Computers & Operations Research 38 (2011) 63–74

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Reactive scheduling in the multi-mode RCPSP

Filip Deblaere *, Erik Demeulemeester, Willy Herroelen

Research Center for Operations Management, Department of Decision Sciences and Information Management, Faculty of Business and Economics, Katholieke Universiteit Leuven, Belgium

ARTICLE INFO

Available online 11 January 2010

Keywords:
Project scheduling
Reactive scheduling
MRCPSP

ABSTRACT

The multi-mode resource-constrained project scheduling problem (MRCPSP) involves the determination of a baseline schedule of the project activities, which can be executed in multiple modes, satisfying the precedence relations and resource constraints while minimizing the project duration. During the execution of the project, the baseline schedule may become infeasible due to activity duration and resource disruptions. We propose and evaluate a number of dedicated exact reactive scheduling procedures as well as a tabu search heuristic for repairing a disrupted schedule, under the assumption that no activity can be started before its baseline starting time. We report on promising computational results obtained on a set of benchmark problems.

1. Introduction

The resource-constrained project scheduling problem or RCPSP (problem $m,1|cpm|C_{max}$ in the notation of Herroelen et al. [1]) involves the scheduling of project activities subject to resource and precedence constraints, under the objective of minimizing the project makespan. In the multi-mode RCPSP or MRCPSP (problem $m,1T|cpm,mu|C_{max}$), activities can be executed in multiple modes and the availability of resource types may not only be specified per time period, but also for the entire project horizon T (in that case, we speak of *nonrenewable* resource types). The activity modes are characterized by a certain resource requirement and a duration. For instance, one can either build a wall with one worker in three days or with two workers in two days. A number of solution procedures for the deterministic MRCPSP have been proposed in the literature, both exact and heuristic. We refer the interested reader to Speranza and Vercellis [2], Drexl and Grünewald [3], Hartmann and Sprecher [4], Sprecher et al. [5], Hartmann and Drexl [6], Sprecher and Drexl [7], De Reyck and Herroelen [8], Hartmann [9], Jozefowska et al. [10], Alcaraz et al. [11], Heilmann [12], Buddhakulsomsiri and Kim [13], Lova et al. [14,15], Zhang et al. [16], Zhu et al. [17], Sabzehparvar and Seyed-Hosseini [18], Jarboui et al. [19] and Van Peteghem and Vanhoucke [20].

In recent years there has been a growing conscience that the traditional deterministic project scheduling models are in plain conflict with a reality that is characterized by uncertainty. As correctly noted by Hildum [21], an optimal schedule is only optimal to the extent that the reality behaves as expected during the execution of the schedule. For review papers on production scheduling under uncertainty, we refer to Davenport and Beck [22], Vieira et al. [23] and Aytug et al. [24]. For a review of project scheduling under uncertainty, we refer the reader to Herroelen and Leus [25,26].

In project scheduling, uncertainty can take many different forms. Activity duration estimates may be off, resources may break down, work may be interrupted due to weather delay, new unanticipated activities may be identified, etc. All of these types of uncertainty may result in the infeasibility of the project baseline schedule. In general, project management wants to avoid these schedule breakages. This can be achieved by generating a baseline schedule in a *proactive* way, trying to anticipate certain types of disruptions so as to minimize their effect if they occur. If the schedule would still break down despite these proactive planning efforts, a *reactive* scheduling policy will be needed to repair the infeasible schedule.

In the field of proactive and reactive project scheduling for the single-mode RCPSP some work has already been done. The problem of coping with activity duration variability has been tackled in Van de Vonder [27] and Van de Vonder et al. [28,29]. The problem of uncertainty with respect to resource availability has been addressed by Lambrechts et al. [30,31]. The literature on proactive/reactive scheduling policies in the multi-mode RCPSP, however, is virtually void. To the best of our knowledge, there is only the work by Zhu et al. [32], where a branch-and-cut procedure is proposed for a general class of reactive scheduling problems. The authors assume the presence of a *recovery window* that limits the set of feasible reactive schedules to those schedules that are "back on track" from a certain time point onward. Computational results are reported for the generation of reactive schedules in response to an activity duration disruption on a set of 20-activity instances.

* Corresponding author.
E-mail address: filip.deblaere@econ.kuleuven.be (F. Deblaere).

0305-0548/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2010.01.001

In this paper, we propose a number of solution procedures for the multi-mode reactive scheduling problem that are capable of solving 20-activity as well as 30-activity benchmark instances within a reasonable computation time. The procedures are able to handle both activity duration disruptions as well as resource disruptions. The structure of the paper is as follows. In Section 2, we give a description of the basic MRCPSP and the associated reactive scheduling problem. For solving this problem, a number of dedicated exact search procedures as well as a tabu search heuristic are proposed in Section 3. In Section 4 we test the procedures on 20- and 30-activity benchmark problems. In the final section, we provide overall conclusions.

2. Problem description

2.1. The basic deterministic MRCPSP

The basic deterministic MRCPSP can be described as follows. We assume that the project is represented as an activity-on-the-node network $G(N,A)$ with a set of nodes $N=\{1,2,\ldots,n\}$ representing the project activities and a set of arcs A representing the zero-lag finish–start precedence constraints between the activities. We assume that activities 1 and n denote the dummy start and the dummy end activity, respectively. We denote M_i as the set of available modes for an activity i, each mode being the combination of a certain activity duration and a certain resource requirement. The duration of an activity i executed in a mode $m_i \in M_i$ is denoted as d_{im_i}.

We assume the presence of a set $\mathcal{K}^\rho$ of n^ρ *renewable* resource types and a set $\mathcal{K}^\nu$ of n^ν *nonrenewable* resource types. Contrary to renewable resource types, the availability of nonrenewable resource types is specified for the entire project horizon T. These resource types are useful for modelling e.g. a limited budget for the execution of the project. Extra activity execution costs (such as overtime or weekend work) corresponding with shorter execution modes can then be modelled through the use of a higher amount of these nonrenewable resources. *Doubly constrained* resource types can be considered as the resource types in the intersection of $\mathcal{K}^\nu$ and $\mathcal{K}^\rho$. Hence, we do not need to consider them explicitly. The project activities $i \in N$ executed in a mode $m_i \in M_i$ require an integer per-period amount $r^\rho_{im_ik}$ of the renewable resource type k, $k \in \mathcal{K}^\rho$ and an integer amount $r^\nu_{im_ik}$ of the nonrenewable resource type k, $k \in \mathcal{K}^\nu$. We denote a^ρ_k as the per-period availability of the renewable resource type k ($k \in \mathcal{K}^\rho$) and a^ν_k as the total availability of the nonrenewable resource type k ($k \in \mathcal{K}^\nu$).

The objective of the basic MRCPSP is to determine a schedule consisting of activity starting times s_i and activity execution modes m_i ($m_i \in M_i$) such that the precedence as well as the resource constraints are satisfied, while minimizing the project makespan. An example MRCPSP instance along with a (suboptimal) baseline schedule is shown in Fig. 1. The example project depicted in Fig. 1(a) has one renewable resource type with a per-period availability equal to five units and no nonrenewable resource types. The possible activity durations are shown above the activity nodes and the corresponding resource requirements per time period are shown below the nodes. When multiple execution modes are possible, these are separated by a vertical bar. For instance, activity 3 can either be executed in two time units using four units of resource per period or in three time units using only two units of resource per period. Activities 1 and 9 are the dummy start and the dummy end activity, respectively. The resource profile of the baseline schedule is shown in Fig. 1(b). Whenever multiple activity execution modes are possible, the selected mode is indicated between brackets. The example project shown in Fig. 1(a) along with its baseline schedule shown in Fig. 1(b) will be used as an illustrative example throughout this paper.

2.2. Schedule disruptions and rescheduling costs

In a realistic project environment one has to cope with a considerable amount of uncertainty. If an unanticipated disruptive event occurs (e.g. a machine breakdown), the baseline schedule may become infeasible. Project management must then rely on a *reactive procedure* to adapt the schedule to the new information and to restore its feasibility. In most situations it is important that the repaired schedule bears close resemblance to the original baseline schedule. Any deviation from the baseline schedule may lead to undesirable side-effects, such as having to change agreements with subcontractors, accumulating inventory costs, dealing with employee malcontent, etc. Moreover, if resources with scarce availability need to be reserved in advance, any disruption of the schedule may lead to a large delay of the activities in need of the scarce resources, which may ultimately lead to a violation of the project deadline.

