Decomposition Algorithms and Parallel Computing for Chance-Constrained and Stochastic Integer Programs with Applications

by

Yan Deng

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Industrial and Operations Engineering)
in the University of Michigan
2016

Doctoral Committee:

Assistant Professor Siqian M. Shen, Chair
Associate Professor Brian Denton
Assistant Professor Ruiwei Jiang
Professor Jon Lee
Associate Professor Clayton D. Scott

2016

A C K N O W L E D G M E N T S

First and foremost, I express the deepest thanks to my advisor Prof. Siqian Shen. She went above and beyond in guiding and supporting me through my four-year doctorate study. I appreciate the flexibility she allows me in exploring interesting ideas and the great help she offers in my job search.

I want to thank Prof. Brian Denton, Prof. Jon Lee, and Prof. Shabbir Ahmed at Georgia Tech for their enlightening ideas and continuous encouragement. Working with them has been a privilege. I also would like to thank Prof. Clayton Scott and Prof. Ruiwei Jiang. I greatly enjoy their courses, and appreciate the valuable perspectives they brought up in committee meetings.

In the Department of IOE, I'd like to thank Chris and Tina for always patiently attending to my questions and requests. Also thanks to my fellow students who accompany and help me through the thorns.

I'm also thankful to my advisors in Tsinghua University, University of Cambridge, and University of Hong Kong for guiding me toward such a fulfilling experience at Michigan.

I would like to acknowledge with much love and gratitude my family.

TABLE OF CONTENTS

Acknowledgments . . . . . **ii**

List of Figures . . . . . **vi**

List of Tables . . . . . **vii**

List of Appendices . . . . . **viii**

List of Abbreviations . . . . . **ix**

List of Notations . . . . . **x**

Abstract . . . . . **xi**

Chapter

1 Introduction **1**

1.1 Background 1
1.1.1 Stochastic Programming Models 1
1.1.2 Uncertainty Description and Model Reformulation 3
1.1.3 Decomposition Algorithms 6
1.2 Thesis Overview 11

2 Chance-Constrained Surgery Planning under Uncertain or Ambiguous Surgery Durations **14**

2.1 Introductory Remarks 14
2.1.1 Literature Review 14
2.1.2 Motivation and Contributions 15
2.2 Problem Formulation 17
2.2.1 Chance-Constrained Model 17
2.2.2 Mixed-Integer Programming Reformulation 19
2.3 Decomposition-based Branch-and-Cut Method 20
2.3.1 Master Problem 20
2.3.2 Pre-filtering and Packing Cuts 21
2.3.3 Basic Separation and Scheduling Cuts 24
2.3.4 Recursion-based Separation and Strengthened Scheduling Cuts 25
2.3.5 Branch-and-Cut Algorithm and Computational Enhancements 27
2.4 Distributionally Robust Variant 28

2.4.1 Discrete Support and Empirical Distribution of ξ . . . 29
2.4.2 Reformulating with ϕ-Divergence Confidence Set . . . 30
2.4.3 Confidence Set Configuration . . . 32
2.5 Computational Studies . . . 33
2.5.1 Experimental Design and Setup . . . 33
2.5.2 Computational Efficacy . . . 36
2.5.3 Chance-Constrained Model versus Cost-Based Model . . . 37
2.5.4 Integrating Versus Separating Allocation and Scheduling . . . 39
2.5.5 Incorporating Data Ambiguity via Distributionally Robust Model . . . 41
2.6 Concluding Remarks . . . 46

3 Solving Chance-Constrained 0-1 Programs with Decomposition and Parallelization . 49

3.1 Introductory Remarks . . . 49
3.2 Dual Decomposition . . . 52
3.2.1 Lagrangian Relaxation . . . 52
3.2.2 Bound-and-Cut Algorithm . . . 54
3.2.3 Cut Aggregation . . . 56
3.3 Parallel Implementation Schemes . . . 57
3.4 Computational Results . . . 60
3.4.1 Instances and Experimental Setup . . . 60
3.4.2 Results of Serial Implementation . . . 60
3.4.3 Results of Parallel Implementation . . . 63
3.5 Concluding Remarks . . . 64

4 Solving Risk-Averse 0-1 Stochastic Programs with Decomposition and Parallelization 67

4.1 Introductory Remarks . . . 67
4.2 Decomposition Methods . . . 68
4.2.1 Problem Formulation . . . 68
4.2.2 Dual Decomposition Framework . . . 69
4.2.3 DD1 by Using $g(0)$ . . . 70
4.2.4 DD2 by Optimizing $g(\lambda)$ Using a Cutting-Plane Method . . . 72
4.2.5 DD3 by Using a Subgradient Method . . . 74
4.3 Distributionally Robust Variants . . . 75
4.4 Parallel Implementation Schemes . . . 76
4.4.1 Overview . . . 76
4.4.2 Basic Parallel . . . 77
4.4.3 Master-worker Parallel with Barriers . . . 77
4.4.4 Master-worker Parallel without Barriers . . . 79
4.5 Computational Results . . . 82
4.5.1 Instances and Experimental Setup . . . 82
4.5.2 Results of Serial Implementation . . . 84
4.5.3 Results of Parallel Algorithms . . . 86
4.5.4 Results of Stochastic Mean-Risk Programs . . . 90
4.5.5 Results of Distributionally Robust Variants . . . 91
4.6 Concluding Remarks . . . 92

Appendices **94**

Bibliography **98**

LIST OF FIGURES

2.1 Comparison of the CCSP model (left) and the cost-based model (right) in $(\mathcal{T}^{\omega}_{\text{wait}}, \mathcal{T}^{\omega}_{\text{over}})$ 40
2.2 Distance tolerances d for KL- and χ^2-divergences according to the number of observed samples (N_{obs}) 46

3.1 A schematic view of using four processes to solve a 12-scenario problem 59
3.2 Speedup versus number of processes under the parallel implementation of DD1 64

4.1 An overview of implementation schemes for dual decomposition algorithms 77
4.2 A schematic view of MWN 82
4.3 Speedup versus number of processes for implementing DD1 86
4.4 Communication time vs Number of processes (N) 88

LIST OF TABLES

2.1 Summary statistics of surgeries of all types based on the extracted data 34
2.2 Normalized opening costs and operating hours of the ORs 34
2.3 Comparison of the proposed branch-and-cut approach versus directly calling the solver in solving the MIP reformulation of CCSP 37
2.4 Comparison of the chance-constrained model and the cost-based model in cost and zero-overtime reliability 38
2.5 Comparison of the integrated model CCSP versus the allocation-only model CCBP . . 41
2.6 Performance of CCSP solutions under different distribution assumptions 42
2.7 Probability of waiting $> \epsilon_i$ among all 25 surgeries given by CCSP 43
2.8 Comparison of CCSP and DR-CCSP for $\beta = 0.90$ and $\Omega = \Omega_{\text{Log}}(N_{\text{MC}})$ 45
2.9 Probability of waiting $> \epsilon_i$ among all 25 surgeries given by CCSP and DR-CCSP models 46
2.10 The effect of N_{obs} on the performance of DR-CCSP solutions 47
2.11 Probability of waiting $> \epsilon_i$ given by DR-CCSP models under different N_{obs} 47

3.1 Comparison of the four schemes in serial computational time (or optimality gap) . . . 61
3.2 Other computational details of the three dual decomposition schemes 62
3.3 Comparison of `DDA` and existing methods in runtime 63
3.4 Parallel efficiency under the two parallel schemes 65

4.1 Scales of different instances and performance of their LP relaxations 84
4.2 Time (in seconds) and iteration counts in the serial implementation 85
4.3 Number of evaluated x-solutions 87
4.4 The percentage of communication time contributed by the master 89
4.5 Computational time of all tested schemes 90
4.6 Results of mean-risk model variants and its expectation counterpart under MWB . . . 91
4.7 Computational results of distributionally robust risk-averse problem under MWB . . . 91

LIST OF APPENDICES

A Appendix for Chapter 2 . . . . . **94**

B Appendix for Chapter 3 . . . . . **95**

C Appendix for Chapter 4 . . . . . **96**

LIST OF ABBREVIATIONS

SAA	sample average approximation	3
CVaR	conditional value-at-risk	3
MIP	mixed-integer program	4
DR	distributionally robust	5
NAC	nonanticipativity constraint	10
OR	operating room	11
CCSP	chance-constrained surgery planning	16
CCBP	chance-constrained bin packing	21
DR-CCSP	distributionally robust chance-constrained surgery planning	28
PMF	probability mass function	28
KL	Kullback-Leibler	33
CCSS	chance-constrained surgery scheduling	39
BP	Basic Parallel	76
MWB	Master-worker Parallel with Barriers	76
MWN	Master-worker Parallel without Barriers	76

LIST OF NOTATIONS

$A^\top$	transpose of matrix A
$\|S\|$	cardinality of set S
$S \setminus T$	set minus $\{x \in S : x \notin T\}$
$S \subseteq T$	subset
$\lfloor x \rfloor$	largest integer y such that $y \leq x$
$\mathbf{1}(\cdot)$	indicator function that returns 1 if $\cdot$ is true and 0 otherwise

ABSTRACT

The primary focus of this dissertation is to develop solution methods for stochastic programs, especially those with binary decisions and risk-averse features such as chance constraint or risk-minimizing objective. We approach these problems through a scenario-based reformulation, e.g., sample average approximation, which is more amenable to solution by decomposition methods. The reformulation is often of intractable scale due to the use of a large number of scenarios to represent the uncertainty. Our goal is to develop specialized decomposition algorithms that take advantage of the problem structure and solve the problem in reasonable time.

We first study a surgery planning problem with uncertainty in surgery durations. A common practice is to first assign operating rooms to surgeries and then to develop schedules. We propose a chance-constrained model that integrates these two steps, yielding a better tradeoff between cost and the quality of service. A branch-and-cut algorithm is developed, which exploits valid inequalities derived from a bin packing problem and a series of single-machine scheduling problems. We also discuss models and solutions given ambiguous distributional information. Computational results demonstrate the efficacy of the proposed algorithm and provide insights into enhancing performance by the integrated model, managing quality of service via chance constraint, and using data to guide planning under distributional information ambiguity.

Next, we study general chance-constrained 0-1 programs, where decisions made before the realization of uncertainty are binary. As most of the existing methods fail when the number of scenarios is large, we develop dual decomposition algorithms that find solutions through bounds and cuts efficiently. Then we derive a proposition about computing the Lagrangian dual whose application substantially reduces the number of subproblems to solve, and develop a cut aggregation method that accelerates the solution of individual subproblems. We also explore non-trivial parallel schemes to implement our algorithms in a distributed system. All of them are shown to improve the speed of the algorithms effectively.

We then continue to study dual decomposition, but for risk-averse stochastic 0-1 programs, which do not have chance constraints but minimize the risk of some random outcome measured by a coherent risk function. Using generic dual representations for coherent risk measures, we derive

an equivalent risk-neutral minimax reformulation for the considered problem, to which dual decomposition methods apply. Motivated by some observation of inefficiency in the foregoing work, we investigate in more depth how to exploit the Lagrangian relaxation by comparing three different approaches for computing lower bounds. We also study parallelism more comprehensively, testing schemes that represent different combinations of basic/master-worker, synchronous/asynchronous and push/pull systems, and identify that the best is a master-worker, asynchronous and pull scheme, which achieves near-linear or even super-linear speedup.

CHAPTER 1

Introduction

1.1 Background

In this dissertation, we study structure-exploiting algorithms for solving stochastic programs. Stochastic programming is an optimization branch that considers models containing uncertain data parameters. A few examples include military (Langer et al., 2012), energy (Carøe and Schultz, 1998; Wang et al., 2012), finance (Kouwenberg, 2001; Yu et al., 2003), healthcare (Denton et al., 2007; Salmerón and Apte, 2010) and supply chain (Goh et al., 2007; Santoso et al., 2005). We refer the reader to Ruszczynski and Shapiro (2003), Birge and Louveaux (2011) and Shapiro et al. (2014) for more extensive discussions on stochastic programming. In what follows, we review some basic models and algorithms that we use extensively throughout the dissertation.

1.1.1 Stochastic Programming Models

In stochastic programming, the values of uncertain data become known only after experiments. Decisions are classified into two groups: those made before the realization of uncertainty as *first-stage decisions*, and those made after as *second-stage decisions* (or *recourse*). In this chapter, we represent the random parameters as a vector ξ, the first-stage decision variables as x, and the second-stage decision variables as $y(\xi)$. Depending on the attitude toward risk, stochastic programs have the following two categories.

Risk-neutral stochastic programs. When the uncertainty recurs far more frequently than decision making, a decision maker tends to account equally for every possible realization and evaluate a decision based on its *average* performance. For instance, consider a problem about locating distribution centers to satisfy customer demand. Once the locations are picked, they remain unchanged in the long run and serve demands, which vary on a daily basis. Therefore to evaluate a choice of the locations, one can calculate the expected payoff as the time-discounted expectation

of sales minus the one-time construction cost. Traditional stochastic programming often refers to two-stage risk-neutral stochastic programs of the form:

$$\min \quad c^\top x + \mathbb{E}[f(x,\xi)] \tag{1.1a}$$

$$\text{s.t.} \quad x \in X, \tag{1.1b}$$

where X is a non-empty set in $\mathbb{R}^d$, and c is a vector of magical cost values for first-stage decisions. The expectation in (1.1a) is taken with respect to the probability distribution of ξ. The real-valued function $f(x,\xi)$ represents the cost from a recourse problem where the second-stage decisions are made after the uncertainty ξ is disclosed:

$$f(x,\xi) = \min \quad \sigma(\xi)^\top y(\xi) \tag{1.2a}$$

$$\text{s.t.} \quad y(\xi) \in \mathcal{Y}(x,\xi). \tag{1.2b}$$

Here, $\sigma(\xi)$ is a linear cost vector for the second-stage decisions. Without loss of generality, one can assume that uncertain parameters only appear in the constraints and thus replace $\sigma(\xi)$ with σ (Birge and Louveaux, 2011). $\mathcal{Y}(x,\xi)$ is a non-empty set in $\mathbb{R}^{d'}$ parameterized by x and ξ. An example of $\mathcal{Y}(x,\xi)$ with $\xi = (\tilde{T}, \tilde{W}, \tilde{h})$ is

$$\mathcal{Y}(x,\xi) = \left\{ y \in \mathbb{R}^{d'}_+ : \tilde{T}x + \tilde{W}y \geq \tilde{h} \right\} \tag{1.3}$$

where $\tilde{T}$, $\tilde{W}$ and $\tilde{h}$ are properly-sized matrices (vectors) that contain random components. Sets of this form are typically seen in two-stage stochastic linear programs.

Risk-averse stochastic programs. When decisions are made more frequently such that each goes through only a small number of uncertainty recurrences, the decision maker tends to be more concerned with the risk of a decision. One way to model such a risk-averse attitude is *chance-constrained programming* (cf. Birge and Louveaux, 2011; Shapiro et al., 2014), where some constraints are expressed in terms of probabilistic statements about the first-stage decisions. This is particularly useful when the cost or benefit of the second-stage decisions is difficult to assess. A generic chance-constrained program has the form:

$$\min \quad c^\top x \tag{1.4a}$$

$$\text{s.t.} \quad \mathbb{P}\{x \in \mathcal{X}(\xi)\} \geq 1 - \epsilon \tag{1.4b}$$

$$x \in X \tag{1.4c}$$

Here $\mathcal{X}(\xi)$ is a non-empty set parameterized by ξ, and $\epsilon \in [0,1)$ is a risk tolerance, typically close to 0, that limits the probability of some undesirable outcome described by $x \notin \mathcal{X}(\xi)$. An example of $\mathcal{X}(\xi)$ with $\xi = (\tilde{T}, \tilde{W}, \tilde{h})$ is

$$\mathcal{X}(\xi) = \left\{ x \in \mathbb{R}^d_+ : \exists y \in \mathbb{R}^{d'}_+ \text{ s.t. } \tilde{T}x + \tilde{W}y \geq \tilde{h} \right\} \tag{1.5}$$

which typically appears in chance-constrained *linear* programs.

Another way to model risk aversion is to include some risk measure explicitly to the objective of a two-stage stochastic program, and construct a mean-risk stochastic program (Ahmed, 2006) as:

$$\begin{aligned} \min \quad & w_1 \left(c^\top x + \mathbb{E}[f(x,\xi)] \right) + w_2 \mathbb{D}[f(x,\xi)] \\ \text{s.t.} \quad & x \in X. \end{aligned}$$

Here $\mathbb{D}$ is a risk measure, and w_1 and w_2 are the non-negative weights to trade off the expected cost and risk. Classical examples include the mean-variance portfolio optimization model (Markowitz, 1968) where variance is used as $\mathbb{D}$, and the mean-CVaR model (Ahmed, 2006) where conditional value-at-risk (CVaR) (Rockafellar and Uryasev, 2000) is used as $\mathbb{D}$.

1.1.2 Uncertainty Description and Model Reformulation

In this dissertation, we assume that the uncertainty ξ is independent from the decisions made. This may not be true in some applications (Cooper et al., 2006; Peeta et al., 2010; Solak et al., 2010). See Goel and Grossmann (2006) for an extensive study of stochastic programs with decision-dependent uncertainty.

Sample average approximation. When the probability distribution of ξ is known, the computation of $\mathbb{E}[f(x,\xi)]$ can be difficult because it may involve high-dimensional integration. Most theories and techniques in stochastic programming are therefore based on using a discrete probability distribution with finite support to approximate the original distribution of ξ. Specifically, consider only a finite number of realizations: $\{\xi^1, \ldots, \xi^K\}$. Each is referred to as a *scenario*, with an occurring probability $p_k > 0$. These probabilities satisfy $\sum_{k=1}^K p_k = 1$. We can then approximate the expectation in (1.1a) as

$$\sum_{k=1}^K p_k f(x, \xi^k),$$

and turn (1.1a)–(1.1b) into a deterministic optimization problem. When the scenarios are from a Monte Carlo sample of ξ, this approach is known as sample average approximation (SAA) (Kley-

wegt et al., 2002; Shapiro and Homem-de Mello, 2000), and the obtained deterministic problem is called *the SAA reformulation*. For example, the SAA problem for the two-stage stochastic linear program (1.1a)–(1.3) is

$$\min \quad c^\top x + \sum_{k=1}^{K} p_k \sigma^\top y^k \tag{1.6a}$$

$$\text{s.t.} \quad T^k x + W^k y^k \geq h^k, \ \forall k = 1, \ldots, K \tag{1.6b}$$

$$x \in X, \ y^k \geq 0, \ \forall k = 1, \ldots, K \tag{1.6c}$$

where $y^k = y(\xi^k)$, and (T^k, W^k, h^k) is the realization of $(\tilde{T}, \tilde{W}, \tilde{h})$ in each scenario k. The expectation (taken with respect to the choice of the sample) of the optimal objective value of this SAA problem converges to the optimal objective value of the original program at a rate of $O(|K|^{-1/2})$ (Shapiro, 1993).

On the other hand, Luedtke and Ahmed (2008) and Pagnoncelli et al. (2009) study the SAA approach for chance-constrained programs that take the form of (1.4a)–(1.4c). Specifically, they approximate the chance constraint (1.4b) by

$$\sum_{k=1}^{K} p_k \mathbf{1}\left(x \in \mathcal{X}(\xi^k)\right) \geq 1 - \epsilon \tag{1.7}$$

where $\mathbf{1}(\cdot)$ is an indicator function returning 1 if $\cdot$ is true and 0 otherwise. If X is finite, the resulting SAA problem yields an optimal solution of the original chance-constrained program, with probability approaching 1 exponentially fast as K increases. Using chance-constrained linear programs of the form (1.4a)–(1.5) for example, the SAA problem can be written as

$$\min \quad c^\top x \tag{1.8a}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} p_k z_k \leq \epsilon \tag{1.8b}$$

$$T^k x + W^k y^k + z_k m^k \geq h^k, \ \forall k = 1, \ldots, K \tag{1.8c}$$

$$x \in X, \ y^k \geq 0, \ z_k \in \{0, 1\}, \ \forall k = 1, \ldots, K. \tag{1.8d}$$

Here $z_1, \ldots, z_K$ are artificial binary decision variables. For each $k \in \{1, \ldots, K\}$, m^k is a vector of sufficiently large scalars such that $T^k x + W^k y^k + z_k m^k \geq h^k$ is relaxed when $z_k = 1$.

Overall, SAA provides an avenue to approximately solve both risk-neutral and risk-averse stochastic programs, by transforming them to deterministic linear programs or mixed-integer program (MIP). Using a larger number of scenarios makes it a better approximation, though more

difficult to solve. Therefore, to get a good solution reasonably fast, it is important to have efficient algorithms for the SAA formulations.

Distributionally robust stochastic programming. Since the beginning of this chapter, we have been assuming that the probability distribution of ξ is perfectly known. However, this assumption may not be realistic, because *(i)* it may be challenging to specify a precise probability distribution for real world components, and *(ii)* the solution may be sensitive to the ambiguous probability distribution, and thus be suboptimal in practice. To address this issue, distributionally robust (DR) approaches are developed (see, e.g., Calafiore and El Ghaoui, 2006; Delage and Ye, 2010; Erdoğan and Iyengar, 2006; Ghaoui et al., 2003; Scarf et al., 1958). Let P_ξ denote the probability distribution of ξ. In DR stochastic programming, P_ξ is assumed belonging to a pre-defined *uncertainty set* $\mathcal{P}$ consisting of all possible distributions. This set is data-driven, for example, a ball around a density function estimated from data samples. The optimization is then performed over the *worst-case* expectation or risk as P_ξ being from set $\mathcal{P}$. For classical stochastic programs presented in (1.1a)–(1.1b), the DR model is:

$$\min_{x\in X} \max_{P_\xi\in\mathcal{P}} \quad c^\top x + \mathbb{E}\left[f(x,\xi)\right]. \tag{1.9}$$

For chance-constrained programs presented in (1.4a)–(1.4c), the DR model is

$$\min_{x\in X} \left\{ c^\top x : \inf_{P_\xi\in\mathcal{P}} \mathbb{P}\{x \in \mathcal{X}(\xi)\} \geq 1-\epsilon \right\} \tag{1.10}$$

where the chance constraint is required to be satisfied for each $P_\xi \in \mathcal{P}$.

The construction of $\mathcal{P}$ can use any accessible distributional information from historical data and/or the decision maker's knowledge. For example, $\mathcal{P}$ can consist of all the distributions that have the first and second moments matching the empirical (Calafiore and El Ghaoui, 2006; Vandenberghe et al., 2007; Zymler et al., 2013a,b). Pre-known structural properties like unimodality, symmetry and convexity can also be incorporated to define $\mathcal{P}$ (Popescu, 2005; Van Parys et al., 2015). Other than moments, similarity of probability distributions can also be drawn based on density functions. Jiang and Guan (2015) construct $\mathcal{P}$ as the set of distributions whose density functions are sufficiently close, measured by ϕ-divergence, to an empirical distribution. By exploiting the structures of $\mathcal{P}$ and the original stochastic program, the DR models often have equivalent reformulations or approximations that are tractable. Interested readers can refer to Calafiore and El Ghaoui (2006); Delage and Ye (2010); Ghaoui et al. (2003); Zymler et al. (2013a,b) for more extensive discussions.

1.1.3 Decomposition Algorithms

This dissertation mainly focuses on studying efficient algorithms for solving the SAA reformulations of risk-neutral and risk-averse stochastic programs. As the SAA problems are often computationally demanding due to size, decomposition algorithms based on conquering a series of smaller subproblems are central to finding optimal solutions. They have long been used for solving stochastic programs (see, e.g., Ahmed, 2013; Carøe and Schultz, 1999; Higle and Sen, 1991; Luedtke et al., 2010; Van Slyke and Wets, 1969). Below, we describe the *Benders decomposition*, a very common approach. Many algorithms studied in this dissertation are extensions of Benders decomposition, and/or compared against it.

Benders decomposition. In general, it allows the solution of large linear programs that have a special block structure. Its key idea is to construct a convex approximation (of the original problem) defined by a set of valid inequalities that are generated over iterations. The approximation problem and the generated inequalities are often called *the master problem* and *cuts* (or cutting planes), respectively. The cuts are identified based on feasibility criteria or optimality conditions in a *separation problem*, which is solvable through smaller *subproblems*. The block structure is often seen in stochastic programs. A typical example is the SAA problem of two-stage stochastic linear programs presented in (1.6a)–(1.6c). In that model, scenarios $1,\dots,K$ are connected by the first-stage decision x. When x is fixed, the remaining problem becomes solvable independently for each scenario k:

$$g_k(x) := \min_{y^k \geq 0} \left\{ \sigma^\top y^k : W^k y^k \geq h^k - T^k x \right\} \tag{1.11}$$

$$= \max_{q^k \geq 0} \left\{ \left(h^k - T^k x\right)^\top q^k : (q^k)^\top W^k \leq \sigma^\top \right\}, \tag{1.12}$$

where the second equality follows from strong duality. The original problem (1.6a)–(1.6c) can easily be rewritten as:

$$\min_{x \in X, u_1, \dots, u_K} \left\{ c^\top x + \sum_{k=1}^{K} p_k u_k : u_k \geq g_k(x),\ \forall k = 1,\dots,K \right\} \tag{1.13}$$

To exploit the decomposable structure, the algorithm approximates (1.13) by the following master problem:

$$\min_{x \in X, u_1, \dots, u_K} \quad c^\top x + \sum_{k=1}^{K} p_k u_k \tag{1.14a}$$

$$\text{s.t.} \quad \alpha u_k + \beta^\top x \geq \gamma, \ \forall (\alpha, \beta, \gamma) \in \mathcal{L}_k, \ k = 1, \ldots, K. \tag{1.14b}$$

Here, $\mathcal{L}_k \subseteq \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}$ represents a set of inequalities that have been generated for scenario k so far through the algorithm. The master problem contains only a subset of constraints that are necessary to describe (1.13), and thus is a relaxation.

