The Dissertation Committee for Guidong Zhu
certifies that this is the approved version of the following dissertation:

Disruption Management for Project Scheduling Problem

Committee:

Gang Yu, Supervisor

Jonathan Bard, Supervisor

Anant Balakrishnan

Leon Lasdon

David Morton

Disruption Management for Project Scheduling Problem

by

Guidong Zhu, B.Eng., D.Eng.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2005

Acknowledgments

I would like to express my gratitude to my advisors, Professor Gang Yu and Professor Jonathan Bard, for their support, patience, and encouragement throughout my Ph.D. studies. Their technical and editorial advice was essential to the completion of this dissertation. I have learned numerous lessons and insights from them on the work of academic research.

I also would like to thank my dissertation committee members, Professors Anant Balakrishnan, Leon Lasdon and David Morton, for providing suggestions in the early stage of this research.

Friendship of many fellow graduate students in the McCombs School of Business, The University of Texas at Austin is much appreciated. I want to thank them for suggestions to my research and an enjoyable environment for studying and working.

Last but not least, I would like to thank my wife Hui, for her understanding and love throughout all these years. She has always been there when I needed most.

Guidong Zhu

Disruption Management for Project Scheduling Problem

Publication No. _____

Guidong Zhu, Ph.D.
The University of Texas at Austin, 2005

Supervisors: Gang Yu
Jonathan Bard

This dissertation studies the resource-constrained project scheduling problem under disruptions. The focus is on how to formulate such problems and solve them efficiently. The work includes mainly three parts.

First, we present an exact branch-and-cut algorithm for the multi-mode resource-constrained project scheduling problem based on an integer linear programming (ILP) formulation (Chapter 4). We proposed several techniques that accelerate the solution process, such as variable reduction, special branching and bound-tightening schemes, and cuts. To find good feasible solutions in the early stages of the computations, a high level neighborhood search strategy known as local branching is included. As implemented, the full algorithm is exact in nature, but can be applied as an heuristic when solution time is limited.

Second, we study the problem of how to react when an ongoing project is disrupted (Chapter 5). 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.

Finally, we investigate the problem of setting target finish times for project activities with random durations (Chapter 6). Using two-stage integer linear stochastic programming, target times are determined in the first stage followed by the development of a detailed project schedule in the second stage. The objective is to balance the cost of project completion as a function of activity target times with the expected penalty cost incurred by deviating from the specified values. It is shown that the results may be significantly different when deviations are considered, compared to when activities are scheduled as early as possible in the traditional way. To find solutions, an exact algorithm is developed for the case without a budget constraint and is used as a part of a heuristic when crashing cost is limited.

For all the aspects, detailed examples and numerical results are provided to show the details of modeling process and the efficiency of proposed algorithms. We also find that project scheduling problem under disruptions is very different from a deterministic project scheduling problem, and leads to several useful managerial insights.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter provides a brief introduction to our research topic, and lays out an outline of this dissertation. We will also discuss the background and motivation for our research, and how our work fits into the literature.

1.1 Background and Motivation

A *project* is a temporary endeavor undertaken to create a unique product or service. It consists of a set of activities coordinated by precedence relations. Operated in an environment full of uncertainty, a project is expected to accomplish pre-determined goals subject to restrictions in resources and time. *Project management* is to apply knowledge, skills, tools and techniques to project activities to meet project requirements (PMI, 2000).

For most practical projects, a *project schedule*, which specifies when to start (and finish) project activities and how scarce resources are allocated, must be developed to plan and monitor the progress of the project. Given activities and resources, *project scheduling* is aimed to develop project schedules that meet performance requirements.

In practice, project scheduling problems are difficult for two reasons. First, the completion of most projects is subject to time and resource constraints, giving rise to the resource constrained project scheduling problem (RCPSP). Multiple resources and exponential number of ways for assigning resources to activities make RCPSP an extremely difficult combinatorial optimization problem. The problem

generalizes several known NP-hard problems, such as knapsack problems and machine scheduling problems. Second, real projects are subject to uncertainty. Some parameters may not be fully known at the time of decision-making. Scheduling based on expected parameters may not the best decision in this case. In addition, external or internal disruptions may cause the initial schedule no longer optimal or feasible. Some recovery actions may be necessary for this situation. However, problems remain in how the recovery should be done and measured.

It is well known that the project scheduling problem without resource constraints can be solved efficiently with the critical path method (CPM), but becomes NP-hard when only a single resource is present. When there are several different modes that may be selected for an activity we have what is called the multi-mode RCPSP (MRCPSP). Each mode corresponds to a different time-resource trade-off option for the activity under consideration. A feasible schedule specifies the implementation mode, as well as the start and finish times for each activity.

The RCPSP has gained widespread attention during the past few years due to its practical importance and computational challenge. While some of this effort has focused on refining the basic model, a majority of the research has been aimed at developing better solution methods. Most studies begin with a standard integer linear programming (ILP) formulation of the problem, but have not been successful applying branch and bound based on the linear programming (LP) relaxation. The limiting factor seems to be the large number of binary variables in problems of practical size. Motivated by this, the first part of this dissertation (Chapter 4) is devoted to develop an efficient branch and cut algorithm based on an ILP formulation of MRCPSP. This work also help us gain valuable insights in solving schedule recovery problems in case of disruptions.

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. Another purpose of this dissertation (Chapter 5) is to provide a structural framework for examining and resolving this type of problem.

The randomness of project parameters is a widely acknowledged aspect of project management. While some uncertainty will normally be resolved before any major disruptions occur, it is rarely possible to predict how random events will play out. Nevertheless, many decisions have to be made before all uncertainty is resolved. For example, when bidding on a project, a contractor has to submit a schedule and budget long before many critical pieces of information become known. Because the proposed schedule affects the likelihood of winning the bid, there may be a tendency to over promise. If awarded the contract, though, performance will be measured by how closely the schedule is followed, with penalties being assessed for activity delays and failure to meet major milestones.

This dissertation is also to present a new approach to dealing with project uncertainty that takes into account activity target times (Chapter 6). Unlike the deterministic case, all scheduling decisions for a project with uncertainty cannot be made with confidence in the planning stage. One obvious reason is that some

decisions require exact information. But even if were possible to fix the schedule at the beginning of the project, it is not likely that the corresponding decisions would be optimal with respect to the original objective after the uncertainty was realized. In light of this situation, a common practice is to make only those decisions that are absolutely necessary and then update them as circumstances unfold. Any remaining decisions are made as more definitive information becomes available. The issue here is to balance the benefits of ex-ante and ex-post decisions. By developing a stochastic optimization model and investigating several special cases, we have been able to gain a deeper understanding of the decision making process and offer a range of managerial insights.

1.2 Outline of the Dissertation

A comprehensive review of the literature related to our work is given in Chapter 2. We focus on the literature on three aspects: resource-constrained project scheduling problems, uncertainty in projects, and optimization techniques for disrupted operations – disruption management.

In Chapter 3, we define the notation, and describe settings of the project scheduling problem we are considering. An an integer linear programming model for the problem is presented. The complexity of several special cases are analyzed.

In Chapter 4, we present a branch-and-cut procedure for the MRCPSP based on the ILP model in Chapter 3. We highlight the solution methodology which includes the following components: preprocessing, variable reduction, cut identification, branching rules, bound tightening, and local branching. The key elements of the solution procedure are illustrated with a detailed example. Our computational experience solving the 1106 largest benchmark problems available is reported.

We study the problem of how to react when an ongoing project is disrupted in Chapter 5. 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, which is solved by a hybrid mixed-integer programming/constraint programming procedure. An example is given to illustrate the model and solution procedure in detail. This is followed by a 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.

In Chapter 6, we investigate the problem of setting target finish times for project activities with random durations. We present the two-stage stochastic linear programming model and discuss some of its properties. The second stage problem of the stochastic programming is a special case of the schedule recovery problem in Chapter 5, where the crashing budget limit is the only resource constraint. To solve the problem, we begin with solving the LP relaxation of the stochastic integer problem to obtain a lower bound on the optimal objective function value. For the case without a budget constraint, we show that the relaxed linear programming (LP) solution is indeed optimal. For the more general case, an LP-based heuristic is presented and shown to find good feasible solutions. We also show that in the absence of a budget constraint, the second stage subproblem can be transformed into a minimum cost network flow problem, and therefore, efficiently embedded in a stochastic programming algorithm. A detailed example is given along with our computational results for a standard set of benchmark projects.

We conclude our work with a summary in Chapter 7. Possible future work in related areas is also discussed.

Chapter 2

Literature Review

A real-world project is often complicated in two ways. First, it has a complex structure. A project usually has to complete a set of activities in a sequence order primarily determined by technology. Each of these activities may involve many different parties. Very often, activities compete for limited resources. The interaction between activities, resources and project stakeholders makes decision-making in projects a very large scale optimization problem. The other aspect of the complexity comes from uncertain nature of projects. A project is carried out in the uncertain world, where every aspect related to a project could change. With uncertainty, the decision-making in project management becomes an on-going process.

Project management is usually centered around a project schedule, which is used to plan, monitor and measure the project. In this chapter, we review the literature related to deriving a schedule for a complex project. We first look at the research on deterministic project scheduling problem in Section 2.1, which is followed by a review of work in dealing with uncertainty in projects. Another area closely related our work is disruption management, which studies how to react in disrupted operations. We give a review on this topic in Section 2.3.

2.1 Project Scheduling Problem

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); e.g., see Herroelen et al. (1998).

Two main types of constraints characterize an RCPSP. The first are resource constraints which impose a limit on resource usage in one or more time periods during the life of the project. Two special types of resources are often identified: renewable resources, which have the same usage limit in each time period (e.g., machines available per day); and non-renewable resources, which are only constrained in the aggregate. In reality, some resources are subject to both types of restrictions.

Temporal constraints are in the second category and specify the timing of activities that must be followed during their execution. The most common are the finish-start precedence constraints which state that one activity can only start after another has finished. More general temporal constraints (Brucker et al., 1999) include mode dependent restrictions on the time lags between activity start/finish times.

Due to the increased interest in the RCPSP over the past decade, a vast amount of literature has appeared since then so we will limit our discussion to algorithmic developments. For a review of models, algorithms, classification schemes, and benchmark problems, see the survey papers (Bottcher et al., 1999; Herroelen et al., 1998; Kolisch and Padman, 2001; Özdamar and Ulusoy, 1995). For an overview of issues related to maximum time lags (i.e., the maximum length of time permitted between the start of a pair of activities), see the book by Neumann et al. (2002).

Most existing algorithms for minimizing the makespan of an RCPSP fall into one of the following categories: priority rule-based sequencing heuristics, metaheuristics, and explicit branch and bound (by "explicit" we mean the partial enumeration of schedules to create a search tree rather than the use of fractional solutions obtained from the LP relaxation). Work on variations or combinations of these methods can also be found.

Heuristics for project scheduling do not work directly on the finish/start time of the activities. Instead, they try to generate an activity sequence that satisfies precedence constraints. The sequence is then transformed into a feasible schedule by a schedule generation scheme (SGS) (Kolisch, 1996). There are two different types of SGSs available. The so-called serial SGS selects one activity at a time from the ordered list and schedules it as early as possible. The second type, parallel SGS, is similar to a time incremented simulation. At each step or point in time, all eligible activities are assigned start times as long as feasibility is maintained. The clock is then incremented and all remaining activities are considered at the next point in time.

It is known that for single-mode minimum makespan problem, there exists an optimal solution that can be generated from an activity sequence by the serial SGS (Hartmann, 1999). However, finding the "correct" sequence remains NP-hard, as does the problem of finding a feasible sequence when non-renewable resources are present. In general, rule-based heuristics assign each activity a priority value that is used to construct an ordered list of activities. Common rules (Kolisch, 1996) include GRPW (greatest rank positional weight), LFT (latest finish time), LST (latest start time), MSLK (minimum slack), MST (most total successors), and WRUP (weighted resource utilization and precedence). The priority values may be determined before scheduling or calculated dynamically from one iteration to the next. Variations of the basic approach have been found to improve performance by 10 to 15%. Multi-pass schemes apply different rules on the same problem and simply choose the best solution. To increase the diversification, probabilistic sampling methods can be used to break ties among activities or to choose priority rules randomly. Scheduling activities backward as well as forward may also lead to better solutions.

Although these heuristics do not require much computational effort, their effectiveness is mixed. For 30-job, single mode instances with 4 renewable resources, priority rules and their variations have been shown to produce solutions with gaps

ranging from 4% to 30% (Klein, 2000). Recent developments in metaheuristic approaches have made it possible to obtain much better results for these problems in reasonable amounts of time. Genetic algorithm(GA) (Hartmann, 1999), tabu search (TS) (Nonobe and Ibaraki, 2002) and simulated annealing (SA) (Bouleimen and Lecocq, 2003) approaches have all been used to solve RCPSPs.

To move around a neighborhood efficiently, most metaheuristics work on a representation of a schedule rather than on an actual realization. Two of the most popular representations are activity lists and random key lists. The latter serve the same purpose as priority values. A mode list indicating the mode to be used for each activity is also necessary for multi-mode projects. Applying a particular SGS, the lists are transformed into actual time schedules so that objective values can be calculated. This can be done in $O(n^2 k_r)$ time (Hartmann, 1999), where n is the number of activities and k_r is the number of renewable resources. For any change in the schedule representation, the objective value must be re-calculated, often from scratch. From a practical point of view, this limits the number of schedules that can be searched. For 20-job multi-mode instances, Hartmann (1999) obtained a gap of 0.49% with a GA algorithm running for 5 seconds on a 133Hz PC. He was able to successively reduce the gap by half with a 5-fold increase in computation time for each reduction.

Most exact methods for solving the RCPSP incrementally enumerate schedules within a branch-and-bound framework (Brucker et al., 1998; Hartmann, 1999; Klein and Scholl, 2000; Mingozzi et al., 1998; De Reyck and Herroelen, 1999; Sprecher and Drexl, 1998). The first step is to decide how to construct the search tree. One way to do this is to start with a partial schedule and then try to add more activities at each iteration until a complete schedule is obtained. Working with the precedence graph is a natural approach since this assures that all precedence constraints will be satisfied automatically at each node. If more than one activity/mode is eligible to be scheduled at a certain level, then a node is created for each combination. Back-

tracking is performed under the usual circumstances. Another way of constructing a search tree is to start with a precedence feasible schedule, and resolve the resource conflicts one step at a time. Instead of a feasible partial schedule, each node in the tree is usually an infeasible complete schedule with some delay or mode alternatives imposed. A detailed description of these approaches can be found in (Hartmann, 1999) along with strategies for preprocessing the data and the benefits associated with the use of dominance rules.

The efficiency of branch-and-bound methods heavily depends on finding good feasible solutions and fathoming infeasible branches as early as possible. There has been a fair amount of work devoted to obtaining good lower bounds for the RCPSP. The CPM bound is the easiest one to compute, but has not proven to be very effective. Improvements are possible when resource usage is taken into account. Alternatively, solving different relaxations of the original ILP is the best way to produce good lower bounds. Lagrangian relaxation is one approach, or simply ignoring some of the constraints is another. Brucker et al. (1998) developed a branch-and-bound algorithm using lower bounds based on the LP relaxation and constraint propagation (Brucker and Knust, 2000). Mingozzi et al. (1998) formulated the RCPSP as a 0-1 integer program with an exponential number of variables. In their model, each variable corresponds to a subset of activities that could be executed simultaneously. Several relaxations of IP were solved to obtain a range of lower bounds. For a summary of existing lower bounds and techniques for tightening them, see Klein and Scholl (1999).

Some exact branch-and-bound methods can serve as heuristics when run time limits are imposed. So-called truncated branch and bound (Hartmann, 1999; Sprecher and Drexl, 1998) restricts the size of the search tree by using heuristics to select only a subset of nodes for branching, and falls in the general category of beam search methods.

The idea of reducing search space by updating minimal temporal distance

between activities has been employed by Bartusch et al. (1988). Dorndorf et al. (2000) used a time-oriented branching scheme in a branch-and-bound procedure for RCPSP with generalized temporal constraints, and applied constraint propagation techniques to reduce the search space.

Although the ILP model for the RCPSP is well known, only Icmeli and Rom (1996) report efforts to solve it using a general purpose MIP (mixed-integer programming) code. To overcome computational difficulties, they replaced the basic time unit with project "milestones" and required that the resource constraints hold only at these points. For 30-activity single mode problems, 6 evenly spaced milestones were chosen for the grid. Nevertheless, they were still not able to solve many of their 110 test problems within a 15-minute time limit on an IBM 3090.

2.2 Uncertainty in Projects

The types of uncertainty that arise in project management can be roughly divided into three categories (Miller and Lessard, 2001): (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 (Pich et al., 2002). 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 (e.g., quality or service capacity of a facility being built) is often compromised.

For the most part, uncertainties come 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 (e.g., Chapman, 1997). 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. Historical data are used to gain statistical insights and to assess the consequences of unplanned outcomes.

To increase the likelihood of successful completion of projects, project risk management has been an very important part of project management. The goal is to identify and evaluate uncertainty associated with different aspects of the project, and make proper actions to alleviate the affects of uncertainty. Uncertainty identification is to estimate possible events and their probabilities by looking at historical statistics and current situation. Uncertainty evaluation commonly involves work of obtaining the probabilistic information of a system given the uncertainty information of its elements. However, due to the complexity from the uncertainty and project structure, the tools developed in project risk management have been unable to handle many real world situations, such as resource constraints, interdependence and ambiguities (lack of knowledge). Even though, they have provided very useful insights in managing complex projects.

Stochastic project networks, which represent resource-unconstrained projects composed of activities with stochastic durations, have been studied for a long time. Two problems have caught most of the research attention. The first problem is to

estimate the distribution of project completion time given the probabilistic information of activities. Independence of activity durations is often assumed. Ludwig et al. (2001) computationally evaluated several procedures to bound or approximate the distribution function of the makespan of stochastic project networks.

A second problem in stochastic project networks is how to measure the relative importance of activities. Similar to critical path method (CPM), program evaluation and review technique (PERT) identifies a critical path in a stochastic network using expected durations. An activity is critical if it is on a critical path in CPM. For each activity in a stochastic project network, an index called criticality is defined as the probability of the activity is on a critical path. Elmaghraby (2000) reviewed several indices that are aimed to characterize the sensitivity of the mean and variance of project completion time to changes in the mean and variance of individual activities. None of the them can be consistently meaningful for all project cases. Most of these indices are hard to evaluate analytically. There is still no effective approach to evaluate the interaction between the parameter changes of two or more activities. The impacts of uncertainty on a project also depends on the topology of the project network. Gutierrez and Paul (2000) found that different subcontracting mechanisms should be applied for homogeneous projects with parallel or serial subprojects.

If there are resource constraints in addition to the stochastic durations, the project scheduling problem become much harder to solve. First, with resource constraints, the corresponding deterministic scheduling problem becomes NP-hard. Second, some concepts for unconstrained cases are no longer meaningful. For example, it is hard to say if an activity is critical because with different resource allocation scheme, it may or may not be on a critical path (Bowers, 2000). If there is no resource constraints, it is assumed that if a delay occurs in the early stage of the project, the followed activities are delayed to make a feasible schedule. However, with resource constraints, simply delaying may not always give us feasible schedules.

Also, there may be much better solutions than simple delaying.

Though there has been numerous papers on deterministic RCPSP, only very a few have addressed the resource constrained stochastic project scheduling. A heuristic procedure was proposed by Golenko-Ginzburg and Gonik (1997) to minimize the expected makespan for resource constrained project with stochastic activity durations. Valls et al. (1998) proposed a scatter search approach for project scheduling with random activity interruptions. The objective is a linear combination of the mean and variance of the project completion time. Möhring and Stork (2000) studied linear pre-selective scheduling policies for resource constrained stochastic project scheduling problem.

An area that is closely related to resource constrained stochastic project scheduling is stochastic machine scheduling, which can be view as a special case of project scheduling problem. Birge and Dempster (1996) presented a framework that applies stochastic programming approaches to stochastic scheduling. Optimization procedures are incorporated into the problems as part of a decision hierarchy. While it is very difficult to obtained optimal solutions for many practical scheduling problems, scheduling policies (heuristics) are used to find good schedules quickly. Möhring et al. (1999) studied parallel machine scheduling problem with stochastic job processing times. They provided LP-based priority policies and derived constant performance guarantees.

Herroelen and Leus (2004) gave a survey on project scheduling under uncertainty. They discussed several approaches for obtaining schedules with uncertain information: reactive scheduling, stochastic project scheduling, fuzzy scheduling, robust scheduling, and sensitivity analysis. Some future work directions are also discussed. Al-Fawzana and Haouari (2005) considered total slacks of a project schedule as a measure of robustness, and proposed a tabu search algorithm to minimize the makespan as well as total slacks.

2.3 Disruption Management

Many operational environments require managers to deal with deviations from an original plan in real time. Such deviations, which we refer to as *disruptions*, occur when events do not match expectations. Disruption management (Yu and Qi, 2004) is an emerging field in which operations research techniques are applied to help resolve uncertainties as they unfold. In disruption management, we treat disruptions as given reality. The problem is to how to model and make decisions for the disrupted operations. 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.

Yu and Qi (2004) presented a general framework of disruption management, and its applications in various fields, such as airline industry, scheduling, production and supply chain management. Disruption management emphasizes on the real time revision of an operational plan when disruptions occur. When generating a new plan in case of disruptions, the deviation cost associated with the transition from the original plan to the new plan must be considered, especially when the initial plan is publicly published. Different from initial planning, disruption management must obtain solutions quickly with real-time optimization techniques. Sometime, partial solutions may be desirable for a very complicated situation.

As disruption management is an emergency handling approach, it is desirable to return to the original plan some time after the disruption. In many complex operations, it is very hard to estimate the cost of plan changes. For this reason, disruption management also emphasizes on local resolutions for disruptions: restricting plan changes in local areas of the system and in a certain time window.

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 hours and even days. In this regard, Yu et al. (2003) examine crew management issues, Argüello et al. (1997) look at aircraft routings, and Thengvall et al. (2001) consider multi-fleet interactions. For a general discussion of disruption management, see Clausen et al. (2001).

The consideration of deviation penalties has been most studied in machining scheduling problems, where a job is penalized for being late or early (Baker and Scudder, 1990). More recently, Sourd and Kedad-Sidhoum (2003) presented a branch and bound scheme for single machine scheduling with earliness and tardiness, in which lower bounds based on preemptive schedules are used. Vanhoucke et al. (2001) studied project scheduling problem with given due dates to minimize the total weighted earliness and tardiness. Chrétienne and Sourd (2003) considered general convex cost project scheduling problem without resource constraints.

