Decision Sciences
Volume 38 Number 1
February 2007

Robust Resource Allocation Decisions in Resource-Constrained Projects*

Filip Deblaere†
Research Center for Operations Management, K.U.Leuven, Naamsestraat 69, 3000 Leuven, Belgium, e-mail: filip.deblaere@econ.kuleuven.be

Erik Demeulemeester
Research Center for Operations Management, K.U.Leuven, Naamsestraat 69, 3000 Leuven, Belgium, e-mail: erik.demeulemeester@econ.kuleuven.be

Willy Herroelen
Research Center for Operations Management, K.U.Leuven, Naamsestraat 69, 3000 Leuven, Belgium, e-mail: willy.herroelen@econ.kuleuven.be

Stijn Van de Vonder
Research Center for Operations Management, K.U.Leuven, Naamsestraat 69, 3000 Leuven, Belgium, e-mail: stijn.vandevonder@econ.kuleuven.be

ABSTRACT

The well-known deterministic resource-constrained project scheduling problem involves the determination of a predictive schedule (baseline schedule or pre-schedule) of the project activities that satisfies the finish–start precedence relations and the renewable resource constraints under the objective of minimizing the project duration. This baseline schedule serves as a baseline for the execution of the project. During execution, however, the project can be subject to several types of disruptions that may disturb the baseline schedule. Management must then rely on a *reactive scheduling* procedure for revising or reoptimizing the baseline schedule. The objective of our research is to develop procedures for allocating resources to the activities of a given baseline schedule in order to maximize its stability in the presence of activity duration variability. We propose three integer programming–based heuristics and one constructive procedure for resource allocation. We derive lower bounds for schedule stability and report on computational results obtained on a set of benchmark problems.

Subject Areas: Heuristic, Integer Programming, Project Management, Resource-Constrained Project Scheduling.

INTRODUCTION

The research on resource-constrained project scheduling has significantly expanded over the last few decades. The vast majority of these research efforts focuses on

*This research has been supported by Project OT/03/14 of the Research Fund K.U.Leuven and Project 06163 supported by the National Bank of Belgium.

†Corresponding author.

the development of exact and heuristic procedures for the generation of a workable baseline schedule (preschedule or predictive schedule), assuming complete information and a static and deterministic environment. Such a baseline schedule is usually constructed by solving the so-called resource-constrained project scheduling problem (RCPSP). The RCPSP is also known as problem $m, 1|cpm|C_{max}$ in the notation of Herroelen, De Reyck, and Demeulemeester (2000). $m,1$ indicates that we have m renewable resource types for which the availability is specified for the unit duration period; *cpm* means that the precedence relations are of the zero-lag finish–start type, and C_{max} denotes the minimum makespan objective. This problem involves the determination of a schedule that satisfies both the zero-lag finish–start precedence constraints between the activities and the renewable resource constraints under the objective of minimizing the project duration. For reviews, we refer to Herroelen, De Reyck, and Demeulemeester (1998); Brucker, Drexl, Möhring, Neumann, and Pesch (1999); Kolisch and Hartmann (1999); Kolisch and Padman (1999); and Demeulemeester and Herroelen (2002).

A baseline schedule serves a number of important functions in assistance to project decision makers, such as facilitating resource allocation, providing a basis for planning external activities (i.e., activities to be performed by subcontractors) and visualizing future work for employees (Mehta & Uzsoy, 1998; Aytug, Lawley, McKay, Mohan, & Uzsoy, 2005). *Baseline schedules* are the starting point for communication and coordination by the project manager and team with external entities in the company's inbound and outbound supply chain; they are the basis for agreements with suppliers and subcontractors, as well as for commitments to customers.

During execution, however, a project may be subject to considerable uncertainty, which may lead to numerous schedule disruptions. Many types of disruptions have been identified in the literature (we refer to Wang (2005) and Zhu, Bard, and Yu (2005) for an overview of several schedule disruption types). Activities can take longer than expected, resource requirements or availability may vary (Lambrechts, Demeulemeester, & Herroelen, 2006a,b), due dates may change, new activities may have to be inserted due to unforeseen changes in the project scope (Artigues & Roubellat, 2000), and so forth.

When disruptions occur during schedule execution, the baseline schedule must be rescheduled. If we wish to explore the aforementioned coordination and planning purposes of a baseline schedule to the best possible extent, it is desirable that the actual start of each activity occurs as closely as possible to its baseline starting time. Also, as will be pointed out in the next section, deviations from planned activity starting times can lead to numerous types of costs, such as storage costs, organizational costs, and so forth. A baseline with express anticipation of disruptions, which is protected against certain undesirable consequences of rescheduling, is called *robust*. Van de Vonder, Demeulemeester, Herroelen, and Leus (2005) distinguish between two types of robustness: quality robustness and solution robustness. *Quality robustness* is defined as the probability that a project ends within the project deadline. *Solution robustness*, also referred to as *stability*, is defined as a quality of the scheduling environment when there is little deviation between the baseline and the executed schedule. The option that we explore in this article is to introduce stability into the baseline schedule through proper allocation of the resources (for more information on solution robust project scheduling, we refer to Herroelen and Leus (2004a,b), Leus and Herroelen (2004), and Herroelen and Leus (2005)).

We propose a reactive scheduling policy in the presence of activity duration variability that requires the resource allocation to remain constant. This means that, given a certain baseline schedule, we will determine for every resource unit the sequence of activities that will use that particular resource unit. As a consequence, precedence relations will arise between activities using the same resource unit(s). When activities take longer than expected, the schedule can then be revised by simply calculating a new early start schedule, given the precedence relations induced by resource allocation decisions. We develop three integer programming–based heuristics and one constructive procedure for making the resource allocation decisions in such a way that schedule stability is achieved.

The robust resource allocation decision problem described in this article is encountered in diverse environments, including new product development, commercial shipbuilding, construction, and software development, for example, where project plans must be developed subject to precedence and resource constraints, but then must be modified during execution due to changes in scope, underestimation in project activity durations, and other deviations from the initial baseline schedule.

While the need for planning for uncertainty in resource-constrained project schedules has resulted in new planning methodologies such as the critical chain methodology developed by Goldratt (1997), evidence is now available that the buffering techniques applied in this methodology—implemented through supporting software—may overprotect the project makespan and may lead to unnecessary high project due dates and that the procedure may generate unstable schedules caused by the fact that the feeding buffer mechanism may fail to prevent the propagation of schedule disruptions throughout the baseline schedule. Herroelen and Leus (2001); Elmaghraby, Herroelen, and Leus (2003); and Van de Vonder et al. (2005) have evaluated the critical chain methodology using a computational experiment on an extensive set of test instances, reaching the paradoxical conclusion that the critical chain scheduling procedure—being essentially a scheduling procedure that tries to protect the project makespan—is hard to defend, especially for those projects where due date performance is deemed important.

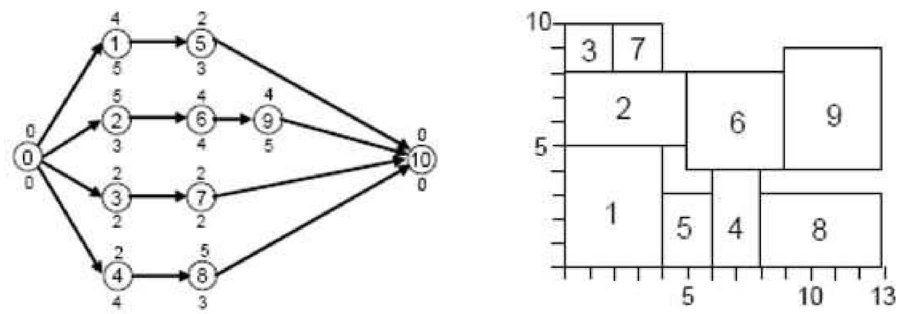
The structure of the article is as follows. The next section introduces the basic definitions and the concept of resource flow networks used to represent the resource allocation decisions. It concludes with a formal statement of the problem under investigation. Then we offer a review of the literature, followed by the resource allocation heuristics developed in this article. In a subsequent section we present lower bounds on schedule stability that can be used to validate our algorithms. Following that, we present computational results obtained on a set of benchmark problems. We compare the performance of our algorithms to some previously developed procedures, and in the last section we provide conclusions.

RESOURCE ALLOCATION AND RESOURCE FLOW NETWORKS

Basic Definitions and Notation

We assume a project network consisting of a set N of $n + 1$ activities in activity-on-the-node representation with a single zero-duration dummy–start node 0 and a

Figure 1: An example resource-constrained project scheduling problem instance.



(a) Network representation. (b) Makespan minimizing schedule.

single zero-duration dummy end node n. Project activities j (j=1, 2, ..., $n-1$) have stochastic activity durations $\boldsymbol{d}_j$, are subject to zero-lag finish–start precedence constraints, and require an integer per-period amount r_{jk} of one or more renewable resource types k ($k \in K$ with $K = \{1, \ldots, m\}$) during their execution. The renewable resource types (e.g., labor or equipment) have a constant per-period availability a_k. The dummy activities have zero duration and zero resource usage. We assume a precedence and resource feasible baseline schedule S has been generated using deterministic activity durations d_j. This schedule provides the scheduled activity start times s_j, $j = 0, \ldots, n$.

Figure 1(a) shows an example project. The number above a node denotes the corresponding deterministic activity duration, while the number below a node denotes the per-period requirement for a single renewable resource type. The resource type has a per-period availability of 10 units. Figure 1(b) shows a minimum duration baseline schedule for the project generated by the branch-and-bound procedure of Demeulemeester and Herroelen (1992, 1997). We define δ as the vector of baseline starting times. For the example network, we have δ = (0, 0, 0, 0, 6, 4, 5, 2, 8, 9, 13). This problem instance will be used as an illustrative example throughout the article.

Because of the stochastic nature of projects and project activities, disturbances may occur during a project, causing the actually realized activity start times $\boldsymbol{s}_j$ to differ from the planned activity start times s_j. The project manager should attempt to respect the baseline schedule to the best extent possible in order to avoid system nervousness and constant resource rescheduling, in other words, to maintain stability in the system and reduce disruption costs. Therefore, we opt for a so-called *railway execution mode* by never starting activities earlier than their prescheduled start time in the baseline schedule. Effectively, the baseline start times become release dates for schedule execution. This type of constraint is inherent to course scheduling, sports timetabling, and railway and airline scheduling. In a project setting, activity execution cannot start before the necessary materials have been delivered to the site (Smith-Daniels & Aquilano, 1987), and the parties responsible for these prerequisites have normally been told the baseline starting time of the initial schedule is the due date.