Given a master-problem solution $(\hat{x}, \hat{u}_1, \ldots, \hat{u}_K)$, we solve (1.12) with $\hat{x}$ plugged in for x for every scenario k. Any feasible x of (1.6a)–(1.6c) must allow (1.12) to have an optimal solution. Therefore, if (1.12) is unbounded, we identify a positive-cost extreme ray $\hat{d}^k$ and add the following inequality to the master problem:

$$\left(\hat{d}^k\right)^\top \left(h^k - T^k x\right) \leq 0, \tag{1.15}$$

i.e., $\mathcal{L}_k = \mathcal{L}_k \cup \left\{\left(0, (T^k)^\top \hat{d}^k, (h^k)^\top \hat{d}^k\right)\right\}$. If (1.12) has an optimal solution $\hat{q}^k$, we need to check whether

$$u_k \geq \left(h^k - T^k x\right)^\top \hat{q}^k, \tag{1.16}$$

which is a valid inequality for (1.13). If not, we add (1.16) to the master problem, i.e., $\mathcal{L}_k = \mathcal{L}_k \cup \left\{\left(1, (T^k)^\top \hat{q}^k, (h^k)^\top \hat{q}^k\right)\right\}$. Note that (1.15) and (1.16) are for ensuring the master-problem solution x to be feasible and optimal to the original problem, and thus they are called *feasibility cut* and *optimality cut*, respectively. The above procedure is repeated iteratively until the gap between the upper bound, the smallest objective value $c^\top \hat{x} + \sum_{k=1}^K p_k g_k(\hat{x})$ among all the feasible $\hat{x}$, and the lower bound, the optimal objective value of the master problem, are sufficiently small.

Many applications involve discrete decisions, especially binary decisions to model logical conditions. As integer programming is generally NP-hard, the discrete nature of decisions further amplifies the computational intractability of stochastic programs which are already difficult due to the large scale caused by data uncertainty. Most of the problems studied in this dissertation are stochastic integer programs, where some decision variables are restricted to be integer. Next, we review an extended Benders decomposition algorithm (Laporte and Louveaux, 1993) which is extensively used for solving stochastic programs with a binary first stage. More importantly, it falls into the *branch-and-cut* framework which is a successful technique for solving mixed-integer programs. (See Wolsey and Nemhauser (2014) for more extensive discussions on the branch-and-cut method.) Chapters 2 and 3 discuss many specialized branch-and-cut algorithms to solve stochastic programs of certain structures.

The integer L-shaped method. Consider the two-stage risk-neutral stochastic program (1.6a)–(1.6c), but this time x is a vector of binary variables, i.e., $X \subseteq \{0,1\}^d$. If we construct a

branch-and-bound tree to solve the problem, each node corresponds to fixing some components of x to 0 or 1. For any node n, the node problem can be written as:

$$\min_{x\in X, v} \left\{ c^\top x + v : v \geq \sum_{k=1}^{K} p_k g_k(x),\ x_j = 0,\ \forall j \in F_0(n),\ x_j = 1,\ \forall j \in F_1(n) \right\} \tag{1.17}$$

in which $g_k(x)$ is defined in (1.11). $F_0(n)$ and $F_1(n)$ are disjoint subsets of $\{1,\dots,d\}$. Applying the idea of Benders decomposition, we approach (1.17) through an approximation problem as

$$\min_{x\in X, v} \quad c^\top x + v \tag{1.18a}$$

$$\text{s.t.} \quad \alpha v + \beta^\top x \geq \gamma,\ \forall (\alpha,\beta,\gamma) \in \mathcal{L} \tag{1.18b}$$

$$x_j = 0,\ \forall j \in F_0(n),\ x_j = 1,\ \forall j \in F_1(n). \tag{1.18c}$$

Set $\mathcal{L}$ collects cuts that are similar to (1.15) and (1.16). However, due to the discrete nature of x, they are formulated differently. To exclude some solution $\hat{x}$ that leads to some $g_k(\hat{x})$ infeasible, we can still use (1.15), but also the following cut:

$$\sum_{j\in\{1,\dots,d\}:\hat{x}_j=1} (1 - x_j) + \sum_{j\in\{1,\dots,d\}:\hat{x}_j=0} x_j \geq 1 \tag{1.19}$$

which forbids $\hat{x}$ through its 0-1 pattern. To ensure $v \geq \sum_{k=1}^{K} p_k g_k(x)$, we can use the following cut:

$$v \geq \ell + \left(\sum_{k=1}^{K} p_k g_k(\hat{x}) - \ell \right) \left(1 - \sum_{j\in\{1,\dots,d\}:\hat{x}_j=1} (1 - x_j) - \sum_{j\in\{1,\dots,d\}:\hat{x}_j=0} x_j \right) \tag{1.20}$$

where ℓ is some pre-known lower bound for the expected second-stage cost. Note that all of these cuts are valid inequalities with respect to the current node problem, but also to the original program. They are added to $\mathcal{L}$ which is not node-dependent, so that all of the nodes become affected. Just like branch-and-bound, this approach also maintains an upper bound μ. The steps are outlined in Algorithm 1.1.

Other decomposition algorithms. Most existing solution methods for stochastic programs work with some scenario-based reformulations/approximations, e.g., the SAA problem, which are more amenable to decomposition algorithms. They specialize in exploiting the particular structure of the underlying programs, and use different master problems and cuts.

To give a brief survey of decomposition algorithms in two-stage (risk-neutral) stochastic programming, we categorize the programs according to their variable types, and label each class by

Algorithm 1.1 The integer L-shaped method to solve binary-first-stage stochastic programs

$\mu \leftarrow +\infty$, $\mathcal{L} \leftarrow \emptyset$.
repeat
  pick some unexplored node n
  repeat
    solve the linear programming relaxation of (1.18a)–(1.18c)
    if feasible and the optimal objective value $< \mu$ **then**
      let $(\hat{x}, \hat{v})$ be an optimal solution
      if $\hat{x} \notin \{0,1\}^d$ **then**
        create two new nodes by branching on a fractional component of $\hat{x}$.
      else
        solve $g_1(\hat{x}), \ldots, g_K(\hat{x})$
        if some $g_k(x) = +\infty$ ((1.11) infeasible or (1.12) unbounded) **then**
          add cut (1.19) to $\mathcal{L}$.
        else
          $\mu \leftarrow \min\left\{\mu, \sum_{k=1}^{K} p_k g_k(\hat{x})\right\}$
          if $\sum_{k=1}^{K} p_k g_k(\hat{x}) > \hat{v}$ **then**
            add cut (1.20) to $\mathcal{L}$.
          end if
        end if
      end if
    end if
  until no new cut added to $\mathcal{L}$
until all nodes explored

$*/*$. The first $*$ characterizes the first-stage variables: C if all continuous, B, if all binary, G if all general integers, MB (or MG) if it is a mixed of binary (or general integer) and continuous variables, and U if not restricted to any type. In the same way, the second $*$ characterizes the second stage. Carøe and Tind (1998) propose applying lift-and-project cuts generated from one scenario but valid for all to solve MB/G. Sen and Higle (2005) develop disjunctive cuts for B/MB, and Sen and Sherali (1985) then extend the method for B/MG by incorporating a branch-and-cut approach. For MG/G, Carøe and Tind (1998) propose a conceptual method that generates Gomory cuts and construct non-convex optimality cuts, and Ahmed et al. (2004) propose a branch-and-bound method. Kong et al. (2006) also study a branch-and-bound method, which is then applied to a superadditive dual formulation for solving G/G. Gade et al. (2014) and Zhang and Küçükyavuz (2014) resort to Gomory cuts to solve B/G and G/G, respectively. Ahmed (2013) proposes a method for B/U that iteratively explores and cuts off candidate solutions from solving scenario subproblems. Other studies apply decomposition to dual formulations rather than the primal. For MG/MG, Carøe and Schultz (1999) recover a decomposable structure by dualizing nonanticipativity constraint (NAC) (cf. Escudero et al., 1993; Fernandez, 1995) and apply decomposition in a branch-and-bound framework to solve the resulting dual formulation. Extending from this method, Lubin et al. (2013) develop a formulation that permits a parallel solution of the master program.

Regarding chance-constrained programs, SAA can be used to find good solutions and statistical bounds to the original programs (Luedtke and Ahmed, 2008; Pagnoncelli et al., 2009). The SAA problem, e.g., (1.8a)–(1.8d), however, faces a challenge in addition to the large-scale nature and the existence of discrete variables. Its linear programming relaxation is weak due to the use of very large coefficients to deactivate the chance constraint in some scenarios, like m^k in (1.8c) which are often called *big-M*s. This makes the SAA problem even harder to solve. Existing work takes advantage of the structure of problems in particular applications to tighten the values of big-M coefficients or to develop alternative valid inequalities (Qiu et al., 2014; Song and Luedtke, 2013; Song et al., 2014). For more general problems, Luedtke et al. (2010) and Küçükyavuz (2012) develop strengthened formulations which can be solved faster for chance-constrained programs with only right-hand-side random (i.e., $\mathcal{X}(\xi) = \{x : Tx \geq h(\xi)\}$). Zhang et al. (2014) propose strong valid inequalities for multi-stage chance-constrained programs and observe that decomposition algorithms are needed to solve large instances. For problems where decisions are binary, Ahmed et al. (2015b) develop decomposition algorithms based on dual formulations which are obtained from dualizing nonanticipativity constraints. Luedtke (2014) develop a branch-and-cut algorithm for general chance-constrained programs where the inequalities with big-M coefficients are not enforced explicitly but through tight cuts generated during the algorithm. In another line of work, decomposition algorithms are developed for *two-stage* chance-constrained programs which additionally include the expected second-stage cost in the objective to minimize along with the de-

terministic cost from the first stage. This class of programs models the tradeoff between cost and risk very well. The new objective is the same as the objective in risk-neutral stochastic programs (e.g., (1.1a)). Wang and Shen (2012) and Wang et al. (2012) extend Benders decomposition to solve such problems, which, however, do not tackle the computational difficulty brought by big-M coefficients. In addressing this challenge, Zeng et al. (2014) propose another decomposition algorithm which is based on big-M-free bilinear feasibility and optimality cuts. Liu et al. (2015), in a concurrent work, develop specialized strong valid inequalities that enhance the computational efficiency substantially.

1.2 Thesis Overview

In Chapter 2, we study a practical problem of surgery planning, motivated by the large uncertainty in surgery durations, the high cost of operating room (OR) resources and the importance of surgical service quality. We consider the uncertainty in surgery durations, and build a model which minimizes the cost of opening ORs subject to a chance constraint limiting OR overtime and surgery waiting time. We use a chance-constrained model rather than the traditional cost-based model, because there is rarely a fair measure to map overtime or waiting time into monetary cost. In contrast to a common practice that first assigns OR to surgeries and then develops schedules for individual ORs, we integrate these two steps into a single model to seek better-performing solutions which are made available by the extra flexibility. The model turns out to be a complex chance-constrained program that involves two stages of mixed-binary decisions. To solve it, we design a specialized branch-and-cut algorithm. The algorithm uses valid inequalities, which are built upon an analogous bin packing problem and a series of single-machine scheduling problems. Furthermore, we allow surgery durations to follow ambiguous distributions and develop a DR variant that controls the worst-case probability for the chance constraint to be satisfied. By employing the results from Jiang and Guan (2015), we transform the DR model into a standard chance-constrained program with a perturbed risk level that the proposed algorithm can cope with. Testing the models and the algorithms on real data from an outpatient surgery center, we show *(i)* the efficacy of the proposed algorithm compared with state-of-the-art MIP solvers; *(ii)* the feasibility and necessity of solving the integrated surgery-planning problems in practice; *(iii)* the reliability of solutions to the chance-constrained model compared with solutions to a cost-based model that penalizes waiting and overtime; *(iv)* the better tradeoff between cost and service quality achieved by solutions to our integrated model compared with the tradeoff from models that separate OR allocation and surgery scheduling; and finally *(v)* the robustness of solutions to the DR variant compared with solutions to the original model, especially in scenarios with surgery cancellations. Moreover, although the models and the algorithms are motivated by a surgery planning application, they can be applied to

more general stochastic scheduling problems.

In Chapter 3, we investigate a general class of chance-constrained discrete problems: chance-constrained 0-1 programs, where decisions made before the realization of uncertainty are binary. We study a decomposition method which follows a procedure of iteratively exploring and cutting off candidate solutions, which are discovered from scenario subproblems, until the lower and upper bounds for the optimal objective value are sufficiently close. In this case, we apply decomposition to a dual formulation obtained from relaxing a NAC (cf. Escudero et al., 1993; Fernandez, 1995) and a packing inequality (e.g., (1.8b)) in the MIP reformulation of the considered program. We derive a proposition that simplifies the computation of the Lagrangian dual. It enables us to replace some subgradient iterations with simple arithmetic, and effectively reduces the number of subproblems to solve. Our experiments show that the change has substantially accelerated the algorithm. We also develop a subroutine of cut aggregation that accelerate the solution of individual subproblems, which makes the algorithm even faster. To further enhance the efficacy, we explore two parallel schemes for implementing the algorithm in a distributed system: one which simply distributes the computation of subproblems evenly across parallel processes, and one which adopts a Master-Worker structure which reduces duplicate efforts and potentially alleviates waiting caused by synchronization. We observe speedup under both schemes, and the second scheme outperforms when the number of processes is large. Despite the speed advantage of the proposed algorithms and schemes, the results also indicate the time-expense of the subgradient method and the inefficiency of parallel execution as the number of processes increases due to due to synchronization barriers. Therefore we are motivated to further investigate these two aspects in the following chapter.

In Chapter 4, we consider another class of risk-averse stochastic 0-1 programs, which do not use chance constraint but build objective functions upon a *coherent risk measure*. Using a generic dual representation for coherent risk measures, we derive an equivalent risk-neutral minimax reformulation for the considered problem. We show that the reformulation is well suited to solution by the dual decomposition approach that we apply to solve chance-constrained 0-1 programs in Chapter 3. We develop three algorithms, DD1, DD2, and DD3, which differ in recovering a lower bound from the Lagrangian dual. DD1 simply uses the functional value of the Lagrangian relaxation at zero (i.e., when the multiplier is set to zero). Although this corresponds to relaxing the NAC and essentially allowing decisions to be made for each specific scenario, it may still yield a good lower bound if most of the scenarios are similar and agree on the same best decision. DD2 and DD3 contain an inner loop for optimizing the Lagrangian relaxation to produce a tighter lower bound. Strategically, they are similar to the decomposition method discussed in Chapter 3. However, motivated by the observed inefficiency of the subgradient subroutine in that approach, here we *(i)* still use a subgradient method but based on a different NAC in DD3, and *(ii)* use a cutting-plane method instead in DD2. On the other hand, we show that the proposed algorithms work

well for solving the DR variant of the considered problem. Furthermore, we design three parallelization schemes representing different combinations of basic-parallel/master-worker-parallel, synchronous/asynchronous, and push/pull systems, enabling a comprehensive examination of the parallel execution. To test the models, algorithms and implementation schemes, we use CVaR as the risk measure and stochastic programming instances in popular contexts like server location and knapsack problem from SIPLIB (Ahmed et al., 2015a). We demonstrate the speed advantage of the algorithms for cases with large numbers of scenarios, and the good scalability of the master-worker, asynchronous, and pulling parallel schemes which achieve near-linear and even super-linear speedups.

CHAPTER 2

Chance-Constrained Surgery Planning under Uncertain or Ambiguous Surgery Durations

2.1 Introductory Remarks

Surgery planning must balance the competing goals of high utilization and low overtime of ORs under uncertain surgery durations. These goals are motivated in part by the high cost of surgery resources and the cost of nurse overtime, which can significantly affect nurse turnover in hospitals (see a report by Nursing Solutions, Inc., 2013). In addition to utilization of ORs and overtime, hospital administrators must also pay attention to limiting delays of individual surgeries and patient waiting. All these efforts are closely related to the improvement of service quality, effectiveness, and efficiency when planning surgical care delivery.

In this chapter, we build optimization models integrating OR allocation and surgery planning decisions and we use a chance constraint to limit the probability of having inadmissible surgery waiting and OR overtime. To solve the model which is a complex chance-constrained mixed-integer program, we develop strong valid inequalities and propose a specialized branch-and-cut algorithm. We also investigate a problem variant with distributional ambiguity, and study a DR variant for producing more reliable surgery plans under limited data and misspecified problem parameters. In computational experiments, we demonstrate the efficacy of the proposed algorithms, and compare different models through post-optimization simulation to analyze how they balance service quality and resource utilization.

2.1.1 Literature Review

Surgery Planning under Uncertainty Most hospitals make decisions about which patients to schedule for a particular day sequentially, typically in the 12 weeks prior to the day of surgery. This defines a surgical listing. Approximately 24–48 days prior to surgery, a detailed schedule is generated providing the OR assignment and the planned start time of each surgery. In this

chapter, we focus on this latter problem of OR allocation and surgery scheduling. The majority of the stochastic optimization literature for surgery planning has separately covered scheduling surgeries in a single OR (e.g., Denton and Gupta, 2003; Vanden Bosch and Dietz, 2000; Weiss, 1990), and surgery-to-OR allocation (e.g., Denton et al., 2010; Min and Yih, 2010). The former focuses on assigning time intervals between surgeries given a predefined sequence, to minimize surgery waiting, OR under-utilization, and OR overtime. The latter decides the number of ORs to open and surgery-to-OR assignment to minimize the total cost of operating ORs. We refer the readers to Cardoen et al. (2010); Gupta (2007) and Erdogan and Denton (2011) for comprehensive surveys of different models and solution approaches in surgery planning and scheduling.

Chance-Constrained Programming Shylo et al. (2012) formulate a chance-constrained program for allocating surgery blocks to ORs, with the goal of minimizing under-utilization of ORs subject to restricted risk of OR overtime. They reformulate the problem as an equivalent convex program by assuming that surgery durations follow independent normal distributions. To the best of our knowledge, it is the only paper in the literature that builds a chance-constrained model for stochastic OR allocation under surgery duration uncertainty. In this chapter, we allow generally-distributed surgery durations. We formulate surgery planning as a chance-constrained program that involves two stages of mixed-binary decisions. The model is large and structurally complex due to the integration of assignment and scheduling decisions. It is difficult to solve using the current methods, for which we develop a branch-and-cut algorithm that we show is very effective in our computational tests.

Distributionally Robust Optimization Kong et al. (2013) formulate a DR appointment scheduling model for minimizing the worst-case expected penalty cost of patient waiting and doctor overtime in an outpatient clinic. They build a cross-moment uncertainty set to include all probability distributions of the random service time with common support, mean vectors, and covariance matrices. They reformulate the problem as a copositive conic program and approximate it by a semidefinite program. Mak et al. (2015) consider a similar DR model to minimize the worst-case expected waiting and overtime, but build the uncertainty set using only marginal moments of the service time of individual appointments. They reformulate the problem as an equivalent semidefinite program and demonstrate the benefit of taking into account distribution ambiguity in appointment scheduling. The results of both papers can be applied to single-OR surgery scheduling, but not to the problem of surgery-to-OR allocation.

2.1.2 Motivation and Contributions

The motivations for our research are:

- **OR overtime and surgery waiting:** Hospitals in the United States lose approximately $\$300,000$ per year on average for each 1% increase of their nurse turnover rate (Hayes et al., 2006). The national nurse turnover rate reached 14.7% in 2013, for which the primary cause was the increase of nurses' overtime working hours in ORs (Nursing Solutions, Inc., 2013). Moreover, decreasing OR overtime simultaneously reduces the idle time in ORs and can lead to high utilization of surgery resources. Meanwhile, long surgery waiting is an important and non-negligible issue in surgery planning, which may become more severe given under the objective of compressing the OR idle time and overtime.

- **Constraining the risk of OR overtime:** It is difficult or impossible to accurately estimate the penalty associated with surgery waiting and OR overtime, since they affect multiple stake-holders, including patients, nurses, and surgeons. It is easy to estimate the per hour cost of overtime for members of the OR team; however, this does not account for the loss of "goodwill" due to the impact on family from a delayed shift. In our experience with several hospitals, reliable shift completion is important to staff morale and retention, and it is especially important in the case of specialized nurses due to a nationwide shortage of highly qualified nursing workforce. Moreover, patient safety is frequently tied to the frequency and length of overtime in ORs. Since these factors are difficult to model with a monetary cost, we consider a chance-constrained model to guarantee sufficiently low probability of overtime according to decision-makers' preference and priority.

- **Integrating allocation and scheduling decisions:** Few research studies of surgery planning have combined surgery-to-OR allocation with the problem of sequencing and scheduling surgeries. This is mainly due to the complexity of the integrated models and the difficulty of deriving efficient solution algorithms. In this chapter, we develop integrated models to show the benefits of simultaneously optimizing OR allocation and surgery scheduling.

- **Ambiguous distributions of surgery durations:** Previous studies of stochastic surgery planning assume that the distributional information of random surgery durations is known. When the distribution is ambiguous, traditional stochastic optimization methods could fail to provide surgery plans with high reliability. Thus, we employ DR optimization to plan surgeries against the worst-case distribution of surgery durations that can be derived from observed data or previous knowledge. We test a variety of instances with different sample sizes to demonstrate the performance of DR optimal solutions and the power of data-driven optimization in surgery planning.

To our best knowledge, we are the first to study chance-constrained surgery planning (CCSP) under generally distributed surgery durations. We build on the previous literature in stochastic

surgery planning by combining allocation and scheduling decisions within a single optimization model. We are also the first to consider CCSP in a DR context, which expands the research dimension of stochastic surgery planning, given that distributions of random surgery durations are often unknown in practice. Our results provide insights into *(i)* managing risk and the quality of service via the use of chance constraints, *(ii)* the importance of combining allocation and scheduling decisions, and *(iii)* using data for guiding surgery planning under distributional information ambiguity. From a methodological perspective, given the large-scale SAA reformulation, we develop a branch-and-cut algorithm that exploits the scheduling problem structure. While this algorithm is motivated by a surgery planning application, it can be applied to more general parallel machine scheduling problems.

The remainder of Chapter 2 is organized as follows. Section 2.2 presents the CCSP model and its SAA reformulation with enhanced big-M coefficients. Section 2.3 proposes strong valid inequalities and a specialized branch-and-cut algorithm. Section 2.4 presents the DR model and an equivalent reformulation that allows us to reapply the proposed algorithm. Section 2.5 presents numerical experiments to illustrate the computational efficiency of the proposed algorithms, and the benefits of the features in our models. Section 2.6 concludes the chapter.

2.2 Problem Formulation

Consider a set $\mathcal{S} = \{1, \ldots, S\}$ of surgeries to be scheduled in a set $\mathcal{R} = \{1, \ldots, R\}$ of ORs that are each available for a period of time, $T_{\max}$. Each OR $j \in \mathcal{R}$ has a fixed opening cost, c_j, and a standard operating time limit $T_j \in (0, T_{\max}]$. A random vector $\xi = (\xi_1, \ldots, \xi_S)^\top \in \mathbb{R}_+^S$ denotes surgery durations, where ξ_i is the duration time of surgery i for all $i \in \mathcal{S}$. Before realizing the value of ξ, we assign every surgery a start time and an open OR. Given uncertain durations, surgeries may not start punctually at the assigned time if the OR is still occupied. The goal is to minimize the total cost of opening ORs while requiring that waiting time for each surgery i must not exceed a maximum tolerable waiting time, ϵ_i, and with sufficiently high probability that there is no overtime operation in any OR. Note that the waiting time restriction differentiates our CCSP model from a general chance-constrained bin packing problem (see Song et al., 2014) where items (i.e., surgeries in this chapter) can be packed "seamlessly".

2.2.1 Chance-Constrained Model

Let $\Omega = \{\xi^\omega = (\xi_1^\omega, \ldots, \xi_S^\omega)^\top : \omega = 1, \ldots, |\Omega|\}$ be the set of scenarios, with equal probability, i.e., $p_\omega = 1/|\Omega|$, that have been extracted from a Monte Carlo sample. We can now define the decision variables and present the related constraints. First, allocation and scheduling decisions are made

before realizing surgery durations. We define two sets of binary first-stage decision variables $x \in \{0,1\}^R$ and $z \in \{0,1\}^{R\times S\times S}$ for which

- $x_j = 1$ if we open OR j, and $x_j = 0$ otherwise, for all $j \in \mathcal{R}$;
- $z_{ik}^j = 1$ if surgery i is assigned the k^{th} position in the sequence of surgeries in OR j, and $z_{ik}^j = 0$ otherwise, for all $j \in \mathcal{R}$, $i \in \mathcal{S}$, $k = 1,\ldots,S$. (For notational clarity, we use i to index surgery, and k to index the order of surgeries in each OR. Note that there are S positions in each OR at maximum, and thus $k = 1,\ldots,S$.)

The decisions x and z must satisfy

$$\sum_{j\in\mathcal{R}}\sum_{k=1}^{S} z_{ik}^j = 1 \quad \forall i \in \mathcal{S} \tag{2.1}$$

$$\sum_{i\in\mathcal{S}} z_{ik}^j \le x_j \quad \forall k = 1,\ldots,S,\ j \in \mathcal{R} \tag{2.2}$$

$$\sum_{i\in\mathcal{S}} z_{ik+1}^j \le \sum_{i\in\mathcal{S}} z_{ik}^j,\ \forall k = 1,\ldots,S, \tag{2.3}$$

where constraint (2.1) ensures that every surgery is assigned a position in an OR; constraint (2.2) assigns no more than one surgery to one position in each open OR, and do not assign surgeries to any position in a closed OR; and constraint (2.3) prohibits assigning surgeries to position $k+1$ when position k is still vacant, for every OR.