We assume that during the execution of the project, no activity can be started before its baseline starting time. This constraint is commonly referred to as the *railroad scheduling policy*, due to the analogy with railroad scheduling, where trains do not depart before their scheduled time of departure. In a practical context, it will often be impossible to start activities earlier than planned, for instance because the necessary materials have not yet been delivered to the site or because of inflexible agreements with subcontractors. As a consequence of this constraint, the reactive starting times s'_i must always be at least as large as their baseline counterparts s_i, and deviations from the baseline starting times will always be positive deviations. We can capture the magnitude

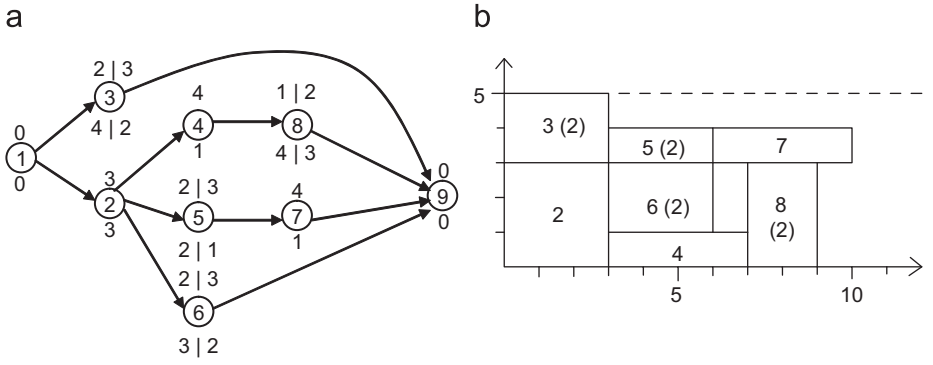

Fig. 1. An example MRCPSP instance.

of the costs incurred when increasing the starting time of an activity $i \in N$ by one time unit through a nonnegative *inflexibility weight* w_i. To give an intuitive explanation for the weight w_i, consider e.g. a construction project. An activity suffering from a positive deviation may for instance require a crane (i.e. an inflexible resource that needs to be reserved in advance) and construction materials. The weight w_i could reflect the cost of the prolonged need for the crane as well as additional storage costs for the required materials. In our computational experiments (see Section 4) we simply assume that the w_i are nonnegative, but in a practical context one could for instance let the value of the w_i depend on the "closeness" of the planned starting time of activity i relative to the time when a disruption occurs causing the need to reschedule. Indeed, activities that are planned to start in the "far future" may for the time being still be considered as flexible, because no binding agreements have already been made that are based on their planned starting times. In the reactive scheduling problem, this can be reflected by setting the inflexibility weight of the corresponding activities equal to zero.

In a multi-mode context, in addition to delaying activities in order to repair a broken schedule, we might also consider changing the mode of certain activities. Indeed, changing an activity's mode can enable us to speed up the execution of that activity, allowing us to get the schedule back on track sooner, at the price of a certain *mode switching cost* $c_{i(m_i,m'_i)}$ reflecting the cost of switching the baseline mode m_i of activity i to the reactive mode m'_i. Note that monetary costs that are inherent in executing an activity in a certain mode should be reflected by their nonrenewable resource consumption. The difference between those monetary costs when switching from one mode to another should thus not be included into the mode switching costs. The mode switching costs should only reflect "administrative" costs that need to be made when a certain mode switch is executed.

Since we assume that the baseline modes of all activities i are given, we can simplify the notation of the mode switching costs to $c_{im'_i}$. We assume that no mode switching cost is incurred when the reactive mode is the same as the baseline mode. In other words, we assume $c_{im_i} = 0, \forall i \in N$. This reflects our desire not to change the mode of an activity unless this results in a reduced cost incurred due to activity starting time deviations. The inflexibility weights and the mode switching costs for the example instance of Fig. 1 are shown in Table 1. Note that in the baseline schedule of Fig. 1(b) all activities with two modes are executed in their second mode, hence we have $c_{i2} = 0$ for these activities. Evidently, mode switching costs are irrelevant for activities with only one execution mode.

Table 1
Inflexibility weights and mode switching costs.

i	1	2	3	4	5	6	7	8	9
w_i	0	2	1	2	2	5	1	12	15
c_{i1}	0	0	5	0	4	1	0	4	0
c_{i2}	–	–	0	–	0	0	–	0	–

Given a baseline schedule of activity starting times $s_1, s_2, \ldots, s_n$ and execution modes $m_1, m_2, \ldots, m_n$, the objective of the reactive scheduling procedure we propose is to produce a revised schedule of reactive starting times $s'_1, \ldots, s'_n$ with $s'_i \geq s_i$ for $i = 1, \ldots, n$; and execution modes $m'_1, \ldots, m'_n$ such that the *rescheduling cost* $\mathcal{C}$ given by Eq. (1) is minimized.

$$\mathcal{C} = \sum_{i \in N} w_i(s'_i - s_i) + \sum_{i \in N} c_{im'_i} \tag{1}$$

2.2.1. Activity duration disruptions

In this paper two types of schedule disruptions will be explicitly considered, namely activity duration disruptions and resource disruptions. In the case of an activity duration disruption we assume that during the execution of some activity i, it becomes clear that the duration of that activity will be an amount Δ_i higher than its deterministic baseline activity duration d_{im_i}. This case is illustrated in Fig. 2. At time instant $t = 2$ it becomes clear that activity 2 will take four instead of three time units to complete ($\Delta_2 = 1$). This disruption is illustrated in Fig. 2(a). Fig. 2(b) presents an optimal reactive schedule, in which the starting times of activities 4, 5, 6 and 8 are delayed for one time unit. The mode of activity 5 has been switched, enabling activity 7 to start at its baseline starting time, which ensures that the baseline project makespan of 10 time units is met. The reader can verify that for this example the rescheduling cost $\mathcal{C}$ equals 25.

2.2.2. Renewable resource disruptions

In the case of a renewable resource disruption we assume that the availability of a renewable resource type $k \in \mathcal{K}^\rho$ at time instant t suddenly decreases by an amount Δ_k^ρ. We furthermore assume that the project manager is able to make a prediction of the time period $t + \delta_t$ where the availability of that resource type will be restored to its original level. If the disruption is sufficiently severe, we may not be able to continue the execution of the project as planned in the baseline schedule. In that case we need to select a set of activities that are active at time instant t such that, when delaying the execution of these activities to a later point in time, the remaining activities that are active at time instant t can be executed within the reduced resource availability. Given such a (minimal) delaying alternative, renewable resource disruptions can be treated in a way that is very similar to activity duration disruptions. For reasons of clarity, we assume a preempt-repeat environment, where interrupted activities must be re-executed from scratch. Although this is perhaps not the most realistic setting, using this assumption will allow us to avoid issues regarding residual work content of preempted activities that would arise in a preempt-resume environment.

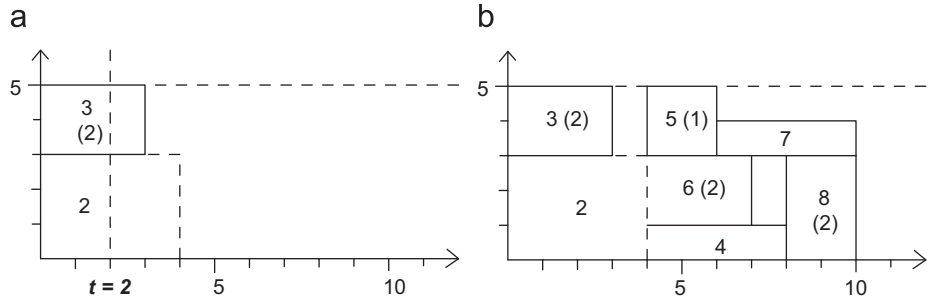

Fig. 2. Activity duration disruption.

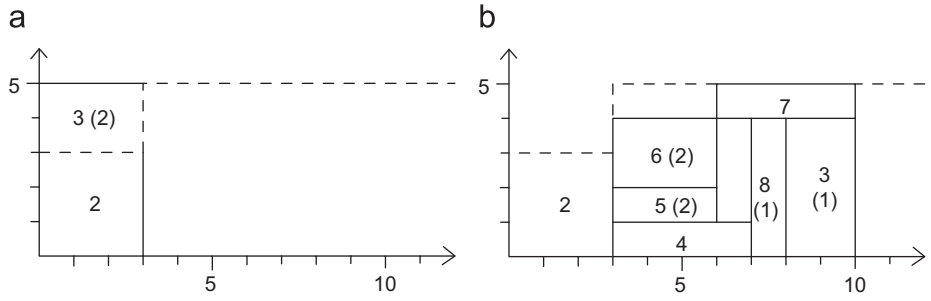

Fig. 3. Resource disruption.

An example of a renewable resource disruption is given in Fig. 3. Already at time instant $t=0$ the availability of the single renewable resource type drops with an amount $\Delta_1^\rho=2$ and is predicted to be restored at time instant $t+\delta_t=3$, as shown in Fig. 3(a). As a consequence, the execution of activities 2 and 3 cannot be continued as planned and reactive measures need to be taken. An optimal reactive schedule is presented in Fig. 3(b). The highly flexible activity 3 ($w_3=1$, see Table 1) is delayed for eight time units, until time instant $t=8$. By switching the modes of activities 3 and 8, we are again able to meet the projected makespan of 10 time units. This set of reactive measures results in a total rescheduling cost $\mathcal{C}=17$.

2.2.3. Nonrenewable resource disruptions

In the case of a nonrenewable resource disruption, it is assumed that the availability of a nonrenewable resource type $k\in\mathcal{K}^\nu$ drops with an amount Δ_k^ν. If this results in an infeasible schedule with respect to the nonrenewable resources, a reactive schedule must be sought such that the reactive modes allow all activities to be executed within the reduced nonrenewable resource availability, while the resulting rescheduling costs are minimized.

2.2.4. Multiple disruptions

The algorithms we describe in the next section treat the case where a single disruption has occurred at a certain time instant t^*. During the execution of the project, however, multiple disruptions can occur over time. In that case, the reactive scheduling procedure may be invoked more than once. In this section we describe how the model can be used to treat multiple subsequent disruptions.