Following Leus (2003), Herroelen and Leus (2004a,b), and Leus and Herroelen (2004), we adopt as a measure of preschedule stability the expected weighted deviation in start times in the actual schedule from those in the baseline schedule. In other words, we aim to minimize $\sum w_j E(\mathbf{s}_j - s_j)$, where E denotes the expectation operator and $w_j \in \mathbb{N}$ denotes the weight of activity j, which is the marginal cost of starting activity j later than planned in the baseline schedule. This 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. These costs may also include disruptions to schedules of other projects in the multiproject environment (Bock & Patterson, 1990) including loss of productivity due to learning and forgetting costs brought on by disruptions (Ash & Smith-Daniels, 1999). We always set $w_0 = 0$; minimization of expected makespan is the special case where $w_j = 0$, $j \neq n$, and $w_n \neq 0$.

Resource Flow Networks

The way in which renewable resources are passed on between the various project activities in the baseline schedule can be represented by a resource flow network (Artigues, Michelon, & Reusser, 2003; Leus, 2003; Leus & Herroelen, 2004). It has the same set of nodes (N) as the original project network $G = (N, A)$ (with nodes N and arcs A), but resource arcs (A_R) are connecting two nodes i and j if there is a resource flow f_{ijk} of any resource type k from activity i (when it finishes) to activity j (when it starts). We assume that for every resource type k the sum of all flows out of the dummy start activity equals the sum of all flows into the dummy end activity, both equal to the total resource availability a_k. Formally:

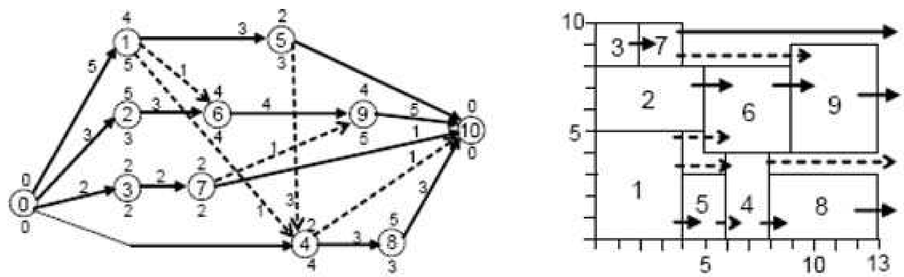
$$\sum_{j \in N} f_{0jk} = \sum_{j \in N} f_{jnk} = a_k, \quad \forall k \in K. \tag{1}$$

Moreover, a feasible resource flow network must satisfy the flow conservation constraints at the intermediate nodes. For every resource type k and for every non-dummy activity $i \neq 0, n$, the sum of flows into this activity must equal the sum of flows out of this activity, which must be equal to the resource requirement r_{ik}. This results in the following constraints:

$$\sum_{j \in N} f_{ijk} = \sum_{j \in N} f_{jik} = r_{ik}, \quad \forall i \in N \backslash \{0, n\}, \forall k \in K. \tag{2}$$

Figure 2(a) shows a possible feasible resource flow network for the example schedule in Figure 1(b). The solid arcs represent the original precedence relations, while the dashed arcs indicate extra precedence relations imposed by the resource flow network. The example project only requires the use of a single resource type, so, in order to simplify notation, we omit the index k. Positive flows f_{ij} are indicated next to each arrow, corresponding to the activity pair (i, j). The nonzero flows are: $f_{0,1} = 5; f_{0,2} = 3; f_{0,3} = 2; f_{1,4} = 1; f_{1,5} = 3; f_{1,6} = 1; f_{2,6} = 3; f_{3,7} = 2; f_{4,8} = 3;$ $f_{4,10} = 1; f_{5,4} = 3; f_{6,9} = 4; f_{7,9} = 1; f_{7,10} = 1; f_{8,10} = 3; f_{9,10} = 5$. The resource profile representation in Figure 2(b) contains the same information as the network representation in Figure 2(a). The resource profile can be seen as consisting of

Figure 2: A feasible resource flow network.



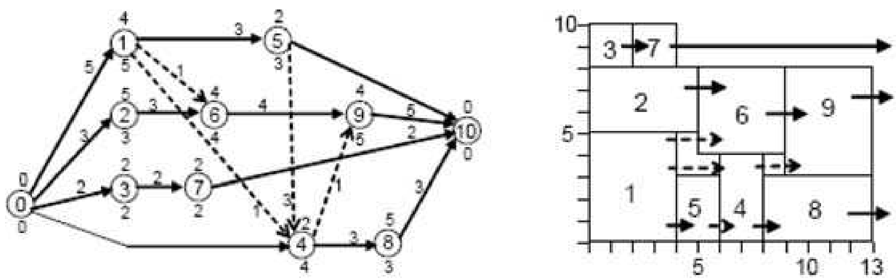
(a) Network representation. (b) Resource profile representation.

ten horizontal bands (not drawn here), one for each available resource unit. Every resource unit is transferred between the activities allocated to its corresponding band. For instance, the horizontal band corresponding to the tenth resource unit in Figure 2(b) indicates that a resource unit will be transferred from the dummy start activity to activity 3, then from activity 3 to activity 7, and finally from activity 7 to the dummy end activity. Again, full arcs represent the original precedence relations, while dashed arcs represent additional precedence constraints imposed (or added) as the result of resource constraints.

The resource flow network in Figure 2(a) and the resource profile shown in Figure 2(b) indicate that five of the available resource units are transferred from the end of the dummy start activity to the start of activity 1. Similarly, three and two units are transferred from the end of the dummy start activity to the start of activities 2 and 3, respectively. At time $t = 2$, two resource units are released by activity 3 and transferred to the start of its immediate successor, activity 7. At time $t = 4$, activity 1 releases its resources. Three resource units are transferred to the start of its successor, activity 5. Of the remaining two resource units, one unit is transferred to the start of activity 6 and another to the start of activity 4. These resource flows $f_{1,4} = 1$ and $f_{1,6} = 1$ impose two extra resource arcs indicated by the dotted arcs (1, 4) and (1, 6). These arcs induce extra zero-lag finish–start precedence constraints that were not present in the original project network. In the same way, resource flows $f_{5,4} = 3$ and $f_{7,9} = 1$ impose two extra precedence relations (5, 4) and (7, 9). Note that the resource flow $f_{4,10} = 1$ does not result in an extra precedence constraint. Indeed, activity 10 (the dummy end activity) was already a transitive successor of activity 4 in the original project network. Also, the precedence arcs (0, 4) and (5, 10) are not used to transfer any resources.

Figure 3 shows an alternative flow network and, as a result, an alternative resource allocation for the same minimal makespan schedule shown in Figure 1(b). In this flow network, the resource arc (7, 9) has disappeared and is replaced by an arc (4, 9), carrying a flow $f_{4,9} = 1$.

Figure 3: A second feasible resource flow network.



(a) Network representation. (b) Resource profile representation.

Activity Disruptions and Stability

It should be clear that it is often possible to make different resource allocation decisions for the same baseline schedule, each represented by a different resource flow network. The possibility of generating different resource flows for the same baseline schedule may have a serious impact on the robustness of the corresponding reactive scheduling procedure.

In this article, we assume that uncertainty stems from activity duration variability. When information becomes known about durations $\boldsymbol{d_j}$ that take on a realization different from d_j, the schedule might need to be revised. In this schedule revision process, we require the resource allocation to remain constant, that is, the same resource flow is maintained. Such a reactive policy is preferred when specialist resources (e.g., expert staff) cannot be transferred between activities at short notice, for instance, in a multiproject environment in which it is necessary to book key staff or scarce equipment with high learning requirements (Ash & Smith-Daniels, 1999) or setup costs (e.g., a crane) in advance to guarantee their availability, which makes last-minute changes in resource allocation unachievable (Bowers, 1995; Leus & Herroelen, 2004).

Refer again to the resource flow networks shown in Figures 2 and 3. The project manager can only obtain four of the five required resource units from activity 9's immediate predecessor, activity 6. In Figure 2, activity 9 receives its fifth resource unit from activity 7, whereas in Figure 3 it gets it from activity 4. Because activity 7 is scheduled to end at time $t = 4$, while activity 4 is scheduled to end at time $t = 8$, the resource flow network in Figure 2 is probably the better choice. Indeed, activity 7 has to undergo a delay of at least six time units (we call this the *float* between activity 7 and activity 9) before it will affect the start of activity 9, while a delay of two time units of the end of activity 4 suffices to delay the start of activity 9.

Formal Problem Statement

Given a certain baseline schedule S with activity start times $s_0, \ldots, s_n$, our objective is to generate the resource flows f_{ijk} such that the stability of the baseline schedule is maximized. Formally:

[Problem $P1$]

$$\text{minimize} \sum_{j \in N} w_j E(\boldsymbol{s_j} - s_j) \tag{3}$$

subject to

$$\sum_{j \in N} f_{0jk} = \sum_{j \in N} f_{jnk} = a_k, \forall k \in K \tag{4}$$

$$\sum_{j \in N} f_{ijk} = \sum_{j \in N} f_{jik} = r_{ik}, \forall i \in N \backslash \{0, n\}, \forall k \in K \tag{5}$$

$$\boldsymbol{s_j} = \max\left(s_j, \max_{i \in \Pr ed_j}(\boldsymbol{s_i} + \boldsymbol{d_i})\right), \forall j \in N \tag{6}$$

$$f_{ijk} \in \mathbb{N}, \forall i, j \in N; \forall k \in K. \tag{7}$$

The objective function in equation (3) is to maximize schedule stability, that is, to minimize the weighted expected deviation between planned and realized activity start times. Equations (4) and (5), shown earlier as equations (1) and (2), are the flow feasibility constraints imposed on a feasible resource flow network. Equation (6) specifies the railway scheduling reactive policy: $\boldsymbol{s_j}$, the realized start time of activity j, should be the maximum of the planned start time s_j in the baseline schedule and the maximum finish time of the predecessors $Pred_j$ of activity j in the network $G(N, A \cup A_R)$. Equation (7) imposes integrality on the flow variables.

Problem $P1$ has been shown to be ordinarily *NP*-hard by Leus (2003) for the single disruption case (for additional *NP*-hardness proofs of a number of machine scheduling problems with stability objective, we refer to Leus and Herroelen (2005)).