Next, we define a continuous first-stage decision variable $s_k^j \ge 0$ to represent the planned start time of the k^{th} surgery in every OR j, and a continuous second-stage decision variable $\delta_k^j(\xi) \ge 0$ dependent on surgery durations ξ to represent the actual time of finishing the k^{th} surgery in OR j. We abbreviate $\delta_k^j(\xi^\omega)$ as $\delta_k^{j\omega}$. The constraints defining the feasible sequences of surgeries are:

$$s_k^j - s_{k-1}^j \ge 0 \quad \forall k = 2,\ldots,S,\ j \in \mathcal{R} \tag{2.4}$$

$$\delta_k^{j\omega} \ge \delta_{k-1}^{j\omega} + \sum_{i\in\mathcal{S}} \xi_i^\omega z_{ik}^j \quad \forall k = 2,\ldots,S,\ j \in \mathcal{R},\ \omega \in \Omega \tag{2.5}$$

$$\delta_k^{j\omega} \ge s_k^j + \sum_{i\in\mathcal{S}} \xi_i^\omega z_{ik}^j \quad \forall k = 1,\ldots,S,\ j \in \mathcal{R},\ \omega \in \Omega \tag{2.6}$$

$$\delta_{k-1}^{j\omega} \le s_k^j + \sum_{i\in\mathcal{S}} \epsilon_i z_{ik}^j \quad \forall k = 2,\ldots,S,\ j \in \mathcal{R},\ \omega \in \Omega. \tag{2.7}$$

Constraint (2.4) forms a valid sequence to schedule surgeries assigned to OR j, for all $j \in \mathcal{R}$; constraint (2.5) prohibits starting the k^{th} surgery before finishing the $(k-1)^{\text{th}}$ surgery in each OR j; constraint (2.6) ensures that each surgery i does not start earlier than their planned start time, and will last for ξ_i^ω time in scenario ω; and constraint (2.7) applies to the waiting times of individual

surgeries reformulated from the individual chance constraints. It ensures that each surgery does not wait longer than the given waiting tolerance to start, based on the actual finishing time of their previous surgeries.

Consider a chance constraint that ensures at least β probability that every OR j completes all the assigned surgeries by time T_j, or equivalently, no overtime occurs in any ORs. Although the number of surgeries assigned to each OR j is, likely, smaller than S, we use $\delta_S^j(\xi)$ as the time that all the assigned surgeries are completed in OR j for all $j \in \mathcal{R}$. Assuming that the number of surgeries assigned to each OR j is $S(j)$, we verify that we can maintain the feasibility of a given solution without violating constraints (2.4)–(2.7), by letting $s_k^j = s_{S(j)}^j$ and $\delta_k^j(\xi) = \delta_{S(j)}^j(\xi)$ for $k = S(j)+1, \ldots, S$ and $j \in \mathcal{R}$. Thus, the chance constraint is given by

$$\mathbb{P}\left\{\delta_S^j(\xi) \le T_j,\ \forall j \in \mathcal{R}\right\} \ge \beta. \tag{2.8}$$

The whole CCSP model is given by

$$\min\left\{c^\top x : (2.1)\text{–}(2.8),\ x, z \text{ binary},\ s, \delta \ge 0\right\}. \tag{2.9}$$

2.2.2 Mixed-Integer Programming Reformulation

We define variable $\kappa_\omega \in \{0,1\}$ for each scenario ω such that $\kappa_\omega = 0$ implies that $\delta_S^{j\omega} \le T_j,\ \forall j \in \mathcal{R}$ in scenario ω. To attain the probability threshold, $\kappa = (\kappa_1, \ldots, \kappa_{|\Omega|})^\top$ must belong to the following β-parameterized set:

$$\Gamma(\beta) := \left\{\kappa \in \{0,1\}^{|\Omega|} : (1/|\Omega|)\sum_{\omega\in\Omega}(1-\kappa_\omega) \ge \beta\right\} = \left\{\kappa \in \{0,1\}^{|\Omega|} : \sum_{\omega\in\Omega}\kappa_\omega \le \lfloor \beta|\Omega|\rfloor\right\}.$$

With $\kappa \in \Gamma(\beta)$, the SAA reformulation of model (2.9) is

$$\begin{aligned}
\min\quad & c^\top x && \text{(2.10a)}\\
\text{s.t.}\quad & (2.1)\text{–}(2.7) && \text{(2.10b)}\\
& \delta_S^{j\omega} \le T_j + \kappa_\omega(M_{j\omega} - T_j), \quad \forall j \in \mathcal{R},\ \omega \in \Omega && \text{(2.10c)}\\
& x, z \text{ binary},\ s, \delta \ge 0,\ \kappa \in \Gamma(\beta) && \text{(2.10d)}
\end{aligned}$$

where $M_{j\omega}$ is sufficiently large that it deactivates inequality (2.10c) if $\kappa_\omega = 1$. We describe its value in the following. First, note that the first surgery at any OR always starts on time, and thus $\delta_1^{j\omega} = s_1^j + \sum_{i\in\mathcal{S}} \xi_i^\omega z_{i1}^j$ (with $s_1^j = 0$) for every $\omega \in \Omega$ and $j \in \mathcal{R}$. Any subsequent surgery starts either at its planned start time or at the time when the preceding surgery finishes. As a result, the

following recursion holds for any $k \geq 2$:

$$\delta_k^{j\omega} = \max\left\{s_k^j,\ \delta_{k-1}^{j\omega}\right\} + \sum_{i\in\mathcal{S}} \xi_i^\omega z_{ik}^j,\ \forall j \in \mathcal{R},\ \omega \in \Omega, \tag{2.11}$$

from which we derive the closed-form expressions:

$$\delta_k^{j\omega} = \max_{k'=1,\ldots,k}\left\{s_{k'}^j + \sum_{\ell=k'}^{k}\sum_{i\in\mathcal{S}} \xi_i^\omega z_{i\ell}^j\right\} \quad \forall k = 1,\ldots,S,\ j \in \mathcal{R},\ \omega \in \Omega. \tag{2.12}$$

To guarantee (2.10c), $M_{j\omega}$ must be at least

$$\begin{aligned}&\max\left\{\delta_S^{j\omega} : (x,z,s,\delta,\kappa) \text{ is feasible to model (2.10)}\right\}\\ =&\max_{k'=1,\ldots,S}\left\{s_{k'}^j + \sum_{\ell=k'}^{S}\sum_{i\in\mathcal{S}} \xi_i^\omega z_{i\ell}^j : (2.1)\text{–}(2.7), (2.8),\ x,z \text{ binary},\ s \geq 0\right\}.\end{aligned}$$

Meanwhile, given any threshold $\beta > 0$, we have $s_{k'}^j \leq T_j,\ \forall k' = 1,\ldots,S$, in any OR $j \in \mathcal{R}$, i.e., all surgeries start before the time limit T_j in every OR j. If not, overtime exists with probability one, and the corresponding solution will not satisfy the chance constraint (2.8). Thus, for each $j \in \mathcal{R}$ and $\omega \in \Omega$, we set $M_{j\omega}$ to $T_j + \sum_{i\in\mathcal{S}} \xi_i^\omega$, which is a valid upper bound of $\delta_k^{j\omega}$ for any $k = 1,\ldots,S$.

2.3 Decomposition-based Branch-and-Cut Method

Even though an off-the-shelf MIP solver can be used to solve the MIP problem (2.10), in Section 2.5 we show that directly solving it is very slow due to the presence of many binary variables and scenarios. Below, we develop a branch-and-cut algorithm that recovers single-OR-based and single-scenario-based subproblems that can be solved very efficiently. We also propose valid inequalities based on an analogous bin packing problem to pre-filter solutions before using preciser but more complex separation procedures.

2.3.1 Master Problem

In problem (2.10), if variables x, z, and κ are fixed, the remaining problem is OR-separable. According to this feature, we split this problem into two stages. At the first stage, consider a master problem with cross-OR decisions (x,z,κ):

$$\min_{x,z,\kappa}\left\{c^\top x : (2.1)\text{–}(2.3),\ (z,\kappa) \in Q,\ x,z \text{ binary},\ \kappa \in \Gamma(\beta)\right\}, \tag{2.13}$$

where $Q = \{(z, \kappa) : \exists\ s, \delta \geq 0$ that satisfy (2.4)–(2.7), (2.10c)$\}$. The constraints to form $(z, \kappa) \in Q$ are not explicitly presented; rather, we enforce $(z, \kappa) \in Q$ through a cut generation procedure (analogous to the integer L-shaped method reviewed in Section 1.1.3). Now that z and κ are binary, a natural approach to cut off an infeasible solution is to add the no-good cut (1.19). We also derive valid inequalities based on the scheduling structure of our problem. Section 2.3.2 demonstrates deriving valid inequalities upon a chance-constrained bin packing (CCBP) problem, which we show is a relaxation of the original CCSP problem. Section 2.3.4 discusses another set of valid inequalities built through a series of single-OR scheduling problems, which we further strengthen and develop a heuristic approach to identify in Section 2.3.4. We combine everything into a branch-and-cut framework and present other computational enhancements in Section 2.3.5.

2.3.2 Pre-filtering and Packing Cuts

We introduce binary variables y_{ji} for every surgery i and every OR j, to explicitly capture surgery-to-OR assignment, such that $y_{ji} = 1$ if surgery i is assigned to OR j, and $y_{ji} = 0$ otherwise. Consider a CCBP problem:

$$\min_{x,y \text{ binary}} \quad c^\top x \tag{2.14a}$$

$$\text{s.t.} \quad \mathbb{P}\left\{\sum_{i \in \mathcal{S}} \xi_i y_{ji} \leq T_j,\ \forall j \in \mathcal{R}\right\} \geq \beta \tag{2.14b}$$

$$\sum_{j \in \mathcal{R}} y_{ji} = 1 \quad \forall i \in \mathcal{S} \tag{2.14c}$$

$$y_{ji} \leq x_j \quad \forall i \in \mathcal{S},\ j \in \mathcal{R} \tag{2.14d}$$

Constraints (2.14c) and (2.14d) ensure allocating every surgery to an open OR, and it is clear that any feasible solution $y = (y_{ji} : j \in \mathcal{R}, i \in \mathcal{S})^\top$ subject to these two constraints, and any feasible solution z to the original problem (2.10) subject to constraints (2.2) and (2.3) must satisfy $y_{ji} = \sum_{k=1}^{S} z_{ik}^j,\ \forall i \in \mathcal{S},\ j \in \mathcal{R}$. See Appendix A.1 for a formal proof that the CCBP problem is a relaxation of the CCSP problem. The intuitive explanation is as follows. The operating time length of any OR j is equal to the total time duration of assigned surgeries plus possible idle time between adjacent surgeries. In (2.14b), however, we pack surgeries into ORs without the possibility of idle time. Therefore, for any feasible z to CCSP, letting $y_{ji} = \sum_{k=1}^{S} z_{ik}^j,\ \forall i \in \mathcal{S},\ j \in \mathcal{R}$ results in a feasible y to the CCBP problem. With $\kappa \in \Gamma(\beta)$, we can rewrite (2.14b) as

$$\sum_{i \in \mathcal{S}} \xi_i^\omega y_{ji} \leq T_j + (\dot{M}_{j\omega} - T_j)\kappa_\omega \tag{2.15}$$

where each $\dot{M}_{j\omega}$ is a big-M coefficient, and further with y_{ji} replaced by $\sum_{k=1}^{S} z_{ik}^{j}$, as

$$\sum_{i\in\mathcal{S}} \xi_i^{\omega} \left(\sum_{k=1}^{S} z_{ik}^{j} \right) \leq T_j + (\dot{M}_{j\omega} - T_j)\kappa_{\omega}. \tag{2.16}$$

It follows that (2.16) is a valid inequality with respect to the original problem (2.10). We use it as a cutting plane in the branch-and-cut algorithm described in Section 2.3.5 and refer to it as a *packing cut*.

The strength of such a cut is dependent on the value of $\dot{M}_{j\omega}$. Next, we discuss the choice of $\dot{M}_{j\omega}$ to balance between cut strength and computational tractability. First, note that $\dot{M}_{j\omega}$ must be at least

$$\max_{y,\kappa} \left\{ \sum_{i\in\mathcal{S}} \xi_i^{\omega} y_{ji} : (2.14c),\ (2.15),\ y \text{ binary},\ \kappa \in \Gamma(\beta) \right\}, \tag{2.17}$$

which is the maximum left-hand-side value of constraint (2.15) for any feasible CCBP solutions. A straightforward choice is to let $\dot{M}_{j\omega} = \sum_{i\in\mathcal{S}} \xi_i^{\omega}$. We describe two approaches for strengthening $\dot{M}_{j\omega}$ from the literature, and integrate the second one into our algorithm for generating the packing cuts.

- Following the approach by Qiu et al. (2014), we can initialize $\dot{M}_{j\omega} = \sum_{i\in\mathcal{S}} \xi_i^{\omega}$ and iteratively solve the linear programming (LP) relaxation of problem (2.17) with the most updated $\dot{M}_{j\omega}$-value used in constraint (2.15), to obtain a new value of $\dot{M}_{j\omega}$. Repeat the process until the attained $\dot{M}_{j\omega}$-values converge. Note that solving the linear programming relaxation of (2.17) can be time consuming, especially when R is large. Therefore, the second approach is preferable.

- We solve a relaxation of (2.17) by approximating the chance constraint (2.14b) (i.e., (2.15) and $\kappa \in \Gamma(\beta)$ in (2.17)) with $\mathbb{P}\left\{\sum_{i\in\mathcal{S}} \xi_i y_{ji} \leq T_j\right\} \geq \beta$, for all $j \in \mathcal{R}$. We then relax constraints (2.14c), and attain a set of individual-OR-based chance-constrained programs, for all $j \in \mathcal{R}$:

$$\max_{y_j} \left\{ \sum_{i\in\mathcal{S}} \xi_i^{\omega} y_{ji} : \mathbb{P}\left\{ \sum_{i\in\mathcal{S}} \xi_i y_{ji} \leq T_j \right\} \geq \beta,\ y_j \text{ binary} \right\}, \tag{2.18}$$

  where $y_j = (y_{j1}, \ldots, y_{jS})^{\mathsf{T}}$. Update the value of $\dot{M}_{j\omega}$ as the optimal objective value of model (2.18) for each corresponding $j \in \mathcal{R}$ and $\omega \in \Omega$.

  As a chance-constrained program, (2.18) may still be difficult to solve exactly, especially when $|\Omega|$ is large. We attain the value of $\dot{M}_{j\omega}$ as an upper bound of the optimal objective value of (2.18) by following a scenario sorting method (see Luedtke, 2014; Song et al., 2014).

Specifically, for each $j \in \mathcal{R}$ and $\omega \in \Omega$, consider a series of deterministic problems:

$$\dot{m}_{j\omega}(\omega') = \max_{y_j} \left\{ \sum_{i \in \mathcal{S}} \xi_i^{\omega} y_{ji} : \sum_{i \in \mathcal{S}} \xi_i^{w'} y_{ji} \le T_j,\ y_j \text{ binary} \right\} \quad \forall w' \in \Omega. \tag{2.19}$$

Sort $\dot{m}_{j\omega}(\omega'),\ \forall w' \in \Omega$ such that $\dot{m}_{j\omega}(\omega'_1) \le \cdots \le \dot{m}_{j\omega}(\omega'_{|\Omega|})$, and let $\dot{M}_{j\omega} = \dot{m}_{j\omega}(\omega'_{\theta+1})$ where $\theta = \lfloor \beta |\Omega| \rfloor$. Moreover, rather than solving the 0-1 single knapsack problems in (2.19), we compute their LP relaxations via the greedy algorithm, which still yields valid $\dot{M}_{j\omega}$, $\forall j \in \mathcal{R}$ and $\omega \in \Omega$.

Given a master-problem solution $(\hat{x}, \hat{z}, \hat{\kappa})$ that is not necessarily integral, if

$$\sum_{i \in \mathcal{S}} \xi_i^{\omega} \left(\sum_{k=1}^{S} \hat{z}_{ik}^{j} \right) > T_j$$

for any $j \in \mathcal{R}$ and $w \in \Omega_0(\hat{\boldsymbol{\kappa}}) = \{\omega \in \Omega : \hat{\kappa}_\omega = 0\}$, we generate a packing cut (2.16). We refer to this procedure as *pre-filtering*, and present the detailed steps in Algorithm 2.1. In particular, Steps 4–12 describe the greedy algorithm for optimizing the LP relaxations of (2.19) that are continuous 0-1 knapsack problems. Theoretically, this pre-filtering procedure can be extended to every scenario ω that currently has $\hat{\kappa}_\omega < 1$. In our experiments, however, we find this extension result in an overwhelmingly large number of cuts that lengthen total computation time.

Algorithm 2.1 GenCut_P($\hat{x}, \hat{z}, \hat{\kappa}$): a packing-cut generation subroutine

1: **for** $j \in \mathcal{R}(\hat{x})$, $\omega \in \Omega_0(\hat{\boldsymbol{\kappa}})$ **do**
2: **if** $\sum_{i \in \mathcal{S}} \xi_i^{\omega} (\sum_{k=1}^{S} \hat{z}_{ik}^{j}) > T_j$ **then**
3: **for** $\omega' \in \Omega$ **do**
4: sort $r(i) = \xi_i^{\omega} / \xi_i^{\omega'}, \forall i \in \mathcal{S}$ such that $r(i_1) \ge \cdots \ge r(i_S)$
5: $\ell \leftarrow 1$.
6: $v \leftarrow 0$, $b \leftarrow T_j$.
7: **while** $b > 0$ **do**
8: $q \leftarrow \min\{1, b/\xi_{i_\ell}^{\omega'}\}$.
9: $v \leftarrow v + q\xi_{i_\ell}^{\omega}$, $b \leftarrow b - q\xi_{i_\ell}^{\omega'}$.
10: $\ell \leftarrow \ell + 1$.
11: **end while**
12: $\dot{m}_{j\omega}(\omega') \leftarrow \lfloor v \rfloor$.
13: **end for**
14: $\dot{M}_{j\omega} \leftarrow$ the $(\theta+1)^{\text{th}}$ smallest number among $\dot{m}_{j\omega}(\omega'), \forall w' \in \Omega$.
15: exit and return a packing cut (2.16).
16: **end if**
17: **end for**
18: exit and return `null` (no cuts).

2.3.3 Basic Separation and Scheduling Cuts

Since packing cut (2.16) does not suffice for verifying a solution $(z,\kappa) \in Q$, we need a precise separation step as follows. For any integer (binary) solution $(\hat{x},\hat{z},\hat{\kappa})$ that passes the pre-filtering (i.e., Algorithm 2.1 returns `null`), we explicitly know

- The set $\mathcal{R}(\hat{x}) = \{j \in \mathcal{R} : \hat{x}_j = 1\}$ of the ORs that are open.
- The number of surgeries assigned to each open OR j: $A_j(\hat{z}^j) = \sum_{k=1}^{S} \sum_{i\in\mathcal{S}} \hat{z}^j_{ik}$.
- The k^{th} surgery in each open OR j: $i_k(\hat{z}^j) = i^*$ such that $\hat{z}^j_{i^*k} = 1$, for $k = 1,\ldots,A_j(\hat{z}^j)$.
- The set $\Omega_0(\hat{\kappa}) = \{\omega \in \Omega : \hat{\kappa}_\omega = 0\}$ with no overtime.

First, note that it suffices to consider ORs in $\mathcal{R}(\hat{x})$. Plugging in the closed-form expression of δ-solutions in (2.12), we have a linear separation problem for each OR $j \in \mathcal{R}(\hat{x})$, with a feasibility region defined as:

$$
\begin{aligned}
\mathbf{SP}_j(\hat{z}^j,\hat{\boldsymbol{\kappa}}) := \ \Big\{ & s_{k+1} - s_k \geq 0,\ \forall k = 1,\ldots,A_j - 1 \\
& s_{k'} + \sum_{\ell=k'}^{k-1} \xi^{\omega}_{i_\ell} - s_k \leq \epsilon_{i_k},\ \forall k' = 1,\ldots,k-1,\ k = 2,\ldots,A_j,\ \omega \in \Omega \qquad (2.20) \\
& s_{k'} + \sum_{\ell=k'}^{A_j} \xi^{\omega}_{i_\ell} \leq T_j,\ \forall k' = 1,\ldots,A_j,\ \omega \in \Omega_0(\hat{\kappa}) \qquad (2.21) \\
& s_k \geq 0,\ \forall k = 1,\ldots,A_j \Big\}.
\end{aligned}
$$

In the above, we simplify the notation $A_j(\hat{z}^j)$, $i_k(\hat{z}^j)$ as A_j, i_k, respectively. Constraint (2.20) imposes the maximum waiting-time tolerance in all the scenarios in Ω, while constraint (2.21) ensures zero overtime only for scenarios in $\Omega_0(\hat{\kappa})$. If $\mathrm{SP}_j(\hat{z}^j,\hat{\boldsymbol{\kappa}})$ is infeasible, we generate a no-good cut

$$
\sum_{k=1}^{A_j} \left(1 - z^j_{i_k k}\right) + \sum_{\omega\in\Omega_0(\hat{\kappa})} \kappa_\omega \geq 1, \qquad (2.22)
$$

which prohibits allocating the current sequence of surgeries to OR j in the future, if we continue to have $\kappa_\omega = 0$ for all $\omega \in \Omega_0(\hat{\kappa})$. Noting that such a cut excludes one specific surgery sequence coupled with one specific combination of scenarios at a time, it could be rather inefficient considering the enormous number of possibilities. Thus, we propose another separation approach which generates stronger cuts, each of which excludes a much shorter surgery sequence coupled with one single scenario. This separation approach also avoids solving an optimization problem by using recursive arithmetic.

2.3.4 Recursion-based Separation and Strengthened Scheduling Cuts

We use a to represent a sequence (or a prefix of a sequence) of surgeries. We let $a_k \in \mathcal{S}$ denote the k^{th} element (surgery) in sequence a, and let $a_{1:k}$ be the k-sized prefix of sequence a (which is also a sequence of surgeries). Let $\pi_\omega(a)$ represent the time in each scenario ω that the last surgery in sequence a finishes. Note that this time value is not dependent on the OR, to which the sequence is assigned. A recursive procedure for computing $\pi_\omega(a)$ is as follows.

Given an integer solution $(\hat{x}, \hat{z}, \hat{\kappa})$ of the master problem (2.13) that passes pre-filtering in Algorithm 2.1, we can recover the sequence a^j of surgeries at each open OR j with its k^{th} element denoted by a^j_k. In particular, for any $k \in \{1, \ldots, A_j\}$,

$$a^j_{1:k} = (i_1(\hat{z}^j), \ldots, i_k(\hat{z}^j)).$$

By definition, $\pi_\omega(a^j_{1:k}) = \delta^{j\omega}_k$. It is reasonable to assume that at least one of the waiting-time constraints in (2.7) attains equality. Incorporating the recursive expressions of δ-variables in (2.11), we compute $\pi_\omega(a^j_{1:1}), \ldots, \pi_\omega(a^j_{1:A_j})$ recursively based on the incumbent solution $\pi_\omega(a^j_{1:1}) = \xi^\omega_{a^j_1}$, and

$$\pi_\omega(a^j_{1:k}) = \max\left\{s^j_k,\ \pi_\omega(a^j_{1:k-1})\right\} + \xi^\omega_{a^j_k},\ \text{for } k = 2, \ldots, A_j, \text{where } s^j_k = \max_{\omega \in \Omega}\left\{\pi_\omega(a^j_{1:k-1})\right\} - \epsilon_{a^j_k}. \quad (2.23)$$

Our separation procedure iteratively computes $\pi_\omega(a^j_{1:k}),\ \forall \omega \in \Omega$ and compares their values to T_j, with a goal to identify a minimal prefix of a^j that cannot be finished within time T_j in some scenario $w \in \Omega$. The output of the method is a set $\mathcal{E}$ of three-tuples (a, ω, j) which suggests that if a scenario ω has $\kappa_\omega = 0$, we should avoid assigning surgery sequence a (or any sequence that contains a as a prefix) to OR j.

For some open OR j, suppose we compute $\pi_\omega(a^j_{1:k}),\ \forall \omega \in \Omega$ for some $k \in \{1, \ldots, A_j\}$. Let τ be a mapping that maps the set $\{\pi_\omega(a^j_{1:k}) : \forall \omega \in \Omega\}$ to a sequence of $\theta + 1$ scenarios, such that the corresponding $\pi_\omega(a^j_{1:k})$'s are the $\theta + 1$ largest values among $\{\pi_\omega(a^j_{1:k}) : \omega \in \Omega\}$ and follow a non-increasing order. We abbreviate the sequence $\tau(\{\pi_\omega(a^j_{1:k}) : \omega \in \Omega\})$ as τ, and denote the n^{th} element in a sequence τ as τ_n. By definition,

$$\pi_{\tau_1}(a^j_{1:k}) \geq \cdots \geq \pi_{\tau_{\theta+1}}(a^j_{1:k}) \geq \pi_\omega(a^j_{1:k}),\ \forall \omega \text{ not contained in sequence } \tau.$$

If $\pi_{\tau_1}(a^j_{1:k}) \leq T_j$, it implies that the current surgery sequence $a^j_{1:k}$ can be completed within the time limit T_j for all scenarios in Ω. Thus, we can compute $\pi_\omega(a^j_{1:k+1}),\ \forall \omega \in \Omega$ to check a longer sequence $a^j_{1:k+1}$. Otherwise, we compare $\pi_{\tau_{\theta+1}}(a^j_{1:k})$ and T_j. If $\pi_{\tau_{\theta+1}}(a^j_{1:k}) > T_j$, since it is impossible to find $|\Omega| - \theta$ scenarios in which we can finish the sequence $a^j_{1:k}$ of surgeries within T_j, we avoid this sequence in the future iterations. This essentially adds $|\Omega|$ three-tuples $(a^j_{1:k}, \omega, j),\ \forall \omega \in \Omega$ to

$\mathcal{E}$. If $\pi_{\tau_{\theta+1}}(a^j_{1:k}) \le T_j$, we iterate through $(\tau_2, \ldots, \tau_\theta)$ to search for some scenario $\tau_n \in \Omega_0(\hat{\boldsymbol{\kappa}})$ that has $\pi_{\tau_n}(a^j_{1:k}) > T_j$. If found, add tuple $(a^j_{1:k}, \tau_n, j)$ to $\mathcal{E}$.

After the k^{th} iteration, if $\mathcal{E} \neq \emptyset$ we do not proceed to the $(k+1)^{\text{th}}$ iteration, but generate cuts to exclude the tuples in $\mathcal{E}$ and return to the first stage to re-compute the master problem (2.13). After finishing the A_j iterations for some OR j, we reset $k = 1$ and perform the same iterative steps for the next open OR. After verifying all the open ORs, an empty set $\mathcal{E}$ indicates that $(\hat{z}, \hat{\boldsymbol{\kappa}}) \in Q$, or equivalently, $(\hat{x}, \hat{z}, \hat{\boldsymbol{\kappa}})$ is feasible to the CCSP model.

