MODELS AND METHODS FOR MULTIPLE RESOURCE CONSTRAINED JOB SCHEDULING UNDER UNCERTAINTY

by

Brian Daniel Keller

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF SYSTEMS AND INDUSTRIAL ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2009

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Brian Daniel Keller entitled Models and Methods for Multiple Resource Constrained Job Scheduling under Uncertainty and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Güzin Bayraksan — Date: 20 May 2009

Larry Head — Date: 20 May 2009

Simge Küçükyavuz — Date: 20 May 2009

Ken Baker — Date: 20 May 2009

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.
I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dissertation Director: Güzin Bayraksan — Date: 20 May 2009

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Brian Daniel Keller

ACKNOWLEDGEMENTS

It was a privilege working with my advisor, Dr. Güzin Bayraksan. I am grateful for her guidance and support throughout my studies and for believing in me when times were tough. I am indebted to my committee for their mentoring and for their comments and suggestions on the dissertation. Thank you also to my friends in the department for your encouragement, especially my officemates over the years: Rashid, Maria, John, Carrie, and Matt. Finally, I have greatly appreciated the support from my family – Mom, Dad, Jeff, Kevin, and Faith.

DEDICATION

I dedicate this dissertation to my wife, Faith, for her constant support, encouragement, and love. The process would have been infinitely more difficult without you.

TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

We consider a scheduling problem where each job requires multiple classes of resources, which we refer to as the *multiple resource constrained scheduling problem* (MRCSP). Potential applications include team scheduling problems that arise in service industries such as consulting and operating room scheduling. We focus on two general cases of the problem. The first case considers uncertainty of processing times, due dates, and resource availabilities consumption, which we denote as the stochastic MRCSP with uncertain parameters (SMRCSP-U). The second case considers uncertainty in the number of jobs to schedule, which arises in consulting and defense contracting when companies bid on future contracts but may or may not win the bid. We call this problem the stochastic MRCSP with job bidding (SMRCSP-JB).

We first provide formulations of each problem under the framework of two-stage stochastic programming with recourse. We then develop solution methodologies for both problems. For the SMRCSP-U, we develop an exact solution method based on the L-shaped method for problems with a moderate number of scenarios. Several algorithmic enhancements are added to improve efficiency. Then, we embed the L-shaped method within a sampling-based solution method for problems with a large number of scenarios. We modify a sequential sampling procedure to allow for approximate solution of integer programs and prove desired properties. The sampling-based method is applicable to two-stage stochastic integer programs with integer first-stage variables. Finally, we compare the solution methodologies on a set of test problems.

For SMRCSP-JB, we utilize the disjunctive decomposition (D^2) algorithm for stochastic integer programs with mixed-binary subproblems. We develop several enhancements to the D^2 algorithm. First, we explore the use of a cut generation

problem restricted to a subspace of the variables, which yields significant computational savings. Then, we examine generating alternative disjunctive cuts based on the generalized upper bound (GUB) constraints that appear in the second-stage of the SMRCSP-JB. We establish convergence of all D^2 variants and present computational results on a set of instances of SMRCSP-JB.

CHAPTER 1

INTRODUCTION

Utilization of resources is a key component of profitability for any business. As many companies are facing more competition both domestically and internationally, it is essential for companies to manage their resources in the most efficient way. In domains such as revenue management, vehicle routing, and capital budgeting, optimization has played a role in improving resource utilization. In this dissertation, we apply optimization techniques to improve utilization of resources when scheduling jobs that share multiple common resources. We refer to this problem as the *multiple resource-constrained scheduling problem* (MRCSP) and consider several uncertainties that are present in the system.

1.1 Motivation for Research

This dissertation is concerned with developing models and solution methods for stochastic multiple resource-constrained scheduling (SMRCSP). The SMRCSP arises in many business and service environments. Below, we provide a few examples with various sources of uncertainty.

Example 1.1.1. (Team Scheduling) *A large-scale engineering and manufacturing company employs around 9,000 engineers and works on about 3,000 projects at any given time. Teams of different types of engineers are assigned to each project. The amount of time required to complete each project may be uncertain but may be modeled with a probability distribution. Further, while the company is working on current projects, they are also bidding on future contracts that they may or may not win. The company would like to maximize the number of projects they complete within the year given financial limitations and staff levels.*

Example 1.1.2. (Operating Room Scheduling) *A hospital must perform a set of surgeries but has a limited number of operating rooms. Each surgery requires different types of doctors, nurses, anesthesiologists, and special equipment. The length of each surgery may be uncertain and patients requiring surgeries may unexpectedly arrive from the emergency room. The hospital would like to maximize the utilization of resources to lower its operating costs.*

Example 1.1.3. (Construction Management) *A regional construction company specializes in excavation for different types of construction projects and works on multiple projects at any given time. Different projects require different types of equipment and special operators. The company can subcontract some of the equipment and operators if they do not have enough. Project duration, costs, and equipment availability may be uncertain. The company would like to complete its projects on time to avoid contractual penalties while minimizing its costs due to subcontractors.*

The aim of each of the above applications is to schedule a number of "jobs" that share "multiple resources". The jobs are engineering projects in Example 1.1.1, surgeries in example 1.1.2, and excavation projects in Example 1.1.3. The multiple resources represent people with different skill sets, specialized equipment, budget, or the operating rooms. We note that other applications of SMRCSP are possible and each application can have its own special side constraints, but in this dissertation we focus on a general model. The above and other examples exhibit many inherent uncertainties:

- The time required to complete each job can be uncertain.
- The amount of resources required by a job (e.g., budget, manpower, etc.) is often estimated and is uncertain.
- The amount of available resources can be uncertain in situations where equipment is subject to failure or when the delivery date of materials is uncertain.
- The number of jobs to be scheduled can be uncertain. In many applications, companies bid on future contracts without knowing the outcome of the bids.

In operating room scheduling, the number of surgeries to be performed can be uncertain due to emergency room arrivals.

Clearly, both uncertainty and multiple resources must be accounted for if one wishes to create models and methods that can actually be implemented. The above examples are similar to project scheduling, which ensures that certain jobs are completed before others begin. However, this dissertation focuses on scheduling jobs that are independent of each other except for the shared resources. In other words, the models considered in this dissertation do not have precedence constraints.

We focus on two general cases of the problem. The first case considers uncertainty of processing times, due dates, resource availability, and resource consumption, which we denote as the stochastic MRCSP with uncertain parameters (SMRCSP-U). The second case considers uncertainty in the number of jobs to schedule, which arises in consulting and defense contracting when companies bid on future contracts but may or may not win the bid. We call this problem the stochastic MRCSP with job bidding (SMRCSP-JB). In both models, resource expansion is allowed for a certain cost. This is equivalent to hiring temporary workers, renting additional equipment, etc.

Throughout the dissertation we will use the term *scenario* to describe a single outcome from the possible set of uncertain future outcomes. For example, consider a scheduling problem with one job with two possible processing times of length 4 and 8 and a single resource with either 3, 5, or 7 units available. Assuming independence between processing time and resource availability, there are 6 possible (processing time, resource availability) scenarios: (4,3), (4,5), (4,7), (8,3), (8,5), and (8,7). This example assumed independence, but independence is not required in general.

1.2 Organization of Dissertation

The remainder of the dissertation is organized as follows. In Chapter 2, we review the relevant literature. The SMRCSP models we study involve concepts from stochastic integer programming and scheduling. As such, we first review integer programming

and stochastic programming, and then tie the two disciplines together in a review of stochastic integer programming, which will be used extensively in this dissertation. We then review relevant scheduling literature. In an attempt to keep the literature review tractable, we first review machine scheduling and then project scheduling, a problem closely related to those studied in this dissertation. We discuss both deterministic and stochastic variants of these problems and briefly discuss general stochastic scheduling paradigms.

Chapter 3 introduces the time-indexed modeling framework shared by SMRCSP-U and SMRCSP-JB. These types of models handle scheduling decisions via binary variables that take the value 1 if the job is started in a given time period and 0 otherwise. This type of modeling allows the use of many common objectives found in the scheduling literature without causing any change in the problem structure. However, the size of the model can quickly grow large. We present and discuss the model formulations in Chapter 3.

In Chapter 4, we solve the SMRCSP-U formulated as a two-stage stochastic program with binary first-stage variables. First, we develop an exact solution method based on the L-shaped method for problems with a moderate number of scenarios. Several algorithmic enhancements are added to improve efficiency. Then, we embed the L-shaped method within a sampling-based solution method for problems with a large number of scenarios. We modify a sequential sampling procedure to allow for approximate solution of integer programs and prove desired properties. Finally, we compare the solution methodologies on a set of test problems.

Chapter 5 considers the solution of SMRCSP-JB. Due to integer variables in the second-stage, we utilize the disjunctive decomposition (D^2) algorithm for stochastic integer programs with mixed-binary second-stage decisions. The D^2 algorithm extends the L-shaped method by adding disjunctive cutting planes to subproblems. We develop computational enhancements that are applicable to SMRCSP-JB and other types of problems. First, we explore the use of a cut generation problem restricted to a subspace of the variables, which yields significant computational savings. Then, we examine generating alternative disjunctive cuts based on the generalized upper

bound (GUB) constraints that appear in the second-stage of the SMRCSP-JB. We establish convergence of all D^2 variants and present computational results on a set of test problems.

We conclude the dissertation in Chapter 6 by providing a summary of contributions and suggestions for future research directions.

CHAPTER 2

LITERATURE REVIEW

This chapter reviews literature related to the topics discussed in this dissertation. We begin in Section 2.1 with relevant topics in mathematical programming. An overview of mixed-integer programming, stochastic programming, and stochastic integer programming is presented with emphasis on concepts related to this dissertation. In Section 2.2, we briefly review the scheduling literature focusing on deterministic and stochastic machine scheduling and deterministic and stochastic project scheduling.

2.1 Relevant Mathematical Programming Literature Review

First, we review mixed-integer programming and stochastic programming. Then, the two concepts are tied together in our review of stochastic integer programming, which will be used extensively in this dissertation.

2.1.1 Mixed-Integer Programming

Many real-world optimization problems involve integer variables. A few examples of classic integer programming problems include the knapsack problem and its variations (Martello and Toth, 1990), traveling salesman problem (Lawler et al., 1985), facility location problems (Mirchandani and Francis, 1990), network flows (Ahuja et al., 1993), network routing problems (Ball et al., 1995), production and inventory problems (Graves et al., 1993), and scheduling problems (Pinedo, 2008; Schultz,

1996). The general mixed-integer program (MIP) can be written as

$$\min_{x} \quad cx \tag{2.1a}$$
$$\text{s.t.} \quad Ax \geq b, \tag{2.1b}$$
$$x_j \in \{0,1\}, \qquad j \in B, \tag{2.1c}$$
$$x_j \text{ integer}, \qquad j \in I, \tag{2.1d}$$
$$x_j \geq 0, \qquad j \in J, \tag{2.1e}$$

where $c^{\mathrm{T}} \in \mathbb{R}^{d_x}$, $A \in \mathbb{R}^{m \times d_x}$, and $b \in \mathbb{R}^m$ are known parameters, and B is the set of binary variables, I is the set of general integer variables, and $J \supseteq (B \cup I)$ is the set of all variables with $|J| = d_x$, the dimension of the decision vector x.

2.1.1.1 Solution Approaches

Mixed-integer programs can be very challenging to solve and, hence, many solution approaches have been developed. The approaches can be briefly categorized as branch-and-bound, decomposition-based methods such as Benders decomposition and Dantzig-Wolfe, and cutting plane or polyhedral-based methods. These methods can be combined to solve MIPs and sometimes heuristics are used to obtain solutions quickly. In this dissertation, we make use of Benders decomposition and cutting plane methods to solve our stochastic scheduling problems. Below, we briefly review these approaches, providing some additional details on polyhedral methods and specifically on disjunctive cuts. We leave the discussion on Benders decomposition (referred to as the L-shaped method in the stochastic programming literature) to Section 2.1.2.

Most solution approaches in some way incorporate *branch-and-bound (B&B)*, in which a series of smaller, easier problems are solved and information from each problem is recombined to solve the original problem. The smaller, easier problems are formed as linear programming (LP) relaxations $\min\{cx : Ax \geq b, x \geq 0\}$ intersected with bounds on the integer variables. The different bounds on integer variables

partition the search space into finer and finer partitions, forming upper and lower bounds on the optimal integer objective. The partitioning continues until the upper and lower bounds are equal or some other termination criteria is met. An in-depth description of B&B can be found in (Nemhauser and Wolsey, 1988). Land and Doig (1960) were the first to present a B&B algorithm for integer programming, and Little et al. (1963) successfully applied the B&B algorithm to the traveling salesman problem. Another fundamental solution approach, often combined with B&B, involves generating constraints to cut off fractional solutions without eliminating any integer feasible solutions. Dantzig et al. (1954) introduced the so-called *cutting plane algorithm* in the context of solving the traveling salesman problem. We review the cutting plane algorithm in more detail later. Combining cutting plane algorithms with B&B results in the *branch-and-cut algorithm*, which is at the heart of many commercial MIP solvers.

Decomposition-based methods include Lagrangian relaxation, Dantzig-Wolfe decomposition, and Benders decomposition and are effective for solving MIPs with a large number of variables or constraints. The choice of which type of decomposition to use depends on problem structure. *Lagrangian relaxation*, first developed by Held and Karp (1970, 1971), considers MIPs with a set of "easy" constraints and a set of "complicating" constraints. Constraints are easy in the sense that an MIP with only easy constraints is easily solved (e.g., network constraints) . The MIP takes the form

$$\begin{aligned} z_{MIP} = \min_{x} \quad & cx \\ \text{s.t.} \quad & Dx \leq d, \\ & x \in X, \end{aligned}$$

where X is a feasible region defined by easy constraints and $Dx \leq d$ are the complicating constraints. We can compute a lower bound on z_{MIP} by solving

$$z(u) = \min_{x} \{cx + u(Dx - d) : x \in X\}.$$

The complicating constraints are added to the objective and u is the "price" or

Lagrangian multiplier associated with the complicating constraints. The tightest lower bound on z_{MIP} is found by solving

$$z_{LD} = \max_{u}\{z(u) : u \geq 0\},$$

which is known as the Lagrangian dual. In general $z_{MIP} \geq z_{LD}$; however, equality holds under certain conditions; see (Wolsey, 1998) for a more thorough discussion. Methods for solving the Lagrangian dual include the outer-linearization technique in which piecewise linear approximations are used to define $z(u)$ and subgradient optimization procedures in which u is iteratively updated in a way to guarantee convergence to its optimal value. Bazaraa et al. (1993) provide detailed descriptions of both methods. Often convergence of z_{LD} is quite slow as both methods suffer from the *tailing off* effect in which little or no improvement is found in z_{LD} from one iteration to the next. However, Lagrangian decomposition has been successfully embedded in branch and bound algorithms because it can quickly find improved lower bounds at nodes in the search tree. Other Lagrangian decomposition schemes such as variable splitting, can be used depending on problem structure (Wolsey, 1998).

Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960) is suitable when the number of columns in the problem is quite large. Often for sizable problems, most columns never enter the basis and therefore, those columns do not need to be present in the problem. The Dantzig-Wolfe algorithm starts with a *master problem* containing a small subset of columns from the original problem. At each iteration, the master problem is solved and a variable x_i with negative reduced cost is added to the master problem along with its associated column. This process continues until there are no variables with negative reduced cost to be added. For many problems, new columns can be found by an optimization problem called the *pricing problem*. The objective of the pricing problem is to "price in" a new column by minimizing the reduced cost over all possible columns not yet added to the master. A negative objective indicates that the pricing problem has generated a new column that

should be added. A nonnegative objective indicates that no more columns need to be added to the master problem and that the optimal solution has been found. A classic application of Dantzig-Wolfe decomposition is to the cutting stock problem (Gilmore and Gomery, 1961, 1963). Dantzig-Wolfe decomposition can be embedded into a B&B framework resulting in a class of column generation algorithms referred to as *branch-and-price*.

Benders decomposition, introduced by Benders (1962), can be thought of as the dual of Dantzig-Wolfe decomposition. Instead of generating columns when necessary, Benders decomposition generates rows when necessary. Benders decomposition is quite effective on problems with a *block-angular* structure. We review Benders decomposition more thoroughly in the context of stochastic programming in Section 2.1.2. The stochastic scheduling models studied in this dissertation have the block-angular structure and extensions of Benders decomposition are used extensively in Chapters 4 and 5.

Polyhedral methods study the polyhedral structure of feasible regions of MIPs and add constraints to the LP relaxation of (2.1) that cut off fractional solutions but do not eliminate any integer solutions. Theoretically, we can add enough cutting planes to the LP relaxation of (2.1) so that all extreme point solutions of the LP relaxation will be integer, thereby obtaining the convex hull of the feasible region. However, in practice, there might be exponentially many such cutting planes, and not all cutting planes may have an explicit characterization. Polyhedral methods aim to effectively approximate the convex hull of feasible regions of MIPs. We first define some relevant terminology and provide an overview of a general cutting plane algorithm. Finally, we describe a family of cuts that we use extensively in Chapter 5.

Consider the MIP (2.1) and let X be the feasible region defined by the m constraints (2.1b) and define the d_x-dimensional polyhedron $P = \{x \in X : x \geq 0\}$. The following definitions can be found in (Wolsey, 1998).

Definition 2.1.1. *An inequality* $\pi x \geq \pi_0$ *is a* valid inequality *for* $X \subseteq \mathbb{R}^{d_x}$ *if* $\pi x \geq \pi_0$ *for all* $x \in X$.

Definition 2.1.2. *If $\pi x \geq \pi_0$ and $\mu x \geq \mu_0$ are two valid inequalities for P, $\pi x \geq \pi_0$* dominates *$\mu x \geq \mu_0$ if there exists $u > 0$ such that $\pi \leq u\mu$ and $\pi_0 \geq u\mu_0$, and $(\pi, \pi_0) \neq (u\mu, u\mu_0)$.*

Definition 2.1.3. *The points $x^1, \ldots, x^k \in \mathbb{R}^{d_x}$ are* affinely independent *if the $k-1$ directions $x^2 - x^1, \ldots, x^k - x^1$ are linearly independent, or alternatively the k vectors $(x^1, 1), \ldots, (x^k, 1)$ are linearly independent.*

Definition 2.1.4. *The* dimension *of P, denoted dim(P), is one less then the maximum number of affinely independent points in P.*

Definition 2.1.5. *If P is a* full dimensional *polyhedron, it has a unique minimal description $P = \{x \in \mathbb{R}^{d_x}_+ : a^i x \geq b^i \text{ for } i = 1, \ldots, m\}$, where each inequality is unique to a positive multiple. Alternatively, P is* full dimensional *if and only if $dim(P) = d_x$.*

Definition 2.1.6.
(i) F defines a face *of the polyhedron P if $F = \{x \in P : \pi x = \pi_0\}$ for some valid inequality $\pi x \geq \pi_0$ of P.*
(ii) F is a facet *of P if F is a face of P and $dim(F) = dim(P) - 1$.*

Facets (definition 2.1.6(ii)) are the strongest cuts that can be generated. If P is full-dimensional (definition 2.1.5), a valid inequality $\pi x \geq \pi_0$ is necessary in the description of P if and only if it defines a facet of P. In Chapter 5, we generate valid inequalities to strengthen the LP relaxation of our scheduling problem. Under certain conditions, the cuts we generate are facet-defining and, theoretically, we can generate the convex hull of our problem, if necessary (see Chapter 5 for details).

The general cutting plane algorithm solves the LP relaxation of the MIP, generates a cut, adds the cut to the LP relaxation, and resolves the LP. This process continues until the MIP is solved or some other termination criteria is met. The process of generating the cut is often referred to as the *separation problem* since generating the cut corresponds to generating a hyperplane that separates the current LP solution from the set of feasible solutions of the MIP. Often, the generated cut

contains only a subset of the variables in the problem. The cut can be strengthened considerably, sometimes into a facet, by a process called *lifting* in which variables are added to the cut to lift it into a higher dimension. We utilize the concept of lifting in Chapter 5 when generating cuts for one of our stochastic scheduling problems.

MIPs are generally quite difficult to solve. Although many methods have been developed to solve MIPs, sometimes one must resort to *heuristic methods* to generate good solutions. Two elementary and frequently used heuristics include greedy search and local search (see Wolsey, 1998, for an overview). However, these algorithms are susceptible to getting stuck in local optimal solutions. Tabu search (Glover, 1989, 1990), simulated annealing (Kirkpatrick et al., 1983), and genetic algorithms (Holland, 1975; Goldberg, 1989) use a combination of greedy and local search as subroutines within a more complex algorithmic structure and can escape from local optima. The LP-based relax-and-fix heuristic solves the LP relaxation of (2.1), rounds some or all of the fractional integer variables to integer values, and resolves the remaining problem. In Chapters 4 and 5, we use a variation of the relax-and-fix heuristic to find good starting solutions to our algorithms.

2.1.1.2 Disjunctive Cuts

Here, we describe in detail a class of cuts called *disjunctive cuts* as we will be using them to solve one of our stochastic scheduling problems. Disjunctive cuts approximate the convex hull of disjunctive sets of the form

$$S = \cup_{h \in H} S_h \tag{2.2}$$

where H is a finite index set and the sets S_h are polyhedrons of the form

$$S_h = \{x : A_h x \geq b_h, x \geq 0\}. \tag{2.3}$$

Disjunctive sets are useful in modeling logical conditions such as "either A or B is true" or "one of the following is true…". The most simple and general disjunction

found in MIPs is the disjunction $\{x_j \leq 0\} \cup \{x_j \geq 1\}$ for some $j \in B$. In fact, any solution to a 0-1 MIP can be written as the union of polyhedra. However, $|H|$ can be exponentially large and an explicit representation would be computationally intractable. Instead, we can generate a convex relaxation of the nonconvex set S using valid disjunctive inequalities. The following result is known as the disjunctive cut principle and is due to Balas (1975) and Blair and Jeroslow (1978).

Theorem 2.1.7. *Let S and S_h, $h \in H$, be defined as in (2.2) and (2.3), respectively, and let the column vector A_{hj} denote the j^{th} column of the matrix A_h. If $\lambda_h \geq 0$ for all $h \in H$, then*

$$\sum_j \{\max_{h \in H} \lambda_h A_{hj}\} x_j \geq \min_{h \in H} \lambda_h b_h \tag{2.4}$$

is a valid inequality for S. Conversely, suppose that $\pi x \geq \pi_0$ is a valid inequality and $H^* = \{h \in H | S_h \neq \emptyset\}$. Then, there exists nonnegative $\{\lambda_h\}_{h \in H^*}$ such that

$$\pi_j \geq \max_{h \in H^*} \lambda_h A_{hj} \text{ and } \pi_0 \leq \min_{h \in H^*} \lambda_h b_h. \tag{2.5}$$

The latter part of Theorem 2.1.7 explains how to choose cut coefficients for the disjunctive cut. If $\bar{x}$ is not in S, then there exists some (π, π_0) such that $\pi \bar{x} \geq \pi_0$ is violated. We can generate the deepest cut, in the sense of Euclidean distance, by minimizing $\pi \bar{x} - \pi_0$ subject to (2.5).

For example, consider the MIP given in (2.1) and its LP relaxation i.e., (2.1) without constraints (2.1c) and (2.1d). Suppose $I = \emptyset$ and we solve the LP relaxation of (2.1) and find a solution $\bar{x}$ fractional for some $j^* \in B$. Let $\tilde{A}$ and $\tilde{b}$ denote the matrix A and vector b with constraints $x_j \leq 1$ for $j \in B$ appended. We can form a disjunction

$$S = S_0 \cup S_1,$$

where

$$S_0 = \{x \geq 0 : \tilde{A}x \geq \tilde{b}, \tag{2.6a}$$

$$-x_{j^*} \geq 0\}, \tag{2.6b}$$

$$S_1 = \{x \geq 0 : \tilde{A}x \geq \tilde{b}, \tag{2.6c}$$

$$x_{j^*} \geq 1\} \tag{2.6d}$$

are used.

Let λ_{01} denote the vector of multipliers associated with the right-hand-side constraints in (2.6a) and λ_{02} the scalar multiplier associated with (2.6b). Similarly, let λ_{11} and λ_{12} denote the multipliers associated with (2.6c) and (2.6d), respectively. Define

$$I_{j^*} = \begin{cases} 1, & \text{if } j = j^* \\ 0, & \text{otherwise.} \end{cases}$$

The following separation problem determines a disjunctive cut such that $\pi\bar{x} < \pi_0$ and cuts off the current point $\bar{x}$:

$$\min_{\pi,\pi_0,\lambda} \quad \pi\bar{x} - \pi_0 \tag{2.7a}$$

$$\text{s.t.} \quad \pi_j \geq \lambda_{01}\tilde{A}_j - I_{j^*}\lambda_{02}, \qquad j \in J, \tag{2.7b}$$

$$\pi_j \geq \lambda_{11}\tilde{A}_j + I_{j^*}\lambda_{12}, \qquad j \in J, \tag{2.7c}$$

$$\pi_0 \leq \lambda_{01}\tilde{b}, \tag{2.7d}$$

$$\pi_0 \leq \lambda_{11}\tilde{b} + \lambda_{12}, \tag{2.7e}$$

$$-1 \leq \pi_j \leq 1, \qquad j \in J, \tag{2.7f}$$

$$-1 \leq \pi_0 \leq 1, \tag{2.7g}$$

$$\lambda_{01}, \lambda_{02}, \lambda_{11}, \lambda_{12} \geq 0, \tag{2.7h}$$

where $\tilde{A}_j$ denotes the j^{th} column of matrix $\tilde{A}$. Constraints (2.7f) and (2.7g) are normalization constraints that ensure (2.7) is bounded. There are other possible

normalizations and many are discussed in (Balas and Perregaard, 2002). We will refer to the separation problem for disjunctive cuts as the *cut generation LP* (CGLP).

Balas (1979) has shown that extreme point solutions to CGLP correspond to facets of the closure of the convex hull of S. For our example above, assuming (π, π_0) is an extreme point solution to CGLP, the generated cut corresponds to a facet of the closure of the convex hull of

$$\begin{bmatrix} \tilde{A}x & \geq & b \\ x_{j^*} & \leq & 0 \end{bmatrix} \bigcup \begin{bmatrix} \tilde{A}x & \geq & b \\ x_{j^*} & \geq & 1 \end{bmatrix}.$$

2.1.2 Stochastic Programming

In Section 2.1.1 we assumed that input parameters to the optimization problem c, A, and b are known with certainty. Often, however, input data such as demand, price, and costs can only be described with a probability distribution. Stochastic programming extends deterministic mathematical programming by incorporating the probabilistic information into the model. We first review two main classes of stochastic programs and then discuss some solutions methods.

Chance-constrained problems and *recourse problems* are the two most common stochastic programs. Chance-constrained programming, introduced by Charnes and Cooper (1959, 1963), handles uncertainty with constraints that must be satisfied with a certain probability. Suppose the parameters A and b in (2.1) are uncertain. We denote them by a random vector $\tilde{\omega} = (A(\tilde{w}), b(\tilde{w}))$ with a known distribution. Chance constraints take the form

$$\Pr\{A(\tilde{\omega})x \geq b(\tilde{\omega})\} \geq p,$$

where Pr denotes the probability measure and $0 < p < 1$ is the desired probability level for the constraint to hold. An example of a chance-constraint is choosing a product inventory level that will satisfy demand 95% of the time. Chance-constrained programs have appeared in applications such as water resource manage-

ment, telecommunications, energy production, and finance (Prékopa, 2003). Despite theoretical progress and practical importance, chance-constrained programs remain largely intractable. One promising solution approach involves using binary variables that take the value 1 when the constraint is satisfied and 0 otherwise. Recasting the problem in this manner and using sampling when necessary allows it to be solved as an MIP (Ahmed and Shapiro, 2008; Küçükyavuz, 2009).