ALGORITHMS FOR STABLE RESOURCE ALLOCATION

Literature Overview

Generating feasible resource flows

Artigues et al. (2003) present a simple method to generate a feasible resource flow by extending a parallel schedule generation scheme to derive the flows during scheduling. The algorithm iteratively reroutes flow quantities until a feasible overall flow is obtained. The allocation routine can easily be decoupled from the schedule generation. For all resource types k, flow f_{0nk} is initialized with value a_k, while all other flows are set to 0. Recall that δ is defined as the set of time instances in the input schedule that correspond with activity start times: $t \in \delta \Leftrightarrow \exists j \in N : t = s_j$. The remaining steps of the procedure are described in Algorithm 1. This algorithm attempts to generate a feasible resource flow network without attempting to maximize schedule stability or any other measure of performance. It will be used as the worst-case benchmark in the computational experiment described later on.

Branch and bound

Leus (2003) and Leus and Herroelen (2004) propose a branch-and-bound model for resource allocation for projects with variable activity durations. The allocation

Algorithm 1: Generate a feasible flow.

```
for increasing t in δ do
    for j := 1 to (n − 1) do
        if s_j = t then
            for every resource type k do
                req_k := r_jk;
                m := 0;
                while req_q > 0 do
                    if s_m + d_m ≤ s_j then
                        q := min(req_k, flow_mnk);
                        req_k := req_k − q;
                        flow_mnk := flow_mnk − q;
                        flow_mjk := flow_mjk + q;
                        flow_jnk := flow_jnk + q;
                    m := m + 1;
```

is required to be compatible with a deterministic baseline schedule and the objective is the stability objective given by equation (3). Constraint propagation is applied during the search to accelerate the algorithm. The authors obtain computational results on a set of randomly generated networks. However, they restrict their attention to a single resource type and assume exponential activity disruption lengths. Extension 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.

Chained form partial order schedules

Policella (2005) (see also Policella, Oddi, Smith, & Cesta, 2004) proposes a procedure referred to as *chaining* for constructing a *chained Partial Order Schedule* (POS) from a given precedence and resource feasible baseline schedule. The author defines a *Partial Order Schedule* (POS) as a set of solutions for the RCPSP that can be compactly represented by a temporal graph $G(N, A \cup A_R)$, which is an extension of the precedence graph $G(N, A)$, where N denotes the set of nodes (activities) and A denotes the precedence arcs, with a set of additional arcs A_R, introduced to remove the so-called minimal forbidden sets. A *minimal forbidden set* (Igelmund & Rademacher, 1983a,b) is defined for an RCPSP instance as the minimal set of precedence-unrelated activities that cannot be scheduled together due to the resource constraints. The chained POS generated by the chaining procedure has the property that its earliest start schedule corresponds to the baseline schedule used as input.

The chaining procedure for the generation of a chained POS is presented in Algorithm 2. The first step sorts all activities in increasing order of their starting times in the baseline schedule. Then the activities are incrementally allocated to the different chains. Where an activity requires more than one unit of one or more resource types, it will be allocated to a number of chains equal to the overall number of required resource units.

>**Algorithm 2:** Generate chained partial order schedule.

Sort all activities according to their start times in the input schedule
Initialize all chains empty
for each resource type k **do**
  for each activity j **do**
    for 1 to r_{jk} **do**
      $m \leftarrow SelectChain(j, k)$;
      $last(m) \leftarrow$ last activity in chain m;
      add constraint $last(m) \prec j$;
      last activity in chain $m \leftarrow j$;
return chained POS

The function *SelectChain(j, k)* is the core of the procedure. In its basic chaining form, it chooses for each activity the first available chain of its required resource type k (given an activity j, a chain m is available if the end time of the last activity allocated on it, *last(m)*, is not greater than the start time of activity j).

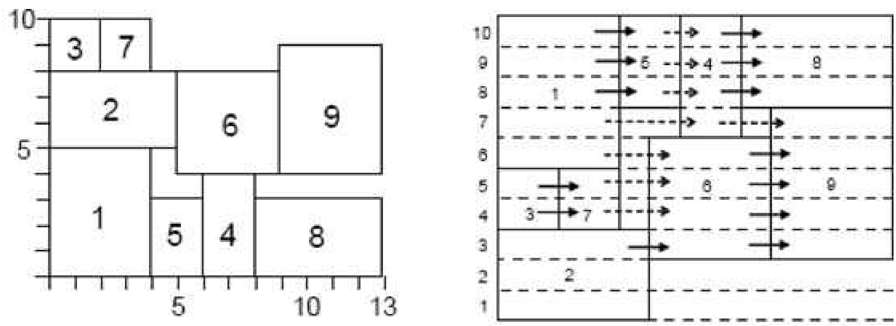
Assuming that the schedule of Figure 1(b) (repeated here as Figure 4(a)) is taken as input, the sorting step will yield the sequence of activities presented in Table 1. The procedure takes activity 1 as the first activity on the list and randomly selects five chains to fulfill its resource requirement. The only chains available are those belonging to activity 0 (dummy start), so five chains $<0, 1>$ will be created: these are chains 6 through 10 in Figure 4(b). Activity 1 is then the last activity on these chains. The next two activities in the list, activities 2 and 3, are treated in a similar way. Activity 2 is assigned to chains 1 through 3, and activity 3 is assigned to chains 4 and 5. For activity 7, the next activity in the list, only two chains are eligible: chains 4 and 5. Adding activity 7 to these chains, we get two chains $<0, 3, 7>$. The procedure continues in this way, adding activities to random eligible chains, finally yielding the chained POS shown in Figure 4(b). Note that resource flow networks and chained POSs are related concepts: a resource flow network is determined globally for all resource types k, whereas Policella's chains are computed separately for every resource type k.

Let us take a closer look at Figure 4(b). Because of the randomness in the basic chaining procedure, activity 6 is allocated to chains belonging to three different activities (activities 1, 2, and 7). This will tie together the execution of activity 1 and 6, and activity 7 and 6, two pairs of previously unrelated activities. Such interdependencies, or *synchronization points*, tend to degrade the stability of the schedule. To reduce the number of such synchronization points, Policella et al. (2004) develop two additional heuristics, *ISH* and ISH^2.

Table 1: Activities sorted according to their baseline starting time.

i	1	2	3	7	5	6	4	8	9
s_i	0	0	0	2	4	5	6	8	9

Figure 4: Chained partial order schedule (POS).



(a) Original baseline schedule. (b) Chained POS.

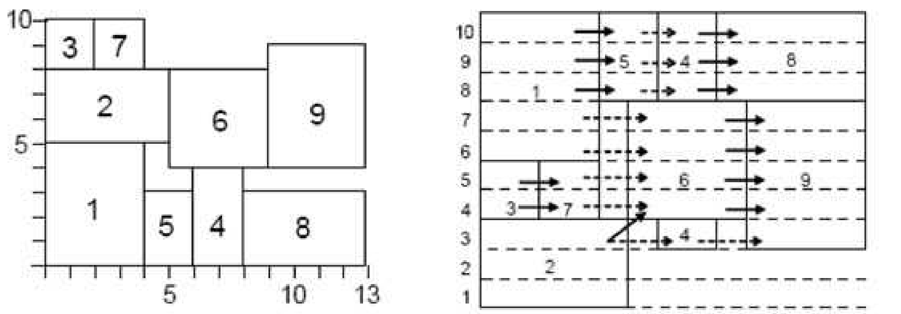
ISH tries to favor the allocation of activities to common chains by allocating an activity *j* according to the following four steps:

1. An initial chain *m* is randomly selected from among those available for activity *j* and the constraint $last(m) \prec j$ is imposed;
2. If activity *j* requires more than one resource unit, then the remaining set of available chains is split into two subsets: the set of chains that has *last(m)* as last element, $C_{last(m)}$, and the set of chains which does not, $C^{-}_{last(m)}$;
3. To satisfy all remaining resource requirements, activity *j* is allocated first to chains belonging to the first subset, $m' \in C_{last(m)}$, and,
4. In case this set is not sufficient, the remaining units of activity *j* are then randomly allocated to the first available chains, m'', of the second subset, $m'' \in C^{-}_{last(m)}$.

Assume that *ISH*, in making the allocation decision for activity 6 in the problem instance of Figure 1(b) (repeated here as Figure 5(a)) randomly selects the seventh chain $<0, 1>$ imposing the constraint $1 \prec 6$. As activity 6 requires more than one resource unit, the set of available chains is split into two subsets C_1 and C^{-}_{1}, and activity 6 is allocated to the first available chain in C_1, that is, chain 6. As this action empties the set C_1, the remaining two resource units will have to be supplied by chains belonging to C^{-}_{1}. In our example, chains 4 and 5 are selected to complete the resource allocation for activity 6. Figure 5(b) shows the complete chained POS generated by the *ISH* procedure.

While the *ISH* procedure has reduced the number of resource predecessors of activity 6 from three to two, a second type of synchronization point emerges in Figure 5(b). Activities 2 and 6 are allocated on different chains, but their precedence relation makes the execution of chain 4 dependent of the execution of chain 3. *ISH*^{2} tries to minimize this kind of interdependencies by replacing the first step of *ISH* with a more informed choice that takes into account existing ordering relations with those activities already allocated in the chaining process. More precisely, step 1 of *ISH* is replaced by the following sequence of steps:

Figure 5: Chained partial order schedule (POS) with common chains.



(a) Original baseline schedule. (b) Chained POS with common chains.

1.1. The chains m for which their last element *last(m)* is already ordered with respect to activity j are collected in the set P_j;

1.2. If $P_j \neq \emptyset$, a chain $m \in P_j$ is randomly picked; otherwise a chain m is randomly selected among the available ones;

1.3. A constraint $last(m) \prec j$ is imposed;

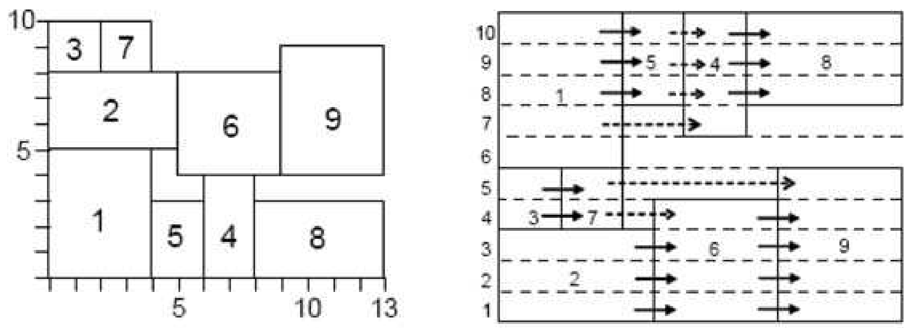
Application of ISH^2 on the problem instance of Figure 1(b) (repeated here as Figure 6(a)) may proceed as follows. First, activities 1, 2, and 3 will be allocated to the available chains. Activity 7 will be allocated to chains 4 and 5 because there is no other option. The next activity in our list is activity 5. P_5 consists of chains 6 through 10 (activity 1 being an immediate predecessor of activity 5), so a random chain will be selected from this set, and a constraint $1 \prec 5$ will be imposed. The remaining two resource units will be obtained by selecting two other chains m with $last(m) = 1$. Something similar happens to activity 6, the next activity in the list. The algorithm will first try to assign this activity to chains m with $last(m) = 2$, the immediate predecessor of activity 6. Figure 6(b) presents the complete chained POS generated by the ISH^2 procedure. The synchronization point caused by activities 2 and 6 being allocated to different chains has disappeared.

