



# Disruption management for resource-constrained project scheduling

G Zhu, JF Bard\* and G Yu

The University of Texas, Austin, TX, USA

In this paper, we study the problem of how to react when an ongoing project is disrupted. The focus is on the resource-constrained project scheduling problem with finish–start precedence constraints. We begin by proposing a classification scheme for the different types of disruptions and then define the constraints and objectives that comprise what we call the *recovery problem*. The goal is to get back on track as soon as possible at minimum cost, where cost is now a function of the deviation from the original schedule. The problem is formulated as an integer linear program and solved with a hybrid mixed-inter programming/constraint programming procedure that exploits a number of special features in the constraints. The new model is significantly different from the original one due to the fact that a different set of feasibility conditions and performance requirements must be considered during the recovery process. The complexity of several special cases is analysed. To test the hybrid procedure, 554 20-activity instances were solved and the results compared with those obtained with CPLEX. Computational experiments were also conducted to determine the effects of different factors related to the recovery process.

*Journal of the Operational Research Society* (2005) 56, 365–381. doi:10.1057/palgrave.jors.2601860

Published online 27 October 2004

**Keywords:** project management; disruption management; scheduling; integer programming; constraint propagation; resource

## Introduction

Projects are often performed under high levels of uncertainty related to such factors as resource availability, unproven technology, team competence, and the commitment of upper management. Sometimes, even the project goal is not well defined when the work begins. For most projects, though, a schedule specifying the implementation details must be developed before uncertainties are resolved. Without any historical data or past experience, expert opinion and rough estimates might be the only way to quantify activity costs and durations in the initial planning stages. What results is an initial schedule designed to optimize some objective within the limits of uncertainty.

As the project unfolds, differences between planned and actual costs, activity durations, and resource requirements begin to emerge. When the deviations become noticeable, we say that the project schedule is disrupted. For small deviations, the initial schedule may still be followed with little or no need for adjustment. In more serious cases, the initial schedule may no longer be optimal with respect to the original objective, and may not even be feasible. The primary purpose of this paper is to provide a structural framework for examining and resolving this type of problem. Our work falls in the growing field of *disruption management* (DM),<sup>1,2</sup> which

finds applications in the such diverse areas as transportation, ship building, and production planning, to name a few.

Although there are some similarities between the original scheduling problem and the one that must be solved after a disruption, the differences are significant. In the latter case, decisions need to be made in a more timely manner. There is usually a tradeoff between making good decisions and speeding up the recovery process to avoid further difficulties. In addition, there may be new constraints and new commitments associated with activities underway, especially with respect to future activities that were not anticipated when the original schedule was drawn up.

Another important distinction is the objective used to guide the analysis. Any schedule changes could have much wider implications for project stakeholders than simply the need to increase the budget. Due to the complex dynamics of projects, Eden *et al*<sup>3</sup> pointed out that it is very hard to estimate the cost of delay and disruption for most real world projects. An action taken to avoid delays sometimes can be a disruption itself. Therefore, the new objective must not only minimize the cost of handling the disruptions, but also minimize the deviation from the original schedule while getting back on track as soon as possible. When solving the *recovery problem*, a completely different schedule might emerge than the one obtained by resolving the problem with the original objective of, say, minimum makespan. In fact, modifying a schedule too much could turn a project with a promising return on investment into an outright failure after a full account is made of the deviation costs.

\*Correspondence: JF Bard, Graduate Program in Operations Research & Industrial Engineering, The University of Texas, Austin, TX 78712-1063, USA.

E-mail: jbard@mail.utexas.edu

At the time of a disruption, certain options may be available that were not feasible when the initial schedule was developed. The use of consultants or subcontractors, for example, may not have been considered initially because of company policies or pressure from upper management to use available staff. When disruptions occur, however, priorities may shift, opening the way for new options to be considered in the recovery process.

In the next section, we give a brief review of project scheduling issues and discuss how uncertainty relates to disruption management. In the subsequent section, the various types of disruptions and recovery options are highlighted and a classification scheme is developed. We also present an integer linear programming model for the recovery problem. This is followed by a discussion of some special cases and an analysis of their complexity. In the section thereafter, we present a hybrid mixed-integer programming/constraint propagation (MIP/CP) procedure to solve the general project schedule recovery problem. An example is given in the next section followed by a section on summary of our computational results. We first demonstrate the effectiveness of the procedure on 554 20-activity instances, and then investigate how the recovery problem is affected by several related factors, such as initial schedules, size of the disruption and the recovery window.

## Background

The goal of project scheduling is to develop a detailed plan specifying activity start and end times subject to precedence and resource constraints. The most common objective is to minimize makespan but other possibilities include the minimization of cost, the maximization of some quality measure, or a combination thereof. The problem is referred to in the literature as the resource-constrained project scheduling problem (RCPSP); for example, see Herreolen *et al.*<sup>4</sup>

Due to the increased interest over the past decade, a large amount of literature on RCPSP has appeared. Most existing algorithms for minimizing the makespan of an RCPSP fall into one of the following categories: priority rule-based sequencing heuristics, metaheuristics, and sequence enumeration based branch-and-bound. Work on variations or combinations of these methods can also be found. For a review of models, algorithms, classification schemes, and benchmark problems, we refer to some recent survey papers.<sup>4-6</sup>

### *Uncertainty in project management*

Disruptions occur when future events do not match expectations and so are closely associated with uncertainties. The types of uncertainties that arise in project management can be roughly divided into three categories:<sup>7</sup> (1) market-

related, such as demand, competition and the supply chain; (2) completion related, such as technical, construction and operational; (3) institutional, such as regulatory, cultural and extra-national. At the project level, once the strategic decisions have been made, the challenge is to achieve the technical goals within the time and cost constraints imposed by management.

Uncertainty is so prevalent, though, that few projects finish without time or cost overruns.<sup>8</sup> A delay in one activity may affect the schedule of all subsequent activities further causing disruptions in material supply, human resources, and possibly other projects. When milestones are critical and budgets are tight, the technical goal (for example, quality or service capacity of a facility being built) is often compromised.

Disruptions in projects not only bring challenges to the operational aspect of project management, but also may play a critical role when contracts are contested. Pickavance<sup>9</sup> analysed delays and disruptions in the construction industry and discusses how contracts should be prepared to hedge against disruptions. A disruption usually does not occur in isolation, but often as a result of other disturbances.<sup>3</sup> System dynamics models have long been used to investigate disruptions and to determine causality. The issue is important for both the client and the contractors because it determines who is responsible for budget or schedule overruns.<sup>10,11</sup> In contrast, our research emphasizes the operational aspect of disruption management, that is, how to make optimal operational decisions in the face of disruptions.

For the most part, uncertainties come in two general forms. The first is due to the stochastic nature of events. For example, it is rarely possible to specify the exact duration of an activity so some amount of estimation is necessary. Expert opinion, historical data, and industrial engineering methods are commonly used in this regard. The second type of uncertainty is associated with rare events of perhaps large consequence. Examples include natural disasters, the bankruptcy of a supplier, or the sudden loss of key personnel. These occurrences are difficult to predict and hard to prepare for.

The typical way project managers address uncertainty is with parametric analysis supplemented by risk assessment.<sup>12</sup> The general idea is to identify and evaluate the uncertainty associated with different aspects of the project and to take precautionary steps to reduce their impacts. Through Monte Carlo simulation, historical data are used to gain statistical insights and to assess the consequences of unplanned outcomes.

The second type of uncertainty, which we refer to as disruptions, are extremely difficult to deal with in project management. Traditionally, they are resolved by the experience and skill of the project manager, but often at a great cost. With few exceptions, such efforts are not likely to lead to optimal decisions. The work presented here is aimed at this shortcoming, and to the best of our knowledge, is the

first attempt to model and analytically solve disruption management problems that arise in project scheduling.

### Disruption management

Many operational environments require managers to deal with deviations from an original plan in real time. Disruption management is an emerging field in which operations research techniques are applied to help resolve uncertainties as they unfold. The problem that must be solved may be significantly different than the initial planning problem because it contains new decision variables, new constraints, and a new objective.

The airline industry has been one of the most active areas in which disruption management has been applied. The performance of an airline largely depends on how well it can follow its published schedule. Constant disruptions due to mechanical failures, personnel shortages, and severe weather can play havoc with the equipment, throwing flight schedules off for 24 h and days. In this regard, Yu *et al*<sup>13</sup> examine crew management issues, Argüello *et al*<sup>14</sup> look at aircraft routings, and Thengvall *et al*<sup>15</sup> consider multi-fleet interactions. For a more general discussion, see Clausen *et al*.<sup>1</sup>

### Disruption management for project scheduling

Figure 1 shows how the disruption management paradigm can be integrated into a project management system. As shown, disruptions act as inputs during the execution of the baseline schedule. One of the functions of the control system is to monitor progress and to flag deviations. Small

deviations might attenuate without the need for direct action because of inherent flexibilities and slack in the schedule. If the deviation between the actual and planned schedule exceeds a certain threshold, corrective action must be taken. In such circumstances, a schedule recovery problem has to be solved to get the project back on track. The overall process may be repeated many times.

### Representation of an initial schedule

We consider resource-constrained project scheduling with finish–start precedence constraints. A project consists of  $n$  activities denoted by the set  $\mathcal{A}$ , along with a number of milestones. Each milestone is associated with a set of activities. Its event time is defined as the earliest time that all such activities are completed. In addition to the real event time in a schedule, a milestone may also have a target time. In our model, time is measured in evenly spaced increments and each activity consumes resources at a constant rate during execution. Moreover, resource usage is limited to prespecified time slots, which are sets of discrete time periods not necessarily contiguous.

In the analysis, we start with an initial schedule in which each activity has a fixed start and end time and uses fixed amounts of various resources. The schedule is assumed to minimize a given objective function while satisfying the precedence and resource constraints included in the problem statement.