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.

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 (2000) analyzed 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 (Eden et al., 2000). 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 (Williams and Eden, 2000; Williams, 2003). In contrast, our research em-

phasizes the operational aspect of disruption management, i.e., how to make optimal operational decisions in the face of disruptions.

Chapter 3

MILP Model for Resource-Constrained Project Scheduling Problem

3.1 Problem Description and Notation

We consider the multi-mode and multiple resource RCPSP with finish-start precedence constraints. A project consists of n activities denoted by the set $\mathcal{A}$. For each activity $i \in \mathcal{A}$, one of M_i implementation options must be selected. Each option $m \in \mathcal{M}_i$ represents a unique resource usage and time duration combination, and is called a mode. Time is measured in evenly spaced increments that are defined so that it is possible for each activity to be assigned a start time at the beginning of a period and finish at the end of another. If an activity starts at time 4 and finishes at time 6, for example, it has a processing time (duration) of 3 time units and consumes resources in periods 4, 5 and 6.

Each resource $k \in \mathcal{K}$ has usage limits on some predefined time slots. The so-called partially renewable resource constraints were first studied by Bottcher et al. (1999) and include the traditional renewable and non-renewable constraints as special cases. To facilitate the analysis, we introduce two dummy activities to represent the start and finish milestones for the project, i.e., activities 0 and $n + 1$, respectively. Neither consumes any resources or time. The problem is to find a schedule that specifies modes and start times for all activities so that the makespan is minimized.

We use the following notation in describing our model.

Indices and sets

$\mathcal{A}$ = set of all actual activities; $\mathcal{A} = \{0, 1, \ldots, n+1\}$, where 0 and $n+1$ are dummy activities indicating the start and completion of the project, respectively

i, j = activity indices; $i, j \in \mathcal{A}$

$\mathcal{M}_i$ = set of modes for activity i

m = index for modes; $m \in \mathcal{M}_i = \{1, \ldots, M_i\}$; $M_0 = 1, M_{n+1} = 1$

$\mathcal{K}$ = set of resources

k = index for resources; $k \in \mathcal{K} = \{1, \ldots, K\}$

Π_k = set of time slots on which resource k is constrained

π_k = index for time slots on which resource k is constrained; $\pi_k \in \Pi_k$

$\mathcal{T}$ = set of time periods

t = time index; $t \in \mathcal{T} = \{0, 1, \ldots, T\}$, where T is an upper bound of the project competition time

$\mathcal{P}$ = 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; $(0, j) \in \mathcal{P}, \forall\ j \neq 0$; $(i, n+1) \in \mathcal{P}, \forall\ i \neq n+1$

Parameters

p_{im} = processing time (time duration) of activity i in mode m

r_{imk} = resource requirement per time period of resource k for activity i implemented in mode m

R_{π_k} = usage limit of resource k on time slot π_k

e_i = CPM lower bound on the earliest finish time for activity i

l_i = CPM upper bound on the latest finish time for activity i; $l_0 = 0$ because the beginning of the project is not delayed

Variables

x_{imt} = 1 if activity i finishes at time t in mode m, 0 otherwise; $i \in \mathcal{A}, m \in \mathcal{M}_i, e_i \leq t \leq l_i$

3.2 MILP Model

With the notation defined above, the MILP model for MRCPSP with the objective of minimizing makespan is

$$\text{Minimize} \sum_{t=e_{n+1}}^{l_{n+1}} t x_{n+1,1,t} \tag{3.1}$$

$$\text{subject to} \sum_{m \in \mathcal{M}_i} \sum_{t=e_i}^{l_i} x_{imt} = 1, \quad i \in \mathcal{A} \tag{3.2}$$

$$\sum_{m \in \mathcal{M}_i} \sum_{t=e_i}^{l_i} t x_{imt} - \sum_{m \in \mathcal{M}_j} \sum_{t=e_j}^{l_j} (t - p_{jm}) x_{jmt} \leq 0, \quad (i,j) \in \mathcal{P} \tag{3.3}$$

$$\sum_{t \in \pi_k} \sum_{i \in \mathcal{A}} \sum_{m \in \mathcal{M}_i} \sum_{q=\max(t,e_i)}^{\min(t+p_{im}-1,l_i)} r_{imk} x_{imq} \leq R_{\pi_k}, \quad \pi_k \in \Pi_k, k \in \mathcal{K} \tag{3.4}$$

$$x_{imt} \in \{0,1\}, \quad \forall\ i,m,t \tag{3.5}$$

The objective (3.1) is to minimize the project makespan. Because our solution approach is based on solving the LP relaxation of the ILP, any objective that is a linear function of the variables, such as the weighted sum of finish times or resource consumption, can be used in place of (3.1). Constraint (3.2) ensures that each activity finishes in exactly one time period and in one mode. Constraint (3.3) guarantees that the precedence relations are satisfied while constraint (3.4) limits resource usage to the amounts available in each time slot.

This formulation is sufficiently general to handle all types of resources. For example, consider two special cases. If

$$\Pi_k = \{\{1\}, \{2\}, \ldots, \{T\}\}$$

then resource k is called renewable. The usage of a renewable resource is limited in each time period, but there is no limit on total usage. Typical examples include

machines and certain types of labor. Constraint (3.4) for a renewable resource can be written as

$$\sum_{i\in\mathcal{A}}\sum_{m\in\mathcal{M}_i}\sum_{q=\max(t,e_i)}^{\min(t+p_{im}-1,l_i)} r_{imk}x_{imq} \le R_k, \quad t=1,\ldots,T; k\in\mathcal{K}_r \tag{3.6}$$

In contrast, a resource that is only limited by the total amount consumed over the project's life is called non-renewable. Money and materials often fall into this category. For non-renewable resource k,

$$\Pi_k = \{1,2,\ldots,T\}$$

and constraint (3.4) becomes

$$\sum_{i\in\mathcal{A}}\sum_{m\in\mathcal{M}_i} r_{imk}\sum_{t=e_i}^{l_i} x_{imt} \le R_k, \quad k\in\mathcal{K}_n \tag{3.7}$$

Note that in this formulation r_{imk} represents the total usage of resource k for activity i if mode m is selected, rather than the usage per unit time as is the case with renewable resources.

In model (3.1) – (3.5) the binary variables are indexed by activity, mode and time. Models with time-indexed zero-one variables have been proposed in the early research of project-scheduling (e.g., Pritsker et al., 1969). In our case, each additional renewable resource brings $|\mathcal{T}|$ more constraints to the model. For problems of realistic size, the large number of binary variables and constraints has frustrated attempts to solve RCPSPs directly by working with the ILP formulation. Nevertheless, the model does have its advantages. In practice, resource constraints may be much more complicated than those mentioned. Some resources may only be partially renewable, while others may have limits that vary with time. In addition, constraints may exist on certain combinations of resources with respect to a single activity. These types of restrictions do not add much complexity to the ILP, but

they are very difficult to handle with heuristics and explicit branch-and-bound. For a general linear objective function, there are few lower bounding methods except the LP relaxation. A solution approach based on (3.1) – (3.5) can take advantage of commercial MIP solvers, and is readily adaptable to problem variations. This is one of the primary motivations for our work.

3.3 Special Cases

In this section, we study some special cases related to resource requirements. In these special cases, we consider a general linear objective function other than the makespan.

3.3.1 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. However, the CPM is no longer applicable for the general linear objective function. Two subcategories are relevant.

Single mode case. In this case, no mode alternatives are available for the activities. Thus, the objective value depends only on the start and finish times of the activities. This problem is called resource-unconstrained project scheduling with start time dependent costs.

An integer programming model for this problem can be obtained by removing the resource-related constraints and variables from model (3.1) – (3.5). Using a tighter form of precedence constraints, we have

$$\text{(P0) Minimize } \sum_{i \in \mathcal{A}} \sum_{t \in \mathcal{T}_i} \alpha_{it} x_{it} \tag{3.8}$$

$$\text{subject to} \sum_{t \in \mathcal{T}_i} x_{it} = 1, \quad \forall\ i \in \mathcal{A} \tag{3.9}$$

$$\sum_{l=t-p_j+1}^{T} x_{il} + \sum_{l=0}^{t} x_{jl} \leq 1, \quad \forall\ (i,j) \in \mathcal{P}, t \in \mathcal{T} \tag{3.10}$$

$$x_{it} \in \{0,1\}, \quad \forall i \in \mathcal{A}, t \in \mathcal{T}_i \tag{3.11}$$

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 3.3.1. *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. (2001) showed that the LP relaxation of (3.8) – (3.11) is integral, implying that P0 is polynomially solvable. More recently, Möhring et al. (2003) showed that P0 can be transformed into a minimum cut problem on a directed graph which can be solved in time $O(nmT^2 \log(n^2T/m))$, where n is the number of activities and m is the number of precedence constraints. For each binary variable x_{it} in model (3.8) – (3.11), 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. The ILP model for this problem can be written as

$$\text{(P1) Minimize} \sum_{i \in \mathcal{A}} \sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} w_{imt} x_{imt} \tag{3.12}$$

$$\text{subject to} \sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} x_{imt} = 1, \quad \forall\ i \in \mathcal{A} \tag{3.13}$$

$$\sum_{m\in\mathcal{M}_i}\sum_{t\in\mathcal{T}_i}(t-p_{jm})x_{jmt}-\sum_{m\in\mathcal{M}_i}\sum_{t\in\mathcal{T}_i}tx_{imt}\geq 0,\ \forall\ (i,j) \quad (3.14)$$

$$x_{imt}\in\{0,1\},\quad \forall\ i\in\mathcal{A}, t\in\mathcal{T}_i \quad (3.15)$$

Theorem 3.3.2. *If each activity has at least two different modes, the resource unconstrained project scheduling problem with start-time dependent costs* (P1) *is NP-hard.*

PROOF. We will show that the 0-1 knapsack problem, which is known to be NP-hard (Garey and Johnson, 1979), is polynomially reducible to a multi-mode resource-unconstrained project scheduling problem with start-time dependent costs.

Given a set of items $\mathcal{N}=\{1,2,\ldots,n\}$ 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

$$\min\ \left\{\sum_{i\in\mathcal{N}}v_ix_i:\sum_{i\in\mathcal{N}}w_ix_i\leq b, x_i\in\{0,1\},\ \forall\ i\in\mathcal{N}\right\} \quad (3.16)$$

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 (3.16) into a multiple choice knapsack problem. Let $y_{j1}\equiv x_i$ and $y_{i2}\equiv 1-x_i\ \forall\ i\in\mathcal{N}$. Then it is obvious that the solution to (3.16) can be obtained by solving the following multiple choice knapsack problem

$$\text{Minimize}\quad \sum_{i\in\mathcal{N}}\sum_{m=1}^{2}v_{im}y_{im} \quad (3.17)$$

$$\text{subject to}\quad y_{i1}+y_{i2}=1,\quad \forall\ i\in\mathcal{N} \quad (3.18)$$

$$\sum_{i\in\mathcal{N}}\sum_{m=1}^{2}w_{im}y_{im}\leq b+\sum_{i\in\mathcal{N}}w_{i2} \quad (3.19)$$

$$y_{i1},y_{i2}\in\{0,1\},\quad \forall\ i\in\mathcal{N} \quad (3.20)$$

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 (3.17) – (3.20). 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 (3.17) – (3.20) as follows.

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

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 (3.19) of the multiple choice knapsack problem is infeasible. On the other hand, if constraint (3.19) 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 (3.19) is satisfied for the multiple choice knapsack problem and we can construct an optimal solution as follows: $y^*_{im} = \sum_{t \in \mathcal{T}_i} x^*_{imt}, \forall\ i \in \mathcal{N}, m \in \{1,2\}$. 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 $b + \sum_{i \in \mathcal{N}} w_{i2}$.

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 $O(|\mathcal{N}|)$ time, we conclude that the special case of (P1) examined above, and hence the general version of (P1), is NP-hard. □

3.3.2 Case with One Non-Renewable 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. When we 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 (3.12) – (3.15)

$$\sum_{i \in \mathcal{A}} \sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} r_{im} x_{imt} \leq R \tag{3.21}$$

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 3.3.3. *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). Also, the problem with only resource constraint is a multiple choice knapsack problem, which we have also showed to be NP-hard.

3.3.3 Case with One Renewable Resource

In this case, each activity has only one mode so activity $i \in \mathcal{A}$ 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.

$$\sum_{i \in A} \sum_{q=t}^{t+p_i-1} r_i x_{iq} \leq R, \quad \forall\ t \in \mathcal{T} \tag{3.22}$$

Combining constraint (3.22) with (3.8) – (3.11) gives an ILP model for the case with one non-renewable resource. We denote this problem as (P3).

Theorem 3.3.4. *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 (Garey and Johnson, 1979). □

3.4 Summary

In this chapter, we describe an integer linear programming model for the MRCPSP we are considering. It is shown that some very simplified cases are NP-hard.

Chapter 4

A Branch-and-Cut Procedure for the Multi-mode Resource-Constrained Project Scheduling Problem

4.1 Introduction

In the literature, most studies begin with a integer linear programming (ILP) formulation of RCPSP similar to the one in Section 3.2 , but have not been successful applying branch and bound based on the linear programming (LP) relaxation. The limiting factor seems to be the large number of binary variables in problems of practical size.

In this chapter, we present a branch-and-cut procedure for solving the MR-CPSP directly using the ILP model. The basic idea is to exploit the precedence constraints as much as possible. A precedence relation specifies a minimum time-distance between a pair of activities. We first show how lower bounds on the minimum distance can be derived from the precedence constraints and then propose a scheme for improving them based on the resource limits. These lower bounds are used to reduce the number of variables and to generate cuts that tighten the LP relaxations of the ILP. Because the minimum distances must be maintained in every feasible schedule, a bound tightening scheme can be applied at each node in the search tree. Finally, to obtain better incumbent solutions more quickly during the enumeration, we employ a a high level search strategy called *local branching* (Fischetti and Lodi, 2003). This strategy is designed to maintain the exactness of the core procedure but can be used as a heuristic if convergence is too slow. To date, there is no published work that exploits this approach.

4.2 Solution Procedure

Branch-and-cut (B&C) is a special version of branch-and-bound (B&B) where the LP relaxation of the ILP is used to obtain a bound at each node in the search tree. If a node has a fractional solution and cannot be fathomed, we try to find cuts (valid inequalities) that are violated by the fractional solution but are satisfied by all feasible integer solutions. If no cut can be found, branching is performed to create new nodes in the tree. (B&C) has proven to be effective in solving different combinatorial optimization problems, especially the traveling salesman problem (TSP) and its variants (Bard et al., 2002).

There are generally three types of cuts that can be used to tighten the LP bounds. The first are general purpose cuts that are applicable for any fractional extreme point, such as Gomory cuts and 0-1 disjunctive cuts. The second are derived either from constraints that take a standard form and are violated by the continuous problem, or from the logical implications of integer variables in the ILP model. For example, knapsack constraints give rise to cover inequalities and fixed charge network structures give rise to flow cover inequalities. Cuts such as these have been studied widely and most commercial codes provide options for generating them from the LP relaxation. The third type are derived from specific knowledge of a problem. Examples would be the comb inequalities for the TSP and the box constraints for the VRP. When trying to solve combinatorial optimization problems, branch-and-cut has often proved to be the most effective method. The results have been mixed, though, for less structured models.

Our solution approach makes use of the cut generating features built into the MIP solver that comes with CPLEX (ILOG, 2002). In addition, we provide problem-specific cuts to tighten the LP bounds as well as alternative bound tightening schemes and branching rules to accelerate convergence. Though not part of this paper, we have also developed a genetic algorithm to find feasible solutions (upper

bounds) when necessary. In the reminder of this section, we describe the various components of our approach.

4.2.1 Data Preprocessing

If optimality is not affected, it is desirable to remove unnecessary mode options and resource constraints in the ILP model. This will result in fewer variables and constraints, and perhaps a considerable savings in computational effort. The work by Hartmann (1999) on heuristics and branch-and-bound methods for the MRCPSP provides guidance for developing a preprocessing step to reduce model size.

Definition 4.2.1. Non-executable mode: an activity mode that cannot be executed in any feasible schedule.

Definition 4.2.2. Inefficient mode: an activity mode that is at least as long in duration and uses no less resources than another mode of the same activity.

Definition 4.2.3. Redundant non-renewable resource: a non-renewable resource such that the sum of maximal requests for this resource over all activities does not exceed its limit.

While it is obvious how to find inefficient modes and redundant non-renewable resources, not all non-executable modes can be found easily. If a mode has a renewable resource requirement larger than the limit, then it is non-executable. Another type of non-executable mode may be derived from the violation of non-renewable constraints. Suppose that the minimum amount of non-renewable resource k that could be requested over the life of the project is X and that the minimal usage of resource k for activity i is Y. If $r_{imk} - Y > R_k - X$, then mode m of activity i is non-executable because the limit of resource k will be violated if mode m is selected. Similar violations associated with combinations of non-renewable resources can also be found. The preprocessing procedure is outlined below.

Algorithm 4.2.1. DATA PREPROCESSING

begin

for $i \in \mathcal{A}, m \in \mathcal{M}_i, k \in \mathcal{K}_r$ **do**

if $r_{imk} > R_k$ **then** remove mode m of activity i

for $i \in \mathcal{A}, m \in \mathcal{M}_i, k \in \mathcal{K}_n$ **do**

if $r_{imk} - \min\{r_{ilk} : l \in \mathcal{M}_i\} > R_k - \sum_{j \in \mathcal{A}} \min\{r_{jlk} : l \in \mathcal{M}_j\}$ **then**

remove mode m of activity i

repeat

for $k \in \mathcal{K}_n$ **do**

if $\sum_{i \in \mathcal{A}} \bar{r}_{ik} < R_k$, where $\bar{r}_{ik} = \max\{r_{imk} : m \in \mathcal{M}_i\}$ **then**

remove resource k

for $i \in \mathcal{A}, m_1, m_2 \in \mathcal{M}_i$ **do**

if $(r_{im_1k} \leq r_{im_2k}, \forall\, k \in \mathcal{K})$ **and** $p_{im_1} \leq p_{im_2}$ **then**

remove mode m_2 of activity i

until no more inefficient modes are removed

end

The "repeat" part of the algorithm is needed because the removal of inefficient modes may reduce the maximal request of some non-renewable resources, which may lead to more redundant non-renewable resources. For the remainder of the paper, we assume that all problems have been preprocessed.

4.2.2 Reduction of Variables

An upper bound on the project completion time T and bounds on the earliest and latest finish times (e_i, l_i) are required to fully define the ILP model (3.1) - (3.5). For minimum makespan problems, the upper bound T can be the makespan of the best feasible solution, perhaps obtained with a heuristic. It is sometimes imposed as a constraint if other objective functions are used.

The earliest finish time (EFT) of activity i is defined as the earliest time

period t such that there exists a feasible schedule in which activity i finishes at time t. Similarly, the latest finish time (LFT) of activity i is defined as the latest time period t such that there exists a feasible schedule in which activity i finishes at time t for a given upper bound on the project makespan. Of course, there is no need to define binary variables for the cases where an activity finishes before its EFT or after its LFT; however, finding these values is an NP-hard problem. For the minimum makespan problem, if we know the EFT for activity $n+1$, then we have the optimal solution. In any case, it is not necessary to know EFT and LFT exactly for all activities to formulate a valid ILP model. It is sufficient if we can calculate lower and upper bounds, respectively, on these values, though better bounds will lead to smaller models.

Activity i cannot finish before the lower bound e_i, which is also viewed as a lower bound on the length of time between the finish of dummy 0 and activity i. Similarly, $T-l_i$ is a lower bound on the time between the finish time of activity i and the finish time of dummy $n+1$. The distances e_i and $T-l_i$ are similar to the concepts of head (or release time) and tail (or delivery time) in machine scheduling (Balas et al., 1995). However, they are a result of precedence and resource constraints, rather than imposed as parameters. Those time distances must be maintained in every feasible solution. In addition, for each pair of activities $(i,j) \in \mathcal{P}$, there is also a minimum distance between the finish time of activity i and the start time of activity j (we will simply refer this as the distance between i and j). As we will see later, these minimum distances are helpful in improving the bounds on finish times and reducing the size of subproblems in the branch-and-cut procedure.

For each $(i,j) \in \mathcal{P}$, let $d_{ij}+1$ be a lower bound on the length of time between the finish of activity i and the start of activity j (d_{ij} is not defined for $(i,j) \notin \mathcal{P}$). We call the $(n+1) \times (n+1)$ matrix $D \equiv [d_{ij}]$ the distance matrix for the project. Letting $\underline{p}_i = \min\{p_{im} : m \in \mathcal{M}_i\}$, we have the following.

Proposition 4.2.1. *If $(i,k) \in \mathcal{P}$ and $(k,j) \in \mathcal{P}$, then $d_{ij} \geq d_{ik} + d_{kj} + \underline{p}_k$.*

The proof is straightforward since activity k must be executed after activity i finishes and before the start of activity j. This proposition indicates that an increase in d_{ij} may result in an increase in the lower bound on the distance between any predecessor of i and any successor of j. The distance matrix is defined in the way so that it is independent of mode selection of activities in a multi-mode project. This definition of distance matrix is sightly different from those in the literature (Bartusch et al., 1988), where the distances between start times are used.

The relationship in Proposition 4.2.1 can be viewed as a transitive property of the matrix D. The following procedure, which is similar to the Floyd-Warshall algorithm, can be used to transform the values of d_{ij} so that they satisfy the condition in the proposition.

Algorithm 4.2.2. Update Distance Matrix
Input: 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** $d_{i,i+k} < d_{ij} + d_{j,i+k} + \underline{p}_j$ **then**
          $d_{i,i+k} = d_{ij} + d_{j,i+k} + \underline{p}_j$
end

In fact, d_{ij} is a lower bound on the makespan of a subproject with activity set $\{k : (i, k), (k, j), (i, j) \in \mathcal{P}\}$. Therefore, any lower bound method for MRCPSP can be employed to obtain D. For CPM lower bounds, we assign 0 as the initial value of d_{ij} for $(i, j) \in \mathcal{P}$. Now, letting D^0 be the distance matrix after applying Algorithm 4.2.2, we have $e_i = d^0_{0i} + \underline{p}_i$ and $l_i = T - d^0_{i,n+1}$.