Policella et al. (2004) measure schedule robustness using two metrics: fluidity and flexibility. The *fluidity* metric is taken from Cesta, Oddi, and Smith (1998) and defined as follows:

$$fldt = 100 \times \frac{\sum_{q \neq r} Float(q, r)}{H \times n \times (n-1)}, \tag{8}$$

where H is the project horizon of the problem (i.e., the sum of the activity durations), n is the number of activities, and *Float(q,r)* is the width of the allowed distance interval between the end time of activity q and the start time of activity r. This metric characterizes the *fluidity* of a solution, that is, the ability to absorb temporal variation in the execution of activities. The hope is that the higher the value of *fldt*, the less the risk of a domino effect that affects the project completion date

Figure 6: Chained partial order schedule (POS) with removed synchronization point.



(a) Original baseline schedule.

(b) Chained POS with removed synchronization point.

and duration, and the higher the probability of localized changes. The network in Figure 5(b) has *fldt* $= 43.4$, while the network in Figure 6(b) has *fldt* $= 46.3$.

The second metric is taken from Aloulou and Portmann (2003) and is called *flexibility*, *flex*. This measure counts the number of pairs of activities in the solution that are not related by simple precedence constraints. The rationale for this measure is that when two activities are not related it is possible to move one without moving the other one. The higher the value of *flex* the lower the degree of interaction among the activities. The networks in Figures 5(b) and 6(b) both have *flex* $= 22$.

Policella et al. (2004) do not directly optimize for *fldt* and *flex*. They apply an iterative sampling search in which they execute the chaining operator described above a number of times from the same initial schedule and pick the best solution with respect to *fldt* or *flex*.

Reducing Problem Complexity

Before presenting our procedures for stable resource allocation, we will first establish a way to reduce the complexity of the problem, by identifying so-called unavoidable resource arcs. 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 strict positive amount of resource units f_{ijk} of some resource type k to be sent from activity i to activity j. Defining $A_U \subset A_R$ as the set of unavoidable resource arcs in a feasible resource flow network $G(N, A \cup A_R)$, the conditions to be satisfied by activities i and j can be formally specified as follows:

$$\forall i \in N; \forall j \in N \text{ with } s_j \geq s_i + d_i :$$

$$(i, j) \in A_U \Leftrightarrow$$

$$\exists k : a_k - \sum_{l \in P_{s_j}} r_{lk} - \max\left(0, r_{ik} - \sum_{z \in Z} r_{zk}\right) < r_{jk} \tag{9}$$

with P_{s_j} as the set of the activities that are in progress at time s_j and Z the set of activities that have a baseline starting time s_z: $s_i + d_i \leq s_z < s_j$. The left-hand side of equation (9) 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 included in the set of unavoidable resource arcs A_U.

The schedule in Figure 1(b) requires an unavoidable resource arc from activity 5 to activity 4. At time $s_4 = 6$ only activity 6 is in progress with $r_6 = 4$. Because $s_5 + d_5 = s_4$, Z is obviously void, and the left-hand side of equation (9) evaluates to $10 - 4 - 3 = 3$, which is less than $r_4 = 4$. The arc (5, 4) should thus be added to A_U. Let us investigate whether activity 6 has incoming unavoidable resource arcs. At $s_6 = 5$, only activity 5 is active, with $r_5 = 3$. The set Z of activities z with $s_1 + d_1 \leq s_z < s_6$ contains only activity 5 with $r_5 = 3$, so the left-hand side of equation (9) is equal to $10 - 3 - \max(0, (5 - 3)) = 5$, which is greater than $r_6 = 4$. This means that a feasible resource allocation for activity 6 is possible without extra unavoidable resource arcs. The complete set of unavoidable resource arcs for the schedule in Figure 1(b) is equal to $A_U = \{(0, 1), (0, 2), (0, 3), (3, 7), (1, 5), (5, 4), (4, 8), (6, 9)\}$. Of course, we are only interested in resource arcs between activities that were precedence unrelated in the original project network. Let *TA* denote the set of transitive arcs of the original project network, then the set of unavoidable resource arcs between precedence unrelated activities is equal to $A_U \backslash TA = \{(5, 4)\}$.

IP-Based Algorithms

As was mentioned previously, Problem $P1$ is an *NP*-hard problem. In this section we describe three heuristic algorithms based on alternative linear integer programming formulations that aim at avoiding the use of stochastic variables.

Minimize the number of extra arcs

It should be clear that reducing the number of extra precedence relations imposed by resource flows will lead to resource flow networks that are generally more stable. The mixed integer programming model presented in this section aims at minimizing the number of extra arcs imposed by the resource allocation decisions. We define a binary integer variable x_{ij}, taking the value 1 if there is a precedence relationship between activities i and j, 0 otherwise. Minimizing the sum of these x_{ij} variables is then equivalent to minimizing the total number of additional precedence relations. This results in problem *MinEA*:

$$\text{minimize} \sum_{i \in N} \sum_{j \in N} x_{ij} \tag{10}$$

subject to

$$\sum_{j \in N} f_{0jk} = \sum_{j \in N} f_{jnk} = a_k, \forall k \in K \tag{11}$$

$$\sum_{j \in N} f_{ijk} = \sum_{j \in N} f_{jik} = r_{ik}, \forall i \in N \backslash \{0, n\}, \forall k \in K \tag{12}$$

$$f_{ijk} \leq M x_{ij}, (i, j) \in PEA, \forall k \in K \tag{13}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \in N \tag{14}$$

$$f_{ijk} \in \mathrm{N}, \forall i, j \in N; \forall k \in K. \tag{15}$$

The objective function (10) minimizes the number of extra arcs imposed by the resource allocation decisions. Constraints (11) and (12) are again the flow feasibility constraints shown earlier as equations (1) and (2). Equation (13), with M a sufficiently large integer, imposes extra arcs linking nodes i and j when needed. As soon as, for any resource type k, a resource flow f_{ijk} takes a value strictly larger than zero, the corresponding x_{ij} variable is set equal to 1. This constraint is defined for every activity pair (i, j) in the set of *possible extra arcs* (*PEA*). This set consists of all pairs of activities (i, j), except those pairs that are already directly or indirectly precedence related in $G(N, A \cup A_U)$, or the pairs that can never be precedence related, due to their starting times in the baseline schedule. Note that the use of the set A_U of unavoidable arcs makes the set *PEA* (and the number of decision variables x_{ij}) smaller. One can verify that in our example instance of Figure 1, $PEA = \{(1, 6), (1, 9), (2, 4), (2, 8), (3, 4), (3, 5), (3, 6), (3, 8), (3, 9), (4, 9), (5, 9), (7, 4), (7, 5), (7, 6), (7, 8), (7, 9)\}$. Finally, equation (14) defines the 0-1 decision variables, while equation (15) imposes integrality conditions on the flow variables.

Maximize the sum of pairwise floats

We first introduce some notation. Given a project network $G(N, A)$, we define for all pairs of activities (i, j) with $s_i + d_i \leq s_j$, the *pairwise float* PF_{ij} as the time difference between the start of activity j and the end of activity i: $PF_{ij} = s_j - (s_i + d_i)$. We then define $MSPF_{ij}$ as the minimal sum of pairwise floats on all paths from activity i to activity j. This gives us the maximum amount of time (possibly zero) by which the end of activity i may be delayed without delaying the start of activity j (provided no other disruptions occur). For instance, in the resource flow network presented in Figure 3, there are three paths from activity 1 to activity 9. There is the path 1-6-9 with the sum of pairwise floats equal to $PF_{16} + PF_{69} = 1 + 0 = 1$, the path 1-4-9 with the sum of pairwise floats equal to $PF_{14} + PF_{49} = 2 + 1 = 3$, and the path 1-5-4-9 with sum of pairwise floats equal to 1. Hence, $MSPF_{19} = \min(1, 3, 1) = 1$. This means that the end of activity 1 can be delayed for one time unit without affecting the start of activity 9. Clearly, high $MSPF_{ij}$ values will result in a more stable resource flow network.

We define the set Q as the set of all (ordered) activity pairs (i, j) for which a positive resource flow from activity i to activity j is possible: $Q = \{(i, j) \mid s_i + d_i \leq s_j\}$. We then formulate problem *MaxPF* as follows:

$$\text{maximize} \sum_{(i,j) \in Q} MSPF_{ij} \tag{16}$$

subject to

$$\sum_{j \in N} f_{0jk} = \sum_{j \in N} f_{jnk} = a_k, \quad \forall k \in K \tag{17}$$

$$\sum_{j \in N} f_{ijk} = \sum_{j \in N} f_{jik} = r_{ik}, \quad \forall i \in N \backslash \{0, n\}, \forall k \in K \tag{18}$$

$$f_{ijk} \leq Mx_{ij}, \ (i, j) \in PEA, \forall k \in K \tag{19}$$

$$MSPF_{ij} \leq PF_{ik} + MSPF_{kj}, (i, j) \in Q, \forall (i, k) \in A \cup A_U, (k, j) \in Q \tag{20}$$

$$MSPF_{ij} \leq PF_{ik} + MSPF_{kj} + M(1 - x_{ik}), (i, j) \in Q, \forall (i, k) \in PEA, \\ (k, j) \in Q \tag{21}$$

$$MSPF_{ii} = 0, \forall i \in N \tag{22}$$

$$MSPF_{ij} \leq C, (i, j) \in Q \tag{23}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \in N \tag{24}$$

$$f_{ijk} \in \mathbb{N}, \forall i, j \in N; \forall k \in K \tag{25}$$

$$MSPF_{ij} \in \mathbb{N}, (i, j) \in Q. \tag{26}$$

The objective function (16) maximizes the minimal sum of pairwise floats over all pairs of activities (i, j) satisfying the inequality $s_i + d_i \leq s_j$. Equations (17)–(19) are the flow and extra arc constraints, which we already explained in the previous section. In equations (20)–(23) we calculate the minimal sum of pairwise floats in a recursive way. Equation (20) splits off one arc (i, k), where activity k is either a direct successor of activity i, or an unavoidable resource successor of activity i. The remaining pairwise float $MSPF_{kj}$ is calculated recursively. Equation (21) does the same, but now activity k is a possible extra successor of activity i (i.e., $(i, k) \in PEA$). If the possible extra arc (i, k) is not present in the current solution, the variable x_{ik} will be equal to zero and the corresponding equation (21) will not be binding (M again being a large integer). If for a certain pair of activities i and j all $MSPF_{ij}$ constraints are not binding, then these activities are both precedence and resource independent in the current resource flow network, and the $MSPF_{ij}$ variable will obtain its maximum value C, as enforced by equation (23), C being a positive constant. High values of C will result in an approach in which we try to maximize the number of resource and precedence independent activities. Low values of C will result in an approach in which we are willing to sacrifice the independency of a pair of activities if this results in a total increase of other $MSPF_{ij}$ values. This increase should then at least be equal to C. In our experiments, we set $C = 10$. Finally, equation (22) makes certain the recursion ends, and equations (24)–(26) are the binary and integrality constraints.