For each tuple $(a, \omega, j) \in \mathcal{E}$, let $|a|$ be the number of surgeries in sequence a. We generate a feasibility cut

$$\sum_{k=1}^{|a|} \left(1 - z^j_{a_k k}\right) + \kappa_\omega \ge 1, \tag{2.24}$$

which is a strengthened variant of the no-good cut (2.22) in Section 2.3.3. Furthermore, note that the values of $\pi_\omega(a_{1:\ell})$, $\forall \omega \in \Omega$ have been readily computed for any prefix $a_{1:\ell}$ of a. Therefore, we derive an auxiliary cut that exploits those values; specifically, consider the following conditional inequalities:

$$z^j_{a_k k} = 1,\ \forall k = 1, \ldots, \ell \quad \Rightarrow \quad \pi_\omega(a_{1:\ell}) \le T_j + (\ddot{M}^{j\omega} - T_j)\kappa_\omega \quad \forall \ell = 1, \ldots, |a|, \tag{2.25}$$

where $\ddot{M}^{j\omega}$ is a sufficiently large scalar that will deactivate the above inequality when $\kappa_\omega = 1$. Constraint (2.25) infers adopting surgery sequence $a_{1:\ell}$ (or a surgery sequence that contains $a_{1:\ell}$ as a prefix) in OR j only if $\pi_\omega(a_{1:\ell}) \le T_j$ for every $\omega \in \Omega_0(\kappa)$. Clearly, any feasible solution (x, z, κ) to the CCSP model must satisfy these conditions. Therefore, we mix the $|a|$ inequalities in (2.25) and attain a mixing cut (see Günlük and Pochet, 2001):

$$\sum_{\ell=1}^{|a|} (\pi_\omega(a_{1:\ell}) - \pi_\omega(a_{1:\ell-1})) z^j_{a_\ell \ell} \le T_j + (\ddot{M}^{j\omega} - T_j)\kappa_\omega, \tag{2.26}$$

where we set $\pi_\omega(a_{1:0}) = 0$. Similar to Section 2.3.2, to compute valid $\ddot{M}^{j\omega}$ for each $j \in \mathcal{R}$ and $\omega \in \Omega$, we optimize the LP relaxations of a set of 0-1 single knapsack problems:

$$\ddot{m}^{j\omega}(\omega') = \max \left\{ \sum_{\ell=1}^{|a|} (\pi_\omega(a_{1:\ell}) - \pi_\omega(a_{1:\ell-1})) z^j_{a_\ell \ell} : \sum_{\ell=1}^{|a|} (\pi_{\omega'}(a_{1:\ell}) - \pi_{\omega'}(a_{1:\ell-1})) z^j_{a_\ell \ell} \le T_j,\ 0 \le z \le 1 \right\},\ w' \in \Omega$$

via the greedy algorithm. We set $\ddot{M}^{j\omega}$ to the $(\theta+1)^{\text{th}}$ smallest number of $\ddot{m}^{j\omega}(\omega')$, $\forall w' \in \Omega$.

We refer to cuts (2.24) and (2.26) as *scheduling cuts*, to distinguish from the packing cuts (2.16) in Section 2.3.2. We summarize the recursion-based separation approach in Algorithm 2.2, which

takes a solution $(\hat{x}, \hat{z}, \hat{\kappa})$ of the master problem (2.13) as an input, and returns a set $\mathcal{C}$ of scheduling cuts. An empty set $\mathcal{C}$ implies that the solution $(\hat{x}, \hat{z}, \hat{\kappa})$ is feasible to the CCSP model.

Algorithm 2.2 GenCut_S($\hat{x}, \hat{z}, \hat{\kappa}$): a scheduling-cut generation subroutine

$\mathcal{E} \leftarrow \emptyset$.
for every open OR $j \in \mathcal{R}(\hat{x})$ **do**
  for $k = 1, \ldots, A_j$ **do**
    compute $\delta_k^{j\omega}$ (or equivalently, $\pi_\omega(a^j_{1:k})$) for every $\omega \in \Omega$, according to (2.23).
    $\tau \leftarrow \tau(\{\delta_k^{j\omega} : \omega \in \Omega\})$.
    if $T_j \in (-\infty, \delta_k^{j\tau_{\theta+1}})$ **then**
      $\mathcal{E} \leftarrow \{(a^j_{1:k}, \omega, j) : \omega \in \Omega\}$, and go to Step 20.
    end if
    if $T_j \in [\delta_k^{j\tau_{\theta+1}}, \delta_k^{j\tau_1})$ **then**
      $n \leftarrow 2$.
      while $\delta_k^{j\tau_n} > T_j$ **do**
        if $\tau_n \in \Omega_0(\hat{\kappa})$ **then**
          $\mathcal{E} \leftarrow \{(a^j_{1:k}, \tau_n, j) : \omega \in \Omega\}$, and go to Step 20.
        end if
        $n \leftarrow n + 1$.
      end while
    end if
  end for
end for
$\mathcal{C} \leftarrow \emptyset$.
for $(a, \omega, j) \in \mathcal{E}$ **do**
  add cuts (2.24) and (2.26) to $\mathcal{C}$.
end for
return $\mathcal{C}$.

2.3.5 Branch-and-Cut Algorithm and Computational Enhancements

We consolidate the foregoing pre-filtering and recursion-based separation procedures in a branch-and-cut framework. At each branching node ν, we solve a linear program as:

$$
\begin{aligned}
\mathbf{NR}(\nu): \quad \min \quad & c^\top x \\
\text{s.t.} \quad & (2.1)\text{–}(2.3),\ 0 \le x \le 1,\ 0 \le z \le 1 \\
& \sum_{\omega \in \Omega} \kappa_\omega \le \theta,\ 0 \le \kappa \le 1 && \text{(2.27a)} \\
& \alpha^\top z + \sigma^\top \kappa \ge \gamma,\ \forall (\alpha, \sigma, \gamma) \in \mathcal{L} && \text{(2.27b)} \\
& x_j = 0, \forall j \in F_x^0(\nu),\ x_j = 1, \forall j \in F_x^1(\nu) && \text{(2.27c)}
\end{aligned}
$$

$$z^j_{ik} = 0, \forall (j,i,k) \in F^0_z(\nu),\ z^j_{ik} = 1, \forall (j,i,k) \in F^1_z(\nu) \tag{2.27d}$$

$$\kappa_\omega = 0, \forall \omega \in F^0_\kappa(\nu),\ \kappa_\omega = 1, \forall \omega \in F^1_\kappa(\nu). \tag{2.27e}$$

Constraint (2.27a) represents the linear relaxation of $\Gamma(\beta)$; set $\mathcal{L}$ collects the generated packing and scheduling cuts that aim at enforcing $(z,\kappa) \in Q$; and constraints (2.27c)–(2.27e) specify the values of x, z and κ at node ν. Specifically, $F^0_x(\nu)$ and $F^1_x(\nu)$ are disjoint subsets of $\mathcal{R}$, $F^0_z(\nu)$ and $F^1_z(\nu)$ are disjoint subsets of $\mathcal{R}\times\mathcal{S}\times\mathcal{S}$, and $F^0_\kappa(\nu)$ and $F^1_\kappa(\nu)$ are disjoint subsets of Ω.

If $\mathbf{NR}(\nu)$ is infeasible or it attains an optimal objective value that is no better than an incumbent upper bound $\overline{\text{obj}}$, we fathom node ν. Otherwise, let $(\hat{x},\hat{z},\hat{\kappa})$ denote the current solution, and pass it to Algorithm 2.1, and then pass it to Algorithm 2.2 if it is not pre-filtered. Recompute model $\mathbf{NR}(\nu)$ whenever a packing cut or a scheduling cut is generated into the set Q' of cuts in (2.27b). This is computationally more efficient than recomputing after collecting all possible cuts. If no cuts are derived from subroutines in Algorithms 2.1 and 2.2, then $(\hat{x},\hat{z},\hat{\kappa})$ is feasible. We use its associated objective value to update the upper bound $\overline{\text{obj}}$, and proceed to other branching nodes that have not been fathomed. Algorithm 2.3 explains the steps of the branch-and-cut approach.

We implement some computational enhancements to Algorithm 2.3 as follows. First, we order ORs $j = 1,\ldots,R$ such that $T_1 \geq \cdots \geq T_R$. Therefore, both the packing cut (2.16) and the scheduling cuts (2.24) and (2.26), derived based on a particular OR j, are also valid for any $j' = j+1,\ldots,R$. We therefore propagate each cut to every applicable OR, which tends to strengthen the model $\mathbf{NR}(\nu)$ at each branching node ν. Moreover, we store the values of $\pi_\omega(a),\ \forall \omega \in \Omega$ for all distinct surgery sequences a, so that if we encounter the same sequence again, we do not have to re-compute the result. This avoids repeatedly computing $\pi_\omega(a),\ \forall \omega \in \Omega$ for some common prefix a choices.

2.4 Distributionally Robust Variant

We consider a DR variant of the CCSP model, for the case when the distribution of ξ is unknown, but only N_{obs} data samples, $\{\hat{\xi}^n\}_{n=1}^{N_{\text{obs}}}$, are known. We formulate distributionally robust chance-constrained surgery planning (DR-CCSP) to ensure the worst-case (i.e., maximum) probability of having OR overtime being no more than $1-\beta$ for any distribution of ξ that can be deduced from data. This section explains *(i)* how to derive a discrete support and an empirical probability mass function (PMF) of ξ; *(ii)* how to construct an uncertainty set for the ambiguous distribution of ξ using the support and the empirical PMF derived in Step *(i)*; and *(iii)* how to solve the DR model based on the uncertainty set constructed in Step *(ii)*. We use the ϕ-divergence measures in statistics (recently used in the stochastic optimization literature such as Ben-Tal et al., 2013; Jiang and Guan, 2015) to build the uncertainty set.

Algorithm 2.3 A branch-and-cut procedure for algorithm for solving model (2.10)

$\overline{\text{obj}} \leftarrow \infty$.
Initialize NodeList $\leftarrow \{\nu_0\}$, where $F_x^n(\nu_0) = \emptyset,\ F_z^n(\nu_0) = \emptyset,\ F_\kappa^n(\nu_0) = \emptyset$ for $n = 0$ and 1.
choose a branching node $\nu \in$ NodeList.
solve $\mathbf{NR}(\nu)$.
if attain $(\hat{x}, \hat{z}, \hat{\kappa})$ with $c^\top \hat{x} < \overline{\text{obj}}$ **then**
    `cut` $\leftarrow$ GenCut_P$(\hat{x}, \hat{z}, \hat{\kappa})$.
    if `cut` $\neq$ `null` **then**
        add `cut` to $\mathcal{L}$, and go to Step 4.
    end if
    if $(\hat{x}, \hat{z}, \hat{\kappa})$ is integral **then**
        $\mathcal{C} \leftarrow$ GenCut_S$(\hat{x}, \hat{z}, \hat{\kappa})$.
        if $\mathcal{C} \neq \emptyset$ **then**
            add `cut` to $\mathcal{L}$ for every `cut` in $\mathcal{C}$, then go to Step 4.
        else
            $\overline{\text{obj}} \leftarrow c^\top \hat{x}$.
        end if
    else
        branch on a binary variable having a fractional solution value and add two new branching nodes to NodeList.
    end if
end if
NodeList $\leftarrow$ NodeList$\backslash\{\nu\}$.
if NodeList is empty **then**
    report $\overline{\text{obj}}$ as the optimal objective value.
else
    go to Step 3.
end if

2.4.1 Discrete Support and Empirical Distribution of ξ

Although the random vector ξ resides in a continuous subspace of $\mathbb{R}_+^S$, we consider a finite and discrete support Ξ by partitioning the outcome space of ξ into K subregions, denoted by $\Xi = \{\mathcal{B}_1, \ldots, \mathcal{B}_K\}$. The true PMF $f : \Xi \to \mathbb{R}_+$ that provides the chance of ξ landing in each subregion is ambiguous. We use an empirical PMF $f_0 : \Xi \to \mathbb{R}_+$ associated with the observations $\{\hat{\xi}^n\}_{n=1}^{N_{\text{obs}}}$ as an estimation, calculated by

$$f_0(\mathcal{B}_k) = \frac{\sum_{n=1}^{N_{\text{obs}}} \mathbf{1}\left(\hat{\xi}^n \in \mathcal{B}_k\right)}{N}, \tag{2.28}$$

where $\mathbf{1}(\cdot)$ is an indicator function that returns value 1 if $\cdot$ is true and 0 otherwise. When the durations of surgeries in $\mathcal{S}$ are mutually independent, construct Ξ and f_0 from their marginal counterparts as follows. For every surgery i, we let $[\underline{\xi}_i, \overline{\xi}_i]$ be the range of the random duration ξ_i. We

partition $[\underline{\xi}_i, \overline{\xi}_i]$ into intervals $\mathcal{B}^i_1, \ldots, \mathcal{B}^i_{K_i}$, resulting in a finite and discrete support $\Xi_i = \{\mathcal{B}^i_k\}_{k=1}^{K_i}$ for ξ_i. For each $i \in \mathcal{S}$, let f^i and $f^i_0 : \Xi_i \to \mathbb{R}_+$ be the true and the empirical marginal PMFs, respectively. We calculate f^i_0 of each random $\xi_i,\ i \in \mathcal{S}$ as

$$f^i_0(\mathcal{B}^i_{k_i}) = \frac{\sum_{n=1}^{N_{\text{obs}}} \mathbf{1}\left(\hat{\xi}^n_i \in \mathcal{B}^i_{k_i}\right)}{N},\ k_i = 1, \ldots, K_i \tag{2.29}$$

Then, the support of ξ is the Cartesian product of the supports of $\xi_i,\ i \in \mathcal{S}$. That is, $\Xi = \{\mathcal{B}_1, \ldots, \mathcal{B}_K\} = \bigtimes_{i \in S} \Xi_i$, where $K = \prod_{i \in \mathcal{S}} K_i$ and each subregion $\mathcal{B}_k$ is given by an S-tuple $(\mathcal{B}^1_{k_1}, \ldots, \mathcal{B}^S_{k_S})$ that chooses one $\mathcal{B}^i_{k_i}$ from the support Ξ_i for each $i \in \mathcal{S}$. The empirical PMF f_0 is the tensor product of the marginal PMFs, and thus for any $\mathcal{B}_k \in \Xi$, we obtain

$$f_0(\mathcal{B}_k) = \prod_{i \in \mathcal{S}} \left(\frac{\sum_{n=1}^{N_{\text{obs}}} \mathbf{1}\left(\hat{\xi}^n_i \in \mathcal{B}^i_{k_i}\right)}{N} \right). \tag{2.30}$$

The above procedures discretize the support of a continuous random vector to derive an approximate PMF. Using this discrete support, we employ ϕ-divergence measures to bound the distance between the true and empirical distribution functions of ξ, based on the results discussed in Ben-Tal et al. (2013). The authors show the bounds on the divergence distance based on the choice of ϕ-functions, allowed maximum distance, size of data, dimension of decisions, and a specified confidence level.

2.4.2 Reformulating with ϕ-Divergence Confidence Set

We formulate the DR-CCSP model by replacing the chance constraint (2.8) in the CCSP model with

$$\inf_{f \in \mathcal{F}} \mathbb{P}\left\{ \delta^j_S(\xi) \le T_j,\ \forall j \in \mathcal{R} \right\} \ge \beta,$$

where $\mathcal{F}$ represents an uncertainty set for the unknown PMF f of ξ. Following Jiang and Guan (2015), we reformulate such a DR chance constraint into a regular chance constraint if using a class of uncertainty sets that are constructed based on statistical ϕ-divergence measures. The results for DR-CCSP are specified as follows.

Given a function $\phi : \mathbb{R} \to \mathbb{R}$ that is convex over $\mathbb{R}_+$, and satisfies

(P1) $\phi(1) = 0$;

(P2) $0\phi(a/0) = a \lim_{t \to \infty} \phi(t)/t$ for any $a > 0$, and $= 0$ if $a = 0$;

(P3) $\phi(a) = +\infty$ for $a < 0$,

measure the distance between the true and empirical PMFs, f and f_0, respectively, by ϕ-divergence:

$$\mathcal{D}_\phi(f\|f_0) = \sum_{k=1,\ldots,K} \phi\left(\frac{f(\mathcal{B}_k)}{f_0(\mathcal{B}_k)}\right) f_0(\mathcal{B}_k).$$

Examples of ϕ-divergence are: Kullback-Leibler divergence with $\phi_{\mathrm{KL}}(t) = t\log t - t + 1$; Hellinger divergence with $\phi_{\mathrm{H}}(t) = (\sqrt{t}-1)^2$; modified χ^2-distance divergence with $\phi_{\chi^2}(t) = (t-1)^2$; and variation distance divergence with $\phi_v(t) = |t-1|$. See Pardo (2005) for the mathematical forms of more ϕ-divergence measures.

The reformulation of DR chance constraint is based on a class of uncertainty sets $\mathcal{F}_\phi(d)$, $\forall d \geq 0$. Each $\mathcal{F}_\phi(d)$ consists of distribution functions having a ϕ-divergence distance to f_0 no more than the given parameter d, i.e.,

$$\mathcal{F}_\phi(d) = \left\{ f : \mathcal{D}_\phi(f\|f_0) \leq d,\ \sum_{k=1,\ldots,K} f(\mathcal{B}_k) = 1,\ f(\mathcal{B}_k) \geq 0,\ \forall k = 1,\ldots,K \right\}.$$

The reformulation requires a mild assumption that $\lim_{t\to+\infty} \phi(t)/t = +\infty$, which is satisfied by many ϕ-functions that obey (P1)–(P3).

Theorem 2.1. *(Jiang and Guan, 2015) For any constant $d \geq 0$, the following DR chance constraint*

$$\inf_{f\in\mathcal{F}_\phi(d)} \mathbb{P}\left\{\delta_S^j(\xi) \leq T_j,\ \forall j \in \mathcal{R}\right\} \geq \beta \tag{2.31}$$

is equivalent to

$$\mathbb{P}_0\left\{\delta_S^j(\xi) \leq T_j,\ \forall j \in \mathcal{R}\right\} \geq \min\{1, \mathcal{V}_\phi(\beta, d)\} \tag{2.32}$$

$$\text{where } \mathcal{V}_\phi(\beta,d) = \min_{\rho>0,\lambda\in\mathbb{R}} \left\{ \frac{\beta + \rho d - \lambda + \rho\phi^*\left(\frac{\lambda}{\rho}\right)}{\rho\left(\phi^*\left(\frac{\lambda}{\rho}\right) - \phi^*\left(\frac{\lambda-1}{\rho}\right)\right)} \right\},$$

where $\mathbb{P}_0$ is the probability measure defined by the empirical PMF f_0, and function $\phi^* : \mathbb{R} \to \mathbb{R}\cup\{+\infty\}$ is the conjugate of ϕ defined as $\phi^*(t) = \sup_{l\in\mathbb{R}}\{tl - \phi(l)\}$.

Replacing (2.8) with (2.32), obtain the DR-CCSP model

$$\min\left\{c^\mathsf{T} x : (2.1)\text{–}(2.7),\ x, z \text{ binary},\ s, \delta \geq 0,\ (2.32)\right\}, \tag{2.33}$$

which remains a regular chance-constrained program, to which the algorithms discussed in Section 2.3 can be applied.

2.4.3 Confidence Set Configuration

When constructing an uncertainty set $\mathcal{F}_\phi(d)$, decision makers can choose the value of d according to their risk-aversion preference. In general, it is desirable to keep d smaller to yield a tighter uncertainty set $\mathcal{F}_\phi(d)$. However, when there are few observations (i.e., small N_{obs}) or the distribution function is complex (e.g., large K), d must be increased to maintain the confidence level. Specifically, Ben-Tal et al. (2013) propose to use

$$d_{\text{J}} = \frac{\phi^{''}(1)}{2N_{\text{obs}}}\chi^2_{m-1,1-\alpha}, \tag{2.34}$$

based on the statistical inference results given by Pardo (2005), to *asymptotically* guarantee that set $\mathcal{F}_\phi(d_{\text{J}})$ contains the true PMF at $(1-\alpha)$ confidence when N_{obs} goes to infinity. Here, $\phi^{''}(1)$ is the second derivative of function ϕ evaluated at 1; m is the cardinality of the support of the random vector; and $\chi^2_{m-1,1-\alpha}$ is the $1-\alpha$ percentile of the χ^2_{m-1}-distribution, i.e., $\mathbb{P}(X \geq \chi^2_{m-1,1-\alpha}) = \alpha$, of which X follows a χ^2_{m-1}-distribution. In our problem, note that $m = K = \prod_{i\in\mathcal{S}} K_i$ in our problem, which could be extremely large dependent on the number of surgeries and the size K_i of the discrete support for every ξ_i. This could lead to a very large d_{J} and subsequently a very conservative uncertainty set $\mathcal{F}_\phi(d_{\text{J}})$, even for small- or medium-sized observation samples.

Recalling that we consider independent distributions of surgery durations $\xi_i,\ i \in \mathcal{S}$, we use a marginal counterpart of d_{J} denoted as

$$d_{\text{M}} = \sum_{i\in\mathcal{S}} d_i,$$

where each d_i is derived according to (2.34) for bounding the ϕ-divergence distance of the marginal PMFs f^i and f^i_0, such that $d_i = \frac{\phi''(1)}{2N_{\text{obs}}}\chi^2_{K_i-1,1-\alpha}$. Comparing $\sum_{i\in\mathcal{S}}\chi^2_{K_i-1,1-\alpha}$ with $\chi^2_{K-1,1-\alpha}$, the value of d_{M} is usually much smaller than d_{J}, yielding a less conservative DR-CCSP model. To justify the calculation of d_{M}, we consider the set, for each $i \in \mathcal{S}$:

$$\widetilde{\mathcal{F}}^i_\phi = \left\{ f^i : \mathcal{D}_\phi(f^i \| f^i_0) \leq d_i,\ \sum_{k_i=1,\ldots,K_i} f^i(\mathcal{B}^i_{k_i}) = 1,\ f^i(\mathcal{B}^i_{k_i}) \geq 0,\ \forall k_i = 1,\ldots,K_i \right\},$$

which is an asymptotically $1-\alpha$-confidence uncertainty set of the unknown f^i. Given the independence of $\xi_i,\ i \in \mathcal{S}$ and letting $\otimes$ denote the tensor product, we join the sets $\widetilde{\mathcal{F}}^i_\phi$ of all $i \in \mathcal{S}$ and attain an uncertainty set of the unknown PMF f as

$$\widetilde{\mathcal{F}}_\phi = \left\{ f = f^1 \otimes \cdots \otimes f^S : f^i \in \widetilde{\mathcal{F}}^i_\phi \right\}, \tag{2.35}$$

which is tighter than $\mathcal{F}_\phi(d_{\text{J}})$ where d_{J} is given by (2.34). Note that $\mathcal{F}_\phi(d_{\text{M}})$ trades off between

$\mathcal{F}_\phi(d_{\mathrm{J}})$ and $\widetilde{\mathcal{F}}_\phi$. In particular, for Hellinger and Kullback-Leibler (KL) divergence (respectively denoted by ϕ_{H} and ϕ_{KL} below), we have $\mathcal{F}_\phi(d_{\mathrm{M}}) \supseteq \widetilde{\mathcal{F}}_\phi$, because a joint Hellinger (or KL) distance is bounded above by (or equal to) its marginal counterpart (see Pollard, 1997), i.e.,

$$\mathcal{D}_{\phi_{\mathrm{H}}}\left(f^1 \otimes \cdots \otimes f^S \| f_0^1 \otimes \cdots \otimes f_0^S\right) \leq \sum_{i \in \mathcal{S}} \mathcal{D}_{\phi_{\mathrm{H}}}\left(f^i \| f_0^i\right),$$
$$\mathcal{D}_{\phi_{\mathrm{KL}}}\left(f^1 \otimes \cdots \otimes f^S \| f_0^1 \otimes \cdots \otimes f_0^S\right) = \sum_{i \in \mathcal{S}} \mathcal{D}_{\phi_{\mathrm{KL}}}\left(f^i \| f_0^i\right).$$

2.5 Computational Studies

In this section, we conduct computational studies for demonstrating our models and solution approaches. We derive insights about stochastic surgery planning under distributional ambiguity of random parameters, via testing instances sampled from probability density functions fit to real data from an Ambulatory Surgery Center (ASC). We describe the data and our experimental design in Section 2.5.1. We demonstrate the computational efficacy of our approaches for the CCSP model in Section 2.5.2. We compare the performance of solutions from CCSP versus from cost-based models (Section 2.5.3), integrated versus separated allocation and scheduling (Section 2.5.4), and CCSP versus DR-CCSP (Section 2.5.5).

2.5.1 Experimental Design and Setup

We generate all the instances based on the data collected in an ASC at the Mayo Clinic in Rochester, Minnesota (see Gul et al., 2011, for a detailed description of this surgery practice). The original data contains three key steps of surgical services: *(i) intake*, which starts when a patient arrives at the surgical suite to initiate check-in, and ends when the patient reaches an OR bed; *(ii) intra-operative care*, which starts when a patient is admitted to the OR area and ends when the patient is taken to a recovery bed; and *(iii) recovery*, which starts when a patient is admitted to a recovery area and ends when the patient is discharged. We use probability density functions fit to the data for surgical procedures in step *(ii)*.

Data description: Table 2.1 above summarizes the mean, standard deviation, and number of cases of each surgery type (defined by type of surgery and its surgical group). We consider 25 surgeries and specify the type of each surgery according to the percentages of different types of surgeries at the ASC as shown in Column **Percentage**. Using these percentages, we compute the number of surgeries of each type among the 25 surgeries, which could be fractional. We round the numbers to integers (see Column **Tested #**) that add up to 25.