If a method that gives a better lower bound than CPM is available, we can use it iteratively to reduce the number of variables in model (3.1) - (3.5). Each

time we find a better bound, we apply Proposition 4.2.1 to try to improve the bounds for other distances as well. If any tighter bounds on EFTs or LFTs are obtained, the ILP model is updated by removing variables that must be zero. Each time before applying the lower bounding method, we run our genetic algorithm (GA) to try to find a feasible schedule with a makespan of d_{ij} for the subproject $\{k : (i,k) \in \mathcal{P}, (k,j) \in \mathcal{P}\}$. If successful, there is no need to apply the lower bounding method because the lower bound on d_{ij} cannot be improved. This step can save some time because the GA runs much faster than lower bounding methods for subprojects.

In the variable reduction procedure that follows, it is assumed that the best lower bound available is used. This bound may be obtained from any number of methods.

Algorithm 4.2.3. VARIABLE REDUCTION

Input: Precedence matrix $\mathcal{P}$ and distance matrix D

begin

for $(i,j) \in \mathcal{P}$ **do** $d_{ij} = 0$

call UpdateDistanceMatrix$(D, \underline{p})$

for $(i,j) \in \mathcal{P}$ **do**

begin

Apply GA to subproject $\{k : (i,k) \in \mathcal{P}, (k,j) \in \mathcal{P}\}$

if no feasible schedule is found with makespan $\leq d_{ij}$ **do**

begin

Calculate a new lower bound d^* for the distance between i and j

if $d_{ij} < d^*$ **then**

begin

$d_{ij} = d^*$

call UpdateDistanceMatrix$(D, \underline{p})$

for $k = 1$ **to** $n + 1$ **do**
begin
if $d_{0k} > d^0_{0k}$ **then**
for $m \in \mathcal{M}_k, e_k \leq t \leq d_{0k} + p_{km} - 1$ **do** $x_{kmt} = 0$
if $d_{k,n+1} > d^0_{0,n+1}$ **then**
for $m \in \mathcal{M}_k, l_k \leq t \leq T - d_{k,n+1}$ **do** $x_{kmt} = 0$
end
end
end
end

The use of Algorithm 4.2.3 is illustrated with the example given in Figure 4.1 which shows a multi-mode project with 5 activities and 1 renewable resource.

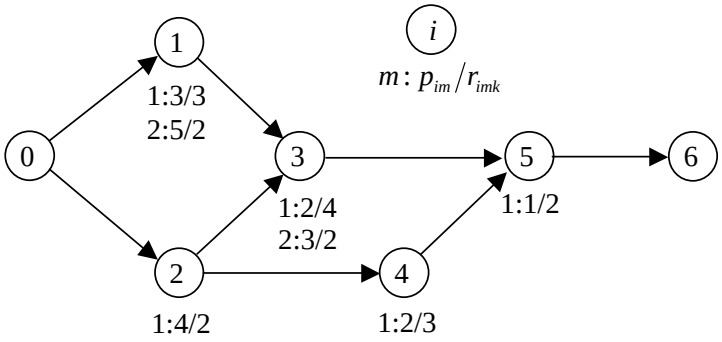

Figure 4.1: A 5-activity project

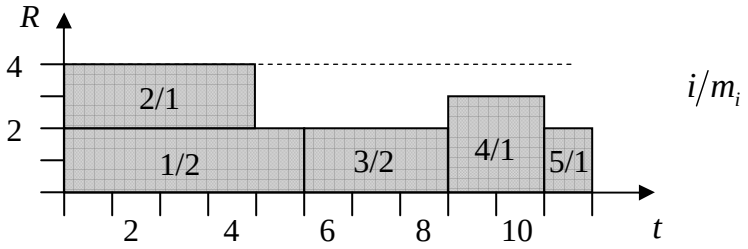

Figure 4.2: Feasible schedule for example project

The CPM computations give the following distance matrix.

$$D^0 = \begin{bmatrix} 0 & 0 & 4 & 4 & 6 & 7 \\ & - & 0 & - & 2 & 3 \\ & & 0 & 0 & 2 & 3 \\ & & & - & 0 & 1 \\ & & & & - & 1 \\ & & & & & 0 \end{bmatrix}$$

Table 4.1 lists the lower bounds on EFTs and upper bounds on LFTs computed from the distance matrix D^0. The range of possible finish times (in square brackets) for each activity and the corresponding binary variables defined in the ILP model are as follows.

Activity 1: [3 - 8], $x_{113} - x_{118}, x_{123} - x_{128}$

Activity 2: [4 - 8], $x_{214} - x_{218}$

Activity 3: [6 - 10], $x_{316} - x_{3,1,10}, x_{326} - x_{3,2,10}$

Activity 4: [6 - 10], $x_{416} - x_{4,1,10}$

Activity 5: [7 - 11], $x_{517} - x_{5,1,11}$

Activity 6: [7 - 11], $x_{617} - x_{6,1,11}$

Now, because e_i is calculated using the smallest processing time for activity i, variables $\{x_{imt} : e_i \leq t < e_i + p_{im} - \underline{p}_i, m \in \mathcal{M}_i\}$ can be set to zero. For example, mode 2 of activity 1 has a processing time of 5. Therefore, its earliest finish time is

Table 4.1: EFT and LFT bounds for example project

	Lower bound on EFT, e_i		Upper bound on LFT, l_i	
Activity, i	CPM	Improved	CPM	Improved
0	0	0	0	0
1	3	3	8	8
2	4	4	8	6
3	6	7	10	10
4	6	6	10	10
5	7	9	11	11
6	7	9	11	11

5 and we can set x_{123} and x_{124} to zero. Similarly, x_{326} can be set to zero because it can only start after both activity 1 and activity 2 have finished. This implies that its earliest start time is 4, which, in turn, implies that its earliest finish time is 7.

We now look at how an improved distance matrix can reduce the range of finish times. Because finding the minimum distance between a pair of activities is itself an NP-hard problem, lower bounding methods are used to improve the distances in Algorithm 4.2.3. For the example, we obtain the improved distances by enumerating all possible schedules for the subprojects. From Figure 4.1 we see that there are two activities (1 and 2) between activities 0 and 3. If mode 1 is selected for activity 1, then activity 1 and 2 cannot overlap due to a resource conflict. Therefore, the distance between activity 0 and 3 must be at least 5. Similarly, the distance between 2 and 5 must be at least 4. Let $d_{03} = 5$ and $d_{25} = 4$ in D^0. After updating

the matrix to satisfy Proposition 4.2.1, we get the following.

$$D = \begin{bmatrix} 0 & 0 & 5 & 4 & 8 & 9 \\ & - & 0 & - & 2 & 3 \\ & & 0 & 0 & 4 & 5 \\ & & & - & 0 & 1 \\ & & & & - & 1 \\ & & & & & 0 \end{bmatrix}$$

The improved bounds on the EFTs and LFTs are also shown in Table 4.1. We see that the upper bound on the LFT for activity 2 is reduced from 8 to 6, while the lower bounds on the EFT for activities $3, 5$ and 6 are increased. As a result, we can also set $x_{217}, x_{218}, x_{316}, x_{327}, x_{517}, x_{518}, x_{617}, x_{618}$ to zero.

Any lower bounding method for the makespan minimization problem can be used in the variable reduction procedure outlined in Algorithm 4.2.3. We use two. The first is based on the usage of renewable resources (Hartmann, 1999). The second is derived from a truncated solution to the ILP. We begin solving the model with B&C, but halt the computations after some predetermined amount of time. In the process, cuts are generated aggressively to obtain a tight lower bound. The time limit is adaptively set according to the number of activities and variables in the model.

4.2.3 Valid Inequalities for the MRCPSP

Cuts are redundant to the ILP formulation, but not to the LP relaxation, so adding them may tighten the LP bounds by removing fractional solutions. We now discuss several cuts that can be derived from the resource and precedence constraints.

Cuts from resource conflicts. As we see in model (3.1) - (3.5), all resource constraints are in the form of knapsack constraints with generalized upper bound (GUB) vari-

ables (Gu et al., 1998). That is, all variables are divided into mutually exclusive subsets, and in each subset one variable must be one. Let $\mathcal{N}_i, i \in \mathcal{I} = \{1, 2, \cdots, I\}$ be the variable sets such that $\mathcal{N}_i \cap \mathcal{N}_j = \emptyset, \forall\ i \neq j \in \mathcal{I}$. The knapsack constraint with GUB variables can be written as

$$\sum_{j \in \cup_{i \in \mathcal{I}} \mathcal{N}_i} a_j x_j \leq b, \sum_{j \in \mathcal{N}_i} x_j = 1, \quad x_j \in \{0, 1\}, \quad \forall i \in \mathcal{I} \tag{4.1}$$

The GUB cover inequality

$$\sum_{j \in \mathcal{C}} x_j \leq |\mathcal{C}| - 1 \tag{4.2}$$

is a valid inequality for the polytope associated with the constraints in (4.1) if $\sum_{j \in \mathcal{C}} a_j > b$ and no two elements of $\mathcal{C}$ belong to the same subset. Extending and lifting procedures can be applied to make these cuts tighter. For details on the complexity and the computation of GUB cuts and lifting procedures, see Gu et al. (1998, 1999).

These cuts can be developed whenever there is a resource conflict. For a non-renewable resource, a GUB cover cut means that a certain combination of modes among activities is not allowed due to resource limits. For a renewable resource, a GUB cover cut prevents two or more activities from being executed at the same time if the combined resource usage exceeds the limit. For example, in the project shown in Figure 4.1, if mode 1 of activity 1 is chosen, then activity 1 and 2 cannot be executed at the same time in any time period. This results in the following GUB cover cuts corresponding to resource conflicts in different time period.

$t = 1$: $x_{113} + x_{214} \leq 1$

$t = 2$: $x_{113} + x_{114} + x_{214} + x_{215} \leq 1$

$t = 3$: $x_{113} + x_{114} + x_{115} + x_{214} + x_{215} + x_{216} \leq 1$

$t = 4$: $x_{114} + x_{115} + x_{116} + x_{214} + x_{215} + x_{216} \leq 1$

$t = 5$: $x_{115} + x_{116} + x_{117} + x_{215} + x_{216} \leq 1$

$t = 6$: $x_{116} + x_{117} + x_{118} + x_{216} \leq 1$

Generating GUB cover cuts from given constraints has been implemented in CPLEX 7.5. Because our resource constraints have the standard structure of a knapsack problem with GUB variables, we choose to use the built-in algorithm to generate the cuts given in (4.2).

Cuts from precedence relations. For $(i,j) \in \mathcal{P}$, if activity i finishes after time t, then activity j can start no earlier than $t + d_{ij} + 1$. Therefore we have the following valid inequalities

$$\sum_{m \in \mathcal{M}_i} \sum_{q=e_i}^{t} x_{imq} \geq \sum_{m \in \mathcal{M}_j} \sum_{q=e_j}^{\min(l_j, t+p_{jm}+d_{ij})} x_{jmq}, \tag{4.3}$$

where $\max(e_i - 1, e_j - d_{ij} - \underline{p}_j - 1) \leq t \leq \min(l_i, l_j - \underline{p}_j - d_{ij})$

Some of the cuts in (4.3) may be satisfied automatically when variables are set to zero in Algorithm 4.2.3. For precedence relation $(2,4)$ in the example project, $d_{24} = 0$, which means that activity 4 can start immediately after activity 2 finishes. If $x_{214} = 0$, which means that activity 2 finishes after time 4, then activity 4 cannot finish earlier than time 7 (or start earlier than time 6), which leads to $x_{416} = 0$. Therefore,

$$x_{214} \geq x_{416}$$

is a valid inequality. Similarly, if $x_{214} = 0$ and $x_{215} = 0$, then activity 2 cannot finish earlier than time 6 implying that $x_{416} = 0$ and $x_{417} = 0$ because activity 4 has a duration of 2. This leads to the cut

$$x_{214} + x_{215} \geq x_{416} + x_{417}$$

Fortunately, it is not necessary to consider cuts from all precedence relations because many are redundant. This is stated formally in the following proposition.

Proposition 4.2.2. *For $(i,j) \in \mathcal{P}$, if there exists an activity k such that $(i,k) \in \mathcal{P}, (k,j) \in \mathcal{P}$ and $d_{ij} = d_{ik} + d_{kj} + \underline{p}_k$, then precedence cuts associated with precedence relation (i,j) are implied by cuts for (i,k) and (k,j).*

Proof. If $e_j - d_{ij} - \underline{p}_j - 1 \leq t = e_i - 1$, Eq. (4.3) becomes

$$0 \geq \sum_{m \in \mathcal{M}_j} \sum_{q=e_j}^{\min(l_j, t+p_{jm}+d_{ij})} x_{jmq} \tag{4.4}$$

If $e_i - 1 \leq t = e_j - d_{ij} - \underline{p}_j - 1$, Eq. (4.3) becomes

$$\sum_{m \in \mathcal{M}_i} \sum_{q=e_i}^{t} x_{imq} \geq 0 \tag{4.5}$$

For $t < \max(e_i - 1, e_j - d_{ij} - \underline{p}_j - 1)$, Eq. (4.3) is implied by either Eq. (4.4) or Eq. (4.5). Similarly, cuts for $t > \min(l_i, l_j - \underline{p}_j - d_{ij})$ are implied by cuts at $t = \min(l_i, l_j - \underline{p}_j - d_{ij})$, so Eq. (4.3) holds for any $t > 0$.

Now, for $t > 0$, precedence relation (i, k) gives cut

$$\sum_{m \in \mathcal{M}_i} \sum_{q=e_i}^{t} x_{imq} \geq \sum_{m \in \mathcal{M}_k} \sum_{q=e_k}^{\min(l_k, t+p_{km}+d_{ik})} x_{jmq} \tag{4.6}$$

$$\geq \sum_{m \in \mathcal{M}_k} \sum_{q=e_k}^{\min(l_k, t+\underline{p}_k+d_{ik})} x_{kmq} \tag{4.7}$$

Replacing t with $t + \underline{p}_k + d_{ik}$ in Eq. (4.3), precedence relation (k, j) gives

$$\sum_{m \in \mathcal{M}_k} \sum_{q=e_k}^{\min(l_k, t+\underline{p}_k+d_{ik})} x_{kmq} \geq \sum_{m \in \mathcal{M}_j} \sum_{q=e_j}^{\min(l_k, t+\underline{p}_k+d_{ik}+p_{jm}+d_{kj})} x_{jmq} \tag{4.8}$$

Combining Eq. (4.7) and (4.8), leads to

$$\sum_{m \in \mathcal{M}_i} \sum_{q=e_i}^{t} x_{imq} \geq \sum_{m \in \mathcal{M}_j} \sum_{q=e_j}^{\min(l_j, t+\underline{p}_k+d_{ik}+p_{jm}+d_{kj})} x_{jmq} \tag{4.9}$$

which is identical to the inequality in Eq. (4.3) for precedence relation (i, j) when $d_{ij} = d_{ik} + d_{kj} + \underline{p}_k$. □

Cuts for precedence relations (i, k) and (k, j) maintain a distance of at least $d_{ik} + d_{kj} + \underline{p}_k$ between activities i and j. This is the distance to be enforced by

cuts for precedence relation (i, j) if $d_{ij} = d_{ik} + d_{kj} + \underline{p}_k$. Regarding implementation, Eq. (4.3) is used to generate cuts if either there is no k such that $(i, k), (k, j) \in \mathcal{P}$, or the condition in Proposition 4.2.2 is not satisfied. Because the number of these constraints is limited to $O(|\mathcal{P}| \times \max_i\{l_i - e_i\})$, we identify them in advance and let CPLEX add them when a violation is detected. This eliminates the need to solve the separation problem during B&B.

4.2.4 Branching Rules and Bound Tightening

For an ILP with a large number of binary variables, branching on a single variable is usually inefficient because the two subproblems that result are almost identical to the original. In our model, the binary variables $\{x_{imt} : m \in \mathcal{M}_i, e_i \leq t \leq l_i\}, \forall\ i \in \mathcal{A}^+$ naturally divide into type 1 special ordered sets (see (3.2)). When CPLEX is used, each variable in a set must be assigned a unique weight. Given a fractional LP solution, branching can be performed on the set by dividing it into two subsets according to the variable weights. The rule that is followed is that the variables in one subset all have weights greater than the average weight of the fractional solution value, and the variables in the other subset all have weights less than the average value. Two branches are created by respectively setting the variables in each subset to zero.

We choose time t as the weight for variable x_{imt} so that $\sum_{m \in \mathcal{M}_i} \sum_{t=e_i}^{l_i} t x_{imt}$ represents the the weighted average finish time of activity i. This means that branching is performed on the finish time of an activity. In the implementation, we actually use $t + (m-1)\epsilon$, where ϵ is a very small positive number, as the weight for x_{imt} because of the uniqueness requirement. This small modification does not affect branching behavior.

Now suppose that a branch is to be created based on the weighted average finish time t_b of activity i where t_b is fractional. Activity i can only finish before t_b in one branch and after t_b in the other. This is equivalent to letting $d_{i,n+1} = T - \lceil t_b \rceil$

in one branch and $d_{0i} = \lfloor t_b \rfloor - \underline{p}_i$ in the other branch. As discussed in Section 4.2.2, once one entry in the distance matrix is changed, we may be able to fix additional variables along a path in the search tree by updating the distance matrix according to Proposition 4.2.1. This bound tightening scheme is executed every time a special ordered set (SOS) is used for branching. We have implemented it as a CPLEX branch callback function, but do not specify the logic used to select the set. We allow CPLEX to do this automatically.

Algorithm 4.2.4. SOS Bound Tightening

Input: LP solution $\bar{x} = \{\bar{x}_{imt}\}$ associated with the current node, upper bounds $u = \{u_{imt}\}$ on $x = \{x_{imt}\}$ at the current node

begin

for SOS branching **do**

begin

let i_b = activity selected for branching

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

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

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

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**

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

$D^1 = \bar{D}; D^2 = \bar{D}$

$D^1(i_b, n + 1) = T - \lceil t_b \rceil$

call UpdateDistanceMatrix$(D^1, \underline{p})$

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

Create the first branch by setting upper bounds on variables in S_1 to 0

$D^2(0, i_b) = \lfloor t_b \rfloor - \underline{p}_{i_b}$

call UpdateDistanceMatrix$(D^2, \underline{p})$

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

Create the second branch by setting upper bounds on variables in S_2 to 0

end

end

We again use the example project to illustrate the ideas in the algorithm. Suppose that after solving the LP relaxation we get a fractional solution in which the values for variables associated with activity 4 are

$$x_{416} = 0.3,\ x_{417} = 0,\ x_{418} = 0.7,\ x_{419} = 0,\ x_{4,1,10} = 0$$

Recall that for SOS branching, the time t is used as the weight for variable x_{imt}. Therefore, the weighted average solution value for the accompanying SOS is $0.3 \times 6 + 0.7 \times 8 = 7.4$. If branching is to be performed on the set $\{x_{41t}, 6 \leq t \leq 10\}$, two branches will be created based on the value 7.4. In branch 1, $x_{418}, x_{419}, x_{4,1,10}$ (variables with weight greater than the weighted average) are set to zero, and in branch 2, x_{416}, x_{417} are set to zero.

We now show how the precedence relations allow us to set additional variables to zero in each branch. Because activity 2 is a predecessor of activity 4, it must finish at least 2 time periods (the duration of activity 4) earlier than activity 4. After setting $x_{418}, x_{419}, x_{41,10}$ to zero, the new latest time for activity 4 in branch 1 is 7 so the latest possible finish time for activity 2 is $7 - 2 = 5$. If this were not the case, the precedence relation between activity 2 and 4 would be violated in this branch. As a result, we can set $x_{216}, x_{217}, x_{218}$ to zero in branch 1. Subsequently, x_{617} and x_{618} in branch 2 can also be set to zero.

For the minimum makespan problem, additional bound tightening is possible by noting that the upper bound for LFT, $l_i = T - d^0_{i,n+1}$, depends of the upper bound on the project completion time T. If an incumbent solution with shorter makespan $T^* < T$ is found, then an improved lower bound on LFT is $T^* - d^0_{i,n+1}$. Those variables associated with activities that finish after the new lower bounds can be set to zero.

4.2.5 High-level Search Strategy: Local Branching

Despite the various techniques that we have developed to accelerate the solution process, initial testing indicated that many problem instances were still difficult to solve, primarily due to the large number of binary variables. In response, we adopted a high-level solution strategy developed by Fischetti and Lodi (2003) called *local branching*. The procedure is based on ideas common to metaheuristics, but neighborhoods are defined by linear inequalities called local branching cuts rather than by exchanges and relocations. The neighborhood search is performed implicitly by the MIP solver. The strategy is designed to produce improved solutions at early stages of the computations, and can be implemented as an exact procedure or as a heuristic. Kilby et al. (1998) applied a similar idea called *large neighborhood search* in solving vehicle routing problems, where the neighborhood is defined as the region in which a part of a feasible solution is fixed.

In our model, every feasible integer solution has exactly $n + 2$ variables that are equal to 1 and the others 0. Given a feasible reference solution $\bar{x}$, let $\bar{S} = \{j : \bar{x}_j = 1\}$ be the binary support of $\bar{x}$. For some positive parameter k, a $k - OPT$ neighborhood $\mathcal{N}(\bar{x}, k)$ is defined as the set of feasible solutions of model (3.1) - (3.5) satisfying the additional constraint

$$\triangle(x, \bar{x}) \equiv \sum_{j \in \bar{S}} (1 - x_j) \leq k \tag{4.10}$$

The idea is to partition the current solution space into two branches. The left

branch, which is the neighborhood $\mathcal{N}(\bar{x}, k)$, includes all feasible solutions that have at least $n - k + 2$ activities with the same mode and finish time as the reference solution. In other words, at most k variables in the supporting set $\bar{S}$ can become zero. The right branch is defined by reversing the local branching constraint (4.10); i.e.,

$$\triangle(x, \bar{x}) \geq k + 1 \tag{4.11}$$

The approach is based on the premise that it is possible to quickly find better solutions that are within some small "distance" of the incumbent. At each iteration, we move from one reference solution to another if a new incumbent is found. If an improved solution cannot be found within some time limit, diversification strategies are used to move to a new starting point. Four parameters must be specified to implement the procedure: the neighborhood size k, the node time limit (*node_t_lim*), the total time limit (*total_t_lim*), and the maximum number of diversification steps (*div_lim*). At termination, the current best solution is returned. The outline of the procedure follows.