To formally describe a feasible schedule, we introduce the following notation.

### Indices and sets

$\mathcal{A}$	set of all activities
$i, j$	activity indices; $i, j \in \mathcal{A} = \{0, 1, \dots, n + 1\}$ , where 0 and $n + 1$ are dummy activities indicating the start and completion of the project, respectively
$\mathcal{K}$	set of resources
$k$	index for resources; $k \in \mathcal{K} = \{1, \dots, K\}$
$\mathcal{T}$	set of time periods
$\mathcal{T}_i$	set of time periods at which activity $i$ can finish; $\mathcal{T}_i \subseteq \mathcal{T}$
$t$	time index; $t \in \mathcal{T} = \{0, 1, \dots, T\}$ , where $T$ is an upper bound on the project completion time
$\Pi_k$	set of time slots on which resource $k$ is constrained
$\pi_k$	index for time slots on which resource $k$ is constrained; $\pi_k \in \Pi_k$ , $\pi_k \subseteq \mathcal{T}$
$\mathcal{P}$	precedence set
$(i, j)$	index for precedence relations; $(i, j) \in \mathcal{P}$ means that activity $j$ can only start after activity $i$ has finished; $(0, j) \in \mathcal{P}$ , $\forall j \neq 0$ ; $(i, n + 1) \in \mathcal{P}$ , $\forall i \neq n + 1$
$\Theta$	set of milestones
$\theta$	index for milestones; $\theta \in \Theta$
$B(\theta)$	set of activities whose completion defines milestone $\theta$ ; $B(\theta) \subseteq \mathcal{A}$ , $\theta \in \Theta$

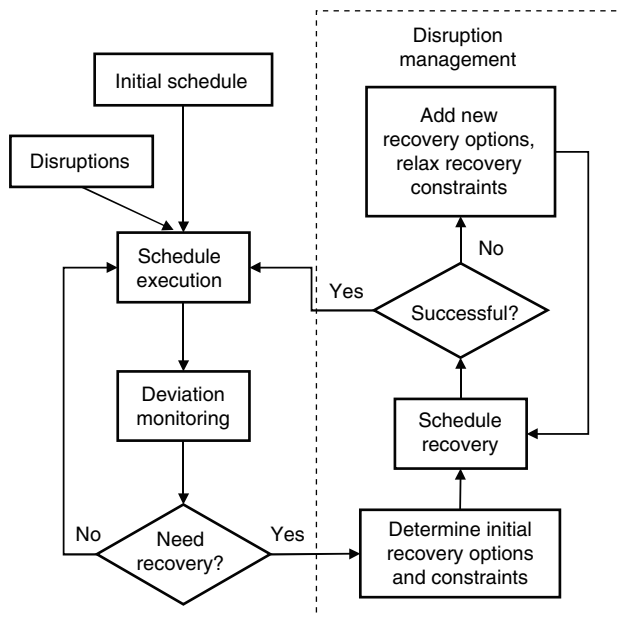


Figure 1 Disruption management for project scheduling.

### Schedule specifications

$\bar{f}_i$	finish time of activity $i$ ; $\bar{f}_i \in \mathcal{T}_i$
$\bar{t}_\theta$	target time of milestone $\theta$
$p_i$	processing time (duration) of activity $i$ ; $p_0 = 0$ , $p_{n+1} = 0$
$r_{ik}$	amount of resource $k$ required by activity $i$ per period
$R_{\pi_k}$	usage limit of resource $k$ on time slot $\pi_k$

### Types of disruptions

In the most general case, all parameters that define a project schedule may be disrupted. In developing a classification scheme, we divide the various types of disruptions into three categories: (1) the project network, (2) Activities, and (3) resources. Each will have a different impact on the project, and will be modelled and solved differently.

**Project network disruptions.** The structure of a project network is defined by the basic activities and the precedence relations among them. During execution, it is possible that activities may be added or removed, or precedence relations revised. For example, engineering change orders from the customer may require that new activities be introduced into the schedule, while design errors may require the structure of the network to be changed.

**Definition 1 (New activity disruption  $(\mathcal{A}_N, \mathcal{P}_N, \mathcal{A}_R)$ )** A set of new activities  $\mathcal{A}_N$  and corresponding precedence relations  $\mathcal{P}_N$  need to be added to the project network. The activities in set  $\mathcal{A}_R$  are no longer necessary for the project.

**Definition 2 (Precedence disruption  $(\mathcal{P}_A, \mathcal{P}_R)$ )** The project network for  $\mathcal{A}$  needs to satisfy the additional precedence relations in  $\mathcal{P}_A$ , but no longer needs to satisfy the relations in  $\mathcal{P}_R \subset \mathcal{P}$ .

**Activity disruptions.** An activity is said to be disrupted when either its duration or resource usage deviates from the planned values. The deviations can be either positive or negative. Typical examples for duration change include delays due to technical difficulties, resource shortages, or the need for rework. Also, unexpected external conditions or problems may force the use of more resources than planned. To formally describe such disruptions, we have the following definitions.

**Definition 3 (Activity duration disruption  $\delta_i$ )** Activity  $i \in \mathcal{A}$  takes  $\delta_i$  more time to complete than initially planned.

**Definition 4 (Activity resource disruption  $\gamma_{ik}$ )** Activity  $i \in \mathcal{A}$  uses  $\gamma_{ik}$  more of resource  $k$  during its execution than planned.

**Resource disruptions.** Resource shortage is probably the most common type of disruption in project management. It may be caused by a variety of factors such as machine breakdown, sudden loss of personnel, and resource overuse by other activities or projects. The primary impact is schedule infeasible.

**Definition 5 (Resource disruption  $\rho_{\pi_k}$ )** Availability of resource  $k$  in time slot  $\pi_k \in \Pi_k$  decreases by the amount of  $\rho_{\pi_k}$ .

**Milestone disruptions.** A milestone is a point in time so a disruption is not associated with a parameter as in the previous cases. Although a milestone disruption does not affect the feasibility of the ongoing schedule, it may be desirable to revise the schedule to better align the objective function with respect to the new milestone.

**Definition 6 (Milestone disruption  $\varepsilon_\theta \in \mathbf{R}$ )** The target time of milestone  $\theta$  changes from  $\bar{t}_\theta$  to  $\bar{t}_\theta + \varepsilon_\theta$ .

### Recovery options

Recovery options are the feasible decisions associated with the recovery problem and are specified in terms of the model's parameters. Interestingly, some disruptions and recovery options are associated with the same parameters, but they are essentially different. Disruptions are externally imposed on the project and take the form of deviations from the original plan, while recovery options serve as the decision variables.

Three options are included in our model. The first is *rescheduling*, which assigns finish times to activities that are different than the ones in the original schedule. The second is called *mode alternative*, and uses a different resource-duration mode for an activity. The third, *resource alternative*, increases resource availability. For example, resource alternative  $(\pi_k, R(\pi_k), g(\pi_k))$  means that the usage limit of resource  $k$  in time slot  $\pi_k$  increases by  $R(\pi_k)$ , and a cost of  $g(\pi_k)$  is incurred. Any penalty costs associated with the three options, are included in the recovery objective function.

For each activity, we define  $\mathcal{M}_i$  to be the set of all possible modes for activity  $i$ . This set includes the mode in the original schedule (referred to as mode  $m_0$ ) and the disrupted mode  $m_d$  if applicable. One special case for the mode alternative is *subcontracting*. A subcontracted activity requires only budgetary resources from the project, but must adhere to the precedence constraints. Another special case included in mode alternative is *activity cancellation*. If an activity is cancelled, it consumes no resources and time, but may remain in the project network. A penalty is incurred for each cancellation.

We now define the following decision variables:

$x_{imt}$	1 if activity $i$ is completed in mode $m$ at time $t$ ; 0 otherwise
$y_r$	1 if resource alternative $r$ is selected; 0 otherwise

*Recovery objective*

Once resources are allocated and commitments are made, any change in the schedule can affect both the performance of the project and the financial position of the stakeholders. Therefore, the recovery objective must take into account the initial plan, the deviations resulting from the disruption, and the cost of getting back on track. These considerations lead to the following general form of the new objective function that is to be minimized:

$$Q(\mathbf{x}, \mathbf{y}, \mathbf{t}) = \sum_{i,m,t} \alpha_{imt} x_{imt} \tag{1}$$

$$+ \sum_r g(r) y_r + \sum_{i,m} C_{im} \sum_t x_{imt} \tag{2}$$

$$+ \sum_i \left( \beta_i^1 \left[ \sum_{m,t} t x_{imt} - \bar{f}_i \right]^+ + \beta_i^2 \left[ \bar{f}_i - \sum_{m,t} t x_{imt} \right]^+ \right) \tag{3}$$

$$+ \sum_{\theta} (\lambda_i^1 [t_{\theta} + \varepsilon_{\theta} - \bar{t}_{\theta}]^+ + \lambda_i^2 [\bar{t}_{\theta} - t_{\theta} - \varepsilon_{\theta}]^+) \tag{4}$$

The symbols  $\alpha$ ,  $\beta$  and  $\lambda$  are given weights and  $[z]^+ = \max\{0, z\}$ . The event time of milestone  $\theta$  in the new schedule is defined as  $t_{\theta} = \max\{\sum_m \sum_i t x_{imt} : i \in \mathcal{B}(\theta)\}$  in (4).

The first term (1) measures the performance of the updated schedule and may be related to the original objective function. Recovery costs associated with different mode alternatives and increasing resource usage are reflected in the second term (2). The penalty incurred for both positive and negative deviations from the original schedule is represented by term (3), while term (4) captures the penalty for milestone deviations. The weights  $\alpha$ ,  $\beta$ , and  $\lambda$  allow the user to specify the relative importance of each component of each term. Of course, the individual values must be appropriately scaled.