Table 2.1: Summary statistics of surgeries of all types based on the extracted data

Type	Group No.	Mean	St. dev.	# of cases	Percentage	Tested #
Oral Maxillofacial	1	33.00	19.11	1472	14.04%	4
	2	36.00	33.88	1919	18.30%	4
Pain Medicine	1	19.78	12.12	58	0.55%	0
	2	20.49	10.86	244	2.33%	1
	3	20.93	15.08	1551	14.79%	4
	4	40.50	26.12	24	0.23%	0
	5	34.01	17.42	970	9.25%	2
Ophthalmology	1	41.63	16.43	1696	16.17%	4
	2	77.66	44.03	589	5.62%	1
Urology	1	53.30	27.70	329	3.14%	1
	2	31.30	16.37	640	6.10%	2
	3	138.16	56.77	153	1.46%	0
	4	55.78	22.89	345	3.29%	1
	5	80.33	43.76	496	4.73%	1

Table 2.2: Normalized opening costs and operating hours of the ORs

	OR j							
	1	2	3	4	5	6	7	8
c_j	0.9	0.9	0.9	1.0	1.0	1.1	1.1	1.1
T_j (hours)	8	8	8	9	9	9	10	10

Each OR in the ACS can be used by any surgery group for any surgery type. (For situations when this is not the case, one can simply enforce the corresponding z or y variables being zero, if surgery i cannot be performed in OR j.) Table 2.2 presents the normalized cost c_j of opening OR j for T_j hours. The ASCs often stagger the closing time of ORs, i.e., three operate 7am–3pm, three operate 8am–5pm, and two operate 8am–6pm. The cost of opening an OR varies among different hospitals, depending on the fixed cost of equipment maintenance and the cost of nurses and other personnel required to support surgery operations. Thus, we normalize the values of c_j, $j \in R$, and use $1:1$ ratio of a fixed daily cost of operating an OR to an hourly variable cost. We set $c_j = 1 = 0.1 + 0.1 \times 9$ (hours) for an OR j that operates from 8am to 5pm. Consequently, the costs of operating the other two types of ORs, open for 8 and 10 hours, are 0.9 and 1.1, respectively.

Instance generation: Gul et al. (2011) conduct hypothesis tests and show that the durations of each surgery type are well approximated by Log-Normal distributions. In practice, it may not be feasible to collect enough data for deriving full distributional information. We consider N_{obs} observed data samples, for which we vary the size N_{obs} in our tests. We sample $\hat{\xi}_i^n$, $n =$

$1,\ldots,N_{\text{obs}}$ from S independent Log-Normal distributions, with the mean μ_i and standard deviation σ_i of each surgery $i \in \mathcal{S}$ from Table 2.1. We classify the models into two types: *(i)* two-stage stochastic optimization models (including CCSP and other model variants that assume fully known distributions), for which we fit the distribution of ξ to match $\{\hat{\xi}^n\}_{n=1}^{N_{\text{obs}}}$; and *(ii)* DR-CCSP which we optimize by solving its CCSP counterpart and using an empirical distribution of ξ that could be the same as the distribution we derive for the corresponding type *(i)* models. For both type *(i)* and type *(ii)* models, we apply the Monte Carlo sampling to generate independent samples of ξ from the hypothesized distribution. The MIP equivalent formulations are constructed based on these samples, corresponding to the set Ω in our previous models. The details are as follows.

- For stochastic optimization models, to emulate an instance with insufficient observations, we compute the empirical mean and standard deviation of each surgery's duration time, but have to assume specific distribution types. We pick three distributions that are frequently used for modeling service durations. We use the Monte Carlo sampling approach to generate N_{MC} scenarios of ξ by following independent Normal, Log-Normal and Gamma distributions of ξ_i with mean $\hat{\mu}_i$ and standard deviation $\hat{\sigma}_i$ computed based on $\{\hat{\xi}_i^n\}_{n=1}^{N_{\text{obs}}}$, for all $i \in \mathcal{S}$. We denote the three sets of samples as $\{\xi^\omega : \omega \in \Omega_{\text{Nor}}(N_{\text{MC}})\}$, $\{\xi^\omega : \omega \in \Omega_{\text{Log}}(N_{\text{MC}})\}$ and $\{\xi^\omega : \omega \in \Omega_{\text{Gam}}(N_{\text{MC}})\}$. In particular, if a sampled $\xi_i^\omega < 0$ for some scenario $\omega \in \Omega_{\text{Nor}}(N_{\text{MC}})$, and surgery $i \in \mathcal{S}$, we replace the value with zero.

- In DR-CCSP, for each surgery i we let $[\underline{\xi}_i, \overline{\xi}_i]$ be the observed data range, with $\underline{\xi}_i$ and $\overline{\xi}_i$ being the minimum and the maximum values in $\{\hat{\xi}_i^n\}_{n=1}^{N_{\text{obs}}}$, respectively. We then discretize the range into Δ_i-minute intervals, for some positive integer Δ_i. Specifically, we have $K_i = \lfloor (\overline{\xi}_i - \underline{\xi}_i)/\Delta_i \rfloor$ intervals, such that for each $i \in \mathcal{S}$, we let intervals $\mathcal{B}_k^i = [\overline{\xi}_i - \Delta_i(K_i - k + 1), \overline{\xi}_i - \Delta_i(K_i - k)]$, for all $k = 2,\ldots,K_i$, and set the interval $\mathcal{B}_1^i = [\underline{\xi}_i, \overline{\xi}_i - \Delta_i(K_i - 1)]$. (Note that if the range is not divisible by Δ_i, we choose to extend the first interval, given that we are restricting the probability of OR overtime that is affected more by excessively long surgery durations than short ones. The purpose is to keep the precision for the right tail of the empirical distribution.) To solve the CCSP with the new chance constraint (2.32), we follow the empirical PMFs f_0^i of ξ_i for all $i \in \mathcal{S}$ according to (2.29), and sample a set of intervals $\{(\mathcal{B}_{\omega_1}^1, \ldots, \mathcal{B}_{\omega_S}^S) : \omega = (\omega_1, \ldots, \omega_S) \in \Omega_{\text{DR}}\}$ with $|\Omega_{\text{DR}}| = N_{\text{MC}}$. We obtain realizations of surgery durations as $\{\xi^\omega = (\xi_1^{\omega_1}, \ldots, \xi_S^{\omega_S}) : \omega = (\omega_1, \ldots, \omega_S) \in \Omega_{\text{DR}}\}$ where each $\xi_i^{\omega_i}$ is the midpoint of interval $\mathcal{B}_{\omega_i}^i$.

To evaluate the performance of each solution, we use a post-optimization simulation sample denoted by $\{\xi^\omega : \omega \in \Theta\}$ containing $N_{\text{sim}} = 10{,}000$ independent scenarios, in which ξ_i^ω, $\omega \in \Theta$ are sampled from the Log-Normal distributions with mean μ_i and standard deviation σ_i of each $i \in \mathcal{S}$.

Moreover, we numerically test the robustness of optimal surgery plans produced by the CCSP and DR-CCSP models on instances with random surgery cancellations. We perform "stress tests" by assuming probability q_i of canceling surgery i, $\forall i \in \mathcal{S}$. Then for each surgery i, we multiply every realization ξ_i^ω, $\forall \omega \in \Theta$ by a 0-1 Bernoulli random variable that has probability q_i for being 0. We use $q_i = 0.005$ and $q_i = 0.01, \forall i \in \mathcal{S}$ to attain two more simulation samples having N_{sim} scenarios denoted as $\{\xi^\omega : \omega \in \Theta_{0.005}\}$ and $\{\xi^\omega : \omega \in \Theta_{0.01}\}$, respectively.

Other parameters and configurations: We set the waiting tolerance of surgery i to $\epsilon_i = w_1\mu_i + w_2\sigma_i$ according to the mean μ_i and standard deviation σ_i in Table 2.1, where $(w_1, w_2) = (0.2, 0.1)$. The intuition is that the waiting tolerance is larger if a surgery takes longer time and has larger variance. Also, the ratio $w_1/w_2 = 2$ indicates that ϵ_i has higher dependence on the mean μ_i than the standard deviation σ_i, for each surgery i. Testing several choices of (w_1, w_2) showed that the results were not very sensitive to changes in the neighborhood of $(0.2, 0.1)$. For larger changes, due to too large or too small ϵ_i, the waiting-time constraints are either relaxed, or we were unable to obtain a feasible solution to one or both of CCSP and DR-CCSP models.

All computations are performed on a Linux workstation with four 3.4 GHz processors and 16 GB memory. All involved MIP and linear programming models are solved by CPLEX 12.6 via ILOG Concert Technology. We set the computational time limit to 7200 seconds. For each test with the same set of parameters, we test ten independently generated replications.

Post-optimization simulation: To evaluate the performance of a solution (x, z, s) in any simulation sample (i.e., Θ, $\Theta_{0.005}$, or $\Theta_{0.01}$), we first derive the actual schedules $\delta^{j\omega}$ for every open OR $j \in \mathcal{R}(x)$ in every scenario ω according to the closed-form expressions for $\delta^{j\omega}$ in (2.12). For each scenario w, we calculate:

- total overtime: $\mathcal{T}^\omega_{\text{over}} = \sum_{j\in\mathcal{R}} \max\{0, \delta^{j\omega}_S - T_j\}$;
- total waiting time: $\mathcal{T}^\omega_{\text{wait}} = \sum_{j\in\mathcal{R}} \sum_{k=2}^{S} \max\{0, \delta^{j\omega}_{k-1} - s^j_k\}$;
- total idle time: $\mathcal{T}^\omega_{\text{idle}} = \sum_{j\in\mathcal{R}} (\delta^{j\omega}_S - \sum_{k=1}^{S} \sum_{i\in\mathcal{S}} \xi^\omega_i z^j_{ik})$.

The next section summarizes a variety of statistics of the above outcome values for analyzing solution effectiveness and robustness.

2.5.2 Computational Efficacy

First, we compare the branch-and-cut algorithm explained in Section 2.3 with directly calling the solver to solve the MIP reformulation of the CCSP problem, i.e., (2.10), on samples $\{\xi^\omega : \omega \in$

$\Omega_{\text{Log}}(N_{\text{MC}})\}$, for $N_{\text{MC}} = 100\sim500$, under $\beta = 0.90$ and $\beta = 0.95$. We use $N_{\text{obs}} = 50$ data observations to obtain the empirical means and standard deviation for service durations that follow independent Log-Normal distributions. For each N_{MC} and parameter setting, we present the results of the ten replications in Table 2.3. Since no MIP formulations can be directly solved within the time limit, we report the average, maximum, and minimum optimality gaps reported by CPLEX; for the branch-and-cut algorithm, we report the average, maximum, and minimum solution time together with the average number of packing cuts and scheduling cuts generated.

Table 2.3: Comparison of the proposed branch-and-cut approach versus directly calling the solver in solving the MIP reformulation of CCSP

		Computing MIP Reformulation directly			Branch-and-Cut Algorithm 2.3				
		optimality gap			time (seconds)			# of cuts	
β	N_{MC}	avg	max	min	avg	max	min	packing	scheduling
0.90	100	9%	43%	4%	57	125	19	72	20
	200	33%	84%	17%	170	595	47	297	117
	300	46%	84%	29%	788	1351	453	289	325
	400	72%	84%	10%	3444	4270	1833	433	544
	500	-	-	72%	3524	6890	2434	374	2370
0.95	100	55%	87%	22%	23	75	3	77	20
	200	62%	82%	5%	99	201	27	314	666
	300	63%	82%	5%	812	2998	313	462	651
	400	-	-	72%	3900	4832	2111	690	960
	500	-	-	72%	4175	7101	3772	840	3170

"-": the solver fails to attain a finite gap for some instances within 7200 seconds.

As compared to directly solving the MIP formulation, Table 2.3 shows that the branch-and-cut approach in Algorithm 2.3 efficiently solves all the instances to optimum. As N_{MC} increases, the number of scheduling cuts increases faster than the number of packing cuts; the average computational time for $\beta = 0.95$, which is initially shorter, increases faster than the average time for $\beta = 0.90$ when N_{MC} increases, and becomes longer than for the case of $\beta = 0.90$ when $N_{\text{MC}} \geq 300$.

2.5.3 Chance-Constrained Model versus Cost-Based Model

The existing literature on stochastic surgery planning applies cost-based approaches, which penalize undesirable outcomes (e.g., overtime, waiting time, idle time, etc.) and minimize the total expected penalty cost. We compare CCSP with a cost-based model, which penalizes the expected overtime

$$\min\left\{\sum_{j\in R} c_j x_j + \frac{1}{|\Omega|}\sum_{\omega\in\Omega} c_{\text{over}} \sum_{j\in\mathcal{R}} \max\{0, \delta_S^{j\omega} - T_j\} : (2.1)\text{–}(2.7),\ x, z \text{ binary},\ s, \delta \geq 0\right\}, \tag{2.36}$$

where c_{over} denotes unit overtime cost. We set the smallest $c_{\text{over}} = 0.50$, as in Denton et al. (2010), which represents the case where two hours of OR overtime costs the same as opening a new OR that operates from 8am to 5pm. We continue using $N_{\text{obs}} = 50$ data observations. For $\Omega = \Omega_{\text{Log}}(N_{\text{MC}})$ with $N_{\text{MC}} = 500$, we compare CCSP and the cost-based model (2.36) for various β- and c_{over}-values. We evaluate each solution in the simulation sample $\{\xi^\omega : \omega \in \Theta\}$, and present the simulation results in Table 2.4. In particular, we report the mean (`mean`), standard deviation (`stdev`), quantiles (`50%` and `95%`), and the skewness (`skew`) of the outcomes $\mathcal{T}^\omega_{\text{over}}$, $\omega \in \Theta$. We also report the simulated probability of existing overtime in any OR (`Prob` $= (1/|\Theta|)\sum_{\omega\in\Theta} \mathbf{1}(\mathcal{T}^\omega_{\text{over}} > 0)$) and the total cost of opening ORs followed by the number of open ORs ($\texttt{C}^{\text{open}}$ (# of ORs)) for each solution. In all the tables of this chapter, the results are similar for all ten replications; the variation is no more than 5%, suggesting a tight confidence interval.

Table 2.4: Comparison of the chance-constrained model and the cost-based model in cost and zero-overtime reliability

CCSP							
β	$\texttt{C}^{\text{open}}$ (# of ORs)	overtime $\mathcal{T}^\omega_{\text{over}}$, $\omega \in \Theta$					
		`Prob`	`mean`	`stdev`	`50%`	`95%`	`skew`
1.00	2.9 (3)	0.10	6	26	0	31	7.6
0.95	2.2 (2)	0.12	13	38	0	56	5.2
0.90	2.1 (2)	0.21	18	46	0	66	4.5
0.85	2.1 (2)	0.32	21	48	0	82	4.0
The Cost-based Model							
c_{over}	$\texttt{C}^{\text{open}}$ (# of ORs)	overtime $\mathcal{T}^\omega_{\text{over}}$, $\omega \in \Theta$					
		`Prob`	`mean`	`stdev`	`50%`	`95%`	`skew`
1.25	2.1 (2)	0.93	47	87	53	152	1.6
1.00	2.0 (2)	0.99	196	101	184	325	0.8
0.75	2.0 (2)	1	266	110	252	408	0.9
0.50	1.1 (1)	1	345	120	333	492	0.7

In Table 2.4, increasing β and c_{over} reduces the mean of overtime monotonically in the two models. The CCSP model, however, allows better control of the overtime probability. As β increases from 0.85 to 1.00 (or equivalently, the admissible probability of OR overtime decreases from 0.15 to 0), the chance of observing overtime in the simulation decreases from 0.32 to 0.10. Although these values do not meet the corresponding β targets, they are significantly better than the values yielded by the cost-based solution (which are all higher than 0.90 for any c_{over} between 0.50 and 1.25). The CCSP solutions also make the simulated overtime, desirably, less variable and more right-skewed. They also perform better, based on the mean, standard deviation, and the 50% and 95% quantiles. In particular, all of the CCSP solutions yield zero overtime at the 50% quantile, meaning that at least half of the scenarios in the simulation sample do not have OR over-

time if implementing the CCSP solutions. On the other hand, CCSP solutions yield higher cost of opening ORs. In particular, the CCSP model opens three ORs, whereas the cost-based model only opens two when $\beta = 1.00$, and the CCSP model opens two ORs, whereas the cost-based model only opens one when $\beta = 0.85$.

To better demonstrate the tradeoff between overtime and waiting time given by the two models, Figure 2.1 plots the summed $(\mathcal{T}^{\omega}_{\text{wait}}, \mathcal{T}^{\omega}_{\text{over}})$ over the ten replications of instances for each scenario $\omega \in \Theta$. From Figure 2.1, we observe that the optimal CCSP solutions result in shorter waiting in most of the scenarios, and thus have much better average waiting time than the solutions given by the cost-based model. In the worst case, the cost-based model almost doubles the total waiting time than the CCSP model for any (β, c_{over})-values we test. Although the cost-based solutions have lower cost of opening ORs, they tend to over-pack ORs and produce schedules that perform poorly in terms of overtime probability.

2.5.4 Integrating Versus Separating Allocation and Scheduling

We want to determine if using the integrated model brings any significant gains. Thus, we compare CCSP with a model where we separately consider a CCBP problem and then a chance-constrained surgery scheduling (CCSS) problem. The procedure is as follows. First, we pack surgeries with random durations into ORs seamlessly, such that the probability of exceeding the standard operating time length of any OR is no more than $1-\beta$. The formulation is presented in (2.14). After solving that CCBP problem, we pass its optimal solution (x', y') to the following CCSS which enforces the waiting time restriction, while maximizing the probability of no overtime in any OR, i.e.,

$$\begin{aligned}
\bar{\beta}(x', y') = \max \quad & \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \mathbf{1}\left(\delta_S^{j\omega}(\xi) \le T_j,\ \forall j \in \mathcal{R}\right) \\
\text{s.t.} \quad & \sum_{k=1}^{S} z_{ik}^{j} = y'_{ij} \quad \forall i \in \mathcal{S},\ j \in \mathcal{R} && (2.37) \\
& \sum_{i \in \mathcal{S}} z_{ik}^{j} \le x'_j \quad \forall k = 1, \ldots, S,\ j \in \mathcal{R} && (2.38) \\
& (2.3)\text{–}(2.7),\ z \text{ binary},\ s, \delta \ge 0
\end{aligned}$$

where (2.37) and (2.38) build the relations between z and the given y' and x', respectively. Letting $\Omega = \Omega_{\text{Log}}(N_{\text{MC}})$ with $N_{\text{MC}} = 500$, we optimize CCSS to obtain a solution (z', s', δ'). If $\tilde{\beta}(x', y') < \beta$, it implies that the allocation and scheduling solution given by solving CCBP followed by CCSS, is not feasible to the original model CCSP.

We compare a CCSP solution $(\hat{x}, \hat{y})$ and a CCBP solution (x', y'), for the cost of opening ORs

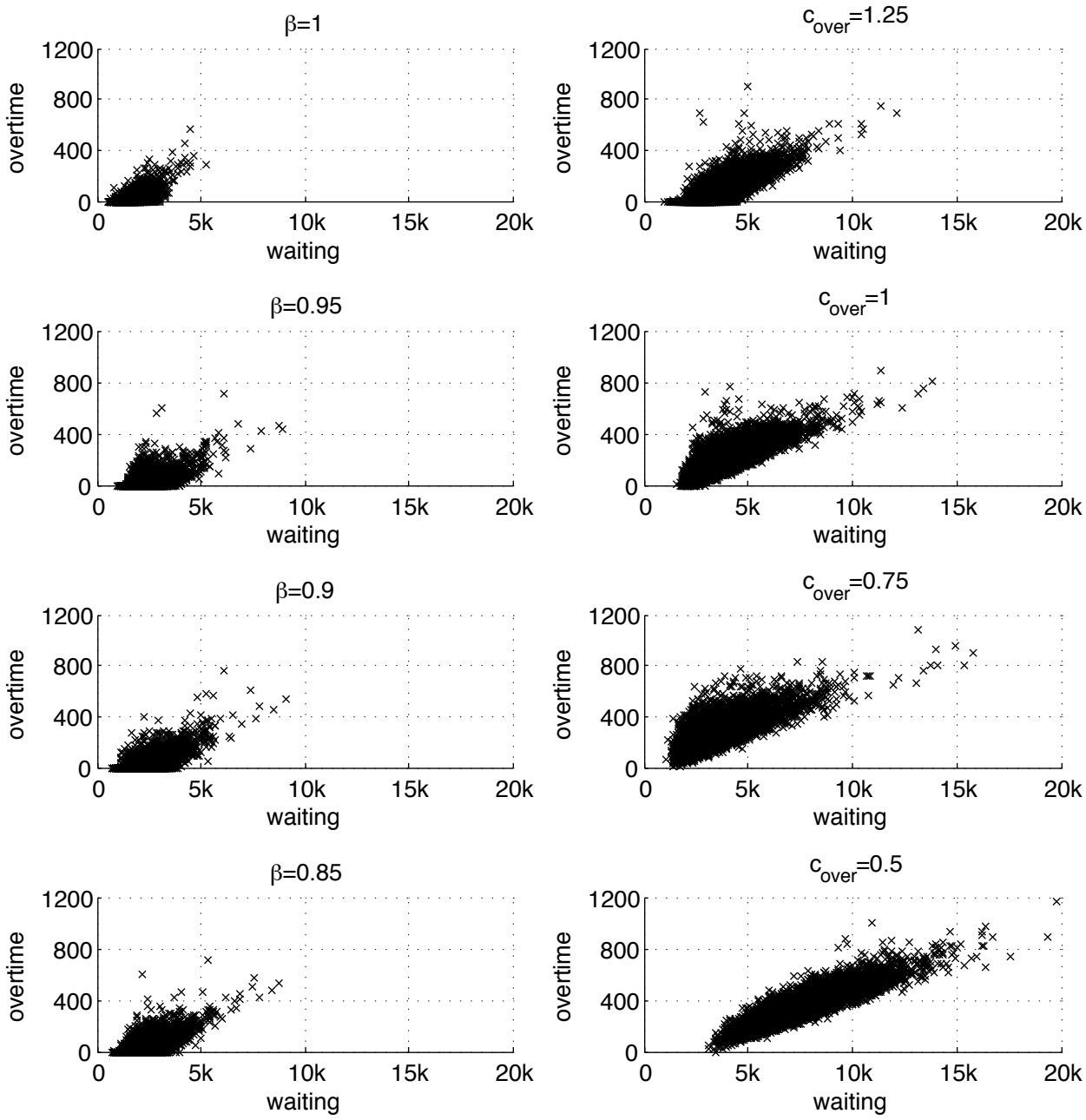

Figure 2.1: Comparison of the CCSP model (left) and the cost-based model (right) in $(\mathcal{T}^{\omega}_{\text{wait}}, \mathcal{T}^{\omega}_{\text{over}})$

Table 2.5: Comparison of the integrated model CCSP versus the allocation-only model CCBP

	C^{open} (# of ORs)			$\bar{\beta}(\cdot)$ as in CCSS		
β =	0.85	0.90	0.95	0.85	0.90	0.95
CCBP	2.0 (2)	2.0 (2)	2.2 (2)	0.75	0.79	0.82
CCSP	2.1 (2)	2.1 (2)	2.9 (3)	0.89	0.93	0.95

Overtime	Prob			mean			50%			95%		
β =	0.85	0.90	0.95	0.85	0.90	0.95	0.85	0.90	0.95	0.85	0.90	0.95
CCBP	0.46	0.31	0.25	24.2	20.1	14.8	3.1	2.6	2.1	104.9	93.6	74.1
CCSP	0.32	0.21	0.12	21.2	17.6	13.1	0.4	0.3	0.0	82.1	65.8	55.7

(C^{open}), and the maximum probability of no overtime when fixing them in CCSS, i.e., the values of $\bar{\beta}(\hat{x},\hat{y})$ and $\bar{\beta}(x',y')$. We also implement the solutions in the post-optimization simulation sample Θ, and present the simulated probability of overtime existence (`Prob`) and the mean of overtime (`mean`; in minute). We test $\beta = 0.85$, 0.90 and 0.95, and present the results in Table 2.5.

As expected, the allocation-only CCBP model yields lower cost of opening ORs, since it is a relaxation of CCSP which assumes no idle time between adjacently allocated surgeries. However, given an allocation decision passed from CCBP, there might not exist feasible schedules to guarantee the desired β probability of no overtime. In Table 2.5, all of the CCSP solutions yield $\bar{\beta}(\hat{x},\hat{y}) > \beta$ in CCSS, which must be true because the target probability β is imposed on sufficiently many scenarios in $\Omega_{\text{Log}}(N_{\text{MC}})$ to satisfy the chance constraint in CCSP. In contrast, CCBP solutions perform poorly and have $\bar{\beta}(x',y') < \beta$ for the three β-values tested. Moreover, CCBP yields solutions with significantly higher overtime probability in the simulation sample, longer means of overtime, and longer overtime at both the 50% and 95% quantiles. To avoid such a performance gap, a decision maker may have to intentionally increase the target probability β when using CCBP for planning ORs, but doing so may lead to multiple computational iterations and overly conservative solutions with unnecessarily high OR opening cost.

2.5.5 Incorporating Data Ambiguity via Distributionally Robust Model

We conduct numerical experiments to illustrate the value of considering distribution ambiguity and solving DR-CCSP. Section 2.5.5.1 investigates the sensitivity of CCSP solutions to distributional assumptions. Section 2.5.5.2 designs the parameters for DR-CCSP and compares the results with CCSP. Section 2.5.5.3 shows how the data size N_{obs} affects the performance of DR-CCSP solutions.

2.5.5.1 CCSP solution sensitivity to the distribution type

In addition to Log-Normal, we consider two other distributions, Normal and Gamma, that are commonly used for modeling random service durations. We solve CCSP on the three randomly generated samples $\{\xi^\omega : \omega \in \Omega_{\text{Nor}}(N_{\text{MC}})\}$, $\{\xi^\omega : \omega \in \Omega_{\text{Log}}(N_{\text{MC}})\}$, and $\{\xi^\omega : \omega \in \Omega_{\text{Gam}}(N_{\text{MC}})\}$, all with $N_{\text{MC}} = 100$. We set β to 0.95, 0.90 and 0.85. Table 2.6 presents the simulation results of CCSP solutions in the simulation sample Θ.

