational complexity of PERT problems. *Networks* 18(2):139–147, 1988.

[12] W. Herroelen and R. Leus. Robust and reactive project scheduling: A review and classification of procedures. *International Journal of Production Research* 42(8):1599–1620, 2004.

[13] W. Herroelen and R. Leus. Project scheduling under uncertainty—Survey and research potentials. *European Journal of Operational Research* 165(2):289–306, 2005.

[14] W. Herroelen, B. De Reyck, and E. Demeulemeester. Resource-constrained scheduling: A survey of recent developments. *Computers and Operations Research* 25(4):279–302, 1998.

[15] W. Herroelen, B. De Reyck, and E. Demeulemeester. On the paper "Resource-constrained project scheduling: Notation, classification, models and methods" by Brucker et al. *European Journal of Operational Research* 128(3):679–688, 2001.

[16] H. Hoogeveen. Multicriteria scheduling. *European Journal of Operational Research* 167(3): 592–623, 2004.

[17] G. Igelmund and F. J. Radermacher. Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks* 13(1):29–48, 1983.

[18] A. Kéri and T. Kis. Primal-dual combined with constraint propagation for solving RCPSP-WET. *Proceedings of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications*, MISTA, New York. 748–751, 2005.

[19] N. Kinnock. Communication of the European Commission to the Council concerning the Berlaymont Building. 1094, 2002.

[20] R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *Omega* 49(3):249–272, 1999.

[21] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Kluwer Academic Publishers, Boston, MA, 1997.

[22] O. Lambrechts, E. Demeulemeester, and W. Herroelen. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of Scheduling*. Forthcoming.

[23] O. Lambrechts, E. Demeulemeester, and W. Herroelen. A tabu search procedure for developing robust predictive project schedules. *International Journal of Production Economics*. Forthcoming.

[24] O. Lambrechts, E. Demeulemeester, and W. Herroelen. Time-slack-based techniques for generating robust project schedules subject to resource uncertainty. Research report KBI, Department of Decision Sciences and Information Management (KBI), Katholieke Universiteit Leuven, Leuven, Belgium, 2007.

[25] O. Lambrechts, E. Demeulemeester, and W. Herroelen. Exact and suboptimal reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. Research report KBI 0702, Department of Decision Sciences and Information Management (KBI), Katholieke Universiteit Leuven, Leuven, Belgium, 2007.

[26] V. J. Leon, S. D. Wu, and R. H. Storer. Robustness measures and robust scheduling for job shops. *IIE Transactions* 16(5):32–43, 1994.

[27] R. Leus. The generation of stable project plans. Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 2003.

[28] R. Leus and W. Herroelen. On the merits and pitfalls of critical chain scheduling. *Journal of Operations Management* 19(5):559–577, 2001.

[29] R. Leus and W. Herroelen. Stability and resource allocation in project planning. *IIE Transactions* 36(7):667–682, 2004.

[30] R. Leus and W. Herroelen. The complexity of machine scheduling for stability with a single disrupted job. *Operations Research Letters* 33(2):151–156, 2005.

[31] S. V. Mehta and R. M. Uzsoy. Predictive scheduling of a job shop subject to breakdowns. *IEEE Transactions on Robotics and Automation* 14(3):365–378, 1998.

[32] R. H. Möhring. Scheduling under uncertainty: Bounding the makespan distribution. H. Alt, ed. *Computational Discrete Mathematics: Advanced Lectures*. Springer, New York, 2001.

[33] N. Policella, A. Oddi, and A. Cesta. Generating robust partial order schedules. *Proceedings of CP2004*. Springer, Toronto, Canada, 2004.

[34] B. Roy. Robustesse de quoi et vis-à-vis de quoi mais aussi robustesse pourquoi en aide à la décision? J. Figueira, C. Henggeler-Anthunes, J. Climaco, eds. *Proceedings of the 56th Meeting of the European Working Group on Multiple Criteria Decision Making*, Coimbra, Portugal, 2002.

[35] E. Sanlaville. Ordonnancement sous conditions changeantes—Habilitation à diriger des recherches. Ph.D. thesis, Université Blaise Pascal, Clermont-Ferrand, France, 2004.

[36] F. Stork. Stochastic resource-constrained project scheduling. Ph.D. thesis, School of Mathematics and Natural Sciences, Technical University of Berlin, Berlin, Germany, 2001.

[37] S. Van de Vonder. Proactive-reactive procedures for robust project scheduling. Ph.D thesis, Department of Decision Sciences and Information Management (KBI), Katholieke Universiteit Leuven, Leuven, Belgium, 2006.

[38] S. Van de Vonder, E. Demeulemeester, and W. Herroelen. Heuristic procedures for generating stable project baseline schedules. *European Journal of Operational Research*, 2007. Forthcoming.

[39] S. Van de Vonder, E. Demeulemeester, and W. Herroelen. An investigation of efficient and effective predictive-reactive project scheduling procedures. *Journal of Scheduling*, Forthcoming.

[40] S. Van de Vonder, F. Ballestin, E. Demeulemeester, and W. Herroelen. Heuristic procedures for reactive project scheduling. *Computers & Industrial Engineering* 52(1):11–28, 2007.

[41] S. Van de Vonder, E. Demeulemeester, W. Herroelen, and R. Leus. The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics* 97(2):227–240, 2005.

[42] S. Van de Vonder, E. Demeulemeester, W. Herroelen, and R. Leus. The trade-off between stability and makespan in resource-constrained project scheduling. *International Journal of Production Research* 44(2):215–236, 2006.

[43] S. Van de Vonder, E. Demeulemeester, R. Leus, and W. Herroelen. Proactive-reactive project scheduling—Trade-offs and procedures. J. Jozefowska and J. Weglarz, eds. *Perspectives in Modern Project Scheduling. International Series in Operations Research and Management Science*. Springer, New York, 2006.

[44] M. Vanhoucke, E. Demeulemeester, and W. Herroelen. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research* 102:179–196, 2001.

[45] G. Vieira, J. Herrmann, and E. Lin. Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling* 6(1):39–62, 2003.

[46] J. Wang. Constraint-based schedule repair for product development projects with time-limited constraints. *International Journal of Production Economics* 95(3):399–414, 2005.

[47] S. D. Wu, R. H. Storer, and P. C. Chang. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers and Operations Research* 20(1):1–14, 1993.

[48] G. Yu and X. Qi. *Disruption Management—Framework, Models and Applications*. World Scientific, Singapore, 2004.

[49] G. Zhu, J. Bard, and G. Yu. Disruption management for resource-constrained project scheduling. *Journal of the Operational Research Society* 56(4):365–381, 2005.