Although the components in square brackets in terms (3)–(4) are piecewise linear convex functions of the activity and milestone finish times, no transformation is necessary to achieve a linear model. This follows from the fact that the finish time of each activity  $i$  is represented by  $\sum_{m,t} t x_{imt}$ , thereby allowing us to explicitly assign the deviation penalties as the objective function coefficients of  $x_{imt}$ . This is one advantage of using time-indexed variables.

*Recovery constraints*

In addition to penalizing schedule deviations in the objective function, we can also limit on the size of a deviation by introducing a constraint that bounds it. Although feasibility may not be the issue, if a deviation becomes too large, its consequence may not be acceptable. For example, we may want to resolve a delay of a critical activity within a few weeks because the competition already has a working system

and too long a delay might jeopardize our market position. The project crashing problem can be viewed as a special case of the recovery problem with a new constraint on the makespan.

To implement this idea, let  $T_0$  be the current time and  $[T_a, T_b]$  be the *recovery time window*, where  $0 \leq T_0 \leq T_a \leq T_b$ . All activities whose finish time is outside of the recovery window will have the same schedule as originally planned. If an activity has been completed, it is removed from the problem and the set of precedence constraints is updated accordingly. Let  $\mathcal{A}_F$  be the set of activities outside the recovery window that are unfinished. For  $i \in \mathcal{A}_F$  with planned finish time  $\bar{f}_i$  and mode alternative  $m_0$ , the recovery constraint can be written simply as

$$x_{im_0 \bar{f}_i} = 1 \quad \forall i \in \mathcal{A}_F \tag{5}$$

For activities whose planned finish time falls within the recovery window, there also may be recovery constraints. For example, if an activity has to be completed between  $t_1$  and  $t_2$ , we have

$$\sum_m \sum_{t_1 \leq t \leq t_2} x_{imt} = 1 \tag{6}$$

*ILP model for the recovery problem*

In formulating a mathematical model for the recovery problem, we assume that all parameters, activity sets, indices, and activity mode sets have been updated to reflect the disruption. Our model is as follows.

$$(RP) \min z = Q(\mathbf{x}, \mathbf{y}) \tag{7}$$

$$\text{s.t.} \quad \sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} x_{imt} = 1, \quad i \in \mathcal{A} \cup \mathcal{A}_N \tag{8}$$

$$\sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} t x_{imt} - \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}_i} (t - p_{jm}) x_{imt} \leq 0, \tag{9}$$

$$(i, j) \in \mathcal{P}_N \cup \mathcal{P}_A \cup (\mathcal{P} \setminus \mathcal{P}_R)$$

$$\sum_{t \in \pi_k} \sum_{i \in \mathcal{A} \cup \mathcal{A}_N} \sum_{m \in \mathcal{M}_i} \sum_{q=t}^{t+p_{im}-1} r_{imk} x_{imq} \leq R_{\pi_k} - \rho_{\pi_k} \tag{10}$$

$$+ y_{\pi_k} R(\pi_k), \quad \pi_k \in \Pi_k, k \in \mathcal{K}$$

$$x_{im_0 \bar{f}_i} = 1 \quad \forall i \in \mathcal{A}_F \tag{11}$$

$$\sum_{m \in \mathcal{M}_i} \sum_{t_1 \leq t \leq t_2} x_{imt} = 1 \quad \text{for some } i \in \mathcal{A} \setminus \mathcal{A}_F \tag{12}$$

$$\sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} t x_{imt} \leq t_{\theta} \quad \forall i \in \mathcal{B}(\theta), \theta \in \Theta \tag{13}$$

$$x_{imt} \in \{0, 1\} \quad \forall i \in \mathcal{A} \cup \mathcal{A}_N, m \in \mathcal{M}_i, t \in \mathcal{T}_i \tag{14}$$

$$y_{\pi_k} \in \{0, 1\} \quad \forall \pi_k \in \Pi_k, k \in \mathcal{K} \quad (15)$$

Equation (8) guarantees that each remaining activity has a unique finish time, while (9) and (10) respectively ensure that precedence and resource constraints are satisfied. Equation (11) guarantees that all activities outside the recovery window  $[T_a, T_b]$  are executed as originally planned. Constraints (12) and (13) represent the recovery restrictions imposed on activity and milestone completion times, respectively.

Model (7)–(15) can be considered a subproject with respect to the original project because some variables have been removed and some resources have been consumed. Nevertheless, due to the presence of recovery constraints and a composite objective function, the new model may be significantly different than the one solved during the planning process to obtain the initial schedule.

### Some special cases

Model (7)–(15) is general enough to account for virtually all possible types of resource and schedule disruptions. In this section, we study some special cases related to resource requirements.

#### Resource-unconstrained case

In the absence of resource constraints, only the precedence constraints drive the schedule. The critical path method can be used to solve the resultant problem when minimum makespan is the objective. In the case of a disruption, the cost associated with each affected activity may be a function of its completion time so CPM is no longer applicable, assuming that the objective is total cost minimization. When activity durations can be crashed, two subcategories are relevant.

*Single mode case.* In this case, an activity or precedence relation is disrupted, but no mode alternatives are available for the activities. Thus, the objective value depends only on the start and finish times of the activities. An integer programming model for this problem can be obtained by removing the resource-related constraints and variables from model (7)–(15) and considering the first term in the recovery objective function. Using a tighter form of precedence constraints, we have

$$(P0) \quad \min \sum_{i \in \mathcal{A}} \sum_{t \in \mathcal{T}_i} \alpha_{it} x_{it} \quad (16)$$

$$\text{s.t.} \quad \sum_{t \in \mathcal{T}_i} x_{it} = 1 \quad \forall i \in \mathcal{A} \quad (17)$$

$$\sum_{l=t-p_j+1}^T x_{il} + \sum_{l=0}^t x_{jl} \leq 1 \quad \forall (i, j) \in \mathcal{P}, t \in \mathcal{T} \quad (18)$$

$$x_{it} \in \{0, 1\} \quad \forall i \in \mathcal{A}, t \in \mathcal{T}_i \quad (19)$$

Note that the mode index  $m$  has been dropped from  $x_{imt}$  because each activity has only one mode. We call this problem P0.

**Theorem 1** *If each activity has only one mode, then the resource-unconstrained project scheduling problem with start-time dependent costs (P0) is polynomially solvable.*

Möhring *et al*<sup>16</sup> showed that the LP relaxation of (16)–(19) is integral, implying that P0 is polynomially solvable. More recently, Möhring *et al*<sup>17</sup> showed that P0 can be transformed into a minimum cut problem on a directed graph. For each binary variable  $x_{it}$  in model (16)–(19), a node is included in the directed graph. The precedence constraints are enforced by introducing arcs with infinite capacity.

*Multi-mode case.* Here, we still consider the resource-unconstrained case, but allow an activity to have different durations, each incurring a different penalty cost in the objective function. This corresponds to the situation where there are tradeoff opportunities between an activity duration and its cost. Duration–cost tradeoffs are necessary if the disruption makes the current schedule infeasible. The ILP model for this problem can be written as

$$(P1) \quad \min \sum_{i \in \mathcal{A}} \sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} w_{imt} x_{imt} \quad (20)$$

$$\text{s.t.} \quad \sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} x_{imt} = 1 \quad \forall i \in \mathcal{A} \quad (21)$$

$$\sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} (t - p_{jm}) x_{jmt} - \sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} t x_{imt} \geq 0 \quad \forall (i, j) \in \mathcal{P} \quad (22)$$

$$x_{imt} \in \{0, 1\} \quad \forall i \in \mathcal{A}, t \in \mathcal{T}_i \quad (23)$$

**Theorem 2** *If each activity has at least two different modes, the resource-unconstrained project scheduling problem with start-time dependent costs (P1) is NP-hard.*

The proof of Theorem 2 is given in the Appendix. If cost is only a function of the mode chosen for an activity, we have a minimum cost project crashing problem. This problem is very important in the context of disruption management because it resolves the disruption locally so that no activities outside the recovery window are affected. When the time–cost tradeoff functions are linear and continuous, it is well known that the crashing problem can be formulated as an LP. For discrete time–cost tradeoffs, we are faced with an ILP.

### Case with one nonrenewable resource

For a project, the budget, materials, and labor hours can be viewed as renewable resource for which only the total amount of usage is limited. Instead of adding a penalty cost to the objective function as in problem (P1) to account for a disruption, we will impose a constraint on the availability of the resource in question. The ILP model for this case can be obtained by adding the following constraint to (20)–(23)

$$\sum_{i \in A} \sum_{m \in \mathcal{M}_i} \sum_{t \in T_i} r_{im} x_{imt} \leq R \quad (24)$$

where  $R$  is the resource limit and  $r_{im}$  is the resource requirement for activity  $i$  when mode  $m$  is selected. We call the augmented model (P2).

**Theorem 3** *The multi-mode project scheduling problem (P2) with one non-renewable resource constraint is NP-hard.*

The proof is straightforward since (P1) is a special case of (P2).

### Case with only one renewable resource

In this case, each activity has only one mode so activity  $i \in A$  monopolizes a fixed amount of resource  $r_i$  during its execution. Because the amount of the resource that is available at time  $t \in \mathcal{T}$  is fixed at  $R$ , the following constraint must be satisfied at each point in time.

$$\sum_{i \in A} \sum_{q=t}^{t+p_i-1} r_i x_{iq} \leq R \quad \forall t \in \mathcal{T} \quad (25)$$

Combining constraint (25) with (16)–(19) gives an ILP model for the case with one non-renewable resource. We denote this problem as (P3).