Stochastic programs with recourse are another common class of stochastic programs (Dantzig, 1955; Beale, 1955). In these types of models, an initial decision is made before uncertainty emerges and recourse actions are allowed after the uncertainty is disclosed. The goal is to take all future outcomes into consideration while making the initial decision so as to hedge against the uncertain future. Stochastic programs with recourse can be multi-stage, where recourse decisions are made at multiple future times, or can be two-stage, where recourse decisions are made at only one time in the future. The models considered in this dissertation are two-stage models (see Chapter 3). Below, we review two-stage stochastic linear programming. For more details on multi-stage models, we refer the reader to (Birge and Louveaux, 1997).

The general two-stage model considered is of the form

$$\begin{aligned} \min_{x,y(\omega)} \quad & cx + E[q(\omega)y(\omega)] & & \\ \text{s.t.} \quad & Ax & = b, & \\ & T(\omega)x + W(\omega)y(\omega) & = r(\omega), & \omega \in \Omega, \\ & x & \geq 0, & \\ & y(\omega) & \geq 0, & \omega \in \Omega, \end{aligned} \tag{2.8}$$

where Ω is the set of realizations of $\tilde{\omega}$, ω is a realization of $\tilde{\omega}$, or equivalently, ω denotes a scenario, E is the expectation operator taken with respect to the distribution of $\tilde{\omega} = (q(\tilde{\omega}), T(\tilde{\omega}), W(\tilde{\omega}), r(\tilde{\omega}))$, x are the first-stage decisions, and $y(\omega)$ are the second-stage decisions for each $\omega \in \Omega$. The distribution of $\tilde{\omega}$ is assumed known and does not depend on x. In the stochastic programming literature, the matrix

$W(\omega)$ is known as the *recourse matrix* and (2.8) is said to have *fixed recourse* if $W(\omega) = W$ for all $\omega \in \Omega$. The matrix $T(\omega)$ is known as the *technology matrix* and the vector $\chi(\omega) = T(\omega)x$ is called the *tender*. If for a given x, $\chi(\omega) = \chi$ for all $\omega \in \Omega$, then (2.8) is said to have *fixed tenders*.

Often, (2.8) is equivalently written as

$$\begin{aligned} \min_{x} \quad & cx + E[f(x,\omega)] \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0, \end{aligned}$$

where (2.9)

$$\begin{aligned} f(x,\omega) = \min_{y} \quad & q(\omega)y \\ \text{s.t.} \quad & W(\omega)y = r(\omega) - T(\omega)x, \\ & y \geq 0. \end{aligned}$$

This representation highlights the dependence of the second-stage decisions y and second-stage value function $f(x,\omega)$ on x. As x changes, not only does $f(x,\omega)$ change but also the feasible region for y. Although second-stage decisions y depend on ω, the ω is often suppressed in notation.

2.1.2.1 Solution Approaches

Stochastic programming algorithms usually fall into one of three categories: exact solution procedures, approximation and bounding schemes, and sampling-based methods. *Exact solution procedures* consider all $\omega \in \Omega$ during the optimization. Algorithms include direct solution methods that exploit special structures (Kall, 1979; Lustig et al., 1991), decomposition based methods such as the L-shaped method (van Slyke and Wets, 1969), and the Lagrangian relaxation-based progressive hedging algorithm (Rockafellar and Wets, 1991). Exact solution procedures are effective when the number of scenarios is small and become more computationally demanding as the number of scenarios grow.

Approximation and bounding algorithms calculate upper and lower bounds on the

optimal objective function value. As the algorithm progresses, the approximation becomes finer and the bounds improve. Deterministic upper and lower bounds are often calculated using the Jensen and Edmundson-Madansky bounds (Birge and Wets, 1985, 1989). Such methods are referred to as *sequential approximation methods*. These bounds are progressively tightened by finer partitioning of Ω.

Sampling-based algorithms approximate $E[f(x,\omega)]$ with the *sample average approximation*, $\frac{1}{n}\sum_{l=1}^{n} f(x,\omega^l)$, where ω^l is the l^{th} observation from a sample of $\tilde{\omega}$. They are particularly useful when the number of scenarios is quite large or possibly infinite as in the case of continuous distributions. Sampling-based algorithms can be classified as internal and external sampling methods. Internal sampling-based algorithms are adaptations of deterministic optimization algorithms where gradients and subgradients are estimated via sampling. Then include include stochastic quasigradient methods (Ermoliev, 1983) and the stochastic cutting plane algorithm of Higle and Sen (1991) called stochastic decomposition. External sampling methods solve sample average approximations of (2.8). The asymptotic properties of optimal solutions and objective function values of sample average approximation problems have been extensively studied in the literature. See Shapiro (2003) for a survey. In the last ten years, much progress has been made in assessing solution quality of a sampling solution (Mak et al., 1999; Bayraksan and Morton, 2006). Bayraksan and Morton (2009) introduced a sequential sampling procedure that specifies the solution quality a priori and stops the algorithm once the desired solution quality is reached. Chapter 4 modifies the sequential sampling algorithm to allow for approximate solutions of stochastic integer programs with integer first-stage variables.

2.1.2.2 L-shaped Method

The solution procedures in Chapters 4 and 5 make extensive use of the L-shaped method and extensions, and we provide a review of the algorithm here. If $|\Omega| = L$ or if we solve a sampling problem with L samples $\omega^1, \omega^2, \ldots, \omega^L$, we can think of the two-stage stochastic LP as one large linear program. For clarity of exposition, let the subscript l denote dependence on scenario ω^l and let $\Pr_l$ denote the probability

of scenario ω^l. We can rewrite formulations (2.8) or (2.9) as

$$\begin{array}{lllllll}
\min\limits_{x,y_l} & cx + \Pr_1 q_1 y_1 + \Pr_2 q_2 y_2 + \ \ldots \ + \Pr_L q_L y_L & & & & & \\
\text{s.t.} & Ax & & & & & = b, \\
& T_1 x + & W_1 y_1 & & & & = r_1, \\
& T_2 x & & + & W_2 y_2 & & = r_2, \\
& \vdots & & & & \ddots & \vdots \\
& T_L x & & & & + & W_L y_L = r_L, \\
& x, & y_1, & & y_2, & \ldots, & y_L \geq 0.
\end{array}$$

The above formulation has a special *block-angular* structure that makes decomposition possible. The L-shaped method is based on the idea that once x is determined, the problem can be decomposed into L LP subproblems. The subproblems take the form

$$\begin{array}{rrl}
z_l(x) = \min\limits_{y_l} & q_l y_l & \\
\text{s.t.} & W_l y_l & = r_l - T_l x, \\
& y_l & \geq 0.
\end{array} \tag{2.10}$$

Now, to optimize x, we must solve the master problem

$$\begin{array}{rrl}
\min\limits_{x} & cx + \sum\limits_{l=1}^{L} \Pr_l z_l(x) & \\
\text{s.t.} & Ax & = b, \\
& x & \geq 0.
\end{array} \tag{2.11}$$

To solve (2.11), we replace the functions $z_l(x)$ with piecewise linear approximations constructed from dual information from the subproblems. The dual of subproblem l can be written as

$$\begin{array}{rrl}
z_l^D(x) = \max\limits_{\alpha_l} & \alpha_l \left(r_l - T_l x\right) & \\
\text{s.t.} & \alpha_l W_l & \leq q_l,
\end{array} \tag{2.12}$$

where α_l is the vector of dual variables corresponding to constraints (2.12) for each $l = 1, \ldots, L$. If (2.10) is infeasible for a given x, then (2.12) is unbounded. Let β_l be the extreme ray corresponding to the unbounded dual problem. We can ensure a finite dual solution by bounding the extreme ray, which in effect cuts off the x causing the infeasibility in (2.10). To do so, we add the *feasibility cut*

$$\beta_l(r_l - T_l x) \leq 0 \tag{2.13}$$

to the master problem. If (2.10) is feasible for the given x, then $z_l^D(x)$ equals $z_l(x)$ and provides a lower bound on $z_l(x)$ as x changes. The piecewise linear approximation of $z_l(x)$ is given by the *optimality cut*

$$\alpha_l(r_l - T_l x) \leq \theta_l,$$

where $\theta_l \leq z_l(x)$ is a variable used to approximate $z_l(x)$. If the optimality cut was generated at iteration k, then $\alpha_l(r_l - T_l x^k) = \theta_l = z_l(x^k)$. Otherwise, $\alpha_l(r_l - T_l x) \leq \theta_l \leq z_l(x^k)$. In the *single cut version* of the L-shaped method, a single variable θ is used to approximate $E[f(x, \tilde{\omega})]$, and at most one optimality cut may be added to the master problem at any iteration. The optimality cut is a probability-weighted sum of the optimality cuts from each scenario, $\theta = \sum_l \Pr_l \theta_l$, and takes the form

$$\sum_{l=1}^{L} \Pr_l \alpha_l^i(r_l - T_l x) \leq \theta. \tag{2.14}$$

In the *multicut version*, an optimality cut may be added for each scenario. See (Birge and Louveaux, 1997) for details of the multicut version. The master problem

for the single cut version can now be written as

$$\min_{x,\theta} \quad cx + \theta \tag{2.15a}$$
$$\text{s.t.} \quad Ax = b, \tag{2.15b}$$
$$\sum_{l=1}^{L} \Pr_l \alpha_l^i (r_l - T_l x) \leq \theta, \qquad \forall i, h, \tag{2.15c}$$
$$\beta_l^\kappa (r_l - T_l x) \leq 0, \qquad \forall \kappa, h, l, \tag{2.15d}$$
$$x \geq 0, \tag{2.15e}$$

where $i = 1, \ldots, \mathcal{I}$ index the optimality cuts and $\kappa = 1, \ldots, \mathcal{K}$ index the feasibility cuts. The single cut L-shaped method is summarized as follows.

Step 0. Set iteration count $k = 0$. Set $UB = \infty$ as the upper bound on the optimal solution and $LB = -\infty$ as the lower bound. Initialize the master problem (2.15) without any constraints (2.15c) or (2.15d).

Step 1. Set $k \leftarrow k + 1$. Solve the master problem, get solution x^k, and set $LB = cx^k + \theta$. If $UB - LB \leq \varepsilon$, then quit.

Step 2. Formulate subproblem (2.12) for each l using x^k and solve.

Step 3. If (2.12) is unbounded for any l, add a feasibility cut (2.13) to the master problem and go to Step 1.

Step 4. If all subproblems are feasible, add one feasibility cut (2.14) to the master problem. Set $UB \leftarrow \min\{UB, cx^k + \sum_{l=1}^{L} \Pr_l z_l^D(x^k)\}$. Go to Step 1.

2.1.3 Stochastic Mixed-Integer Programming

Stochastic MIPs (SMIPs) comprise one of the most difficult classes of optimization problems as they combine the large-scale nature of stochastic programs with the computational challenges of integer programming. The general two-stage stochastic

integer program with integer recourse can be written as

$$\begin{aligned} \min_{x} \quad & cx + E[f(x,\omega)] & \\ \text{s.t.} \quad & Ax \geq b, & \\ & x_j \in \{0,1\} & j \in B_1, \\ & x_j \text{ integer}, & j \in I_1, \\ & x_j \geq 0, & j \in J_1, \end{aligned} \tag{2.16}$$

where

$$\begin{aligned} f(x,\omega) = \min_{y} \quad & qy & \\ \text{s.t.} \quad & W(\omega)y \geq r(\omega) - T(\omega)x, & \\ & y_j \in \{0,1\}, & j \in B_2, \\ & y_j \text{ integer}, & j \in I_2, \\ & y_j \geq 0, & j \in J_2, \end{aligned}$$

where B_s is the set of binary variables in stage s, I_s is the set of general integer variables in stage s, and $J_s \supseteq (B_s \cup I_s)$ is the set of all variables in stage $s = 1, 2$. SMIPs are difficult for several reasons. First, the second-stage problem for each scenario is an MIP, which may be difficult to solve. Second, evaluating the expected second-stage cost for a fixed first-stage decision may be impossible if uncertain parameters have a continuous distribution. For discrete distributions, evaluation still requires solving a large number of MIPs. Finally, the SMIP value function $f(x,\omega)$ is nonconvex and often discontinuous. Several algorithms have been developed to overcome these difficulties, but most of the algorithms depend on the problem structure. Here, we briefly review algorithms for two-stage SMIPs. We point the reader to (Klein Haneveld and van der Vlerk, 1999), (Schultz, 2003), and (Sen, 2005) for an in-depth review of both two-stage and multi-stage SMIPs.

Some of the earliest papers on SMIPs examined structural properties. Stougie (1987) and Schultz (1993, 1995) showed that $E[f(x,\tilde{\omega})]$ is real-valued and lower semicontinuous if $f(x,\omega)$ is finite for all x and ω. Schultz et al. (1998) showed that for SMIPs with mixed-integer first-stage $E[f(x,\tilde{\omega})]$ is piecewise constant over

polyhedral regions and the optimal solution lies at an extreme point of one of the polyhedra.

One of the earliest algorithms for SMIPs is the *integer L-shaped method* (Laporte and Louveaux, 1993). The integer L-shaped method extends the optimality and feasibility cuts of the L-shaped method to be valid for problems with binary first-stage variables and arbitrary second-stage variables. The linearity of the optimality and feasibility cuts relies upon the binary first-stage variables. However, this method can be slow in practice due to requiring MIP solutions of subproblems and weak optimality and feasibility cuts.

Louveaux and van der Vlerk (1993) and Klein Haneveld et al. (1995, 1996) studied the simple integer recourse problem in which the second-stage problem penalizes integer deviations from a target value. An example of simple integer recourse would be determining the number of delivery trucks for a product, realizing a demand scenario, and then having to purchase an integer number of trucks to satisfy realized demand. The algorithm builds a convex approximation of the expected recourse function by enumeration over each dimension in the second-stage variables.

Carøe (1998) and Carøe and Tind (1997) developed a branch-and-cut algorithm based on ideas from disjunctive programming (Balas, 1975; Blair and Jeroslow, 1978) for 0-1 MIPs in both stages with general random data. The generated cuts are in the $\{x, y(\omega)\}_{\omega\in\Omega}$ space. Extensions and computational results are presented in Ntaimo and Tanner (2008). Sen and Higle (2005) developed a similar decomposition-based method called *disjunctive decomposition* (D^2) that convexifies the feasible regions of MIP subproblems. Cuts are generated in the $\{y(\omega)\}_{\omega\in\Omega}$ space and result in an improving sequence of subproblem LP relaxations. Ntaimo and Sen (2008) provide computational results indicating the effectiveness of D^2 on stochastic supply chain and stochastic server location problems. Sen and Sherali (2006) extend D^2 to approximate the MIP subproblem value functions, $f(x, \tilde{\omega})$, utilizing information from partially solved branch-and-bound trees. In addition to MIP subproblem feasible region convexification, disjunctions are formed on the optimal objective function values at the nodes in the partially solved branch-and-bound tree and a disjunctive

cut on the MIP value function is passed to the master problem. Sen and Sherali (2006) call this algorithm D^2-BAC (disjunctive decomposition with branch-and-cut). Ntaimo and Sen (2008) provide a comparison of D^2 and D^2-BAC noting that while both algorithms outperform direct solution and the integer L-shaped algorithm of Laporte and Louveaux (1993), neither D^2 nor D^2-BAC consistently solve faster than the other. D^2 was initially developed for binary first-stage and MIP second-stage with fixed recourse, but variants of D^2 have been developed to accommodate continuous first-stage variables (Ntaimo and Sen, 2007) and random recourse matrix, $W(\omega)$, (Ntaimo, 2009), and computational speed-ups have been developed for the cut generation phase of the algorithm (Yang and Sen, 2009). We use D^2 extensively in Chapter 5 and provide a detailed description of the algorithm there.

Several other solution approaches to SMIP have been proposed. Ahmed et al. (2004) developed a specialized branch-and-bound procedure for problems with continuous first-stage variables and integer second-stage variables under the restriction of fixed tenders. Their algorithm exploits the fact that the expected recourse function is convex over hyper-rectangles (boxes) and partitions the feasible region into these hyper-rectangles. The algorithm transforms the optimization problem into the space of the fixed tenders to keep the partitioning tractable. Carøe and Schultz (1999) introduced *dual decomposition* in which the Lagrangian dual problem is used to find lower bounds in a branch-and-bound scheme. Kong et al. (2006) used the super additive property of pure integer stochastic programs to find bounds and developed a branch-and-bound procedure to find optimal tenders. Guan et al. (2006) derived cutting planes for an uncapacitated stochastic lot-sizing problem and Guan et al. (2009) generated cuts based on combining inequalities that are valid for individual scenarios. The branch-and-price method has also been applied to SMIPs (Lulli and Sen, 2004; Sliva and Wood, 2006).

Finally, sampling-based algorithms have been used to solve SMIPs. Kleywegt et al. (2001) established convergence of the sample average approximation and provided theoretical bounds on sample size for ε-optimal solutions for SMIPs with pure integer first-stage. Ahmed and Shapiro (2002) extended these results for problems

with pure integer second-stage and continuous variables in the first-stage. Part of this dissertation modifies a sequential sampling procedure to solve SMIPs with binary first-stage efficiently by allowing approximate solutions (see Chapter 4, Section 4.2).

2.2 Relevant Scheduling Literature Review

In the last 50 years, a multitude of papers and books have been published on scheduling. Some recent books include (Brucker, 2001; Pinedo, 2008; Baker and Trietsch, 2009a). As Brucker (2001) states, "The theory of scheduling is characterized by a virtually unlimited number of problem types." In an attempt to keep the literature review tractable, we first review machine scheduling, the foundation of scheduling theory, and then review project scheduling, a problem closely related to those studied in this dissertation. We will review both deterministic and stochastic scheduling for machine and project scheduling. Finally, we briefly review general stochastic scheduling paradigms.

2.2.1 Machine Scheduling

Deterministic machine scheduling has its roots in three early papers. The first paper by Johnson (1954) found the optimal sequence of running jobs through two machines to minimize *makespan*, the time from the start of the first job to the completion of the last job. Jackson (1955) studied the problem of minimizing the maximum *tardiness* of jobs, where tardiness is defined as the maximum of zero and the difference between the completion time of the job and its due date. Smith (1956) solved several simple sequencing models by scheduling jobs ordered by nondecreasing processing times. The major results from these papers were simple sorting rules such as shortest processing time (SPT), shortest weighted processing time (SWPT), and earliest due date (EDD) that could solve a variety of sequencing problems (see Brucker, 2001, for details). Most models and solution approaches in the years to come built upon the work of these three papers. See (Baker and Trietsch, 2009b,c) for a detailed

history of machine scheduling.

Early researchers realized the importance of considering uncertainty in practical scheduling situations and sought solutions to stochastic versions of already well solved deterministic problems. Rothkopf (1966) showed that several stochastic problems can be transformed into equivalent problems with deterministic processing times, which is referred to as the deterministic counterpart. Banerjee (1965) showed that jobs ordered by earliest due date minimize the maximum probability of lateness of any job. Hodgson (1977) showed that Lawler's algorithm is optimal for minimizing maximum expected deferral cost, where deferral cost is monotonic nondecreasing in time.

Recent work on stochastic scheduling for single machines includes a stochastic branch-and-bound procedure for minimizing expected total weighted tardiness (Gutjahr et al., 1999), minimizing expected total weighted earliness/tardiness for jobs with exponential processing times and a common exponentially distributed due date (Jia, 2001), a heuristic based on SWPT for minimizing expected total weighted completion time of jobs with release dates (Chou et al., 2006), and a stochastic dominance rule for minimizing expected total weighted number of early and tardy jobs (Soroush, 2007). We refer the reader to (Baker and Trietsch, 2009a) for a detailed treatment of stochastic machine scheduling.

2.2.2 Project Scheduling

Project scheduling involves scheduling a set of activities (or jobs) subject to precedence constraints on the activities. Precedence constraints ensure that some activities do not start until other activities have finished. For example, when building a house, the foundation must be completed before the walls can be built. The most common objective is to minimize makespan of the project, or the completion time of the last activity, although other objectives are possible.

In this dissertation, we refer to problems without precedence constraints as *job scheduling* and problems with precedence constraints as *project scheduling*. Multiple resource-constrained job scheduling does not have precedence constraints since

the jobs are assumed to be "projects" independent of each other except for shared resources. For instance, in operating room scheduling most patients only have one surgery at a time. There are typically no precedence constraints on the surgeries but rather a collection of independent surgeries that must be scheduled. The single resource-constrained job scheduling problem has many applications studied in the literature. Chen and Lee (1999) apply the single resource-constrained job scheduling problem to berth allocation where shipping vessels may occupy more than one berth for loading and unloading. Other applications include semiconductor circuit design (Lee and Cai, 1999) and management of multiprocessor computer systems (Drozdowski, 1996).

The first algorithm for project scheduling with precedence constraints was the program evaluation and review technique (PERT) (Malcolm et al., 1959) developed by Booz Allen Hamilton under a US Navy contract. PERT analysis considered uncertainty in activity durations through estimates of optimistic, pessimistic, most likely, and expected activity durations. The stochastic analysis of PERT was quite advanced from its inception, and Clark (1961) showed how to calculate approximately the moments of a project makespan distribution in spite of the precedence relations of activities.

The addition of resource constraints to project scheduling results in the *resource-constrained project scheduling problem*, which cannot be optimally or even ε-optimally solved in polynomial time. Resource constraints may be renewable, nonrenewable, or double-constrained. Renewable resource constraints limit the amount of resource consumed by all activities being processed during a single time period. Each period, the capacity renews itself. An example of a renewable resource is the number of dump trucks available for an excavation project. Nonrenewable resource constraints limit the total amount of resource consumed during the project's lifetime. An example of a nonrenewable resource constraint is budget. Doubly-constrained resources are renewable resources limited on both a per-period and total project basis. An example of a doubly-constrained resource is manpower under labor contracts limiting the total labor expenditure for the project. For solution procedures

to deterministic resource-constrained project scheduling models, see the survey by Herroelen et al. (1998).

The stochastic resource-constrained project scheduling problem with precedence constraints is typically solved in a two phase procedure consisting of first finding a feasible schedule and then strengthening the schedule to make it robust (van de Vonder et al., 2007; Herroelen, 2007). The initial schedule may be found by solving, either exactly or heuristically, a deterministic problem obtained by replacing the uncertain parameters with their average values. Schedule strengthening is accomplished by robust resource allocation (Leus and Herroelen, 2004) or by inserting buffer times into the schedule to discourage propagation of schedule disruptions (van de Vonder, 2006). Although variations of both robust resource allocation and buffer insertion time approaches could be applied to stochastic multiple resource-constrained job scheduling, it is likely the performance would suffer since the search space could not be reduced via precedence constraints.

Exact procedures for stochastic resource-constrained project scheduling are typically limited to a single resource class and assume exponentially distributed activity duration disruption lengths (Leus and Herroelen, 2004). However, heuristic approaches for the multiple resource-constrained versions have been developed (Deblaere et al., 2007).

2.2.3 Stochastic Scheduling Paradigms

Several models and solution approaches have been suggested for stochastic scheduling and typically fit into one of two categories: *proactive* or *reactive scheduling* (Aytug et al., 2005). Proactive scheduling constructs a predictive schedule that will perform well under a wide variety of situations. Reactive scheduling, on the other hand, takes place online at the time of job execution incorporating up-to-date information and changes the schedule as disruptions take place. The models studied in this dissertation are proactive.

Baker and Trietsch (2007) observe that many stochastic scheduling algorithms make use of expected values of parameters and solve the deterministic counterpart

problem using the expected values. For problems with objectives such as minimizing total weighted flowtime, the stochastic solution and expected value solution yield the same optimal objective and solution. However, for problems with objectives such as minimizing total tardiness, the stochastic solution and expected value solution do not correspond. Baker and Trietsch advocate *safe scheduling*, which inserts buffer time in between jobs. Buffer times and safe schedules are determined by considering service levels or by minimizing expected costs of failing to meet due dates. Service level constraints are met when the probability of a job finishing before its due date exceeds a certain threshold. Minimizing expected costs of due date violations is a viable approach when good cost data are available. In the absence of good cost data, the authors recommend using service levels. They show that for some classes of problems when jobs are stochastically ordered, optimal solutions can be found efficiently. In other cases, heuristic solutions can be found using an evolutionary algorithm.

2.3 Concluding Remarks

In the next chapter, we present two models for stochastic scheduling for jobs sharing multiple resources that are two-stage stochastic mixed-integer programs with recourse. Then, in the following two chapters, we present exact and sampling-based approximate solution approaches to solve these models. In Chapter 4, we propose an exact procedure for a general model with multiple resource classes and place only two easily satisfied requirements on processing time distributions. Our algorithm requires only one phase whereas project scheduling algorithms require two phases. Our methodology in Chapter 4 can be applied to problems with any combination of uncertainty in processing times, resource consumption, and resource availability. However, existing algorithms for project scheduling are typically tailored for one type of uncertainty (Herroelen, 2007). In Chapter 5, we apply enhancements to the D^2 algorithm to solve a resource-constrained job scheduling problem with an uncertain number of jobs. The methods in Chapter 5 are also exact procedures and

require only one phase.

We note that disjunctive programming has been used to solve deterministic scheduling problems with precedence constraints. Balas (1985b) studied the job shop problem and exploited the precedence constraints to derive a partial characterization of the scheduling polyhedron using disjunctive programming. Balas (1985a) applied disjunctive programming to the job shop problem to determine a set of relaxations that forms a hierarchy spanning the spectrum from the LP relaxation to the convex hull of the feasible set itself. Our models do not have precedence constraints. Therefore, in Chapter 5 we focus on a special constraint structure that appears in the second-stage of the two-stage SMIP.

CHAPTER 3

STOCHASTIC SCHEDULING MODELS

The models studied in this dissertation consider the problem of determining a schedule for jobs, each job requiring multiple classes of resources. MRCSP arises naturally in many applications. For instance, companies in industries such as consulting and engineering services build teams composed of people with different skill sets and typically do project based work. In this application, jobs correspond to different projects or products the company is developing, and the multiple resources correspond to people with different skill sets such as electrical and industrial engineers, accountants, managers, or equipment and capital constraints. As mentioned in 1, another example is operating room scheduling where the jobs correspond to operations to be performed, and the constrained resources correspond to doctors, nurses, anesthesiologists, special equipment, and the operating room. While specific applications can have their own particular side constraints, in this dissertation, we focus on two variations of a general model: (i) stochastic MRCSP with uncertain processing times, resource consumptions, resource availability, and due dates, which we denote as SMRCSP-U, and (ii) stochastic MRCSP where the number of jobs is uncertain due to bidding on future jobs, which we denote as SMRCSP-JB.

We model the job scheduling problems using two-stage stochastic programming with recourse and build a proactive and, in the terminology of Herroelen and Leus (2005), quality robust schedule. Stochastic programming has been used rarely in the case of stochastic project or job scheduling (Birge and Dempster, 1996; Denton and Gupta, 2003; Morton and Popova, 2004; Zhu et al., 2007). We use a time-indexed formulation that allows us to consider different objectives (e.g., tardiness, net present value) without any change in solution methodologies. The resulting models are amenable to solution via variations of the L-shaped method. Both models share a similar modeling framework, which we present below. Then, we introduce notation

specific to each model and formulate and describe both models.

3.1 Modeling Framework

We model both job scheduling problems with a *time-indexed* formulation where time is partitioned into discrete units of time t. We follow the usual convention where period t starts at $t-1$ and ends at t. Hence, many of the objectives considered below require an adjustment of -1. The time-indexed model offers two strong advantages over other formulations. However, the number of variables is usually much larger than other formulations. The first advantage of the time-indexed formulation is its flexibility. It allows modeling many different scheduling costs without changing the structure of the model, hence, *without requiring different solution algorithms*. Let t be the time period, p_j be the processing time of job j, and d_j be the due date of job j. The costs of starting job j at time t, c_{jt}, can accommodate:

- completion time ($c_{jt} = t + p_j - 1$),
- tardiness ($c_{jt} = \max\{0, t + p_j - d_j - 1\}$),
- earliness/tardiness ($c_{jt} = \max\{-t - p_j + d_j + 1, t + p_j - d_j - 1\}$),
- weighted completion, tardiness, and earliness/tardiness,
- complicated nonlinear contract penalties (set c_{jt} equal to the penalty cost paid if job j finishes at time $t + p_j - 1$),
- negative of net present value (NPV) of job completions (to "maximize" NPV).