Earlier in this section, we have mentioned that the inflexibility weight of an activity can vary over time depending on the planned starting time of the activity. We can formalize this by associating with every activity i an *inflexibility period length* $\mathcal{T}_i$ with $0\leq\mathcal{T}_i\leq s_i$, such that any agreements that need to be made with third parties that rely on an accurate estimate of the starting time of the activity must be made by time instant $s_i-\mathcal{T}_i$. If a disruption occurs in the interval $[0,s_i-\mathcal{T}_i)$, the inflexibility weight w_i may be set equal to zero, because no binding agreements will have already been made. If a disruption occurs during the *inflexibility period* $[s_i-\mathcal{T}_i,s_i+d_{im_i})$, the planned starting time s_i has become "locked in" and the weight of activity i must be set equal to its normal value, reflecting the costs per time unit of deviation from s_i. When a reactive schedule $S'=\{s'_1,\ldots,s'_n\}$ has been constructed and deployed, the reactive starting times s'_i will act as the new "baseline" starting times s_i. Likewise, the reactive modes m'_i will become the new baseline modes m_i and the mode switching costs $c_{im'_i}$ should be updated accordingly (recall that $c_{im'_i}$ is shorthand for $c_{i(m_i,m'_i)}$).

In what follows, we assume that the s_i, m_i, w_i and $c_{im'_i}$ have the correct values given the disruption time instant t^*.

3. Tree-based exact search techniques for the reactive scheduling problem

3.1. Branching scheme

We assume that the baseline schedule is executed as planned, up to a certain decision time t^*, at which a disruption occurs. As mentioned in the previous section, this can either be an activity duration disruption or a resource disruption. The objective is then to construct a reactive schedule of the unfinished activities that is feasible with respect to the newly available information, and minimizes the rescheduling cost $\mathcal{C}$ as defined in Eq. (1). To solve this reactive scheduling problem tree-based search techniques can be used that are similar to those devised for solving the basic MRCPSP. As mentioned in Hartmann and Drexl [6], three categories of branching schemes for the MRCPSP are described in the literature, namely precedence tree enumeration, branching on mode and delaying alternatives, and branching on mode and extension alternatives. In this paper, we propose a branching procedure based on mode and delaying alternatives.

The reactive scheduling procedure is formally described in Algorithm 1, and this procedure uses the branching scheme shown in Algorithm 2, the logic of which largely corresponds with the branching scheme based on mode and delaying alternatives described by Sprecher et al. [5]. In Algorithm 2, a new node is obtained at the end of Step 1. Such a node, obtained on a level l of the search tree, is essentially characterized by a resource and precedence feasible partial schedule PS_l. This partial schedule consists of a decision time t_l, a set of scheduled activities $\mathcal{S}_l$, a set of reactive starting times s'_i, $\forall i\in\mathcal{S}_l$ and a set of reactive modes m'_i, $\forall i\in\mathcal{S}_l$. In the procedure, we will keep track of three auxiliary activity sets $\mathcal{A}_l$, $\mathcal{P}_l$ and $\mathcal{E}_l$. At the end of Step 1, the contents of these sets can be derived from the partial schedule PS_l. The set $\mathcal{A}_l$ is the set of active activities, i.e. those activities $i\in\mathcal{S}_l$ that have not yet finished by time instant t_l. From PS_l we can also calculate $\mathcal{P}_l$, the set of *pending* activities: this is the set of unscheduled activities for which all predecessors have already finished at or before time instant t_l. The set $\mathcal{E}_l\subseteq\mathcal{P}_l$ of *eligible* activities is then defined as those pending activities that can be started without violating the railroad scheduling constraint. In Step 2, the eligible activities are started. Some of these activities will have been assigned a mode at a higher level of the tree, but were delayed because of a renewable resource conflict. The other eligible activities have not yet been assigned a mode. This set of currently "modeless" activities is denoted as $\mathcal{E}'_l\subseteq\mathcal{E}_l$.

Algorithm 1. Reactive scheduling procedure.

set $l=1$, $\mathcal{S}_l=\{i\in N|s_i\le t^*\}$ and $\mathcal{A}_l=\{i\in\mathcal{S}_l|s_i+d_{im_i}>t^*\}$
$\forall i\in\mathcal{S}_l$, set $s'_i=s_i$ and $m'_i=m_i$.

- in the case of a **duration disruption** of activity i:
  set $d_{im'_i}=d_{im'_i}+\Delta_i$
  perform Algorithm 2 (branching on mode and delaying alternatives)
- in the case of a **renewable resource disruption**:
  identify all minimal delaying alternatives $\mathcal{D}_{0j}$ that solve the resource conflict caused by the resource disruption
  for every $\mathcal{D}_{0j}$:
    set $\mathcal{S}_l=\mathcal{S}_l\backslash\mathcal{D}_{0j}$, $\mathcal{A}_l=\mathcal{A}_l\backslash\mathcal{D}_{0j}$
    $\forall i\in\mathcal{D}_{0j}$, discard the mode assignment made to activity i
    perform Algorithm 2 (branching on mode and delaying alternatives)
- in the case of a **nonrenewable resource disruption**:
  given the affected resource type k, set $a_k^v=a_k^v-\Delta_k^v$
  perform Algorithm 2 (branching on mode and delaying alternatives)

Algorithm 2. Branching on mode and delaying alternatives.

Step 1 (**Next decision time**):
  if $n\in\mathcal{A}_l$ store the current solution and go to Step 7
  calculate $t_l=\min(\min_{i\in\mathcal{A}_l}(s'_i+d_{im'_i}),\min_{i\in N\backslash\mathcal{S}_l}(s_i))$
  set $\mathcal{A}_l=\mathcal{A}_l\backslash\{i\in\mathcal{A}_l|s'_i+d_{im'_i}=t_l\}$
Step 2 (**Determine and start eligible activities**):
  calculate $\mathcal{P}_l=\{j\in(N\backslash\mathcal{S}_l)|\forall(i,j)\in A: i\in\mathcal{S}_l \text{ and } s'_i+d_{im'_i}\le t_l\}$
  calculate $\mathcal{E}_l=\{i\in\mathcal{P}_l|t_l\ge s_i\}$
  if $\mathcal{E}_l=\emptyset$, go to Step 1, else start the activities in $\mathcal{E}_l$: set $\mathcal{S}_l=\mathcal{S}_l\cup\mathcal{E}_l$, $\mathcal{A}_l=\mathcal{A}_l\cup\mathcal{E}_l$ and $\forall i\in\mathcal{E}_l$, set $s'_i=t_l$
Step 3 (**Calculate mode alternatives**):
  determine $\mathcal{E}'_l\subseteq\mathcal{E}_l$ as the set of eligible activities for which no mode has been assigned at a higher level $l'<l$ of the tree.
  calculate $\mathcal{M}_l=\{\mathcal{M}_{l1},\ldots,\mathcal{M}_{lp}\}$, the set of mode alternatives for $\mathcal{E}'_l$ that do not violate the nonrenewable resource constraints.
Step 4 (**Select next mode alternative**):
  if no untested mode alternative remains, go to Step 7
  select the next untested mode alternative $\mathcal{M}_{lj}\in\mathcal{M}_l$
  execute the mode alternative: $\forall i\in\mathcal{E}'_l$: set $m'_i=\mathcal{M}_{lj}(i)$
  if $\exists k\in\mathcal{K}^\rho:\sum_{i\in\mathcal{A}_l}r^\rho_{im'_ik}>a^\rho_k$, go to Step 5
  else set $\mathcal{S}_{l+1}=\mathcal{S}_l$, $\mathcal{A}_{l+1}=\mathcal{A}_l$, $l=l+1$ and go to Step 1
Step 5 (**Calculate minimal delaying alternatives**):
  calculate $\mathcal{D}_l=\{\mathcal{D}_{l1},\ldots,\mathcal{D}_{lq}\}$, the set of minimal delaying alternatives
Step 6 (**Select next minimal delaying alternative**):
  if no untested minimal delaying alternative remains, go to Step 4
  select the next untested minimal delaying alternative $\mathcal{D}_{lj}\in\mathcal{D}_l$
  set $\mathcal{S}_{l+1}=\mathcal{S}_l\backslash\mathcal{D}_{lj}$, $\mathcal{A}_{l+1}=\mathcal{A}_l\backslash\mathcal{D}_{lj}$, $l=l+1$ and go to Step 1
Step 7 (**Backtracking**):
  set $l=l-1$. If $l=0$, STOP; else go to Step 6

For the activities in $\mathcal{E}'_l$, all mode alternatives will be considered (see Step 3). A *mode alternative* $\mathcal{M}_{lj}$ is a mapping that assigns a certain mode $\mathcal{M}_{lj}(i)\in M_i$ to every activity $i\in\mathcal{E}'_l$. To illustrate how mode alternatives are enumerated, suppose that at a certain decision time there are two activities i and j that need to be assigned a mode, and that both activities have three modes, ordered according to increasing duration. In that case, the reactive modes (m'_i,m'_j) will be enumerated as follows: (0,0); (0,1); (0,2); (1,0); (1,1); (1,2); (2,0); (2,1); (2,2). This way, a certain preference will be given to shorter durations. Note that there is no guarantee that shorter modes will lead to better reactive schedules. Given a mode alternative, starting all eligible activities may lead to a renewable resource conflict. To solve this resource conflict, a number of minimal delaying alternatives will be identified in Step 5. By delaying the start of the activities of such a minimal delaying alternative, we can eliminate the resource conflict at the current decision time. Formally, a *delaying alternative* $\mathcal{D}_{lj}$ is a subset of the set of active activities $\mathcal{A}_l$ such that $\sum_{i\in\mathcal{A}_l\backslash\mathcal{D}_{lj}}r^\rho_{im'_ik}\le a^\rho_k,\forall k\in\mathcal{K}^\rho$. A delaying alternative $\mathcal{D}_{lj}$ is called *minimal* if no proper subset of $\mathcal{D}_{lj}$ is also a delaying alternative. In Step 6, the next untested minimal delaying alternative with the lowest lower bound (to be discussed in the next section) is selected for branching. The corresponding activities are delayed and we return to Step 1 where the next decision time for the new node is calculated.