**Theorem 4** *Single mode project scheduling problem with one non-renewable resource (P3) is NP-hard.*

**Proof** We identify several special cases of (P3) that are NP-hard. First, if every activity only uses one unit of the resource during its execution, we have a parallel machine scheduling problem, where each unit of the resource can be viewed as a machine. For a general linear objective function, the parallel machine scheduling problem is known to be NP-hard. An even simpler situation occurs when the resource limit is 2 and there are no precedence constraints. For makespan minimization, this is equivalent to a 2-partition problem, which is also NP-hard.<sup>18</sup> □

### Hybrid MIP/CP solution approach

MIP and more recently, constrained programming, are two popular ways of approaching general combinatorial optimization problems. The latter was originally developed to find good feasible solutions to constraint satisfaction problems.<sup>19</sup> It works by performing CP at each iteration of an enumerative process to reduce the domain of decision variables and to detect infeasibility caused by constraint conflicts.

The project scheduling recovery problem has features that are difficult to handle with either MIP or CP individually. A complicated objective function that includes various costs and penalties makes the problem difficult for CP. Precedence constraints, though, only involve pairs of activities so CP should do well with them. Realizing the efficiencies of either approach motivated us to develop a hybrid MIP/CP procedure.

#### Procedure

In designing our hybrid algorithm,<sup>20</sup> we use a branch-and-cut strategy to construct a search tree and to tighten the LP relaxation of (7)–(15) at each node. In addition, constraint propagation is performed to remove dominated variables. Related work on hybrid modelling and constraint classification can be found in Bockmayr and Kasper<sup>21</sup> and Jain and Grossmann.<sup>22</sup>

The main steps of the hybrid MIP/CP procedure are summarized below. Because violations of precedence constraints are mainly caused by branching on special ordered sets (SOS) (ie, Equations (8) and (12)), we perform CP immediately before new nodes are created in the tree rather than before the LP relaxation is solved.

#### Algorithm 1 HYBRID MIP/CP PROCEDURE

**Input:** Recovery problem instance and incumbent objective value  $\bar{z}$ , if available

**Output:** Optimal objective function value  $z^*$  or ‘infeasible’  
**begin**

Set  $z^* = \min\{\bar{z}, \infty\}$ .

Let  $\Omega$  be the set of enumeration nodes to be explored.

Perform constraint propagation at the root node of the search tree.

**if feasible then**

Add the root node to  $\Omega$ .

**while**  $\Omega \neq \emptyset$  **do**

**begin**

Select node  $\omega \in \Omega$  and solve the corresponding LP relaxation,  $LP(\omega)$

Resolve  $LP(\omega)$  if new cuts are added. Let  $z^\omega$  be the objective value.

**if**  $LP(\omega)$  is infeasible **or**  $z^\omega \geq z^*$ , **then**

$\Omega = \Omega \setminus \{\omega\}$

**else**

```

if optimal solution of  $LP(\omega)$  is integer then
  Set  $z^* = \min\{z^*, z^\omega\}$ ,  $\Omega = \Omega \setminus \{\omega\}$ 
else
  begin
  if SOS branching to be performed then
     $(S_1, S_2) = \text{call ConstraintPropagation}(\omega)$ 
    Create the first node by setting the upper bounds
    on variables in  $S_1$  to zero.
    Create the second node by setting the upper
    bounds on variables in  $S_2$  to zero.
  else
    Create two nodes by setting the branching
    binary variable to 0 and 1, respectively.
    Add the two created nodes to  $\Omega$ .
  end
end
Report optimal solution  $z^*$  or ‘infeasible.’
end

```

Algorithm 1 can be implemented with any MIP solver that permits callback functions at the nodes in the search tree. We used CPLEX version 7.5 in conjunction with ILOG’s Concert technology<sup>23</sup> for implementing the CP routine. Owing to the way CPLEX works, CP is actually performed after the branching decision is made but before setting up the MIP at the node to be explored. Therefore, each node created from SOS branching has already been processed by CP.

### Branch-and-cut

When applying branch-and-cut, it is common to start with a compact model and solve the LP relaxation. If integrality is not achieved, a ‘separation’ problem is solved to identify valid inequalities (cuts) that are violated by the LP solution. These cuts are added to the model and the process is repeated. If no cuts are found or improvement is minimal, the solution space is partitioned by selecting a variable for branching. Adding cuts to the model at each node in the search tree gives tighter LP bounds and hence may reduce the number of nodes that must be explored (see Bard *et al*<sup>24</sup> for an example).

For our problem, cuts can be developed from both the precedence and resource constraints. The latter (10) are in the form of knapsack constraints. Moreover, the variables that define the schedule for each activity  $i$  are divided into mutually exclusive special ordered sets. The cuts that can be derived from these restrictions are called GUB cover cuts<sup>25</sup> and are generated automatically by CPLEX when requested.

Precedence cuts can be obtained by explicitly enumerating the possible finish times for activities with precedence relations. For example, if activity  $i$  precedes activity  $j$ , which has a duration of  $p_j$ , then we know that if activity  $i$  finishes at  $t$  activity  $j$  must finish after  $t + p_j$ . Each of the enumerated

cases can be written in the form of a cut that can be used to tighten the LP bounds. For more detail, see Zhu *et al.*<sup>26</sup>

In standard branching, the bound on one variable at a time is fixed in the construction of the search tree. In SOS branching, the variables that make up each special ordered set are partitioned into two subsets, and all the variables in one subset are set to zero on each branch. This scheme can be exploited to great advantage during CP.

### Constraint propagation

The purpose of CP in our approach is to fix the value of some variables before solving the LP relaxations. In particular, we use CP to tighten the finish time windows at each node in the search tree by maintaining the consistency of precedence constraints and by fixing variables in a way that excludes inferior (ie, dominated) solutions. In the extreme case, if the finish time window for an activity is 0 at a certain node, we know that the corresponding problem is infeasible so the node can be fathomed.

*Consistency of precedence constraints.* In model (7)–(15), variables associated with activity  $i$  are defined on the subset  $\mathcal{T}_i = \{e_i, \dots, l_i\}$ , where the earliest finish time  $e_i$  and the latest finish time  $l_i$  are a function of the precedence relations and the makespan limit. At a node in the search tree, the time window in which activity  $i$  can finish  $[\hat{e}_i, \hat{l}_i]$  is usually smaller than  $[e_i, l_i]$ . Owing to precedence constraints, any change in the finish time window of one activity may affect the windows of both its predecessors and successors.

For  $(i, j) \in \mathcal{P}$ , the set of activities that are predecessors of  $j$  and successors of  $i$ ,  $\mathcal{A}_{ij} = \{k : (i, k) \in \mathcal{P}, (k, j) \in \mathcal{P}\}$ , can be viewed as a subproject. The minimum makespan of this subproject, call it  $d_{ij}$ , gives a lower bound on time (distance) between the finish of  $i$  and the start of  $j$ . We call the  $(n+1) \times (n+1)$  matrix  $D \equiv (d_{ij})$  the *distance matrix* for the project. We now define what we meant to be the consistency of precedence constraints at a node in the search tree.

**Definition 7** Let  $p_i^{\min}$  be the minimum possible duration of activity  $i$  and let  $T$  be an upper bound on the project makespan. A node in the search tree with finish time windows  $[\hat{e}_i, \hat{l}_i]; \forall i \in \mathcal{A}$ , satisfies the consistency of precedence constraints if (1)  $d_{ij} \geq d_{ik} + d_{kj} + p_k^{\min} \forall (i, k) \in \mathcal{P}$  and  $(k, j) \in \mathcal{P}$ , and (2)  $\hat{e}_i \geq d_{0i} + p_i^{\min}$ ,  $\hat{l}_i \leq T - d_{i, n+1} + 1 \forall i \in \mathcal{A}$ .

The transitivity of the distance matrix  $D$  is given by condition (1) and is necessary for the satisfaction of the precedence constraints. Condition (2) ensures that the finish time window of each activity is consistent with the precedence constraints. When any element of matrix  $D$  changes, condition (1) can be maintained by Algorithm 2. At

any node in the search tree, all variables associated with finish times outside of the window  $[\hat{e}_i, \hat{l}_i]$  should be set to zero. The idea of reducing search space by updating minimal temporal distance between activities has been employed in some other enumeration schemes (see Bartusch *et al*<sup>27</sup> for an example).

**Algorithm 2** UPDATE DISTANCE MATRIX

**Input:** Distance matrix  $D$ ; minimum durations of activities  $p_i^{\min}, i \in \mathcal{A}$

**Output:** Updated distance matrix  $D$

**begin**

**for**  $k=2$  **to**  $n+1$  **do**

**for**  $i=0$  **to**  $n+2-k$  **do**

**for**  $j=i+1$  **to**  $i+k-1$  **do**

**if**  $(i,j) \in \mathcal{P}$  **and**  $(j,i+k) \in \mathcal{P}$  **and**  $d_{i,i+k} < d_{ij} + d_{j,i+k} + p_k^{\min}$  **then**  
           $d_{i,i+k} = d_{ij} + d_{j,i+k} + p_k^{\min}$

**end**

In the recovery model, there are two situations that may cause a violation of the consistency of precedence constraints. The first concerns branching on activity finish times. In SOS branching, the finish time window of an activity is divided in half to create two branches. One branch has a new  $e_i$  and the other has a new  $l_i$ . These changes can be propagated to reduce the finish time windows of other activities by maintaining consistency of precedence constraints.

The other situation arises when multiple resource modes exist. Here, distances between precedence constrained activities can only be bounded from below by the largest possible resource limit. Branching on the variables associated with the resource alternatives, though, leads to a reduction of the resource limit along the corresponding branch. This may allow us to increase some elements of the matrix  $D$  and hence reduce the finish time windows of other activities through propagation.

The procedure of maintaining consistency of precedence constraints was implemented as a callback function in CPLEX. The initial distance matrix  $D$  was obtained by finding the critical path in the unconstrained network, but any lower bounding method for minimum makespan problems could have been used.

The main steps of the procedure are presented in Algorithm 3. CP is performed when SOS branching is applied. On each branch in the search tree, we identify the activity finish time windows and maintain the consistency of precedence constraints. As a result, additional variables are fixed to zero before the LP relaxations are solved. Using the incumbent objective function value  $\bar{z}$ , we also fix to zero those variables that cannot possibly produce a better solution. This is taken up next.