To see how the objective function (16) evaluates different resource flow networks, let us take another look at Figures 2 and 3. Note that $MSPF_{39}$, $MSPF_{79}$, $MSPF_{59}$, and $MSPF_{49}$ will have a different value in both resource flow networks, while the value of all other $MSPF_{ij}$ variables will be the same. In Figure 2, $MSPF_{39} = MSPF_{79} = 5$, while activity 9 is resource (and precedence) independent of activities 5 and 4, yielding $MSPF_{59} = MSPF_{49} = C$. In Figure 3, activity 9 is resource and precedence independent of activities 3 and 7. Thus, $MSPF_{39} = MSPF_{79} = C$. However, activity 9 is now resource dependent of activities 5 and 4, and only one time unit separates the end of activity 4 from the start of activity 9, so we get $MSPF_{59} = MSPF_{49} = 1$. Because $5 + 5 + 2C > 1 + 1 + 2C$, the resource flow network in Figure 2 will be preferred over the resource flow network in Figure 3. This is a logical choice, Because a float of five time units will be sufficient to absorb most (if not all) disturbances coming from activities 3 and 7, while the single time unit of float will not always suffice to absorb disturbances coming from activities 5 and 4. This is already an improvement over our previous model, which was unable to distinguish between the networks in Figures 2 and 3 (both networks having an equal number of extra arcs).

Minimize the estimated disruption

The previous heuristic aimed at generating robust resource allocations by maximizing the sum of the pairwise floats over all pairs of activities (i,j) satisfying $s_i + d_i \leq s_j$. The heuristic described in this section attempts to minimize the propagation impact of the estimated disruptions by being more selective in the selection of pairwise floats. Activities for which the baseline starting time is close to the project makespan will have a higher probability of being disrupted than activities that are scheduled close to the start of the project, because disruptions will be propagated and accumulated throughout the network, resulting in a kind of snowball effect. Therefore, we should give higher value to float occurring at the end of the schedule, compared to an equal amount of float occurring early in the schedule, because the former float is more likely to be used by a delayed activity.

To obtain this more informed selection of pairwise floats, we simplify our original problem $P1$ and solve this simplified problem to optimality. The simplified version of $P1$ makes use of the following two assumptions:

Assumption 1: Only one activity duration disruption $d_i + \delta_i$ will occur during the execution of the project, with δ_i known and equal to d_i, the deterministic duration of activity i.

Assumption 2: Each activity has an equal probability of being subjected to this disruption.

The formulation for solving this problem introduces new variables es_{jl}, which are the realized start times of activities j when scheduled according to railway execution mode in a certain disturbance scenario l. This railway execution mode implies that activities will not start earlier than their planned start time in the baseline schedule. The formulation for problem *MinED* looks as follows:

$$\text{minimize} \sum_{l \in N \setminus \{0,\text{n}\}} \sum_{j \in N} w_j \times es_{jl} \tag{27}$$

subject to

$$\sum_{j \in N} f_{0jk} = \sum_{j \in N} f_{jnk} = a_k, \forall k \in K \tag{28}$$

$$\sum_{j \in N} f_{ijk} = \sum_{j \in N} f_{jik} = r_{ik}, \forall i \in N \setminus \{0, n\}, \forall k \in K \tag{29}$$

$$f_{ijk} \leq Mx_{ij}, (i, j) \in PEA, \forall k \in K \tag{30}$$

$$es_{0l} = 0, \forall l \in N \setminus \{0, n\} \tag{31}$$

$$es_{jl} \geq es_{ll} + 2 \times d_l, (l, j) \in A \cup A_U, \forall l \in N \setminus \{0, n\} \tag{32}$$

$$es_{jl} \geq es_{il} + d_i, (i, j) \in A \cup A_U, \forall l \in N \setminus \{0, i, n\} \tag{33}$$

$$M(1 - x_{lj}) + es_{jl} \geq es_{ll} + 2 \times d_l, (l, j) \in PEA, \forall l \in N \setminus \{0, n\} \tag{34}$$

$$M(1 - x_{ij}) + es_{jl} \geq es_{il} + d_i, (i, j) \in PEA, \forall l \in N \setminus \{0, i, n\} \tag{35}$$

$$es_{jl} \in \mathbb{N}, \forall j \in N, \forall l \in N \setminus \{0, n\} \tag{36}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \in N \tag{37}$$

$$.f_{ijk} \in \mathbb{N}, \forall i, j \in N; \forall k \in K. \tag{38}$$

The objective function (27) sums over n-1 different scenarios, where in each scenario an activity l suffers from an activity duration disruption δ_l equal to d_l. For each such scenario l, we sum the weighted realized start times $w_j \times es_{jl}$ of activities j, applying the railway execution policy in the current resource flow network. Equations (28)–(30) are again the flow conservation and extra arc constraints, which we described earlier. Equations (31)–(35) then calculate the realized activity start times according to the railway execution mode. Equations (32) and (34) ensure that activity l suffers from an activity duration disruption with a magnitude equal to its deterministic duration. Equations (34) and (35) are only binding in the presence of the associated possible extra arc, imposing an additional precedence constraint (M is again a large integer). Finally, equations (36) and (38) are the integrality constraints associated with the realized activity start times and the resource flows, while equation (37) indicates the binary constraints associated with the x_{ij} variables.

Because activities with baseline starting times close to the project makespan will be disrupted in more scenarios than activities starting in the first time periods of the baseline schedule, the formulation will automatically emphasize the preservation of extra float between activities starting at later time periods in the baseline schedule. Also, while our previous heuristics optimized a certain characteristic of robust resource flow networks, the objective function (27) resembles the original stability objective (3) more closely. Moreover, the objective function (27) allows for a very natural integration of activity weights. We hope that all this will result in a better approximation of the stability objective.

A Constructive Resource Allocation Procedure

In this section, we present a constructive resource allocation procedure, which we call MABO (*myopic activity-based optimization*). The procedure is myopic

Algorithm 3: Myopic activity-based optimization.

Initialize: $A_R = A_U$ and $\forall k \in K : alloc_{ok} = a_k$
For each activity $i \in N \setminus \{0, n\}$, calculate the estimated stability cost contribution c_i
Sort the project activities by increasing s_j (tie-break: decreasing c_j)
For every activity j in the sorted list
1. Calculate $Avail_{jk}(A \cup A_R) = \sum\limits_{\forall i:(i,j)\in A\cup A_R} alloc_{ik}$ 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 $(h, j) \in H_j \Leftrightarrow$
        $(h, j) \notin A \cup A_R$
        $s_h + d_{\mathrm{h}} \leq S_j$
        $\exists k : alloc_{hk} > 0$ and $Avail_{jk}(A \cup A_R) < r_{jk}$
    2.2 Determine all minimal subsets $H_j^1, H_j^2, \ldots, H_j^q \subseteq H_j$ such that
        $\forall k \in K : Avail_{jk}(A \cup A_R \cup H_j^i) \geq r_{jk}, \quad i = 1, \ldots, q$
    2.3 Identify the subset $H_j^* \in \{H_j^1, H_j^2, \ldots, H_j^q\}$ such that
        $Stability\_cost(A \cup A_R \cup H_j^*)$ is minimized
    2.4 Add H_j^* to A_R
3. Allocate resource flows f_{ijk} to the arcs $(i, j) \in A \cup A_R$, as follows:
    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 $s_i + d_i$
            Tie-break 2: Decreasing variance σ_i^2 of d_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
                $f_{ijk} = \min\,(alloc_{ik}, r_{jk} - alloc_{jk})$
                Add f_{ijk} to $alloc_{jk}$
                Substract f_{ijk} from $alloc_{ik}$

because we do not look at other activities while deciding upon the best possible resource allocation for an activity. Unlike most existing resource allocation procedures, MABO is activity based rather than resource based. MABO consists of three steps that must 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 a next step with a minimal impact on stability. Step 3 then defines resource flows f_{ijk} from predecessor activities i to activity j. The detailed steps of the procedure are presented in Algorithm 3.

In the initialization step, the set of resource arcs A_R is initialized to the set of unavoidable arcs A_U. 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 in a list in increasing order of their planned starting times using decreasing estimated stability cost contribution c_i as tie-break rule. These values c_i are calculated as follows. For each activity i, we calculate the average delay δ_{s_i} in its start time due to activity duration disruptions of its predecessors in the network $G(N, A \cup A_U)$ by means of simulation. Then, we apply the railway execution policy to all transitive successors of activity i in the network $G(N, A \cup A_U)$, when activity i has a realized start time $\boldsymbol{s_i} = s_i + \delta_{s_i}$ and a realized activity duration $\boldsymbol{d_i} = 1.25 \times d_i$. Given these realized start times, we set the value of c_i to the sum of all weighted start time deviations of the transitive successors of activity i. This value of c_i gives us a measure of the contribution of an activity to the total stability cost.

In Step 1 of MABO, we calculate 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, we need to find additional sources for that particular resource type, resulting in new precedence constraints that have to be added to A_R. This is what we do in Step 2. We define the set H_j of all possible arcs between a possible resource source h of the current activity j and j itself. By solving a small recursion problem we can find the subset H_j^* of H_j 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 $\sum_{j \in N} w_j E(\boldsymbol{s_j} - s_j)$, computed through simulation of 100 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 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 myopically.

In Step 3, we allocate the actual resource flows f_{ijk} to the predecessors of j in $(A \cup A_R)$ and we update $alloc_{ik}$, the number of resource items allocated to each activity. If $Avail_{jk}(A \cup A_R) > r_{jk}$ for resource type k, we have to decide which predecessors account for the resource flows. We try to do this in an intelligent way, because a greedy algorithm would even reinforce the myopic character of MABO imposed in Step 2. 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 of 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 we restart the three-step procedure for the next activity in the list until we have obtained a complete feasible resource allocation 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 the MABO procedure on the minimal makespan schedule of Figure 1. This project has only one resource type, so we will omit the index k. We start by sorting the nondummy activities according to increasing start time, yielding the list (1, 2, 3, 7, 5, 6, 4, 8, 9). All available resource units are allocated to the dummy start activity ($alloc_0 = 10$).