The solution approach of the time-indexed model is independent of these cost structures – a nice property considering that specialized single machine scheduling and resource-constrained project scheduling algorithms typically depend on the modeled cost. For instance, changing the objective from minimizing job tardiness to maximizing net present value for resource-constrained project scheduling problems requires changing the solution algorithm (Herroelen et al., 1998). The makespan objective can be modeled by introducing a new variable that is the maximum of the completion times of all jobs, which is then minimized. This slightly alters the

formulations given below. However, makespan may not be a desirable objective in applications such as scheduling consulting teams to jobs since revenues and costs typically do not depend on the makespan of all consulting jobs.

The second advantage of the time-indexed formulation is that for several different problem classes, the LP relaxation of the time-indexed formulation provides tight bounds on the optimal integer solution. Dyer and Wolsey (1990) and Queyranne and Schulz (1994) show that the time-indexed formulation is the strongest formulation known for the single machine scheduling problem. Chan et al. (1998) show that the optimal LP objective from a time-indexed formulation equals the optimal integer objective for some parallel machine scheduling problem classes. For time-indexed formulations of the resource-constrained project scheduling problem, the LP relaxation tightens as the resource constraints become less tight (Möhring et al., 2001, 2003). We observe a similar effect in Chapter 4 that leads to shorter solution times. Good LP bounds provide valuable information for solving integer programs and are critical for efficient solution of the scheduling problems studied here. In fact, we can find a very good starting solution by solving the LP relaxation of the stochastic program and then applying a rounding heuristic (see Chapters 4 and 5).

Below, we describe sets, indices, parameters, and decision variables that both models have in common. In Chapter 2 we used $j \in J$ to denote the index set of decision variables. From now on, we use $j \in J$ to denote the set of jobs to be scheduled.

Common Indices and Sets

$j \in J$	set of jobs that must be scheduled
t	time periods $t = 1, 2, \ldots, T$
k	resource classes $k = 1, 2, \ldots, K$
$\omega \in \Omega$	random future scenario, Ω is the set of all future scenarios

Common Parameters

$\Pr(\omega)$ probability of scenario ω

Common Decision Variables

x_{jt} 1 if job j starts in period t, 0 otherwise

$z_{tk}(\omega)$ amount of temporary resource expansion in period t in scenario ω

In our models, the binary variables x_{jt} take the value 1 if job $j \in J$ is scheduled to start in time period $t \in \{1, 2, \ldots, T\}$ and 0 otherwise. These are the first-stage decision variables. The random future scenario $\omega \in \Omega$ is model dependent. In our first model, we allow many uncertainties such as random processing times and random resource consumption and availabilities. In our second model, we focus on a subset of jobs to be scheduled such that a scenario $\omega \in \Omega$ gives which of these jobs need to be scheduled in the second stage. This occurs when we know a certain number of jobs that must be scheduled (first-stage decisions) and the remaining jobs may or may not need to be scheduled according to a probability distribution. Both models allow for temporary resource expansions and these are captured via the second-stage decision variables $z_{tk}(\omega)$.

3.2 SMRCSP-U

Our first model, which we refer to as SMRCSP-U, allows for uncertain processing times, due dates, and resource consumption and availabilities as in real-world scheduling applications. Each resource class has a capacity, although the capacity can be temporarily expanded at a cost. The temporary expansions are analogous to outsourcing, hiring temporary workers, or renting equipment. Not all resources can be flexible. These hard resource constraints can be enforced by assigning a large penalty cost to the expansion.

The formulation below can model any combination of uncertain processing times, due dates, resource consumption, and resource availabilities without changing the implementation of the two-stage stochastic programming algorithm. First-stage decisions are when to start the jobs (before processing times and other uncertainties

are observed) while the second-stage decisions measure how much temporary resource expansion should be used in each scenario (recourse decisions). Additional notation is introduced to model the uncertain parameters, and we also impose a limit on the amount of temporary resource expansion.

Additional Sets

$S^{\omega}(j,t)$ the interval of time that job j would be processed in if it finished in period t in scenario ω, $S^{\omega}(j,t) = \{\max\{1, t - p_j(\omega) + 1\}, \ldots, \min\{t, T - p_j(\omega) + 1\}\}$

Additional Parameters

$c_{jt}(\omega)$ cost of starting job j in period t in scenario ω

$\bar{c}_{jt}$ expected cost of starting job j in period t, $\bar{c}_{jt} = \sum_{\omega \in \Omega} \Pr(\omega) c_{jt}(\omega)$

$p_j(\omega)$ processing time of job j in scenario ω

p_j^{max} $\max_{\omega \in \Omega} p_j(\omega)$

$d_j(\omega)$ due date of job j in scenario ω

$r_{jk}(\omega)$ amount of resource from class k consumed by job j during a period in scenario ω

$R_{tk}(\omega)$ total amount of resource k available during period t in scenario ω

U_{tk} upper limit on temporary expansion of resource k at time t

b_{tk} per unit penalty for exceeding resource capacity $R_{tk}(\omega)$ in period t that is within upper bound U_{tk}

B_{tk} per unit penalty for exceeding U_{tk}

Additional Decision Variables

$w_{tk}(\omega)$ amount of resource consumed beyond capacity $R_{tk}(\omega) + U_{tk}$ in scenario ω

Formulation

$$\min_{x} \sum_{j\in J} \sum_{t=1}^{T-p_j^{max}+1} \bar{c}_{jt}x_{jt} + \sum_{\omega\in\Omega} \Pr(\omega) f(x,\omega) \tag{3.1}$$

$$\text{s.t.} \sum_{t=1}^{T-p_j^{max}+1} x_{jt} = 1, \qquad j \in J, \tag{3.2}$$

$$x_{jt} \in \{0,1\}, \qquad j \in J, \quad t = 1,\ldots,T-p_j^{max}+1, \tag{3.3}$$

where

$$f(x,\omega) = \min_{z(\omega),w(\omega)} \sum_{t=1}^{T}\sum_{k=1}^{K} (b_{tk}z_{tk}(\omega) + B_{tk}w_{tk}(\omega)) \tag{3.4}$$

$$\text{s.t.} \quad z_{tk}(\omega) + w_{tk}(\omega) \geq \sum_{j\in J}\sum_{s\in S^{\omega}(j,t)} r_{jk}(\omega)x_{js} - R_{tk}(\omega), \quad t = 1,\ldots,T, \quad k = 1,\ldots,K, \tag{3.5}$$

$$0 \leq z_{tk}(\omega) \leq U_{tk}, \qquad t = 1,\ldots,T, \quad k = 1,\ldots,K, \tag{3.6}$$

$$w_{tk}(\omega) \geq 0, \qquad t = 1,\ldots,T, \quad k = 1,\ldots,K. \tag{3.7}$$

We can write the above model more compactly as

$$\min_{x\in X} E[F(x,\omega)], \tag{3.8}$$

where E is the expectation operator, taken with respect to the distribution of ω, $F(x,\omega) = \left(\sum_{j\in J}\sum_{t=1}^{T-p_j^{max}+1} \bar{c}_{jt}x_{jt} + f(x,\omega)\right)$, and X denotes the set of decisions that satisfies (3.2) and (3.3). Let c, b, B, $z(\omega)$, $w(\omega)$, $R(\omega)$, U, and x denote the appropriately sized vectors corresponding to $\bar{c}_{jt}$, b_{tk}, B_{tk}, $z_{tk}(\omega)$, $w_{tk}(\omega)$, $R_{tk}(\omega)$, U_{tk},

and x_{jt}, respectively. Then, we can write (3.8) as

$$\min_{x \in X} cx + E[f(x, \omega)],$$

where

$$\begin{aligned} f(x, \omega) = \min_{y(\omega)} \quad & qy(\omega) \\ & \mathcal{W}y(\omega) \geq h(\omega) - \mathcal{T}(\omega)x, \\ & y(\omega) \geq 0 \end{aligned}$$

with $q = [b, B]$, $y(\omega) = \left[\begin{smallmatrix} z(\omega) \\ w(\omega) \end{smallmatrix}\right]$, $\mathcal{W} = \left[\begin{smallmatrix} I & I \\ -I & 0 \end{smallmatrix}\right]$, $h(\omega) = \left[\begin{smallmatrix} -R(\omega) \\ -U \end{smallmatrix}\right]$, and $\mathcal{T}(\omega)$ is a $TK \times |J|(T-1) - \sum_j p_j^{max}$ matrix with consecutive elements in row tk corresponding to the resource demands for resource class k at period t. This is in the form of (2.9) presented in Chapter 2. As mentioned before, in the stochastic programming literature, $E[f(x, \omega)]$ is called the *recourse function* and $\mathcal{T}$ and $\mathcal{W}$ are called the technology and recourse matrices, respectively.

The first term in (3.1) is the expected cost of starting job j at time t, and the second term is the expected cost of temporary resource expansions. Constraints (3.2) ensure that each job is completed by time T. This model can easily be changed to a *job selection and scheduling* problem without any change in methodology by changing constraints (3.2) to "$\leq$" and changing the objective to maximizing some profit measure. Equations (3.4)-(3.7) model the second-stage, where costs from temporary resource expansions such as outsourcing or renting are minimized. The right hand side of constraints (3.5) is the amount of resource expansion required at time t for class k. Temporary resource expansion is measured by $z_{tk}(\omega)$ and cannot exceed U_{tk}. We add the auxiliary variables $w_{tk}(\omega)$ to ensure second-stage feasibility for the sampling procedure presented in Chapter 4. By setting B_{tk} large, we can ensure that the auxiliary variables will not appear in the optimal solution unless resources are insufficient. In this case, the auxiliary variables provide managerial insight into which resources are bottlenecks.

Since the time-indexed formulation partitions time into discrete units, we impose two mild requirements on the processing time distribution. First, we require that processing time realizations with non-zero probability mass should be natural numbers. Note that a discrete approximation may be used in place of a continuous distribution. Second, since jobs must be started such that they finish by the end of planning horizon T in all scenarios, the distribution on processing times must have a bounded support $0 \leq p_j \leq T$ for all $j \in J$. In real-world applications, jobs are either completed or terminated in a finite amount of time. Therefore, it is reasonable to truncate unbounded distributions to satisfy the bounded support requirement. Note that T can be adjusted and we further investigate this issue in Chapter 4.

Several common scheduling problems are special cases of SMRCSP. Consider a special case of SMRCSP with a single scenario, one resource class with unit consumption and availability, i.e., $r_j = 1, \forall j$, $R_t = 1$ and $U_t = 0, \forall t$. Set B_t sufficiently large so that no optimal solution will have $w_t > 0, \forall t$. The resulting problem is a single machine scheduling problem. Keeping the same changes and setting $R_t = m, \ \forall t$, results in the parallel machine scheduling problem. Complexity results for single and parallel machine scheduling problems are well known (see Brucker, 2001). For instance, minimizing the sum of completion times of jobs on a single machine can be solved in polynomial time and minimizing the weighted sum of tardiness of jobs on a single machine is strongly NP-hard.

The multiprocessor scheduling problem with dedicated processors is also a special case of SMRCSP. To see this, consider SMRCSP with a single scenario, $r_{jk} = 0$ or $1, \ \forall j, k$, $R_{tk} = 1$, $U_{tk} = 0, \ \forall t, k$, and set B_{tk} sufficiently large so that no optimal solution will have $w_{tk} > 0, \ \forall t, k$. Following standard scheduling notation (Brucker, 2001), the problem can be described as $MPT||f$ with f representing the objective function. Minimizing the sum of completion times, $MPT||\sum C_j$, is a generalization of $MPT2||\sum C_j$ and $MPT|p_j = 1|\sum C_j$, which are known to be strongly NP-hard (Cai et al., 1998; Hoogeveen et al., 1994). Minimizing the sum of tardiness, $MPT||\sum T_j$, is a generalization of minimizing makespan, $MPT||C_{\max}$, which was shown by Kubale (1987) to be strongly NP-hard. The weighted versions

of these objectives are generalizations of the non weighted versions. SMRCSP under these conditions is a generalization of $MPT||f$ and is therefore strongly NP-hard for many common objectives. The difficulty of solving the SMRCSP increases when considering more than one scenario due to stochastic data parameters.

3.3 SMRCSP-JB

Our second model, which we refer to SMRCSP-JB, is a stochastic scheduling problem with an uncertain number of jobs faced by companies in industries such as consulting, engineering services, and defense contracting. These companies bid on future contracts but may or may not win the contract, resulting in a subset of jobs that must be scheduled and a subset that may or may not require scheduling. The multiple resources correspond to people with different skill sets such as electrical and industrial engineers, accountants, managers, or equipment and capital constraints. Stochastic scheduling problems with uncertain number of jobs arise in other contexts. For instance, in operating room scheduling there are typically a set of operations that must be performed. However, new operations need to be scheduled during the day as emergency room surgeries arrive.

SMRCSP-JB, like SMRCSP-U, allows resources to be temporarily expanded for a penalty cost, corresponding to costs due to outsourcing, overtime, or equipment rental. Moreover, each job has T time periods in which it could be processed. In the first-stage, the known jobs are scheduled, and each known job must finish by time period T. At time period $1 \leq T_0 \leq T$, we learn which job bids have been accepted and schedule the accepted jobs so that they finish by time period $T + T_0$. During time periods $t \in [T_0 + 1, \ldots, T]$ both known jobs and jobs with accepted bids can be processed. In our computational experiments, we set $T_0 = \lceil 0.25T \rceil$.

One major difference between SMRCSP-JB and SMRCSP-U is the uncertainties present in the models. SMRCSP-U considers uncertainty in processing times, due dates, and resource consumptions and availabilities. The resulting model has binary first-stage variables and continuous second-stage variables. On the other

hand, SMRCSP-JB considers only uncertainty in whether bids on jobs have been accepted or not. The resulting model presented below has mixed-binary first-stage variables and mixed-binary second-stage variables. Since subproblems are MIPs, SMRCSP-JB is more challenging to solve. We present this model below and present our solution methodology and results of our computational tests on instances of SMRCSP-JB in Chapter 5.

Additional Sets

$j \in J_B$	set of bid jobs that may or may not require scheduling
t	time periods $t = 1, 2, \ldots, T + T_0$
$S(j,t)$	the interval of time that job $j \in J$ would be processed in if it finished in period t, $S(j,t) = \{\max\{1, t - p_j + 1\}, \ldots, \min\{t, T - p_j + 1\}\}$
$S_B(j,t)$	the interval of time that job $j \in J_B$ would be processed in if it finished in period t, $S_B(j,t) = \{\max\{T_0 + 1, t - p_j + 1\}, \ldots, \min\{t, T + T_0 - p_j + 1\}\}$

Additional Parameters

T_0	time period at which we learn which job bids $j \in J_B$ are accepted
c_{jt}	cost of starting job j in period t
p_j	processing time of job j
d_j	due date of job j
r_{jk}	amount of resource from class k consumed by job j during a period
R_{tk}	total amount of resource k available during period t
b_{tk}	per unit penalty for exceeding resource capacity R_{tk} in period t
$a_j(\omega)$	1 if bid on job $j \in J_B$ is accepted in scenario ω, 0 otherwise

Additional Decision Variables

$y_{jt}(\omega)$	1 if job $j \in J_B$ starts in period $t \in [T_0 + 1, T + T_0]$ in scenario ω, 0 otherwise
z_{tk}	amount of temporary resource expansion in period $t \in [1, T_0]$

$z_{tk}(\omega)$ amount of temporary resource expansion in period $t \in [T_0+1, T+T_0]$ in scenario ω

Formulation

$$\min_{x,z} \sum_{j\in J}\sum_{t=1}^{T-p_j+1} c_{jt}x_{jt} + \sum_{t=1}^{T_0}\sum_{k=1}^{K} b_{tk}z_{tk} + \sum_{\omega\in\Omega}\Pr(\omega)f(x,\omega) \tag{3.9}$$

$$\text{s.t.} \quad \sum_{t=1}^{T-p_j+1} x_{jt} = 1, \qquad j \in J, \tag{3.10}$$

$$\sum_{j\in J}\sum_{s\in S(j,t)} r_{jk}x_{js} - z_{tk} \le R_{tk} \qquad t = 1,\dots,T_0 \quad k = 1,\dots,K \tag{3.11}$$

$$x_{jt} \in \{0,1\}, \qquad j \in J, \quad t = 1,\dots,T-p_j+1, \tag{3.12}$$

$$z_{tk} \ge 0 \qquad t = 1,\dots,T_0, \quad k = 1,\dots,K, \tag{3.13}$$

where

$$f(x,\omega) = \min_{y(\omega),z(\omega)} \sum_{j\in J_B}\sum_{t=T_0+1}^{T+T_0-p_j+1} c_{jt}y_{jt}(\omega) + \sum_{t=T_0+1}^{T+T_0}\sum_{k=1}^{K} b_{tk}z_{tk}(\omega) \tag{3.14}$$

$$\text{s.t.} \quad \sum_{t=T_0+1}^{T+T_0-p_j+1} y_{jt}(\omega) = a_j(\omega), \qquad j \in J_B, \tag{3.15}$$

$$\sum_{j\in J_B}\sum_{s\in S_B(j,t)} r_{jk}y_{js}(\omega) - z_{tk}(\omega) \le R_{tk} - \sum_{j\in J}\sum_{s\in S(j,t)} r_{jk}x_{js}, \quad t = T_0+1,\dots,T+T_0, \quad k = 1,\dots,K, \tag{3.16}$$

$$y_{jt}(\omega) \in \{0,1\}, \qquad j \in J_B, \quad t = T_0+1,\dots,T+T_0-p_j+1, \tag{3.17}$$

$$z_{tk}(\omega) \ge 0, \qquad t = T_0+1,\dots,T+T_0, \quad k = 1,\dots,K \tag{3.18}$$

The objective (3.9) is to minimize the expected cost of scheduling jobs that are known with certainty plus costs due to temporary resource expansion plus the expected cost of the second-stage. The second-stage objective (3.14) is to minimize the cost of scheduling jobs whose bids have been accepted plus any additional temporary resource expansion. Constraints (3.10) require that each job is started so that it will finish by the end of the planning horizon T. Constraints (3.11) measure the amount of temporary resource capacity required before bid acceptance is known. Constraints (3.12) force the decision variables to be binary. The second-stage constraints (3.15) ensure that jobs with winning bids in scenario ω are scheduled so that they complete by the end of the rolling horizon $T + T_0$. The parameter $a_j(\omega)$ takes the value 1 in scenarios in which the bid on job j, $j \in J_B$, has been accepted, thereby making constraint (3.15) active. If the bid is rejected in the scenario, the parameter $a_j(\omega)$ is zero, making the constraint inactive. Constraints (3.16) measure the amount of temporary resource capacity required for each scenario after the bid acceptance is known. Constraints (3.17) force the job start decisions to be binary and constraints (3.18) ensure the temporary resource expansions are nonnegative.

The model can easily be modified to determine which jobs to bid on, how to schedule them, and how much more resources to acquire by changing the objective to maximize some profit measure, removing the t subscript from z_{tk} variables, and changing (3.15) to $\sum_{t=T_0+1}^{T+T_0-p_j+1} y_{jt} = a_j(\omega)w_j$ for all $j \in J_B$, where w_j, for all $j \in J_B$ is a first-stage variable that takes the value 1 if we bid on job j in the first stage, 0 otherwise. Additionally, if idle resources are a concern to the decision maker, an extra constraint similar to (3.11) and (3.16) can be added along with an extra decision variable for each time period and resource class to measure the amount of idle resources. Adding the new variables and idle costs to the objective will ensure the objective function considers the cost of idle resources. The idle resources modification can be applied similarly to the SMRCSP-U.

Remark 3.3.1. *The D^2 algorithm we present in Chapter 5 is restricted to problems with only binary first-stage variables. SMRCSP-JB presented above has continuous*

first-stage variables z_{tk}, $t = 1, \ldots, T_0$, $k = 1, \ldots, K$. *However, this does not affect* D^2 *since the continuous variables do not affect the second-stage.*

In the following two chapters, we develop solution methodologies for SMRCSP-U and SMRCSP-JB, respectively, and provide computational results and further analysis into what makes problem instances difficult to solve.

CHAPTER 4

SOLUTION METHODS FOR SMRCSP-U

In this chapter, we solve the SMRCSP-U formulated in Chapter 3. We use two-stage stochastic integer programming to determine an optimal schedule for jobs requiring multiple classes of resources under uncertain processing times, due dates, resource consumption and availabilities. We allow temporary resource capacity expansion for a penalty. First, we develop an exact solution approach based on the L-shaped method for problems with a moderate number of scenarios. Several algorithmic enhancements are added to improve efficiency. Then, we embed the L-shaped method within a sampling-based solution method for problems with a large number of scenarios. We modify a sequential sampling procedure to allow for approximate solution of integer programs and prove desired properties. Finally, we compare the solution methodologies on a set of test problems. Parts from this chapter come from (Keller and Bayraksan, 2009).

4.1 Exact Solution Methodology

When the number of scenarios, $|\Omega|$, is small-to-moderate, SMRCSP-U can be solved exactly (i.e., within typical integer programming tolerances) in a reasonable amount of time via the L-shaped method. While the number of scenarios that allow for exact solution is problem dependent, in our computational results, we observed a slowing around 2500 scenarios. Alternatively, one can try solving the SMRCSP-U directly as one large MIP. However, this approach is usually not practical, and we comment on this in more detail in Section 4.1.3. In Section 4.1.1, we briefly review the L-shaped method for our problem. Next, in Sections 4.1.2 and 4.1.3, we describe computational enhancements aimed at improving computation times and report on the results of the enhancements. Finally, in Section 4.1.4, we investigate

the solution characteristics of the SMRCSP-U under various changes in number of resource classes, planning horizon, and resource capacities.

4.1.1 L-Shaped Method

The L-shaped method works by decomposing the problem into one master problem and $|\Omega|$ subproblems, one for each scenario. At iteration i of the L-shaped method, the master problem is solved to obtain feasible first-stage decision variables, x^i. Then, the second-stage problem for each scenario is evaluated using the x^i provided by the master problem. Dual information from the second-stage problems are used to form a cut that is added to the first-stage problem. The cut either improves the piecewise linear lower approximation to the recourse function, resulting in an *optimality cut* or cuts off an infeasible first-stage solution, resulting in a *feasibility cut*. Since SMRCSP-U given in (3.1)-(3.7) has relatively complete recourse (i.e., second-stage problems are always feasible for any given x), only optimality cuts must be added.

The master problem can be written as

$$\begin{aligned} \min_{x,\theta} \quad & \sum_{j=1}^{J} \sum_{t=1}^{T-p_j^{max}+1} \bar{c}_{jt} x_{jt} + \theta \\ \text{s.t.} \quad & \text{constraints (3.2), (3.3),} \\ & -\mathcal{G}^i x + \theta \geq \mathcal{G}_0^i, \qquad i = 1, \ldots, \mathcal{I}, \end{aligned}$$

where θ is a continuous variable corresponding to the expected value of the second-stage objective, $\mathcal{I}$ is the number of cuts generated so far, and $-\mathcal{G}^i$ is the cut gradient and $\mathcal{G}_0^i$ is the cut intercept for cut $i = 1, \ldots, \mathcal{I}$. Here, we present a single-cut version of the algorithm where cuts from a given scenario are aggregated: $\mathcal{G}^i x = \sum_{\omega \in \Omega} \Pr(\omega) \mathcal{G}^i(\omega) x$, $\mathcal{G}_0^i = \sum_{\omega \in \Omega} \Pr(\omega) \mathcal{G}_0^i(\omega)$. In our computational tests, we experimented with the multicut version (see, e.g., Birge and Louveaux (1997)) and found it to be too slow for this problem. Hence, discussion and computations are based on the single-cut version. Subproblems for a given first-stage x and scenario

ω are given in (3.4)-(3.7). Let $\alpha_{tk}(\omega)$ be the optimal dual variables corresponding to constraints (3.5) and $\beta_{tk}(\omega)$ be the dual variables corresponding to upper bound constraints (3.6). The dual problem can be formulated as

$$\begin{aligned}
\max_{\alpha,\beta} \sum_{t=1}^{T}\sum_{k=1}^{K}\left[\left(R_{tk}(\omega) - \sum_{j\in J}\sum_{s\in S^{\omega}(j,t)} r_{jk}(\omega)x_{js}\right)\alpha_{tk}(\omega) + U_{tk}\beta_{tk}(\omega)\right] \\
\text{s.t.} \quad -\alpha_{tk}(\omega) + \beta_{tk}(\omega) \leq b_{tk}, \quad t=1,\ldots,T,\ k=1,\ldots,K, \\
-\alpha_{tk}(\omega) \leq B_{tk}, \quad t=1,\ldots,T,\ k=1,\ldots,K, \\
\alpha_{tk}(\omega),\ \beta_{tk}(\omega) \leq 0, \quad t=1,\ldots,T,\ k=1,\ldots,K.
\end{aligned}$$

The optimal dual solution results from one of the three following cases:

1. If $\sum_{j=1}^{J}\sum_{s\in S^{\omega}(j,t)} r_{jk}(\omega)x_{js} \leq R_{tk}(\omega)$, then $\alpha_{tk}(\omega) = 0$ and $\beta_{tk}(\omega) = 0$.

2. If $R_{tk}(\omega) < \sum_{j=1}^{J}\sum_{s\in S^{\omega}(j,t)} r_{jk}(\omega)x_{js} \leq R_{tk}(\omega) + U_{tk}$, then $\alpha_{tk}(\omega) = -b_{tk}$ and $\beta_{tk}(\omega) = 0$.

3. If $\sum_{j=1}^{J}\sum_{s\in S^{\omega}(j,t)} r_{jk}(\omega)x_{js} > R_{tk}(\omega) + U_{tk}$, then $\alpha_{tk}(\omega) = -B_{tk}$ and $\beta_{tk}(\omega) = b_{tk} - B_{tk}$.

Then, the optimality cut is obtained by

$$-\mathcal{G}^{i}(\omega)x = \sum_{t=1}^{T}\sum_{k=1}^{K}\sum_{j=1}^{J}\sum_{s\in S^{\omega}(j,t)} \alpha_{tk}(\omega)r_{jk}(\omega)x_{js}. \tag{4.1a}$$

$$\mathcal{G}_{0}^{i}(\omega) = \sum_{t=1}^{T}\sum_{k=1}^{K}\left(\alpha_{tk}(\omega)R_{tk}(\omega) + \beta_{tk}(\omega)U_{tk}\right), \tag{4.1b}$$

The dual of the SMRCSP-U second-stage problem can be solved directly without requiring an LP solution algorithm. Hence, (4.1b) and (4.1a) are calculated very efficiently.

4.1.2 Computational Enhancements

Even though the L-shaped method enables efficient solution of SMRCSP-U, several computational enhancements were added to further decrease computation time. These are: (i) trust regions, (ii) branching on generalized upper bound (GUB) constraints, (iii) integer cutting planes on the optimality cuts, (iv) warm-starting with the LP solution, and (v) approximate solution of the master problem. Below, we explain these in more detail, and in Section 4.1.3 we provide the results of our computational tests on the effectiveness of these enhancements.

Trust Regions: In early iterations of the L-shaped method, the solution often oscillates wildly from one iteration to the next thereby slowing convergence. For continuous problems, a penalty can be added to the objective corresponding to the l_2 norm of the new master problem solution from the previous solution. The penalty encourages solutions to remain closer to each other from one iteration to the next. This method is known as regularized decomposition, and has been frequently used (Higle and Sen, 1996; Hiriart-Urruty and Lemaréchal, 1993; Kiwiel, 1990; Ruszczyński, 1986). Implementing regularized decomposition on a stochastic integer program would result in a quadratic integer master problem. We can avoid solving a quadratic integer program while reducing oscillation by instead implementing a trust region. The trust region is a set of constraints that defines a region around the previous solution such that the next solution must lie within this region. Trust regions have been used successfully in stochastic linear programs (Linderoth and Wright, 2003). However, since the SMRCSP-U decision variables are binary, the l_∞ norm typically used for continuous problems would be meaningless. Instead, we implement two different trust regions and compare their effectiveness.