The branching scheme described in Algorithm 2 is used by Algorithm 1 for reactive scheduling after a disruption. Given the part of the schedule that has already been executed and given a disruption, Algorithm 1 will construct a partial reactive schedule PS_1 and a set of active activities $\mathcal{A}_1$ to serve as inputs for Algorithm 2.

3.2. Lower bound

When a first feasible solution for the reactive scheduling problem is found, the objective value of that solution can be used as an upper bound. Nodes in the search tree can then be dominated by means of the lower bound on the rescheduling cost that is computed by means of Algorithm 3. The lower bound consists of the sum of two terms. The first term simply consists of the rescheduling costs that directly result from the scheduling decisions involving the activities in $\mathcal{S}_l$, the set of scheduled activities in the partial reactive schedule. The second term consists of unavoidable rescheduling costs due to activity starting time deviations of the unscheduled activities in PS_l. To calculate this term, we need the earliest starting times es_i of the unscheduled activities $i\in(N\backslash\mathcal{S}_l)$. These starting times are calculated ignoring the renewable resource constraints and assuming that all unscheduled activities i are executed in their shortest nonrenewable resource feasible mode m_i^*. Clearly, as the earliest starting times es_i are lower bounds on the actual reactive starting times s'_i obtainable given the partial reactive schedule, Algorithm 3 gives us a lower bound on the rescheduling cost $\mathcal{C}$ obtainable by branching from PS_l.

Algorithm 3. Lower bound.

Input: partial reactive schedule PS_l
Output: lower bound LB
$\forall i\in N\backslash\mathcal{S}_l$: determine the shortest mode $m_i^*\in M_i$ such that:
$r^v_{im_i^*k}+\sum_{j\in\mathcal{S}_l}r^v_{jm'_jk}\le a^v_k,\quad\forall k\in\mathcal{K}^v$
  if no such mode exists, set $LB=\infty$ and STOP
Given the partial reactive schedule PS_l, determine the earliest starting times $es_i\ge s_i$, $\forall i\in N\backslash\mathcal{S}_l$, assuming that these activities are executed in their shortest feasible modes m_i^* and ignoring the renewable resource constraints.
$LB=\sum_{i\in\mathcal{S}_l}[w_i(s'_i-s_i)+c_{im'_i}]+\sum_{i\in N\backslash\mathcal{S}_l}w_i(es_i-s_i)$

The lower bound can be calculated in Step 4 of Algorithm 2 if the current mode assignment does not result in a renewable resource conflict, or in Step 6 otherwise. Evidently, whenever a

new best solution is found, the upper bound should be updated as well.

3.3. Dominance rules

3.3.1. Data reduction

A first set of dominance criteria can be achieved implicitly by pre-processing the input data. Redundant renewable and nonrenewable resources can be identified, and redundant or non-executable activity modes can be removed. Furthermore, the availabilities of the nonrenewable resource types as well as the requirements for these nonrenewable resources can be decreased, so as to detect nonrenewable resource conflicts earlier in the search process. These preprocessing steps have been well documented by other authors, hence we will not dwell upon the details here. We refer the interested reader to e.g. Sprecher et al. [5].

3.3.2. Left-shift dominance rule

Another well-known dominance rule that can be readily applied to our reactive scheduling problem is the left-shift rule (see e.g. Hartmann and Drexl [6]). Indeed, because our objective function is a regular performance measure, left-shifting an activity can never lead to an increase in the rescheduling cost. The variant we use is known as the *single-mode local left-shift rule*. This rule states that a partial reactive schedule in which an activity i can be locally left-shifted without changing its mode and without violating the railroad scheduling constraint $s'_i \geq s_i$ and the renewable resource constraints, is dominated. This rule can be checked in Step 6 of Algorithm 2, before branching into the next untested minimal delaying alternative. For details on the implementation of such a left-shift rule, we refer the interested reader to Demeulemeester and Herroelen [33].

3.3.3. Cutset rule

The so-called cutset rule is a very powerful dominance rule. It has been successfully applied in a wide variety of project scheduling problems (see e.g. Demeulemeester and Herroelen [33], Hartmann and Drexl [6] and Demeulemeester et al. [34]). To the best of our knowledge, the cutset dominance rule has never been formally described by other authors for the use in a branching scheme for the MRCPSP based on mode and delaying alternatives. Hartmann and Drexl [6] give an informal description of such a rule, but immediately add that in their computational experiments, their implementation of the cutset rule was unable to achieve a speed-up. As a consequence, the authors do not give a formal description of the rule. In this paper, we describe a cutset dominance rule for multi-mode resource constrained projects when a branching scheme similar to the one shown in Algorithm 2 is used, adapted to work with our objective function (1). As will be shown in Section 4, our cutset dominance rule does yield an important reduction in computation times.

The gist of any cutset rule is that, given the partial reactive schedule obtained in the current node of the search tree, no better solutions will be found when branching from this node than the ones that have been obtained by branching from a certain previously visited node. Given a partial reactive schedule PS_l, we define a cutset as $\mathcal{S}_l$, the set of scheduled activities in PS_l. When denoting the starting time, finish time and mode of an activity $i \in PS_l$ as s'_{il}, f'_{il} and m'_{il}, respectively, we can formulate the following theorem:

Theorem 1 (*Cutset dominance*). *A cutset $\mathcal{S}_b$ is dominated by a previously saved cutset $\mathcal{S}_a$ encountered on a different path of the search tree, if all of the following conditions are satisfied*:

- $\mathcal{S}_a = \mathcal{S}_b$ *and* $t_a \leq t_b$;
- $\sum_{i \in \mathcal{S}_a} r^v_{im'_{ia}k} \leq \sum_{j \in \mathcal{S}_b} r^v_{jm'_{jb}k}$, $\forall k \in \mathcal{K}^v$;
- $\forall i \in \mathcal{A}_a \backslash \mathcal{A}_b : f'_{ia} \leq t_b$;
- $\forall i \in \mathcal{A}_a \cap \mathcal{A}_b : f'_{ia} \leq f'_{ib}$, *and* $m'_{ia} = m'_{ib}$;
- $\sum_{i \in \mathcal{S}_a \backslash (\mathcal{A}_a \cap \mathcal{A}_b)} [w_i(s'_{ia} - s_i) + c_{im'_{ia}}] \leq \sum_{i \in \mathcal{S}_b \backslash (\mathcal{A}_a \cap \mathcal{A}_b)} [w_i(s'_{ib} - s_i) + c_{im'_{ib}}]$.

The proof of Theorem 1 can be found in Deblaere et al. [35]. If the above conditions are satisfied, then any schedule that can be obtained by completing the partial reactive schedule PS_b can also be obtained by completing PS_a with no higher resulting rescheduling cost. Hence, no better solutions will be found by completing PS_b, and the node associated with the cutset $\mathcal{S}_b$ can be fathomed. If the first four conditions of Theorem 1 are satisfied, it is easy to see that any schedule that can be obtained by completing PS_b can also be obtained by completing PS_a. If in addition to the first four conditions, the final condition is also satisfied, this latter continuation of PS_a will have no higher rescheduling cost than the corresponding continuation of PS_b. Note that in the final condition, we need to exclude the activities $i \in (\mathcal{A}_a \cap \mathcal{A}_b)$ from the two sides of the inequality, because it may be the case for some $i \in (\mathcal{A}_a \cap \mathcal{A}_b)$ that $s'_{ia} < s'_{ib}$. For the cutset dominance rule to be correct, the reduction in rescheduling costs that will follow from this latter inequality may not be used to compensate for other sunk costs inherent in PS_a that are not present in PS_b.