Activity 1 comes first in the list. This activity has only one predecessor, such that $Avail_1 = alloc_0 = 10$, which is sufficient to fulfill the resource requirement $r_1 = 5$. We set $f_{0,1} = \min(alloc_0, r_1 - alloc_1) = 5$, $alloc_0 = 5$, and $alloc_1 = 5$.

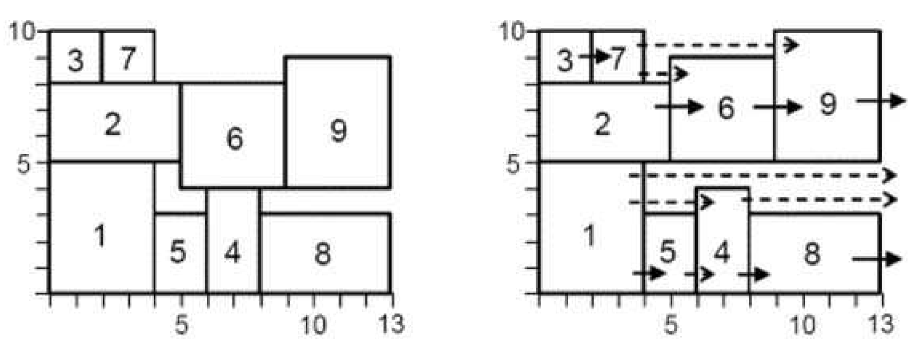
Activities 2 and 3 are treated in the same way. Both take their resources from their only predecessor, activity 0. This depletes the allocation for activity 0: $alloc_0 = 0$.

The next activity in the list is activity 7. $Avail_7 = alloc_3 = 2$, which suffices to fulfill the requirement $r_7 = 2$. We set $f_{3,7} = \min(alloc_3,\ r_7 - alloc_7) = 2$, $alloc_3 = 0$, and $alloc_7 = 2$.

Activity 5 poses no problem either. We calculate $Avail_5 = alloc_1 = 5 > r_5$. We set $f_{1,5} = 3$, $alloc_1 = 5 - 3 = 2$, and $alloc_5 = 3$.

The resource allocation for activity 6, the next activity in the list, is more complicated. We have $Avail_6 = alloc_2 = 3$, which is smaller than the resource requirement of activity 6, $r_6 = 4$. We must obtain one more resource unit from one of the activities that have already finished. The eligible activities are activities 7 and 1, so we set $H_6 = \{(7,6), (1,6)\}$. Two subsets must be evaluated, namely $\{(7,6)\}$ and $\{(1,6)\}$. Simulation shows that both subsets have an instability cost of 0.11 (which means that, on average, the start of activity 6 will be delayed for 0.11 time units as a result of the extra precedence relation), so we arbitrarily select the first subset: $H_6^* = \{(7,6)\}$ and add this arc to A_R. We arrive at step 3 of the MABO procedure, with the set of predecessors to be sorted being equal to $\{2,7\}$. Activity 2 has no successors starting later than $s_6 = 5$, and the only successor of activity 7 is the dummy end, so we have to resort to our first tiebreak. Activity 2 ends later than activity 7 in the baseline schedule, so activity 2 will appear first in the list, and we arrive at step (3.2) with the sorted list of predecessors equal to $\{2,7\}$. A first pass through the while loop results in $f_{2,6} = 3$, $alloc_2 = 0$, and $alloc_6 = 3$. We go through the while loop a second time, and set $f_{7,6} = \min(2, 4 - 3) = 1$, $alloc_7 = 2 - 1 = 1$, and $alloc_6 = 3 + 1 = 4$, completing the resource allocation for activity 6. The procedure then continues until a complete feasible resource allocation is found. The schedule representation of the complete resource flow network generated by the MABO procedure is shown in Figure 7.

Figure 7: Resource flow network obtained with the myopic activity-based optimization procedure.



(a) Original baseline schedule. (b) Resource flow network.

LOWER BOUNDS ON SCHEDULE STABILITY COST

In this section, we derive a lower bound on the schedule stability cost for a given schedule. Deriving a tight lower bound is important because it allows us to evaluate the performance of our algorithms with regard to our performance measure. The lower bound calculations identify the stability cost contributions that are indispensable. These include stability cost contributions due to the original precedence relations A and the unavoidable resource arcs A_U. $Stability\_cost(A \cup A_U)$ is thus a lower bound on the schedule stability cost that is independent of the resource allocation decisions. We will refer to this weak lower bound as LB_0.

Algorithm 4 presents a tighter lower bound, which can be found by focusing on the resource allocation decisions that are not resolved by taking into account the unavoidable arcs $A \cup A_U$. We calculate for each activity j the best case scenario to solve the myopic resource allocation problem. We begin by calculating the minimal number of resource items $alloc_{ik}$ allocated at time s_j to each activity i with $s_i < s_j$ as $\max(0, r_{ik} - \sum_{z \in Z_i} r_{zk})$. As in the previous section, Z_i denotes the set of activities that have a baseline starting time s_z: $s_i + d_i \leq s_z < s_j$. Summing up $\sum_{i \in N} alloc_{ik}$might result in a total number of allocated resource units that is smaller than a_k. The difference $alloc_{xk} = a_k - \sum_{i \in N} alloc_{ik}$ represents the number of resource units of type k from unknown origin at time s_j.

As in step 1 of the MABO procedure, we need to know the number of resource units that are allocated to predecessors of activity j in $A \cup A_U$. We do not know whether the unknown origins of the $alloc_{xk}$ units are predecessors of activity j or not. In any case, there are no more than $Avail_{jk} = alloc_{xk} + \sum_{\forall i:(i,j) \in A \cup A_U} alloc_{ik}$ resource units of type k allocated to predecessors of j at time s_j. If there exists a k

Algorithm 4: LB_1.

$LB_1 \leftarrow Stability\_cost(A \cup A_U)$ (initialize LB_1 with the value of LB_0)
for each activity j with $s_j > 0$ **do**
  for each resource type k **do**
    for each activity i with $s_j < s_j$ **do**

$$alloc_{ik} = \max\left(0, r_{ik} - \sum_{Z \in Z_i} r_{zk}\right)$$

$$alloc_{xk} = a_k - \sum_{i \in N, s_{i<s_j}} alloc_{ik}$$

$$Avail_{jk}(A \cup A_U)^{\max} = alloc_{xk} + \sum_{\forall i:(i,j) \in A \cup A_U} alloc_{ik}$$

  if $\exists k : Avail_{jk}(A \cup A_U)^{\max} < r_{jk}$ **then**
    Define the set of arcs H_j with $(h, j) \in H_j \Leftrightarrow$
      $(h, j) \notin A \cup A_U$
      $s_h + d_h \leq s_j$
      $\exists k : alloc_{hk} > 0$ and $Avail_{jk}(A \cup A_U)^{\max} < \boldsymbol{r}_{jk}$
    Determine all minimal subsets $H_j^1, H_j^2, \ldots, H_j^q \subseteq H_j$ such that
      $\forall k \in K : Avail_{jk}(A \cup A_U \cup H_j^i)^{\max} \geq r_{jk}, \quad i = 1, \ldots, q$
    Identify the subset $H_j^* \in \{H_j^1, \ldots, H_j^q\}$ such that
      $Stability\_cost(A \cup A_U \cup H_j^*)$ is minimized
    $LB_1 \leftarrow \max(Stability\_cost(A \cup A_U \cup H_j^*), LB_1)$

---

for which $Avail_{jk} < r_{jk}$, activity j must have nonpredecessors as resource suppliers. Steps 2.1, 2.2, and 2.3, of MABO decide upon the best non-predecessors to supply extra resource units and calculate $Stability\_cost(H_j^* \cup A \cup A_U)$. The current minimal allocations $alloc_{ik}$ are used as input.

After doing this for every activity, we identify the activity j^* that is the most costly to resolve and calculate $Stability\_cost(H_{j^*}^* \cup A \cup A_U)$. The resultant stability cost is a tighter lower bound on the schedule stability cost. We will refer to this improved lower bound as LB_1.

Let us illustrate the computation of this updated lower bound on activity 9 of our example schedule of Figure 1. We start by calculating the $alloc_i$'s at time $s_9 = 9$. Obviously, $alloc_6 = 4$ and $alloc_8 = 3$ because no activities have started since their ending times, that is, $Z_6 = Z_8 = \{\}$. For activity 4, we have $alloc_4 = \max(0, r_4 - r_8) = 1$. For all other activities i, it can be verified that $alloc_i = 0$. This results in $alloc_x = 10 - (4 + 3 + 1) = 2$ resource units of unknown origin. All of this means we are certain that at least one resource unit is still allocated to activity 4. Similarly, at least 4 and 3 resource units are allocated to activities 6 and 8, respectively. This leaves us with two resource units for which we do not know to what activities they are allocated. In our lower bound, we make the assumption that these resource units are allocated to predecessors of activity 9, so we get $Avail_9 = alloc_6 + alloc_x = 6$, which is greater than r_9, so no extra stability cost is incurred to solve the resource allocation problem for activity 9.

During the calculation of the lower bound LB_1, we might encounter some activities for which the resource allocation problem cannot be solved without extra stability cost, that is, $Stability\_cost(H_j^* \cup A \cup A_U) > LB_0$ for certain activities j. If this number of stability cost increasing activities is at least two, we can tighten the lower bound LB_1 even further, by looking at the combined effect of solving the resource allocation problem for each of these activities. The detailed steps of this tightened lower bound LB_2 are presented in Algorithm 5. The procedure consists of two steps. The first step is very similar to the calculation of LB_1: we identify all stability cost increasing activities, add them to a set I, and store the subsets of arcs able to solve their resource allocation problem. In a second step, we calculate the stability cost for all possible combinations of these subsets. As we are certain that one of these combinations of subsets will appear in an optimal resource flow network (w.r.t. schedule stability), the combination of subsets with minimal stability cost gives us a tightened lower bound.