The first trust region limits the number of first-stage variables that can be changed from iteration to iteration, similar to the one implemented by Santoso et al. (2005), called the Hamming distance. Suppose that x^i is the master problem solution at the i^{th} iteration. Let $X^i = \{(j,t) : x^i_{jt} = 1\}$. Then, the Hamming

distance trust region is

$$\sum_{(j,t)\in X^i} (1 - x_{jt}) + \sum_{(j,t)\notin X^i} x_{jt} \leq \Delta^i, \tag{4.2}$$

where Δ^i denotes the limit on the number of variables that can be changed from iteration i to $i+1$.

The Hamming distance is a versatile trust region that can be applied to most binary master problems. The second trust region is geared towards our formulation and limits the change in start times for each job from one iteration to the next. Let τ_j be the start time of job j in the previous iteration. Then, the trust region based on job start times is

$$-\Delta^i \leq \sum_{t=1}^{T-p_j+1} t x_{jt} - \tau_j \leq \Delta^i, \qquad j = 1, \ldots, J, \tag{4.3}$$

where Δ^i is the permitted deviation in start times from one iteration to the next.

Intelligent choices for the size of the trust region can make it more effective. In our implementation, we set $\Delta^1 = \lceil \frac{J}{2} \rceil$ for (4.2), so that initially we allow half of the jobs to change start times. Similarly, we set $\Delta^1 = \lceil \frac{T}{2} \rceil$ for (4.3), which restricts job start times to half of the horizon. If the objective evaluated at x^i is greater than the upper bound, we shrink the trust region by setting $\Delta^{i+1} = \lceil 0.7\Delta^i \rceil$. If both the objective at x^i is less than or equal to the upper bound and any trust region constraint is binding, then we expand the trust region by setting $\Delta^{i+1} = \lceil 1.3\Delta^i \rceil$. The multipliers 0.7 and 1.3 were chosen after a little experimentation so that the trust region would not expand or contract too quickly and lose effectiveness. Convergence is not guaranteed when using trust regions with integer master problems. However, since oscillation is most prominent in early iterations, we remove the trust regions once the L-shaped optimality gap has reached a specified threshold.

GUB Branching: Constraints (3.2) are known as generalized upper bound (GUB) constraints since only one variable in each constraint can be set to 1. Define the GUB set for job j as $G_j = \{(j,t) : 1 \leq t \leq T - p_j^{max} + 1\}$. When solving

the master problem, the standard branching rule is to split the branch and bound tree into one partition with $x_{jt}^{LP} = 0$ and another partition with $x_{jt}^{LP} = 1$ for some fractional x_{jt}^{LP}, $(j,t) \in G_j$. However, this leads to an unbalanced tree since there are roughly $T-1$ possibilities in G_j for nonzero variables in the first branch and only one in the second branch. Let $t_1, t_2, \ldots, t_k$ be some ordering of the fractional variables in G_j. Realizing that variables in G_j must sum to 1, we can create one branch with $x_{jt_i} = 0,\ i = 1, \ldots, r$ and another branch with $x_{jt_i} = 0,\ i = r+1, \ldots, k$, where $r = \min\{l : \sum_{i=1}^{l} x_{jt_i}^{LP} \geq 1/2\}$. This leads to a more balanced tree and can greatly improve solution times (Wolsey, 1998).

Integer Cuts on Optimality Cuts: The optimality cuts for our problem, given in (4.1), are the same as knapsack constraints with one continuous variable. One way to improve performance is to add integer cuts on the optimality cuts. Marchand and Wolsey (1999) and Miller et al. (2000) have derived families of facet defining cutting planes for continuous knapsack problems. The continuous knapsack problem is characterized by the set

$$Y = \left\{ (y,s) \in \mathbb{B}^n \times \mathbb{R}_+^1 : \sum_{j \in N} a_j y_j \leq b + s \right\}, \tag{4.4}$$

where $N = \{1, \ldots, n\}$ is the set of elements in the knapsack, $a_j > 0$, $j \in N$, and $b > 0$. For the optimality cuts of SMRCSP-U, x_{jt} correspond to y_j, the cut gradient G to $[a_1, \ldots, a_j]$, cut intercept g to b, and θ to s in (4.4).

An (i, C, T) cover pair is defined by an index i, set C, and set T such that $C \cap T = i$, $C \cup T = N$, $\lambda = \sum_{j \in C} a_j - b > 0$, and $a_i > \lambda$. The cuts are given by the following theorem.

Theorem 4.1.1. *(Marchand and Wolsey, 1999) Given an (i, C, T) cover pair for Y, order the elements of C such that $a_{[1]} \geq \cdots \geq a_{[r_C]}$, where r_C is the number of*

elements of C *with* $a_j > \lambda$. *Let* $A_0 = 0$, $A_j = \sum_{p=1}^{j} a_{[p]}$, $j = 1, \ldots, r_C$, *and define*

$$\phi_C(u) = \begin{cases} (j-1)\lambda, & if \quad A_{j-1} \le u \le A_j - \lambda, \\ & \quad j = 1, \ldots, r_C, \\ (j-1)\lambda + [u - (A_j - \lambda)], & if \quad A_j - \lambda \le u \le A_j, \\ & \quad j = 1, \ldots, r_C - 1, \\ (r_C - 1)\lambda + [u - (A_{r_C} - \lambda)], & if \quad A_{r_C} - \lambda \le u. \end{cases}$$

The inequality

$$\sum_{j \in C} \min\{\lambda, a_j\} y_j + \sum_{j \in T \setminus i} \phi_C(a_j) y_j \le \sum_{j \in C \setminus i} \min\{\lambda, a_j\} + s \tag{4.5}$$

defines the convex hull of Y.

We implement cuts (4.5) and provide results in the following section.

LP Warm-Starting: We can find a good starting solution by solving the linear programming relaxation of the stochastic program and then applying a rounding heuristic. We experimented with several heuristics and found the following to be the most effective in our tests.

Step 1. Solve LP relaxation. Get fractional solution x^{LP}.

Step 2. For each variable, if $x_{jt}^{LP} = 0$, then remove variable from formulation. If $x_{jt}^{LP} = 1$, then start job j at time t. Subtract its effects from the GUB constraints and optimality cuts, and remove variable from formulation.

Step 3. The remaining x_{jt}^{LP} are the fractional variables from x^{LP}. Solve the reduced-size IP to find an integer solution x^{IP}.

Since the second-stage problems are continuous, the optimality cuts generated in solving the LP solution are valid for the integer program and provide valuable information on the shape of the recourse function. In our computations, we noticed that the integer master problem took longer to solve at each L-shaped iteration as more constraints were present. We experimented with keeping all cuts from the LP solution and dropping different portions of the cuts and found that dropping the

first half of optimality cuts resulted in the best performance. As will be seen below, the LP warm-start facilitates the solution of problems otherwise unsolvable in the given time limit.

Approximate Master Solve: Most of the computational effort is spent solving the master problem. At the early iterations of the L-shaped method, the master problem solutions are often far away from the optimal solutions. Computational effort may be saved if the master problem is solved to within ε-optimality, $\varepsilon \geq 0.1\%$. We resume the exact solution of the master problem when the optimality gap is small enough or if the same master problem solution is generated in consecutive iterations.

Implementation Details: Since the subproblems are efficiently solved, the enhancements discussed above are mainly aimed at improving master problem solution times. L-shaped optimality cuts were added to the master problem after it was solved to integer optimality, except when using the approximate master solve enhancement. In this case, optimality cuts were added once a 1% optimal solution was found. The continuous knapsack cuts were implemented on the master problem within a branch-and-cut framework, generating and adding cuts at each branch-and-bound node. Cut generation was discontinued when the ratio of cuts added to number of searches fell below 10% to prevent spending time searching for cuts when good cuts were not likely to be found. Continuous knapsack cuts generated in previous L-shaped iterations were retained for future iterations. The GUB branching was implemented using the SOS1 feature in CPLEX 10.1. We used LP warm-starting only in initialization of the L-shaped method to provide a set of optimality cuts and a good starting solution for the MIP solver. The trust regions were used on the MIP master problems and removed once the L-shaped optimality gap fell below 5%.

4.1.3 Computational Results

The L-shaped method with various computational enhancements was tested on a set of 9 test problems to find the best combination of enhancements. The generated problems are described by a series of numbers A.B.C, where A is the number of jobs (20, 40, or 80), B is the number of resource classes (5), and C denotes the problem

parameter	generation scheme	parameter	generation scheme
d_j	U(1,10)	p_j	U(1,50)
r_{jk}	U(1,5)	R_k	between $\bar{p}\bar{r}_k J/T$
b_k	U(1,10)		and $\bar{p}\bar{r}_k J/(0.6T)$
B_k	$\max\{2b_k, 10\}$	U_k	$\lceil 0.1R_k \rceil$

Table 4.1: Description of how the test problem parameters were generated. $\bar{p}$ is the average processing time for the problem instance, $\bar{r}_k$ is the average resource consumption for resource K.

instance for an A.B combination. The planning horizon, T, is set to 50 time periods for all problems, and costs, $c_{jt}(\omega)$, are set to tardiness for all problems.

Parameters are generated as described in Table 4.1. Each problem has 10 jobs with two possible processing times. The remaining jobs in each problem have deterministic processing times resulting in a total of $2^{10} = 1024$ scenarios. We explore much larger problems in Section 4.2.2. For jobs with uncertain processing times, the first processing time is generated from integer uniform(1,50). The second processing time is set to the first processing time plus 5. If this results in a processing time larger than T, we instead set the second processing time equal to the first processing time minus 5. We set $R_{tk}(\omega) = R_k$ for all $t = 1, \ldots, T$ and $\omega \in \Omega$ and $r_{jk}(\omega) = r_{jk}$ for all $\omega \in \Omega$, restricting uncertainty to processing times only. R_k, given in Table 4.1, ensures that R_k is exceeded by a small amount in most time periods. Termination criteria is an optimality gap of less than 1% or a five hour (18,000 seconds) time limit. The algorithms were coded in C++ using the CPLEX 10.1 callable library running on a 900 MHz UltraSPARC-III with 4 GB of memory.

Before we present our results on the L-shaped method and its enhancements, we comment on the direct solution of SMRCSP-U without any decomposition. A direct solution of the deterministic equivalent of SMRCSP-U is computationally too expensive, requiring us to resort to a decomposition method like the L-shaped. For instance, even for 20-job problems, the root node LP was not solved after 5 hours of computation time and provided no optimality gap estimate. Table 4.2 compares the results of the enhancements to plain L-shaped method. In Table 4.2, we only provide results from the 20 job problems for brevity and present a summary for all

	No Enhancements			Hamming			Job Start			GUB		
problem	time	it	gap	time	it	gap	time	it	gap	time	it	gap
20.5.1	18000	203	1.53%	18000	192	1.39%	11525	205	0.98%	7533	219	0.99%
20.5.2	18000	123	6.58%	18000	109	5.80%	18000	151	3.78%	13784	197	0.92%
20.5.3	18000	72	25.59%	18000	71	7.73%	18000	107	6.84%	18000	157	13.03%

	GUB + Hamming			GUB + Job Start			Cuts				
problem	time	it	gap	time	it	gap	time	it	gap	ncuts	ratio
20.5.1	3069	174	0.99%	5313	195	0.99%	18000	208	1.18%	1573	0.09999
20.5.2	9210	158	0.92%	5900	168	0.92%	18000	121	8.87%	1804	0.09999
20.5.3	18000	166	7.41%	18000	183	5.06%	18000	72	25.59%	943	0.09999

	LP					Approx Master		
problem	LP time	IP time	total time	IP it	gap	time	it	gap
20.5.1	382	536	918	47	0.99%	5747	210	0.98%
20.5.2	283	18000	18283	117	2.08%	10362	182	0.92%
20.5.3	813	18000	18813	97	1.84%	18000	212	6.36%

Table 4.2: Effectiveness of the computational enhancements.

problems in Table 4.3. All methods enhanced plain L-shaped performance except for the integer cuts on the optimality cuts. Regular knapsack cuts in the presence of GUB constraints are generally not facet defining unless certain conditions are met (Wolsey, 1990). We believe that the GUB constraints present in SMRCSP-U affect the continuous knapsack cuts in a similar manner thereby reducing their effectiveness. The LP warm-start and approximately solving the master problem resulted in the most improvement. These two enhancements are even more effective when combined with the job start trust region and GUB branching. Table 4.3 shows the results of this combination on problems with up to 80 jobs. The enhancement combination solves 8 of the 9 test problems to the termination criteria of 1% optimality within the five hour time limit. We also tested the enhancements on objectives of minimizing total completion time and total lateness. The results are similar to Tables 4.2 and 4.3 and are not presented for the sake of brevity.

	No Enhancements			GUB + Job Start + LP + Approx Master				
problem	time	it	gap	LP time	IP time	total time	IP it	gap
20.5.1	18000	203	1.53%	386	156	542	27	0.87%
20.5.2	18000	123	6.58%	287	715	1002	71	0.92%
20.5.3	18000	72	25.59%	189	5435	5624	121	0.81%
40.5.1	18000	53	9.68%	1576	18000	19576	56	3.05%
40.5.2	10237	217	0.73%	911	1638	2549	97	0.98%
40.5.3	18000	132	3.24%	950	2332	3282	92	0.99%
80.5.1	18000	21	13.38%	1762	4162	5924	162	0.99%
80.5.2	18000	47	1.27%	973	86	1059	3	0.42%
80.5.3	18000	38	17.18%	3263	1185	4448	14	0.92%
total	**154,237**					**44,006**		

Table 4.3: Improvements in running time compared to plain L-shaped method due to computational enhancements.

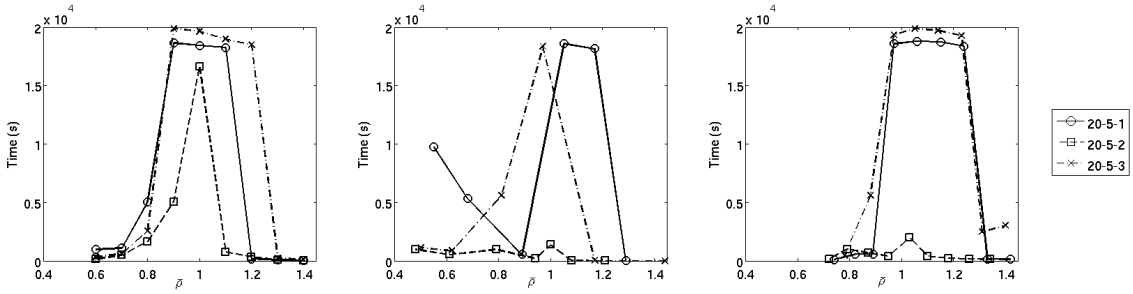
4.1.4 Further Analysis

In this section, we investigate (i) what makes a problem instance difficult to solve and (ii) characteristics of the optimal solutions.

Instance Difficulty: The computational effort required to solve the problems can change dramatically as we vary the resource demands and capacities, planning horizon, and number of jobs. The analysis here provides insight into what makes a problem instance difficult to solve. We have found that the patterns in solution times are best explained by what we refer to as the resource density, ρ. For the test problems, $R_{tk}(\omega) = R_k$ for all $t = 1, \ldots, T$ and $\omega \in \Omega$ and $r_{jk}(\omega) = r_{jk}$ for all $\omega \in \Omega$. As such, we define the density of resource class k as

$$\rho_k = \frac{\bar{p}\bar{r}_k J}{T R_k},$$

where $\bar{p} = \frac{1}{J}\sum_{j=1}^{J} \bar{p}_j$ is the average processing time for all jobs and $\bar{r}_k = \frac{1}{J}\sum_{j=1}^{J} r_{jk}$ is the average resource consumption of class k. The numerator is a rough estimate of the total resource consumption, and the denominator is the total amount of resource k available over the planning horizon T. For $\rho_k < 1$ we expect to have more resources available than consumption over the horizon and for $\rho_k > 1$ we expect to have more consumption than resource available.

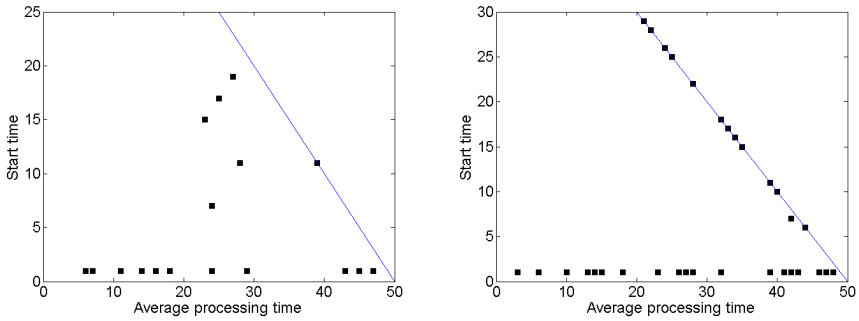


(a) changing $\bar{r}_k/R_k$ (b) changing T (c) changing J

Figure 4.1: Solution times are strongly influenced by $\bar{\rho}$. The figures show solution times when $\bar{\rho}$ is varied by changing $\bar{r}_k/R_k$, T, and J, respectively.

Figure 4.1 shows the sensitivity of solution times to $\bar{\rho} = \frac{1}{K}\sum_{k=1}^{K}\rho_k$. In Figure 4.1a, we changed $\bar{\rho}$ by only changing the ratio $\bar{r}_k/R_k$. In Figure 4.1b, we changed $\bar{\rho}$ by adjusting the planning horizon T by increments of 10, and in Figure 4.1c, we changed $\bar{\rho}$ by solving versions of the problems with 16 to 34 jobs while fixing all other parameters. For $\bar{\rho} \leq 0.8$, there is enough resource capacity so second-stage costs are negligible. The optimization then minimizes the first-stage costs while keeping resource consumption below capacity. On the other hand, for $\bar{\rho} \geq 1.3$, resource capacities almost always require expansion. In this case, first-stage costs are dominated by second-stage costs and the optimal solution is quickly found. For values close to 1 the problems are difficult to solve. In this case, there is a delicate balance between start times of jobs (first-stage) and resource capacity expansions (second-stage), and the algorithm spends considerable time trying to minimize total cost. Note that in Figure 4.1b, T increases as $\bar{\rho}$ decreases. Larger T values result in larger problem sizes accounting for the increase in solution times for the smallest $\bar{\rho}$ values.

As $\bar{\rho}$ increases, the computational difficulty of the problem transitions from easy to hard to easy. These *phase transitions* have been observed in several NP-hard problems including resource-constrained project scheduling (Herroelen and Reyck, 1999). Existing measures of resource consumption and availability include the resource factor (RF) (Pascoe, 1966), resource strength (RS) (Cooper, 1976), and



(a) ample resources, $\bar{\rho} = 0.79$ (b) tight resources, $\bar{\rho} = 0.97$

Figure 4.2: Problems with small resource expansion display a peak at medium processing times whereas problems with larger resource expansion display an angular schedule.

resource-constrainedness (RC) (Patterson, 1976). RF is the average portion of resources requested per activity (job). However, it does not consider processing times of the activities. RS utilizes precedence-based early start schedules to determine a ratio of resource availability to peak consumption. While RS can be calculated for SMRCSP-U, the measure may lose some meaning since SMRCSP-U lacks precedence constraints. RC measures the ratio of average resource consumption of activities to resource availability. Our measure, which we refer to as the resource density (RD), is similar to RC but we capture total consumption and availability (rather than average). Hence, we incorporate the planning horizon T into RD as the planning horizon has an effect on resource availability.

Solution Characteristics: We now examine observed patterns in the optimal solutions. Figure 4.2 displays optimal job start times versus average processing times for two representative test problems. The lines indicate the latest time in which a job can start $(T - p_j^{max} + 1)$. The solution in Figure 4.2a corresponds to a problem with $\bar{\rho} = 0.79$, indicating that only small resource expansions are necessary for this schedule. In this case, solutions tend to start jobs with small and large processing times early but medium processing times later, resulting in a peak pattern around the middle. The solution in Figure 4.2b corresponds to a problem with $\bar{\rho} = 0.97$, indicating tight resources and larger resource expansion for

this schedule. Solutions in this case display an angular structure where jobs are either started as early as possible or as late as possible. The optimization tries to minimize the cost of resource expansion and does this by scheduling jobs as early or late as possible. Examining problems with large values of $\bar{\rho}$ (e.g., $\bar{\rho} = 1.3$), we again observed the angular pattern. Patterns in solutions have been observed in other scheduling problems. For instance, Denton and Gupta (2003) observed that scheduled appointment lengths are initially increasing and then decreasing as the schedule progresses in stochastic appointment scheduling with independent and identically distributed (i.i.d) appointment processing times.

4.2 Sampling-Based Approximate Solution Methodology

For problems with a moderate number of scenarios, the exact solution method of the L-shaped method can be used. However, modeling even small problems can result in an intractable number of scenarios. For example, a problem with 10 independent jobs where each job has 5 processing times results in $5^{10} = 9,765,625$ scenarios, which is too large to be solved exactly. In this case, we approximate SMRCSP-U with a sample average.

Let z^* be the optimal objective function value of the SMRCSP-U, i.e., $z^* = \min_{x \in X} E[F(x, \omega)]$, recall (3.8). When the number of scenarios is large, this problem becomes intractable and can be approximated by sampling from the distribution of $\omega \in \Omega$ resulting in a sampling approximation

$$z_n^* = \min_{x \in X} \frac{1}{n} \sum_{i=1}^{n} F(x, \omega^i), \tag{4.6}$$

with an optimal solution x_n^*. While other sampling schemes are possible, we assume $\omega^1, \omega^2, \ldots, \omega^n$ are i.i.d as $\omega \in \Omega$. As $n \to \infty$, z_n^* converges to z^* and similar consistency properties exist for x_n^*; see e.g., the survey by (Shapiro, 2003). For SMRCSP-U, under i.i.d sampling, since $|X| = |J|T$ is finite, consistency for z_n^* and x_n^* is achieved when $E[|F(x, \omega)|] < \infty$ for all $x \in X$, see Proposition 2.1 of (Kley-

wegt et al., 2001). This condition is satisfied for SMRCSP-U, for instance, when the random parameters have bounded support. However, practical implementations need a finite sample size and a reliable means to stop sampling. Bayraksan and Morton (2009) provide rules to increase sample sizes and to stop a sequential sampling procedure. In this section, we present a modified version of this procedure, show that similar desired theoretical properties hold for the modified version and present computational results for SMRCSP-U.

4.2.1 Sequential Sampling

The idea behind sequential sampling is simple. First, a candidate solution is generated. This is typically done by solving a sampling problem (4.6) with increasing sample size. Then, the quality of this candidate solution is evaluated. That is, a statistical estimator of the optimality gap of the candidate solution is formed. If the optimality gap estimate falls below a desired value, then the procedure stops. Otherwise, the procedure continues with a larger sample size. Per iteration, the procedure typically requires solution of at least two sampling problems, one for generating the candidate solution and at least one more for calculating the statistical estimator of the optimality gap. For integer programs, solution of these optimization problems can easily become burdensome. Since through sequential sampling, we aim to find a quality *approximate* solution, we can solve these sampling problems approximately, saving considerable computational effort. Our modifications allow this. In particular, when we solve the sampling problems approximately (e.g., 2.5% optimality), our observed quality for the solution obtained is slightly lower (e.g., within 3% optimality) since the modified statistical gap estimate includes the optimality gap of the sampling problem in addition to sampling error.

First, we introduce some relevant notation. Let X^* denote the set of optimal solutions to SMRCSP-U. Similarly, let $X^*(\delta)$ denote the set of $100\delta\%$-optimal solutions, i.e., $X^*(\delta) = \{x \in X : E[F(x,\omega)] - z^* \leq \delta|z^*|\}$ with $\delta > 0$. We also use the notation $x_n^*(\delta)$ to denote a $100\delta\%$-optimal solution to (4.6). Note that $x_n^*(0) = x_n^*$. For any $x \in X$, let $\mu_x = E[F(x,\omega)] - z^*$ and $\sigma^2(x) = \text{var}[F(x,\omega) - F(x_{\min}^*(\delta),\omega)]$,

where $x^*_{\min}(\delta) \in \text{argmin}_{y \in X^*(\delta)} \text{var}[F(x,\omega) - F(y,\omega)]$. In words, μ_x denotes the optimality gap of x, and $\sigma^2(x)$ denotes the "minimum" variance of the difference random variables, $F(x,\omega) - F(y,\omega)$ over $y \in X^*(\delta)$.

Optimality Gap Estimators: Given a candidate solution $x \in X$, we denote the point estimator of the optimality gap, μ_x, as $G_n(x)$ and the point estimator of the variance, $\sigma^2(x)$, as $s^2_n(x)$. To obtain these estimators, we modify the single replication procedure (SRP) developed in (Bayraksan and Morton, 2006) such that the sampling problem is solved to $100\delta\%$ optimality. Let $\bar{F}_n(x) = \frac{1}{n}\sum_{i=1}^n F(x,\omega^i)$. The resulting estimators are:

$$G_n(x) = \bar{F}_n(x) - \bar{F}_n(x^*_n(\delta)) \tag{4.7a}$$

$$s^2_n(x) = \frac{1}{n-1}\sum_{i=1}^{n}\left[(F(x,\omega^i) - F(x^*_n(\delta),\omega^i)) - G_n(x)\right]^2. \tag{4.7b}$$

Note that in (4.7) a sampling problem (4.6) with sample size n is solved approximately (e.g., via L-shaped) to obtain $x^*_n(\delta)$, and the same set of observations is used in all terms in (4.7) above. Since the sampling problem is solved to $100\delta\%$ optimality, we do not know z^*_n, rather, we know upper and lower bounds on it, i.e., $\underline{z}^*_n(\delta) \le z^*_n \le \overline{z}^*_n(\delta)$. Note that with the notation above, we can equivalently write $z^*_n = \bar{F}_n(x^*_n)$ and $\overline{z}^*_n(\delta) = \bar{F}_n(x^*_n(\delta))$. We point out that estimators in (4.7) are different than ε-optimal SRP discussed in (Bayraksan and Morton, 2006). In ε-optimal SRP, lower bounds, $\underline{z}^*_n(\delta)$, are used instead of upper bounds, $\overline{z}^*_n(\delta)$, in the second term of (4.7a). ε-optimal SRP is designed for very small ε , while δ is typically larger to allow for efficient solution of stochastic integer programs. With large δ, when lower bounds are used, the estimators become overly conservative and the sequential sampling procedure does not stop in a reasonable amount of time. With the use of upper bounds, (4.7a) now underestimates μ_x and we correct this by appropriately inflating the confidence interval on the candidate solution found.