The integration of the cutset dominance rule in Algorithm 2 can be achieved by modifying Step 2 as follows:

Step 2 (**Determine and start eligible activities**):
calculate $\mathcal{P}_l = \{j \in (N \backslash \mathcal{S}_l) | \forall (i,j) \in A : i \in \mathcal{S}_l$ and $s'_i + d_{im'_i} \leq t_l\}$
calculate $\mathcal{E}_l = \{i \in \mathcal{P}_l | t_l \geq s_i\}$; if $\mathcal{E}_l = \emptyset$, go to Step 1, else
calculate $\tilde{\mathcal{E}}_l \subseteq \mathcal{E}_l$: the set of eligible activities to which a mode has been assigned at some level $l' < l$ of the tree;
set $\mathcal{E}_l = \mathcal{E}_l \backslash \tilde{\mathcal{E}}_l$ and start the activities in $\tilde{\mathcal{E}}_l$:
set $\mathcal{S}_l = \mathcal{S}_l \cup \tilde{\mathcal{E}}_l$, $\mathcal{A}_l = \mathcal{A}_l \cup \tilde{\mathcal{E}}_l$ and $\forall i \in \tilde{\mathcal{E}}_l$, set $s'_i = t_l$
if $\nexists k \in \mathcal{K}^\rho : \sum_{i \in \mathcal{A}_l} r^\rho_{im'_i k} > a^\rho_k$, check for cutset dominance. If $\mathcal{S}_l$ is dominated, go to Step 7, else save the cutset $\mathcal{S}_l$, the decision time t_l, the set of active activities $\mathcal{A}_l$, their starting times s'_i and execution modes m'_i and the nonrenewable resource requirements req^v_k of the partial reactive schedule PS_l, defined as $req^v_k = \sum_{i \in \mathcal{S}_l} r^v_{im'_i k}, \forall k \in \mathcal{K}^v$.
start the remaining eligible activities:
set $\mathcal{S}_l = \mathcal{S}_l \cup \mathcal{E}_l$, $\mathcal{A}_l = \mathcal{A}_l \cup \mathcal{E}_l$ and $\forall i \in \mathcal{E}_l$, set $s'_i = t_l$

As can be seen from the revised Step 2, we check for cutset dominance *after* we restart any delayed activities (these are the activities in the set $\tilde{\mathcal{E}}_l$) but *before* we start any activities that have become eligible for the first time on the path from the root node to the current node and as a consequence, have not yet been assigned a mode.

3.3.4. Resource infeasibility rule

Throughout the search, it can happen that the mode assignments that are made in a partial reactive schedule PS_l render a feasible mode assignment (w.r.t. the nonrenewable resource constraints) impossible for the set of unscheduled activities $N \backslash \mathcal{S}_l$. In that case, the node corresponding to the partial reactive schedule PS_l needs no further exploration. This kind of infeasibility can be detected in a relatively efficient way by means of a breadth-first search procedure. The details of this procedure can be found in Deblaere et al. [35]. In short, the procedure enumerates mode alternatives for the set of unscheduled

activities and tries to find a mode alternative that can be executed within the residual resource availability. If no such mode alternative can be found, the corresponding node can be fathomed.

3.4. Search strategy

3.4.1. Regular branch-and-bound

Using the branching scheme presented in Algorithm 2 and the dominance rules described above, we can construct a classical branch-and-bound procedure, selecting at each level l of the search tree the node with the lowest lower bound and branching from that node in a depth-first fashion. This approach usually works fine, but for this particular scheduling problem, other approaches might work better. There are a number of reasons for this:

- The basic MRCPSP (and hence also the reactive scheduling problem) is computationally highly intractable. When n nondummy activities can be scheduled in m different modes, this results in a total of m^n possible mode alternatives, each of which can be seen as an instance of the basic RCPSP.
- The lower bound we propose (see Algorithm 3) is very weak, because no strong assumptions can be made about the modes of the unscheduled activities. Moreover, Algorithm 2 does not select mode alternatives in an intelligent way. We cannot use the lower bound for selecting mode alternatives, because if we did, preference would always be given to modes yielding short activity durations, as renewable resource constraints are ignored in the lower bound.
- Due to the structure of the problem (i.e. the presence of activity weights and the nature of the objective function), large parts of the tree will contain only inferior solutions. Bad branching choices (either with respect to mode alternatives or with respect to delaying alternatives) can lead us into such a region. If this happens early in the search process, we will be wasting a lot of time finding nothing but bad solutions. Moreover, after finishing the exploration of such a region, the upper bound will be still so high that hardly any progress will have been made.

The above considerations, along with empirical results confirming these, have led us to explore other search strategies that try to avoid these problems.

3.4.2. Iterative deepening A^*

Iterative Deepening A^* or IDA^* (Korf [36]) is a tree-based optimal search technique that is related to best-first search. Given a root node *root*, a lower bound function *LB* and a function *generate_children* that generates the child nodes of a given node, the IDA^* procedure can be formulated as shown in Algorithm 4. The procedure IDA^* uses an auxiliary procedure *lb_limited_search*, shown in Algorithm 5.

Algorithm 4. IDA^*.

```
lb ← LB(root)
while (no leaf reached)
  lb ← lb_limited_search(lb)
```

Algorithm 5. *lb_limited_search*.

Input: lower bound *lb*

Output: either a new lower bound *lb_new* or an optimal leaf node

```
QUEUE ← {root}
lb_new ← ∞
while ((QUEUE ≠ ∅ and (no leaf reached))
  node ← first node in QUEUE
  remove node from QUEUE
  C ← generate_children(node)
  ∀c ∈ C:
    if c is a leaf: STOP with the optimal solution c
    else if LB(c) ≤ lb: add c to the front of QUEUE
      else lb_new = min(lb_new, LB(c))
```

Basically, the algorithm proceeds by *lb*- contours. Given a current lower bound *lb*, a depth-first branch-and-bound is initiated, discarding all nodes with a lower bound greater than *lb*. If no leaf node is found, the nodes that were discarded are used to determine a new lower bound $lb\_new > lb$. As no *lb*- contours are skipped, the first leaf node that is found during the search will contain an optimal solution to our problem. In that regard, IDA^* resembles best-first search, but avoids prohibitive memory use by branching in a truncated depth-first manner. The obvious drawback of the procedure is that large portions of the tree may be generated more than once. The branching scheme presented in Algorithm 2 and the dominance rules described in Section 3.3 all remain valid when using IDA^* as a search strategy. Note, however, that all cutset information must be discarded between two subsequent *lb_limited_search* passes.

A number of conditions must be satisfied for IDA^* to be an appropriate search strategy for the problem at hand. An important condition is that the number of *lb*- contours be limited. Indeed, in the worst case, the value *lb* is incremented one by one, and the size of the tree generated in subsequent *lb_limited_search* passes only increases with a handful of nodes. To avoid this phenomenon, we have developed a technique we call *look-ahead check*. The details of this technique are described in Deblaere et al. [35]. In short, *look-ahead check* will generate additional children of the rejected nodes in an attempt to increase the lower bound value of these nodes. If successful, this will result in a higher value for *lb_new* and consequently, in fewer *lb*- contours generated during a full execution of the IDA^* procedure.

3.4.3. Branch-and-bound with tabu search

As will be shown in Section 4, both regular branch-and-bound and IDA^* have trouble solving certain harder instances of the reactive scheduling problem. Regular branch-and-bound may take very long before finding a first good solution, while IDA^* sometimes grinds its way through a vast number of *lb*-contours, thereby losing its possible advantage over regular branch-and-bound. In an attempt to combine the good parts of both branching strategies, we have developed a tabu search procedure (Glover [37,38]) that searches for a heuristic solution for the reactive scheduling problem. The objective value of this heuristic solution can then be used as an upper bound for the regular branch-and-bound procedure. If the tabu search procedure finds a relatively good solution, we reduce the threat posed by large inferior regions, while eliminating the excessive revisiting of large portions of the tree entailed by the use of an IDA^* procedure.

The tabu search procedure we propose relies on fixing the modes of all activities that are unscheduled at the time of the disruption, and subsequently solving this reactive scheduling problem to optimality, using a "modeless" version of Algorithm 2. Indeed, if the modes are fixed, there is no need for branching on mode alternatives, so the solution space will be a lot smaller. Furthermore, the lower bound becomes stronger (since we know the modes of all unscheduled activities), and a number of conditions for cutset dominance (see Theorem 1) can be omitted or relaxed. All of this makes this problem considerably easier to

solve in comparison to its multi-mode variant. As for the choice of branching strategy for this easier reactive scheduling problem, empirical results w.r.t. computation times have shown that regular branch-and-bound has the edge on *IDA**. Since the tabu search procedure relies on a fixed set of reactive modes, the search amounts to finding a mode vector that minimizes the rescheduling costs as much as possible. Given a current mode assignment for the activities, the neighborhood of the tabu search procedure consists in finding an alternative mode assignment for one of the activities. The neighborhood will be explained in greater detail later in this section.