Algorithm 4.2.5. Local Branching

Input: feasible solution $\bar{x}$, objective value $f(\bar{x})$, k, *node_t_lim*, *total_t_lim*, *div_lim*

Initialization: $dv = 0$; *diversify* = **false**, *first* = **false**, $TL = node\_t\_lim$;

$UB = bestUB = f(\bar{x})$, $x^* = \bar{x}$

begin

repeat

Search $\mathcal{N}(\bar{x}, k)$ for solutions with objective value less than UB within time limit TL; search stops after one feasible solution is found **if** $first =$ **true**

if problem is infeasible **or** solved optimally **then**

reverse the local branching constraint and add it to the ILP model

else remove the local branching constraint

if solution $\hat{x}$ is found **then do**

begin

if (f($\hat{x}$) $< bestUB$) **then**

$x^* = \hat{x}$; $bestUB = f(\hat{x})$

$\bar{x} = \hat{x}$; $UB = f(\hat{x})$;

$diversify =$ **false**; $first =$ **false**; $TL = node\_t\_lim$;

end

if problem is infeasible **then do**

if $diversify =$ **true then**

strong diversification: $UB = \infty; TL = \infty; first =$ **true**; $dv = dv + 1$

else soft diversification: $k = k + \lfloor k/3 \rfloor; diversify =$ **true**

if no better solution found **then do**

if $diversify =$ **true then**

strong diversification: $UB = \infty; TL = \infty; first =$ **true**; $dv = dv + 1$

else soft diversification: $k = k - \lfloor k/3 \rfloor$; $diversify =$ **true**

until $dv > dv\_lim$ **or** elapsed computation time $> total\_t\_lim$

Solve the problem with local branching constraints added if total elapsed time $< total\_t\_lim$

output x^*

end

When strong diversification is needed, Fischetti and Lodi (2003) proposed using the MIP solver to find new reference solutions. Stopping when the first solution is found is one option; however, regardless of the logic chosen, the time spent at this step is "wasted" in the sense that it does not reduce the solution space. In addition, our testing has shown that the new reference solutions are usually worse than the incumbent. As an alternative, we run our GA to obtain new starting reference solutions when the need for diversification is indicated. The advantage of using heuristics for this purpose is that they can often find new solutions very quickly. If the computations do not yield a new solution with makespan less than the incumbent

makespan + 2 in a predefined amount of time, we quit local branching and solve the problem directly with CPLEX (possibly with some local branching constraints included).

The neighborhood defined by the local branching constraint (4.10) requires that a new schedule must have at least $n - k + 2$ activities with the same mode and finish time as the reference solution. This condition is very restrictive for a project schedule because activity start times most likely depend on each other. For example, if we change the mode of an activity, all activities scheduled after it may be delayed for some amount of time while keeping roughly the same relative positions. To cope with this situation, we define a new neighborhood for local branching. Suppose we have a reference solution $\{\bar{x}_{i\bar{m}_i\bar{t}_i} : i \in \mathcal{A}^+\}$. The support set for the new neighborhood is defined as

$$\bar{S}(\delta) = \{(i, \bar{m}_i, t) : \max(e_i, \bar{t}_i - \delta) \leq t \leq \min(l_i, \bar{t}_i + \delta), i \in \mathcal{A}^+\} \tag{4.12}$$

The set $\{x_{imt} : (i, m, t) \in \bar{S}(\delta)\}$ includes variables which are within a distance of δ from the activity finish times of the reference solution. This support set leads to the same neighborhood defined by (4.10) when $\delta = 0$.

4.3 Computational Results

Benchmark problems developed by Kolisch et al. (1995) have been used widely to test various exact and heuristic methods for project schedules. We evaluated our methodology by solving their 20-job and 30-job instances of the MR-CPSP (denoted by $J20$ and $J30$, respectively). These are the most difficult problem sets available for the model we are considering. All 554 instances of $J20$ were solved optimally by others using explicit branch-and-bound (see Sprecher and Drexl, 1998), but none of the solutions reported for the 552 instances in $J30$ were confirmed to be optimal. The data sets and best available solutions can be found at (http://www.bwl.uni-kiel.de/Prod/psplib/).

All codes were written in GNU C++ and run on a Dell Linux workstation with a 1.8GHz Xeon processor. The MIP solver in CPLEX 7.5 was called through ILOG Concert Technology 1.2 during B&C. Initial feasible solutions were obtained with our genetic algorithm with both generation number and population number set to 100. The better of two runs was chosen. Priority rule based heuristics were used to initialize the GA.

Neighborhoods defined by (4.12) with $\delta = 1$ were used in local branching. The parameter k was set to 6 for the 20-job problems and 5 for the 30-job problems. Soft diversification was performed by increasing k by $\lfloor k/3 \rfloor$ when the ILP at the current node proved to be infeasible, or decreasing it by $\lfloor k/3 \rfloor$ when no feasible solution was found within $node\_t\_lim = 1000$ seconds. The total solution time ($total\_t\_lim$) was set to 3600 seconds for the 30-job problems and 1800 seconds for the 20-job problems. However, if the absolute optimality gap dropped to less than 1 within these time limits, the computations were halted. When strong diversification was needed, we applied the GA to find a new starting feasible solution with a makespan less than the current best makespan +2. If the number of strong diversification steps reached a predefined limit, or the GA could not find a new starting point in 5 runs of $3n$ generations by $3n$ populations, where n is the number of activities, we stopped the local branching and called CPLEX for the remaining time. The maximum number of strong diversification steps was set to 6 for the 20-job problems.

The CPLEX MIP solver has many parameters that can be set to customize the computations. Better performance can often be obtained by experimenting with various settings. In our case, the best results were achieved by setting MIP emphasis to "feasibility" and using the primal simplex algorithm to solve the LPs. To limit the size of the LPs arising at each node, we fixed the maximum number of additional cuts to 5 times the number of constraints (by setting *cutsfactor* to 6). While more cuts lead to tighter bounds and hence fewer LPs, as the size of the LPs

grow improvement in the objective value slows. This is known as the tailing off effect. At some point, overall performance will begin to level off and then decline if the constraints are not managed properly.

CPLEX also provides parameter settings for several types of cuts, such as GUB cover cuts, clique cuts, and cover cuts. It is possible to either disable the generation of any one of them, generate them aggressively, or let the default logic handle the process automatically. Our initial testing was aimed at determining the most effective settings.

In the previous section, we discussed four possible ways directly with CPLEX improving the solution process; namely, variable reduction (VR), precedence cuts (PC), local branching (LB), and SOS bound tightening (BT). Each of these components can be applied separately or in combination. The following notation is used in reporting the results. All times are in seconds.

t_{TOT}	=	total CPU time
t_{OPT}	=	CPU time to find the optimal solution
t_{BEST}	=	CPU time to find the best solution when a problem is not solved optimally
t_{VR}	=	CPU time for the variable reduction (VR) procedure
t_{BT}	=	CPU time for the bound tightening (BT) procedure
t_{GA}	=	CPU time for the genetic algorithm before B&C
t_{SD}	=	CPU time for strong diversification (looking for new reference solutions) in local branching
t_{BB}	=	CPU time for prior results obtained with explicit B&B (Intel 80486/66MHz, Sprecher and Drexl (1998))
N	=	number of problem instances in a group
N_o	=	number of instances solved optimally
N_b	=	number of instances for which best solutions are obtained
N_{clq}	=	number of clique cuts applied

N_{cov}	=	number of cover cuts applied
N_{gub}	=	number of GUB cover cuts applied
N_{pc}	=	number of precedence cuts applied
LB_{CPM}	=	CPM lower bound
LB_{LP}	=	LP lower bound
LB_{BC}	=	lower bound obtained with B&C if not proven to be optimal
UB_{BEST}	=	best known solution
GA Gap	=	% gap between the GA solution and optimal/best-known solution
BC Gap	=	% gap between the B&C solution and optimal/best-known solution
Optimal	=	minimum makespan if problem is solved optimally

Before applying the full methodology, we evaluated the effectiveness of the GUB cuts and precedence cuts (PC) by solving the 10 J2013 (j2013_1 − j2013_10) instances with different parameter settings. Variable reduction and bound tightening were applied in all the cases, and clique cuts were set to be generated aggressively. For either GUB cover cuts disabled (GUB = −1) or generated most aggressively (GUB = 2), we solved the problems with and without precedence cuts. Table 4.2 lists the computation results for each case, including CPLEX's default settings. The entries represent average values for the 10 instances. The average number of precedence cuts generated after the variable reduction procedure but before B&C was 324.

The results show that precedence cuts do help when GUB cover cuts are disabled. In this case, 231 cuts were added to the model. However, when GUB cover cuts were generated aggressively, adding precedence cuts increased the overall solution time. GUB cover cuts are more effective than precedence cuts for our problem because all the resource constraints are in the form of knapsack constraints with GUB constrained variables. It should be mentioned that if the highest level

for GUB cover cuts is not explicitly chosen and the default setting is used, many more cover cuts than GUB cover cuts are generated.

Table 4.2: Effect of cuts on the 10 instances in J2013

Settings	t_{TOT}	t_{OPT}	Nodes	LP iterations	N_{clq}	N_{cov}	N_{gub}	N_{pc}
CPLEX default	913.7	460.0	33,086	3,163,775	544	2	32	–
GUB = −1	1,457.7	798.6	56,119	4,929,339	31	572	–	–
GUB = −1, PC	1,017.4	332.9	46,135	3,501,995	26	327	–	231
GUB = 2	143.3	79.2	1,680	324,607	17	59	443	–
GUB = 2, PC	173.1	112.2	2,780	424,706	16	41	277	216

For the remaining tests, we excluded precedence cuts and set CPLEX to generate GUB cover cuts and clique cuts most aggressively. Table 4.3 gives the average results for different combinations of VR, BT and LB for the 10 instances in J1013. As expected, CPU time correlates positively with the number of nodes explored and the number of iterations performed in solving the LPs. The data in the table show that each technique improves computational performance, and that improvement is more dramatic when the ILP is more difficult for CPLEX to solve. Individually, BT performed best with respect to CPU time, mainly because it removes unnecessary variables at each node of the B&C tree. In addition, the computational effort associated with the BT procedure is minimal.

The average number of variables for the 10 instances in J2013 is 872. The second column in Table 4.3 indicates that VR removed an average of 197 variables in about 48 seconds. Recall that VR also is designed to increase the LP lower bound. As can be seen in the table, LB_{LP} goes from 28.70 to 34.2 when the procedure is used.

Another observation from this phase of the testing is that local branching is

Table 4.3: Results for J2013 instances with different techniques

(a)

Components	LB_{LP}	t_{TOT}	t_{OPT}	t_{VR}	t_{BT}	t_{SD}
–	28.70	277.85	77.80	–	–	–
VR	34.20	183.97	67.20	47.98	–	–
BT	28.70	160.85	72.90	–	0.85	–
LB	28.70	219.05	15.70	–	–	4.87
VR + BT	34.20	142.36	79.00	48.09	0.66	–
VR + LB	34.20	161.25	38.70	48.03	–	4.14
BT + LB	28.70	118.32	11.70	–	0.64	4.75
VR + BT + LB_1	34.20	116.49	36.50	47.92	0.34	4.12
VR + BT + LB_2	34.20	423.24	36.70	48.00	3.38	263.11

(b)

Components	Variables removed	Nodes	LP iterations	N_{cover}	N_{clique}	N_{gub}
–	–	2,751	962,953	55	16	542
VR	197	1,741	481,638	58	18	472
BT	–	2,005	507,752	53	17	520
LB	–	2,086	820,614	–	–	–
VR + BT	197	1,680	324,613	59	17	439
VR + LB	197	1,274	441,977	–	–	–
BT + LB	–	1,562	426,399	–	–	–
VR + BT + LB_1	197	919	258,551	–	–	–
VR + BT + LB_2	197	8,456	1,666,649	–	–	–

most beneficial when the GA solution is not optimal. If we have a feasible solution that happens to be optimal, the task of B&C is only to prove that there is no better

solution. Local branching is not designed for this purpose and so provides little benefit. We tested two versions of local branching, one using the GA to uncover new reference solutions during strong diversification (denoted by VR + BT + LB_1 in Table 4.3), and the other using the CPLEX MIP solver (VR + BT + LB_2) to find reference solutions. The GA version proved superior, primarily because it requires much less time.

For the full testing, we attempted to solve all J20mm and J30mm instances with B&C using the VR + BT + LB_1 procedures. In general, a resource constrained project instance is characterized by several factors, such as problem size, resource availability and network complexity. The resource availability, indicated by the resource factor (RF) and the resource strength (RS), is closely related to the hardness of an instance. De Reyck and Herroelen (1996) studied the relations between the parameters of an instance and the hardness of the problem. It is shown that in general both very high and very low resource availability result an easier problem.

The benchmark data sets contain two levels for renewable resource factor RF_r (0.5, 1) and four levels for renewable resource strength RS_r (0.25, 0.5, 0.75, 1) (actual RS_r values deviate slightly from the target due to randomness). Renewable resources play a more critical role than non-renewable resources because the latter are only constrained in total usage while the former are constrained in each time period. We calculated the actual RF_r values and the average RS_r values from the data sets, and divided the projects into 8 groups, one for each RF-RS combination. The average solution properties for each group are reported in Tables 4.4, 4.5 and 4.6.

For the range of RS values attending the benchmark instances, problem difficulty increases as RS decreases, which is consistent with the finding of De Reyck and Herroelen (1996). For a particular resource k, a larger RF increases the density of the constraints while a smaller RS induces more resource constraints to be active in the LPs. Either situation increases the difficulty of a problem in the benchmark

data set. On the other hand, if the renewable RS is large enough, the earliest start time schedule provided by CPM will automatically satisfy the resource limits. Thus all corresponding constraints will be redundant, and hence, nonbinding. As the RS value decreases, the resource becomes more critical and effectively increases the number of resource constraints that are binding in each time period.

Generally speaking, when the resource strength is large, the feasible region is predominately shaped by the precedence relations. When the resource strength is small, activities are more likely to be scheduled in sequence than in parallel so the resource constraints will dominate.

The average results for the 20-job problems are summarized in Table 4.4. All 554 instances were solved optimally, most within 10 to 15 seconds except for those in group 2 ($RF_r = 1$ and $RS_r \in [0.2, 0.3]$). This group provided the biggest challenge, both for our method and the explicit B&B method of Sprecher and Drexl. The GA also had the most difficulty with this group. Larger RF values and smaller RS values imply lower resource availability, and hence longer makespans. This relationship can be observed in Table 4.4 where the difference between the optimal makespan and the CPM lower bound increases as the difficulty in solving a problem increases.

Table 4.5 and 4.6 summarize the average results for the 30-job instances for a CPU time limit of 3600 seconds. The maximum number of strong diversifications for these instances was adaptively set at $\lceil 3(RF_r/RS_r)^2 \rceil$. Again, the problems in group 2 were the most difficult to solve. Of the 552 instances, we optimally solved 506 and obtained the best known makespan in 529. In the remaining 23 instances, our solution was marginally inferior to the best available. Most of those were found with tabu search, but no accompanying solution times are reported in the literature (Nonobe and Ibaraki, 2002). In 5 instances, we found solutions with makespans smaller than the best known values. These were all optimal. It should be noted that none of the solutions identified up until this time were confirmed optimal except in a handful of cases in which the makespan was coincidentally equal to the CPM

Table 4.4: Test results for the 20-job instances

(a)

Group	RF_r	RS_r	N	LB_{CPM}	LB_{LP}	Optimal	GA gap
1	0.5	0.20 – 0.30	69	24.48	28.10	29.07	1.40
2	1	0.20 – 0.30	71	24.55	32.76	36.06	1.69
3	0.5	0.45 – 0.55	70	24.13	25.83	26.46	0.42
4	1	0.45 – 0.55	71	24.97	27.48	28.37	0.95
5	0.5	0.70 – 0.80	72	23.56	25.22	25.61	0.41
6	1	0.70 – 0.80	71	24.48	26.21	26.73	0.57
7	0.5	1	60	22.93	23.30	23.45	0.31
8	1	1	70	23.99	25.41	25.70	0.35
Total / average			554	24.12	26.83	27.71	0.79

(b)

Group	t_{TOT}	t_{GA}	t_{BB}	t_{OPT}	t_{VR}	t_{BT}	t_{SD}	Nodes	LP iterations
1	12.93	4.13	84.54	4.51	3.59	0.02	2.30	43	11,504
2	204.45	4.43	1,289.70	60.39	47.05	0.80	4.97	1,704	510,886
3	5.06	2.37	76.95	1.57	0.23	0.01	1.68	13	2,679
4	13.52	3.86	247.02	5.51	3.30	0.03	2.67	84	16,442
5	4.50	2.06	44.84	0.90	0.08	0.00	1.79	4	980
6	5.13	2.39	81.82	1.36	0.19	0.00	1.81	8	1,793
7	2.20	1.20	7.65	0.60	0.05	0.00	0.85	2	292
8	3.54	1.71	50.59	0.89	0.06	0.00	1.28	4	1,040
Average	32.06	2.78	238.68	9.69	6.97	0.11	2.18	238	69,849

bound. For comparative purposes, the last column in Table 4.6 gives the relative

performance of B&C. The values reported were calculated as follows.

$$\text{BC gap} = \frac{\text{B\&C solution } - UB_{BEST}}{UB_{BEST}} \times 100\%$$

A value of zero indicates that our approach attained the best known solution. If our best solution was better than the best known solution (UB_{BEST}), then the gap is negative.

Table 4.5: Test results for the 30-job instances, Part 1

Group	RF_r	RS_r	N	N_b	N_o	LB_{CPM}	LB_{LP}	LB_{BC}	UB_{BEST}	GA gap	BC gap
1	0.5	0.20 – 0.30	70	70	70	30.73	33.29	34.10	34.10	3.54	0.00
2	1	0.20 – 0.30	70	47	24	28.89	37.21	40.13	41.73	5.87	0.88
3	0.5	0.45 – 0.55	70	70	70	29.10	30.93	31.39	31.39	1.25	0.00
4	1	0.45 – 0.55	70	70	70	30.41	32.44	33.23	33.27	2.92	−0.11
5	0.5	0.70 – 0.80	70	70	70	30.64	32.07	32.40	32.40	0.99	0.00
6	1	0.70 – 0.80	71	71	71	30.75	32.01	32.48	32.49	1.57	−0.05
7	0.5	1	60	60	60	29.48	29.65	29.78	29.78	0.46	0.00
8	1	1	71	71	71	29.68	31.14	31.54	31.54	0.65	0.00
Total / average			552	529	506	29.97	32.39	33.19	33.40	2.18	0.09

Table 4.6: Test results for the 30-job instances, Part 2

Group	t_{TOT}	t_{GA}	t_{BEST}	t_{VR}	t_{BT}	t_{SD}	Nodes	LP iterations
1	100.61	5.90	56.95	11.83	0.48	10.99	552	202,992
2	2,852.02	6.66	827.43	324.56	16.56	64.38	18,910	6,713,912
3	8.64	2.82	5.39	0.57	0.02	3.17	26	6,953
4	108.11	4.76	77.62	15.31	0.65	15.86	1,046	221,994
5	6.08	2.55	3.73	0.23	0.01	2.47	10	1,814
6	14.65	2.44	11.12	0.63	0.08	5.88	110	19,745
7	2.32	1.70	1.76	0.13	0.00	0.37	2	282
8	7.68	2.54	3.69	0.24	0.01	3.77	14	2,586
Average	393.13	3.70	125.25	44.83	2.26	13.56	2621	909,310

4.4 Summary

In the past, exact methods for solving the MRCPSP have mainly focused on enumerating all possible activity sequences and mode options within a branch-and-bound framework. In this chapter, we study the ILP formulation of the MRCPSP and develop a branch-and-cut procedure for general linear objective functions and extended types of resource constraints. Based on the fact that precedence relations must be maintained in every feasible schedule, we derived a minimum distance matrix for each pair of precedence-related activities. When the objective is to minimize the makespan we then showed how these distances can be improved by using any lower bounding scheme for project completion time.

The primary contribution of the research centers on the development of several techniques for accelerating the solution process, including variable reduction, cut generation, and bound tightening. In addition, a high level search strategy referred to as local branching was applied to find good feasible solutions in the early stages of the computations. Empirical tests on benchmark problems showed that our approach gives markedly improved results with respect to any one implementation reported in literature. A primary advantage of the methodology is its flexibility in the selection of heuristics and lower bounding schemes at the subroutine level. For example, we used a genetic algorithm to find feasible solutions during local branching, but any heuristic would have sufficed. Another advantage is that it can be easily extended to problems that include general temporal constraints.

Like other time-indexed models, the number of variables and constraints in the integer programming formulation of the MRCRSP is proportional to the project's makespan. This is a primary concern, although the use of time-indexed binary variables is necessary when non-renewable resources are present. To deal with this concern, we proposed several techniques to reduce the number of variables and size of subproblems during branch-and-cut. For projects with relatively long makespans or activities whose finish time intervals are large, more efficient solution

methods are still needed. Time scaling to reduce the number of periods is one possible approach. For problems of realistic size, there is always a tradeoff between model tractability and solution quality.

Chapter 5

Schedule Recovery for Disrupted Resource-Constrained Projects

5.1 Introduction

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. (2000) 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.

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.

We also present an integer linear programming model for the recovery problem. In Section 5.3, we present a hybrid mixed-integer programming/constraint programming procedure to solve the general project schedule recovery problem. Numerical computations are performed to show the details of modeling and solution procedure.

5.2 Schedule Recovery Model

Figure 5.1 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, however, 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.