Outline of the Procedure: At iteration $l \ge 1$, we select sample sizes, m_l and n_l, and use m_l to generate a candidate solution, denoted $\hat{x}_l$, and use n_l to evaluate the quality of this solution. To generate $\hat{x}_l$, we solve a sampling problem

Input:	Values for $h > h' > 0$, $\epsilon > \epsilon' > 0$, $0 < \alpha < 1$, $p > 0$ and $0 \leq \delta_1^l < 1$ such that $\delta_1^l \downarrow 0$ as $l \to \infty$ and $0 \leq \delta_2 < 1$.
Output:	A candidate solution, $\hat{x}_L$, and a $(1-\alpha)$-level confidence interval on μ_L.
Step 1.	(Initialize) Set $l = 1$, calculate n_l as given in (4.10) and set $m_l = 2n_l$. Sample i.i.d observations $\omega_1^1, \omega_1^2, \ldots, \omega_1^{m_l}$ and independently sample i.i.d observations $\omega_2^1, \omega_2^2, \ldots, \omega_2^{n_l}$ from the distribution of ω.
Step 2.	(Generate Candidate Solution) Use $\omega_1^1, \omega_1^2, \ldots, \omega_1^{m_l}$ to solve a sampling problem to $100\delta_1^l\%$ optimality to obtain $x^*_{m_l}(\delta_1^l)$. Set $\hat{x}_l = x^*_{m_l}(\delta_1^l)$.
Step 3.	(Assess Solution Quality) Given $\hat{x}_l$, use $\omega_2^1, \omega_2^2, \ldots, \omega_2^{n_l}$ to solve a sampling problem to $100\delta_2\%$ optimality to form G_l and s_l^2, given in (4.7).
Step 4.	(Check Stopping Criterion) If $\{G_l \leq h's_l + \epsilon'\}$, then set $L = l$, and go to 6.
Step 5.	(Increase Sample Size) Set $l = l + 1$ and calculate n_l according to (4.10). Set $m_l = 2 \cdot n_l$. Sample $m_l - m_{l-1}$ i.i.d observations, $\omega_1^{m_{l-1}+1}, \ldots, \omega_1^{m_l}$, and independently sample $n_l - n_{l-1}$ i.i.d observations $\omega_2^{n_{l-1}+1}, \ldots, \omega_2^{n_l}$ from the distribution of ω. Update δ_1^l. Go to 2.
Step 6.	(Output Approximate Solution) Output candidate solution $\hat{x}_L$ and a one-sided confidence interval on μ_L, $\left[0,\ hs_T + \epsilon + \delta_2 \overline{z}^*_{n_L}(\delta_2)\right]$

Figure 4.3: Sequential sampling procedure to obtain quality approximate solutions as implemented in our computational results.

with m_l observations to $100\delta_1^l\%$ optimality with $0 \leq \delta_1^l < 1$ such that $\delta_1^l \downarrow 0$ as $l \to \infty$ and set $\hat{x}_l = x^*_{m_l}(\delta_1^l)$. To evaluate $\hat{x}_l$'s quality, we solve a sampling problem with n_l observations to $100\delta_2\%$ optimality, $0 \leq \delta_2 < 1$, to obtain the estimators given in (4.7). In our implementation, we set $\delta_1^1 = \delta_2$ and divide δ_1^l by 2 if the procedure does not stop for the next 25 iterations. We augment the observations from the previous iteration by generating $m_l - m_{l-1}$ and $n_k - n_{k-1}$ additional independent observations. To simplify notation, from now on, we drop the dependence on the candidate solution, $\hat{x}_l$, and the sample size n_l, and simply denote $\mu_l = \mu_{\hat{x}_l}$, $\sigma_l^2 = \sigma^2(\hat{x}_l)$, $G_l = G_{n_l}(\hat{x}_l)$ and $s_l = s_{n_l}(\hat{x}_l)$.

A brief algorithmic statement of the sequential sampling procedure, as imple-

mented in our computational results, is given in Figure 4.3. The procedure terminates the first time G_l falls below $h's_l > 0$ plus a small positive number ϵ', i.e.,

$$L = \inf_{l \geq 1} \{l : G_l \leq h's_l + \epsilon'\}. \tag{4.8}$$

Let $h > h' > 0$ and $\epsilon > \epsilon' > 0$ (typically, epsilon terms are small, e.g., around 10^{-7}). When the algorithm terminates with approximate solution $\hat{x}_L$, a confidence interval on its optimality gap, μ_L, is given by

$$[0,\ hs_L + \epsilon + \delta_2 \overline{z}^*_{n_L}(\delta_2)], \tag{4.9}$$

where $\overline{z}^*_{n_L}(\delta_2) = \bar{F}_{n_L}(x^*_{n_L}(\delta_2))$ is obtained from G_L (see (4.7a)). We show below this confidence interval (CI) is asymptotically valid when the sample sizes n_l satisfy

$$n_l \geq \left(\frac{1}{h - h'}\right)^2 \left(c_p + 2p \ln^2 l\right), \tag{4.10}$$

where $c_p = \max\{2 \ln \left(\sum_{j=1}^{\infty} j^{-p \ln j} / \sqrt{2\pi}\alpha\right), 1\}$. Here, $p > 0$ is a parameter that affects the number of samples we generate. See (Bayraksan and Morton, 2009) for more details on the parameters.

Theoretical Properties: Two desired properties are: (i) the procedure stops in a finite number of iterations with probability one (w.p.1) and (ii) terminates with a quality solution with a desired probability. The theorem below summarizes conditions under which these properties hold.

Theorem 4.2.1. *Suppose $X \neq \emptyset$, $|X| < \infty$, $E[\sup_{x \in X} |F(x, \omega)|] < \infty$ and that the distribution of ω has a bounded support. Let $\omega_i^1, \omega_i^2, \ldots$, $i = 1, 2$, be i.i.d as ω. Let $\epsilon > \epsilon' > 0$, $p > 0$, $0 < \alpha < 1$ and $0 \leq \delta_2 < 1$ be fixed and let $0 \leq \delta_1^l < 1$ be such that $\delta_1^l \downarrow 0$ as $l \to \infty$. Consider the sequential sampling procedure where n_l is increased according to (4.10), $m_l \to \infty$ as $l \to \infty$ and the optimality gap estimators are calculated according to (4.7). If the procedure stops at iteration L according to (4.8) then, (i) $P(L < \infty) = 1$, and (ii) $\liminf_{h \downarrow h'} P\left(\mu_L \leq hs_L + \epsilon + \delta_2 \overline{z}^*_{n_L}(\delta_2)\right) \geq 1 - \alpha$.*

Proof.

$$
\begin{aligned}
\text{(i)} \qquad P(L=\infty) &\leq \liminf_{l\to\infty} P(G_l > h's_l + \epsilon') \\
&\leq \liminf_{l\to\infty} P(\bar{F}_{n_l}(\hat{x}_l) - \bar{F}_{n_l}(x^*_{n_l}) > \epsilon') \qquad (4.11) \\
&\leq \liminf_{l\to\infty} P(|\bar{F}_{n_l}(\hat{x}_l) - \bar{F}_{n_l}(x^*_{n_l}) - \mu_l| > \epsilon' - \mu_l), \qquad (4.12)
\end{aligned}
$$

where (4.11) follows from $h's_l > 0$, and $G_l \leq \bar{F}_{n_l}(\hat{x}_l) - \bar{F}_{n_l}(x^*_{n_l})$. The right hand side of (4.12) is zero, since $\bar{F}_{n_l}(x)$ converges to $E[F(x,\omega)]$ uniformly in X; $\bar{F}_{n_l}(x^*_{n_l}) = z^*_{n_l}$ converges to z^*; and $\mu_l \downarrow 0$ w.p.1, for $|X|$ finite, as $l \to \infty$, see Proposition 2.1 of (Kleywegt et al., 2001).

(ii) Let $\Delta h = h - h'$, $\Delta\epsilon = \epsilon - \epsilon'$ and define $D_l = \bar{F}_{n_l}(\hat{x}_l) - \bar{F}_{n_l}(x^*_{\min}(\delta_2))$, where $x^*_{\min}(\delta_2) \in \operatorname{argmin}_{y\in X^*(\delta_2)} \operatorname{var}[F(\hat{x}_l,\omega) - F(y,\omega)]$. Note that

$$
\begin{aligned}
P\left(\mu_L > hs_L + \epsilon + \delta_2 \overline{z}^*_{n_L}(\delta_2)\right) &\leq P\left(\mu_L \geq G_L + \delta_2 \overline{z}^*_{n_L}(\delta_2) + \Delta h s_L + \Delta\epsilon\right) \\
&\leq \sum_{l=1}^{\infty} P\left(G_l + \delta_2 \overline{z}^*_{n_l}(\delta_2) - \mu_l \leq -\Delta h s_l - \Delta\epsilon\right) \\
&\leq \sum_{l=1}^{\infty} P\left(D_l - \mu_l \leq -\Delta h s_l - \Delta\epsilon\right), \qquad (4.13)
\end{aligned}
$$

where (4.13) follows from the fact that $G_l + \delta_2 \overline{z}^*_{n_l}(\delta_2) \geq D_l$, w.p.1 for all l. The rest of the proof is same as in proof of Theorem 3 in (Bayraksan and Morton, 2009). We note that the required assumptions of the existence of moment generating functions, the central limit theorem for the difference random variables D_l and $\liminf_{n\to\infty} s_n(x) \geq \sigma^2(x)$, w.p.1 for all $x \in X$ are satisfied since ω has a bounded support, sampling is done in an i.i.d fashion and $|X|$ is finite. □

Part (ii) of Theorem 4.2.1 implies that the CI on the optimality gap of the candidate solution found by the procedure, given in (4.9), is a valid CI for values of h close to h'. We note that while this is an asymptotic result, CIs formed in much simpler settings such as for the mean of a random variable are often only asymptotically valid. Also note that the hypotheses of Theorem 4.2.1 are satisfied

by SMRCSP-U, that is, $X \neq \emptyset$, $|X| = |J|T < \infty$ and SMRCSP-U has relatively complete recourse, and ω has bounded support, thus, $E[\sup_{x \in X} |F(x, \omega)|] < \infty$, provided the cost terms are finite.

Remark 4.2.2. *When $\delta_2 = 0$, the above is same as the sequential procedure in (Bayraksan and Morton, 2009) with estimators SRP. However, for $\delta_2 > 0$, the estimators (4.7) and CI (4.9) are different. These changes, along with approximate solution of the sampling problems for generating the candidate solutions (i.e., $\delta_1^l > 0$) are essential to enable efficient solution of stochastic integer programs via a sampling method.*

4.2.2 Computational Results

We first run the sequential sampling procedure on the eight test problems (all but 40.5.1) from Section 4.1.3 that were solved to within 1% optimality. We use these small problems to *verify* the sampling procedure by comparing against the solutions found by the exact procedure. We then test the performance of the sampling procedure on larger problems with up to 10^{24} scenarios. We set $h = 0.312$ and $h' = 0.025$ resulting in $n_1 = 50$ and $m_1 = 100$ and set $\delta_1^1 = \delta_2 = \delta$. Following Bayraksan and Morton (2009), the other parameters are set to $\alpha = 0.10$, $\epsilon' = 1 \times 10^{-7}$, $\epsilon = 2 \times 10^{-7}$, and $p = 1.91 \times 10^{-1}$.

Table 4.4 presents the results of the sampling procedure performed on the test

$\delta = 2.5\%$	Average of 10 runs for each problem			
problem	L	m_L	CI%	gap%
20.5.1	2.0 ± 0.7	102.8 ± 1.9	3.78% ± 0.62%	0.37% ± 0.09%
20.5.2	1.6 ± 0.4	101.4 ± 1.0	3.27% ± 0.59%	0.19% ± 0.16%
20.5.3	1.5 ± 0.4	101.2 ± 1.0	5.32% ± 1.26%	1.99% ± 0.23%
40.5.2	2.2 ± 1.1	103.2 ± 3.2	3.55% ± 0.48%	0.24% ± 0.19%
40.5.3	1.6 ± 0.6	101.6 ± 1.5	4.25% ± 0.68%	1.41% ± 0.05%
80.5.1	2.3 ± 0.7	103.4 ± 1.9	3.04% ± 0.14%	0.28% ± 0.19%
80.5.2	1.4 ± 0.4	101.0 ± 1.0	2.86% ± 0.10%	-0.08% ± 0.13%
80.5.3	3.1 ± 1.1	106.0 ± 3.4	2.63% ± 0.51%	0.35% ± 0.27%

Table 4.4: Results from the sampling procedure when the sampling problems were solved to 2.5% optimality. 79/80 = 99% of runs resulted in actual optimality gaps within the gap estimate.

problems from Section 4.1.3. Column "L" reports the number of iterations at termination, column "m_L" lists the sample size used to generate the solution found by the procedure ($m_L = 2n_L$), and column "CI%" reports the output confidence interval's width given in (4.9) divided by the exact solution objective from the solutions on the right-hand-side of Table 4.3. The "gap%" reports the percentage difference between the actual objective function values of the sampling solution and the exact solution. In some cases, the sampling procedure found a better solution than the exact solution as indicated by negative gap% values. All values are reported with a 90% CI around the means. While the output confidence interval on the gap is slightly inflated, the optimality gap compared to the exact objective remains small. The procedure is run with $1 - \alpha = 0.90$ (see part (ii) of Theorem 4.2.1), resulting in an empirical coverage probability of 0.99 ± 0.01 (79 out of 80 runs). Note that this is an empirical coverage probability ($\hat{p}$) along with a 90% CI half-width from 80 runs ($\pm 1.645\sqrt{\hat{p}(1-\hat{p})/80}$).

The main advantage of the sampling procedure is to allow approximate solution of problems of practical scale. Therefore, we now examine the algorithm's performance on problems with a large number of scenarios. We modified the test problems from Section 4.1.3 giving each job two processing time scenarios for a total of 2^J scenarios, for $J = 20, 40$, and 80. We label the modified test problems with a large number of scenarios by adding * to the instance number. For instance, problem 20.5.1 has $2^{10} = 1024$ scenarios and problem 20.5.1* has 2^{20} scenarios. Initially, all modified problems reached the 10 hour time limit on the sampling procedure. However, we found that this was due to $\bar{\rho}$ values close to 1. By extending the planning horizon from $T = 50$ to $T = 60$, we were able to lower the densities to around 0.8 - 0.9 (see Section 4.1.4). The slight change in density resulted in a dramatic difference in solution times that are reported in Tables 4.5-4.7. We report on selected problems for brevity. It is impossible to solve even the 20 job problem exactly since with 2^{20} scenarios the computer runs out of memory at the first iteration. Therefore, we cannot report the actual quality of the solutions but we can provide an upper bound on the quality by comparing the sampling solution CI to the solution of the expected

$\delta = 1\%$	Average of 10 runs for each problem					
problem	$\lvert\Omega\rvert$	time	L	m_L	CI	$\overline{\text{CI}}\%$
20.5.1*	1.05×10^6	24,426 ± 5657	1.7 ± 0.3	101.4 ± 0.5	10.3 ± 2.6	2.20% ± 0.56%
40.5.2*	1.10×10^{12}	2556 ± 2908	1.4 ± 0.4	101.0 ± 1.0	23.2 ± 4.7	1.83% ± 0.37%
80.5.1*	1.21×10^{24}	25,964 ± 7757	2.2 ± 0.6	103.4 ± 1.9	44.5 ± 3.9	1.90% ± 0.17%

Table 4.5: Solving problems to 1% optimality results in solutions with optimality gaps of around 2%.

$\delta = 2\%$	Average of 10 runs for each problem					
problem	$\lvert\Omega\rvert$	time	L	m_L	CI	$\overline{\text{CI}}\%$
20.5.1*	1.05×10^6	2692 ± 2411	3.6 ± 1.5	107.4 ± 4.3	12.8 ± 3.1	2.73% ± 0.66%
40.5.2*	1.10×10^{12}	437 ± 125	3.3 ± 1.5	106.4 ± 4.2	26.1 ± 4.2	2.05% ± 0.33%
80.5.1*	1.21×10^{24}	938 ± 223	3.8 ± 1.6	107.8 ± 4.5	46.8 ± 5.1	1.99% ± 0.22%

Table 4.6: Lowering δ to 2% reduces computation time by an order of magnitude and still provides quality solutions.

value problem, which provides a valid lower bound on the stochastic problem. The expected value processing times were calculated as $\lfloor \bar{p}_j \rfloor$ for $j = 1, \ldots, J$. The "$\overline{\text{CI}}\%$" column is CI divided by the objective from the expected value problem.

Tables 4.5-4.7 indicate that larger values of δ can considerably reduce solution times while still obtaining high-quality solutions. In particular $\delta = 2.5\%$ problems are solved very quickly. We note that $\overline{\text{CI}}\%$ provides an upper bound on CI% and in light of Table 4.4, we believe the actual solutions are much closer to optimality.

4.3 Concluding Remarks

SMRCSP-U addresses the problem of determining an optimal schedule of multiple resource consuming jobs under uncertainty of processing times, resource consumptions, and resource capacities while allowing temporary resource capacity expansions for a cost. SMRCSP-U arises in operating room scheduling and many manufacturing and consulting applications. The single resource version arises in shipping berth allocation, semiconductor circuit design, and multiprocessor computer systems. The model presented creates schedules that minimize expected cost under uncertainty, and the formulation is generic allowing application of proposed methods to a variety of different objectives. The algorithm does not require input schedules to generate

$\delta = 2.5\%$	Average of 10 runs for each problem					
problem	$\lvert\Omega\rvert$	time	L	m_L	CI	$\overline{\text{CI}}\%$
20.5.1*	1.05×10^6	668 ± 348	1.4 ± 0.4	101.0 ± 1.0	21.3 ± 1.9	4.54% ± 0.40%
40.5.2*	1.10×10^{12}	343 ± 24	2.9 ± 0.9	105.2 ± 2.4	44.5 ± 1.8	3.50% ± 0.14%
80.5.1*	1.21×10^{24}	694 ± 46	2.5 ± 0.8	104.4 ± 2.2	85.4 ± 2.8	3.64% ± 0.12%

Table 4.7: Quality solutions are quickly found for problems with up to 10^{24} scenarios.

the proactive schedules. We solve the model with the L-shaped method, and we significantly improve the performance of the L-shaped method with GUB branching, trust regions, LP warm starting, and approximately solving the master problem. The combination of enhancements resulted in more than a 70% reduction in total solution time over 9 test problems. As our model handles uncertainties in processing times, resource consumption, and resource availabilities, problems of practical scale can quickly become intractable due to the prohibitively large number of scenarios. To alleviate this difficulty, we present and prove desired properties of a sequential sampling procedure that obtains high quality solutions without having to evaluate every scenario. Our computational results with up to 10^{24} scenarios indicate that quality solutions can be found quickly ($\leq$ 1000 seconds) with this methodology. We conclude this chapter by summarizing insights gained:

- Problems are difficult to solve when the resource density, a measure of the ratio of total resource consumption to availability, is close to 1. When the density is ≤ 0.8 or ≥ 1.3, problems are typically computationally easier to solve.
- A small increase in T can lower resource densities to a more desirable level. This also indicates that the jobs can be more efficiently scheduled within a longer time horizon.
- Alternatively, the number of jobs can be reduced by changing the problem to a job selection and scheduling problem. This can be achieved by changing "=" to "$\leq$" in constraints (3.2) and changing the objective to maximize some profit measure. This change does not affect the model structure, and the solution method developed in this chapter remains valid.
- Schedules resulting in $w_{tk}(\omega) > 0$ for a sufficiently large number of $\omega \in \Omega$ indicate resource bottlenecks in resource k at time t. Resources k with $z_{tk}(\omega) >$

0 for a sufficiently large number of $\omega \in \Omega$ indicate resources that may benefit from permanent expansion.

- When the resources are tight ($\bar{\rho} \geq 1$), jobs are optimally started either as early as possible or as late as possible. As the resources become more available, jobs with medium processing times tend to start later while jobs with small and large processing times tend to start early.
- The sequential sampling procedure may be quite slow to converge when resource densities are close to 1. By a small increase in T, the procedure stops in a reasonable time. Preliminary computational results indicate 110 scenarios can be enough to find a quality approximate solution for this problem class.

CHAPTER 5

SOLUTION METHODS FOR SMRCSP-JB

The disjunctive decomposition (D^2) algorithm has emerged as a powerful tool to solve stochastic integer programs. In this chapter, we utilize D^2 to solve the SMRCSP-JB and develop several computational enhancements to D^2. First, we provide a detailed review of the D^2 algorithm. Then, we explore the use of a cut generation problem restricted to a subspace of the variables, which yields significant computational savings. Next, we examine the generalized upper bound (GUB) constraints in the second-stage of SMRCSP-JB and exploit this structure to generate cuts based on alternative disjunctions. We establish convergence of D^2 with our enhancements. Finally, we present computational results on a set of SMRCSP-JB test problems and show that our enhancements reduce computation time on average by 45% for the set of test problems.

5.1 Review of Disjunctive Decomposition

Stochastic mixed integer programs (SMIP) comprise one of the most difficult classes of optimization problems as they possess the large-scale nature of stochastic programs and the computational challenges of integer programming. As mentioned in Chapter 2, integer variables in the second-stage lead to a nonconvex and discontinuous recourse function, $E[f(x, \tilde{\omega})]$, that is often quite difficult to optimize.

In our discussion of the D^2 algorithm, we consider the following SMIP:

$$\min_{x \in X \cap \mathbb{B}} cx + E[f(x, \tilde{\omega})], \tag{5.1}$$

where $X = \{x \in \mathbb{R}^{d_x} : Ax \geq b, x \geq 0\}$, $\mathbb{B} \subset \mathbb{R}^{d_x}$ is the set of binary vectors with dimension d_x, $\tilde{\omega}$ is a random vector whose distribution is assumed known and does

not depend on x. The support of $\tilde{\omega}$ is denoted by Ω and a realization of $\tilde{\omega}$ is denoted by ω. E is the expectation operator and expectation in (5.1) is taken with respect to the distribution of $\tilde{\omega}$. For any $\omega \in \Omega$

$$\begin{aligned} f(x,\omega) = \min_{y} \quad & qy \\ \text{s.t.} \quad & Wy \geq r(\omega) - T(\omega)x, \\ & y \geq 0,\ y_j \in \{0,1\} \text{ for } j \in B, \end{aligned} \tag{5.2}$$

where B is an index specifying the binary variables. Problem (5.2) is referred to as the second-stage subproblem. For a given $(x,\omega) \in (X,\Omega)$, it is a 0-1 mixed-integer program (MIP). Throughout the chapter, we refer to (5.2) as MIP subproblems. Above, y is the decision vector corresponding to a particular scenario subproblem and hence depends on ω. However, we suppress this in (5.2). In the rest of the chapter, we use $y(\omega)$ interchangeably to represent the subproblem decisions. The SMRCSP-JB has deterministic q and W although concepts presented here may be applied to problems with random cost vectors, $q(\omega)$, and random recourse matrix, $W(\omega)$, with appropriate technical modifications (Ntaimo, 2009). We assume Ω is a finite set, i.e. $|\Omega| < \infty$, and that the feasible region of (5.2) is bounded and nonempty for all $(x,\omega) \in (X,\Omega)$. The latter condition results in $f(x,\omega) < \infty$ for all $(x,\omega) \in (X,\Omega)$ and may be assured by adding a variable with arbitrarily high cost to each constraint in the subproblem to ensure feasibility.

We remind the reader that the D^2 algorithm presented below is restricted to problems with only binary first-stage variables. SMRCPS-JB presented in Chapter 3 has continuous first-stage variables z_{tk}, $t = 1,\ldots,T_0$, $k = 1,\ldots,K$. However, this does not affect D^2 since the continuous variables do not affect the second-stage. That is, $f(x,\omega)$ only depends on x but not on the other first-stage variables z_{tk}. Moreover, convergence of D^2 for this problem is not affected for the same reason and by the fact that the master problem can be solved in a finite number of iterations (Step 4 of D^2 in Section 5.1.1).

5.1.1 Overview of the Algorithm

The D^2 algorithm is a decomposition-based algorithm similar to Benders decomposition and the L-shaped method. At each iteration of D^2, LP relaxations of MIP subproblems are solved. If the subproblem solution does not satisfy integrality, disjunctive cuts of the form $\pi^{\mathrm{T}} y \geq \pi_0(x, \omega)$ are generated to convexify the subproblem. As the algorithm progresses, the convex approximation of the subproblem feasible region improves. Under the fixed recourse assumption, the left-hand-side coefficients, π, remain the same for all scenarios with only the right-hand-side of the cuts depending on the subproblem. The following theorem, referred to as the Common Cut Coefficients (C^3) theorem, allows the same set of cut coefficients, π, to be used for each scenario subproblem.

Theorem 5.1.1. *(Sen and Higle, 2005). Consider the stochastic program with fixed recourse as stated in (5.1)-(5.2). For $(x, \omega) \in (X, \Omega)$, let $Y(x, \omega) = \{y : Wy \geq r(\omega) - T(\omega)x,\ y \geq 0,\ y_j \in \{0, 1\},\ j \in B\}$ denote the set of mixed-integer feasible solutions for the second-stage MIP. Suppose that $\{C_h, c_h\}_{h \in H}$ is a finite collection of appropriately dimensioned matrices and vectors such that for all $(x, \omega) \in (X, \Omega)$,*

$$Y(x, \omega) \subseteq \cup_{h \in H} \{y \geq 0 : C_h y \geq c_h\}.$$

Let

$$S_h(x, \omega) = \{y \geq 0 : Wy \geq r(\omega) - T(\omega)x, C_h y \geq c_h\},\ \text{and}\ S(x, \omega) = \cup_{h \in H} S_h(x, \omega).$$

Let $(\bar{x}, \bar{\omega})$ be given and suppose that $S_h(\bar{x}, \bar{\omega})$ is nonempty for all $h \in H$ and $\pi y \geq \pi_0(\bar{x}, \bar{\omega})$ is a valid inequality for $S(\bar{x}, \bar{\omega})$. Then there exists a function, $\pi_0 : X \times \Omega \mapsto \mathbb{R}$ such that for all $(x, \omega) \in (X, \Omega)$, $\pi y \geq \pi_0(x, \omega)$ is a valid inequality for $S(x, \omega)$.

Proof. See Sen and Higle (2005). □

The common cut coefficients, π, can be found by solving an LP called the cut generation linear program (CGLP). However, the right-hand-side, $\pi_0(x, \omega)$, is a piece-

wise linear concave function in x, which is not computationally efficient. Since only binary first-stage solutions are relevant, $\pi_0(x, \omega)$ can be linearized in x. This is achieved by solving a set of LPs for each $\omega \in \Omega$, called the right-hand-side linear programs (RHSLPs). The resulting functions are denoted $\pi_c(x, \omega)$, and cuts of the form $\pi^{\mathrm{T}} y \geq \pi_c(x, \omega)$ are added to convexify the subproblem feasible regions.

For a given x^i at iteration i, define the scenario subproblem LP relaxation as

$$\begin{aligned} f_c(x^i, \omega) = \min_{y} \quad & qy \\ \text{s.t.} \quad & W^i y \geq \rho_c^i(x^i, \omega), \\ & y \geq 0, \end{aligned} \tag{5.3}$$

where $\rho_c^i(x^i, \omega) = r^i(\omega) - T^i(\omega)x^i$. The elements are initialized with $W^1 = W$, $r^1(\omega) = r(\omega)$, and $T^1(\omega) = T(\omega)$. We also assume that W and $r(\omega)$ include the constraints $-y_j \geq 1$ for $j \in B$. If a new cut is generated at iteration i, the new cut coefficients, π^i, are appended to W^i to form W^{i+1}. For each scenario, elements of $\pi_c^i(x, \omega)$ generated from RHSLP are appended to $r^i(\omega)$ and $T^i(\omega)$ forming $r^{i+1}(\omega)$ and $T^{i+1}(\omega)$, respectively. The D^2 algorithm can be summarized as follows.

0. Initialize. Let $\epsilon > 0$ and $x^1 \in X \cap \mathbb{B}$ be given. Let $i \leftarrow 1$ and initialize an upper bound $V_0 = \infty$ and a lower bound $v_0 = -\infty$. Set $W^1 \leftarrow W$, $T^1(\omega) \leftarrow T(\omega)$, and $r^1(\omega) \leftarrow r(\omega)$.

1. Solve one LP Subproblem for each $\omega \in \Omega$. Set $V_i \leftarrow V_{i-1}$. Use W^i and $\rho_c^i(x^i, \omega)$ to solve subproblem LP relaxation (5.3) for each $\omega \in \Omega$. If $y^i(\omega)$ satisfies integrality for all $\omega \in \Omega$, $V_i \leftarrow \min\{cx^i + E[f(x^i, \tilde{\omega})], V_i\}$ and go to step 4.

2. Generate Cuts.

2.1. Solve CGLP. Choose disjunction variable $j(i) \in B$ and obtain π^i by solving CGLP formulated using W^i and cuts generated from variables with index less than $j(i)$. If a nonnegative objective is found, arbitrarily drop one scenario from the conditional expectation and reoptimize, repeating

until a negative objective is found. Once found, define W^{i+1} by appending π^i to W^i.

2.2. Solve RHSLP. For each $\omega \in \Omega$, solve $\pi_c^i(x^i, \omega)$ LP. Use solution to update $\rho_c^{i+1}(x^i, \omega)$.

3. Update and Solve LP Subproblem for each $\omega \in \Omega$. For each $\omega \in \Omega$ solve the updated LP with W^{i+1} and $\rho_c^{i+1}(x^i, \omega)$. If $y^i(\omega)$ satisfies integrality for all scenarios, then $V_i \leftarrow \min\{cx^i + E[f(x^i, \tilde{\omega})], V_i\}$.