A formal description of the tabu search procedure is given in Algorithm 6. The procedure uses the function *modeless_b*&b$(\mathcal{M}, ub)$ to calculate the optimal objective value of the reactive scheduling problem given a mode alternative $\mathcal{M}$ and an upper bound *ub* on the rescheduling cost $\mathcal{C}$. If the procedure *modeless_b*&b$(\mathcal{M}, ub)$ finds an optimal solution with an objective value $\mathcal{C}^* < ub$, the value $\mathcal{C}^*$ is returned; else, the value ∞ is returned. Other variables used in the tabu search procedure are the current iteration number *iter*, the maximum number of iteration steps *max_iter* and a list $\mathcal{L}$ indicating for every activity i an iteration number $\mathcal{L}(i)$ that reflects the tabu status of that activity: activity i is tabu (i.e. its mode cannot be switched) as long as $\mathcal{L}(i) > iter$. The number of iteration steps an activity stays tabu is given by the parameter *tabu_time*. The tabu search procedure we propose uses a diversification scheme. During the procedure we keep track of the number of times the mode of every activity i has been switched and store this information in the variables frq_i. We use these values to calculate a frequency penalty $\mathcal{P}_f$. This penalty is a monotonously increasing function in frq_i with a maximum value equal to $\alpha \cdot global\_best$, a fraction of the objective value of the best known solution so far. In our

Algorithm 6. Tabu search.

set initial mode alternative $\mathcal{M}$:
- in the case of a nonrenewable resource disruption: $\mathcal{M} \leftarrow local\_search(m_1, m_2, \ldots, m_n)$
- else: $\mathcal{M} \leftarrow (m_1, m_2, \ldots, m_n)$

$global\_best \leftarrow modeless\_b\&b(\mathcal{M}, \infty)$
set $iter = 0$ and $\forall i \in N$: set $\mathcal{L}(i) = 0$, $frq_i = 0$
while $(iter \neq max\_iter)$
  $\mathcal{N} \leftarrow$ generate random neighborhood of $\frac{n}{4}$ activities such that $\forall i \in \mathcal{N} : \mathcal{L}(i) \leq iter$
  $neigh\_best \leftarrow \infty$
  for all $i \in \mathcal{N}$:
    for all $m \in M_i \backslash \{\mathcal{M}(i)\}$:
      $old\_mode \leftarrow \mathcal{M}(i)$
      set $\mathcal{M}(i) = m$
      given $\mathcal{M}$, calculate i^v according to Eq. (2)
      $\mathcal{P}_v \leftarrow global\_best \cdot \left(\frac{i^v}{i^v_{threshold}}\right)^2$
      if $(\max_j(frq_j) = 0)$
        $\mathcal{P}_f \leftarrow 0$
      else
        $\mathcal{P}_f \leftarrow \min\left(frq_i,\ \alpha \cdot global\_best \cdot \frac{frq_i - \min_j(frq_j)}{\max_j(frq_j) - \min_j(frq_j)}\right)$
      end if
      if $(i^v = 0)$
        $\mathcal{C} \leftarrow modeless\_b\&b(\mathcal{M}, neigh\_best)$
        if $(\mathcal{C} < global\_best)$
          set $m^* = m$, $i^* = i$, $global\_best = \mathcal{C}$, $neigh\_best = \mathcal{C}$
        else
          calculate $\mathcal{C}' = \mathcal{C} + \mathcal{P}_f$
          if $(\mathcal{C}' < neigh\_best)$
            set $m^* = m$, $i^* = i$, $neigh\_best = \mathcal{C}'$
          end if
        end if
      else
        $\mathcal{C} \leftarrow modeless\_b\&b(\mathcal{M}, neigh\_best - (\mathcal{P}_v + \mathcal{P}_f))$
        calculate $\mathcal{C}' = \mathcal{C} + \mathcal{P}_v + \mathcal{P}_f$
        if $(\mathcal{C}' < neigh\_best)$
          set $m^* = m$, $i^* = i$, $neigh\_best = \mathcal{C}'$
        end if
      end if
      set $\mathcal{M}(i) = old\_mode$
    end for
  end for
  set $iter = iter + 1$, set $\mathcal{M}(i^*) = m^*$, $\mathcal{L}(i^*) = iter + tabu\_time$, $frq_{i^*} = frq_{i^*} + 1$
end while

procedure, we chose $\alpha = 0.09$, as empirical experiments showed that this value produces good results. The idea of the parameter α is that the frequency penalty should not dominate the objective value in magnitude.

In order to solve the highly constrained project instances with respect to nonrenewable resources, we allow mode switches to result in a solution that is infeasible with respect to the nonrenewable resource constraints. By doing so, the tabu search procedure is able to escape from resource feasible "islands" in the solution space, allowing it to reach more promising regions with respect to the rescheduling costs. To nevertheless guide the search towards resource feasible solutions, the objective function value is penalized for nonrenewable resource infeasibility. This penalty $\mathcal{P}_v$ increases with increasing nonrenewable resource infeasibility ratio i^v, as defined by

$$i^v = \frac{\sum_{k=1}^{n^v} \max\left(0, \frac{(\sum_{i=1}^{n} r_{i\mathcal{M}(i)k}) - a_k^v}{a_k^v}\right)}{n^v} \quad (2)$$

Clearly, we have $i^v = 0$ if the mode alternative $\mathcal{M}$ is resource feasible and $i^v > 0$ otherwise. The higher the value of i^v, the higher the nonrenewable resource deficit. In Algorithm 6, $\mathcal{P}_v$ increases exponentially for increasing i^v values. A threshold value $i^v_{threshold}$ is used to control the magnitude of the penalty. When i^v reaches the threshold value, the penalty equals the objective value of the best known solution so far. In our procedure, we chose $i^v_{threshold} = 0.16$, so that when the average nonrenewable resource deficit amounts to 16% of the nonrenewable resource availability, the penalty equals *global_best*. This threshold value appeared to give good results in empirical experiments.

As a first step in the tabu search procedure, an initial mode alternative $\mathcal{M}$ is generated. In the case of a nonrenewable resource disruption, a local search is performed on the vector of baseline modes $(m_1, \ldots, m_n)$ to find a nonrenewable resource feasible mode assignment. This local search procedure iteratively switches the mode of one activity in the current mode assignment, under the objective of minimizing i^v. The search completes as soon as a mode vector has been found with $i^v = 0$. In the case of a duration disruption or a renewable resource disruption, we simply use the vector of baseline modes as the initial mode vector. During each step of the tabu search procedure, $n/4$ random nontabu activities are elected for a mode switch. Using the procedure *modeless_b*&b, the best mode switch is determined taking into account the objective value, the penalties and the value of the best solution found so far. If a mode alternative is nonrenewable resource infeasible, we can speed up the branch-and-bound by subtracting the penalty $\mathcal{P}_v + \mathcal{P}_f$ from the best objective value found so far in the current neighborhood. The tabu search procedure terminates when the maximum number of iterations has been reached. Given the obtained heuristic objective value *global_best*, we can now start the regular branch-and-bound procedure described in Section 3.4.1, using *global_best* as upper bound.

3.4.4. Binary search

A fourth optimal reactive scheduling procedure combines all ideas of the previous sections. Given an upper bound *ub* and a lower bound *lb* on the rescheduling cost of a given problem instance, we can initiate a binary search within the interval $[lb, ub]$ to find the optimal solution. The structure of the binary search procedure is shown in Algorithm 7. First, the *IDA** procedure is executed to generate a lower bound. This procedure is stopped as soon as the generation of a contour exceeds a predefined time limit t_c. This results in *IDA** being allowed to calculate an arbitrarily strong lower bound, as long as the tightening of this bound can be performed within a reasonable computation time. If this results in *IDA** finding an optimal solution before the time-based stopping criterion is reached, we are done. If not, we have obtained a lower bound *lb* and proceed to the next step of the binary search procedure.

Algorithm 7. Binary search.

```
lb ← time-limited IDA*
if IDA* found a solution:
  stop with C* = lb
else ub ← tabu search
  while ub ≠ lb
    m ← ⌊(lb+ub)/2⌋
    perform b&b_firstleaf(m)
    if b&b_firstleaf(m) found a solution with objective value C:
      set ub = C
    else
      set lb = m+1
    end if
  end while
  stop with C* = lb = ub
end if
```

If a lower bound *lb* has been calculated without *IDA** finding an optimal solution, we will initiate a tabu search procedure (Algorithm 6) to generate an upper bound *ub*. What follows then is an iterative procedure that continues as long as $lb \neq ub$. Given *lb* and *ub* we calculate the midpoint $m = \lfloor (lb+ub)/2 \rfloor$ and start the procedure *b*&b_*firstleaf* with *m* as an upper bound on the rescheduling cost. This auxiliary procedure is the regular depth-first branch-and-bound version of Algorithm 2 with the important distinction that it stops when a first leaf node (and hence a solution with $\mathcal{C} \leq m$) has been found. The execution of this procedure can have one of the two outcomes. Either a solution has been found, and we can update the upper bound as $ub = \mathcal{C}$, or no solution has been found, and we can set $lb = m+1$. The iterative part of the binary search procedure stops when $lb = ub$ and we have an optimal solution with $\mathcal{C} = lb = ub$.

The binary search procedure combines some of the advantages of the other procedures. For easy instances (i.e. disrupted schedules that are "easy" to repair), *IDA** will in most cases outperform the other algorithms. By running *IDA** for a limited amount of time, we keep this important advantage. Also, the tabu search procedure combined with branch-and-bound may run into problems if the solution found by the tabu search procedure is rather bad when compared to the optimum. For these instances, the binary search will also perform well thanks to the "trial" upper bounds *m* that limit the search of the branch-and-bound procedure. Binary search will then behave like *IDA**, but the number of contours that need to be explored will never exceed $\log_2(ub-lb)$, whereas *IDA** may need to generate $\mathcal{C}^*$ contours in the worst case scenario.