5.2.1 Representation of an Initial Schedule

We still consider resource-constrained project scheduling with finish-start precedence constraints. A project may have a number of milestones. Each milestone is associated with a set of activities. Its event time is defined as the earliest time that

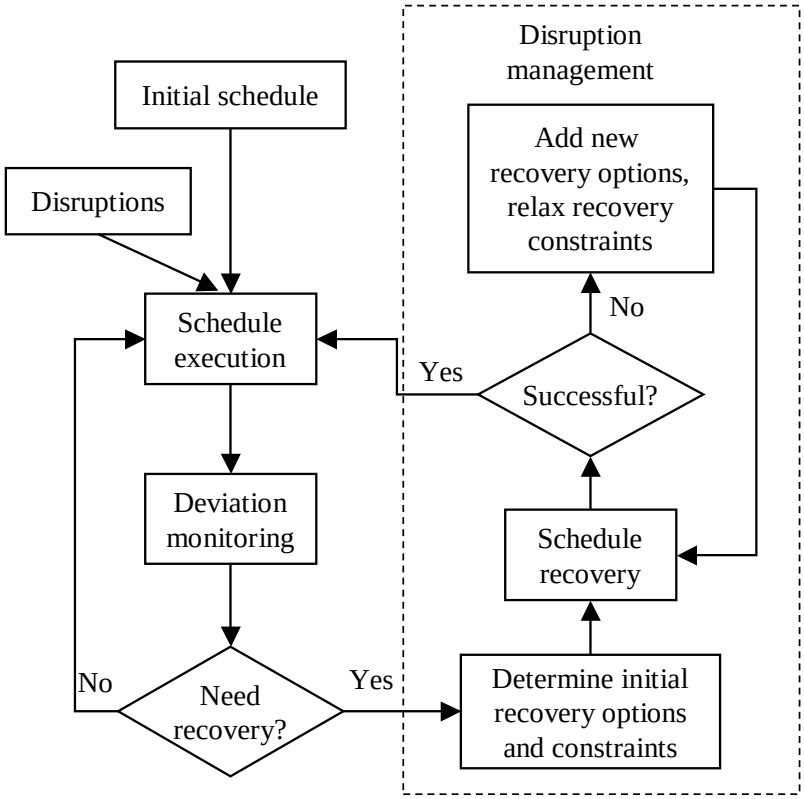

Figure 5.1: Disruption management for project scheduling

all such activities are completed. In addition to the real event time in a schedule, a milestone may also have a target time. 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.

In addition to the notation defined in Chapter 3, we introduce the following notation.

Indices and sets

$$
\begin{aligned}
\mathcal{T}_i &= \text{set of time periods at which activity } i \text{ can finish; } \mathcal{T}_i \subseteq \mathcal{T} \\
\Theta &= \text{set of milestones} \\
\theta &= \text{index for milestones; } \theta \in \Theta \\
\mathcal{B}(\theta) &= \text{set of activities whose completion defines milestone } \theta\text{; } \mathcal{B}(\theta) \subseteq \mathcal{A},\ \theta \in \Theta
\end{aligned}
$$

Schedule specifications

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

5.2.2 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 modeled 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 5.2.1. 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 5.2.2. 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 5.2.3. Activity duration disruption δ_i: Activity $i \in \mathcal{A}$ takes δ_i more time to complete than initially planned.

Definition 5.2.4. Activity resource disruption γ_{ik}: Activity $i \in \mathcal{A}$ uses γ_{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 5.2.5. Resource disruption ρ_{π_k}: Availability of resource k in time slot $\pi_k \in \Pi_k$ decreases by the amount of ρ_{π_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 5.2.6. Milestone disruption $\epsilon_\theta \in \mathbf{R}$: The target time of milestone θ changes from $\bar{t}_\theta$ to $\bar{t}_\theta + \epsilon_\theta$.

5.2.3 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 π_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

5.2.4 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 which is to be minimized.

$$
\begin{aligned}
Q(\mathbf{x},\mathbf{y},\mathbf{t}) &= \sum_{i,m,t} \alpha_{imt} x_{imt} && (5.1)\\
&\quad + \sum_{r} g(r) y_r + \sum_{i,m} c_{im} \sum_{t} x_{imt} && (5.2)\\
&\quad + \sum_{i} \left(\beta_i^1 \left[\sum_{m,t} t x_{imt} - \bar{f}_i \right]^+ + \beta_i^2 \left[\bar{f}_i - \sum_{m,t} t x_{imt} \right]^+ \right) && (5.3)\\
&\quad + \sum_{\theta} \left(\lambda_i^1 \left[t_\theta + \epsilon_\theta - \bar{t}_\theta \right]^+ + \lambda_i^2 \left[\bar{t}_\theta - t_\theta - \epsilon_\theta \right]^+ \right) && (5.4)
\end{aligned}
$$

The symbols α, β and λ are given weights and $[z]^+ = \max\{0, z\}$. The event time of milestone θ in the new schedule is defined as $t_\theta = \max\{\sum_m \sum_t t x_{imt} : i \in \mathcal{B}(\theta)\}$ in (5.4).

The first term (5.1) 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 (5.2). The penalty incurred for both positive and negative deviations from the original schedule is represented by term (5.3), while term (5.4) captures the penalty for milestone deviations. The weights α, β, and λ 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 (5.3) – (5.4) 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.

5.2.5 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

$$x_{im_0\bar{f}_i} = 1 \qquad \forall\, i \in \mathcal{A}_F \tag{5.5}$$

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

$$\sum_{m} \sum_{t_1 \le t \le t_2} x_{imt} = 1 \tag{5.6}$$

5.2.6 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.

$$\text{(RP) Minimize} \quad z = Q(\mathbf{x}, \mathbf{y}) \tag{5.7}$$

$$\text{subject to} \sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} x_{imt} = 1, \quad i \in \mathcal{A} \cup \mathcal{A}_N \tag{5.8}$$

$$\sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} t x_{imt} - \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}_i} (t - p_{jm}) x_{jmt} \le 0, \quad (i,j) \in \mathcal{P}_N \cup \mathcal{P}_A \cup (\mathcal{P} \setminus \mathcal{P}_R) \tag{5.9}$$

$$\sum_{t \in \pi_k} \sum_{i \in \mathcal{A} \cup \mathcal{A}_N} \sum_{m \in \mathcal{M}_i} \sum_{q=t}^{t+p_{im}-1} r_{imk} x_{imq} \le R_{\pi_k} - \rho_{\pi_k} + y_{\pi_k} R(\pi_k), \quad \pi_k \in \Pi_k, k \in \mathcal{K} \tag{5.10}$$

$$x_{im_0 \bar{f}_i} = 1, \quad \forall\ i \in \mathcal{A}_F \tag{5.11}$$

$$\sum_{m \in \mathcal{M}_i} \sum_{t_1 \le t \le t_2} x_{imt} = 1, \quad \text{for some } i \in \mathcal{A} \setminus \mathcal{A}_F \tag{5.12}$$

$$\sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{T}_i} t x_{imt} \le t_\theta, \quad \forall\ i \in \mathcal{B}(\theta),\ \theta \in \Theta \tag{5.13}$$

$$x_{imt} \in \{0,1\}, \quad \forall\ i \in \mathcal{A} \cup \mathcal{A}_N, m \in \mathcal{M}_i, t \in \mathcal{T}_i \tag{5.14}$$

$$y_{\pi_k} \in \{0,1\}, \quad \forall\ \pi_k \in \Pi_k, k \in \mathcal{K} \tag{5.15}$$

Equation (5.8) guarantees that each remaining activity has a unique finish time, while (5.9) and (5.10) respectively ensure that precedence and resource constraints are satisfied. Equation (5.11) guarantees that all activities outside the recovery window $[T_a, T_b]$ are executed as originally planned. Constraints (5.12) and (5.13) represent the recovery restrictions imposed on activity and milestone completion times, respectively.

Model (5.7) – (5.15) 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.

5.3 Hybrid MIP/CP Solution Approach

Mixed integer programming (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 (Baptiste et al., 2001). It works by performing constraint propagation (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.

5.3.1 Procedure

In designing our hybrid algorithm (Rodoseěk et al., 1999), we use a branch-and-cut strategy to construct a search tree and to tighten the LP relaxation of (5.7) – (5.15) at each node. In addition, constraint propagation is performed to remove dominated variables. Some techniques discussed in Chapter 4, such as cuts and SOS branching schemes, are also used in this procedure. Related work on hybrid modeling and constraint classification can be found in Bockmayr and Kasper (1998) and Jain and Grossmann (2001).

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) [i.e., Eqs. (5.8) and (5.12)], we perform constraint propagation immediately before new nodes are created in the tree rather than before the LP relaxation is solved.

Algorithm 5.3.1. 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 Ω 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 Ω.

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

begin

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

Re-solve $LP(\omega)$ if new cuts are added. Let z^ω 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) = **call ConstraintPropagation**(ω)

Create the first node by setting variables in S_1 to zero.

Create the second node by setting 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 Ω.

end

end

Report optimal solution z^* or "infeasible."

end

Algorithm 5.3.1 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 (ILOG, 2002) for implementing the constraint propagation routine. Because of the way CPLEX works, constraint propagation 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 constraint propagation.

5.3.2 Constraint Propagation

The purpose of constraint propagation 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 (i.e., 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 (5.7) – (5.15), variables associated with activity i are defined on the subset $\mathcal{T}_i = \{e_i, \ldots, l_i\}$, 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 $[e_i, l_i]$. Due to precedence constraints, any change in the finish time window of one activity may affect the windows of both its predecessors and successors.

We still use the $(n+1) \times (n+1)$ distance matrix $D \equiv (d_{ij})$ defined in Section 4.2. Similar to Proposition 4.2.1, we now define what we meant to be the consistency of precedence constraints at a node in the search tree.

Definition 5.3.1. 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 (e.g., Bartusch et al., 1988).

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 constraint propagation procedure are presented in Algorithm 5.3.2. Constraint propagation 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 5.3.2. CONSTRAINT PROPAGATION

input : Node in search tree and correspondig 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

$p^{\min} = \{p_i^{\min}\}$, 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**

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

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

$D_1(i_b, n + 1) = T - \lceil t_b \rceil$

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

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

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 (5.3) 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 5.3.3 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 5.3.3 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$(\hat{D}, p^{\min})$

$LB = \sum_{i \in \mathcal{A}} \left(\beta_i^1 \left[\hat{d}_{0i} + p_i^{\min} - \bar{f}_i \right]^+ + \beta_i^2 \left[\bar{f}_i - (T - \hat{d}_{i,n+1}) \right]^+ \right)$

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$(\hat{D}, p^{\min})$

$LB = \sum_{i \in \mathcal{A}} \left(\beta_i^1 \left[\hat{d}_{0i} + p_i^{\min} - \bar{f}_i \right]^+ + \beta_i^2 \left[\bar{f}_i - (T - \hat{d}_{i,n+1}) \right]^+ \right)$

if $LB < \bar{z}$ **then**

$l_i = t$, $d_{i,n+1} = T - t + 1$; **break** "for" loop

end

end

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

for activity i is 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

5.4 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 5.2 along with activity durations and resource requirements. In each period, 10 units of the resource are available. Figure 5.3 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.

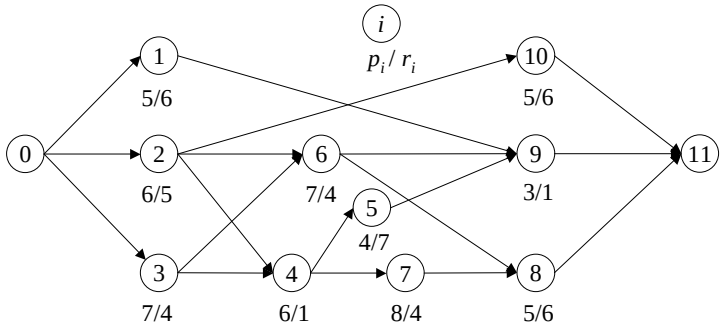

Figure 5.2: Project network

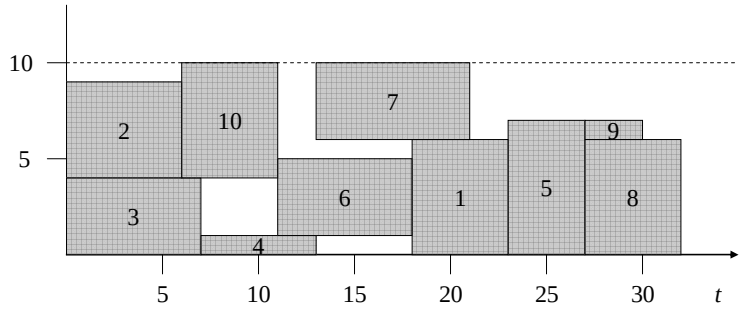

Figure 5.3: Original schedule

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 5.4. 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 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

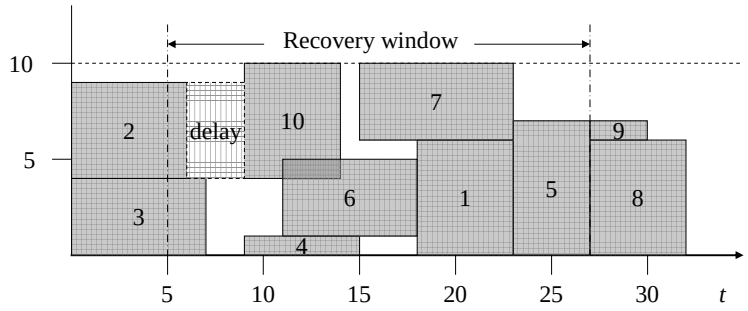

Figure 5.4: Disrupted schedule

Figure 5.4) 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 5.1 lists the alternative modes for activities $1, 4, 5, 6, 7$ in addition to the original mode, denoted as mode 1.

To illustrate constraint propagation, 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 5.2. 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 : $[14, 27]$.

We now discuss three possible cases for which constraint propagation 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 $[18, 27]$

Table 5.1: Mode alternatives for recovery options

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

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

Table 5.2: 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

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 $10 : [14, 21]$.

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 β [that is,

$\beta_i^1 = \beta_i^2 = \beta$ in Eq. (5.3)]. The objective, which only includes the second term in Eq. (5.2) plus Eq. (5.3), can be written as

$$\begin{aligned} Q &= \sum_{i\in\mathcal{A}, m\in\mathcal{M}_i} c_{im} \sum_{e_i\le t\le l_i} x_{imt} \\ &+ \sum_{i\in\mathcal{A}} \beta \left(\left[\sum_{m\in\mathcal{M}_i, e_i\le t\le l_i} t x_{imt} - \bar{f}_i \right]^+ + \left[\bar{f}_i - \sum_{m\in\mathcal{M}_i, e_i\le t\le l_i} t x_{imt} \right]^+ \right) \end{aligned}$$

where mode penalties c_{im} are listed in Table 5.1.

Table 5.3 shows the recovered schedules for $\beta = 0$ and $\beta = 3$. 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.

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 Eq. (5.3). To account for the relative importance of finish time deviations for each activity, different penalty coefficients can be assigned to the corresponding binary variables.

5.5 Computational Results

The hybrid MIP/CP procedure was tested on the 554 20-activity benchmark problems developed by Kolisch et al. (1995) .Each instance has 20 activities, 2 renewable and 2 non-renewable resources, and each activity has 3 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.

Table 5.3: Recovered schedules

		$\beta = 0$		$\beta = 3$	
Activity	Target finish time	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	

Disruption scenario and recovery setup

1. The duration of activity 3 was extended by 3 time periods (Definition 5.2.3).

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 (i.e., only Eq. (5.3) 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 Hartmann (1999). 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, 8 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. (1995) 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 1.8 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^{PRE}	=	number of instances for which the infeasibility is detected by precedence constraints
N_I^{TOT}	=	number of infeasible instances
t_{GA}	=	CPU time for genetic algorithm
N_{VAR}	=	number of variables in the recovery problem

N_{CON}	=	number of constraints in the recovery problem
LB_{LP}	=	LP bound at the root node of search tree
t_{TOT}	=	total CPU time
t_{BEST}	=	CPU time of getting the optimal solution or proving infeasibility
t_{CP}	=	CPU time for constraint propagation
N_{ITER}	=	number of LP iterations
N_{GUB}	=	number of GUB cover cuts applied
N_{PRE}	=	number of precedence cuts applied

5.5.1 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 5.4 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_{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 5.5 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 sec to 45.02 sec, 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. Con-

Table 5.4: Characteristic of recovery problems by group

Group	N	RF_r	RS_r	N_I^{PRE}	N_I^{TOT}	t_{GA}	N_{CON}	N_{VAR}	LB_{LP}	z^*
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	71	1	0.7 – 0.8	25	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

Table 5.5: Comparative results of MILP/CP and CPLEX alone

	MIP/CP procedure							CPLEX alone					
Group	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	2,457	38	26	4.09	1.46	14	4,004	37	28
3	5.08	2.37	0.012	38	6,533	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	1,306	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

vergence occurs almost immediately after the best solution is found. In all cases, the computational effort associated with CP is 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 5.4 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 5.4 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 (i.e., 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.

5.5.2 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 5.6, 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 5.6 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 quantity 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, μ 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 5.6 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.

Table 5.6: Recovery result for different initial schedules

	Initial schedules				Recovery results			
Case	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

5.5.3 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 5.7. 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, computation times increase rapidly at first and then taper off.

Table 5.7: 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

5.5.4 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, $t_{ET} = 0$; i.e., 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 5.8 summarizes the results for $t_{ET} = 0, 1, \ldots, 10$. 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 5.8: 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

5.6 Summary

Disruptions due to such factors as resource shortages, technical difficulties, and loss of personnel are an unavoidable part of any project. In this chapter, 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 constraint propagation 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. Though 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 case discussed herein.

Chapter 6

Project Planning with Uncertain Activity Durations

6.1 Introduction

This chapter studies the problem of how to set the target finish times (due dates) of project activities with uncertain durations. A two-stage decision model is developed for the analysis. In the first stage, target activity times are fixed based on known probabilistic information. After all uncertainty is resolved, a detailed schedule is obtained as a function of the proposed target times. The model is designed to provide a solution that balances (1) the cost associated with target times and (2) the expected cost of deviating from the original plan.

The objective of the first stage is to minimize the sum of weighted target times. The objective of the second stage is to obtain a schedule that has minimum penalty cost associated with deviations from target times (and not one whose makespan is necessarily minimized). Because of the underlying dynamics, the first stage decisions may impose additional constraints on the second stage scheduling problem. For example, an activity may be required to be completed within a certain time interval centered at its target completion time. In fact, the second stage scheduling problem is a special case of schedule recovery in Chapter 5. In this context, the target times obtained in the first stage can be viewed as the *initial* plan. This paper extends the ideas of schedule recovery by considering not only the best recovery strategy for each possible scenario, but also developing baseline schedule (initial plan) that optimizes expected performance under disruptions.

By developing a stochastic optimization model and investigating several special cases, we have been able to gain a deeper understanding of the decision making process and offer a range of managerial insights. In the model, uncertainty is represented by discrete scenarios. We consider the situation in which there is uncertainty between the planning and implementation stages, but few disruptions once implementation begins. The implication of this assumption is that we have a good idea about the probability distribution of activity durations. This allows us to use a two-stage decision model as a basis for analysis.

In Section 6.2, we present the two-stage stochastic linear programming model and discuss some of its properties. Section 6.3 highlights the solution methods. To begin, we solve the LP relaxation of the stochastic integer problem to obtain a lower bound on the optimal objective function value. For the case without a budget constraint, we show that the relaxed linear programming (LP) solution is indeed optimal. For the more general case, an LP-based heuristic is presented and shown to find good feasible solutions. We also show that in the absence of a budget constraint, the second stage subproblem can be transformed into a minimum cost network flow problem, and therefore, efficiently embedded in a stochastic programming algorithm. A detailed example is given in Section 6.4 along with our computational results for a standard set of benchmark projects.

6.2 Model Formulation and Properties

We consider a project with finish-to-start precedence constraints and uncertainty in durations of some activities. We represent the uncertainty as discrete scenarios. The primary reason for this is that we are mainly interested in the effects of uncertainty caused by discrete events (e.g., rework) in projects. Moreover, in reality, some continuous distributions for project parameters are often obtained from experimental observations, which are discrete. Uncertainty with continuous distributions may be approximated with discrete scenarios through sampling. An

important issue in this situation is to control the approximation error while limiting the number of scenarios.

Because the uncertainty undermines our ability to obtain a schedule with deterministic methods, we take a two-stage approach. In the first stage, the goal is to set the target finish times for activities before their durations are actually known. In the second stage, a deterministic scheduling problem is solved with the objective of minimizing the deviation of actual finish times from the predetermined target times, subject to a budget constraint when crashing opportunities exist. The objective of the full problem is to minimize the expected value of a combined measure that considers the target finish times, deviation penalties, and applicable crashing costs.

If milestones associated with multiple activities are considered, we also add dummy activities to the project as in Chapter 5. The target time for a milestone can be treated as the target time for the corresponding dummy activity.

In the development of the model, activity durations are assumed to be multiples of the basic time unit so all scheduling decisions are required to be positive integer values. In addition to notation defined in Chapter 3, the following notation is used to describe the model.

Indices, sets, and data

Ω = $\{\omega_1, \omega_2, \ldots, \omega_S\}$, the set of discrete future scenarios; scenario ω_s occurs with a positive probability $\pi_s = \text{Prob}(\omega = \omega_s)$, and $\sum_{\omega \in \Omega} \pi_s = 1$

$p_i(\omega)$ = processing time (duration) of activity i for scenario $\omega \in \Omega$, $p_0(\omega) = 0,\ p_{n+1}(\omega) = 0$

c_i = cost per unit time associated with the target completion time of activity i

d_i = unit cost of crashing for activity i

q_i^+, q_i^- = penalty cost coefficients associated with the late, early completion of activity i

B = total crashing budget

Variables

t_i = target completion time for activity i; $t_i \in \mathbb{Z}_+$.

x_i = crashing variable for activity i; $x_i \in [0, u_i] \cap \mathbb{Z}_+$

y_i^+, y_i^- = number of time units that activity i is late, early with respect to its target completion time; $y_i^+, y_i^- \in \mathbb{Z}_+$

Note that the decision variables $x_i,\ y_i^+,\ y_i^-$ are all scenario-related and should really be written as a function of ω; that is, $x_i(\omega), y_i^+(\omega), y_i^-(\omega)$. For simplicity, though, this dependency is omitted. We now let $\mathbf{t},\ \mathbf{p},\ \mathbf{c},\ \mathbf{d},\ \mathbf{q}^+,\ \mathbf{q}^-,\ \mathbf{u},\ \mathbf{x},\ \mathbf{y}^+,\ \mathbf{y}^-$ be the column vectors of $t_i,\ p_i,\ c_i,\ d_i,\ q_i^+,\ q_i^-,\ u_i,\ x_i,\ y_i^+,\ y_i^-, i = 0, \ldots, n+1$, respectively, and denote the $m \times (n+2)$ incident matrix of precedence relations by $\mathbf{P}$, where $P_{ki} = -1,\ P_{kj} = 1$ for the kth precedence constraint (i, j) and all other elements in the kth row of $\mathbf{P}$ are 0.

The first stage problem for minimizing the expected cost of the project is

$$\text{(SP)}\ \min_{\mathbf{t}} \left\{ \mathbf{ct} + Q(\mathbf{t}) : \mathbf{t} \in \mathbb{Z}_+^{n+2} \right\} \tag{6.1}$$

where the optimal value function of the second stage is

$$Q(\mathbf{t}) = \sum_{s=1}^{S} \pi_s v(\mathbf{t}, \omega_s) \tag{6.2}$$

The first term in (6.1) represents the weighted sum of completion times. It can be used to specify the relative importance among the project's milestones including the makespan. For each scenario ω_s, $v(\mathbf{t}, \omega_s)$ in Equation (6.2) is the optimal objective function value of the second stage scheduling problem.