4. Update and Solve the Master Problem. Using the dual multipliers from the most recently solved subproblems (either step 1 or step 3), add L-shaped optimality cut to the master. Obtain x^{i+1} by solving the MIP master problem and let v_i be the optimal value of the master. If $V_i - v_i \leq \epsilon \cdot \max\{|V_i|, |v_i|\}$, stop. Otherwise, $i \leftarrow i + 1$ and go to step 1.

Upper bounds can be calculated only when all subproblems are solved to integrality. In practice, when $x^i = x^{i+1}$, the subproblems can be solved to integrality to calculate the upper bound and to speed up convergence of the algorithm.

5.1.2 Cut Generation

In this section, we discuss the cut generation phase of the algorithm in more detail. We especially focus on Step 2.1 of D^2, namely the CGLP and its solution. In the rest of the chapter, we will refer back to these details to explain our enhancements to D^2.

In step 1 at iteration i of D^2, if a fractional solution exists we add disjunctive cuts to eliminate this fractional solution and convexify the subproblems. Let $j(i) \in B$ denote an index j for which $y_j^i(\omega)$ is fractional for some $\omega \in \Omega$. To eliminate this fractional solution, a disjunction of the form

$$S^i(x^i, \omega) = S_{0,j(i)}(x^i, \omega) \cup S_{1,j(i)}(x^i, \omega),$$

where

$$S_{0,j(i)}(x^i,\omega) = \{y \geq 0 : W^i y \geq \rho_c^i(x^i,\omega), \tag{5.4a}$$

$$-y_{j(i)} \geq 0\}, \tag{5.4b}$$

$$S_{1,j(i)}(x^i,\omega) = \{y \geq 0 : W^i y \geq \rho_c^i(x^i,\omega), \tag{5.4c}$$

$$y_{j(i)} \geq 1\} \tag{5.4d}$$

is used.

Let λ_{01} denote the vector of multipliers associated with the right-hand-side constraints in (5.4a) and λ_{02} the scalar multiplier associated with the constraint in (5.4b). Similarly, let λ_{11} and λ_{12} denote the multipliers associated with the right-hand-side constraints in (5.4c) and (5.4d), respectively. Define

$$I_j^i = \begin{cases} 1, & \text{if } j = j(i) \\ 0, & \text{otherwise,} \end{cases}$$

and let W_j^i denote the j^{th} column of W^i. Then the CGLP is the stochastic version of the LP used to generate the lift-and-project cuts. This problem can be viewed as a two-stage stochastic linear program (Yang and Sen, 2009), and we present it below in this form.

$$\underline{\text{CGLP}}: \quad \min_{\pi,\lambda} \quad E[y^i(\tilde{\omega})]\pi + E[-\pi_0(\lambda,\tilde{\omega})] \tag{5.5a}$$

$$\text{s.t.} \quad \pi_j \geq \lambda_{01} W_j^i - I_j^i \lambda_{02}, \qquad \forall j, \tag{5.5b}$$

$$\pi_j \geq \lambda_{11} W_j^i + I_j^i \lambda_{12}, \qquad \forall j, \tag{5.5c}$$

$$-1 \leq \pi_j \leq 1, \qquad \forall j, \tag{5.5d}$$

$$\lambda_{01}, \lambda_{02}, \lambda_{11}, \lambda_{12} \geq 0, \tag{5.5e}$$

where
$$-\pi_0(\lambda,\omega) = \min_{z} \quad -z \tag{5.5f}$$
$$\text{s.t.} \quad -z \geq -\lambda_{01}\rho_c^i(x^i,\omega), \tag{5.5g}$$
$$-z \geq -\lambda_{11}\rho_c^i(x^i,\omega) - \lambda_{12}, \tag{5.5h}$$
$$-1 \leq z \leq 1. \tag{5.5i}$$

The CGLP given in (5.5) aims to maximize the expected depth of the cut generated. The objective in (5.5a) is typically a conditional expectation, taken over the set of scenarios $\omega \in \Omega$ such that $y^i_{j(i)}(\omega)$ is fractional. First-stage decisions are π and λ and recourse decisions determine $\pi_0(\lambda,\omega)$. Note that $\pi_0(\lambda,\omega)$ in (5.5) is the $\pi_0(x,\omega)$ of the disjunctive cut $\pi^{\mathrm{T}}y \geq \pi_0(x,\omega)$ formed as a result of solving the CGLP. In (5.5), the alternative notation $\pi_0(\lambda,\omega)$ is used to present it as a two-stage stochastic linear program with first-stage decisions λ. The CGLP can be solved by the L-shaped method. The simple structure of the second-stage allows forming feasibility and optimality cuts without requiring an LP solver. Below, we briefly explain how to obtain the cuts.

Suppose at iteration t of the L-shaped method used to solve the CGLP, the first-stage decision λ^t leads to infeasibility in the second-stage. Then a feasibility cut needs to be added to eliminate this solution. Denote the right-hand-side of (5.5g) as $\phi(\lambda,\omega) = -\lambda_{01}\rho_c^i(x^i,\omega)$ and similarly the right-hand-side of (5.5h) as $\psi(\lambda,\omega) = -\lambda_{11}\rho_c^i(x^i,\omega) - \lambda_{12}$. If $\phi(\lambda^t,\omega) \geq \psi(\lambda^t,\omega)$, then a feasibility cut of the form

$$\lambda_{01}\rho_c^i(x^i,\omega) \geq -1$$

is generated. Otherwise, a feasibility cut of the form

$$\lambda_{11}\rho_c^i(x^i,\omega) + \lambda_{12} \geq -1 \tag{5.6}$$

is generated.

When λ^t at iteration t of the L-shaped method yields a feasible solution to (5.5f) - (5.5i), the optimal value of the second-stage can be calculated as $-\pi_0(\lambda^t,\omega) =$

$\max\{\phi(\lambda^t,\omega),\psi(\lambda^t,\omega),-1\}$ for all $\omega\in\Omega$. In this case, an optimality cut is generated. Denote the subgradient of $-\pi_0(\lambda,\omega)$ at λ^t as $\nabla\pi_0^\omega(\lambda^t)$. This subgradient consists of four subvectors $\nabla\pi_0^\omega(\lambda^t)=[\nabla_{01}^\omega(\lambda^t),\nabla_{11}^\omega(\lambda^t),\nabla_{02}^\omega(\lambda^t),\nabla_{12}^\omega(\lambda^t)]^{\mathrm{T}}$. The subgradient can be determined by the following cases:

Case 1. If $\phi(\lambda^t,\omega),\psi(\lambda^t,\omega)<-1$, then $\nabla\pi_0^\omega(\lambda^t)=[0,0,0,0]^{\mathrm{T}}$.

Case 2. If $\phi(\lambda^t,\omega)>\psi(\lambda^t,\omega)$ and $\phi(\lambda^t,\omega)\geq-1$, then $\nabla\pi_0^\omega(\lambda^t)=-[\rho_c^i(x^i,\omega),0,0,0]^{\mathrm{T}}$.

Case 3. If $\phi(\lambda^t,\omega)<\psi(\lambda^t,\omega)$ and $\psi(\lambda^t,\omega)\geq-1$, then $\nabla\pi_0^\omega(\lambda^t)=-[0,\rho_c^i(x^i,\omega),0,1]^{\mathrm{T}}$.

Case 4. If $\phi(\lambda^t,\omega)=\psi(\lambda^t,\omega)\geq-1$, then $\nabla\pi_0^\omega(\lambda^t)=-[0.5\rho_c^i(x^i,\omega),0.5\rho_c^i(x^i,\omega),0,0.5]^{\mathrm{T}}$.

Case 1 occurs when constraints (5.5g) and (5.5h) are slack. Case 2 occurs when constraint (5.5g) is binding, and Case 3 occurs when and (5.5h) is binding. Case 4 can be thought of as weighted sum of the gradients in Case 3 and Case 4 where the weights must sum to 1. The subgradient of $E[-\pi_0(\lambda,\tilde{\omega})]$ is denoted by $\nabla E\pi_0(\lambda)$ and at λ^t is equal to $\nabla E\pi_0(\lambda^t)=\sum_{\omega\in\Omega}\Pr(\omega)\nabla\pi_0^\omega(\lambda^t)$, where $\Pr(\omega)$ is the probability of scenario $\omega\in\Omega$. The optimality cut has the form

$$\theta\geq E[-\pi_0(\lambda^t,\tilde{\omega})]+\nabla E\pi_0(\lambda^t)(\lambda-\lambda^t),$$

where θ is a variable that replaces $E[-\pi_0(\lambda,\tilde{\omega})]$ in the L-shaped master problem used to solve the CGLP given in (5.5).

The CGLP is based on a 0-1 disjunction of the variable $y_{j(i)}$. In the next section, we will explore alternative disjunctions based on the GUB structure. If the optimal objective function value of CGLP is negative, then $\pi^{\mathrm{T}}y\geq\pi_0(x,\omega)$ will cut off the current fractional solution y^i. Since $\pi_0(x,\omega)$ is a piecewise linear concave function in x, Step 2.2 of D^2 modifies this to be affine in x. This is achieved by solving a series of LPs for each $\omega\in\Omega$ called RHSLP, which takes as input x^i and optimum λ^i

values from CGLP and outputs the function $\pi_c^i(x, \omega)$. This function is of the form $\pi_c^i(x, \omega) = \bar{r}^i(\omega) - \bar{T}^i(\omega)x$. For details on the RHSLP, see (Sen and Higle, 2005).

5.2 D^2 with Restricted CGLP and GUB Disjunctions

In this section, we discuss several enhancements to D^2. First, we explore generation of cuts using a restricted CGLP. We then introduce disjunctions formed on GUB sets and describe how to incorporate GUB disjunctions into the D^2 algorithm. Finally, we establish convergence of D^2 with GUB disjunctions and with the restricted CGLP.

5.2.1 Restricted Cut Generation Problem

The CGLP becomes quite large for even two disjunctions. For instance at iteration i of D^2, the CGLP has more than twice the nonzeros of W^i. As disjunctive cuts are added in D^2 iterations, and hence, as W^i grows larger, the CGLP can become a bottleneck. However, it is possible to generate disjunctive cuts from a smaller LP by working in the subspace defined by the fractional components of y and lifting the cut into the original space. This method was first used in a cutting plane algorithm (Balas et al., 1993) and later in branch-and-cut (Balas et al., 1996). Here, we extend it to D^2. Our discussion in this section will be focused on the 0-1 disjunctions presented in Section 5.1. Later, as we introduce alternative disjunctions based on GUB constraints, we will discuss necessary modifications.

Recall that B is an index set identifying the binary variables in the second-stage and let C be the index set identifying the remaining continuous variables. The set $B \cup C$ contains the indices of all decision variables in (5.2). We will work with a subset of these variables and denote the index set for this subset as $\mathcal{R} \subseteq (B \cup C)$. Let $\{y^i(\omega)\}_{\omega \in \Omega}$ be the optimal solution to (5.3) for all $\omega \in \Omega$ for a given x^i at iteration

i of D^2. Set $\mathcal{R}$ is formed as follows.

$$\begin{aligned}\mathcal{R} =&\{j \in B : 0 < y_j^i(\omega) < 1 \text{ for at least one } \omega \in \Omega\} \cup \\ &\{j \in C : y_j^i(\omega) > 0 \text{ for at least one } \omega \in \Omega\} \cup \\ &\{j \in B : y_j^i(\omega) \text{ is in a disjunction}\}.\end{aligned} \tag{5.7}$$

In words, the set $\mathcal{R}$ contains indices of all binary variables that are fractional in at least one scenario and only indices of continuous variables that are positive in at least one scenario. Finally, $\mathcal{R}$ must include all indices used to form the disjunction to guarantee a valid cut. When 0-1 disjunctions are used as in (5.5), the index $j(i) \in B$ corresponding to the fractional variable $y_{j(i)}$ is the only index used to form a disjunction, which is already included in $\mathcal{R}$. However, below we will use disjunctions on the GUB structure, and these disjunctions may also contain binary variables that are 0 in the current solution. Nevertheless, indices of these variables must be included in $\mathcal{R}$ to ensure the validity of the generated cut from the restricted CGLP.

We will work with a vector $y^{\mathcal{R}}$ that only contains second-stage decision variables indexed by $\mathcal{R}$. If a variable is not contained in $y^{\mathcal{R}}$, then we can assume it is zero since for those $j \in B$ such that $y_j^i(\omega) = 1$ for all $\omega \in \Omega$, we can complement the variable by setting column W_j^i to $-W_j^i$ and setting $\rho_c^i(x^i, \omega)$ to $\rho_c^i(x^i, \omega) - W_j^i$, $\omega \in \Omega$. Let $\pi^{\mathcal{R}} y^{\mathcal{R}} \geq \pi_0(x, \omega)$ be the cut generated in the subspace. Cut generation can be done in a similar way as discussed in Section 5.1.2 by restricting the CGLP to use only columns indexed by $\mathcal{R}$. Next, we need to lift this cut to the space of the original second-stage variables. The right-hand-side $\pi_0(x, \omega)$ remains the same but we need cut coefficients for variables that were ignored during the restricted CGLP. This can be done as follows:

$$\pi_j = \begin{cases} \pi_j^{\mathcal{R}} & \text{if } j \in \mathcal{R} \\ \max\{\lambda_{01} W_j^i, \lambda_{11} W_j^i\} & \text{if } j \notin \mathcal{R}. \end{cases} \tag{5.8}$$

When $j \notin \mathcal{R}$, the lifted cut coefficients π_j may be outside the normalization range

of $[-1, 1]$ specified in (5.5d). However, Balas et al. (1993) show finite termination of the cutting plane algorithm when (5.5d) is replaced with $-1 \leq \pi_j \leq 1$ for $j \in \mathcal{R}$. Working with the restricted CGLP eliminates many rows and columns from the CGLP resulting in significant performance improvements, which we report in Section 5.3.

The cut $\pi y \geq \pi_0(x, \omega)$ with coefficients obtained from (5.8) can be strengthened by using the integrality conditions on variables other than those used to form the disjunction. The following result, adapted to the D^2 framework, was originally shown by Balas (1979) and Balas and Jeroslow (1980) in the context of deterministic MIPs.

Proposition 5.2.1. *Let $\pi_0(x, \omega)$, λ_{01}, λ_{02}, λ_{11}, and λ_{12} be found by solving the restricted CGLP in (5.5) using only columns indexed by $\mathcal{R}$, and let π be obtained through (5.8). Then, the inequality $\gamma y \geq \pi_0(x, \omega)$ is valid for the subproblem with $(x, \omega) \in (X, \Omega)$, where*

$$\begin{aligned} \gamma_j &= \min\{\lambda_{01} W_j^i + \lambda_{02} \lceil \bar{m}_j \rceil, \lambda_{11} W_j^i - \lambda_{12} \lfloor \bar{m}_j \rfloor\}, && j \in B, \\ \gamma_j &= \max\{\lambda_{01} W_j^i, \lambda_{11} W_j^i\}, && j \in C, \end{aligned}$$

and

$$\bar{m}_j = \frac{\lambda_{11} W_j^i - \lambda_{01} W_j^i}{\lambda_{02} + \lambda_{12}}.$$

Proof. Follows from Theorem 2.2 in Balas et al. (1996). □

The strengthening procedure is performed after the CGLP has been solved, i.e., λ_{01}, λ_{02}, λ_{11}, and λ_{12} have already been determined. Since the strengthened cuts take the form $\gamma y \geq \pi_0(x, \omega)$, the smallest possible coefficients γ_j, $j \in B \cup C$ will lead to tighter cuts. The parameter $\bar{m}_j$ is chosen to make γ_j as small as possible for $j \in B$. When $\lambda_{02} = \lambda_{12} = 0$, the value of $\bar{m}_j$ will not affect γ_j, and therefore, the strengthening procedure cannot be applied. In our computational experiments, we tested the strengthening procedure within D^2 using 0-1 and GUB disjunctions.

5.2.2 GUB Disjunctions

Previous D^2 research has focused on using disjunctive cuts formed on binary disjunctions i.e., $(y_j \leq 0) \cup (y_j \geq 1)$ for $j \in B$ over the second-stage feasible set. In this section, we investigate the use of disjunctions on generalized upper bound (GUB) constraints in the MIP subproblems. GUB constraints take the form of $\sum_{j \in G} y_j = 1$ with $y_j \in \{0, 1\}$ for $j \in G \subseteq B$. GUB constraints appear in many stochastic optimization problems. The GUB structure often contains valuable information that can be used to increase effectiveness of algorithms. For example, GUB cover cuts (Wolsey, 1990) are much more effective than regular cover cuts for problems with GUB structures. Also, in branch-and-bound for deterministic MIP problems, branching on GUB constraints can be more effective than standard 0-1 branching for many problems (Wolsey, 1998). We have applied this to SMRCSP-U in Chapter 4 and observed computational speedups. Extending the idea of GUB branching to disjunctive cuts, we use disjunctions on GUB sets to convexify the MIP subproblem feasible regions. Our initial computational results suggest that disjunctions formed on GUB sets can be more effective than disjunctions formed on 0-1 variables for MIP set convexification within the D^2 algorithm for problems with this structure.

Consider a GUB constraint of the form $\sum_{j \in G} y_j = 1$ with $y_j \in \{0, 1\}$ for $j = G \subseteq B$. Let $j_1, j_2, \ldots, j_{|G|}$ be an ordering of the variables in G and specify $G_0 = \{j_i : i = 1, \ldots, \nu\}$ and $G_1 = \{j_i : i = \nu + 1, \ldots, |G|\}$, where ν can be chosen according to

$$\nu = \underset{t \in G}{\operatorname{argmin}} \left| \sum_{i=1}^{t} y_{j_i} - 0.5 \right|. \tag{5.9}$$

Later, we present another way to select ν in the context of our stochastic scheduling problem (see Section 5.3.2). A *1GUB disjunction* on the GUB set is

$$\left(\sum_{j \in G_0} y_j \leq 0 \right) \bigcup \left(\sum_{j \in G_1} y_j \leq 0 \right). \tag{5.10}$$

Following the disjunctive cut principle, to form 1GUB cuts, (5.4b) is replaced with $-\sum_{j\in G_0} y_j \geq 0$ and (5.4d) is replaced with $-\sum_{j\in G_1} y_j \geq 0$. Appropriately dimensioned multipliers are again denoted by λ_{01}, λ_{02}, λ_{11}, and λ_{12}. Let

$$I^i_{uj} = \begin{cases} 1 & \text{if } j \in G_u \\ 0 & \text{otherwise} \end{cases}$$

for $u = 0, 1$. Then, the 1GUB cuts can be generated using the same CGLP given in (5.5) and the L-shaped solution procedure with the following modifications:

1. Replace constraints (5.5b) and (5.5c) with

$$\pi_j \geq \lambda_{01} W^i_j - I^i_{0j}\lambda_{02}, \quad \forall j,$$
$$\text{and} \quad \pi_j \geq \lambda_{11} W^i_j - I^i_{1j}\lambda_{12}, \quad \forall j,$$

   respectively.

2. Remove λ_{12} from the CGLP subproblem constraints (5.5h), from the $\psi(\cdot)$ function, and from the feasibility cut given in (5.6).

3. Conditions for the optimality cuts remain the same but the gradient subvectors $\nabla^\omega_{02}(\lambda^t)$ and $\nabla^\omega_{12}(\lambda^t)$ are always 0.

In many problems, including our stochastic scheduling problem with uncertain number of jobs, there can be more than one GUB constraint. These GUB constraints can contain disjoint sets of indices. Below, we form a disjunction for this class of problems with disjoint GUB sets, which we refer to as *2GUB* disjunctions. Variations are possible for overlapping GUB sets, however, we focus on the disjoint case. Determine sets G_0 and G_1 as above and determine sets Q_0 and Q_1 in a similar

fashion for the second GUB set, Q. The 2GUB disjunction is given as

$$\left(\sum_{j\in G_0} y_j + \sum_{j\in Q_0} y_j \le 0\right) \bigcup \left(\sum_{j\in G_0} y_j + \sum_{j\in Q_1} y_j \le 0\right) \bigcup \left(\sum_{j\in G_1} y_j + \sum_{j\in Q_0} y_j \le 0\right) \bigcup \left(\sum_{j\in G_1} y_j + \sum_{j\in Q_1} y_j \le 0\right).$$

2GUB cuts can be generated with similar modifications as above, although four disjunctions are now required instead of two. As a result, the CGLP grows twice the size of CGLP for 1GUB. Of course, the restricted CGLP may be used to generate 2GUB cuts as we do in our computational experiments.

The 2GUB disjunctions are stronger than 1GUB disjunctions, and we formally state this below.

Proposition 5.2.2. *Let P_{LP} be the feasible region of (5.3) for fixed $x \in X$ and $\omega \in \Omega$. Let G_0 and G_1 form a partition of G such that $G_0 \neq \emptyset$, $G_1 \neq \emptyset$, $G_0 \cap G_1 = \emptyset$, and $G_0 \cup G_1 = G$, and similarly let Q_0 and Q_1 form a partition of Q such that $Q_0 \neq \emptyset$, $Q_1 \neq \emptyset$, $Q_0 \cap Q_1 = \emptyset$, and $Q_0 \cup Q_1 = Q$. Assume $G \cap Q = \emptyset$. Let*

$$P_{1GUB} = conv\left(P_{LP} \bigcap \left\{y : \left(\sum_{j\in G_0} y_j \le 0\right) \bigcup \left(\sum_{j\in G_1} y_j \le 0\right)\right\}\right)$$

$$P_{2GUB} = conv\left(P_{LP} \bigcap \left\{y : \left(\sum_{j\in G_0} y_j + \sum_{j\in Q_0} y_j \le 0\right) \bigcup \left(\sum_{j\in G_0} y_j + \sum_{j\in Q_1} y_j \le 0\right) \bigcup \left(\sum_{j\in G_1} y_j + \sum_{j\in Q_0} y_j \le 0\right) \bigcup \left(\sum_{j\in G_1} y_j + \sum_{j\in Q_1} y_j \le 0\right)\right\}\right),$$

where conv denotes the convex hull. Then $P_{2GUB} \subset P_{1GUB} \subset P_{LP}$ for any partition G_0, G_1 of G and Q_0, Q_1 of Q.

Proof. $P_{1GUB} \subset P_{LP}$ is immediate. Suppose $y' \in P_{2GUB}$. Then the disjunction $\left(\sum_{j\in G_0} y_j \le 0\right) \cup \left(\sum_{j\in G_1} y_j \le 0\right)$ is satisfied and hence $y' \in P_{1GUB}$. Now suppose $y' \in P_{1GUB}$ with at least one fractional component in Q_0 and at least one fractional component in Q_1. Then $y' \notin P_{2GUB}$. Thus $P_{2GUB} \subset P_{1GUB}$. □

Proposition 5.2.2 is stated for any partition G_0, G_1 of G and Q_0, Q_1 of Q. When implementing cuts using disjunctions based on a GUB structure, the partitions G_0, G_1 and Q_0, Q_1 are chosen to cut off the current fractional solution. One such way to choose partitions is (5.9). The implication of Proposition 5.2.2 is that the strength of the set of cuts generated during D^2 should follow the same hierarchy since each extreme point solution to the CGLP corresponds to coefficients defining facets of the subproblem relaxations.

As noted before, we implement both 1GUB and 2GUB disjunctive cuts within D^2 and also use the restricted CGLP to speed up the computations. We end this section with a remark on the restricted CGLP for 1GUB and 2GUB cuts.

Remark 5.2.3. *The set of variable indices, $\mathcal{R}$, used in the restricted CGLP must include all indices used to form the 1GUB or 2GUB disjunction to guarantee a valid cut. That is, all variable indices in set G (and Q, if for 2GUB) must also be in $\mathcal{R}$ even if some variables in G or Q are zero for all scenarios.*

5.2.3 Convergence

It is important to note that without proper management of cuts it is possible that sequential cutting plane methods such as D^2 may not converge. Balas (1979) shows that closure of the convex hull of *facial disjunctive programs* (FDPs) can be generated via sequential convexification. Convergence of cutting plane algorithms for FDPs have been analyzed by Jeroslow (1980) and Blair (1980) and convergence for more general classes of problems by Sen and Sherali (1985). The implication of these results and how to achieve convergence for a lift-and-project cutting plane algorithm are discussed in (Balas et al., 1993) and for D^2 with 0-1 disjunctive cuts are discussed in (Sen and Higle, 2005). Our discussion here is similar and offers extensions for the restricted CGLP, 1GUB, and 2GUB disjunctions. We start by reviewing FDPs and convergence of D^2 with 0-1 disjunctions.

For all $(x, \omega) \in (X, \Omega)$, we denote the set of mixed-integer feasible solutions for

the second-stage MIP as

$$Y(x,\omega) = \{y : Wy \geq r(\omega) - T(\omega)x, y \geq 0, y_j \in \{0,1\}, j \in B\}, \tag{5.11}$$

and let $Y_{LP}(x,\omega)$ denote the LP relaxation of (5.11). We assume the LP relaxation contains the constraints $y_j \leq 1$, $j \in B$. We can write $Y(x,\omega)$ in *conjunctive normal form* as

$$Y(x,\omega) = \{y : Wy \geq r(\omega) - T(\omega)x, y \geq 0\} \bigcap_{s \in S} \left(\bigcup_{h \in H_s} \{y : d_h y \geq d_{h0}\} \right), \tag{5.12}$$

where $d_h y \geq d_{h0}$ is an inequality defining a disjunction in H_s, H_s is the set of disjunctions on logical condition s, and S is the set of logical conditions. For example, the set $\{y : y_1 + y_2 = 1, y_j \in \{0,1\}, j = 1,2\}$ can be written in conjunctive normal form as

$$\{y_1 + y_2 = 1, 0 \leq y \leq 1\} \bigcap (y_1 \leq 0 \cup y_1 \geq 1) \bigcap (y_2 \leq 0 \cup y_2 \geq 1),$$

where the logical conditions stipulate y_j, $j = 1,2$ must be binary.

$Y(x,\omega)$ is *facial* if every inequality $d_h y \geq d_{h0}$ that appears in a disjunction of (5.12) defines a face of $Y_{LP}(x,\omega)$ for all $h \in H_s$, $s \in S$. For example, 0-1 MIPs are FDPs since either disjunction $(y_j \leq 0)$ or $(y_j \geq 0)$ for all $j \in B$ defines a face of the 0-1 MIP LP relaxation. Balas (1979) has shown that when $Y(x,\omega)$ is facial and $Y_0(x,\omega) = Y_{LP}(x,\omega)$ is bounded, the recursion

$$Y_s(x,\omega) = \text{conv} \left\{ \bigcup_{h \in H_s} \left(Y_{s-1}(x,\omega) \bigcap \{y : d_h y \geq d_{h0}\} \right) \right\}, \quad s = 1, \ldots, |S|, \tag{5.13}$$

generates $\text{conv}Y_{|S|}(x,\omega)$ in finitely many iterations. That is, it is possible to generate the convex hull via sequential convexification.

The D^2 algorithm with 0-1 disjunctions outlined in Section 5.1 is guaranteed to generate the convex hull of the subproblems, if necessary, in a finite number of

iterations. This is possible because (1) at each iteration the generated cut will cut off the current fractional solution for some $\omega \in \Omega$, and (2) there are finitely many cuts to be generated. Recall that $j(i) \in B$ denotes the index on which the 0-1 disjunction $(y_{j(i)}(\omega) \leq 0) \cup (y_{j(i)}(\omega) \geq 1)$ is formed at iteration i of D^2. (1) is guaranteed by using a conditional expectation in (5.5a) over the set of $\omega \in \Omega$ such that $y_{j(i)}(\omega)$ is fractional and by arbitrarily dropping one scenario from the conditional expectation and reoptimizing CGLP until a nonnegative objective is found (Sen and Higle, 2005). (2) is guaranteed by proper management of rows appended to the W matrix when forming the CGLP. To ensure convergence, the CGLP is formed by appending to W disjunctive cuts generated from variable indices $j < j(i)$ at iteration i. However, unlike (5.13) that imposes the disjunctions one by one after fully generating all valid inequalities associated with the current disjunction, this scheme is computationally more efficient and has the same implication – that a finite number of disjunctive cuts will be required to generate $\text{conv}\{Y_{|S|}(x,\omega)\}$. The convex hull of $Y_1(x,\omega)$ can be generated in finitely many iterations because no cuts are appended to W when forming the CGLP due to the $j < j(i)$ index rule, i.e., the CGLP for $j(i) = 1$ has a fixed number of extreme points corresponding to facet defining cuts for $Y_1(x,\omega)$. Once $\text{conv}\{Y_1(x,\omega)\}$ has been generated, no new cuts will be appended to W when forming the CGLP for $j(i) = 2$, and thus, $\text{conv}\{Y_2(x,\omega)\}$ can be generated in finitely many iterations. By induction, $\text{conv}\{Y_{|S|}(x,\omega)\} = Y(x,\omega)$ can be generated in finitely many iterations. Since $|X \cap \mathbb{B}| < \infty$ and $|\Omega| < \infty$, $Y(x,\omega)$ can be generated for all $(x,\omega) \in (X \cap \mathbb{B}, \Omega)$.