4. Computational results

4.1. Computational set-up

To evaluate our procedures, we have generated baseline schedules for the 20-activity and the 30-activity multi-mode instances of the PSPLIB project scheduling library (Kolisch and Sprecher [39]) using a dedicated tabu search procedure we developed for the basic MRCPSP. We generated baseline schedules with a certain optimality gap when compared to the best known

makespan reported in the literature. For the 554 20-activity instances, we have generated baseline schedules with a makespan no higher than 20%, 10% and 0% above the optimal makespan (so this last set is actually optimal). For the 552 30-activity instances, no optimal makespans are known for some of the instances. Therefore, we have generated baseline schedules with a makespan no higher than 40%, 30% and 20% above the best known makespan reported in the literature. It should be clear that a larger optimality gap allows for more rescheduling flexibility, and hence results in an easier instance. It makes sense to test our procedures on non-optimal schedules since these schedules will be better protected against possible disruptions, and lead to lower rescheduling costs. Finally, we remark that the baseline schedules are optimal solutions for the RCPSP instance that is obtained by fixing the modes of the MRCPSP instance to the modes of our generated solution. In other words, the instances have an optimal makespan given the mode assignment, but the mode assignment itself may not be optimal.

For a given instance set and baseline optimality gap, the procedures were tested by generating a single disruption scenario per instance as follows. The type of disruption generated is either a resource disruption or a duration disruption, both with an equal probability. In the case of a duration disruption, a random non-dummy activity i is selected from amongst the activities active during the first time period in the baseline schedule, and a duration increase Δ_i is generated as a uniformly distributed random variable drawn from the interval $[1, d_i]$, so that the maximal magnitude of the disruption equals the deterministic activity duration d_i. In the case of a resource disruption we first select whether the disruption affects a renewable or a nonrenewable resource type, both cases having an equal probability. For renewable resource disruptions, a random disruption is generated at time instant zero in such a way that the set of activities active during the first time period can no longer be executed as planned. The duration δ_t of the resource disruption is generated as a uniformly distributed random variable drawn from the interval $[1,5]$. In the case of a nonrenewable resource disruption, we select a random nonrenewable resource type $k \in \mathcal{K}^v$ and we calculate the total nonrenewable resource requirement req_k of the baseline schedule. We then assume that the availability a_k^v drops at time instant $t=0$ to a value of $0.95 * req_k$, such that the baseline schedule is no longer resource feasible. Note that we deliberately choose for disruptions occurring during the first time periods of the execution of the project, as this will potentially result in the need to revise the entire schedule and hence in a harder reactive scheduling problem.

The inflexibility weights w_j for each non-dummy activity $j \in \{2,3,\ldots,n-1\}$ are drawn from a discrete triangular distribution with $P(w_j = q) = (14-\frac{11}{12}q)\%$ for $q \in \{1,2,\ldots,15\}$. This distribution results in a higher occurrence probability for low weights and in an average weight $w_{avg} = 5.4$. The weight w_n of the dummy end activity denotes the marginal cost of violating the project due date and is fixed at $\lfloor 10 \times w_{avg} \rfloor = 54$. For an extensive evaluation of the impact of the activity weights, we refer to Van de Vonder et al. [40,41]. The mode switching costs c_{im}, $\forall i \in N, m \in M_i \backslash \{m_i\}$ are generated as uniformly distributed random variables drawn from the interval $[0,5]$. Note how we explicitly choose for mode switching costs that tend to be lower than the activity weights. In practice, meeting intermediate milestones and respecting the project deadline will always have very high importance. In that regard it will be advantageous to take mode-related measures (i.e. shifting resources) if this helps us to respect these milestones. Hence the motivation for the rather low mode switching costs in comparison to the activity weights.

All computational results have been obtained on a personal computer equipped with an Intel® Xeon® 2.33 GHz processor and the procedures have all been coded in C++.

4.2. Effect of the dominance rules

In this section, we study the effect of the various refinements and dominance rules for the proposed branching scheme on factors such as the computation time, the number of nodes visited and the number of contours generated throughout the branching procedure. More specifically, we study the incremental effect of the cutset dominance rule, the left shift dominance rule, the resource infeasibility rule and the look-ahead check refinement on the *IDA** procedure. In order to keep the experiment computationally tractable, we have used the rather easy-to-solve 20-activity instances with a makespan optimality gap of 20%. With respect to disruption scenarios, the computational set-up described in the previous section was used.

First, we ran the *IDA** procedure with no dominance rules at all (except for the data reduction rule) and without the look-ahead check refinement. Then, we added the different enhancements and dominance rules one at a time, in the order of decreasing impact on the computation time. The results shown in Table 2 are averages over the 554 instances. Note that the column with the heading "# nodes" shows the average number of nodes visited during the entire *IDA** procedure, including any nodes that may be expanded in the context of the look-ahead check procedure.

In the second row, the results are shown for the algorithm with only look-ahead check (lac). We see that, when compared to the procedure without look-ahead check, the average number of contours drops dramatically from 20 to a value of 11. This is immediately reflected in a lower average number of nodes visited, and consequently, a lower average computation time. The next added dominance rule is the cutset dominance (cs) rule. Again we observe a very large decrease in the average number of nodes visited as well as the average computation time. The third and fourth most influential rules appear to be the left-shift dominance (ls) and the resource infeasibility (ri) rule. Both rules yield a smaller yet still important speed-up.

4.3. Effect of the search strategy

In this section, we compare the performance of the four proposed algorithms (regular branch-and-bound, *IDA**, branch-and-bound with tabu search and binary search) with respect to computation times. More specifically, we compare the performance of these algorithms on 20- and 30-activity instances, with a varying baseline optimality gap. Concerning the generation of inflexibility weights, rescheduling costs and disruptions, we have used the same experimental set-up as in the previous section. For the 20-activity PSPLIB instances, we have generated baseline schedules with makespans no more than 0%, 10% and 20% above the optimal makespan reported in the literature. The average computation times of the reactive procedures over these instances are shown in Table 3.

When looking at the results, a first thing to note is that for every procedure, the computation time increases with decreasing baseline optimality gap. This is no surprise, because near optimal

Table 2
Effect of the dominance rules on the performance of the *IDA** procedure.

Included refinements and dominance rules	# nodes	CPU time (s)	# contours
–	3 681 783	17.38	20
lac	1 345 085	5.69	11
lac+cs	90 964	0.77	11
lac+cs+ls	56 204	0.60	11
lac+cs+ls+ri	38 388	0.48	11

Table 3
CPU times (s) for 20-activity instances.

Baseline optimality gap (%)	Regular B&B	*IDA**	B&B with tabu search	Binary search
20	0.54	0.48	0.52	0.26
10	0.81	1.30	0.75	0.86
0	1.48	3.30	1.57	2.09

Table 4
CPU times (s) for 30-activity instances.

Baseline optimality gap (%)	Regular B&B	*IDA**	B&B with tabu search	Binary search
40	83.0	32.7	13.0	15.1
30	561.3	54.3	24.9	23.7
20	–	138.5	311.4	57.3

baseline schedules will be harder to get back on track when a disruption occurs. In this computational experiment, the computation times of the different procedures are rather close to one another, and none of the procedures really stands out. The regular branch-and-bound procedure appears to be a solid choice, yielding the best results on the hardest instance set.

We have also tested our procedures on the 552 30-activity instances of PSPLIB. For these instances, we have generated baseline schedules with a makespan no more than 20%, 30% and 40% above the smallest obtained makespan reported in the literature. The average computation times over the 552 instances are shown in Table 4. On this instance set, the computation times of regular branch-and-bound increase very rapidly, and for the hard 20% instances, no solution was found within a reasonable time limit. *IDA** on the other hand performs surprisingly well and outperforms branch-and-bound with tabu search on the hardest instance set. But as is very clear in this experiment, binary search is the procedure of choice, yielding reasonable computation times even on the hardest instance sets.

As a final note, it is worth to mention that we have also tested the procedures on 30-activity instances with a 10% optimality gap, but for this set of instances, none of the procedures were able to finish within an acceptable time limit.

4.4. Effectiveness of the tabu search procedure

In this section, we study the performance of the tabu search procedure described in Section 3.4.3 as a stand-alone heuristic for the reactive scheduling problem. For the 20- and 30-activity instances we have used the same computational set-up as described above, the only difference being that all resource disruptions are now renewable resource disruptions. Doing so will allow us to investigate what happens to the optimal objective values if we do not allow mode switches to be performed. For all instances and makespan optimality gaps, we have compared the objective values obtained by the tabu search procedure with the optimal objective values. The results are shown in Tables 5(a) and (b). For every baseline optimality gap, we show the average rescheduling costs calculated by the exact procedure ($\mathcal{C}^*$), the average rescheduling costs obtained by the tabu search procedure ($\mathcal{C}_{TS}$), the average optimal rescheduling costs obtained when the reactive modes are fixed to the baseline modes ($\mathcal{C}^*_{fix}$) and the percentage of problem instances solved to optimality by the tabu search procedure.

Table 5
Performance of the tabu search procedure: (a) results for the 20-activity instances; (b) results for the 30-activity instances.