**Algorithm 3** CONSTRAINT PROPAGATION

**Input:** Node in search tree and corresponding LP solution  $\bar{x} = \{\bar{x}_{imt}\}$ , upper bounds  $u = \{u_{imt}\}$  on  $x = \{x_{imt}\}$  at the current node

**Output:** Two sets of variables  $(S_1, S_2)$  for creating two descendant nodes

**begin**

  Let  $i_b =$  activity for which the variable set is selected for branching

$p_i^{\min} = \{p_i^{\min}\}$ , where  $p_i^{\min} = \min\{p_{im} : m \in \mathcal{M}_i, \sum_t u_{imt} > 0\}$

$t_b = \sum_{m,t} \bar{x}_{i_b,mt}$

  Let  $D =$  distance matrix corresponding to resource limits at the current node

$\bar{D} = D; S_1 = \emptyset; S_2 = \emptyset$

**for**  $i=1$  **to**  $n+1$  **do**

$\bar{D}(0, i) = \min\{t : \sum_{m \in \mathcal{M}_i} u_{im, (t+p_{im})} \geq 1\}$

**for**  $i=1$  **to**  $n$  **do**

$\bar{D}(i, n+1) = T - \max\{t : \sum_{m \in \mathcal{M}_i} u_{imt} \geq 1\}$

$D_1 = \bar{D}; D_2 = \bar{D}$

$D_1(i_b, n+1) = T - [t_b]$

**call** UpdateDistanceMatrix( $D_1, p^{\min}$ )

**if** an incumbent exists **then**

**call** ReductionOfDominatedSpace

**for**  $i=1$  **to**  $i_b$  **do**

**if**  $D_1(i, n+1) > T - l_i$  **then**

      add  $\{x_{imt} : T - D_1(i, n+1) < t \leq l_i\}$  to  $S_1$

$D_2(0, i_b) = [t_b] - p_{i_b}^{\min}$

**call** UpdateDistanceMatrix( $D_2, p^{\min}$ )

**if** an incumbent exists **then**

**call** ReductionOfDominatedSpace

**for**  $i=i_b$  **to**  $n+1$  **do**

**if**  $D_2(0, i) > e_i$  **then**

      add  $\{x_{imt} : e_i \leq t < D_2(0, i) + p_{im}\}$  to  $S_2$

**return**  $(S_1, S_2)$

**end**

*Dominated solution space.* When the recovery objective function consists of the schedule deviation term (3) only, we are able to estimate a lower bound on the finish time window for each activity  $i \in \mathcal{A} \setminus \mathcal{A}_F$  at each node in the search tree. We can then compute a lower bound on the objective function, call it  $LB$  that can be used to fathom nodes when  $LB \geq \bar{z}$ .

Algorithm 4 provides the steps for removing dominated portions of solution space. For each activity, we enumerate possible finish times for the beginning and the end of finish time windows, and estimate the corresponding lower bounds. A finish time that leads to an inferior objective function value is excluded from the finish time window of the corresponding activity. The enumeration stops when we encounter an undominated lower bound. When other objective functions are used, the lower bounding method must be modified accordingly.

**Algorithm 4** REDUCTION OF DOMINATED SPACE

**Input:** Incumbent objective value  $\bar{z}$ , distance matrix  $D$ , finish time windows  $[e_i, l_i]$ ,  $i \in \mathcal{A}$ , recovery window  $[T_a, T_b]$   
**Output:** New distance matrix  $D$  and finish time windows  $[e_i, l_i]$ ,  $i \in \mathcal{A}$

**begin**  
 for activity  $i$  in the recovery window  $[T_a, T_b]$  **do**  
   **begin**  
     for  $t = e_i$  to  $l_i$  **do**  
       **begin**  
          $\hat{D} = D, \hat{d}_{0i} = t - p_i^{\min}, \hat{d}_{i,n+1} = T - t + 1$   
         **call** UpdateDistanceMatrix( $\hat{D}, p^{\min}$ )  
          $LB = \sum_{i \in \mathcal{A}} (\beta_i^1 [\hat{d}_{0i} + p_i^{\min} - \bar{f}_i]^+ + \beta_i^2 [\bar{f}_i - (T - \hat{d}_{i,n+1})]^+)$   
         **if**  $LB < \bar{z}$  **then**  
            $e_i = t, d_{0i} = t - p_i^{\min}$ ; **break** ‘for’ loop  
       **end**  
     for  $t = l_i$  to  $e_i$  **do**  
       **begin**  
          $\hat{D} = D, \hat{d}_{0i} = t - p_i^{\min}, \hat{d}_{i,n+1} = T - t + 1$   
         **call** UpdateDistanceMatrix( $\hat{D}, p^{\min}$ )  
          $LB = \sum_{i \in \mathcal{A}} (\beta_i^1 [\hat{d}_{0i} + p_i^{\min} - \bar{f}_i]^+ + \beta_i^2 [\bar{f}_i - (T - \hat{d}_{i,n+1})]^+)$   
         **if**  $LB < \bar{z}$  **then**  
            $l_i = t, d_{i,n+1} = T - t + 1$ ; **break** ‘for’ loop  
       **end**  
     **end**  
   **end**  
   **call** UpdateDistanceMatrix( $D, p^{\min}$ )  
   for activity  $i$  in the recovery window **do**  
     **if**  $e_i < d_{0i} + p_i^{\min}$ , **then**  $e_i = d_{0i} + p_i^{\min}$ ;  
     **if**  $l_i > T - d_{i,n+1} + 1$ , **then**  $l_i = T - d_{i,n+1} + 1$   
**end**

**Numerical example**

In this section, we illustrate our solution procedure with a 10-activity project constrained by one renewable resource. The project network is shown in Figure 2 along with activity durations and resource requirements. In each period, 10 units of the resource are available. Figure 3 depicts the original schedule which has a makespan of 32. The completion times indicated in the figure are target values and any deviation from them incurs a penalty that is a function of the recovery option selected.

Suppose that the schedule has been executed as planned up to time period 5 when activity 2 is disrupted. An assessment of the situation indicates that 3 periods of rework are needed, thereby extending the duration of activity 2 from 5 to 8. This disruption causes all successors to be delayed, which further causes resource infeasibility, as shown in Figure 4. If we simply delay activities to make the schedule resource feasible, most activities will deviate from their target finish times and the project will have a makespan of 35. To get back on track, we initiate a recovery procedure

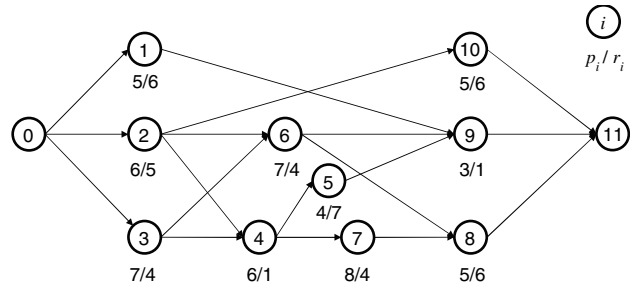


Figure 2 Project network.

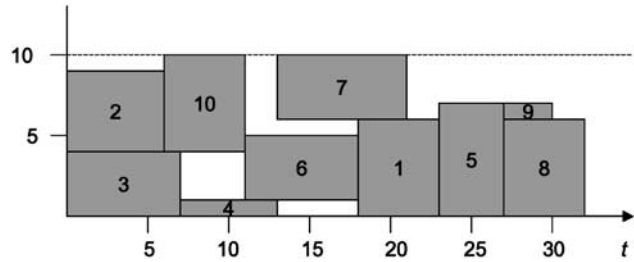


Figure 3 Original schedule.

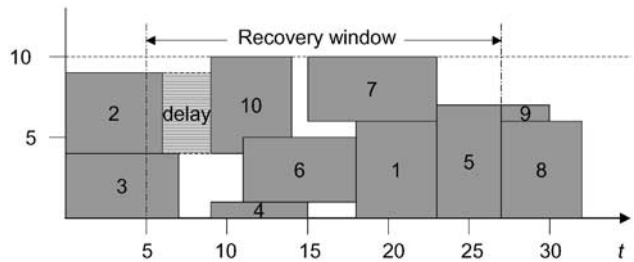


Figure 4 Disrupted schedule.

with the objective of minimizing the deviations from the target finish times.

Suppose that we want the original schedule to be resumed at time 28, which means that everything after time 27 should be exactly the same as in the original schedule. Accordingly, we define the recovery time window  $[T_a, T_b]$  to be  $[6, 27]$  (see Figure 4) and consider activities 1, 4, 5, 6, 7, 10 in the recovery process. To meet our requirement, we must have alternative crashing modes for at least some of these activities; otherwise, the problem may be infeasible. Table 1 lists the alternative modes for activities 1, 4, 5, 6, 7 in addition to the original mode, denoted as mode 1.

To illustrate CP, we first construct the distance matrix for those activities in the recovery window as well as the two dummies 0 and 11. All other activities are fixed. Considering the precedence relations and the minimum duration of each activity, the initial distance matrix is given in Table 2. The corresponding finish time windows are: activity 1:  $[8, 27]$ ; activity 4:  $[12, 24]$ ; activity 5:  $[15, 27]$ ; activity 6:  $[13, 27]$ ; activity 7:  $[18, 27]$ ; and activity 10:  $[14, 27]$ .

**Table 1** Mode alternatives for recovery options

Activity	Target finish time	Mode	Duration	Resource usage	Mode penalty
1	23	1	5	6	0
		2	4	7	5
		3	3	9	3
4	13	1	6	1	0
		2	4	3	6
		3	3	4	10
5	27	1	4	7	0
		2	3	8	9
		3	3	6	25
6	18	1	7	4	0
		2	5	6	6
		3	4	8	2
7	21	1	8	4	0
		2	6	5	4
		3	6	9	1
10	11	1	5	6	0

We now discuss three possible cases for which CP can be used to tighten these windows.