A disjunctive program is *facial* if and only if for every $h \in H_s$, $s \in S$, $Y_{LP}(x,\omega) \subseteq \{y : d_h y \leq d_{h0}\}$ (Balas, 1979). 1GUB disjunctions satisfy this condition since no points in $Y_{LP}(x,\omega)$ are cut off by $\sum_{j \in G_0} y_j \geq 0$ or $\sum_{j \in G_1} y_j \geq 0$ for any G_0 or G_1. For 0-1 SMIPs with all second-stage binary variables in disjoint GUB constraints, the second-stage subproblem is a FDP. A similar argument holds for 2GUB disjunctions.

Convergence of the 1GUB and 2GUB convexification similarly relies on tracking the index of the generated cuts and abiding by the $j < j(i)$ index rule when forming the CGLP. For 0-1 disjunctions, this is easy since the cut index can be the index

of the disjunction variable $y_{j(i)}(\omega)$ on which the cut was formed. The 1GUB and 2GUB disjunctions use multiple variables so the same bookkeeping cannot be used. Let Γ denote the set of disjoint GUB sets $G \in \Gamma$ with $B = \cup_{G\in\Gamma} G$. We first fix an ordering $j_1, j_2, \ldots, j_{|G|}$ of the variables in each GUB set $G \in \Gamma$ and use the same ordering throughout the algorithm. Of course, we can allow for different orderings as the algorithm progresses. However, there are

$$\sum_{i=1}^{\lfloor \frac{|G|}{2} \rfloor} \binom{|G|}{i}$$

unique ways to partitions any single GUB set into two disjoint sets, and the bookkeeping can quickly get out of hand. Therefore, we use a fixed ordering. Once the ordering is established for each GUB set, we may proceed with the cut generation bookkeeping. Let $G \in \Gamma$ be the set on which we wish to form a 1GUB disjunction and choose ν for instance as in (5.9) to determine a partition G_0, G_1. Create a one-to-one mapping $m_1(G, \nu)$ that assigns a unique index to a combination (G, ν), and let $m_1(G, \nu)$ be the index used to track the specific disjunction. Form the CGLP appended only with cuts generated from indices less than $m_1(G, \nu)$.

Likewise for the 2GUB disjunctions, establish an ordering for elements in each GUB set in Γ. Let G and Q be the GUB sets in Γ on which we form the 2GUB disjunction. Choose indices ν_G and ν_Q to create partitions G_0, G_1 and Q_0, Q_1, respectively. Create a one-to-one mapping $m_2(G, \nu_G, Q, \nu_Q)$ that assigns a unique index to a combination (G, ν_G, Q, ν_Q) and let $m_2(G, \nu_G, Q, \nu_Q)$ be the index of the 2GUB cut. Form the CGLP appended only with cuts generated from indices less than $m_2(G, \nu_G, Q, \nu_Q)$.

Proposition 5.2.4. *Consider (5.1) and suppose $X = \{x \in \mathbb{R}^{d_x} : Ax \geq b, x \geq 0\}$ includes the constraints $x \leq 1$. Suppose $|\Omega|$ is finite and $f(x, \omega) < \infty$ for all $(x, \omega) \in (X, \Omega)$. Further, suppose $B = \cup_{G\in\Gamma} G$ and $G \cap Q = \emptyset$ for all G, Q in Γ. Define set $\mathcal{R}$ according to (5.7), and follow the index rule for appending cuts to W to formulate the CGLP as described above for 0-1, 1GUB, and 2GUB disjunctions.*

Solve the CGLP and if a nonnegative objective is found, arbitrarily drop one scenario from the conditional expectation and reoptimize, repeating until a negative objective is found. Suppose the CGLP finds an extreme point solution. Then the following D^2 variants will converge:

1. *D^2 with 0-1 cuts ($D^2$01)*
2. *D^2 with 0-1 cuts and restricted CGLP ($D^2$01-R)*
3. *D^2 with 1GUB cuts ($D^2$1GUB)*
4. *D^2 with 1GUB cuts and restricted CGLP ($D^2$1GUB-R)*
5. *D^2 with 2GUB cuts ($D^2$2GUB)*
6. *D^2 with 2GUB cuts and restricted CGLP ($D^2$2GUB-R)*

Proof. For proof of 1, see (Sen and Higle, 2005). We provide the proof of 2 in two steps: (i) by showing the current fractional solution will be cut off when we use the restricted CGLP, and (ii) that there are finitely many cuts to be generated. This combined with the fact that the first-stage master problem approximations converge in finitely many iterations leads to finite convergence of the D^2 variant. (i) The CGLP$^{\mathcal{R}}$ is formed from $W^{\mathcal{R}}$, where columns $j \notin \mathcal{R}$ are removed from W. Every column removed from W results in one fewer variable and two fewer structural constraints in the CGLP$^{\mathcal{R}}$, but the removed variables only appear in the constraints that have been removed. Thus, the feasible region of CGLP$^{\mathcal{R}}$ is a relaxation of the feasible region of CGLP. Further, the columns removed from W correspond to variables in the CGLP objective with coefficients equal to zero. Therefore, the objective of CGLP$^{\mathcal{R}}$ will be less than or equal to the objective of CGLP, and the current fractional solution will be cut off. The lifting procedure (5.8) does not affect the optimal objective to CGLP$^{\mathcal{R}}$ since the lifted variables have objective coefficients equal to zero. (ii) See Balas et al. (1993) for a discussion of the finiteness of cuts to be generated when using the restricted CGLP. Since $(X \cap \mathbb{B}, \Omega)$ is itself finite, the total number of cuts to be generated for $Y(x, \omega)$ is finite.

For proof of 3, (i) note that any fractional solution can be cut off from cuts based on 1GUB disjunctions. This is true for two reasons. First, there exists a violated 1GUB disjunction for any fractional solution since some GUB set G will contain at least two fractional variables, and we can choose G_0 and G_1 such that both will contain at least one fractional variable. Second, the CGLP will generate a violated cut since the CGLP objective is a conditional expectation over scenarios that violate the 1GUB disjunction. If a nonnegative objective is found, we arbitrarily drop one scenario from the conditional expectation and reoptimize, repeating until a negative objective is found. (ii) Given an x and ω and a (G, ν) combination with the smallest $m_1(G, \nu)$ index, the number of cuts generated by the algorithm is finite as every cut generated corresponds to an extreme point of P_{1GUB}, of which there are finitely many. Since the second-stage with 1GUB disjunctions is facial, the sequential convexification recursion (5.13) holds, and by following the index rule on $m_1(G, \nu)$ one can generate all the cutting planes in finitely many iterations. Since $(X \cap \mathbb{B}, \Omega)$ is itself finite, the total number of cuts to be generated for $Y(x, \omega)$ is finite. Proof of 4, 5, and 6 follow from similar arguments as above.

□

In practice when $x^{i+1} = x^i$, all subproblems are solved to integer optimality and a Laporte and Louveaux (1993) type optimality cut is added to the master problem. Even if theoretical convergence of the disjunctive cuts cannot be shown, the addition of Laporte-Louveaux type cuts when x stabilizes ensures that the D^2 algorithm will converge since the Laporte-Louveaux algorithm converges.

5.3 Computational Experiments

In this section we compare the effectiveness of the various cuts and the restricted CGLP on a set of SMRCSP-JB test problems. We first describe the test problem instance generation and then demonstrate the effectiveness of the various cuts on the problem.

5.3.1 Test Problem Generation

The test problems are labeled as $A.B.C.D$ where $A \in \{5, 10\}$ is the number of jobs that must be scheduled, $B \in \{10, 20\}$ is the number of jobs that are bid on, $C \in \{30, 50\}$ is the number of time periods T, and $D \in \{10, 100, 1000\}$ is the number of scenarios generated. All jobs have $K = 3$ resource classes. Processing times were drawn from a discrete uniform(1,C) distribution. For each bid, the probability of winning a job bid is 0.75 and is independent from other bids. We set $T_0 = \lceil 0.25T \rceil$. Resource demands r_{jk} were generated from a discrete uniform(1,5). Temporary expansion costs b_{tk} are set to be the same for all t and are generated from a discrete uniform(1,10). Resource capacities R_{tk} are set to be the same for all t and are denoted as R_k. The R_k are generated according to

$$R_k = \frac{\bar{p}^C \bar{r}_k^C |J| + 0.75 \bar{p}^B \bar{r}_k^B |J_B|}{(T + T_0)\rho},$$

where $\bar{p}^C = \frac{1}{|J|} \sum_{j \in J} p_j$ is a measure of average processing times for the known jobs, $\bar{r}_k^C = \frac{1}{|J|} \sum_{j \in J} r_{jk}$ is a measure of the average amount of resource k consumed by the known jobs, $\bar{p}^B = \frac{1}{|J_B|} \sum_{j \in J_B} p_j$ is a measure of average processing times for jobs with bids, $\bar{r}_k^B = \frac{1}{|J_B|} \sum_{j \in J_B} r_{jk}$ is a measure of the average amount of resource k consumed by the jobs with bids, and ρ is a factor controlling the ratio of resources consumed to resources available. The numerator is a rough estimate of the total amount of resource k we would expect to be consumed during the planning horizon, and $R_k \cdot (T + T_0)$ is the total resource available during the planning horizon. For $\rho < 1$ there tends to be enough resources for jobs to process in the horizon without exceeding R_k, and for $\rho > 1$ there tends to be more resources consumed than available, resulting in temporary resource expansion. We generated ρ from a uniform(0.7,1.3) distribution. The cost coefficients, c_{jt}, were set to completion time ($c_{jt} = t + p_j - 1$). Five instances for each problem were generated and solved.

Table 5.1 reports the dimensions of the deterministic equivalent problem (DEP) along with the dimensions of the first-stage problems and the subproblems. The column "Constr", "Bin", and "Cvar" gives the number of constraints, binary variables,

	DEP			1^{st} stage			2^{nd} stage		
Problem	Constr	Bin	Cvar	Constr	Bin	Cvar	Constr	Bin	Cvar
5.10.30.10	1029	1925	924	29	155	24	100	177	90
5.10.30.1000	100,029	177,155	90,024	29	155	24	100	177	90
5.10.50.10	1644	2985	1539	44	255	39	160	273	150
5.10.50.1000	160,044	273,255	150,039	44	255	39	160	273	150
10.10.30.10	1034	1952	924	34	132	24	100	182	90
10.10.30.1000	100,034	182,132	90,024	34	132	24	100	182	90
10.10.50.10	1649	2770	1539	49	210	39	160	256	150
10.10.50.1000	160,049	256,210	150,039	49	210	39	160	256	150
10.20.30.10	1134	2868	924	34	188	24	110	268	90
10.20.30.100	11,034	26,988	9024	34	188	24	110	268	90
10.20.50.10	1749	5124	1539	49	204	39	170	492	150
10.20.50.100	17,049	49,404	15,039	49	204	39	170	492	150

Table 5.1: Problem dimensions.

and continuous variables, respectively.

5.3.2 Implementation Details

The D^2 algorithm with each cut generation scheme was applied to SMRCSP-JB presented in Chapter 3. The algorithm was initialized by solving the LP relaxation of the stochastic MIP. All components of the fractional solution equal to zero or one were fixed at their values, and the remaining reduced-size master problem was solved to integer optimality to find a starting integer-feasible solution for the D^2 algorithm. At each D^2 iteration, the master problem was solved to integer optimality. When the first stage solution failed to change from one iteration to the next, all subproblems were solved to integer optimality (without the added disjunctive cuts) and a Laporte-Louveaux type cut was added to the master problem. The algorithm terminated when the optimality gap fell below 0.01% or after 7200 seconds. We ran our experiments using CPLEX 11.1 on a 2.4 GHz processor with 4 GB of memory running Linux.

The disjunction variable for 0-1 cuts was chosen by finding the variable that was fractional in the most number of scenarios. The 1GUB disjunctions were determined

by finding the GUB set G that most frequently contained a fractional variable. For each scenario ω with a fractional variable in G, the weighted starting time of the job $j \in J$ was calculated as $\bar{s}(\omega) = \sum_{t=T_0+1}^{T+T_0-p_j+1} t y_{jt}(\omega)$. The ν value used to partition the GUB set G was chosen as

$$\nu = \lfloor \bar{s}(\omega^*) \rfloor \tag{5.14}$$

where

$$\omega^* = \underset{\omega \in \mathcal{F}}{\operatorname{argmin}} \left| \bar{s}(\omega) - \frac{1}{|\mathcal{F}|} \sum_{\omega' \in \mathcal{F}} \bar{s}(\omega') \right|$$

where $\mathcal{F} \subseteq \Omega$ is the set of scenarios that have fractional elements in GUB set G. Selecting ν in this manner attempts to maximize the number of scenarios in which the 1GUB disjunction is violated while guaranteeing at least one scenario will violate the disjunction, thus preserving the convergence of the algorithm.

The 2GUB disjunctions were chosen by finding the pair of GUB sets G and Q that were fractional together in the most scenarios. Parameters ν_G and ν_Q were chosen using the weighted starting time approach just like ν given in (5.14) for the 1GUB cuts. The initial implementation of the 2GUB CGLP took much longer to solve than either the 0-1 or 1GUB CGLP. Much of the time was spent trying to lower the CGLP optimality gap after the upper bound was already negative. We attempted to improve the 2GUB cut generation time by terminating CGLP at the first iteration after 5 iterations such that the upper bound was negative. This did improve 2GUB performance overall, but the cut generation problem occasionally took a long time for the upper bound to become negative. Due to the longer cut generation times, we only implemented the restricted CGLP.

5.3.3 Computational Results

Tables 5.2 and 5.3 display results for D^2 with 0-1 cuts ($D^2$01), D^2 with 0-1 cuts and restricted CGLP ($D^2$01-R), D^2 with 0-1 cuts and restricted CGLP with strengthening ($D^2$01-RS), D^2 with 1GUB cuts ($D^2$1GUB), D^2 with 1GUB cuts and restricted CGLP ($D^2$1GUB-R), and D^2 with 2GUB cuts and restricted CGLP ($D^2$2GUB-R).

The strengthening procedure was not effective for $D^2$1GUB-R and results are omitted for brevity. There are 12 problems labeled A.B.C.D as explained in Section 5.3.1 and we generated 5 instances for each problem.

Table 5.2 displays the minimum, average, and maximum computational time (in seconds) of the five instances for each problem along with the average number of iterations. Problem instances exceeding the time limit are denoted by "> 7200" in the "max" column followed by the number of instances out of 5 that *were* solved within the time limit. For instance, $D^2$01 for 5.10.50.1000 had 2 instances that were solved within the time limit.

Table 5.3 shows the average improvement in computation time of the different D^2 implementations compared to $D^2$01. The improvement for each instance was calculated as $(D^2 01_{time} - other_{time})/D^2 01_{time}$, where $other_{time}$ is the computational time of the D^2 variant. The improvement reported is the average of the five instances for each problem. Whenever $D^2$01 could not solve all 5 instances, we instead report the number of instances solved by the other D^2 implementation out of the number of instances that were not solved by $D^2$01.

The tables indicate that the restricted CGLP, 1GUB cuts, and strengthening procedure for 0-1 cuts improve the performance of the D^2 algorithm. The restricted CGLP demonstrated significant improvements in computation times. $D^2$01-R solved eight instances left unsolved by $D^2$01, including all five instances of problem 10.20.50.100, and $D^2$1GUB-R solved one more instance than $D^2$01-R within the time limit. The computation time for $D^2$01-R was about 30% less than $D^2$01, and strengthening improved $D^2$01-R by about 3% on average. The computation time of $D^2$1GUB-R was about 45% less on average than $D^2$01. Although the restricted CGLP is a relaxation of the full-sized CGLP, interestingly, the number of iterations for both $D^2$01-R and $D^2$1GUB-R are sometimes less than that of their full-sized CGLP counterparts. Balas et al. (1996) experienced a similar phenomenon when comparing performance of full-sized and restricted CGLPs in a disjunctive cutting plane algorithm. They commented that cuts from full-size CGLP may not improve

	$D^2$01				$D^2$01-R				$D^2$01-RS			
Problem	min	avg	max	iters	min	avg	max	iters	min	avg	max	iters
5.10.30.10	3	48.3	173	39.8	1	12	24	12	1	10.6	21	9.8
5.10.30.1000	717	884.6	1339	13.8	711	849	1270	15	655	797	1199	10
5.10.50.10	32	126.2	263	57	12	65	169	36	11	59	141	29.4
5.10.50.1000	6781		>7200 (2)		6293		>7200 (3)		6231		>7200 (3)	
10.10.30.10	15	30.6	58	35.5	9	19	32	33	8	18.4	31	30.2
10.10.30.1000	562	642.6	768	30.6	493	584	601	33	531	593.8	658	33.4
10.10.50.10	20	102	204	50.4	17	42	120	67	6	46.2	111	54.8
10.10.50.1000	2314	3335.6	4055	66.4	2189	2651	3006	80	2211	2505.2	2848	75.6
10.20.30.10	81	231.4	584	86.2	64	180	390	81	66	171	445	67
10.20.30.100	2861	4595.6	6157	150.2	2615	3856	5084	135	2631	3844.2	5286	126
10.20.50.10	1003		>7200 (2)		50		>7200 (4)		150	1687.6	6699	259.4
10.20.50.100	>7200		>7200 (0)		3731	5426	6921	350	3404		>7200 (4)	

	$D^2$1GUB				$D^2$1GUB-R				$D^2$2GUB-R			
Problem	min	avg	max	iters	min	avg	max	iters	min	avg	max	iters
5.10.30.10	9	26.6	78	33.8	1	12	21	10	1	14.8	25	9.8
5.10.30.1000	810	951.6	1396	20.2	635	760	1130	8	662	1370.6	986	7.8
5.10.50.10	22	108.2	226	46.6	13	54	172	45	14	73.4	198	33.6
5.10.50.1000	6514		>7200 (3)		6111		>7200 (3)		>7200		>7200 (0)	
10.10.30.10	12	21.6	36	27.8	8	15	26	27	9	20.2	40	33.6
10.10.30.1000	446	493.2	520	29.4	419	446	474	29	1294	2154	2694	31.2
10.10.50.10	16	75.4	198	56.6	8	22	41	35	12	76.4	174	56.8
10.10.50.1000	1646	1997.6	2253	63	1521	1798	1992	59	5175		>7200 (0)	
10.20.30.10	81	190.2	447	65	57	146	403	58	107	213.2	368	80.8
10.20.30.100	2776	4092.8	5491	118	2632	3854	5172	118	6585		>7200 (2)	
10.20.50.10	760	3232.8	5876	344.2	253	823	1716	252	438		>7200 (3)	
10.20.50.100	4700		>7200 (3)		2292	3145	4093	269	>7200		>7200 (0)	

Table 5.2: Computation times and average number of iterations for different implementations. No average is reported for problems that had instances not solved within the time limit. In this case, the number in parenthesis indicates the number of instances out of 5 that *were* solved within the time limit.

convexification as much as cuts from restricted CGLPs because the full-size CGLP cuts tend to be more parallel to the objective function. As the convexification procedure continues, having cuts that improve convexification in diverse directions is important for the generation of new cuts. Our results agree.

The 1GUB cuts by themselves were also effective in reducing computation time. $D^2$1GUB solved all problem instances solved by $D^2$01 plus seven more instances not solved by $D^2$01. The average iteration count for $D^2$1GUB is less than that of $D^2$01 in 7 of 9 problems solved by both implementations. The improvement in computation times for $D^2$1GUB is around 20% on average, omitting problems not solved by $D^2$01. $D^2$1GUB-R improves substantially over this and achieves on average 45% improvement in computation times.

Two algorithm modifications were *not* effective: (i) $D^2$1GUB-R with strengthening and (ii) $D^2$2GUB-R. Strengthening can only be applied when $\lambda_{02} + \lambda_{12} \neq 0$, and this condition was rarely achieved during 1GUB cut generation. Hence, strengthening was rarely possible and computational results were basically the same as $D^2$1GUB-R. D^2 with 2GUBcuts was not computationally competitive with the other algorithms due to the larger size of the CGLP even when the restricted CGLP was used. Since $D^2$2GUB-R had four terms in the 2GUB disjunctions, the CGLP was about twice as large as CGLP for 1GUB. Even early termination of CGLP when the upper bound became negative did not reduce computation time to the level of the other algorithms since the CGLP often required many iterations to achieve a negative upper bound. The time required to generate cuts outweighed any advantage of generating stronger cuts.

5.3.4 Further Analysis

The results in Table 5.3 show that computational improvement appears to decrease as $|\Omega|$, the total number of realizations of the random vector $\tilde{\omega}$, increases. To gain a better understanding of this, we solved the problem 10.10.30 with $|\Omega| = 10, 100, 200, \ldots, 1000$ total number of realizations with $D^2$01, $D^2$01-RS, and $D^2$1GUB-R. Figure 5.1 shows the improvement in total computation time and cut

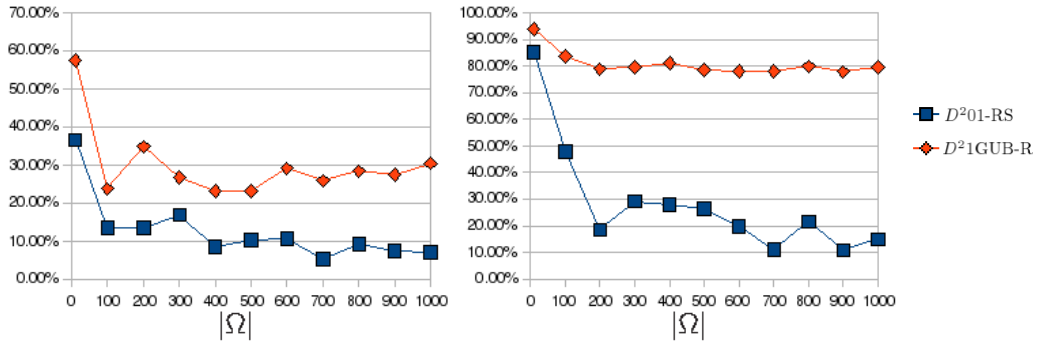
Problem	$D^2$01-R	$D^2$01-RS	$D^2$1GUB	$D^2$1GUB-R	$D^2$2GUB-R
5.10.30.10	74.27%	78.01%	44.81%	75.93%	47.32%
5.10.30.1000	4.05%	9.90%	-7.57%	14.09%	-48.82%
5.10.50.10	48.18%	53.25%	14.26%	56.89%	36.37%
5.10.50.1000	(1/3)	(1/3)	(1/3)	(1/3)	*
10.10.30.10	39.22%	39.87%	29.41%	57.52%	31.50%
10.10.30.1000	9.12%	7.57%	23.25%	30.66%	-234.93%
10.10.50.10	58.82%	54.71%	26.08%	78.82%	34.51%
10.10.50.1000	20.51%	24.90%	40.11%	46.10%	-107.38%
10.20.30.10	22.10%	26.10%	17.80%	36.91%	-16.06%
10.20.30.100	16.09%	16.35%	10.94%	16.13%	**
10.20.50.10	(2/3)	(3/3)	(3/3)	(3/3)	(1/3)
10.20.50.100	(5/5)	(4/5)	(3/5)	(5/5)	(1/5)

Table 5.3: Percentage improvement in computation times over $D^2$01. The two numbers in parenthesis indicate the number of instances solved by the modified D^2 algorithm out of the number of instances not solved by $D^2$01. * indicates no instances were solved. ** indicates two instances solved.

generation time per D^2 iteration for $D^2$01-RS and $D^2$1GUB-R over $D^2$01. The first point, $|\Omega| = 10$, and the last point, $|\Omega| = 1000$, are reported in Table 5.3. From $|\Omega| = 10$ to $|\Omega| = 100$ for both D^2 variants, there is a drop in improvement in total time and improvement in cut generation time per D^2 iteration. The improvement of $D^2$1GUB-R appears to stabilize for $|\Omega| \geq 100$ while the improvement of $D^2$01-RS experiences a slightly negative trend.

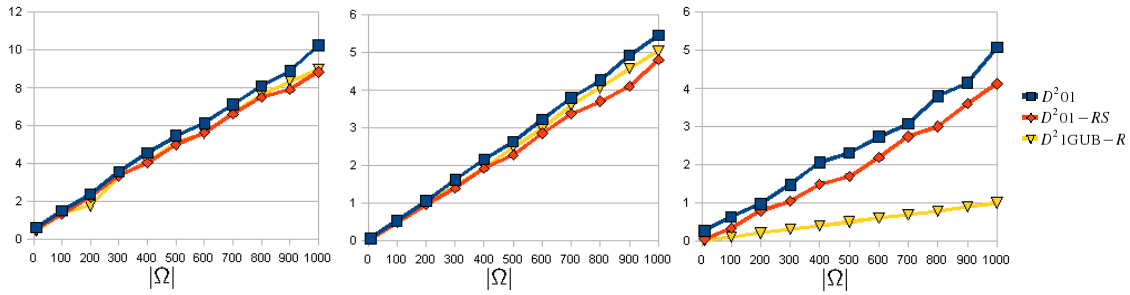
Figure 5.2 shows the absolute amount of time spent per D^2 iteration on cut generation, right-hand-side convexification, and all other computations. Time spent per D^2 iteration on the RHSLP and other computations were similar for the three algorithms. However, time spent per D^2 iteration on CGLP was notably different. $D^2$01 spent the most amount of time per D^2 iteration on cut generation, $D^2$01-RS spent a little less time per D^2 iteration, and $D^2$1GUB-R spent by far the least amount of time per D^2 iteration. The graph suggests reduction in cut generation time per D^2 iteration is the major factor in reduction of total computational time.

The difference in cut generation time per D^2 iteration appears to be due to the number of L-shaped iterations required to generate the cut. We noticed that $D^2$01-RS required around 5 L-shaped iterations to solve the CGLP for $|\Omega| = 10$. As $|\Omega|$



(a) Total computational time (b) CGLP time per D^2 iteration

Figure 5.1: Percentage improvement over $D^2$01 in (a) total computation time and (b) cut generation time per D^2 iteration as $|\Omega|$ varies.



(a) Other (b) RHSLP (c) CGLP

Figure 5.2: Absolute computational time per D^2 iteration for cut generation, right-hand-side convexification, and all other computations as $|\Omega|$ varies.

increased, the number of CGLP L-shaped iterations rose to around 15-20 with an occasional CGLP requiring more. CGLP L-shaped iterations for $D^2$1GUB-R usually remained around 5 regardless of $|\Omega|$. This explains why in Figure 5.2 cut generation time per D^2 iteration for $D^2$01-RS is around 4 times more than cut generation per D^2 iteration for $D^2$1GUB-R CGLP.