Table 2.6: Performance of CCSP solutions under different distribution assumptions

Distribution	β	C^{open} (# of open ORs)	overtime					
			Prob	mean	stdev	50%	95%	skew
Log-Normal	0.95	2.2 (2)	0.12	13	39	0	56	5.2
	0.90	2.1 (2)	0.21	18	46	0	66	4.5
	0.85	2.1 (2)	0.32	21	48	0	82	4.0
Gamma	0.95	2.8 (3)	0.18	8	32	0	49	7.1
	0.90	2.8 (3)	0.19	12	37	0	53	5.3
	0.85	2.7 (3)	0.28	19	42	0	64	4.5
Normal	0.95	2.8 (3)	0.19	9	32	0	50	7.1
	0.90	2.7 (3)	0.20	10	34	0	51	6.8
	0.85	2.2 (2)	0.31	20	45	0	71	5.3

Distribution	β	waiting time			idle time		
		mean	stdev	skew	mean	stdev	skew
Log-Normal	0.95	205.1	60.1	2.3	94	55	0.2
	0.90	208.2	71.8	1.5	43	34	0.8
	0.85	231.1	89.0	0.7	71	66	0.3
Gamma	0.95	141.5	37.9	1.5	273	63	0.0
	0.90	181.4	65.9	1.7	478	48	-0.3
	0.85	195.6	71	1.1	456	78	0.0
Normal	0.95	145.7	49.2	2.1	259	57	-0.5
	0.90	62.8	23.8	3.4	799	76	-0.3
	0.85	94.5	45.1	1.9	540	112	0.0

In Table 2.6, both Gamma and Normal distributions yield more conservative CCSP solutions than the Log-Normal distribution. In terms of overtime probability and statistical measures (i.e., mean, standard deviation, 50%, 95% quantiles) reported by using the simulation sample Θ, solutions based on Gamma and Normal distributions perform very close to those based on the Log-Normal distribution, but have 10% $\sim$ 30% higher OR-opening cost. Moreover, the waiting time is shorter but the idling is longer. The results suggest that decision makers need to assume a precise distribution in situations where waiting restrictions are not binding.

We also observe in some cases that under the same Monte Carlo samples but different values of β, the solutions have the same cost of opening ORs. However, the performance of an optimal

Table 2.7: Probability of waiting $> \epsilon_i$ among all 25 surgeries given by CCSP

Distribution	β	Avg	Max	Min
Log-Normal	0.95	0.05	0.11	0
	0.9	0.06	0.12	0
	0.85	0.07	0.14	0.02
Gamma	0.95	0.03	0.06	0
	0.9	0.04	0.07	0
	0.85	0.04	0.09	0
Normal	0.95	0.03	0.06	0
	0.9	0.02	0.04	0
	0.85	0.02	0.05	0

CCSP solution associated with higher β is better. For example, using $\Omega_{\text{Log}}(N_{\text{MC}})$, both $\beta = 0.90$ and $\beta = 0.85$ suggest to open OR 4 and OR 7 at optimum, but the solution provided by $\beta = 0.85$ has longer OR overtime, waiting, and idleness. We observe similar results when using $\Omega_{\text{Gamma}}(N_{\text{MC}})$ for $\beta = 0.95$ and $\beta = 0.90$. These observations suggest that raising β (without changing optimal ORs to open) may generate operating schedules with better performance.

We also compare the waiting time of each surgery i in every scenario of the simulation sample Θ with the maximum tolerable waiting ϵ_i, and calculate the probability of having significant waiting (i.e., surgery i's waiting time being longer than ϵ_i). Table 2.7 shows that the average and maximum waiting probabilities of the 25 surgeries for different β-values are well below 0.05 for both Gamma and Normal distributed training samples, but higher when surgery durations of a training sample follow Log-Normal distributions. The results also show zero risk of having significant waiting for most of the surgeries. Therefore, if we require at most 0.10 probability of having significant waiting time, the deterministic upper bound constraints on waiting time in CCSP can produce feasible solutions to the individual chance constraints.

2.5.5.2 Results of DR-CCSP

We test two specific ϕ-divergences: *(i)* the modified χ^2-distance divergence $\phi_{\chi^2}(t) = (t-1)^2$ for a scalar $t \geq 0$, and *(ii)* the Kullback-Leibler (KL) divergence, where $\phi_{\text{KL}}(t) = t \ln t - t + 1$ for $t \geq 0$. Note that their ϕ-divergence distances $\mathcal{D}_\phi(f\|f_0) = \mathcal{D}_\phi(t)$ considered as functions of $t > 0$ satisfy $\mathcal{D}_{\phi_{\chi^2}}(t) \geq \mathcal{D}_{\phi_{\text{KL}}}(t)$. We solve DR-CCSP under these two divergence measures in order to compare the results with the results of CCSP above.

According to Theorem 2.1, we can reformulate a DR chance constraint (2.31) as a regular chance constraint (2.32) where the probability is measured based on the empirical distribution function f_0, and bounded by $\mathcal{V}_\phi(\beta, d)$ which is dependent on the choice of ϕ-divergence measure, the original probability threshold β and the distance tolerance d. Specifically, we set $d = d_{\text{M}} =$

$\sum_{i\in\mathcal{S}} d_i$ as described in Section 2.4.3, where d_i yields an asymptotically $1-\alpha = 90\%$-confidence uncertainty set for each marginal PMF f_i. For $\phi = \phi_{\mathrm{KL}}$ and $\phi = \phi_{\chi^2}$, we compute

$$\mathcal{V}_{\phi_{\chi^2}}(\beta, d) = \beta + \frac{\sqrt{d^2 + 4d(\beta-\beta^2)} - (2\beta-1)d}{2d+2}$$
$$\mathcal{V}_{\phi_{\mathrm{KL}}}(\beta, d) = \inf_{t\in(0,1)} \left\{ \frac{e^{-d} t^{\beta} - 1}{t-1} \right\},$$

both of which are obtained based on the general form provided in Theorem 2.1. (In particular, the latter is attained through a bisection search procedure.) We obtain the discrete support Ξ_i of each ξ_i by partitioning its range based on intervals of $\Delta_i = 5$ minutes. Following Section 2.5.1, we recover the empirical marginal distributions f_0^i for every $i \in \mathcal{S}$ and subsequently generate N_{MC} scenarios with respect to the empirical joint distribution f_0.

We then solve the CCSP representation of the DR-CCSP model by using the branch-and-cut algorithm. Table 2.8 compares DR-CCSP and CCSP for $\beta = 0.90$ and $\Omega = \Omega_{\mathrm{Log}}(N_{\mathrm{MC}})$. We simulate the results in the previous simulation sample $\{\xi^{\omega} : \omega \in \Theta\}$, and also two other simulation samples that involve random surgery cancellations, i.e., $\{\xi^{\omega} : \omega \in \Theta_{0.005}\}$ and $\{\xi^{\omega} : \omega \in \Theta_{0.01}\}$.

The solutions of DR-CCSP show better performance in OR overtime and surgery waiting; in most instances, overtime probability, overtime mean, and waiting time mean all decrease by 50%. The DR-CCSP solutions are also more reliable and robust when we introduce random surgery cancellations into the simulation samples. The deteriorations in the overtime probability, and in the means of overtime, waiting time, and idle time are smaller when we implement the solutions of DR-CCSP than those when implementing the solutions of CCSP.

Comparing the two DR-CCSP models, the ϕ_{KL}-divergence produces more conservative solutions. The simulated overtime probabilities by applying those solutions all meet (i.e., being no less than) the probability target $\beta = 0.90$, without too much extra cost for opening ORs (i.e., $\frac{2.9-2.7}{2.7} \times 100\% = 7.41\%$ cost increase). As a result, the mean values of OR overtime and surgery waiting time are shorter, but the idle time means are much longer given by DR-CCSP. In Table 2.6 note that the CCSP solutions with inaccurate distribution assumptions (i.e., the Normal and Gamma distributions) yield OR-opening cost (i.e., $\mathrm{C}^{\mathrm{open}}$) as high as the cost of DR-CCSP solutions. The CCSP solutions, however, perform worse in terms of overtime probability and length of OR overtime.

Table 2.9 shows the waiting probability in post-optimization simulation given by solutions of CCSP and DR-CCSP. In general, both DR-CCSP(ϕ_{χ^2}) and DR-CCSP(ϕ_{KL}) produce more reliable solutions than CCSP, yielding smaller risk of significant waiting of any surgery, on average and at maximum. The DR-CCSP(ϕ_{χ^2}) is slightly more conservative and yields smaller risk of waiting than DR-CCSP(ϕ_{KL}).

Table 2.8: Comparison of CCSP and DR-CCSP for $\beta = 0.90$ and $\Omega = \Omega_{\text{Log}}(N_{\text{MC}})$

Model	Ref.	overtime					
		Prob	mean	stdev	50%	95%	skew
CCSP	Θ	0.210	17.6	46	0	66.3	4.5
$\mathbb{C}^{\text{open}} = 2.1$	$\Theta_{0.005}$	0.206	17.2	46	0	64.2	4.6
(2 open ORs)	$\Theta_{0.01}$	0.202	16.6	45	0	60.9	4.7
DR-CCSP(ϕ_{χ^2})	Θ	0.131	7.2	30	0	27.1	8.4
$\mathbb{C}^{\text{open}} = 2.7$	$\Theta_{0.005}$	0.129	7.1	30	0	26.8	8.5
(3 open ORs)	$\Theta_{0.01}$	0.126	6.8	29	0	25.9	8.7
DR-CCSP(ϕ_{KL})	Θ	0.100	5.7	26	0	21.2	7.6
$\mathbb{C}^{\text{open}} = 2.9$	$\Theta_{0.005}$	0.098	5.5	26	0	20.8	7.6
(3 open ORs)	$\Theta_{0.01}$	0.094	5.3	25	0	20.5	7.8

Model	Ref.	waiting time			idle time		
		mean	stdev	skew	mean	stdev	skew
CCSP	Θ	208.2	71.8	1.5	42.6	34	0.8
$\mathbb{C}^{\text{open}} = 2.1$	$\Theta_{0.005}$	206.6	71.6	1.5	43.9	34	0.8
(2 open ORs)	$\Theta_{0.01}$	204.8	71.2	1.5	45.3	35	0.8
DR-CCSP(ϕ_{χ^2})	Θ	102.6	36.3	2.5	198.9	62	0.0
$\mathbb{C}^{\text{open}} = 2.7$	$\Theta_{0.005}$	101.9	36.3	2.5	201.3	63	0.0
(3 open ORs)	$\Theta_{0.01}$	101.2	36	2.5	203.4	64	0.0
DR-CCSP(ϕ_{KL})	Θ	130	43.3	1.8	225.2	66	-0.2
$\mathbb{C}^{\text{open}} = 2.9$	$\Theta_{0.005}$	129.2	43.3	1.8	227.4	66	-0.2
(3 open ORs)	$\Theta_{0.01}$	128.2	43.1	1.8	229.4	67	-0.2

2.5.5.3 Value of data in DR-CCSP

The results of DR-CCSP in the previous section are based on $N_{\text{obs}} = 50$ observed samples. Intuitively, as the number N_{obs} of observations increases, we can depict the ambiguous probability distribution of ξ more accurately with a smaller d for restricting the ϕ-divergence distance. We test data sizes N_{obs} from 50 to 200, and keep $N_{\text{MC}} = 500$ for optimizing the equivalent CCSP reformulations of the corresponding DR-CCSP models.

We set $d = d_{\text{M}}$ according to the procedures in Section 2.4.3, of which the values for DR-CCSP(ϕ_{KL}) and for DR-CCSP(ϕ_{χ^2}) only differ by 2, because $\phi''_{\text{KL}}(1) = 1$ and $\phi''_{\chi^2}(1) = 2$. Figure 2.2 maps the values of d against the choice of N_{obs}. Table 2.10 presents the post-optimization simulation results of the attained solutions of DR-CCSP(ϕ_{KL}) and DR-CCSP(ϕ_{χ^2}) for $N_{\text{obs}} = 50, 100$, and 200 in the simulation sample Θ.

The proposed value for d absorbs the differences in the divergence measures we use and in the size of data observations, so that when we vary N_{obs}, or switch from using ϕ_{χ^2} to ϕ_{KL}, we have relatively small changes if implementing DR-CCSP solutions. The results suggest that we can maintain good performance of DR-CCSP solutions by using bigger d for fewer observations.

Table 2.9: Probability of waiting > ϵ_i among all 25 surgeries given by CCSP and DR-CCSP models

Model	Ref.	Avg	Max	Min
CCSP	Θ	0.06	0.12	0
	$\Theta_{0.005}$	0.06	0.12	0
	$\Theta_{0.01}$	0.07	0.14	0.02
DR-CCSP(ϕ_{χ^2})	Θ	0.03	0.06	0
	$\Theta_{0.005}$	0.03	0.06	0
	$\Theta_{0.01}$	0.04	0.07	0.01
DR-CCSP(ϕ_{KL})	Θ	0.04	0.08	0
	$\Theta_{0.005}$	0.04	0.09	0
	$\Theta_{0.01}$	0.05	0.11	0.01

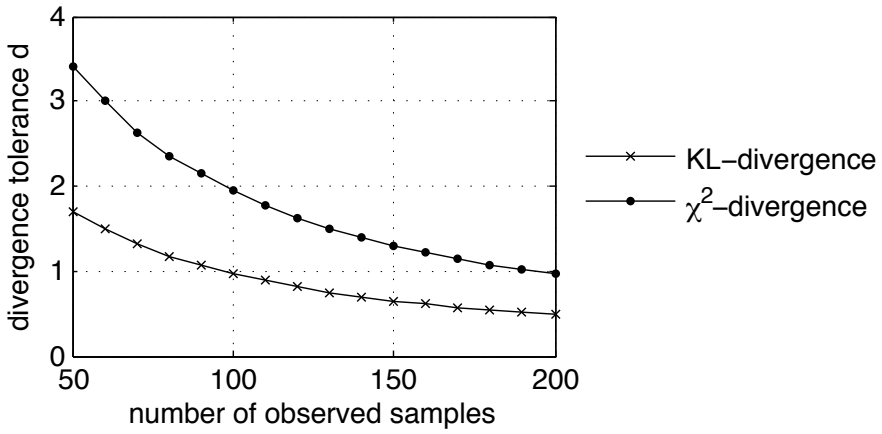

Figure 2.2: Distance tolerances d for KL- and χ^2-divergences according to the number of observed samples (N_{obs})

In turn, we can avoid over-conservative solutions by decreasing the value of d as the size of the observed data increases.

Table 2.11 confirms the low waiting probability of any solution given by the two DR-CCSP models, verified via the simulation samples with 10,000 independent scenarios. By solving DR-CCSP models, there is no more than a 0.05 probability of significant waiting on average among the 25 surgeries in post-optimization simulation. We observe no obvious trending for different N_{obs}.

2.6 Concluding Remarks

In this chapter, we developed new models for stochastic surgery planning where distributions of surgery durations may or may not be known. The models used chance constraints to control OR

Table 2.10: The effect of N_{obs} on the performance of DR-CCSP solutions

Model	N_{obs}	C^{open}	overtime					
			Prob	mean	stdev	50%	95%	skew
DR-CCSP(ϕ_{χ^2})	50	2.7	0.131	7.2	30	0	27.1	8.4
	100	2.8	0.140	7.7	30	0	27.8	7.3
	200	2.8	0.140	7.8	29	0	27.8	7.8
DR-CCSP(ϕ_{KL})	50	2.9	0.100	5.7	26	0	21.2	7.6
	100	2.9	0.108	6.8	29	0	23.7	8.2
	200	2.9	0.106	6.1	26	0	22.2	8.0

Model	waiting time			idle time		
	mean	stdev	skew	mean	stdev	skew
DR-CCSP(ϕ_{χ^2})	102.6	36.3	2.5	198.9	62	0.0
	111.2	34.9	2.1	267.0	56	-0.3
	93.1	32.7	2.2	239.1	63	-0.2
DR-CCSP(ϕ_{KL})	130	43.3	1.8	225.2	66	-0.2
	122.4	42.7	2.3	256.8	69	0.0
	135.3	52.2	2.0	257.0	68	0.0

Table 2.11: Probability of waiting $> \epsilon_i$ given by DR-CCSP models under different N_{obs}

Model	N_{obs}	Avg	Max	Min
DR-CCSP(ϕ_{χ^2})	50	0.03	0.06	0
	100	0.04	0.09	0
	200	0.03	0.07	0
DR-CCSP(ϕ_{KL})	50	0.04	0.08	0
	100	0.05	0.10	0
	200	0.05	0.09	0

overtime and surgery waiting time and integrated surgery-to-OR allocation decisions and surgery scheduling decisions to attain better-performing solutions. The problem was formulated as a complex chance-constrained program whose SAA reformulation was difficult to solve using existing methods. We proposed a branch-and-cut algorithm that took advantage of the scheduling problem structure. In particular, we derived a set of "packing cuts" from a bin packing problem, and another set of "scheduling cuts" based on single-OR scheduling problems. To handle the issue of distribution ambiguity, we formulated a DR variant of the CCSP problem, which, by applying the results of Jiang and Guan (2015), was transformed into a transitional chance-constrained program with a perturbed risk level that the proposed algorithm could handle. We generated test instances based on the operational data from an outpatient surgery center over a six-month time period, and tested the models and the algorithms on planning 25 surgeries across 8 ORs with operating time varying between 8∼10 hours. The results verified that the proposed branch-and-cut algorithm returned the

optimal solutions for all of the tested instances with 100~500 scenarios in two-hour runtime limit, whereas directly calling the state-of-the-art solver failed to solve any instance to optimum.

To evaluate a solution, we implemented it in a post-optimization simulation consisting of 10000 i.i.d. randomly generated scenarios and collected the resulting statistics of overtime and waiting time. The major observations are as follows.

- Compared to the cost-based model, solutions from the chance-constrained model lead to less variable and more right-skewed overtime and waiting, but also lower in mean, and 50% and 95% quantiles.
- Compared with separating allocation and scheduling decisions, our integrated model produces solutions that attain a better tradeoff between cost and the quality of service.
- Compared with the standard chance-constrained model, the DR variant produces solutions that are more reliable: they are not any more sensitive to the underlying distributions. Therefore, the performance is no longer subject to assuming a correct distribution.
- The chance-constrained models and solution algorithms are applicable to a broader class of resource planning and scheduling problems involving other service systems, for which cost-based models are difficult to use.

CHAPTER 3

Solving Chance-Constrained 0-1 Programs with Decomposition and Parallelization

3.1 Introductory Remarks

Chance-constrained programs are in general non-convex and often solved or approximated by a scenario-based MIP reformulation, e.g., the SAA problem discussed in Section 1.1.2. In this chapter, we consider chance-constrained programs where decisions made before the realization of uncertainty are binary:

$$\min \quad f(x) \tag{3.1a}$$

$$\text{s.t.} \quad \mathbb{P}\{x \in \mathcal{X}(\xi)\} \geq 1 - \epsilon \tag{3.1b}$$

$$x \in X \subseteq \{0,1\}^d, \tag{3.1c}$$

where ξ is a random vector that follows a known distribution and has a finite set of realizations $\left\{\xi^1, \ldots, \xi^K\right\}$ from a Monte Carlo sample. We assume these scenarios are equal likely within this chapter. Set $\mathcal{X}(\xi) \subseteq \mathbb{R}^d$ is parameterized by ξ thus also random, and we define $\mathcal{X}_k := \mathcal{X}(\xi^k)$ for each scenario k. Scalar ϵ is a risk tolerance, typically small, that limits the likelihood of some undesirable outcome $x \notin \mathcal{X}(\xi)$. Function f maps the decision x into a real value that the model aims to minimize. We introduce binary variables $z_k \in \{0,1\}$ to indicate whether $x \in \mathcal{X}_k$ in each scenario k. The SAA reformulation of (3.1) is then written as:

$$\min \quad f(x) \tag{3.2a}$$

$$\text{s.t.} \quad z_k = 0 \Rightarrow x \in \mathcal{X}_k, \ \forall k = 1, \ldots, K \tag{3.2b}$$

$$\sum_{k=1}^{K} z_k \leq K' \tag{3.2c}$$

$$x \in X, \ z_k \in \{0,1\}, \ \forall k = 1, \ldots, K. \tag{3.2d}$$

where $\alpha \Rightarrow \beta$ in (3.2b) means event α occurs only if event β occurs. Scalar $K' := \lfloor \epsilon K \rfloor$, and the packing inequality (3.2c) is strengthened from $\sum_{k=1}^{K} (1/K \cdot z_k) \leq \epsilon$.

When $\mathcal{X}(\xi)$ represents the feasibility region of a two-stage stochastic program, e.g., (1.5), its realization $\mathcal{X}_k$ in each scenario k can be described as:

$$\mathcal{X}_k = \left\{ x \in \mathbb{R}^d : \exists y^k \in \mathbb{R}^{d'}_+ \text{ with } T^k x + W^k y^k \geq h^k \right\}. \tag{3.3}$$

where $T^k \in \mathbb{R}^{n \times d}$, $W^k \in \mathbb{R}^{n \times d'}$, and $h^k \in \mathbb{R}^n$. Constraint (3.2b) can then be expressed as explicit inequalities, e.g., (1.8c), as

$$T^k x + W^k y^k + z_k m^k \geq h^k, \ \forall k = 1, \ldots, K, \tag{3.4}$$

where each m^k is a constant vector with big components that may weaken the linear programming relaxation of model (3.2) and make the model difficult to solve. On the other hand, because of the packing inequality (3.2c) which structurally is a "row" made of variables across scenarios, the model does not have a block-angular structure. Thus decomposition algorithms for traditional stochastic programs cannot apply directly.

Many existing solution methods take advantage of the problem structure in the particular application to tighten the values of big-M coefficients (Qiu et al., 2014; Song and Luedtke, 2013; Song et al., 2014). However, when the number of scenarios is large, the model with strengthened coefficients may still be difficult to solve directly. Furthermore, the preparatory step of strengthening coefficients can be computationally burdensome. For more general problems, Luedtke et al. (2010) and Küçükyavuz (2012) develop strengthened formulations which can be solved faster for chance-constrained programs with only random right-hand-sides (i.e., $\mathcal{X}(\xi) = \{x : Tx \geq h(\xi)\}$). Beraldi and Bruni (2010) develop a specialized branch-and-bound method to handle chance-constrained programs without recourse decisions. Luedtke (2014) develops a branch-and-cut algorithm for general chance-constrained programs where the inequalities with big-M coefficients are not enforced explicitly, but rather through tight cuts generated during the algorithm. See Section 1.1.3 for a more comprehensive review of solution methods for chance-constrained programs.

In this chapter, we study dual decomposition methods which are combinations of Lagrangian relaxation and scenario decomposition. The main idea is to "clone" first-stage decisions and use NACs to ensure that their scenario-based copies are identical. Dualizing the NACs then leads to a Lagrangian relaxation of the problem which is decomposable into scenario-based subproblems. The subproblems are tractable and can be computed in a distributed system. This approach was first introduced by Rockafellar and Wets (1976). We refer to Shapiro et al. (2014) for other general results of applying dual decomposition to solving stochastic programs. Carøe and Schultz (1999) apply dual decomposition to obtain strong relaxations of two-stage stochastic integer programs,

and generate bounds used in a branch-and-bound framework. Dentcheva and Römisch (2004) analyze the Lagrangian relaxation of multistage stochastic programs with nonconvex constraints from logic and integrality requirements, and demonstrate the efficacy of scenario decomposition compared with other decomposition algorithms. Moreover, heuristic approaches, including variants of the "progressive hedging" algorithm (see Watson and Woodruff, 2011), are designed based on creating subproblems that can be decomposed by scenario and penalizing violations of NAC via updating the Lagrangian dual to achieve consensus among all the subproblems to seek feasible primal solutions.

Ahmed (2013) proposes a scenario decomposition variant for solving 0-1 stochastic programs with expectation-based objective functions. The key of the algorithm is to identify an optimal solution by iteratively bounding the optimal objective value and cutting off candidate solutions obtained from scenario subproblems. Ahmed (2013) describes serial and synchronous distributed implementations of the algorithm, and demonstrates the efficacy of the approach by solving instances from the SIPLIB test library (see Ahmed et al., 2015a). Ryan et al. (2015) extend the work of Ahmed (2013) by developing an asynchronous parallel implementation of the algorithm, which has better performance on the same set of test instances.

Two of the most relevant works related to this chapter are Watson et al. (2010) and Ahmed et al. (2015b), although they use different methods to decouple the scenarios. Watson et al. (2010) dualize both the NAC and the packing inequality (3.2c), and then use progressive hedging to compute the Lagrangian dual. These heuristics do not have convergence guarantees, but have good performance for solving stochastic programs in a variety of applications (see, e.g., Crainic et al., 2011, 2014). Ahmed et al. (2015b) dualize the NAC only, and compute the Lagrangian relaxation by solving a set of scenario subproblems and then a single-knapsack problem. In particular, they demonstrate the strength of the dual bounds and efficient ways of obtaining them in a distributed algorithm, although details of parallel implementation of the proposed algorithms are not discussed or further explored. In our attempt, we relax both constraints, and show that maximizing the Lagrangian relaxation function has a closed-form solution (see Section 3.2). This proposition enables us to replace some subgradient iterations with simple arithmetics, which substantially accelerates the algorithm as demonstrated by the computational results. In addition, we develop a subroutine of cut aggregation that exploits cropping inequalities to address the very large number of cuts generated during the algorithm.

Finally, we develop parallel schemes for implementing the proposed algorithm. Decomposition algorithms are generally amenable to parallelism. However, it is nontrivial to implement them in a distributed framework (Ahmed, 2013; Lubin et al., 2013; Ryan et al., 2015), especially if aiming for good parallel efficiency. Typically, decomposition algorithms are iterative procedures where each iteration involves solving a set of independent subproblems. Although solving subproblems

can easily be parallelized, the next iteration usually requires some combined result from all the subproblems. In parallel execution, this is a barrier where some processes have to wait for some others. Moreover, the communication traffic incurred from consolidating distributed results and the efficiency of a parallel implementation may deteriorate substantially as the number of processes increases. In this chapter, we explore two parallel schemes: one that simply distributes the computation of subproblems evenly across parallel processes, and one that adopts a Master-Worker structure where a master process acts as a centralized information keeper to avoid duplicate efforts. We observe speedup under both schemes, and the second scheme outperforms when the number of processes is large.

In Section 3.2, we present the dual decomposition algorithm and the enhancement techniques. In Section 3.4.3, we present the parallel schemes. In Section 3.4, we summarize the computational results on a set of resource allocation problem instances. In Section 3.5, we conclude the chapter.

3.2 Dual Decomposition

We generalize the algorithm from a scenario decomposition method proposed by Ahmed (2013) for 0-1 stochastic programs. It mainly follows a procedure of iteratively evaluating and cutting off candidate solutions discovered from scenario subproblems until the upper and lower bounds for the optimal objective value are sufficiently close. The subproblems are obtained from decomposing a Lagrangian relaxation problem. The value of the relaxation function is a valid lower bound, while the objective values of subproblem solutions that are verified feasible are valid upper bounds.