*Case 1 (Branching):*

Suppose the LP relaxation at the root node 0 gives fractional values for the variables that represent the finish time of activity 4. In particular, assume  $\sum_{m,t} tx_{4,m,t} = 14.4$ . If we partition on the finish time of activity 4 we have: branch 1—activity 4 finishes before or at time 14; and branch 2—activity 4 finishes at or after time 15. Based on the propagation of the precedence constraints for branch 2, activity 5 with duration 3 will now have a finish time window of [18, 27] and activity 7 will have a finish time window of [21, 27]. The variables defined for activity finish times outside of these windows can be set to zero. No other activities are affected on branch 2. Also, no additional variables can be set to 0 in branch 1.

*Case 2 (New incumbent solution):*

Suppose that each unit deviation from the target finish time for an activity incurs a cost of 5 and that we have a new incumbent solution with objective value  $\hat{z} = 48$ . All feasible solutions with  $z > 48$  will be dominated so we only need to consider deviations that are  $\leq \lfloor 48/5 \rfloor = 9$ . In other words, any variable  $x_{imt}$  farther than 9 periods from its target finish time will incur a cost greater than 48 and so can be set to 0. In the case of activity 1, which has a target finish time of 23, this means that we can reduce its original window [8, 27] to [14, 27]. Similarly, we have new finish time windows for activity 4: [12, 22]; activity 5: [18, 27]; and activity 10: [14, 21].

**Table 2** Initial distance matrix

$i \setminus j$	1	4	5	6	7	10	11
0	5	9	12	9	12	9	32
1		—	—	—	—	—	5
4			0	—	0	—	8
5				—	—	—	5
6					—	—	5
7						—	5
10							5

*Case 3 (Minimum duration change of an activity):*

Suppose that at some node in the search tree we observe that activity 5 has to be executed in mode 1, which has a duration of 4. Because the latest finish time of activity 5 is 27, this means that activity 4, which is an immediate predecessor of 5 and has duration 4, must finish no later than  $27 - 4 = 23$ . In general, an improved lower bound on the distance between a pair of activities [say,  $(i_1, i_2)$ ], may be propagated to any pair [say,  $(i_0, i_3)$ ] that has  $(i_1, i_2)$  between them. In this example, no further tightening is possible.

To show the effects of deviation penalties on the recovered schedule, we now solve the problem with different penalty costs. Assume that each unit deviation from target finish times, whether positive or negative, incurs a penalty of  $\beta$  [that is,  $\beta_i^1 = \beta_i^2 = \beta$  in Equation (3)]. The objective, which only includes the second term in Equations (2) plus (3), can be written as

$$Q = \sum_{i \in A, m \in M_i} c_{im} \sum_{e_i \leq t \leq l_i} x_{imt} + \sum_{i \in A} \beta \left( \left[ \sum_{m \in M_i, e_i \leq t \leq l_i} tx_{imt} - \bar{f}_i \right]^+ + \left[ \bar{f}_i - \sum_{m \in M_i, e_i \leq t \leq l_i} tx_{imt} \right]^+ \right)$$

where mode penalties  $c_{im}$  are listed in Table 1.

Table 3 shows the recovered schedules for  $\beta = 0$  and 3. In the first case, the optimal objective value  $z^* = 2$ , while the total deviation is 26. When we penalize the deviation by setting  $\beta = 3$ , the total deviation is reduced to 6 and the optimal objective value increases accordingly to 31.

**Table 3** Recovered schedules

Activity	Target finish time	$\beta = 0$		$\beta = 3$	
		Finish time	Mode	Finish time	Mode
1	23	18	1	23	3
4	13	15	1	13	2
5	27	27	1	27	1
6	18	13	3	20	1
7	21	23	1	20	2
10	11	23	1	14	1
Optimal objective, $z^*$			2		31
Total deviation			26		6

Without explicitly considering deviation penalties ( $\beta = 0$ ), the recovered schedule is significantly different than the original. This is usually unacceptable in practice, hence the need for the penalty term in Equation (3). To account for the relative importance of finish time deviations for each activity, different penalty coefficients can be assigned to the corresponding binary variables.

### Computational results

The hybrid MIP/CP procedure was tested on the 554 20-activity benchmark problems developed by Kolisch *et al.*<sup>28</sup> Each instance has 20 activities, two renewable and two non-renewable resources, and each activity has three alternative duration–resource modes. They were generated using different resource factors and resource strengths—measures of resource usage and availability.

We began by minimizing the makespan of each multi-mode RCPSP to get an optimal schedule. All activities not on the critical path were left-shifted to obtain an early start version called the *baseline*. The following disruption scenario and setup for the recovery problem are considered.

#### Disruption scenario and recovery setup

1. The duration of activity 3 was extended by 3 time periods (Definition 3).
2. The recovery window was defined to start at the time period immediately following the planned finish time of activity 3 and to extend through the end of the project.
3. The recovered makespan was not permitted to be more than 2 periods longer than the minimum makespan associated with the baseline schedule.
4. Except for those activities already underway, any mode that was initially available for an activity could be selected.
5. For each activity yet to be completed, the finish times in the baseline schedule were considered to be target finish times.
6. The recovery objective was to minimize the sum of deviation penalties of all activities (ie, only Equation (3) is considered). The penalty coefficients are:  $\beta_i^1 = \beta_i^2 = 1 \forall i \in \mathcal{A} \setminus \{n + 1\}$ ;  $\beta_{n+1}^1 = 1$ ,  $\beta_{n+1}^2 = 8$ .

Several sets of experiments were performed. The first was aimed at determining how efficient our solution approach is relative to the MIP solver in CPLEX. The rest was intended to see how different factors, such as the original schedules, delay time and recovery window, affect the recovery process.

To find feasible solutions to the recovery problem, a genetic algorithm (GA) was developed based on Hartman's procedure for the multi-mode RCPSP.<sup>29</sup> For each of the 554 instances, we ran this GA twice, each time with 100 generations and a population size of 100. If a feasible

solution was found, the corresponding objective value was used to initialize the upper bound parameter in CPLEX.

To allow us to compare like instances, eight groups were created based on the following two measures: (1) the renewable resource factor  $RF_r$ , and (2) the renewable resource strength  $RS_r$ . We refer to Kolisch *et al.*<sup>28</sup> for details on the calculation of these measures.

All codes were written in C++ and all computations were performed on a Linux workstation with a Xeon 1.7 GHz processor. CPU times are reported in seconds. The following notation is used in the presentation of the results.

$N$	total number of instances in a group
$N_I^{\text{PRE}}$	number of instances for which the infeasibility is detected by precedence constraints
$N_I^{\text{TOT}}$	number of infeasible instances
$t_{\text{GA}}$	CPU time for genetic algorithm
$N_{\text{VAR}}$	number of variables in the recovery problem
$N_{\text{CON}}$	number of constraints in the recovery problem
$LB_{\text{LP}}$	LP bound at the root node of search tree
$t_{\text{TOT}}$	total CPU time
$t_{\text{BEST}}$	CPU time of getting the optimal solution or proving infeasibility
$t_{\text{CP}}$	CPU time for CP
$N_{\text{ITER}}$	number of LP iterations
$N_{\text{GUB}}$	number of GUB cover cuts applied
$N_{\text{PRE}}$	number of precedence cuts applied

#### Comparison with CPLEX

In the first set of experiments, we solved all 554 recovery problems starting with the minimum makespan solution for the RCPSP as the baseline. Both the MIP/CP procedure and the CPLEX MIP solver were applied and both found the optimal recovery solution in all instances that were feasible.

Table 4(a) lists the characteristics of the recovery problems by group. Although an average problem consists of 366 variables and 125 constraints and is not very large, we see that the average gap between the LP bound at the root node,  $LB_{\text{LP}}$ , and the optimal objective value,  $z^*$ , is about 112%. For IPs in general, the larger this gap, the more difficult the problem is to solve. This is borne out by the data in Table 4(b), which compares the computational results of the two approaches. The instances in group 1 have the largest resource factors and the smallest resource strengths, and turn out to be the most difficult to solve. Their average gap is 437%. The hybrid MIP/CP procedure is especially effective on these problems when compared to CPLEX. Looking at the results for group 1, we see that the average computation time is reduced from 73.14 to 45.02, or approximately 38%.

The instances in the remaining groups are relatively easy to solve by either approach. This is evidenced by the small number of nodes in the search tree. Convergence occurs almost immediately after the best solution is found. In all cases, the computational effort associated with CP is

**Table 4** Comparative results for 20-activity instances

Group	$N$	$RF_r$	$RS_r$	$N_I^{PRE}$	$N_I^{TOT}$	$t_{GA}$	$N_{CON}$	$N_{VAR}$	$LB_{LP}$	$z^*$			
(a) Characteristics of recovery problems by group													
1	71	1	0.2–0.3	1	21	3.22	140	670	7.8	41.9			
2	69	0.5	0.2–0.3	7	27	3.17	126	369	9.5	25.4			
3	71	1	0.45–0.55	7	23	3.18	125	343	12.1	30.5			
4	70	0.5	0.45–0.55	14	24	3.16	122	296	10.6	18.0			
5	17	1	0.7–0.8	16	33	3.19	123	331	11.4	17.7			
6	72	0.5	0.7–0.8	16	26	3.25	121	291	10.5	12.6			
7	70	1	1	16	23	3.15	120	305	12.4	16.3			
8	60	0.5	1	19	29	3.17	115	192	10.4	10.9			
Total/average				105	206	3.19	125	366	10.6	22.5			
(b) Comparison of Computational results of MIP/CP procedure and CPLEX alone													
Group	MIP/CP procedure							CPLEX alone					
	$t_{TOT}$	$t_{BEST}$	$t_{CP}$	Nodes	$N_{ITER}$	$N_{GUB}$	$N_{PRE}$	$t_{TOT}$	$t_{BEST}$	Nodes	$N_{ITER}$	$N_{GUB}$	$N_{PRE}$
1	45.02	25.92	0.206	441	139 658	144	69	73.14	35.33	557	238 843	142	85
2	4.02	1.45	0.004	10	2457	38	26	4.09	1.46	14	4004	37	28
3	5.08	2.37	0.012	38	6533	53	31	5.65	2.81	49	10 334	55	35
4	3.65	1.48	0.001	6	743	17	17	3.50	1.43	8	1306	19	19
5	3.52	1.29	0.000	3	407	18	18	3.42	1.28	3	591	20	18
6	3.32	1.02	0.000	0	73	19	17	3.33	1.03	1	122	22	25
7	3.38	0.59	0.000	1	186	12	15	3.29	0.60	2	252	13	15
8	3.41	0.58	0.000	0	16	6	11	3.21	0.56	0	16	5	11
Average	10.25	5.15	0.034	77	23 211	55	33	14.66	6.67	98	39 533	57	39

negligible, requiring only a fraction of a second on average. Of the 554 instances, 206 proved to be infeasible for the given recovery requirements. This is shown in the bottom row of Table 4(a) under the column heading  $N_I^{TOT}$ .