The fewer number of CGLP L-shaped iterations required for 1GUB disjunctions may be due to two factors. First, λ_{02} and λ_{12} appear in more constraints in the CGLP for 1GUB disjunctions than in the CGLP for 0-1 disjunctions since 1GUB disjunctions affect all variables in GUB set G whereas 0-1 disjunctions affect only one variable. The extra constraints may provide more information to the CGLP master problem on good values of λ_{02} and λ_{12} so that less feedback is required from the CGLP second-stage. Second, λ_{12} is not present in the CGLP second-stage (5.5h) for 1GUB disjunctions, reducing the likelihood of the second-stage being infeasible for a set of λ's passed from the CGLP master problem. To test this explanation, we implemented an alternative 1GUB disjunction

$$\left(-\sum_{j\in G_0} y_j \geq 0\right) \bigcup \left(\sum_{j\in G_0} y_j \geq 1\right), \tag{5.15}$$

which we denote as the 1GUB01 disjunction. This is equivalent to the 1GUB disjunction in (5.10); however, the 1GUB01 disjunction in (5.15) results in λ_{12} appearing in the CGLP second-stage, whereas λ_{12} is absent in the CGLP second-stage for 1GUB disjunctions. The 1GUB01 disjunction allowed us to examine the effects of λ_{02} and λ_{12} appearing in extra CGLP master problem constraints and of λ_{12} appearing in the CGLP second-stage (5.5f) - (5.5i). Figure 5.3 displays the cut generation time per D^2 iteration reported in Figure 5.2c along with cut generation time per D^2 iteration for D^2 with 1GUB01 disjunctions and restricted CGLP ($D^2$1GUB01-R). The figure shows that $D^2$1GUB01-R takes less cut generation time per D^2 iteration than $D^2$01 and $D^2$01-R but more time than $D^2$1GUB-R. The $D^2$1GUB01-R CGLP required 5-10 L-shaped iterations with an occasional CGLP requiring more. The $D^2$1GUB01-R results support the ideas that (1) the CGLP master problem for

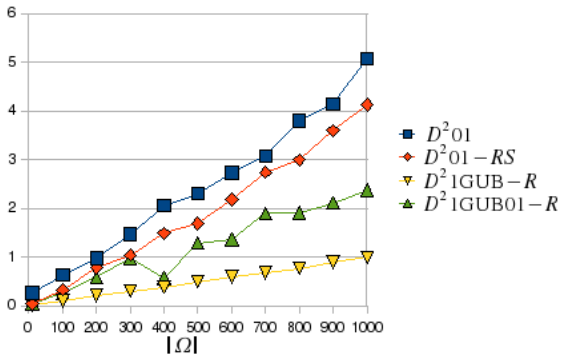


Figure 5.3: CGLP time per D^2 iteration for $D^2$1GUB01-R.

1GUB disjunctions provides more information on good values of λ_{02} and λ_{12} than 0-1 disjunctions, and (2) the presence of λ_{12} in the CGLP second-stage affects the number of iterations required to generate a cut. We note that total computational time for $D^2$1GUB01-R was 20% larger than $D^2$1GUB-R.

5.4 Concluding Remarks

In this chapter, we solved the SMRCSP-JB using the D^2 algorithm, and we explored the use of generating cuts in a subspace of subproblem variables and the use of alternative disjunctions based on the GUB structure in the second-stage of the SMRCSP-JB. While the enhancements we developed were aimed at improving the solution of SMRCSP-JB, the enhancements are applicable to a wide variety of other problems. We found that the restricted CGLP greatly speeds up the D^2 algorithm although the improvement for 0-1 cuts decreases as the number of scenarios increases. However, the total D^2 iterations remains about the same and sometimes even less than when cuts are generated with the full-size CGLP, indicating that the cuts have similar effectiveness. We also found that cut strengthening can be beneficial for 0-1 disjunctions with minimal additional computational burden.

The alternative disjunctions exploit the special GUB structure found in the second-stage of SMRCSP-JB and in many other optimization problems. The 1GUB cuts seem to have less computational burden than the 0-1 cuts and our computa-

tional experiments indicate the 1GUB cuts are more effective for the SMRCPS-JB. Furthermore, the effectiveness of 1GUB cuts seem to be fairly stable as the total number of realizations of $\tilde{\omega}$ increases. The 2GUB cuts sometimes resulted in fewer iterations than 0-1 or GUB cuts. However, due to its size, cut generation for 2GUB becomes computationally burdensome even when the restricted CGLP is used and the cut generation problem is terminated when the first negative objective is found after 5 CGLP L-shaped iterations. D^2 with 1GUB cuts using the restricted CGLP resulted in the most improvement of about 45% on the SMRCSP-JB.

CHAPTER 6

CONCLUSIONS

6.1 Summary

We have applied optimization techniques to improve utilization of resources when scheduling jobs that share multiple common resources under two general cases of uncertainty. The first case considered uncertainty of processing times, due dates, resource availability, and resource consumption, which we denote as the stochastic MRCSP with uncertain parameters (SMRCSP-U). The second case considered uncertainty in the number of jobs to schedule. One example application of this type of uncertainty is when companies in consulting and defense contracting bid on future contracts but may or may not win the bid. We call this problem the stochastic MRCSP with job bidding (SMRCSP-JB). We model both problems as two-stage stochastic integer programs with recourse.

In our study of the models, we developed solution methodologies to generate schedules that minimize expected cost under uncertainty. The formulation is generic allowing application of proposed methods to a variety of different objectives commonly used in the scheduling literature such as expected tardiness or expected lateness. We solved the first model, SMRCSP-U, with the L-shaped method and developed several effective computational enhancements. The combination of GUB branching, trust regions, LP warm starting, and approximate solution of the master problem resulted in more than a 70% reduction in total solution time over 9 test problems. While some implementation details of the enhancements are problem specific, the general concepts we presented are applicable to a wide class of problems.

As the SMRCSP-U handles uncertainties in processing times, resource consumption, and resource availabilities, problems of practical scale can quickly become

intractable due to the prohibitively large number of scenarios. To alleviate this difficulty, we presented and proved desired properties of a sequential sampling procedure that obtains high quality solutions without having to evaluate every scenario. The sequential sampling procedure remains valid for δ-optimal solutions and is applicable to SMRCSP-U and other stochastic integer programs. Our computational results with up to 10^{24} scenarios indicate that quality solutions can be found quickly ($\leq$ 1000 seconds) with this methodology.

In our study of the SMRCSP-JB, we utilized the disjunctive decomposition (D^2) algorithm for stochastic integer programs with binary first-stage and mixed-binary second-stage. We developed enhancements geared towards generating disjunctive cuts more quickly and incorporating problem structure into the disjunctions. We found that the restricted cut generation linear program (CGLP) greatly speeds up the D^2 algorithm although this improvement for the standard 0-1 cuts decreases as the number of scenarios increases. However, the total D^2 iterations remains about the same and sometimes even less than when cuts are generated with the full-size CGLP, indicating that the cuts have similar effectiveness. We also found that cut strengthening can be beneficial for 0-1 disjunctions with minimal additional computational burden. The restricted CGLP and strengthening procedure are general and can be applied to any other problem that can be solved by D^2.

The alternative disjunctions exploited the special GUB structure found in the SMRCSP-JB and many other optimization problems. We developed two types of cuts based on this structure, which we refer to as 1GUB and 2GUB disjunctive cuts. The 1GUB cuts seemed to have less computational burden than the 0-1 cuts, and our computational experiments indicated the 1GUB cuts are more effective for the SMRCSP-JB. The 2GUB cuts sometimes resulted in fewer iterations than 0-1 or GUB cuts. However, due to its size, cut generation for 2GUB becomes computationally burdensome even when the restricted CGLP is used and the cut generation problem is terminated when the first negative objective is found after 5 CGLP L-shaped iterations. Even with these speedups, D^2 using 2GUB cuts is not competitive.

6.2 Future Research Directions

The SMRCSP-U considers uncertainty in most model parameters while assuming the number of jobs is known. The SMRCSP-JB considers an uncertain number of jobs while assuming all other model parameters are known. A natural extension is to consider a combination of the two models in which processing time, due dates, resource availability, resource consumption, and number of jobs is uncertain. This model results in random recourse and may be solved with an extension of D^2 (Ntaimo, 2009).

This dissertation addressed uncertainty by creating *quality robust* schedules that have similar costs across a variety of scenarios. An alternative approach would be to create *solution robust* schedules in which the start dates of jobs remain similar across a variety of scenarios. In some cases, solution robust schedules may be preferred. For practical purposes, resource leveling and utilization could be of interest to companies. Future work could involve addition of constraints that control the variation of resource utilization in the models and study their effect on solution characteristics.

The restricted CGLP resulted in significant computational savings in the D^2 algorithm. Future research should include applying this concept to the RHSLP convexification problem. The restricted problems do not necessarily generate the same coefficients as the full-sized problems, and convergence of D^2 may be lost if using a restricted RHSLP. However, convergence can be guaranteed by embedding the restricted RHSLP inside of D^2-BAC since convergence of D^2-BAC is guaranteed without convexification of the subproblem feasible region.

Perregaard and Balas (2001) explored generating cuts from multiple 0-1 disjunctions. Recall that the nonzeros in the CGLP are proportional to the number of disjunctions resulting in prohibitively large CGLPs for even a small number of disjunctions. Perregaard and Balas overcome this difficulty by decomposing the CGLP based on disjunctions (whereas we decompose based on scenarios) and find favorable results. The disjunctions are chosen corresponding to nodes in a partially

solved branch-and-bound tree. In the SMIP setting, D^2-BAC uses disjunctions from a partially solved branch-and-bound tree to convexify the recourse function. Future D^2-BAC research could investigate combining multiple 0-1 disjunctions for set convexification formed from the partially solved brach-and-bound tree along with recourse function convexification from the same tree.

REFERENCES

Ahmed, S. and A. Shapiro (2002). The sample average approximation method for stochastic programs with integer recourse. *Optimization Online*. Available at: http://www.optimization-online.org.

Ahmed, S. and A. Shapiro (2008). Solving chance-constrained stochastic programs via sampling and integer programming. In *Tutorials in Operations Research*, pp. 261–269. INFORMS.

Ahmed, S., M. Tawarmalani, and N. Sahinidis (2004). A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, **100**, pp. 355–377.

Ahuja, R., T. Magnanti, and J. Orlin (1993). *Network Flows*. Prentice Hall.

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

Baker, K. and D. Trietsch (2007). Safe scheduling. In *Tutorials in Operations Research*, pp. 79–101. INFORMS.

Baker, K. and D. Trietsch (2009a). *Principles of Sequencing and Scheduling*. John Wiley & Sons.

Baker, K. and D. Trietsch (2009b). *Research Notes for Chapter 1*. Principles of Sequencing and Scheduling. John Wiley & Sons. Available at: http://mba.tuck.dartmouth.edu/pss/.

Baker, K. and D. Trietsch (2009c). *Research Notes for Chapter 2*. Principles of Sequencing and Scheduling. John Wiley & Sons. Available at: http://mba.tuck.dartmouth.edu/pss/.

Balas, E. (1975). Disjunctive programming: cutting planes from logical conditions. In Mangasarian, O., R. Meyer, and S. Robinson (eds.) *Nonlinear Programming 2*. Academic Press, New York.

Balas, E. (1979). Disjunctive programming. *Annals of Discrete Mathematics*, **5**, pp. 3–31.

Balas, E. (1985a). Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic and Discrete Methods*, **6**, pp. 466–486.

Balas, E. (1985b). On the facial structure of scheduling polyhedra. *Mathematical Programming Study*, **24**, pp. 179–218.

Balas, E., S. Ceria, and G. Cornuéjols (1993). A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, **58**, pp. 295–324.

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

Balas, E. and R. Jeroslow (1980). Strengthening cuts for mixed integer programs. *European Journal of Operational Research*, **4**, pp. 224–225.

Balas, E. and M. Perregaard (2002). Lift-and-project for mixed 0-1 programming: Recent progress. *Discrete Applied Mathematics*, **123**, pp. 129–154.

Ball, M., T. Magnanti, C. Monma, and G. Nemhauser (eds.) (1995). *Network Routing*, volume 8 of *Handbook in Operations Research and Management Science*. North-Holland.

Banerjee, B. (1965). Single facility sequencing with random execution times. *Operations Research*, **13**, pp. 358–364.

Bayraksan, G. and D. Morton (2006). Assessing solution quality in stochastic programs. *Mathematical Programming*, **108**, pp. 495–514.

Bayraksan, G. and D. Morton (2009). A sequential sampling procedure for stochastic programming. Technical report, Department of Systems and Industrial Engineering, the University of Arizona. Available at: http://www.sie.arizona.edu/faculty/guzinb/publications.htm.

Bazaraa, M., H. Sherali, and C. Shetty (1993). *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, 2 edition.

Beale, E. (1955). Minimizing a convex function subject to linear inequalities. *Journal of the Royal Statistical Society, Series B*, **17**, pp. 173–184.

Benders, J. (1962). Partitioning procedures for solving mixed-variable programming problems. *Numerische Mathematik*, **4**, pp. 238–252.

Birge, J. and M. Dempster (1996). Stochastic programming approaches to stochastic scheduling. *Journal of Global Optimization*, **9**(3-4), pp. 417–451.

Birge, J. and F. Louveaux (1997). *Introduction to Stochastic Programming*. Springer, New York.

Birge, J. and R. Wets (1985). Decomposition and partitioning methods for multistage stochastic linear programs. *Operations Research*, **33**, pp. 989–1007.

Birge, J. and R. Wets (1989). Sublinear upper bounds for stochastic programs with recourse. *Mathematical Programming*, **43**, pp. 131–149.

Blair, C. (1980). Facial disjunctive programs and sequences of cutting planes. *Discrete Applied Mathematics*, **2**, pp. 173–179.

Blair, C. and R. Jeroslow (1978). A converse for disjunctive constraints. *Journal of Optimization Theory and Applications*, **25**, pp. 195–206.

Brucker, P. (2001). *Scheduling Algorithms*. Springer, New York, 3 edition.

Cai, X., C. Lee, and C. Li (1998). Minimizing total completion time in two-processor task systems with prespecified processor allocations. *Naval Research Logistics*, **45**, pp. 231–242.

Carøe, C. (1998). *Decomposition in Stochastic Integer Programming*. Ph.D. thesis, Institute of Mathematical Sciences, Dept. of Operations Research, University of Copenhagen.

Carøe, C. and R. Schultz (1999). Dual decomposition in stochastic integer programming. *Operations Research Letters*, **24**, pp. 37–45.

Carøe, C. and J. Tind (1997). A cutting-plane approach to mixed 0-1 stochastic integer programs. *European Journal of Operational Research*, **101**, pp. 306–316.

Chan, L., A. Muriel, and D. Simchi-Levi (1998). Parallel machine scheduling, linear programming, and parameter list scheduling heuristics. *Operations Research*, **46**, pp. 729–741.

Charnes, A. and W. Cooper (1959). Chance-constrained programming. *Management Science*, **6**, pp. 73–79.

Charnes, A. and W. Cooper (1963). Deterministic equivalents for optimizing and satisficing under chance constraints. *Operations Research*, **11**, pp. 18–39.

Chen, J. and C. Lee (1999). General multiprocessor task scheduling. *Naval Research Logistics*, **46**, pp. 57–74.

Chou, M., H. Liu, M. Queyranne, and D. Simchi-Levi (2006). On the asymptotic optimality of a simple on-line algorithm for the stochastic single-machine weighted complete time problem and its extensions. *Operations Research*, **54**, pp. 464–474.

Clark, C. (1961). The greatest of a finite set of random variables. *Operations Research*, **9**, pp. 145–162.

Cooper, D. (1976). Heuristics for scheduling resource constrained projects: An experimental comparison. *Management Science*, **22**, pp. 1186–1194.

Dantzig, G. (1955). Linear programming under uncertainty. *Management Science*, **1**, pp. 197–206.

Dantzig, G., D. Fulkerson, and S. Johnson (1954). Solution of a large-scale traveling salesman problem. *Operations Research*, **2**, pp. 393–410.

Dantzig, G. and P. Wolfe (1960). The decomposition principle for linear programs. *Operations Research*, **8**, pp. 101–111.

Deblaere, F., E. Demeulemeester, W. Herroelen, and S. V. de Vonder (2007). Robust resource allocation decisions in resource-constrained projects. *Decision Sciences*, **38**, pp. 5–37.

Denton, B. and D. Gupta (2003). A sequential bounding approach for optimal appointment scheduling. *IIE Transactions*, **35**, pp. 1003–1016.

Drozdowski, M. (1996). Scheduling multiprocessor tasks - An overview. *European Journal of Operational Research*, **94**, pp. 215–230.

Dyer, M. and L. Wolsey (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, **26**, pp. 255–270.

Ermoliev, Y. (1983). Stochastic quasigradient methods and their applications to systems optimization. *Stochastics*, **9**, pp. 1–36.

Gilmore, P. and R. Gomery (1961). A linear programming approach to the cutting stock problem. *Operations Research*, **9**, pp. 849–859.

Gilmore, P. and R. Gomery (1963). A linear programming solution to the cutting stock problem: Part II. *Operations Research*, **11**, pp. 863–888.

Glover, F. (1989). Tabu search: Part I. *ORSA Journal on Computing*, **1**, pp. 190–206.

Glover, F. (1990). Tabu search: Part II. *ORSA Journal on Computing*, **2**, pp. 4–32.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Graves, R., A. Rinnooy Kan, and P. Zipkin (eds.) (1993). *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*. North-Holland.

Guan, Y., S. Ahmed, and G. Nemhauser (2009). Cutting planes for multi-stage stochastic integer programs. *Operations Research*. To appear.

Guan, Y., S. Ahmed, G. Nemhauser, and A. Miller (2006). A branch-and-cut algorithm for the stochastic uncapacitated lot-sizing problem. *Mathematical Programming*, **105**, pp. 55–84.

Gutjahr, W., A. Hellmayr, and G. C. Pflug (1999). Optimal stochastic single-machine-tardiness scheduling by stochastic branch-and-bound. *European Journal of Operational Research*, **117**, pp. 396–413.

Held, M. and R. Karp (1970). The traveling salesman problem and minimum spanning trees. *Operations Research*, **18**, pp. 1138–1162.

Held, M. and R. Karp (1971). The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, **1**, pp. 6–25.

Herroelen, W. (2007). Generating robust project baseline schedules. In *Tutorials in Operations Research*, pp. 124–144. INFORMS.

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

Herroelen, W. and B. D. Reyck (1999). Phase transitions in project scheduling. *Journal of the Operational Research Society*, **50**, pp. 148–156.

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

Higle, J. and S. Sen (1991). Stochastic decomposition: An algorithm for two-stage stochastic linear programs with recourse. *Mathematics of Operations Research*, **16**, pp. 650–669.

Higle, J. and S. Sen (1996). *Stochastic Decomposition: A Statistical Method for Large Scale Stochastic Linear Programming*. Kluwer Academic Publishers, Dordrecht.

Hiriart-Urruty, J. and C. Lemaréchal (1993). *Convex Analysis and Minimization Algorithms II. Comprehensive Studies in Mathematics*. Springer-Verlag.

Hodgson, T. (1977). A note on single machine sequencing with random processing times. *Management Science*, **23**, pp. 1144–1146.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hoogeveen, J., S. van de Velde, and B. Veltman (1994). Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics*, **55**, pp. 259–272.

Jackson, J. (1955). Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California, Los Angeles.

Jeroslow, R. (1980). A cutting plane game for facial disjunctive programs. *SIAM Journal on Control and Optimization*, **18**, pp. 264–281.

Jia, C. (2001). Stochastic single machine scheduling with an exponentially distributed due date. *Operations Research Letters*, **28**, pp. 199–203.

Johnson, S. (1954). Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, **1**, pp. 61–68.

Kall, P. (1979). Computational methods for solving two-stage stochastic linear programming problems. *Journal of Applied Mathematics and Physics*, **30**, pp. 261–271.

Keller, B. and G. Bayraksan (2009). Scheduling jobs sharing multiple resources under uncertainty: A stochastic programming approach. *IIE Transactions*. To appear.

Kirkpatrick, S., C. Gelatt, and M. Vecchi (1983). Optimization by simulated annealing. *Science*, **220**, pp. 671–680.

Kiwiel, K. (1990). Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming*, **46**, pp. 105–122.

Klein Haneveld, W., L. Stougie, and M. van der Vlerk (1995). On the convex hull of the simple integer recourse objective function. *Annals of Operations Research*, **56**, pp. 209–224.

Klein Haneveld, W., L. Stougie, and M. van der Vlerk (1996). An algorithm for the construction of convex hulls in simple integer recourse programming. *Annals of Operations Research*, **64**, pp. 67–81.

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

Kleywegt, A., A. Shapiro, and T. Homem-De-Mello (2001). The sample average approximation method for stochastic discrete optimization. *SIAM Journal of Optimization*, **12**, pp. 479–502.

Kong, N., A. Schaefer, and B. Hunsaker (2006). Two-stage integer programs with stochastic right-hand sides: A superadditive dual approach. *Mathematical Programming*, **108**, pp. 275–296.

Kubale, M. (1987). The complexity of scheduling independent two-processor tasks on dedicated processors. *Informations Processing Letters*, **24**, pp. 141–147.

Küçükyavuz, S. (2009). On mixing sets arising in probabilistic programming. Technical report, Industrial and Systems Engineering, The Ohio State University. Available at: http://www.optimization-online.org.

Land, A. and A. Doig (1960). An automatic method for solving discrete programming problems. *Econometrica*, **28**, pp. 497–520.

Laporte, G. and F. Louveaux (1993). The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, **13**, pp. 133–142.

Lawler, E., J. Lenstra, A. R. Kan, and D. Shmoys (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons.

Lee, C. and X. Cai (1999). Scheduling one and two-processor tasks on two parallel processors. *IIE Transactions*, **31**, pp. 445–455.

Leus, R. and W. Herroelen (2004). Stability and resource allocation in project planning. *IIE Transactions*, **36**, pp. 667–682.

Linderoth, J. and S. Wright (2003). Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, **24**, pp. 207–250.

Little, J., K. Murty, D. Sweeney, and C. Karel (1963). An algorithm for the Traveling Salesman Problem. *Operations Research*, **11**, pp. 972–989.

Louveaux, F. and M. van der Vlerk (1993). Stochastic programming with simple integer recourse. *Mathematical Programming*, **61**, pp. 301–325.

Lulli, G. and S. Sen (2004). A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems. *Management Science*, **50**, pp. 786–796.

Lustig, I., J. Mulvey, and T. Carpenter (1991). Formulating two-stage stochastic programs for interior point methods. *Operations Research*, **39**, pp. 757–770.

Mak, W., D. Morton, and K. Wood (1999). Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, **24**, pp. 47–56.

Malcolm, D., J. Roseboom, C. Clark, and W. Fazar (1959). Application of a technique for research and development program evaluation. *Operations Research*, **7**, pp. 646–669.

Marchand, H. and L. Wolsey (1999). The 0-1 knapsack problem with a single continuous variable. *Mathematical Programming*, **85**, pp. 15–33.

Martello, S. and P. Toth (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons.

Miller, A., G. Nemhauser, and M. Savelsbergh (2000). On the capacitated lot-sizing and continuous 0-1 knapsack polyhedra. *European Journal of Operational Research*, **125**, pp. 298–315.

Mirchandani, P. and R. Francis (eds.) (1990). *Discrete Location Theory*. John Wiley & Sons.

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

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

Morton, D. and E. Popova (2004). A Bayesian stochastic programming approach to an employee scheduling problem. *IIE Transactions*, **36**, pp. 155–167.

Nemhauser, G. and L. Wolsey (1988). *Integer and Combinatorial Optimization*. John Wiley & Sons.

Ntaimo, L. (2009). Disjunctive decomposition for two-stage stochastic mixed-binary programs with random recourse. *Operations Research*. To appear.

Ntaimo, L. and S. Sen (2007). A branch-and-cut algorithm for two-stage stochastic mixed-binary programs with continuous first-stage variables. *International Journal of Computational Science and Engineering*, **3**, pp. 232–241.

Ntaimo, L. and S. Sen (2008). A comparative study of decomposition algorithms for stochastic combinatorial optimization. *Computational Optimization and Applications*, **40**, pp. 299–319.

Ntaimo, L. and M. Tanner (2008). Computations with disjunctive cuts for two-stage stochastic mixed 0-1 integer programs. *Journal of Global Optimization*, **41**, pp. 365–384.

Pascoe, T. (1966). Allocation of resources - CPM. *Rev Fr Rech Opér*, **38**, pp. 31–38.

Patterson, J. (1976). Project scheduling: The effects of problem structure on heuristic performance. *Naval Research Logistics*, **23**, pp. 95–123.

Perregaard, M. and E. Balas (2001). Generating cuts from multiple-term disjunctions. In Aardal, K. and B. Gerards (eds.) *Proceedings of IPCO VIII, Lecture Notes in Computer Science*, pp. 348–360. Springer-Verlag.

Pinedo, M. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer, New York, 3 edition.

Prékopa, A. (2003). *Probabilistic Programming*, volume 10 of *Handbooks in Operations Research and Management Science*, chapter 5. Elsevier.

Queyranne, M. and A. Schulz (1994). Polyhedral approaches to machine scheduling. Preprint 408/1994, Dept. of Mathematics, Technical University of Berlin.

Rockafellar, R. and R. Wets (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, **16**, pp. 119–147.

Rothkopf, M. (1966). Scheduling with random service times. *Management Science*, **12**, pp. 707–713.

Ruszczyński, A. (1986). A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, **35**, pp. 309–333.

Santoso, T., S. Ahmed, M. Goetschalckx, and A. Shapiro (2005). A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research*, **167**, pp. 96–115.

Schultz, A. (1996). *Polytopes and Scheduling*. Ph.D. thesis, Department of Mathematics, Technische Universität Berlin, Germany.

Schultz, R. (1993). Continuity properties of expectation functions in stochastic integer programming. *Mathematics of Operations Research*, **18**, pp. 578–589.

Schultz, R. (1995). On structure and stability in stochastic programs with random technology matrix and complete integer recourse. *Mathematical Programming*, **70**, pp. 73–89.

Schultz, R. (2003). Stochastic programming with integer variables. *Mathematical Programming*, **97**, pp. 285–309.

Schultz, R., L. Stougie, and M. van der Vlerk (1998). Solving stochastic programs with complete integer recourse: A framework using Gröbner bases. *Mathematical Programming*, **83**, pp. 229–252.

Sen, S. (2005). Decomposition algorithms for stochastic mixed-integer programming models. In Aardal, K., G. Nemhauser, and R. Weismental (eds.) *Handbook of Discrete Optimization*.

Sen, S. and J. Higle (2005). The C^3 theorem and D^2 algorithm for large scale stochastic mixed-integer programming: Set convexification. *Mathematical Programming*, **104**, pp. 1–20.

Sen, S. and H. Sherali (1985). On the convergence of cutting plane algorithms for a class of nonconvex mathematical programs. *Mathematical Programming*, **31**, pp. 42–56.

Sen, S. and H. Sherali (2006). Decomposition with branch-and-cut approaches for two-stage stochastic mixed-integer programming. *Mathematical Programming*, **106**, pp. 203–222.

Shapiro, A. (2003). Monte Carlo sampling methods. In Ruszczyński, A. and A. Shapiro (eds.) *Handbooks in Operations Research and Management Science, Volume 10: Stochastic Programming*, pp. 353–425. Elsevier.

Sliva, E. and K. Wood (2006). Solving a class of stochastic mixed-integer programs with branch and price. *Mathematical Programming*, **108**, pp. 395–418.

Smith, W. (1956). Various optimizers for single stage production. *Naval Research Logistics Quarterly*, **3**, pp. 59–66.

Soroush, H. (2007). Minimizing the weighted number of early and tardy jobs in a stochastic single machine scheduling problem. *European Journal of Operational Research*, **181**, pp. 266–287.

Stougie, L. (1987). *Design and Analysis of Algorithms for Stochastic Integer Programming*. Ph.D. thesis, Center for Mathematics and Computer Sciences, Amsterdam.

van de Vonder, S. (2006). *Proactive-Reactive Procedures for Robust Project Scheduling*. Ph.D. thesis, Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven, Leuven, Belgium.

van de Vonder, S., E. Demeulemeester, and W. Herroelen (2007). A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling*, **10**, pp. 195–207.

van Slyke, R. and R. Wets (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal of Applied Mathematics*, **17**, pp. 638–663.

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

Wolsey, L. (1998). *Integer Programming*. John Wiley & Sons.

Yang, Y. and S. Sen (2009). Enhanced cut generation methods for decomposition-based branch and cut for two-stage stochastic mixed-integer programs. *INFORMS Journal on Computing*. To appear.

Zhu, G., J. Bard, and G. Yu (2007). A two-stage stochastic programming approach for project planning with uncertain activity durations. *Journal of Scheduling*, **10**, pp. 167–180.