3.2.1 Lagrangian Relaxation

In model (3.2), to decouple scenarios we first make copies of variable x so that each scenario k as a copy x^k. We can then rewrite the model as:

$$\min \quad \sum_{k=1}^{K} f(x^k)/K \tag{3.5a}$$

$$\text{s.t. } z_k = 0 \Rightarrow x^k \in \mathcal{X}_k, \ \forall k = 1, \ldots, K \tag{3.5b}$$

$$\sum_{k=1}^{K} \alpha_k x^k = x^1 \tag{3.5c}$$

$$\text{(3.2c)}, \ x^k \in X, \ z_k \in \{0,1\}, \ \forall k = 1, \ldots, K \tag{3.5d}$$

where $\alpha_1, \ldots, \alpha_K$ are positive scalars summing to 1. Constraint (3.5c) is an NAC to ensure $x^1, \ldots, x^K$ taking the same values. There are various forms of NAC used in the literature. For example, (3.5c)

can be replaced by $x^k = x^1$, $\forall k = 2, \ldots, K$ as the common way to enforce all variable copies being the same (see, e.g., Ryan et al., 2015), which, however, involves $(K-1) \times d$ constraints as compared to the d constraints in (3.5c). To tighten the corresponding dual bound, as demonstrated in our preliminary experiments, takes a long time. Given binary valued $x^1, \ldots, x^K$, we formulate (3.5c) that can only be satisfied when all of the copies = 0, or all of them = 1. Next, we relax the two cross-scenario constraints (3.5c) and (3.2c), and introduce multiplier $\lambda \in \mathbb{R}^d$ and variable $\rho \in \mathbb{R}_+$ to form a Lagrangian relaxation of (3.5) as:

$$
\begin{aligned}
g(\rho, \lambda, S) := \min \quad & \sum_{k=1}^{K} f(x^k)/K + (\alpha_k - \delta_k)\lambda^\top x^k + \rho \left(\sum_{k=1}^{K} z_k - K' \right) \\
\text{s.t.} \quad & \text{(3.5b)} \\
& x^k \in X \setminus S,\ z_k \in \{0,1\},\ \forall k = 1, \ldots, K
\end{aligned}
$$

where $S \subseteq \{0,1\}^d$ represents a set of x solutions that have been excluded by cuts, and for notational convenience, we let $\delta_1 = 1$, and $\delta_k = 0$ for $k = 2, \ldots, K$.

Proposition 3.1. *Let $\mathcal{P}^*$ be the complete set of optimal solutions of* (3.5) *in the space of x. For any set $S \subseteq \{0,1\}^d$, if $\mathcal{P}^* \nsubseteq S$ then $g(\rho, \lambda, S)$ is a lower bound for the optimal objective value of* (3.5)*, for any $\rho \geq 0$ and λ.*

The proof, which is similar to the proof for weak duality, is shown in Appendix B.1. We can rearrange $g(\rho, \lambda, S)$ into a decomposable form as

$$
\begin{aligned}
g(\rho, \lambda, S) &= -K'\rho + \sum_{k=1}^{K} \begin{pmatrix} \min & \rho z_k + (\alpha_k - \delta_k)\lambda^\top x^k + f(x^k)/K \\ \text{s.t.} & \text{the } k^{\text{th}} \text{ constraint in (3.5b)} \\ & x^k \in X \setminus S,\ z_k \in \{0,1\} \end{pmatrix} \\
&= -K'\rho + \sum_{k=1}^{K} \min \left\{ \rho + h_1^k(\lambda, S), h_0^k(\lambda, S) \right\},
\end{aligned}
$$

where $h_1^k(\lambda, S)$ and $h_0^k(\lambda, S)$ correspond to the objective values of deactivating ($z_k = 1$) and activating ($z_k = 0$) the constraint $x^k \in \mathcal{X}_k$, respectively, in each scenario k. Explicitly,

$$
h_1^k(\lambda, S) := \min \left\{ (\alpha_k - \delta_k)\lambda^\top x^k + f(x^k)/K : x^k \in X \setminus S \right\}, \tag{3.6}
$$

$$
h_0^k(\lambda, S) := \min \left\{ (\alpha_k - \delta_k)\lambda^\top x^k + f(x^k)/K : x^k \in \mathcal{X}_k \cap X \setminus S \right\}. \tag{3.7}
$$

Proposition 3.2. *For any fixed vector λ and solution set S,*

$$\overline{g}(\lambda, S) = \max_{\rho \geq 0} \{g(\rho, \lambda, S)\} = \sum_{n=1}^{K-K'} h_0^{\sigma_n}(\lambda, S) + \sum_{n=K-K'+1}^{K} h_1^{\sigma_n}(\lambda, S), \tag{3.8}$$

where $\{\sigma_n\}_{n=1}^K$ is a permutation of scenarios $1, \ldots, K$ such that the values of

$$h_0^{\sigma_1}(\lambda, S) - h_1^{\sigma_1}(\lambda, S), \ldots, h_0^{\sigma_K}(\lambda, S) - h_1^{\sigma_K}(\lambda, S),$$

are nondecreasing.

Proof. Consider the monotonicity of $g(\rho, \lambda, S)$ over ρ:

- For $\rho \in [0, h_0^{\sigma_1}(\lambda, S) - h_1^{\sigma_1}(\lambda, S))$, $g(\rho, \lambda, S) = (K - K')\rho + \sum_{k=1}^K h_1^k(\lambda, S)$, which increases in ρ;
- For $\rho \in [h_0^{\sigma_K}(\lambda, S) - h_1^{\sigma_K}(\lambda, S), +\infty)$, $g(\rho, \lambda, S) = -K'\rho + \sum_{k=1}^K h_0^k(\lambda, S)$, which decreases in ρ;
- For $\rho \in [h_0^{\sigma_n}(\lambda, S) - h_1^{\sigma_n}(\lambda, S), h_0^{\sigma_{n+1}}(\lambda, S) - h_1^{\sigma_{n+1}}(\lambda, S))$ with some n,

$$g(\rho, \lambda, S) = (K - n - K')\rho + \sum_{k=1}^{n} h_0^{\sigma_k}(\lambda, S) + \sum_{k=n+1}^{K} h_1^{\sigma_k}(\lambda, S).$$

  Therefore, for $n \in \{1, \ldots, K - K' - 1\}$, it increases in ρ; for $n = K - K'$, it is a constant, and for $n \in \{K - K' + 1, \ldots, K - 1\}$, it decreases in ρ.

Thus, a maximizer is $\rho^* = h_0^{\sigma_{K-K'}}(\lambda, S) - h_1^{\sigma_{K-K'}}(\lambda, S)$, which yields the proposed maximum value of $g(\rho, \lambda, S)$ over $\rho \geq 0$. □

Intuitively, we interpret $h_0^k(\lambda, S) - h_1^k(\lambda, S)$ as the cost of activating the constraint $x \in \mathcal{X}_k$ in each scenario k. Since that constraint must be activated in at least $K - K'$ scenarios, we choose the $K - K'$ scenarios with the lowest cost. Proposition 3.2 is based on the assumption that all the scenarios have the same probability $1/K$, and can be extended for the general case when the probabilities are not necessarily the same (see, e.g., Ahmed et al., 2015b). It follows from Propositions 3.1 and 3.2 that $\overline{g}(\lambda, S)$ is a valid lower bound for the optimal objective value of the original model (3.5).

3.2.2 Bound-and-Cut Algorithm

We summarize the dual decomposition algorithm in Algorithm 3.1. During the algorithm, we track the best-found lower bound ℓ and upper bound u, as well as a solution $\bar{x}$ where u is obtained, i.e., $f(\bar{x}) = u$. A good initialization for u and $\bar{x}$ can be obtained from solving the robust counterpart of

the original problem:

$$\min\left\{f(x) : x \in X \bigcap_{k=1}^{K} \mathcal{X}_k\right\}, \tag{3.9}$$

where we enforce the chance constraint for all scenarios. Any solution $\hat{x}$ obtained from computing $h_0^k(\lambda, S)$ or $h_1^k(\lambda, S)$, if satisfying

$$\sum_{k=1}^{K} \mathbf{1}(\hat{x} \in \mathcal{X}_k) \geq K - K', \tag{3.10}$$

is feasible to the original chance-constrained problem. If $f(\hat{x}) < u$, we update u and the best-found solution $\bar{x}$ accordingly. On the other hand, the lower bound ℓ is updated by the value of $\overline{g}(\lambda, S)$. The algorithm iteratively cuts off explored solutions and improves the bounds (e.g., increasing ℓ or decreasing u) until the gap between ℓ and u is sufficiently small, e.g., $< \epsilon_{\text{BND}}$ where ϵ_{BND} is a small positive scalar.

Now that a relatively tight lower bound in each iteration is the value of the Lagrangian dual given by:

$$\max_{\rho \geq 0, \lambda} \quad g(\rho, \lambda, S) \tag{3.11}$$

$$= \max_{\lambda} \quad \overline{g}(\lambda, S), \tag{3.12}$$

the algorithm in each iteration runs a subroutine of the subgradient method to approach (3.12). The subroutine presents as an inner loop that iteratively updates λ. For clarification, we refer to the inner loop as the subgradient iterations and the outer loop as the bound-and-cut iterations. In preliminary experiments, we identify the Polyak rule (Polyak, 1977) that produces the most efficient step size for updating λ on our test instances. We let λ^n denote the λ in the n^{th} subgradient iteration. The rule specifies the step size for moving from λ^n to λ^{n+1} as:

$$s_{n+1} = (u - \overline{g}(\lambda^n, S))/\|r(\lambda^n)\|_2^2, \tag{3.13}$$

where $r(\lambda^n)$ is the subgradient of λ^n with respect to the function $\overline{g}(\lambda^n, S)$. Then,

$$\lambda^{n+1} = \lambda^n - s_{n+1}\alpha(\lambda^n). \tag{3.14}$$

We stop the subgradient method if the improvement in the value of $\overline{g}(\lambda, S)$ has been sufficiently

small in a certain number of consecutive iterations, e.g.,

$$0 \leq \overline{g}(\lambda^i, S) - \overline{g}(\lambda^{i-1}, S) \leq \epsilon_{\text{SBG}}, \ \forall i = n - \Delta, \ldots, n \tag{3.15}$$

where $\epsilon_{\text{SBG}} \in \mathbb{R}_{++}$ and $\Delta \in \mathbb{Z}_{++}$.

Another feature of the algorithm is that every explored solution is discarded from future consideration. Considering their binary nature, this is accomplished by adding no-good cuts (1.19) to the scenario-based subproblems (3.6) and (3.7) which we use to compute $h_1^k(\lambda, S)$ and $h_0^k(\lambda, S)$ for $k = 1, \ldots, K$. The original feasible region is within a 0-1 hypercube and thus finite, whereas the new feasible region becomes strictly smaller in each bound-and-cut iteration. We also set a limit on the maximum number of iterations performed in each subgradient subroutine. Therefore the algorithm terminates finitely.

Algorithm 3.1 A dual decomposition algorithm for solving model (3.5)

$\ell \leftarrow -\infty$, $u \leftarrow$ the optimal objective value of (3.9), $\bar{x} \leftarrow$ an optimal solution of (3.9)
$S \leftarrow \emptyset$
repeat
  $\lambda \leftarrow 0$
  repeat
    for $k = 1, \ldots, K$ **do**
      $h_k^1 \leftarrow h_k^1(\lambda, S)$, and keep the optimal solution of (3.6) as $\hat{x}^k(1)$
      $h_k^0 \leftarrow h_k^0(\lambda, S)$, and keep the optimal solution of (3.7) as $\hat{x}^k(0)$
    end for
    sort scenarios $1, \ldots, K$ into permutation σ such that $h_{\sigma_1}^0 - h_{\sigma_1}^1 \leq \cdots \leq h_{\sigma_K}^0 - h_{\sigma_K}^1$.
    calculate $\overline{g}(\lambda, S)$ based on (3.8)
    update λ based on (3.13) and (3.14)
  until the stop condition (3.15) is satisfied
  $\ell \leftarrow \max\{\ell, \overline{g}(\lambda, S)\}$
  $S \leftarrow S \bigcup_{k=1}^{K} \{\hat{x}^k(1), \hat{x}^k(0)\}$.
  for $\hat{x} \in S$ and satisfying (3.10) **do**
    if $f(\hat{x}) < u$ **then**
      $u \leftarrow f(\hat{x})$, $\bar{x} \leftarrow \hat{x}$.
    end if
  end for
until $u - \ell \leq \epsilon_{\text{BND}}$

3.2.3 Cut Aggregation

Having learned that the subgradient method sometimes goes through excessively many iterations, which results in a large number of cuts (i.e., a huge set S) that overwhelm the solution of scenario

subproblems (3.6) and (3.7), we need to explore cropping inequalities (Angulo et al., 2014b; Lee, 2003) for 0-1 hypercubes to form stronger cuts and reduce the number of inequalities that are effectively appended to the subproblems.

In general, cropping inequalities take the general form of

$$\text{CPI}(U,V): \quad \sum_{j\in U}(1-x_j)+\sum_{j\in V}x_j \geq 1 \tag{3.16}$$

where U and V are disjoint subsets of $\{1,\ldots,d\}$. To cut off a single solution $\hat{x}$, we use the no-good cut (1.19), which is a special case of (3.16) with $U = \{j \in \{1,\ldots,d\} : \hat{x}_j = 1\}$ and $V = \{j \in \{1,\ldots,d\} : \hat{x}_j = 0\}$. Given any two cropping inequalities $\text{CPI}(U_1,V_1)$ and $\text{CPI}(U_2,V_2)$, if there exists some $j \in \{1,\ldots,d\}$ such that $U_1 = U_2 \cup \{j\}$ and $V_2 = V_1 \cup \{j\}$, we can easily aggregate them to obtain $\text{CPI}(U_2,V_1)$. In this case, we call either of the cuts being aggregated a *match* of the other.

In applying Algorithm 3.1, Step 15 involves appending cuts to every scenario subproblem (3.6) and (3.7) to cut off solutions that are newly added into the set S. The addition of any single cut can induce a series of aggregation recursively. To accelerate this procedure, we design a specialized type of container for cuts: $\mathcal{G}(u,v)$, with $u,v \in \{0,\ldots,d\}$ and $u+v \leq d$. Each set $\mathcal{G}(u,v)$ keeps cuts in the form of $\text{CPI}(U,V)$ with $|U| = u$ and $|V| = v$. Given any arbitrary cut $\text{CPI}(U,V)$, its match must be from $\mathcal{G}(|U|+1,|V|-1)$ or $\mathcal{G}(|U|-1,|V|+1)$. Therefore, we do not scan through all the existing cuts but only search within these two sets for a match.

We present the detailed steps of cutting off some solution $\hat{x}$, i.e., $S \leftarrow S \cup \{\hat{x}\}$, with cut aggregation in Algorithm 3.2. There could be multiple cuts available at a time that can be aggregated with a certain cut, but they may lead to different maximum aggregation depth (the number of times we aggregate cuts until no more aggregation is available). Desiring to increase the maximum aggregation depth, we design a heuristic that alternates the search for a match between $\mathcal{G}(|U|+1,|V|-1)$ and $\mathcal{G}(|U|-1,|V|+1)$. If neither has a match for the current $\text{CPI}(U,V)$, we stop the search and proceed to add $\text{CPI}(U,V)$ to all the subproblems.

3.3 Parallel Implementation Schemes

Decomposition methods contain parallel jobs such as solving subs or individual scenarios, and evaluating the objective value for individual solutions. While parallel processing saves time, the procedure also reaches a point when exchange or consolidation of the results occurs in order to create jobs for the next iteration. In practice, these points are quite a challenge for parallel implementation, considering the inefficiency of synchronization and the intensity of communication. In addition to the dual decomposition method considered in this chapter, other decomposition methods also feature this scheme. Thus, it is important to develop an efficient parallel scheme.

Algorithm 3.2 The procedure of cutting off $\hat{x}$ (i.e., $S \leftarrow S \cup \{\hat{x}\}$) with cut aggregation

1: $U \leftarrow \{j \in \{1,\ldots,d\} : \hat{x}_j = 1\}$, $V \leftarrow \{j \in \{1,\ldots,d\} : \hat{x}_j = 0\}$
2: NOT_FOUND $\leftarrow$ False
3: **for** any cut CPI(U', V') in $\mathcal{G}(|U|+1, |V|-1)$ **do**
4: **if** $\exists j \in \{1,\ldots,d\}$ such that $U' = U \cup \{j\}$ and $V = V' \cup \{j\}$ **then**
5: remove CPI(U', V') from $\mathcal{G}(|U|+1, |V|-1)$
6: $V \leftarrow V'$, NOT_FOUND $\leftarrow$ False
7: go to Step 15
8: **end if**
9: **end for**
10: **if** (NOT_FOUND) **then**
11: go to Step 27
12: **else**
13: NOT_FOUND $\leftarrow$ True
14: **end if**
15: **for** any cut CPI(U', V') in $\mathcal{G}(|U|-1, |V|+1)$ **do**
16: **if** $\exists j \in \{1,\ldots,d\}$ such that $U = U' \cup \{j\}$ and $V' = V \cup \{j\}$ **then**
17: remove CPI(U', V') from $\mathcal{G}(|U|+1, |V|-1)$
18: $U \leftarrow U'$, NOT_FOUND $\leftarrow$ False
19: go to Step 3
20: **end if**
21: **end for**
22: **if** (NOT_FOUND) **then**
23: go to Step 27
24: **else**
25: NOT_FOUND $\leftarrow$ True
26: **end if**
27: add CPI(U, V) to (3.6) and (3.7) for scenario $k = 1, \ldots, K$.

Most studies use a straightforward synchronous approach, which places a synchronization barrier after solving the jobs and before reiteration (e.g., Ahmed, 2013; Lubin et al., 2013; Nielsen and Zenios, 1997). However, when the time variation in the jobs is big, processors will sit idle for long periods, which is termed load imbalance. To alleviate load imbalance, other studies suggest assigning a single processor, termed the master, to collect and compile distributed information, and decide the next step (see, e.g., Birge et al., 1996; Ruszczyński, 1993; Ryan et al., 2013). In assigning jobs, instead of following a static rule, e.g., all subproblems of scenario 1 go to Processor 1, all subproblems of scenario 2 go to Processor 2, etc., the master queues the jobs and assigns them to processors as they become available. To improve load balance, another approach is force reiteration. Once a certain percentage of parallel jobs are are completed, the next iteration immediately begins, i.e., no waiting (Linderoth and Wright, 2003).

In this section, we develop two parallel schemes for implementing Algorithm 3.1 in a dis-

tributed framework. We refer to the first as *Basic Parallel*, where we simply divide the computation of subproblems evenly across all available parallel processes. It is a common choice in the literature for parallelizing decomposition methods (Ahmed, 2013; Lubin et al., 2013; Nielsen and Zenios, 1997). Figure 3.3a is a schematic view of using four processes to solve a problem with

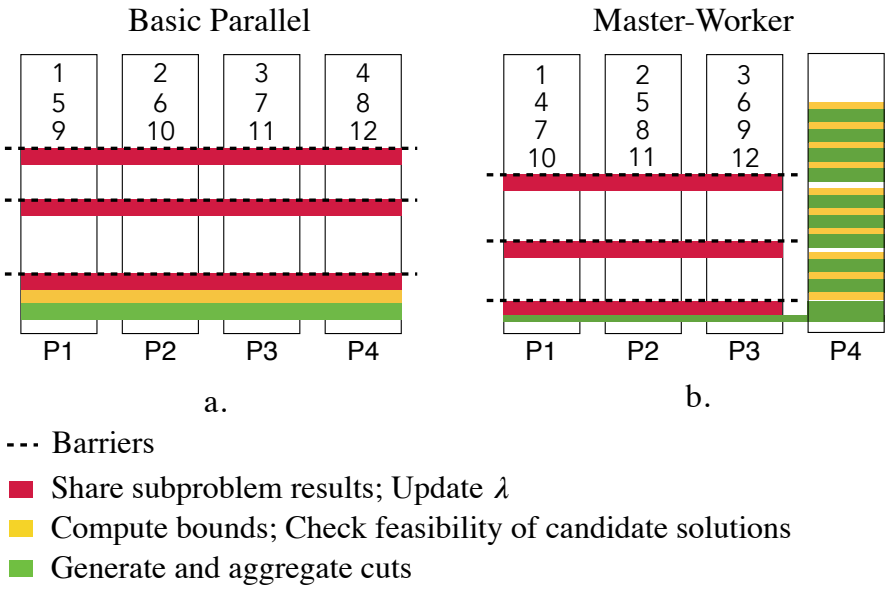

Figure 3.1: A schematic view of using four processes to solve a 12-scenario problem

scenarios $1,\ldots,12$. The charts in the schematic show one bound-and-cut iteration (Steps 3-21 of Algorithm 3.1) nesting three subgradient iterations (Steps 5-13 of Algorithm 3.1). In each subgradient iteration, we assign subproblems to the processes in a round-robin manner (such that (3.6) and (3.7) of each scenario k are assigned to the $((k-1) \bmod N+1)^{\text{th}}$ process. At the end of each subgradient iteration there is a synchronization barrier. The processes all share the subproblem computational results with one another, individually update the dual multiplier λ, and start the next subgradient iteration. At the end of each bound-and-cut iteration, every process individually checks the feasibility of every solution, which can be as much as $2K$ times the number of subgradient iterations. Generating and aggregating the cuts follows. In fact, the result of a subproblem is ready as soon as it is solved. In this scheme, information is not processed until the end of a bound-and-cut iteration. Below, we explain an alternative method.

We refer to the second scheme as *Master-Worker Parallel*. In this scheme, the master updates bounds, generates and aggregates the cuts, and passes them to all of the workers at the end of every bound-and-cut iteration. Load balance and parallel efficiency are achieved because synchronicity by the master is not required.

3.4 Computational Results

We implement parallelization by OpenMPI 1.6 (Gabriel et al., 2004), and perform the computation on Flux HPC cluster, University of Michigan. We use as many as 9 compute nodes; each compute node has twelve 2.67 GHz Intel Xeon X5650 processes and 48GB RAM. All involved optimization models (including subproblems as a result of decomposition) are solved by CPLEX 12.6 via ILOG Concert Technology.

In Section 3.4.1, we discuss test instances and experiment setup. In Section 3.4.2, we present the computational results of implementing our algorithm in serial, show how much the proposed techniques help accelerating the computation, and how efficient our algorithm is compared with existing methods. In Section 3.4.3, we present the results of parallel implementation.

3.4.1 Instances and Experimental Setup

We set up two sets of test instances. based on a resource allocation problem instance from SIPLIB (Ahmed et al., 2015a). We formulate the problem as

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Tx \geq h \\ & \mathbb{P}\left\{(a^n(\xi))^\top x \geq b_n(\xi),\ \forall n = 1,\ldots,N\right\} \geq 1-\epsilon \\ & x \in \{0,1\}^d, \end{aligned}$$

where T and h are the properly-sized deterministic matrix and vector, respectively. Inside the chance constraint is a N-row linear inequality system that varies across scenarios. In particular, $a^n(\xi^k) \in \mathbb{R}^d_+$ and $b_n(\xi^k) \in \mathbb{R}_+$ for each row $n \in \{1,\ldots,N\}$ and in each scenario $k \in \{1,\ldots,K\}$. In both sets of instances, $d = 20$, $K = 200$ and $\epsilon = 0.1$. The first set has one inequality in the chance constraint (i.e., $N = 1$) and the second set has three (i.e., $N = 3$). We refer to these two instance sets as RAS1 and RAS3. Each set consists of three instances m_1, m_2, and m_3, where m = RAS1 or RAS3.

To implementing the algorithm, we use $\alpha_1,\ldots,\alpha_K = 1/K$ in forming the NAC (3.5c). We attempt to solve all of the instances to optimum, i.e., we set the tolerance for the gap between bounds $\epsilon_{\text{BND}} = 0$. In the subgradient method, preliminary experiments suggest using $\epsilon_{\text{SBG}} = 0.5$.

3.4.2 Results of Serial Implementation

The *default* serial scheme directly calls an off-the-shelf solver to solve the SAA reformulation of the chance-constrained program. The three schemes for implementing the proposed dual decom-

Table 3.1: Comparison of the four schemes in serial computational time (or optimality gap)

		RAS1			RAS3		
		_1	_2	_3	_1	_2	_3
default	tot_time	(10.6%)	(11.3%)	(11.2%)	(17.0%)	(12.5%)	(10.2%)
DDG	tot_time	(8.6%)	5657	2976	6447	6262	5508
	#_iter	-	4	4	3	4	4
DDM	tot_time	980	1044	977	4234	4251	4513
	#_iter	4	4	4	3	3	3
DDA	tot_time	824	886	927	3832	3546	3338
	#_iter	4	4	4	3	3	3

position algorithm in serial are:

- DDG: we use a fundamental approach which does not apply Proposition 3.2 and thus has to update both ρ and λ in the subgradient method for computing the Lagrangian dual (3.11). In this case, we also use the Polyak rule to update the step size for ρ, which initially is set to zero.

- DDM: we use Algorithm 3.1. It is built upon Proposition 3.2 which presents a closed-form result of the maximum of $g(\rho,\lambda,S)$ over $\rho \geq 0$. As a result, the subgradient method only needs to update λ.

- DDA: we implement Algorithm 3.1 with the subroutine of cut aggregation presented in Algorithm 3.2.

Table 3.1 reports the serial runtime, in seconds, of the four schemes in *tot_time*. If an instance is not solved to optimum within the runtime limit of two hours, we present the optimality gap with parentheses in *total_time* and omit the other computational details. For the three decomposition schemes, we also present the number of bound-and-cut iterations needed for the upper and lower bounds to converge in *#_iter*.

We fail to solve any instance to optimum under the default scheme, whereas the dual decomposition algorithms solve all, except that DDG fails on RAS1_1. We observe the following runtime trend:

$$\texttt{DDG} > \texttt{DDM} > \texttt{DDA}.$$

Note that the three decomposition schemes require no more than 5 iterations. The enhancement techniques applied in DDM and DDA do not have a general effect on the iteration count, but do improve the runtime effectively on all of the instances. In particular, the impact of avoiding updating ρ in the subgradient method is significant, since, as we switch from DDG to DDM, the drops in tot_time are substantial.