COMPUTATIONAL RESULTS

We conducted a computational experiment in which we evaluated the performance of our resource allocation procedures against the performance of the procedures already described in the literature. The performance measure used in our computational experiment is the schedule stability cost $\sum_{j \in N} w_j \, E(\boldsymbol{s_j} - s_j)$. The procedures have been tested on the J30, J60, and J120 instance sets of PSPLIB (Kolisch & Sprecher, 1997). Some elementary properties of these instance sets are summarized in Table 2. The baseline schedules for the problems of the J30 instance set are generated by the makespan minimizing branch-and-bound algorithm of Demeulemeester

Algorithm 5: LB_2.

$I \leftarrow \emptyset$
$LB_0 \leftarrow Stability\_cost(A \cup A_U)$
Step 1:
for each activity j with $s_j > 0$ **do**
  for each resource type k **do**
    for each activity i with $s_i < s_j$ **do**
      $alloc_{ik} = \max\left(0, r_{ik} - \sum_{z \in Z_i} r_{zk}\right)$
    $alloc_{xk} = a_k - \sum_{i \in N, s_i < s_j} alloc_{ik}$
    $avail_{jk}(A \cup A_U)^{\max} = alloc_{xk} + \sum_{\forall i:(i,j) \in A \cup A_U} alloc_{ik}$
  if $\exists k : Avail_{jk}(A \cup A_U)^{\max} < r_{jk}$ **then**
    Define the set of arcs H_j with $(h, j) \in H_j \Leftrightarrow$
      $(h, j) \notin A \cup A_U$
      $s_h + d_h \leq s_j$
      $\exists k : alloc_{hk} > 0$ and $Avail_{jk}(A \cup A_U)^{\max} < r_{jk}$
    Determine all minimal subsets $H_j^1, H_j^2, \ldots, H_j^q \subseteq H_j$ such that
      $\forall k \in K : Avail_{jk}(A \cup A_U \cup H_j^i)^{\max} \geq r_{jk}, \quad i = 1, \ldots, q$
    Identify the subset $H_j^* \in \{H_j^1, \ldots, H_j^q\}$ such that
      $Stability\_cost(A \cup A_U \cup H_j^*)$ is minimized
    if $Stability\_cost(A \cup A_U \cup H_j^*) > LB_0$ **then**
      $I \leftarrow I \cup \{j\}$
      store $L_j = \{H_j^1, \ldots, H_j^q\}$
Step 2:
Given $I = \{j_1, j_2, \ldots, j_p\}$ with $p \geq 2$, identify the combination of subsets
$H_{j_1}^* \in L_{j_2}^*, H_{j_2}^* \in L_{j_2}, \ldots, H_{j_p}^* \in L_{j_p}$ such that
  $Stability\_cost(A \cup A_U \cup H_{j_1}^* \cup \ldots \cup H_{j_p}^*)$ is minimized.
$LB_2 \leftarrow Stability\_cost(A \cup A_U \cup H_{j_1}^* \cup \ldots \cup H_{j_p}^*)$

Table 2: Properties of the PSPLIB instance sets.

	No. of Activities	No. of Resource Types	No. of Instances
J30	30	4	480
J60	60	4	480
J120	120	4	600

and Herroelen (1992, 1997). Heuristic baseline schedules for the J60 and J120 instance sets have been obtained by the combined crossover algorithm by Debels and Vanhoucke (2006).

All computational results have been obtained on a personal computer equipped with an Intel® Pentium® IV 2.4 GHZ processor. The algorithm by Artigues et al. (2003), the three algorithms developed by Policella et al. (2004), that is, basic chaining, *ISH* and *ISH*^{2}, the MABO procedure, and the lower bounds have been coded in C++. The iterative sampling procedures *ISH* and *ISH*^{2} optimize for the flexibility metric. For each instance, 100 resource flow networks are generated by the heuristic chaining operators and the one with the highest flexibility is withheld.

Problems *MinEA*, *MaxPF,* and *MinED* are solved using the callable libraries of ILOG's CPLEX 8.0. For every problem instance, the MIP solver was given a maximum of 60 seconds of computation time per problem. If necessary, we aborted after 60 seconds with the current best solution (w.r.t. the objective function). The weights w_j for each nondummy activity $j \in \{1, 2, \ldots, n-1\}$ are drawn from a discrete triangular distribution with $P(w_j = q) = (21 - 2q)\%$ for $q \in \{1, 2, \ldots, 10\}$. This distribution results in a higher occurrence probability for low weights and in an average weight $w_{avg} = 3.85$. The weight w_n of the dummy end activity denotes the marginal cost of violating the project due date and is fixed at $\lfloor 10 \times w_{avg} \rfloor = 38$. For an extensive evaluation of the impact of the activity weights, we refer to Van de Vonder et al. (2005) and Van de Vonder, Demeulemeester, Herroelen, and Leus (2006).

For each activity the actual simulated realized activity duration is drawn from a right-skewed beta distribution with parameters 2 and 5 and an expected value equal to the deterministic activity duration. The minimum and maximum values of this distribution equal 0.5 times and 2.25 times the expected activity duration, respectively. For each instance and procedure (both exact and heuristic), 100 simulation runs have been made for the evaluation of the stability objective.

The results obtained on the J30 instance set are presented in Table 3. The second column with the header "Stability" lists the average stability cost $\sum_{j \in N} w_j$ $E(\boldsymbol{s_j} - s_j)$ obtained for each heuristic resource allocation procedure over the 100 simulation runs for each of the 480 J30-problem instances of the PSPLIB. Because neither Artigues et al. (2003) nor Policella et al. (2004) take into account the activity weights w_j in making the resource allocation decisions, we also show in the third column the average stability cost results obtained with all activity weights w_j set to 1; that is, $\sum_{j \in N} E\ (\boldsymbol{s_j} - s_j)$. The fifth column shows the number of instances that could be solved to proven optimality by the MIP solver within the given time limit. Obviously, this column is only relevant for the integer programming heuristics.

The basic chaining procedure shows the worst performance for both stability measures. This is according to expectations, because the procedure allocates resources to activities in a completely random fashion. One thing that draws our attention is the fact that the procedure by Artigues et al. (2003), which only aims

Table 3: Results on the J30 instance set.

	Stability	Stability ($w_j = 1, \forall j \in N$)	CPU Time (s)	# Optimal
Artigues	397.88	67.78	0.646×10^{-3}	n/a
Basic Chaining	446.13	75.92	0.693×10^{-2}	n/a
ISH_{flex}	405.94	69.04	0.784	n/a
ISH^2_{flex}	393.96	66.85	0.815	n/a
MinEA	360.32	61.04	1.12	478
MaxPF	351.89	59.46	0.730	479
MinED	347.07	58.66	1.41	475
MABO	350.32	59.36	0.291×10^{-1}	n/a
LB_2	277.82	56.50	0.145×10^{-1}	n/a

at producing a feasible resource flow network without any stability objective, outperforms the basic chaining procedure as well as the ISH_{flex} procedure. The reason for this lies in the fact that the procedure by Artigues will always consider activities that supply resources to a given activity in the same order (i.e., increasing start times). Hence, the resource suppliers for a certain activity are more likely to be similar for different resource types. By contrast, the basic chaining and ISH_{flex} procedures select the first resource supplier for a given activity and resource type in a random fashion, which in general will lead—when multiple resource types are considered—to more resource dependencies between activities. The ISH^2_{flex} procedure is the only procedure developed by Policella that outperforms the procedure by Artigues for exactly that reason: the randomness is reduced by applying a policy where predecessors are preferred as resource suppliers for a given activity. Also, predecessors in the original project network incur no extra stability cost, yielding an additional positive effect on the stability objective.

Of the heuristics developed in this article, *MinED* performs the best on this instance set, followed by MABO and *MaxPF*, Note that the results of *MinED* for the unweighted stability objective are pretty close to the lower bound LB_2, indicating that the resulting resource flow network is a very good solution with respect to the unweighted stability objective Finally, the *MinEA* heuristic does not perform very well when compared to *MinED*, *MaxPF*, and MABO, but it still yields better results than the procedures developed by Artigues and Policella. The reason for this moderate performance of the *MinEA* heuristic can be found in the coarse approximation of the stability objective, and in the fact that the heuristic is unable to make an informed choice between two resource flow networks with an equal number of extra arcs.

As for computation times, we can see that the IP heuristics have an average computation time of (more or less) one second. Also, almost all problems could be solved to optimality within the given time limit. Finally we note that the MABO procedure obtains results on the stability objective that are slightly better than the *MaxPF* heuristic, while its computation time is on average 25 times shorter.

To determine whether these conclusions also hold for larger problem instances, let us consider Table 4, presenting the results on the 480 problems of the J60 instance set of PSPLIB. First note that the lower bound calculated here is not the lower bound LB_2 presented in the previous section, but a weaker version of it. Because the number of combinations of subsets L_{j_k} can grow very large, we limit the number of stability cost increasing activities in such manner that no more than 10,000 subset combinations have to be evaluated. Of course, this seriously reduces the tightness of the lower bound.

We notice that none of Policella's heuristics outperform the procedure by Artigues on this instance set. Again, this can be attributed to the fact that Policella's algorithms sometimes make very different resource allocation decisions between the different resource types. When looking at the computation times, we can see that the IP heuristics need much more time than on the J30 instance set. The average computation time for these heuristics is now about 40 seconds per problem and the number of problems we were able to solve to optimality drops to less than half of the instances. As a consequence, the performance of the IP heuristics degrades, and our MABO procedure now obtains results which are very comparable to those of our

Table 4: Results on the J60 instance set.

	Stability	Stability ($w_j = 1, \forall j \in N$)	CPU Time (s)	# Optimal
Artigues	960.35	194.75	0.208×10^{-2}	n/a
Basic chaining	1182.68	239.30	0.160×10^{-1}	n/a
ISH_{flex}	1039.67	209.51	2.02	n/a
ISH^2_{flex}	968.23	197.54	2.20	n/a
MinEA	796.94	161.32	39.8	183
MaxPF	764.89	154.49	37.5	222
MinED	737.72	149.17	39.0	189
MABO	739.97	149.50	0.340	n/a
Lower bound	565.85	113.97	0.996	n/a

best IP heuristic, *MinED*. However, if we provide the *MinED* model with the output of the MABO procedure as a starting solution, the results reported for the weighted stability can be improved from 737.72 to 729.77, while the unweighted stability can be reduced from 149.17 to 147.25. Furthermore, the average computation time drops from 39 to 37 seconds. In this manner, we can apply the *MinED* model as a kind of improvement procedure on top of the MABO procedure.