The cost of a project is often related to how closely target activity completion times are met. An early completion may indicate an overuse of resources, while a delay may trigger some form of penalty. These factors have provided the motivation for including deviation costs in our second stage objective function. The formulation given below can be viewed as a special case of the schedule recovery problem in Chapter 5.

For given target completion times $\mathbf{t}$, we have following integer linear programming model for minimizing the total deviation penalty.

$$\text{Minimize} \quad \sum_{i \in \mathcal{A}} \left(q_i^- y_i^- + q_i^+ y_i^+ \right) \tag{6.3}$$

$$\text{subject to} \quad t_j + y_j^+ - y_j^- - t_i - y_i^+ + y_i^- \geq p_j - x_j, \quad \forall\ (i,j) \in \mathcal{P} \tag{6.4}$$

$$\sum_{i \in \mathcal{A}} d_i x_i \leq B \tag{6.5}$$

$$x_i \in [0, u_i], \quad \forall\ i \in \mathcal{A} \tag{6.6}$$

$$x_i, y_i^+, y_i^- \in \mathbb{Z}_+, \quad \forall\ i \in \mathcal{A}; \quad y_0^+ = y_0^- = 0 \tag{6.7}$$

Using the vector notation defined above, model (6.3) – (6.7) can be written as

$$\text{Minimize} \quad \mathbf{q}^- \mathbf{y}^- + \mathbf{q}^+ \mathbf{y}^+ \tag{6.8}$$

$$\text{subject to} \quad \mathbf{P}\mathbf{y}^+ - \mathbf{P}\mathbf{y}^- + \mathbf{S}\mathbf{x} \geq \mathbf{p} - \mathbf{P}\mathbf{t} \tag{6.9}$$

$$\mathbf{d}\mathbf{x} \leq B \tag{6.10}$$

$$\mathbf{0} \leq \mathbf{x} \leq \mathbf{u} \tag{6.11}$$

$$\mathbf{x}, \mathbf{y}^+, \mathbf{y}^- \in \mathbb{Z}_+^{n+2},\ y_0^+ = y_0^- = 0 \tag{6.12}$$

where $\mathbf{S}$ is an $m \times (n+2)$ matrix such that if the kth precedence constraint is (i, j), then $S_{kj} = 1$ and all other elements in the kth row of $\mathbf{S}$ are 0.

The objective in (6.8) is to minimize the total penalty incurred by deviating from the target times. Precedence constraint (6.9) guarantees a feasible schedule and (6.10) is the budget constraint. Note that the time for dummy activity 0, which indicates the beginning of the project, is not allowed to change so the corresponding variables y_0^+ and y_0^- could be removed from the formulation. However, we keep them to avoid having to redefine the rows of the precedence matrix.

If crashing costs are not budget constrained but are viewed as a penalty, then they can be added to the objective function. This leads to the following LP model for the second stage problem

$$\text{Minimize} \quad \mathbf{q}^-\mathbf{y}^- + \mathbf{q}^+\mathbf{y}^+ + \mathbf{dx} \tag{6.13}$$

$$\text{subject to} \quad \mathbf{P}\mathbf{y}^+ - \mathbf{P}\mathbf{y}^- + \mathbf{S}\mathbf{x} \geq \mathbf{p} - \mathbf{P}\mathbf{t} \tag{6.14}$$

$$-\mathbf{x} \geq -\mathbf{u} \tag{6.15}$$

$$\mathbf{x}, \mathbf{y}^+, \mathbf{y}^- \in \mathbb{Z}_+^{n+2},\ y_0^+ = y_0^- = 0 \tag{6.16}$$

where the weights $\mathbf{q}^-$ and $\mathbf{q}^+$ must be set to balance the costs $\mathbf{d}$.

One important question is how to estimate the parameters in the model. Although projects, for the most part, are unique, there are some general guidelines for doing this. The target time costs, c_i, depend on the requirements of the client. In a bidding environment, the proposed completion times may affect the probability of winning the project. After the project is awarded, delays may have serious consequences for the stakeholders, as may completing activities earlier than planned. This is especially true when just-in-time scheduling concepts are used. When it is desirable to encourage earliness, the corresponding coefficients can be set to negative values.

Estimating the penalty coefficients requires not only full contract information, but also some understanding of the costs associated with internal schedule disruptions. In contrast, crashing costs are generally related to overtime, expediting, and possibly, outsourcing fees.

We now discuss some properties of the LP relaxation of the stochastic integer programming model (6.1).

Existence of a bounded solution. Model (6.8) – (6.12) shows that for each scenario, the realized durations $\mathbf{p}(\omega_s)$ appear on the right-hand side of constraint (6.9) in the linear program. Therefore, the second stage optimal objective function can be viewed as a function of $\mathbf{p}(\omega_s) - \mathbf{Pt}$ and (6.2) can rewrite as

$$Q(\mathbf{t}) = \sum_{s=1}^{S} \pi_s v(\mathbf{p}(\omega_s) - \mathbf{Pt}) \tag{6.17}$$

where

$$v(\mathbf{z}) = \min \left\{ \mathbf{q}^- \mathbf{y}^- + \mathbf{q}^+ \mathbf{y}^+ : \mathbf{P}\mathbf{y}^+ - \mathbf{P}\mathbf{y}^- + \mathbf{S}\mathbf{x} \geq \mathbf{z}, (6.10) - (6.12) \right\} \tag{6.18}$$

The following proposition establishes the conditions under which $v(\mathbf{z})$ is bounded for any target times $\mathbf{t}$ and any realization of uncertainty ω.

Proposition 6.2.1. *Recourse value function (6.18) satisfies the following:*

1) $v(\mathbf{z}) < \infty$, $\forall\ \mathbf{z} \in \mathbb{R}^m$

2) $v(\mathbf{z}) > -\infty$, $\forall\ \mathbf{z} \in \mathbb{R}^m$ *if* $\mathbf{q}^+ + \mathbf{q}^- \geq \mathbf{0}$

PROOF. Problem (6.18) is always feasible for $\mathbf{z} \in \mathbb{R}^m$ as long as $B \geq 0$. Therefore $v(\mathbf{z}) < \infty, \forall\ \mathbf{z} \in \mathbb{R}^m$.

Note that because crashing costs are always assumed to be non-negative, whether (6.18) has an unbounded optimal solution solution is not affected by the

budget constraint. For any $\mathbf{z} \in \mathbb{R}^m$, the feasible region of the dual problem of the LP relaxation of (6.18) without the budget constraint can be written as

$$\left\{\boldsymbol{\lambda} \in \mathbb{R}_+^m : \boldsymbol{\lambda}\mathbf{P} \leq \mathbf{q}^+, -\boldsymbol{\lambda}\mathbf{P} \leq \mathbf{q}^-\right\}.$$

If $\mathbf{q}^+ + \mathbf{q}^- \geq \mathbf{0}$, then

$$\left\{\boldsymbol{\lambda} \in \mathbb{R}_+^m : \boldsymbol{\lambda}\mathbf{P} \leq \mathbf{q}^+, -\boldsymbol{\lambda}\mathbf{P} \leq \mathbf{q}^-\right\} \neq \emptyset,$$

and hence (6.18) has a bounded optimal solution, i.e., $-\infty < v(\mathbf{z})$. □

Given that $\mathbf{z}$ appears on the right-hand side of linear programming constraints, we know that $v(\mathbf{z})$ is a convex function of $\mathbf{z}$, or $\mathbf{t}$. However, this does not guarantee a bounded solution for SP. For example, if $c_i + q_i^- < 0$ for activity i, then it is optimal to set the target time t_i as late as possible and schedule the activity as early as possible. If $\mathbf{c} + \mathbf{q}^- < \mathbf{0}$, it is optimal to set all the target times as late as possible (∞). Consequently, we have

Corollary 6.2.2. *SP (6.1) has a bounded optimal solution if* $\mathbf{c} + \mathbf{q}^- \geq \mathbf{0}$ *and* $\mathbf{q}^+ + \mathbf{q}^- \geq \mathbf{0}$.

One trivial case of a bounded solution occurs when $\mathbf{q}^+ \leq \mathbf{c}$ and $\mathbf{q}^- \geq \mathbf{0}$, which means that the lateness penalty is no larger than the target time costs. As a result, it is always optimal to set the target times as 0.

Analogy to newsvendor problem. The stochastic programming problem SP balances the cost of target time excess with the target time shortage penalty. By 'target time excess' we mean the increase in the objective function value (expected cost of the full problem) per unit time when the realized completion times in the early schedule are less than the target times. By 'time shortage penalty' we mean the increase in the objective function value per unit time when the realized completion times in the early schedule are greater than the target times.

This is similar to the tradeoff between the excess and shortage costs in the traditional newsvendor problem (Porteus, 1990). In the newsvendor problem, a

vendor needs to make a decision on the purchasing quantity (at a cost of c per unit) of newspapers before the realization of an uncertain demand $x \geq 0$, which is characterized by a cumulative demand function (CDF) $F(x), x \in \mathbb{R}^+$. After the demand is realized, the vendor sells as many newspapers as possible at a price of p per unit (assuming $p \geq c$). If the realized demand is larger than the quantity she has purchased, she sells all the news paper she has purchased. On the other hand, if the realized demand is less than the quantity she has purchased, she sells newspaper at price p up to the realized demand, and recycles any extra newspapers at salvage price s per unit (assuming $s \leq c$). Let $c_s = p - c$ be the cost of each unit shortage, and $c_e = c - s$ be the cost of each unit excess. It is easy to derive that the vendor's optimal purchasing quantity quantity x^* is determined by

$$F(x^*) = \frac{c_s}{c_s + c_e}$$

Our problem is much more complicated, though, due to the presence of multiple dimensions and precedence constraints. For a project with only one activity (three activities if two dummies are considered), we now show that SP without crashing is equivalent to a traditional newsvendor problem. Suppose that a 1-activity problem has cost coefficients q^+, q^- and c, and that its duration p has a CDF $F(p)$.

Proposition 6.2.3. *A 1-activity version of SP without crashing is equivalent to a newsvendor problem with unit excess cost c_e and unit shortage cost c_s, such that if $q^- \leq 0$, then $c_s = q^+ - c,\ c_e = q^- + c$ and if $q^- \geq 0$, then $c_s = q^+ - c,\ c_e = c$. In addition , the optimal target time t^* is given by $F(t^*) = \frac{c_s}{c_s + c_e}$.*

PROOF. The solution with perfect information is to set $t = p$. If $q^- \leq 0$, then for any target time t, it is optimal to schedule the activity as early as possible. For the case when $t > p$ (excess), the unit excess cost is $c + q^-$ (i.e., the unit target time cost plus the bonus of being early by one time unit); if $t < p$ (shortage), the unit

shortage cost is $q^+ - c$ (i.e., the penalty for being late by one time unit minus the unit target time cost).

If $q^- \geq 0$, then it is optimal to schedule the completion time of the activity as close as possible to the target time t. In this situation, if $t > p$ (excess), it is optimal to schedule the activity to finish at t since earliness is penalized. Thus the unit excess cost is just c. On the other hand, if $t < p$, the unit shortage cost is still $q^+ - c$.

The optimal value of t follows from the solution of the traditional newsvendor problem. □

In the newsvendor problem, it is assumed that $c_e,\ c_s > 0$, which implies that $-q^- < c < q^+$ for a 1-activity project. Extending the condition that product cost is between the salvage value and retail price for this problem, we have that the cost of a predetermined target time is no less than the earliness bonus and no greater than the lateness penalty in the recovery stage of the project. We shall assume that this condition holds for every activity, i.e., $-\mathbf{q}^- < \mathbf{c} < \mathbf{q}^+$.

Bounds for target times. In the newsvendor problem with $c_e,\ c_s > 0$, the optimal order quantity is equal to the demand for the case with perfect information, and is between the largest and smallest demand for the stochastic case. We now consider SP without crashing and with parameters $-\mathbf{q}^- < \mathbf{c} < \mathbf{q}^+$. We call this problem SP1, which can be viewed as a multi-dimensional newsvendor problem with precedence constraint. To bound the target times of SP1 in the general case, we first define the early schedule.

Definition 6.2.1. Early schedule $ES(\mathbf{p})$. Schedule obtained by starting all activities as early as possible, where the activities are assigned deterministic durations $\mathbf{p}$.

A special case of SP1 occurs when $\mathbf{q}^- < \mathbf{0}$, implying that earliness is always rewarded. Therefore, early schedules are optimal for all scenarios and SP1 decomposes into individual newsvendor problems, one for each activity. In the newsvendor problem for activity i, the stochastic demand corresponds to the completion time of activity i and the demand distribution has the discrete form $(f_i(\omega_s), \pi_s),\ s = 1, \ldots, S$, where $f_i(\omega_s)$ is the completion time of activity i in the early schedule $ES(\mathbf{p}(\omega_s))$ under scenario ω_s. The corresponding unit excess and shortage costs are $c_s = q_i^+ - c_i$ and $c_e = q_i^- + c_i$, respectively.

For SP1 with a certain realization of durations $\mathbf{p}(\omega_s)$, it is obvious that the optimal solution is given by the early schedule $ES(\mathbf{p}(\omega_s))$. Let $\underline{t}_i$ and $\bar{t}_i$ be the earliest and the latest completion time of activity i in $\{ES(\mathbf{p}(\omega)) : \omega \in \Omega\}$. Let $\bar{p}_i = \max\{p_i(\omega) : \omega \in \Omega\}$ and $\hat{t}_i$ be completion time for activity i in early schedule $ES(\bar{\mathbf{p}})$. We have the following proposition.

Proposition 6.2.4. *For SP1, the optimal target time t_i^* for activity i satisfies $\underline{t}_i \leq t_i^* \leq \hat{t}_i$.*

PROOF. For arbitrary target times $\mathbf{t}$, the actual completion time for activity i is no less than $\underline{t}_i$ for any scenario. Therefore, if t_i is set less than $\underline{t}_i$, it is always possible to increase t_i to $\underline{t}_i$ and decrease the objective by $(\underline{t}_i - t_i) \times (q_i^+ - c)$. Thus, $t_i^* \geq \underline{t}_i$.

We prove the upper bound by induction. Suppose that we have a project with one activity. Because $-q^- < c < q^+$, from the analogy to newsvendor problem, the optimal target time for this activity is less than the longest possible duration $\bar{p}$. Also, the actual completion of this activity will never be greater than $\bar{p}$. Now consider a partial project such that if it includes activity j, then all predecessors of j are also included. Suppose that the target times for all activities in the partial project satisfy the upper bound condition, and the actual completion time satisfies $t_i \leq \hat{t}_i$ for any activity i already in the partial project.

Next we add an activity k to the partial project as a successor of some of the existing activities. Note that adding this activity will not increase the optimal target times of those activities that are already included. If the target time t_k is set larger than $\hat{t}_k$, we could always decrease t_k to $\hat{t}_k$ and decrease the objective by $(t_k - \hat{t}_k) \times (q_k^- + c)$. Therefore, $t_i^* \leq \hat{t}_i$. □

By definition, $\hat{t}_i \geq \bar{t}_i$ for all i. If activity durations are mutually independent random variables, then there is one scenario for each combination of possible durations and the completion times in one of the early schedules are equal to their upper bounds; i.e., $\hat{t}_i = \bar{t}_i$ for all i. This follows because there will be one scenario in which all activity durations take their largest values. It is this scenario that determines $\hat{t}_i$.

Note that in some situations we could have $t_i^* > \bar{t}_i$, which means that the optimal target time for activity i is greater than its completion time in any scenario. This is caused by the deviation penalties in the scheduling stage, as illustrated in the following example.

Example. Suppose that a project consists of two activities, 1 and 2 (activity 1 precedes 2), and that there are two scenarios in total, each occurring with probability 0.5. Let activity durations be $p_1 = 1,\ p_2 = 2$ for scenario 1 and $p_1 = 2,\ p_2 = 1$ for scenario 2. Moreover, let the target time costs be $c_1 = c_2 = 1$ and the unit earliness and lateness penalties be $q_i^+ = q_i^- = 5,\ i = 1, 2$. The optimal solution for this problem is $t_1^* = 2,\ t_2^* = 4$.

Figure 6.1 depicts the early schedules for the two scenarios as well as the actual schedules under optimal target times. The optimal target time for activity 2 is 4, but as we see, activity 2 always finishes before 3 in the early schedules for both scenarios. The intuition behind this is the following. In order to avoid a lateness penalty for activity 1, its target time is set to 2, its maximum possible duration. In scenario 1, to avoid an earliness penalty, it is optimal to schedule activity 1 to finish at time 2 even though its realized duration is only 1. As a consequence, activity 2

finishes at time 4 so it is optimal to set the target time of activity 2 to 4 to avoid a lateness penalty.

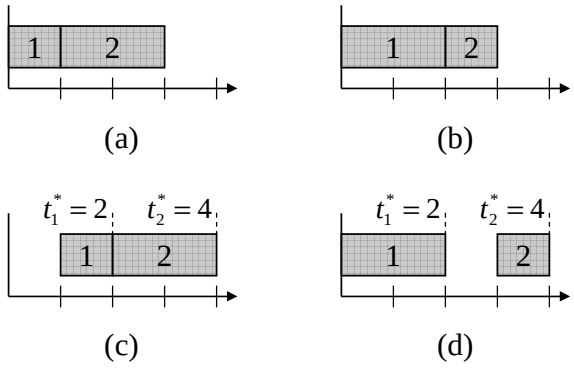

Figure 6.1: Schedules: (a) early schedule for scenario 1; (b) early schedule for scenario 2; (c) schedule with target times for scenario 1; (d) schedule with target times for scenario 2

6.3 Solution Approaches

The L-shaped or dual decomposition method is a well-known solution approach for two-stage stochastic linear programming problems. The idea is to approximate the nonlinear convex objective function and feasible region with linear inequalities. The approach iteratively solves a master problem and one second stage problem for each scenario. Depending on the solution status of the second stage problems, either feasibility cuts are generated to reduce the feasible region of the master problem or optimality cuts are generated to cut off non-optimal solutions of the master problem. Theory guarantees finite convergence (see Birge, 1997).

In our case, we wish to solve a two-stage stochastic integer linear program for which no efficient solution approaches exist (Klein Haneveld and van der Vlerk, 1999). Because of the nonconvexity of the second stage objective function, the L-shaped method is no longer applicable. Although SP can be reformulated as a single integer linear program (ILP) with a dual block angular structure, its size increases

rapidly with the introduction of more activities and scenarios.

Nevertheless, the L-shaped method can still be used to solve the LP relaxation of SP. This will provide us with a lower bound on the optimal objective function value as well as a starting point for constructing feasible integer solutions.

For the case without a budget constraint in the second stage problem, we will show that the solutions obtained from the LP relaxation are indeed integral so the L-shaped method can be used. We will also show that in this case, the second stage scheduling problem can be solved as a minimum cost network flow problem, which can be efficiently embedded in the L-shaped method. For the case with a budget constraint, we derive a feasible solution based on the solution to the LP relaxation and then show how it can be improved by solving a related non-budget constrained stochastic programming problem.

6.3.1 Integer Solutions

Without budget constraint. For this case, we have the following proposition.

Proposition 6.3.1. *If all activity durations are integer, then solving the LP relaxation of SP with the L-shaped method provides integer optimal target finish times,* $\mathbf{t^*}$*, and integer second stage decisions,* $\mathbf{x^*}$ *and* $\mathbf{y^*}$*, for all scenarios.*

PROOF. The stochastic integer programming problem (6.1) with subproblems in the form of (6.13) – (6.16) can be written as a single ILP with variables

$$\left(\mathbf{t}, \left[\mathbf{y}^+(\omega_\mathbf{1}), \mathbf{y}^-(\omega_\mathbf{1}), \mathbf{x}(\omega_\mathbf{1})\right], \ldots, \left[\mathbf{y}^+(\omega_\mathbf{S}), \mathbf{y}^-(\omega_\mathbf{S}), \mathbf{x}(\omega_\mathbf{S})\right]\right)^T$$

and constraint matrix

$$\mathbf{A} = \left(\begin{array}{c:cccccccccc} \mathbf{P} & \mathbf{P} & -\mathbf{P} & \mathbf{S} & & & & & & & \\ \mathbf{P} & & & & \mathbf{P} & -\mathbf{P} & \mathbf{S} & & & & \\ \vdots & & & & & & & \cdots & & & \\ \mathbf{P} & & & & & & & & \mathbf{P} & -\mathbf{P} & \mathbf{S} \end{array}\right) \tag{6.19}$$

Recall that each row of the precedence matrix $\mathbf{P}$ has exactly one $+1$ and one -1 and each row of the matrix $\mathbf{S}$ has only one $+1$. Therefore, the columns of $\mathbf{A}$ can be partitioned into two groups as shown in (6.19). For each row in the left group, the sum of the elements is always 0; for each row in the right group, the sum of the elements is always $+1$. By Theorem 2.7 in Nemhauser and Wolsey (1988, page 542), matrix $\mathbf{A}$ is totally unimodular. Note that augmenting (6.19) with the variable bounds identity matrix will not change this result. Therefore, using the L-shaped method to solve the LP relaxation of SP without a budget constraint will produce integer-valued optimal solutions. □

With budget constraint. When constraint (6.5) is included in the second stage problem, the solution to the LP relaxation of SP is not guaranteed to be integral. As mentioned, the L-shaped cannot be applied directly to a stochastic integer program because the objective function of the second stage problem is not convex. Moreover, solving the equivalent single ILP is only practical when a few scenarios are included in the model.

As an alternative, we propose a heuristic for constructing feasible integer solutions from the relaxed LP solutions. The idea is to first round all fractional target finish times to their nearest integer values. This will give us a feasible integer solution to the first stage problem. Using these values, we then solve the second stage scheduling problem for each scenario to obtain integer values for the crashing variables.

At this point, we construct a new stochastic programming problem without

a budget constraint by subtracting the integer crashing values from the corresponding activity durations for each scenario. This is equivalent to fixing the crashing variables at values that are feasible for each scenario (i.e., satisfying the budget constraint). We then use the L-shaped method to solve this stochastic programming problem without a budget constraint to get the optimal target times for the current set of crashing values. The expectation is that the new target times will provide a better objective function value for SP.

A formal statement of this procedure follows.