Baseline optimality gap (%)	$\mathcal{C}^*$	$\mathcal{C}_{TS}$	$\mathcal{C}^*_{fix}$	% optimal
(a)				
20	61	64	170	69
10	87	89	178	73
0	154	155	176	92
(b)				
40	45	48	208	64
30	52	56	204	67
20	67	70	206	70

The column $\mathcal{C}^*_{fix}$ is included to put the interpretation of the columns $\mathcal{C}^*$ and $\mathcal{C}_{TS}$ in a broader perspective. From the $\mathcal{C}^*_{fix}$- values we can see that a large reduction in rescheduling costs can be obtained when choosing for an optimal or near-optimal reactive policy instead of some myopic heuristic. Furthermore, we notice a relatively small gap between the average optimal rescheduling costs and the average rescheduling costs obtained by the tabu search procedure. Also, the average computation time required by the tabu search procedure is quite reasonable (17.6 s on the hardest instance set), and the procedure succeeds in solving a large part of the instances to optimality. The reader may notice a seemingly strange trend in the results, in that the number of instances solved to optimality by the tabu search procedure increases with decreasing makespan optimality gap. This is because the average number of mode switches that need to be made to reach an optimal solution decreases when the makespan optimality gap decreases. As the neighborhood of the tabu search procedure relies on mode switches, it will be easier to find the optimum if the optimal mode vector closely resembles the baseline mode vector.

5. Conclusions

In this paper, we have formulated a reactive scheduling problem for the multi-mode RCPSP. Given an MRCPSP baseline schedule and a resource disruption or an activity duration disruption that occurs during the execution of the baseline schedule, we want to obtain a reactive schedule that minimizes the rescheduling costs, defined as the sum of the mode switching costs and the costs incurred due to activity starting time deviations, under the additional constraint that no activity may start earlier than its baseline starting time. We have indicated that the literature on reactive scheduling procedures for the MRCPSP is very scarce.

We have proposed a branching scheme based on mode and delaying alternatives for optimally solving the reactive scheduling problem. A number of dominance rules for this branching scheme have been developed, namely a single-mode left-shift rule, a resource infeasibility rule and a cutset dominance rule. In order to circumvent certain problems posed by a regular branch-and-bound procedure, we have proposed an alternative branching strategy for this reactive scheduling problem, namely *IDA**. We have suggested a *look-ahead check* refinement to further speed up the *IDA** procedure. To take advantage of the insights obtained from testing regular branch-and-bound and *IDA** as branching strategies for the reactive scheduling problem, we have attempted to combine the good parts of both schemes into a single procedure. Therefore, we have proposed the use of a tabu search

procedure to obtain a heuristic solution for the reactive scheduling problem, and to use the objective value of this heuristic solution as an upper bound to be used in the regular branch-and-bound procedure. Doing so combines the advantage of *IDA** (not generating inferior nodes) with the efficiency of regular branch-and-bound (generating every node only once). In a further effort to combine the good parts of the previous three exact procedures, we have developed an algorithm that uses *IDA** , regular branch-and-bound and tabu search to obtain the optimal solution through a binary search in the interval $[lb, ub]$, with lb and ub a lower and an upper bound on the optimal solution value, respectively.

We have tested the effect of the dominance rules and the branching strategy on 20- and 30-activity benchmark instances. The results have shown the effectiveness of the look-ahead check refinement and the other dominance rules if used in the *IDA** procedure. When comparing the computation times of regular branch-and-bound, *IDA**, tabu search with branch-and-bound and binary search on instances of varying size and difficulty, we found that the binary search procedure is the most robust, yielding relatively short computation times on all instance sets used in the experiment. The algorithm effectively combines the advantages of regular branch-and-bound, *IDA** and tabu search in a single exact procedure. As for the performance of the tabu search procedure as a stand-alone reactive scheduling heuristic, we found that the average rescheduling costs obtained by the tabu search procedure only slightly deviate from the average optimal rescheduling costs. Also, the required computation time of the tabu search procedure remains within reasonable limits, even for the hardest instances considered in our experiments.

References

[1] Herroelen W, De Reyck B, Demeulemeester E. On the paper "Resource-constrained project scheduling: Notation, classification, models and methods" by Brucker et al. European Journal of Operational Research 2000;128(3):221–30.

[2] Speranza M, Vercellis C. Hierarchical models for multi-project planning and scheduling. European Journal of Operational Research 1993;64(2):312–25.

[3] Drexl A, Grünewald J. Non-preemptive multi-mode resource-constrained project scheduling. IIE Transactions 1993;25(5):74–81.

[4] Hartmann S, Sprecher A. A note on "Hierarchical models for multi-project planning and scheduling". European Journal of Operational Research 1996;94(2):377–83.

[5] Sprecher A, Hartmann S, Drexl A. An exact algorithm for project scheduling with multiple modes. OR Spektrum 1997;19:195–203.

[6] Hartmann S, Drexl A. Project scheduling with multiple modes: a comparison of exact algorithms. Networks 1998;32:283–97.

[7] Sprecher A, Drexl A. Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. European Journal of Operational Research 1998;107(2):431–50.

[8] De Reyck B, Herroelen W. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research 1999;119(2):538–56.

[9] Hartmann S. Project scheduling with multiple modes: a genetic algorithm. Annals of Operations Research 2001;102:111–35.

[10] Jozefowska J, Mika M, Rozycki R, Waligora G, Weglarz J. Simulated annealing for multi-mode resource-constrained project scheduling. Annals of Operations Research 2001;102:137–55.

[11] Alcaraz J, Maroto C, Ruiz R. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society 2003;54(6):614–26.

[12] Heilmann R. A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. European Journal of Operational Research 2003;127(2):348–65.

[13] Buddhakulsomsiri J, Kim D. Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. European Journal of Operational Research 2006;175(1):279–95.

[14] Lova A, Tormos P, Barber F. Multi-mode resource constrained project scheduling, priority rules and mode selection rules: scheduling schemes. Inteligencia artificial 2006;30:69–86.

[15] Lova A, Tormos P, Cervantes M, Barber F. An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. International Journal of Production Economics 2009;117(2):302–16.

[16] Zhang H, Tam C, Li H. Multimode project scheduling based on particle swarm optimization. Computer-Aided Civil and Infrastructure Engineering 2006;21:93–103.

[17] Zhu G, Bard J, Yu G. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. INFORMS Journal on Computing 2006;18(3):377–90.

[18] Sabzehparvar M, Seyed-Hosseini SM. A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags. Journal of Supercomputing 2008;44(3):257–73.

[19] Jarboui B, Damak N, Siarry P, Rebai A. A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Applied Mathematics and Computation 2008;195(1):299–308.

[20] Van Peteghem V, Vanhoucke M. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. European Journal of Operational Research 2010;201(2):409–18.

[21] Hildum DW. Flexibility in a knowledge-based system for solving dynamic resource-constrained scheduling problems. Dissertation, University of Massachusetts, Amherst, MA, USA; 1995.

[22] Davenport A, Beck J. A survey of techniques for scheduling with uncertainty. Unpublished manuscript, 2002.

[23] Vieira G, Herrmann J, Lin E. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. Journal of Scheduling 2003;6(1):35–58.

[24] Aytug H, Lawley M, McKay K, Mohan S, Uzsoy R. Executing production schedules in the face of uncertainties: a review and some future directions. European Journal of Operational Research 2005;161(1):86–110.

[25] Herroelen W, Leus R. Robust and reactive project scheduling: a review and classification of procedures. International Journal of Production Research 2004;42(8):1599–620.

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

[27] Van de Vonder S. Proactive-reactive procedures for robust project scheduling. Dissertation, Katholieke Universiteit Leuven, Leuven, Belgium; 2006.

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

[29] Van de Vonder S, Demeulemeester E, Herroelen W. Proactive heuristic procedures for robust project scheduling: an experimental analysis. European Journal of Operational Research 2008;189(3):723–33.

[30] Lambrechts O, Demeulemeester E, Herroelen W. A tabu search procedure for developing robust predictive project schedules. International Journal of Production Economics 2007;111(2):496–508.

[31] Lambrechts O, Demeulemeester E, Herroelen W. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. Journal of Scheduling 2008;11(2):121–36.

[32] Zhu G, Bard J, Yu G. Disruption management for resource-constrained project scheduling. Journal of the Operational Research Society 2005;56(4):365–81.

[33] Demeulemeester E, Herroelen W. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science 1992;38:1803–18.

[34] Demeulemeester E, De Reyck B, Herroelen W. The discrete time/resource trade-off problem in project networks: a branch-and-bound approach. IIE Transactions 2000;32:1059–69.

[35] Deblaere F, Demeulemeester E, Herroelen W. Exact and heuristic reactive planning procedures for multi-mode resource-constrained projects. Research Report KBI0818, Katholieke Universiteit Leuven, Leuven, Belgium; 2008.

[36] Korf R. Depth-first iterative-deepening: an optimal admissible tree search. Artificial Intelligence 1985;27(1):97–109.

[37] Glover F. Tabu search, Part I. INFORMS. Journal of Computing 1989;1:190–206.

[38] Glover F. Tabu search, Part II. INFORMS. Journal of Computing 1990;2:4–32.

[39] Kolisch R, Sprecher A. PSPLIB—a project scheduling library. European Journal of Operational Research 1997;96:205–16.

[40] Van de Vonder S, Demeulemeester E, Herroelen W, Leus R. The use of buffers in project management: the trade-off between stability and makespan. International Journal of Production Economics 2005;97:227–40.

[41] Van de Vonder S, Demeulemeester E, Herroelen W, Leus R. The trade-off between stability and makespan in resource-constrained project scheduling. International Journal of Production Research 2006;44(2):215–36.