When the recovery problem is infeasible, the disrupted schedule cannot be updated within the guidelines specified. There are two cases where this can occur. The first is associated with the precedence relations and can be characterized by what we call a *time infeasible* situation. Here, the length of the critical path calculated for the most optimistic scenario in which the minimum possible activity durations are used, exceeds the makespan limit. This type of violation can be detected before the ILP is set up. The number of instances that are time infeasible are listed in Table 4(a) under the column  $N_I^{PRE}$ .

Alternatively, if there is a feasible schedule without resource constraints but no feasible schedule with them, we have the *resource infeasible* case. As expected, the computational results indicate that time infeasibility is more prevalent when resources are ample (ie, instances in which the resource strength is large and the resource factor is small). The reason is that the makespan is mainly determined by precedence relations when resources are ample. When resources are scarce, the critical path does not play a primary role in determining the makespan.

### Impact of initial schedule

To see how the initial schedule affects the recovery problem, we ran a set of experiments using the same disruption scenario but starting with different initial schedules. Only the 71 instances in group 1 ( $RF_r = 1$ ,  $RS_r \in [0.2, 0.3]$ ) were evaluated. Also, the restriction on the makespan of the recovery schedule was removed.

In addition to optimal initial schedules, we also applied the disruption scenario to initial schedules obtained by running the GA for different times. Here, we used a population size of 50 and a generation size of 50. As shown in Table 5, different numbers of runs were performed to obtain different optimality gaps for the minimum makespan RCPSP. In Case 3, for example, three runs of the GA gave initial schedules with an average optimality gap of 3.23%.

For each case, Table 5 lists the average makespan, optimality gap and solution time for the initial schedules, as well as the average optimal objective value, makespan and CPU time for the corresponding recovery problems. As seen in the  $z^*$  column, the total penalty increases as the initial schedules approach the optimal schedules. The solution times for the recovery problems also increase.

These observations suggest that there might be a tradeoff between the quality of the initial schedule and the penalty incurred when the recovery problem is solved. To quantify

**Table 5** Recovery result for different initial schedules

Case	Initial schedules			Recovery results				
	Solution approach	Makespan	Gap	CPU time	$z^*$	Makespan	$t_{TOT}$	$z_c^*$
1	GA (50,50)	37.62	4.33%	0.58	36.87	38.77	33.26	46.23
2	GA 2*(50,50)	37.38	3.66%	1.16	36.14	38.58	33.59	44.06
3	GA 3*(50,50)	37.23	3.23%	1.74	37.75	38.49	34.42	44.74
4	GA 4*(50,50)	37.06	2.76%	2.90	39.73	38.44	36.70	45.71
5	GA 6*(50,50)	36.97	2.53%	3.48	39.52	38.34	32.72	44.99
6	GA 8*(50,50)	36.82	2.10%	4.64	41.44	38.31	36.51	45.98
7	GA 10*(50,50)	36.73	1.86%	5.81	42.66	38.28	41.94	46.70
8	Optimal	36.06	0	204.45	48.34	38.11	50.31	48.34

this relationship, we form a combined performance measure that sums the initial schedule and the recovery penalty. Let  $z_c^* = \mu(C - C^*) + z^*$ , where  $C$  is the makespan of the initial schedule,  $C^*$  is the optimal makespan for the initial schedule,  $\mu$  is a user-supplied weighting parameter, and  $z^*$  is the optimal objective for the recovery problem. If we assume that the earlier a delay is detected, the smaller the penalty, we can set  $\mu \leq \beta_{n+1}^2$ .

The last column of Table 5 reports the values of  $z_c^*$  for  $\mu = 6$ . We see that either spending too much effort (case 8) or too little effort (case 1) in solving the initial scheduling problem is not optimal with respect to this combined performance measure.

The results raise a number of questions about constructing an initial schedule. If there is no uncertainty, starting with the minimum makespan would, of course, be best as long as the computational effort to find it is reasonable. When disruptions occur, however, the recovery problem may be harder to solve when we start with an optimal schedule rather than with a 'good' schedule. Our computations also show that an optimal initial schedule is more likely to lead to an infeasible recovery problem than a heuristic initial schedule. This is primarily due to the slack that exists in a non-optimal schedule. By implication, then, it may be desirable to sacrifice a bit on makespan to gain some flexibility for dealing with disruptions.

#### Impact of length of delay

To determine how the length of the delay affects the recovery problem, we considered the same disruption scenario but without a restriction on the makespan. A series of experiments was conducted using the instances in Group 1 for the cases where activity 3 is delayed between 1 and 8 time periods. The results are reported in Table 6. As expected, as the delay increases, the recovery penalty and makespan of the new schedules increase almost linearly. In addition, the recovery problem becomes more difficult to solve, primarily because of the time-indexed model. The longer the delay, the greater the time horizon and the more variables in the recovery IP. As the last column in the table indicates,

**Table 6** Recovery result for different delays

Delay	$z^*$	Makespan	$t_{TOT}$
1	16.01	36.79	8.91
2	31.92	37.46	24.11
3	48.34	38.11	50.31
4	62.46	38.72	67.75
5	76.42	39.44	85.80
6	91.07	40.18	98.17
7	106.65	40.86	107.16
8	122.30	41.54	106.14

computation times increase rapidly at first and then taper off.

#### Impact of recovery window

In our disruption scenario, the recovery window extends from the time period immediately following the target finish time of activity 3 through the planning horizon. In reality, information about a disruption may be known well before it occurs. Therefore, the recovery process may be initiated any time after this information becomes available.

To determine the implications of foreknowledge, we solved all the instances in Group 1 for different recovery windows under the current disruption scenario but without restrictions on the makespan. In the analysis, each set of experiments is defined by an early recovery time  $t_{ET}$  which indicates the period when information about the disruption becomes known. For the original scenario,  $t_{ET} = 0$ ; that is, the recovery window extends from the period immediately following the target finish time of activity 3 through the planning horizon. In general,  $t_{ET} = k$  means the recovery window starts  $k$  periods prior to the disruption.

Table 7 summarizes the results for  $t_{ET} = 0, 1, \dots, 10$ . As we see, the earlier the recovery window starts, the smaller the penalty and the smaller the project makespan. However, due to the extended time horizon, the computational effort, as measured by  $t_{TOT}$ , increases proportionally.

**Table 7** Recovery result for different recovery windows

$t_{ET}$	$z^*$	Makespan	$t_{TOT}$
0	48.34	38.11	50.31
1	46.04	38.06	53.06
2	40.46	37.79	54.93
3	38.14	37.65	54.72
4	38.14	37.65	54.64
5	36.59	37.56	58.90
6	34.99	37.52	58.96

## Summary and conclusions

Disruptions due to such factors as resource shortages, technical difficulties, and loss of personnel are an unavoidable part of any project. In this paper, we studied the problem of how to react in a real-time environment when a project begins to deviate from its original plan. A major contribution of the work has been the development of a classification scheme for identifying recovery options and constraints under various types of disruptions.

The problem is modeled as a 0–1 integer linear program with the composite objective of minimizing undesirable deviations from the original schedule plus getting back on track as quickly as possible. Somewhat surprisingly, even very simple cases turn out to be quite difficult to solve, primarily because they have the same complexity as the original RCPSP. Based on the special characteristics of the precedence and resource constraints in the ILP, we proposed a hybrid MIP/CP procedure for finding reactive solutions. The procedure relies on cut generation and SOS branching to obtain tight bounds and a balanced search tree, and employs CP to reduce the size of the LPs that must be solved during the enumeration process.

Testing on 554 20-activity instances demonstrated the effectiveness of the procedure as well as its efficiency with respect to the MIP solver in CPLEX for the case of an activity disruption. Because the difficulty of a problem depends mostly on the size of the recover time window and not on the particular scenario, similar results could be expected for any type of disruption.

Computational experiments were also performed to study how various types of disruptions affect the recovery problem. Testing showed that spending either too much or too little effort in solving the original scheduling problem may produce inferior results. This calls into question the need and value of obtaining an optimal initial schedule when major disruptions are likely. Although the recovery problem is complicated by a variety of factors, our results imply that there are some general trends that can improve the project manager's ability to model and solve real disruption problems.

Our work is based on a general project scheduling model, which we believe is an accurate reflection of many

real-world situations. The purposed algorithmic procedure is specifically designed to exploit the precedence constraints, the most fundamental restrictions to which project activities must adhere. Although we have not tested the procedure on real instances, our computational experiments suggest that it will perform well on any of the cases discussed herein.