Procedure 6.3.1: Heuristic for SP with a budget constraint

Step 1. Solve the LP relaxation of SP and denote the optimal target times by $\mathbf{t}^*_{LP}$ and the optimal objective function value by z^*_{LP}. Let $\mathbf{t}_R$ be the vector of target times obtained by rounding each component of $\mathbf{t}^*_{LP}$ to the nearest integer.

Step 2. Set $\mathbf{t} = \mathbf{t}_R$ in the second stage problem (6.8) – (6.12) and solve it to obtain integer-valued deviation variables $\mathbf{y}_R(\omega)$ and crashing variables $\mathbf{x}_R(\omega)$ for each scenario $\omega \in \Omega$. Compute the objective function value of the first stage problem and denote it by z_R.

Step 3. Form a new stochastic integer programming problem without a budget constraint by setting $\mathbf{x} = 0$ and replacing $\mathbf{p}(\omega)$ with $\mathbf{p}(\omega) - \mathbf{x}_R(\omega)$ for each scenario $\omega \in \Omega$ in the second stage problem (6.13) – (6.16). Solve this problem with the L-shaped method to obtain integer-valued deviation variables $\mathbf{y}_H(\omega)$ for each scenario $\omega \in \Omega$, and target times $\mathbf{t}_H$ – our heuristic solution. Denote the optimal objective function value of this new stochastic programming problem by z_N. Finally, compute the objective function value of SP for $\mathbf{t} = \mathbf{t}_H$ and denote it by z_H.

Proposition 6.3.2. *Let z^* be the optimal objective function value of our original stochastic integer programming problem* (6.1)*. Then $z^*_{LP} \leq z^* \leq z_H \leq z_N \leq z_R$.*

PROOF. Define

$$\mathbf{x} = (\mathbf{x}(\omega_1), \ldots, \mathbf{x}(\omega_S))$$

and

$$\mathbf{y} = \left(\mathbf{y}^+(\omega_1), \mathbf{y}^-(\omega_1) \ldots, \mathbf{y}^+(\omega_S), \mathbf{y}^-(\omega_S)\right),$$

and let $z(\mathbf{t}, \mathbf{x}, \mathbf{y})$ be the objective function value of SP with a budget constraint for given target times $\mathbf{t}$, feasible crashing values $\mathbf{x}$, and deviation variables $\mathbf{y}$. The set of feasible crashing values is $\mathbb{X} \equiv \{\mathbf{x} : \mathbf{dx}(\omega) \leq B, \ \forall\ \omega \in \Omega\}$.

Solving the optimization problem in Step 2 gives

$$z_R = \min_{\mathbf{y} \in \mathbb{Z}_+^{2(n+1)S}} z(\mathbf{t}_R, \mathbf{x}_R, \mathbf{y}) \tag{6.20}$$

From Step 3, we have

$$z_N = \min_{\mathbf{t} \in \mathbb{Z}_+^{n+2},\ \mathbf{y} \in \mathbb{Z}_+^{2(n+1)S}} z(\mathbf{t}, \mathbf{x}_R, \mathbf{y}) = \min_{\mathbf{y} \in \mathbb{Z}_+^{2(n+1)S}} z(\mathbf{t}_H, \mathbf{x}_R, \mathbf{y}) \tag{6.21}$$

and

$$z_H = \min_{\mathbf{x} \in \mathbb{X},\ \mathbf{y} \in \mathbb{Z}_+^{2(n+1)S}} z(\mathbf{t}_H, \mathbf{x}, \mathbf{y}) \tag{6.22}$$

The relationship $z_N \leq z_R$ follows from (6.20) and (6.21), while $z_H \leq z_N$ follows from (6.21) and (6.22). The remaining relationships follow from the definition of optimality. □

The main computational effort in the above procedure involves solving the two stochastic linear programming problems in Steps 1 and 3, respectively. The objective function value associated with the solution to the LP relaxation obtained in Step 1 provides a lower bound on the optimal solution to SP. Thus, the performance of the heuristic can be measured by the gap between this value, z_{LP}^*, and the objective function value computed in Step 3, z_H. Note that looping through Steps 2 and 3 until some stopping criteria are met did not lead to any noticeable improvement so we did not include this option in the procedure.

6.3.2 Network Flow Formulation for the Second Stage Problem

Now we show that the second stage problem without a budget constraint (6.13) – (6.16) is equivalent to a minimum cost network flow problem (MCNFP). Denote the dual variables for the precedence constraint (6.14) and the crashing upper bound constraint (6.15) by $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, respectively. Then the dual LP of (6.13) – (6.16) can be written as

$$\text{Maximize} \quad \boldsymbol{\lambda}(\mathbf{p} - \mathbf{Pt}) - \boldsymbol{\mu}\mathbf{u} \tag{6.23}$$

$$\text{subject to} \quad \boldsymbol{\lambda}\mathbf{P} \leq \mathbf{q}^{+} \tag{6.24}$$

$$-\boldsymbol{\lambda}\mathbf{P} \leq \mathbf{q}^{-} \tag{6.25}$$

$$\boldsymbol{\lambda}\mathbf{S} - \boldsymbol{\mu} \leq \mathbf{d} \tag{6.26}$$

$$\boldsymbol{\lambda} \geq \mathbf{0}, \ \boldsymbol{\mu} \geq \mathbf{0} \tag{6.27}$$

Note that since $y_0^+ = y_0^- = 0$, (6.24) and (6.25) are not required for activity 0. In an activity-on-node (AON) project network, if we view $\boldsymbol{\lambda}$ as the flows on the precedence arcs, then the term $\boldsymbol{\lambda}\mathbf{P}$ represents the net flows at the nodes. Constraint (6.24) and (6.25) indicate that these flows have to be bounded.

We now describe a procedure that transfers the LP (6.23) – (6.27) into a MCNFP, where $\boldsymbol{\lambda}, \boldsymbol{\mu}$ are the arc flows to be determined and (6.24) – (6.26) are enforced by arc capacities and flow conservation constraints.

Procedure 6.3.2: Constructing an MCNFP for linear program (6.23) – (6.27).

Step 1. Create a network with a node 0 and a dummy node d.

Step 2. For each activity $i \neq 0$ with $u_i = 0$ (i.e., non-crashable):

a) Add node i to the network.

b) Add arc (i, d) with lower bound $-q_i^-$, upper bound q_i^+ and flow cost 0.

Step 3. For each activity $i \neq 0$ with $u_i > 0$ (i.e., crashable):

a) Add three nodes $i^+,\ \underline{i},\ i^-$ to the network.

b) Add the following arcs: $(i^+, \underline{i})$ with upper bound ∞ and flow cost u_i; $(\underline{i}, i^-)$ with upper bound ∞ and flow cost 0; (i^+, i^-) with lower bound $-\infty$, upper bound d_i and flow cost 0.

c) Add arc (i^-, d) with lower bound $-q_i^-$, upper bound q_i^+, and flow cost 0.

Step 4. For each precedence constraint (i, j), depending the nodes added, add to the network one of the following arcs: $(i, j^+), (i^-, j)$ or (i^-, j^+) (only one of these three arcs exists in the network) with upper bound ∞ and flow cost $t_j - t_i - p_j$.

Step 5. Add arc $(d, 0)$ to the network with upper bound ∞ and flow cost 0. Set all non-specified lower bounds to 0. Set demand at all nodes to 0.

According to the above procedure, the network structure created for each activity fits one of the two situations shown in Figure 6.2, depending on whether the activity is crashable. Arcs are labelled by [lower bound, upper bound, flow cost] in the figure. The full network is obtained by linking these substructures through precedence arcs and an arc from the dummy d to node 0. Because there is no external demand at any node, the resultant MCNFP is a flow circulation problem. An example of the transformation is given in Section 6.4.

Let z_{NF} be the minimum cost of the MCNFP constructed with the above procedure. Denote the optimal flow on arc $(i, j^+),\ (i^-, j)$ or (i^-, j^+) by f_{ij}^* and the optimal flow on arc $(i^+, \underline{i})$ by g_i^*. We have the following result.

Proposition 6.3.3. *The optimal objective function value of the LP* (6.23) – (6.27) *is* $-z_{NF}$ *and in the optimal solution,* $\lambda_{ij}^* = f_{ij}^*\ , \forall\ (i, j) \in \mathcal{P}$ *and* $\mu_i^* = g_i^*, \forall\ i \in \{i : u_i > 0\}$.

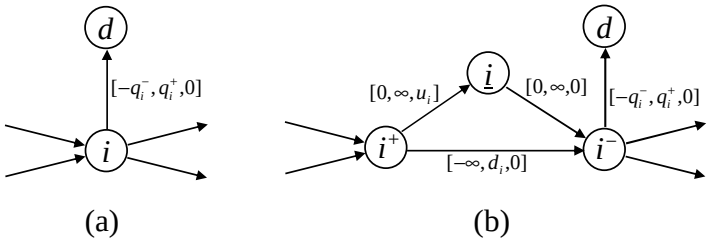

Figure 6.2: Network structures: (a) $u_i = 0$; (b) $u_i > 0$

PROOF. Let $\mathbf{f}^*$ be the vector of $f^*_{ij}, (i,j) \in \mathcal{P}$ and $\mathbf{g}^*$ be the vector of $g^*_i, i \in \{i : u_i > 0\}$. We will show that $\boldsymbol{\lambda} = \mathbf{f}^*$ and $\boldsymbol{\mu} = \mathbf{g}^*$ satisfies constraints (6.24) – (6.26) and maximizes the objective function in (6.23).

In the optimal solution of MCNFP, denote the flow on arc (i,d) or (i^-,d) (only one of these two arcs exists in the network) by $\alpha^*_i, \forall\ i \neq 0$. The flow conservation condition at node i or i^- (if activity i is crashable) can be written as

$$\sum_{(k,i)\in\mathcal{P}} f^*_{ki} - \sum_{(i,j)\in\mathcal{P}} f^*_{ij} = \alpha^*_i \tag{6.28}$$

The flow bounds are defined such that $-q^-_i \leq \alpha^*_i \leq q^+_i$, which, when combined with (6.28), gives

$$-q^-_i \leq \sum_{(k,i)\in\mathcal{P}} f^*_{ki} - \sum_{(i,j)\in\mathcal{P}} f^*_{ij} \leq q^+_i \tag{6.29}$$

By the definition of the precedence matrix $\mathbf{P}$, the left-hand side of (6.28) is simply the ith row of $\mathbf{f}^*\mathbf{P}$. Therefore, constraints (6.24) and (6.25) are satisfied by $\boldsymbol{\lambda} = \mathbf{f}^*$.

For a crashable activity i, g^*_i is the flow on arc $(i^+, \underline{i})$. Denote the flow on arc (i^+, i^-) by β^*_i in the optimal MCNFP solution. Then the flow conservation condition at node i^+ gives

$$\beta^*_i = \sum_{(k,i)\in\mathcal{P}} f^*_{ki} - g^*_i \tag{6.30}$$

where $\sum_{(k,i)\in\mathcal{P}} f^*_{ki}$ is the ith row of $\mathbf{f}^*\mathbf{S}$ and $\beta^*_i \leq d_i$. Thus, constraint (6.26) is satisfied by setting $\boldsymbol{\lambda} = \mathbf{f}^*$ and $\boldsymbol{\mu} = \mathbf{g}^*$.

Finally, recall that in Procedure 6.3.2, the flow costs were set such that the total network cost is $-[\mathbf{f}(\mathbf{p}-\mathbf{Pt})-\mathbf{gu}]$, which is minimized at $\mathbf{f}=\mathbf{f}^*$, $\mathbf{g}=\mathbf{g}^*$. As a result, the objective function $\boldsymbol{\lambda}(\mathbf{p}-\mathbf{Pt})-\boldsymbol{\mu}\mathbf{u}$ in (6.23) is maximized at $\boldsymbol{\lambda}=\mathbf{f}^*$ and $\boldsymbol{\mu}=\mathbf{g}^*$, giving $\mathbf{f}^*(\mathbf{p}-\mathbf{Pt})-\mathbf{g}^*\mathbf{u}$, or $-z_{NF}$. □

A minimum cost network flow problem can usually be solved more efficiently with a specialized algorithm than with a general LP code. However, for each crashable activity, we need to add three nodes and three or four arcs to the network. This may make the proposed approach less attractive when a project has many crashable activities.

6.4 Computations

6.4.1 A 5-Activity Project Example

The solution procedure will be illustrated with a 5-activity (7 activities when dummies are included) project. Figure 6.3 shows the accompanying AON network with baseline activity durations. To introduce uncertainty, we consider the case in which independent activities 1 and 4 each has a 50% chance of being 3 times as long as its baseline duration. If all other activities have deterministic length, there are 4 scenarios: (1) $p_1=3, p_4=2$; (2) $p_1=9, p_4=2$; (3) $p_1=3,\ p_4=6$; (4) $p_1=9,\ p_4=6$. Each scenario has a 0.25 probability.

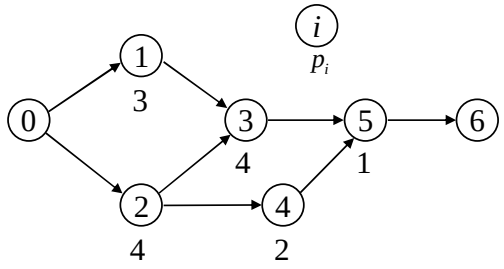

Figure 6.3: Project network

To simplify the presentation, we only consider the problem of finding the optimal target time for project completion, not each activity. Let the cost coefficients be $c_6 = 3,\ q_6^- = 2,\ q_6^+ = 5$ and $c_i = q_i^+ = q_i^- = 0,\ \forall\ i = 0, \ldots, 5$. We also assume that activity 3 can be crashed up to 2 time periods at a cost of 4 per period. The crashing cost is added into the objective function so that the second stage scheduling problem can be solved as an MCNFP.

Following the procedure in Section 6.3.2, we construct the MCNFP network as shown in Figure 6.4. Because we are only considering the project target time, only node 6 (corresponding to activity 6, the dummy activity for project completion) is linked with dummy node d. For crashable activity 3, three nodes (in the dashed box in Figure 6.4) are created in the network to enforce the upper bound on the crashing variable. Flow bounds and flow costs of arcs are set according to the procedure in Section 6.3.2.

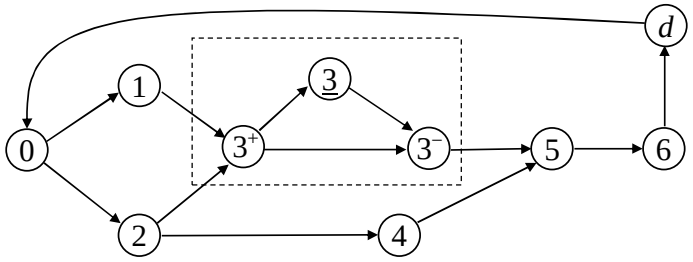

Figure 6.4: MCNFP network structure for example

Solving the SP for this example with the L-shaped method gave an optimal target time for project completion of 11 and a minimum expected cost of 39.5. This solution was obtained after 5 optimality cuts (shown in Figure 6.5) were added to the first stage problem. These cuts approximate the objective function from below until convergence takes place. We also computed the objective values, z, for various values of t_6 and plotted them in Figure 6.5 (black dots).

In Figure 6.6, we show how the optimal solution (black dots) changes as a

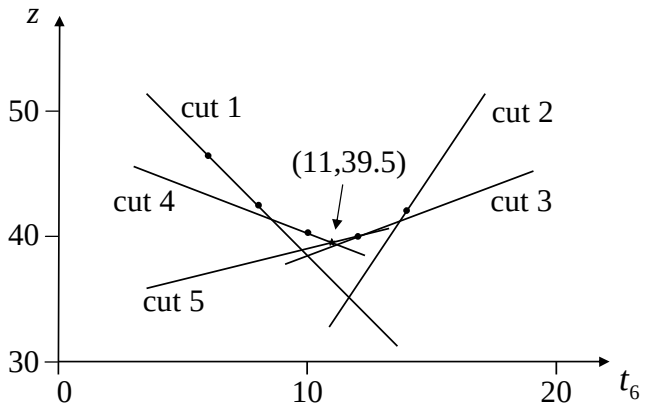

Figure 6.5: Optimality cuts

function of the cost coefficients. As expected, the optimal target time increases when the lateness penalty q_6^+ increases and the target time cost c_6 decreases. It can also be seen that the optimal target time t_6^* does not change continuously as these parameters change.

When there is no crashing option, the 1-variable decision problem is equivalent to a newsvendor problem. For the 4 scenarios, the minimum project makespans are $8, 11, 14$ and 14, respectively. Following Proposition 6.2.3, the equivalent newsvendor problem has $c_s = q_6^+ - c_6,\ c_e = c_6$ and the optimal solution satisfies $F(t_6^*) = \frac{q_6^+ - c_6}{q_6^+}$. The demand for the newsvendor problem has a discrete distribution: (demand, probability) $= (8, 0.25),\ (11, 0.25),\ (14, 0.5)$. The corresponding solutions are shown in Figure 6.6.

From this figure, we also see that when q_6^+ is high or c_6 is low, the optimal target time is higher without the crashing option. On the other hand, when q_6^+ is low or c_6 is high, the optimal target time is lower. In other words, the existence of crashing options tends to smooth the effects of the cost coefficients q_6^+ and c_6. In the extreme case, if all activity durations can be crashed to zero at no cost, then the optimal target time t_i^* will always be zero for $c_i > 0,\ \forall\ i$, implying that the cost coefficients do not affect the outcome in this case.

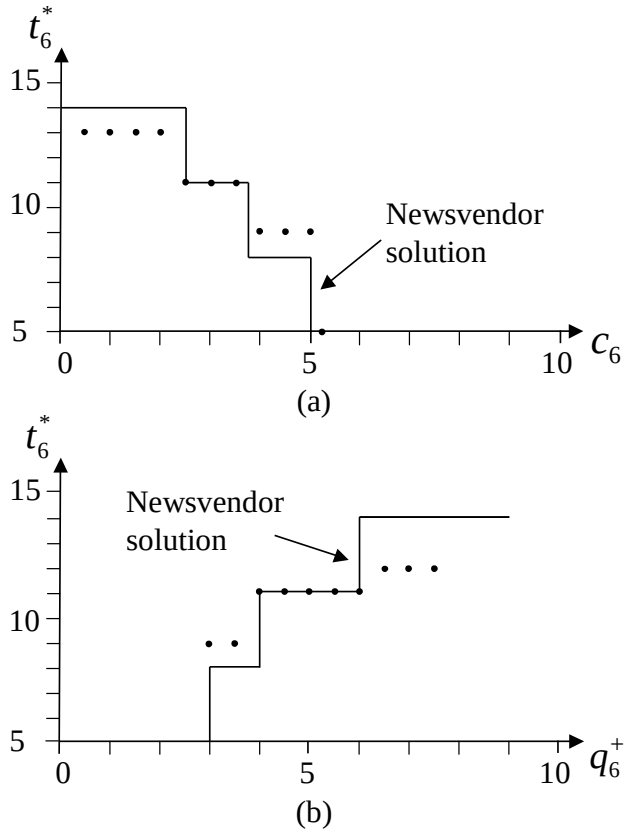

Figure 6.6: Change of optimal target time with respect to parameters: (a) target time cost c_6; (b) lateness penalty q_6^+

6.4.2 Benchmark Problem Results

To assess the proposed models and solution approaches, we solved a set of 90-activity benchmark problems obtained from the online library – PSPLIB (http://www.bwl.uni-kiel.de/Prod/psplib/). The library contains 480 single-mode, 90-activity instances divided into 48 groups. Each group is characterized by a combination of resource usage and availability, network complexity (NC), and duration distributions (Kolisch et al., 1995).

Network complexity, defined as the average number of nonredundant precedence constraints per activity, was the only property relevant to our work. Of the

480 instances, we used 150 (15 groups) in the analysis. For those, three different NC values were used in the generation process: 1.5, 1.8 and 2.1, giving rise to 50 instances for each value (groups 1–5 for $NC = 1.5$, groups 17–21 for $NC = 1.8$ and groups 33–37 for $NC = 2.1$).

To generate scenarios, we assumed that for each project instance, 8 activities (16, 26, 36, 46, 56, 66, 76 and 86) had uncertain duration that took one of two possible values: (1) the same as in the benchmark data sets with probability 0.5, and (2) twice the original duration with probability 0.5. All other activities were assigned the same durations as in the benchmark data sets. It was also assumed that activity durations were mutually independent random variables giving a total of $2^8 = 256$ scenarios in all.

In addition, we allowed 18 activities ($2 + 5i,\ i = 0, \ldots, 17$) to be crashed up to 3 time periods; however, their duration after crashing was not allowed to go below 1. Unit crashing costs (d_i) were randomly generated in the interval $[2, 4]$.

Target times were specified for 17 activities ($5i,\ i = 1, \ldots, 17$) plus the project completion time. Earliness and lateness penalties were only defined for these variables. The target time cost coefficients [i.e., c_i in (6.1)] and the earliness penalty cost coefficients (i.e., q_i^-) were generated randomly in the interval $[0.5, 1]$. The unit lateness penalty (i.e., q_i^+) was set to 4 for project completion and generated randomly in the interval $[2, 3]$ for the 17 activities. In all cases, the random numbers were drawn from a uniform distribution in the specified interval.

In the first analysis, we considered the case without a budget constraint where the crashing costs are penalized in the objective function of the second stage problems (6.13) – (6.16). Recall that these problems were shown to be solvable as MCNFPs; however, for each crashable activity, additional nodes and arcs must be included in the network. To investigate the difficulty of SP as a function of the number of crashable activities, we used an experimental design that varied both the number of crashable activities (N_{CA}) and the network complexity (NC).

As part of the testing, the second stage problems were solved with two approaches: (1) as MCNFPs using the scaling push-relabel method by Goldberg (1997) ; and (2) as general LPs using CPLEX 7.5 (ILOG, 2002). In all cases, CPLEX was used to solve the first stage LPs. All algorithms were implemented in C++ and all computations were performed on a Linux workstation with a Xeon 1.8GHz processor.

The results are reported in Table 6.1 using the following column headers.