The results obtained on the 600 problems of the J120 instance set are presented in Table 5. The *MinED* heuristic found no integer solution within the given time limit, so no results are presented here. Also, the *MinED* model could not be used as an improvement procedure on top of the MABO procedure, as no improvements were found within the time limit. The *MaxPF* heuristic did find integer solutions, but for a small set of problems this took longer than 60 seconds. In that case, the MIP solver was allowed to exceed the given time limit, aborting with the first integer solution found. This is reflected by the average computation time of 155 seconds. The results on the stability objective show that MABO is the clear winner here. Also, it requires only approximately half a second per problem, on average. In comparison to the IP heuristics, the running time of the MABO procedure does not explode when the number of activities increases.

Table 5: Results on the J120 instance set.

	Stability	Stability ($w_j = 1, \forall j \in N$)	CPU Time (s)	# Optimal
Artigues	3561.38	804.17	0.722×10^{-2}	n/a
Basic chaining	4163.92	944.58	0.251×10^{-1}	n/a
ISH_{flex}	3734.31	847.46	4.55	n/a
ISH^2_{flex}	3612.71	812.39	4.99	n/a
MinEA	3134.72	707.25	63.42	68
MaxPF	3125.24	705.24	155.57	90
MinED	–	–	–	–
MABO	2750.68	620.32	0.576	n/a
Lower bound	1605.95	360.12	9.90	n/a

CONCLUSIONS

We have described a reactive scheduling procedure for the RCPSP in the presence of activity duration variability. This procedure requires us to specify a resource flow network that indicates how resources are being transferred from one activity to another. We formulated the problem of determining a resource flow network such that the schedule stability cost (defined as the sum of the weighted deviations between the planned activity start times in the baseline schedule and the actually realized activity start times during project execution) is minimized. Our review of the literature revealed that research efforts in this area are still in an early stage.

We have presented three new heuristics based on surrogate MIP formulations of the basic strongly *NP*-hard problem. The *MinEA* heuristic minimizes the extra precedence relations imposed by the resource allocation decisions; the *MaxPF* heuristic maximizes the sum of pairwise floats in the network $G(N, A \cup A_R)$, and the *MinED* heuristic minimizes an approximation of the weighted stability cost. Furthermore, we developed a myopic resource allocation heuristic called MABO, a single-pass procedure that tries to construct a robust resource flow network by looking at one activity at a time and solving its resource allocation problem as well as possible. We also derived lower bounds on schedule stability cost.

The performances of *MinEA*, *MaxPF*, *MinED*, and MABO have been evaluated against four previously developed procedures on a set of randomly generated benchmark problems. All of the heuristics developed in this article proved to be superior to the existing algorithms. The *MinED* model obtained the best performance on the stability objective on problems with 30 or 60 activities. However, it found no feasible solution on problems with 120 activities within a time limit of 60 seconds. The MABO procedure provides very good results overall on the stability objective within a short computation time. Finally, the models *MinEA* and *MaxPF* did not obtain better results than the MABO procedure, while requiring longer computation times.

In this article, we described a straightforward reactive scheduling approach for projects with uncertain activity durations. It requires the project manager to maintain a fixed flow of resources between the activities throughout the execution of the project. The heuristics and algorithms proposed in this article provide us with an effective procedure for schedule repair, capable of saving on the numerous types of costs (storage costs, organizational costs, . . . , etc.) caused by delays in activity starting times.

The procedures developed in this article hold promise to have a strong managerial impact. They provide initial evidence of practical means for developing project baseline schedules that are robust with respect to disruptions that may occur during project execution. This is in sharp contrast with the "take it or leave it" single-schedule approach provided by existing commercial packages with no provision for scheduling for quality robustness and stability. Our procedures define possible means for evaluating the stability and relative schedule change costs. Deploying the approaches described in this article may reduce the disruptions and delays caused by preemption in multi-project settings (Bock & Patterson, 1990; Ash & Smith-Daniels, 1999) and may be used for enhancing the stability of multi-project schedules that are subject to uncertainty.

The procedures developed in this article assume that the uncertainty resides in the activity durations and that the stability of a baseline schedule can be enhanced through proper resource allocation. Research on the generation of stable baseline schedules subject to renewable resource disruptions is just emerging (Lambrechts et al., 2006a,b). Project management can also rely on alternative procedures for generating robust baseline schedules, for example, through the insertion of time buffers (Van de Vonder et al., 2005, 2006). Our future research efforts aim at the extension of our proactive/reactive scheduling methodology, under general precedence constraints and given different types of time, resource, cost, and scope uncertainty, and its integration within a global risk management procedure (Schatteman, Herroelen, Van de Vonder, & Boone, 2006). [Received: June 2006. Accepted: January 2007.]

REFERENCES

Aloulou, M., & Portmann, M. (2003). An efficient proactive reactive scheduling approach to hedge against shop floor disturbances. *1st Multidisciplinary Conference on Scheduling: Theory and Applications*, Nottingham, UK: Kluwer. 337–362.

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

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

Ash, R., & Smith-Daniels, D. E. (1999). The effects of learning, forgetting, and relearning on decision rule performance in multiproject scheduling. *Decision Sciences, 30*, 47–82.

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

Bock, D. B., & Patterson, J. H. (1990). A comparison of due date setting, resource assignment, and job preemption heuristics for the multiproject scheduling problem. *Decision Sciences, 21*, 387–402.

Bowers, J. (1995). Criticality in resource constrained networks. *Journal of the Operational Research Society, 46*, 80–91.

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

Cesta, A., Oddi, A., & Smith, S. F. (1998). Profile-based algorithms to solve multiple capacitated metric scheduling problems. *4th International Conference on Artificial Intelligence Planning Systems*, Pittsburgh, PA: AAAI Press, 214–231.

Debels, D., & Vanhoucke, M. (2006). *Future research avenues for resource constrained project scheduling: Search space restriction or neighbourhood search extension*. Research report, Ghent University, Ghent, Belgium.

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

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

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

Elmaghraby, S. E. E., Herroelen, W., & Leus, R. (2003). Note on the paper "Resource-constrained project management using enhanced Theory of Constraints." *International Journal of Project Management, 21*, 301–305.

Goldratt, E. (1997). *Critical chain*. Great Barrington, MA: The North River Press.

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

Herroelen, W., De Reyck, B., & 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*, 221–230.

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

Herroelen, W., & Leus, R. (2004a). The construction of stable project baseline schedules. *European Journal of Operational Research, 156*, 550–565.

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

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

Igelmund, G., & Rademacher, F. (1983a). Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks, 13*, 29–48.

Igelmund, G., & Rademacher, F. (1983b). Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks, 13*, 1–28.

Kolisch, R., & Hartmann, S. (1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In J. Weglarz (Ed.), *Project scheduling: Recent models, algorithms and applications*. Amsterdam, The Netherlands: Kluwer.

Kolisch, R., & Padman, R. (1999). An integrated survey of deterministic project scheduling. *Omega, 49*, 249–272.

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

Lambrechts, O., Demeulemeester, E., & Herroelen, W. (2006a). *Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities*. Research Report 0606, Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven.

Lambrechts, O., Demeulemeester, E., & Herroelen, W. (2006b). *A tabu search procedure for generating robust project baseline schedules under stochastic resource availabilities*. Research Report 0604, Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven.

Leus, R. (2003). *The generation of stable project plans*. PhD Thesis, Department of Applied Economics, Belgium: Katholieke Universiteit Leuven.

Leus, R., & Herroelen, W. (2004). Stability and resource allocation in project planning. *IIE Transactions, 36*, 1–16.

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

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

Policella, N. (2005). *Scheduling with uncertainty—A proactive approach using partial order schedules*. PhD Thesis, Italy: Università degli Studi di Roma "La Sapienza."

Policella, N., Oddi, A., Smith, S., & Cesta, A. (2004). Generating robust partial order schedules. *Proceedings of CP2004*, Toronto, Canada: Springer, 496–511.

Schatteman, D., Herroelen, W., Van de Vonder, S., & Boone, A. (2006). *A methodology for integrated risk management and proactive scheduling of construction projects*. Research Report. Research Center for Operations Management, Katholieke Universiteit Leuven, Belgium.

Smith-Daniels, D. E., & Aquilano, N. J. (1987). Using a late-start resource-constrained project schedule to improve project net present value. *Decision Sciences, 18*, 617–630.

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

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

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

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

Filip Deblaere is a research assistant and PhD candidate at the Research Center for Operations Management of the Faculty of Economics and Applied Economics of K.U.Leuven (Belgium). He earned a Master's degree in informatics and a Master's degree in management from K.U.Leuven (Belgium). His research focuses on proactive/reactive project scheduling.

Erik Demeulemeester is a professor of operations management at the Research Center for Operations Management of the Faculty of Economics and Applied Economics of K.U.Leuven (Belgium). Dr. Demeulemeester has published extensively in the project scheduling and health care management fields. He is coauthor of the book, *Project Scheduling—A Research Handbook*, published in the International Series in Operations Research & Management Science by Springer. His work has appeared in journals such as *Management Science*, *Operations Research*, *Journal of Scheduling*, *European Journal of Operational Research*, *International Journal of Production Research*, *International Journal of Production Economics*, *IIE Transactions*, *Journal of the Operational Research Society*, *Annals of Operations Research*, *Naval Research Logistics*, *Computers and Operations Research*, *Project Management Journal*, and *Journal of Medical Systems*. He is currently associate editor for *Computers and Operations Research*, *Journal of Scheduling*, and *European Journal of Industrial Engineering*. His current research focuses on proactive/reactive project scheduling and health care planning and scheduling.

Willy Herroelen is a professor of operations management at the Research Center for Operations Management of the Faculty of Economics and Applied Economics of the K.U.Leuven (Belgium). Dr. Herroelen has published widely in the project scheduling field. He is coauthor of the book, *Project Scheduling–A Research Handbook*, published in the International Series in Operations Research & Management Science by Springer. His work has appeared in journals such as *Management Science*, *Operations Research*, *Journal of Scheduling*, *European Journal of Operational Research*, *International Journal of Production Research*, *International Journal of Production Economics*, *IIE Transactions*, *Journal of Operations Management*, *Journal of the Operational Research Society*, *Annals of Operations Research*, *Naval Research Logistics*, *Omega*, *Operations Research Letters*, *Production and Operations Management*, *Project Management Journal*, and *Computers and Operations Research*. He is currently a senior editor for *Production and Operations Management* and serves on the editorial boards of *International Journal of Production Research*, *Foundation of Computers,Decision Sciences*, and *Journal of Operations and Logistics*. His current research focuses on proactive/reactive resource-constrained project scheduling.

Stijn Van de Vonder holds a Master's degree in management informatics from K.U.Leuven (Belgium) and obtained his PhD in applied economics at the Research

Center for Operations Management of the Faculty of Economics and Applied Economics of the K.U.Leuven (Belgium). He has published in *International Journal of Production Research*, *International Journal of Production Economics*, and various conference proceedings.