In summary, the paper provides a new way of viewing and resolving disruptions as a project unfolds. By defining appropriate recovery time windows and penalty functions, we show that optimal solutions to the recovery problem are well within reach of current technology. Possible future work includes the investigation of more general project settings and the development of more elaborate resource-based CP techniques.

## References

- 1 Clausen J, Hansen J, Larsen J and Larsen A (2001). Disruption management. *ORMS Today* **28**: 40–43.
- 2 Yu G and Qi X (2004). *Disruption Management: Framework, Models, Solutions and Applications*. World Scientific Publishers: Singapore.
- 3 Eden C, Williams T, Ackerman F and Howick S (2000). The role of feedback dynamics in disruption and delay on the nature of disruption and delay (D&D) in major projects. *J Opl Res Soc* **51**: 291–300.
- 4 Herroelen W, De Reyck B and Demeulemeester E (1998). Resource-constrained project scheduling: a survey of recent developments. *Comput Opns Res* **25**: 279–302.
- 5 Bottcher J, Drexel A, Kolisch R and Salewski F (1999). Project scheduling under partially renewable resource constraints. *Mngt Sci* **45**: 543–559.
- 6 Kolisch R and Padman R (2001). An integrated survey of deterministic project scheduling. *OMEGA* **29**: 249–272.
- 7 Miller R and Lessard D (2001). Understanding and managing risks in large engineering projects. *Int J Project Mngt* **19**: 437–443.
- 8 Pich MT, Loch CH and De Meyer A (2002). On uncertainty, ambiguity, and complexity in project management. *Mngt Sci* **48**: 1008–1023.
- 9 Pickavance K (2000). *Delay and Disruption in Construction Contracts*. LLP Professional Publishing: London.
- 10 Williams T and Eden C (1995). The effects of design changes and delays on project costs. *J Opl Res Soc* **46**: 809–818.
- 11 Williams T (2003). Assessing extension of time delays on major projects. *Int J Project Mngt* **21**: 19–26.
- 12 Chapman C (1997). Project risk analysis and management—PARM the generic process. *Int J Project Mngt* **15**: 273–281.
- 13 Yu G, Argüello M, Song G, McCowan SM and White A (2003). A new era for crew recovery at continental airlines. *Interfaces* **33**: 5–22.
- 14 Argüello M, Bard JF and Yu G (1997). A GRASP for aircraft routing in response to groundings and delays. *J Combin Optim* **5**: 211–228.
- 15 Thengvall BG, Bard JF and Yu G (2001). Multiple fleet aircraft schedule recovery following hub closures. *Transp Res Part A* **35**: 289–308.
- 16 Möhring RH, Schulz AS, Stork F and Uetz M (2001). On project scheduling with irregular starting time costs. *Opns Res Lett* **28**: 149–154.

17 Möhring RH, Schulz AS, Stork F and Uetz M (2003). Solving project scheduling problems by minimum cut computations. *Mngt Sci* **49**: 330–350.

18 Garey MR and Johnson DS (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman: New York.

19 Baptiste P, Le Pape C and Nuijten W (2001). *Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers: Boston, MA.

20 Rodosěk R, Wallace MG and Hajian MT (1999). A new approach to integrating mixed integer programming and constraint logic programming. *Ann Opns Res* **86**: 63–87.

21 Bockmayr A and Kasper T (1998). Branch and infer: a unifying framework for integer and finite domain constraint programming. *INFORMS J Comput* **10**: 287–300.

22 Jain V and Grossmann IE (2001). Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS J Comput* **13**: 258–276.

23 ILOG (2002). *ILOG CPLEX 7.5, Reference Manual*. ILOG, Inc.: Mountain View, CA.

24 Bard JF, Kontoravdis G and Yu G (2002). A branch-and-cut procedure for the vehicle routing problem with time windows. *Transp Sci* **36**: 250–269.

25 Gu Z, Nemhauser GL and Savelsbergh MW (1998). Lifted cover inequalities for 0–1 integer programs: computation. *INFORMS J Comput* **10**: 427–437.

26 Zhu G, Bard JF and Yu G (2003). *A branch-and-cut procedure for multi-mode resource constraint project scheduling problem* Working paper, Department of Management Science and Information Systems, The University of Texas, Austin.

27 Bartusch M, Möhring RH and Radermacher EJ (1988). Scheduling project networks with resource constraints and time windows. *Ann Opns Res* **16**: 201–240.

28 Kolisch R, Sprecher A and Drexl A (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Mngt Sci* **41**: 1693–1703.

29 Hartman S (1999). Project scheduling under limited resources. In: *Lecture Notes in Economics and Mathematical Systems*, Vol 478. Springer: Berlin.

**Appendix**

**Proof of Theorem 2** We will show that the 0–1 knapsack problem, which is known to be NP-hard,<sup>18</sup> is polynomially reducible to a multi-mode resource-unconstrained project scheduling problem with start-time dependent costs.

Given a set of items  $\mathcal{N} = \{1, 2, \dots, n\}$  such that each item  $i$  has a value of  $v_i$  and a weight of  $w_i$ , the 0–1 knapsack problem is to select a subset of items that maximizes the total profit while adhering to the restriction that the total weight of the selected items does not exceed  $b$ . The corresponding ILP model is

$$\min \left\{ \sum_{i \in \mathcal{N}} v_i x_i : \sum_{i \in \mathcal{N}} w_i x_i \leq b, x_i \in \{0, 1\} \forall i \in \mathcal{N} \right\} \quad (A.1)$$

It is assumed that  $\max\{w_i : i \in \mathcal{N}\} \leq b$ ,  $\sum_{i \in \mathcal{N}} w_i \geq b$ , and parameters are integral.

We begin by transforming (A.1) into a multiple choice knapsack problem. Let  $y_{j1} \equiv x_i$  and  $y_{j2} \equiv 1 - x_i \forall i \in \mathcal{N}$ . Then

it is obvious that the solution to (A.1) can be obtained by solving the following multiple choice knapsack problem

$$\min \sum_{i \in \mathcal{N}} \sum_{m=1}^2 v_{im} y_{im} \quad (A.2)$$

$$\text{s.t. } y_{i1} + y_{i2} = 1 \quad \forall i \in \mathcal{N} \quad (A.3)$$

$$\sum_{i \in \mathcal{N}} \sum_{m=1}^2 w_{im} y_{im} \leq b + \sum_{i \in \mathcal{N}} w_{i2} \quad (A.4)$$

$$y_{i1}, y_{i2} \in \{0, 1\} \quad \forall i \in \mathcal{N} \quad (A.5)$$

where  $v_{i1} = v_i$ ,  $v_{i2} = 0$ ,  $w_{i1} - w_{i2} = w_i \forall i \in \mathcal{N}$ .

We now construct a multi-mode unconstrained project scheduling problem from (A.2) to (A.5). Suppose a project has  $n$  activities, each corresponding to an item in the 0–1 knapsack problem. Let each activity have two modes and set the parameters in model (A.2)–(A.5) as follows.

$$\begin{aligned} p_{im} &= w_{im} \quad \forall i \in \mathcal{N}, m \in \{1, 2\} \\ p &= \{(1, 2), (2, 3), \dots, (n-1, n)\} \\ w_{imt} &= v_{im} \quad \forall i \in \mathcal{N}, i \neq n, m \in \{1, 2\}, t \in T \\ w_{imt} &= v_{im} \quad \forall i = n, t \leq b + \sum_{i \in \mathcal{N}} w_{i2}, m \in \{1, 2\} \\ w_{imt} &= M \quad \forall i = n, t \leq b + \sum_{i \in \mathcal{N}} w_{i2}, m \in \{1, 2\}, \\ &\text{where } M \gg \sum_i \sum_m \sum_t |w_{imt}| \end{aligned}$$

If there are no precedence relation conflicts, (P1) is always feasible. Suppose we have solved an instance of (P1) with the above parameter values and obtained an optimal solution  $x_{imt}^*$ . The following two cases show that solving an instance of project scheduling problem is equivalent to solving the corresponding multiple choice knapsack problem.

*Case 1:* If each element in the set  $\{x_{imt}^* : i = n, t \leq b + \sum_{i \in \mathcal{N}} w_{i2}, m \in \{1, 2\}\}$  has the value 0, then the project makespan is greater than  $b + \sum_{i \in \mathcal{N}} w_{i2}$ , which means constraint (A.4) of the multiple choice knapsack problem is infeasible. On the other hand, if constraint (A.4) is infeasible, the project makespan must be greater than  $b + \sum_{i \in \mathcal{N}} w_{i2}$ .

*Case 2:* If any of the elements in  $\{x_{imt}^* : i = n, t \leq b + \sum_{i \in \mathcal{N}} w_{i2}, m \in \{1, 2\}\}$  has the value 1, then we know that the project makespan is not greater than  $b + \sum_{i \in \mathcal{N}} w_{i2}$ . Therefore, constraint (A.4) is satisfied for the multiple choice knapsack problem and we can construct an optimal solution as follows:  $y_{im}^* = \sum_{t \in T_i} x_{imt}^*, \forall i \in \mathcal{N}, m \in \{1, 2\}$ . On the other hand, due to the large penalty coefficient for a makespan greater than  $b + \sum_{i \in \mathcal{N}} w_{i2}$ , if the multiple choice knapsack problem has a feasible solution, then the optimal

solution to the project scheduling problem must have a makespan no greater than  $b + \sum_{i \in \mathcal{N}} w_i z_i$ .

Therefore, the 0–1 knapsack problem is reducible to a multiple choice knapsack problem, which is further reducible to a special case of the multi-mode resource-unconstrained project scheduling problem with start-time-dependent

costs. Because both transformations can be performed in  $O(|N|)$  time, we conclude that the special case of (P1) examined above, and hence the general version of (P1), is NP-hard.

*Received March 2004;  
accepted June 2004 after one revision*