N_{CA}	=	number of crashable activities; activity $2 + 5i$, $i = 0, \ldots, N_{CA} - 1$ are crashable;
z^*	=	optimal objective function value
VSS	=	value of stochastic solution given by the difference between the objective function value of the deterministic problem with expected durations and z^*; the value in parentheses in Table 6.1 is $\frac{VSS}{z^*} \times 100$
$EVPI$	=	expected value of perfect information given by the difference between z^* and the expected value with perfect information (with perfect information, the target times are set to activity finish times in $ES(\mathbf{p}(\omega), \forall\ \omega \in \Omega)$; the value in parentheses in Table 6.1 is $\frac{EVPI}{z^*} \times 100$
Nodes	=	number of nodes in the corresponding MCNFP
Arcs	=	number of arcs in the corresponding MCNFP
t_{MCNF}	=	CPU time (seconds) when second stage problems were solved as MCNFPs
t_{LP}	=	CPU time (seconds) when second stage problems were solved with CPLEX LP solver.

The computational results show that the value of the stochastic solution (VSS) is about 2% of the optimal solution on average. Depending on the problem parameters, maximum values ranged from 2.97% to 5.61%, indicating that a stochastic programming approach is frequently justified. For 90-activity projects,

the analysis can be carried out in a reasonable amount of time even with hundreds of scenarios.

The statistics in Table 6.1 also show that computation times increase with NC. Higher NC values produce more interaction between activities, which leads to larger optimal objective function values and more difficult second stage problems.

Solution times in Table 6.1 indicate that solving the second stage problems as MCNFPs is more efficient than solving them as general LPs for all cases examined. Nevertheless, as the number of crashable activities increases, the computation times associated with the MCNFPs increase faster than those associated with the LPs. This is due to the fact that adding a crashable activity requires that two extra nodes and three extra arcs be added to the MCNFP model, while in the LP formulation, only one additional variable is required. It is therefore possible that the LP approach may outperform the MCNFP approach as the number of crashable activities increases.

In the second set of computations, we solved SP with a budget constraint using our heuristic approach. Here, 18 activities were assumed to be crashable (i.e., $N_{CA} = 18$) and the total crashing budget was fixed at 10. All other parameters were the same as for the case without a budget constraint. Table 6.2 lists the CPU time t_{CPU}, LP lower bound z_{LP}, upper bound z_R obtained with the rounding heuristic (the number in parentheses is the relative gap, $\frac{z_R - z_{LP}}{z_{LP}} \times 100$), improved upper bound z_H (the number in parentheses is the relative gap, $\frac{z_H - z_{LP}}{z_{LP}} \times 100$), as well as VSS and $EVPI$ for the final heuristic solution.

Table 6.1: Computational results for 90-activity projects without a budget constraint

			$VSS(\%)$		$EVPI(\%)$					
N_{CA}	NC	z_*	Average	Max	Average	Max	Nodes	Arcs	t_{MCNF}	t_{LP}
	1.5	575.8	5.8 (1.01)	18.2 (3.03)	13.0 (2.23)	37.2 (5.59)	93	175	13.22	20.08
0	1.8	679.6	7.0 (1.03)	20.2 (3.01)	16.7 (2.44)	38.5 (5.52)	93	202	16.40	27.78
	2.1	728.7	8.6 (1.19)	21.7 (2.97)	18.8 (2.56)	51.2 (6.21)	93	231	20.97	36.36
	1.5	569.0	10.8 (1.91)	25.1 (4.70)	12.7 (2.21)	37.2 (5.83)	104	192	17.00	21.62
6	1.8	667.1	11.4 (1.70)	28.8 (4.36)	16.3 (2.43)	38.5 (5.52)	104	219	22.15	29.83
	2.1	715.8	13.0 (1.83)	27.6 (4.29)	17.9 (2.47)	51.2 (6.21)	104	246	26.24	38.33
	1.5	568.1	12.9 (2.28)	29.9 (5.61)	12.1 (2.10)	37.2 (5.83)	115	208	19.35	21.98
12	1.8	666.1	13.8 (2.09)	30.9 (4.50)	15.7 (2.34)	36.7 (5.39)	115	236	24.70	30.70
	2.1	714.1	16.0 (2.26)	30.9 (4.37)	17.1 (2.36)	51.0 (6.19)	115	263	29.68	38.90
	1.5	566.9	15.0 (2.65)	30.0 (5.27)	11.0 (1.92)	36.1 (5.68)	125	224	22.01	23.08
18	1.8	664.7	15.6 (2.36)	33.2 (5.04)	14.4 (2.16)	36.7 (5.35)	125	251	27.42	31.96
	2.1	713.0	18.1 (2.57)	33.9 (5.19)	16.1 (2.24)	50.1 (6.09)	125	279	31.89	40.21

The bottom line of Table 6.2 shows that by using the heuristic proposed in Section 6.3, we are able to reduce the relative optimality gap from approximately 1.02% (for the rounding heuristic solution with gap $\frac{z_R - z_{LP}}{z_{LP}} \times 100\%$) to about 0.34% (for the improved heuristic solution with gap $\frac{z_H - z_{LP}}{z_{LP}} \times 100\%$) on average. Because the heuristic solutions are measured against the LP lower bound, the true optimality gap $\frac{z_H - z^*}{z^*}$ may be even smaller. Computation times (t_{CPU} in Table 6.2) for the IP version of SP with a budget constraint were roughly twice as high as the computation times (t_{LP} in Table 6.1) obtained when SP was solved as an LP without an explicit budget constraint.

Table 6.2: Computational results for 90-activity projects with a budget constraint

NC	t_{CPU}	z_{LP}	$z_R(\%)$	$z_H(\%)$	$VSS(\%)$	$EVPI(\%)$
1.5	50.74	560.8	565.7(0.86)	562.5(0.28)	7.5(1.35)	14.8(2.60)
1.8	67.33	661.0	668.7(1.16)	663.7(0.40)	7.1(1.08)	19.5(2.91)
2.1	87.91	710.0	717.6(1.06)	712.5(0.35)	9.0(1.29)	21.3 (2.96)
Average	68.66	643.9	650.7(1.02)	646.20(0.34)	7.9(1.24)	18.5(2.82)

6.5 Summary

Projects are performed in an uncertain environment, yet some decisions have to be made before the uncertainty is resolved. These decisions affect all project measures as well as the ability to recover in the face of disruptions. Balancing the performance of an initial plan with accumulated costs is critical in many real-world situations where small improvements in performance translate into millions of dollars in savings.

A major contribution of this chapter relates the development of a two-stage stochastic planning model that takes into account deviations from prespecified activity finish times. We presented some properties of the model and draw a link between the project scheduling problem under uncertainty and the traditional newsvendor problem. For the case in which crashing costs were penalized rather than constrained, we developed an efficient exact algorithm by first transforming the second stage scheduling problems into minimum cost network flow problems. For the case with a budget constraint, an LP-based heuristic was proposed and seen to find good feasible solutions.

A second contribution of this work concerns the importance of accounting for potential deviations from early commitments in the planning phase of a project. One insight that resulted from our analysis was that when deviations from prespecified target finish times are considered, early schedules are generally not obtained. Moreover, the optimal target completion time for a project may be greater than the makespan of all early schedules under any scenario. When crashing is permitted, traditional procedures for determining activity target times must be updated to account for uncertainty.

Chapter 7

Conclusions

7.1 Summary

Projects management provides some general guidelines and methods for successful completion of projects, which are often considered to be unique. In most real-world environments, the uncertain aspects of a project, coupled with their scope and magnitude, vastly complicate the decision making process. This dissertation should be viewed as an attempt to address some of these issues as they relate to resource-constrained project planning and scheduling.

In the first phase of the research, we considered a general multi-mode, resource-constrained project scheduling problem. An exact branch-and-cut procedure was developed by exploiting the features of the corresponding integer linear programming formulation. Empirical tests on benchmark problems showed that the proposed approach gave markedly improved results with respect to any of the implementations reported in literature. A primary advantage of the methodology is its flexibility in the selection of heuristics and lower bounding schemes at the subroutine level.

The next component of the research centered on the development of reactive strategies that could be used when some aspect of a project is disrupted. The emphasis was on both modeling and solving the perturbed problems. Tests showed that spending either too much or too little effort in solving the original scheduling problem may produce inferior results when disruptions are inevitable. This calls into question the need and value of obtaining optimal initial schedules when major disruptions are likely. Though 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.

For a simplified case with only a budget constraint, we studied how the disruption recovery action affects the decisions on target times for project activities. We developed a two-stage stochastic planning model that takes into account deviations from pre-specified activity finish times. One insight that resulted from our analysis was that when deviations from pre-specified target finish times are considered, early schedules are generally not obtained. It was shown that the results may be significantly different when deviations are considered, compared to when activities are scheduled as early as possible in the traditional way. For example, the optimal target completion time for a project may be greater than the makespan of the early-start schedules under any scenario.

For all the problems studied, we provided detailed information on modeling and solution procedures. Examples and computations were also given to demonstrate the effectiveness of the solution approaches.

7.2 Future Work

Real-world operations are always subject to disruptions. The emerging research area of disruption management provides theory and methods for dealing with disruptions. Many important issues in disruption management are still to be considered and solved. The following are a few possible directions for future work following this dissertation.

The first centers on models. The goal of this research would be to provide methods that can be used for real-world projects, and managerial insights that can guide the practice of project management. A first step toward this goal is to use models that capture the essence of real problems. More complicated temporal constraints, such as maximum time lags and continuous time, should be considered. The

resource constraints included in the dissertation are not general enough to account for many real-world situations, such as resource substitutability and flexibility. Also, the time-indexed integer programming model was shown to have several weaknesses when it came to time to solve. Alternative ways of describing the scheduling and resource allocation requirements are desired. When uncertainty is considered, extending the two-period model to multi-periods to better represent the actual project execution process, is also a very important goal.

The second major research direction concerns solution approaches. In particular, it may be possible to exploit more of the special structure of the problem. With respect to the hybrid solution approach, for example, more elaborate resource-based constraint propagation techniques may be developed. When possible, a combination of heuristics and exact solution approaches may lead to much improved performance. In addition, new solution approaches should be developed when new elements are added into the model.

Disruption management investigates the reactive opportunities that exist when disruptions occur. Such opportunities come from the managerial flexibilities built into the operations. In fact, disruption management can be viewed as a real option for which the cost and payoff are much more complicated than in existing real option analysis. Therefore, valuation of disruption management is both an interesting and challenging problem. A related extension would focus on how to design managerial flexibility to enhance the value of disruption management.

Another important direction of future research is to apply the ideas developed in the dissertation to common applications, such as software and process improvement projects. This will not only allow us to validate the techniques developed, but also to provide incentives for the use of more sophisticated modeling and solution techniques.

Bibliography

Al-Fawzana, M. and Haouari, M. (2005). A bi-objective model for robust resource-constrained project scheduling. *International Journal of Production Economics*, 95:175–187.

Argüello, M., Bard, J. F., and Yu, G. (1997). A GRASP for aircraft routing in response to groundings and delays. *Journal of Combinatorial Optimization*, 5:211–228.

Baker, K. and Scudder, G. (1990). Sequencing with earliness and tardiness penalties: a review. *Operations Research*, 38(1):22–36.

Balas, E., Ceria, S., and Cornuejols, G. (1996). Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246.

Balas, E., Lenstra, J., and Vazacopoulos, A. (1995). The one-machine problem with delayed precedence constraints and its use in job shop scheduling. *Management Science*, 41(1):94–109.

Baptiste, P., Le Pape, C., and Nuijten, W. (2001). *Constraint-based scheduling: applying constraint programming to scheduling problems*. Kluwer Academic Publishers, Boston, MA.

Bard, J. F., Kontoravdis, G., and Yu, G. (2002). A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, 36(2):250–269.

Bartusch, M., Möhring, R. H., and Radermacher, E. J. (1988). Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16:201–240.

Birge, J. R. (1997). Stochastic programming computation and applications. *INFORMS Journal on Computing*, 9(2):111–113.

Birge, J. R. and Dempster, M. (1996). Stochastic programming approaches to stochastic scheduling. *Journal of Global Optimization*, 9(3-4):417–452.

Bixby, R. E. (2002). Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15.

Bockmayr, A. and Kasper, T. (1998). Branch and infer: A unifying framework for integer and finite domain constraint programming. *INFORMS Journal on Computing*, 10(3):287–300.

Bottcher, J., Drexl, A., Kolisch, R., and Salewski, F. (1999). Project scheduling under partially renewable resource constraints. *Management Science*, 45(4):543–559.

Bouleimen, K. and Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, 149(2):268–281.

Bowers, J. A. (2000). Interpreting float in resource constrained projects. *International Journal of Project Management*, 18(5):358–392.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41.

Brucker, P. and Knust, S. (2000). A linear programming and constraint propagation-based lower bound for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127(2):355–362.

Brucker, P. and Knust, S. (2003). Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research*, 149(2):302–313.

Brucker, P., Knust, S., Schoo, A., and Thiele, O. (1998). A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107(2):272–288.

Chapman, C. (1997). Project risk analysis and management - parm the generic process. *International Journal of Project Management*, 15(5):273–281.

Chrétienne, P. and Sourd, F. (2003). Pert scheduling with convex cost functions. *Theoretical Computer Science*, 292(1):145–164.

Clausen, J., Hansen, J., Larsen, J., and Larsen, A. (2001). Disruption management. *ORMS Today*, 28(5):40–43.

De Reyck, B. and Herroelen, W. (1996). On the use of the complexity index as a measure of complexity in activity networks. *European Journal of Operational Research*, 91(2):347–366.

De Reyck, B. and Herroelen, W. (1999). The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 119(2):538–556.

Dorndorf, U., Pesch, E., and Phan-Huy, T. (2000). A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science*, 46(10):1365–1384.

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. *Journal of the Operational Research Society*, 51:291–300.

Elmaghraby, S. E. (2000). On criticality and sensitivity in activity networks. *European Journal of Operational Research*, 127(2):220–238.

Fischetti, M. and Lodi, A. (2003). Local branching. *Mathematical Programming*, 98(1):23–47.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman, New York, NY.

Goldberg, A. V. (1997). An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22(1):1–29.

Golenko-Ginzburg, D. and Gonik, A. (1997). Stochastic network project scheduling with non-consumable limited resources. *International Journal of Production Economics*, 48(1):29–37.

Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W. (1998). Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal On Computing*, 10(4):427–437.

Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W. (1999). Lifted cover inequalities for 0-1 integer programs: Complexity. *INFORMS Journal on Computing*, 11(1):117–123.

Gutierrez, G. and Paul, A. (2000). Analysis of the effects of uncertainty, risk-pooling, and subcontracting mechanisms on project performance. *Operations Research*, 48(6):927–938.

Hartmann, S. (1999). *Project scheduling under limited resources*. Lecture notes in economics and mathematical systems, Vol. 478. Springer, Berlin, Germany.

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

Herroelen, W. and Leus, R. (2004). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*.

Herroelen, W. S., van Dommelen, P., and Demeulemeester, E. L. (1997). Project network models with discounted cash flows: a guided tour through recent developments. *European Journal of Operational Research*, 100(1):97–121.

Icmeli, O. and Rom, W. O. (1996). Solving the resource constrained project scheduling problem with optimization subroutine library. *Computers & Operations Research*, 23(8):801–817.

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

Jain, V. and Grossmann, I. E. (2001). Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13(4):258–276.

Kilby, P., Prosser, P., and Shaw, P. (1998). A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Constraints*, 5(4):389–414.

Klein, R. (2000). Bidirectional planning: Improving priority rule-based heuristics for scheduling resource-constrained projects. *European Journal of Operational Research*, 127(3):619–638.

Klein, R. and Scholl, A. (1999). Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. *European Journal of Operational Research*, 112(2):322–346.

Klein, R. and Scholl, A. (2000). Progress: Optimally solving the generalized resource-constrained project scheduling problem. *Mathematical Methods of Operations Research*, 52(3):467–488.

Klein Haneveld, W. K. and van der Vlerk, M. H. (1999). Stochastic integer programming: General models and algorithms. *Annals of Operations Research*, 85(1):39–57.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333.

Kolisch, R. and Padman, R. (2001). An integrated survey of deterministic project scheduling. *OMEGA*, 29(3):249–272.

Kolisch, R. and Sprecher, A. (1997). PSPLIB - a project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216.

Kolisch, R., Sprecher, A., and Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41(10):1693–1703.

Lin, E. Y. (1998). A bibliographical survey on some well-known non-standard knapsack problems. *INFOR*, 36(4):274–317.

Ludwig, A., Möhring, R. H., and Stork, F. (2001). A computational study on bounding the makespan distribution in stochastic project networks. *Annals of Operations Research*, 102:49–64.

Mak, W.-K., Morton, D. P., and Wood, R. K. (1999). Monte carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24(1):47–56.

Miller, R. and Lessard, D. (2001). Understanding and managing risks in large engineering projects. *International Journal of Project Management*, 19(8):437–443.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. (1998). An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44(5):714–729.

Möhring, R. H., Schulz, A. S., Stork, F., and Uetz, M. (2001). On project scheduling with irregular starting time costs. *Operations Research Letters*, 28:149–154.

Möhring, R. H., Schulz, A. S., Stork, F., and Uetz, M. (2003). Solving project scheduling problems by minimum cut computations. *Management Science*, 49(3):330–350.

Möhring, R. H., Schulz, A. S., and Uetz, M. (1999). Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the ACM*, 46(6):924–942.

Möhring, R. H. and Stork, F. (2000). Linear preselective policies for stochastic project scheduling. *Mathematical Methods of Operations Research*, 52:501–515.

Mulvey, J. M., Vanderbei, R. J., and Zenios, S. A. (1995). Robust optimization of large-scale systems. *Operations Research*, 43(2):264–281.

Nemhauser, G. and Wolsey, L. (1988). *Integer and Combinatorial Optimization*. John Wiley and sons, Inc., Berlin, Germany.

Neumann, K., Schwindt, C., and Zimmermann, J. (2002). *Project scheduling with time windows and scarce resources: temporal and resource constrained project scheduling with regular and non-regular objective functions*. Lecture notes in economics and mathematical systems, Vol. 508. Springer.

Nonobe, K. and Ibaraki, T. (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem. *C.C. Ribeiro and P. Hansen (eds.) : Essays and Surveys in Metaheuristics*, pages 557–588.

Özdamar, L. and Ulusoy, G. (1995). A survey on resource-constrained project scheduling problem. *IIE Transactions*, 27(3):574–586.

Papadimitriou, C. and Steiglitz, K. (1982). *Combinatorial Optimization*. Dover Publications, Inc., Mineola, New York.

Pich, M. T., Loch, C. H., and De Meyer, A. (2002). On uncertainty, ambiguity, and complexity in project management. *Management Science*, 48(8):1008–1023.

Pickavance, K. (2000). *Delay and Disruption in Construction Contracts*. LLP Professional Publishing, London, UK.

PMI (2000). *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Project Management Institute, Inc, Newtown Square, PA.

Porteus, E. (1990). *Stochastic inventory theory*. D.P. Heyman and M.J. Sobel (eds): Handbooks in operations research and management science, Volume 2: Stochastic models. Elsevier Publishers, Amsterdam.

Pritsker, A., Watters, L., and Wolfe, P. (1969). Multiproject scheduling with limited resources: a zero-one programming approach. *Management Science*, 16(1):93–108.

Qi, X., Bard, J. F., and Yu, G. (2002a). Disruption management for machine scheduling. *Working paper, the University of Texas at Austin*.

Qi, X., Bard, J. F., and Yu, G. (2002b). Supply chain coordination with demand disruptions. *Working paper, Department of Management Science and Information Systems, The University of Texas, Austin*.

Rodošek, R., Wallace, M. G., and Hajian, M. T. (1999). A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research*, 86:63–87.

Salewski, F., Schirmer, A., and Drexl, A. (1997). Project scheduling under resource and mode identity constraints: Model, complexity, methods and application. *European Journal of Operational Research*, 102(1):88–110.

Schirmer, A. (2000). Case-based reasoning and improved adaptive search for project scheduling. *Naval Research Logistics*, 47(3):201–222.

Sourd, F. and Kedad-Sidhoum, S. (2003). The one-machine problem with earliness and tardiness penalties. *Journal of Scheduling*, 6(6):533–549.

Sprecher, A. and Drexl, A. (1998). Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. *European Journal of Operational Research*, 107(2):431–450.

Thengvall, B. G., Bard, J. F., and Yu, G. (2001). Multiple fleet aircraft schedule recovery following hub closures. *Transportation Research Part A*, 35:289–308.

Thomas, P. and Salhi, S. (1998). A tabu search approach for the resource constrained project scheduling problem. *Journal of Heuristics*, 4(2):123–139.

Valls, V., Laguna, M., Lino, P., Perez, A., and Quintanilla, S. (1998). *Project Scheduling with Stochastic Activity Interruptions*. Jan Weglarz (Eds): Project Scheduling: Recent Models, Algorithms and Applications. Kluwer Academic Publishers.

van den Akker, J., Hurkens, C., and Savelsbergh, M. W. (2000). Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124.

Vanhoucke, M., Demeulemeester, E., and Herroelen, W. (2001). An exact procedure for the resource-constrained weighted earlinesstardiness project scheduling problem. *Annals of Operations Research*, 102(1-4):179–196.

Williams, T. (2003). Assessing extension of time delays on major projects. *International Journal of Project Management*, 21:19–26.

Williams, T. and Eden, C. (2000). The effects of design changes and delays on project costs. *Journal of the Operational Research Society*, 46:809–818.

Wolsey, L. (1990). Valid inequalities for 0-1 knapsack and MIPs with generalized upper bound constraints. *Discrete Applied Mathematics*, 29:251–261.

Yang, J., Qi, X., and Yu, G. (2002). Disruption management in production planning. *Working paper, Department of Management Science and Information Systems, The University of Texas, Austin.*

Yu, G., Argüello, M., Song, G., McCowan, S. M., and White, A. (2003). A new era for crew recovery at continental airlines. *Interfaces*, 33(1):5–22.

Yu, G. and Qi, X. (2004). *Disruption Management: Framework, Models, Solutions and Applica- tions.* World Scientific Publishers, Singapore.

Zhu, G., Bard, J. F., and Yu, G. (2003). A branch-and-cut procedure for multi-mode resource constraint project scheduling problem. *INFORMS Journal on Computing.*

Vita

Guidong Zhu was born in Tianchang, Anhui, China on Feburary 1971. He received a Bachelor and a Doctor degree in Engineering Mechanics from the Harbin Institute of Technology in 1992 and 1997, respectively. From 1997 to 1999, he worked as an engineer in Beijing Institute of Astronautical Systems Engineering. He started pursuing a Ph.D. degree in Management Science at The University of Texas at Austin in August, 2000. He is married to Hui Niu, and has one daughter, Helen.

Permanent address: 2501 Lake Austin Blvd, Apt F107
Austin, Texas 78703

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.