Table 3.2: Other computational details of the three dual decomposition schemes

		RAS1			RAS3		
		_1	_2	_3	_1	_2	_3
DDG	#_SBG_iter	-	44	33	41	28	30
DDM	#_SBG_iter	13	14	13	18	18	18
DDA	#_aggregation	83	65	71	49	30	32
	#_no-good_cuts	1563	2598	2483	5525	6120	5707

Table 3.2 presents some other details that enable us to get a closer look. For example, in *#_SBG_iter*, it shows the accumulated number of subgradient iterations (i.e., the inner loop) that DDG and DDA have gone through. To approach $\max_{\rho:\rho\geq 0} g(\rho,\lambda,S)$, DDG uses the subgradient method which involves iterations of solving a series of subproblems, whereas DDM employs the closed-form result presented in Proposition 3.2 which is simple arithmetic. Therefore, DDM saves time. In *#_aggregation*, Table 3.2 shows the number of aggregation DDA has performed as compared to the total number no-good cuts originally generated in *#_no-good_cuts*. Only a small portion of cuts are aggregated, which still leads to notable improvement of runtime in all instances.

Given that DDA is the fastest, we use it to compare the existing methods and to test the parallel schemes. To compare, we select the following four methods that are commonly used in solving chance-constrained programs. The first two focus on tightening the big-M coefficients, and then call the solver to compute the strengthened MIP reformulation. Their runtimes consist of the time computing the MIP reformulation and the time computing the strengthened coefficients. The other two approaches are based on the branch-and-cut algorithm proposed by Luedtke (2014) for solving general chance-constrained programs. The four approaches are:

- $\texttt{MIP}_{\text{iter}}$: The big-M coefficients are strengthened through an iterative method proposed by Qiu et al. (2014), the first coefficient-strengthening approach detailed in Section 2.3.2.

- $\texttt{MIP}_{\text{scen}}$: The big-M coefficients are strengthened through a scenario decomposition method introduced in Luedtke (2014) and Song et al. (2014), the second coefficient-strengthening approach detailed in Section 2.3.2.

- BNC*s*: This algorithm is based on a master problem in the space of x and z, in which the chance constraint is initially relaxed. Given any specific $(\hat{x},\hat{z})$, if for some scenario k with $\hat{z}_k < 1$, $x \notin \mathcal{X}_k$, then a valid inequality with respect to $\mathcal{X}_k$ is lifted over x to become globally valid, and subsequently used as a cut to exclude the current solution. In this scheme, once such a cut is found, it is added to the master problem, and immediately the algorithm fathoms the underlying node and picks another in the branch-and-bound tree. This is usually referred to as a single-cut strategy.

Table 3.3: Comparison of DDA and existing methods in runtime

	RAS1			RAS3		
	_1	_2	_3	_1	_2	_3
DDA	824	886	927	3832	2546	3338
BNC*s*	773	1169	1019	5527	2720	4128
BNC*m*	-	6757	5164	-	-	-
$\texttt{MIP}_{\text{scen}}$	3832	-	3213	-	-	-
$\texttt{MIP}_{\text{iter}}$	4234	6144	2775	-	-	5002

- BNC*m*: This scheme scans the scenarios to identify all possible cuts before adding them to the master problem and switching to process other nodes. This is usually referred to as a multi-cut strategy.

Table 3.3 presents the runtime in seconds. An entry of "-" indicates that the particular instance is not solved to optimum under the particular scheme within the two-hour time limit. Based on the tested instances, DDA and BNC*s* are the fastest two, and DDA is faster in five out of six instances.

3.4.3 Results of Parallel Implementation

We capture the parallel time as the walltime that elapses from the start of the parallel program to the end. For each run we compute *speedup* as the ratio of the serial runtime to the parallel time. We parallelize scheme DDA and test it on all the instances. Figure 3.2 shows how the results of speedup change as we vary the number of processes. Each subfigure presents the results for one particular instance. The vertical and horizontal axes represent "speedup" and "number of processes", respectively. We scale them equally, so the main diagonal (in grey) gives the trending for a perfect parallelism where speedup is equal to the number of processes. We test using 2, 3, 5, and 9 processes. The red and the blue curves are the results of Basic Parallel and Master-Worker Parallel, respectively.

As the number of processes increase, most of the cases scale well in the beginning, but the speedup soon starts to flatten out, or even drops down. Recall that both schemes contain barriers in each iteration as well as collective communication steps in which every processes talks to every other. As the number of processes increase, the communication workload grows as does the waiting caused by the barriers. These drawbacks start to cancel out the advantage brought by increasing processes, i.e., performance "plateaus." This is a common phenomenon in parallel computing. However, a special observation here is that we see in all cases a crossover of the red curve and the blue curve: the red always starts climbing faster, but plateaus earlier. This is probably because Basic Parallel has one more process sharing the computation of subproblems, which is an important advantage when the number of processes is still small. However, without a centralized

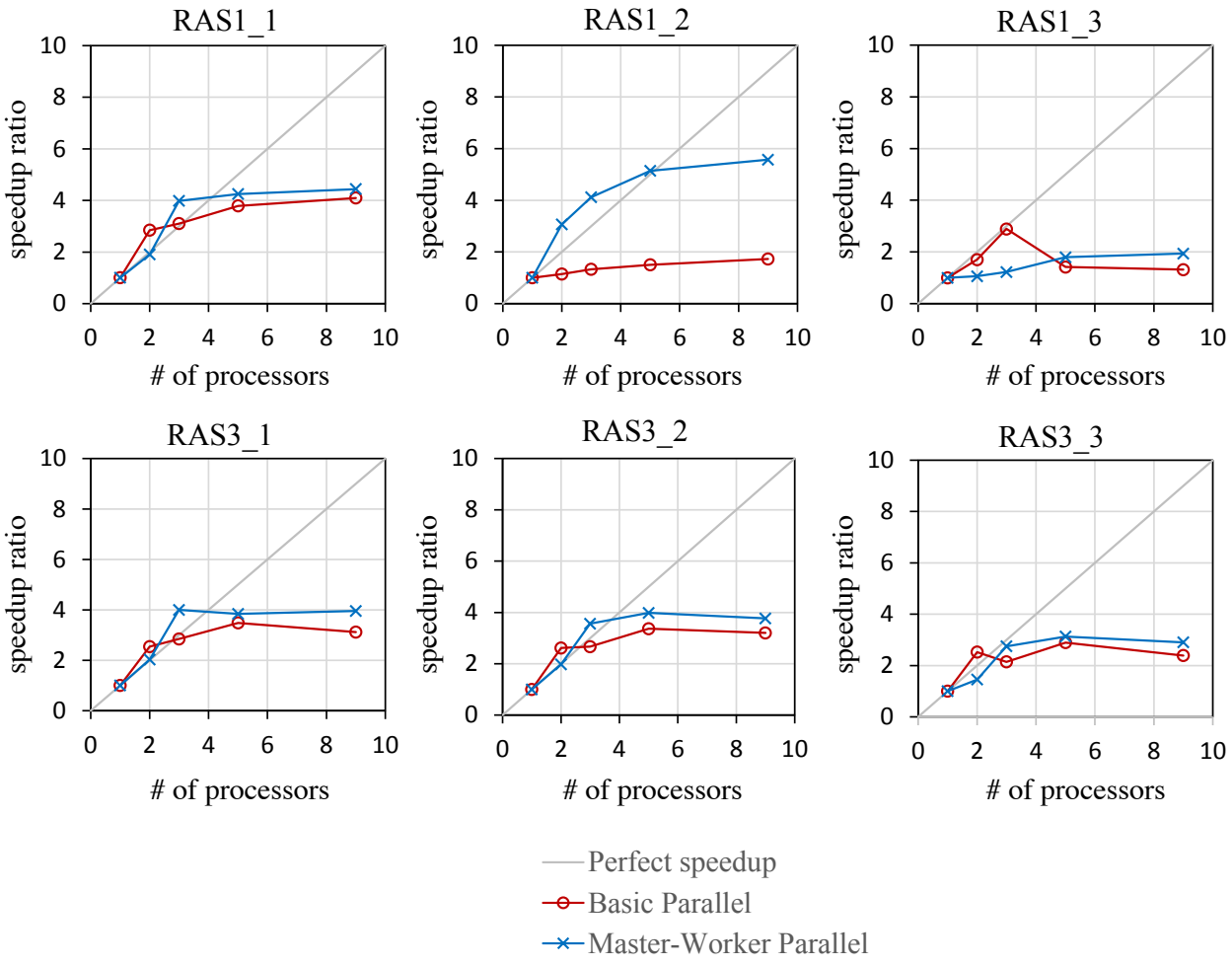

Figure 3.2: Speedup versus number of processes under the parallel implementation of DD1

"keeper" of candidate solutions, duplicate solutions cannot be identified, which requires some of the processes to spend time reevaluating them. Furthermore, every process has to perform the cut aggregation step on its own, which is again a duplicate effort. Table 3.4 shows that the proportion of unproductive time among the total time grows larger as the number of processes increase, parallel efficiency also deteriorates. The maximum parallel efficiency is attained with two processes under Basic Parallel, and with three processes under Master-Worker Parallel.

3.5 Concluding Remarks

In this chapter, we studied general chance-constrained 0-1 programs. We developed a decomposition algorithm which followed a procedure of iteratively evaluating and cutting off candidate solutions discovered from scenario subproblems until the upper and lower bounds were sufficiently close. The algorithm was based on a Lagrangian relaxation obtained from dualizing the NAC and

Table 3.4: Parallel efficiency under the two parallel schemes

		RAS1			RAS3		
	num of processes	_1	_2	_3	_1	_2	_3
Basic Parallel	2	142%	57%	85%	127%	131%	126%
	3	103%	44%	96%	95%	89%	71%
	5	76%	30%	28%	70%	67%	58%
	9	45%	19%	15%	35%	36%	27%
Master-Worker Parallel	2	95%	154%	53%	101%	99%	73%
	3	133%	137%	41%	133%	119%	92%
	5	85%	103%	36%	77%	80%	63%
	9	49%	62%	22%	44%	42%	32%

the packing inequality emerging in the MIP reformulation of the original program. Specifically,

- We proposed a closed-form result about the maximization of the Lagrangian relaxation in Proposition 3.2 which simplified some subgradient iterations to simple arithmetics. Practically, it effectively reduced the number of subproblems to solve;
- We developed a subroutine of cut aggregation that exploits cropping inequalities, to address the issue of a overwhelmingly large number of generated cuts. Practically, it accelerated the solution of individual subproblems.

To study the impact of these techniques, we tested three algorithms that incorporate neither, the first, and both of them, respectively. We also developed parallel schemes to further enhance the efficacy of the proposed algorithm:

- Basic Parallel, where the computation of subproblems are shared evenly by all the processes;
- Master-Worker Parallel, where a master process acts as a centralized information keeper to avoid duplicate efforts.

We conducted numerical experiments on resource allocation problem instances with 20-dimensional decision variable and 200 scenarios. The findings are summarized as follows:

1. In solving the MIP reformulation of the original program, our algorithm is much faster than directly calling the solver. It is also faster than the four existing methods that are widely used in solving large-scale chance-constrained programs.
2. Both Proposition 3.2 and cut aggregation improve the efficacy of the dual decomposition method. In particular, the former reduces the number of subgradient iterations and saves substantial time.

3. When the number of processes is small, Basic Parallel has better speedup. As the number of processes increases, Master-Worker Parallel surpasses it, because the master eliminates duplicate efforts between processes.

4. As the number of processes varies, the maximum parallel efficiency is achieved with two processes under Basic Parallel, and three processes under Master-Worker Parallel.

CHAPTER 4

Solving Risk-Averse 0-1 Stochastic Programs with Decomposition and Parallelization

4.1 Introductory Remarks

In addition to chance-constrained programs, there is another big family of risk-averse stochastic programs that emerge naturally from real-world practice. Different from controlling the risk to a certain extent as in a chance constraint, these problems aim to suppress the risk as much as possible, and formulate an objective such as minimizing the risk of some random outcome. Two studies are particularly relevant. Miller and Ruszczyński (2011) consider a risk-averse two-stage stochastic linear program, in which the objective function is given as a composition of conditional risk measures. They develop two decomposition algorithms, one of which uses a generic cutting plane approach and the other exploits the composite structure of the objective function. Collado et al. (2012) propose a scenario decomposition algorithm for a risk-averse multistage stochastic problem. Their algorithm is based on constructing risk-neutral approximations of the program by exploiting specific structure of dynamic risk measures. Both studies, however, do not consider stochastic integer programs in a risk-averse setting, or explore detailed procedures for implementing the decomposition algorithms in parallel. In this chapter, we consider a class of risk-averse stochastic 0-1 programs (see Ruszczyński, 2013) in which the risk is measured by a coherent risk function. See Pflug and Römisch (2007); Rockafellar and Uryasev (2002); Rockafellar et al. (2006) for a more extensive discussion of coherent risk functions.

Following Shapiro (2012), we represent the coherent risk function in a dual form and arrive at an equivalent risk-neutral minimax reformulation of the problem. We then develop three dual decomposition algorithms for solving the reformulation. Two of them have the bound-and-cut iteration nesting an iterative subroutine (e.g., a cutting-plane method or a subgradient method), for optimizing the Lagrangian relaxation to produce a tighter lower bound, and the third one uses the functional value of Lagrangian relaxation at zero as the lower bound.

Thees algorithms are also applicable to solving the DR variant of the considered risk-averse 0-1 program, and can be implemented in a distributed system. Indeed, decomposition algorithms are amenable to parallelism, which can effectively save computational time (Ahmed, 2013; Ryan et al., 2015). However, it is nontrivial to implement decomposition algorithms as parallel procedures in a distributed framework. Recently, Ahmed (2013) proposes a decomposition method for solving risk-neutral stochastic 0-1 programs. The algorithm also searches for solutions by bounds and cuts, which can be developed through scenario subproblems. The author describes serial and synchronous distributed implementations of the algorithm, and demonstrates the efficacy of the approach on instances from the SIPLIB test library (see Ahmed et al., 2015a). Ryan et al. (2015) then extend the work and develop an asynchronous parallel implementation of the algorithm, which has better performance on the same set of test instances. On the other hand, Ahmed et al. (2015b) develop decomposition approaches for chance-constrained 0-1 programs. In particular, they demonstrate the strength of the dual bounds and efficient ways of obtaining them in a distributed algorithm, although details of parallel implementation are not discussed. In this chapter, we develop three parallel schemes for the proposed decomposition algorithms. They represent different combinations of basic-parallel/master-worker-parallel, synchronous/asynchronous, and push/pull, and enable a comprehensive examination of the parallel execution of the decomposition algorithms.

The remainder of the chapter is organized as follows. Section 4.2 presents a generic model for the considered risk-averse 0-1 stochastic programs, an equivalent risk-neutral minimax reformulation, and the three dual decomposition algorithms. Section 4.4 introduces the parallel schemes. Section **??** presents a DR variant and explains how the algorithms adapt. Section 4.5 summarizes the computational results. Section 4.6 concludes.

4.2 Decomposition Methods

4.2.1 Problem Formulation

We consider risk-averse stochastic 0-1 programs of the form

$$\min_{x} \left\{ \rho(f(x,\xi)) : x \in X \subseteq \{0,1\}^d \right\}, \tag{4.1}$$

where x is a d-dimensional binary decision vector, ξ is a multivariate random vector, and the cost function $f(x,\xi)$ is convex in x for any realized value of ξ. We consider a finite set of realizations $\{\xi^1,\ldots,\xi^K\}$ of ξ having the corresponding probabilities $p_1,\ldots,p_K$. The vector $p = (p_1,\ldots,p_K)^\top$

belongs to the polyhedron set

$$\mathcal{M} := \left\{ p = (p_1, \ldots, p_K)^\top : \sum_{k=1}^{K} p_k = 1,\ p_k \geq 0,\ \forall k = 1, \ldots, K \right\}. \tag{4.2}$$

Here $\rho(\cdot)$ is a generic measure that returns a scalar metric of the random cost $f(x, \xi)$. For example, $\rho(\cdot)$ can be the expectation as in a risk-neutral model, or a specific risk measures as in a risk-averse model. We consider $\rho(\cdot)$ as a coherent risk measure, which leads to an equivalent dual representation (see, e.g., Artzner et al., 1999; Shapiro and Ahmed, 2004) as,

$$\rho(f(x, \xi)) = \max_{q \in \mathcal{Q}_\rho(p)} \left\{ \mathbb{E}_q \left[f(x, \xi) \right] \right\}, \tag{4.3}$$

where we compute the expectation of the random $f(x, \xi)$ based on an unknown probability vector $q = (q_1, \ldots, q_K)^\top$, which belongs to an uncertainty set $\mathcal{Q}_\rho(p) \subseteq \mathcal{M}$ determined by the coherent risk measure ρ and nominal probabilities $p_1, \ldots, p_K$. In (4.3), the value of $\rho(f(x, \xi))$ is equivalent to the worst-case expectation of the random $f(x, \xi)$ measured based on any probability $q \in \mathcal{Q}_\rho(p)$. Therefore, we present an equivalent minimax reformulation of the original problem (4.1):

$$\min_{x \in X} \max_{q \in \mathcal{Q}_\rho(p)} \left\{ \sum_{k=1}^{K} q_k f_k(x) \right\} \tag{4.4}$$

where we use $f_k(x) := f(x, \xi^k)$ for notational convenience. In this chapter, we assume that $\min_{x \in X} \{ f_k(x) \}$ is bounded for all $k = 1, \ldots, K$. Our goal is to optimize model (4.4) via scenario decomposition approaches and efficient implementation schemes.

4.2.2 Dual Decomposition Framework

Following the standard dual decomposition method, we first replace variable x with scenario-based copies $x^1, \ldots, x^K$ to build a decomposable structure. Model (4.4) then reads as:

$$\begin{aligned} \min_{x^1, \ldots, x^K \in X} \max_{q \in \mathcal{Q}_\rho(p)} \quad & \sum_{k=1}^{K} q_k f_k(x^k) \\ \text{s.t.} \quad & \sum_{k=1}^{K} \alpha_k x^k = x^1 \end{aligned} \tag{4.5}$$

where $\alpha_1, \ldots, \alpha_K$ are positive scalars that sum to 1, and (4.5) is the NAC to ensure $x^1, \ldots, x^K$ taking the same values. Next, we relax (4.5) and introduce a dual multiplier $\lambda \in \mathbb{R}^d$. The resulting

Lagrangian relaxation reads as:

$$\min_{x^1,\ldots,x^K\in X}\max_{q\in\mathcal{Q}_\rho(p)}\left\{\sum_{k=1}^{K}\Big((\alpha_k-\delta_k)\lambda^\top x^k+q_k f_k(x^k)\Big)\right\} \tag{4.6}$$

where for notational convenience, we let $\delta_1 = 1$ and $\delta_k = 0$ for $k = 2,\ldots,K$. To recover scenario subproblems, we consider a lower approximation of (4.6) given by:

$$\begin{aligned} g(\lambda) &:= \max_{q\in\mathcal{Q}_\rho(p)}\min_{x^1,\ldots,x^K\in X}\left\{\sum_{k=1}^{K}\Big((\alpha_k-\delta_k)\lambda^\top x^k+q_k f_k(x^k)\Big)\right\} \\ &= \max_{q\in\mathcal{Q}_\rho(p)}\left\{\sum_{k=1}^{K}\min_{x^k\in X}\Big((\alpha_k-\delta_k)\lambda^\top x^k+q_k f_k(x^k)\Big)\right\} \end{aligned} \tag{4.7}$$

For any $\lambda\in\mathbb{R}^d$, $g(\lambda)$ represents a valid lower bound for the optimal objective value of the original problem. Below, we discuss three decomposition approaches that differ in how to recover a lower bound from $g(\lambda)$ or its variations. We refer to them as DD1, DD2 and DD3, respectively.

4.2.3 DD1 by Using $g(0)$

Here, we simply set $\lambda = 0$ and use $g(0)$ to update the lower bound. By doing this, we essentially relax the NAC (4.5), and allow decision x to be made specifically for each scenario. Although this will likely produce an over-optimistic outcome, i.e., a relatively weak lower bound, it can still yield a good outcome if the scenarios are similar and agree on the same best decision.

Plugging $\lambda = 0$ into (4.7) yields:

$$\begin{aligned} g(0) &= \max_{q\in\mathcal{Q}_\rho(p)}\left\{\sum_{k=1}^{K}\min_{x^k\in X}\Big\{q_k f_k(x^k)\Big\}\right\} \\ &= \max_{q\in\mathcal{Q}_\rho(p)}\left\{\sum_{k=1}^{K}q_k\min_{x^k\in X}\Big\{f_k(x^k)\Big\}\right\}, \end{aligned} \tag{4.8}$$

where we are able to pull $q_1,\ldots,q_K$ out of the minimization subproblems. Based on (4.8), we compute $g(0)$ in two steps as follows.

- *Step (i)*: for each $k = 1,\ldots,K$, optimize scenario subproblem

$$\textbf{Scen}(k):\quad \beta_k = \min_x\{f_k(x) : x\in X\}; \tag{4.9}$$

- *Step (ii)*: letting $\beta = (\beta_1, \ldots, \beta_K)$, optimize a maximization problem over variable q as

$$\textbf{Cont}(\beta): \quad \max_{q} \left\{ \sum_{k=1}^{K} \beta_k q_k : q \in \mathcal{Q}_\rho(p) \right\}. \tag{4.10}$$

Note that (4.8) is the dual representation of a coherent risk $\rho(\min_{x\in X} f(x,\xi))$ according to (4.3). Here the random value $\min_{x\in X} f(x,\xi)$ has finite realizations $\beta_1, \ldots, \beta_K$ computed through Step *(i)*, with the corresponding probabilities $p_1, \ldots, p_K$. Therefore, Step *(ii)* can be done by optimizing the risk function $\rho(\min_{x\in X} f(x,\xi))$ directly if possible, rather than using the dual form in (4.10).

Algorithm 4.1 below gives the two steps for the DD1 algorithm, which iterates until ℓ and u are sufficiently close, e.g., $\leq \epsilon$. In each iteration, we solve scenario subproblems **Scen**(k), $\forall k = 1, \ldots, K$, and compute $g(0)$ to update the lower bound ℓ (see steps 4–8). Each subproblem has the same feasible region as the original problem. Thus, we collect all the subproblem solutions $\hat{x}_k$, $k = 1, \ldots, K$ into the set S and then evaluate their objective values as:

$$\textbf{Eval}(\hat{x}) := \rho(f(\hat{x}, \xi)) \tag{4.11}$$

for every $\hat{x} \in S$, to update the upper bound u (see steps 9–11). We then cut off every evaluated solution by adding the no-good cut (1.19) to subproblem **Scen**$(1), \ldots,$ **Scen**(K) (see step 12).

Algorithm 4.1 The DD1 dual decomposition algorithm for solving model (4.4)

1: $u \leftarrow +\infty$, $\ell \leftarrow -\infty$
2: **repeat**
3: $\quad S \leftarrow \emptyset$
4: $\quad$ **for** $k = 1, \ldots, K$ **do**
5: $\quad\quad (\beta_k, \hat{x}^k) \leftarrow$ **Scen**(k)
6: $\quad\quad S \leftarrow S \cup \{\hat{x}^k\}$
7: $\quad$ **end for**
8: $\quad \ell \leftarrow \max\{\ell,$ **Cont**$(\beta)\}$
9: $\quad$ **for** $\hat{x} \in S$ **do**
10: $\quad\quad u \leftarrow \min\{u,$ **Eval**$(\hat{x})\}$
11: $\quad$ **end for**
12: $\quad X \leftarrow X \setminus S$.
13: **until** $u - \ell < \epsilon$

Remark 1. *Without going through Lagrangian relaxation steps, we can still verify* (4.8) *serving*

as a valid lower bound:

$$\begin{aligned}\min_{x\in X}\rho(f(x,\xi)) &= \min_{x\in X}\max_{q\in\mathcal{Q}_\rho(p)}\mathbb{E}_q[f(x,\xi)]\\ &\geq \max_{q\in\mathcal{Q}_\rho(p)}\min_{x\in X}\mathbb{E}_q[f(x,\xi)]\\ &\geq \max_{q\in\mathcal{Q}_\rho(p)}\mathbb{E}_q\left[\min_{x\in X}\{f(x,\xi)\}\right]\\ &= \rho\left(\min_{x\in X} f(x,\xi)\right) = (4.8).\end{aligned}$$

Remark 2. *If subproblem* ***Scen***(k) *becomes infeasible for some* $k\in\{1,\dots,K\}$*, we stop and check the current upper bound value* u *before computing the rest of the subproblems. We claim an optimal solution found as the one attaining the best upper bound if* $u<+\infty$*, or the problem is infeasible if* $u=+\infty$ *(i.e., no feasible solution has been found from any* ***Scen***(k)*. This also applies to DD2 and DD3 in the following sections.*

4.2.4 DD2 by Optimizing $g(\lambda)$ Using a Cutting-Plane Method

Both DD2 and DD3 use the Lagrangian dual $\max_\lambda\{g(\lambda)\}$ to update the lower bound, but differ in their computation. We discuss in this section a cutting-plane method used by DD2, and in Section 4.2.5 a subgradient method used by DD3.

According to (4.7), we formulate a master problem equivalent to $\max_\lambda\{g(\lambda)\}$ over variables $\phi\in\mathbb{R}$, $\lambda\in\mathbb{R}^d$ and $q\in[0,1]^K$ as:

$$\max_{\phi,\lambda,q}\quad \phi \tag{4.12a}$$

$$\text{s.t.}\quad \phi\leq\sum\nolimits_{k=1}^{K}\min_{x^k\in X}\left\{(\alpha_k-\delta_k)\lambda^\top x^k+q_k f_k(x^k)\right\} \tag{4.12b}$$

$$q\in\mathcal{Q}_\rho(p), \tag{4.12c}$$

where constraints (4.12b) are linear in ϕ, q, and λ. We relax (4.12b) and enforce them by generating cuts iteratively. Specifically, given a solution $(\hat\phi,\hat\lambda,\hat q)$ to a relaxed master problem, for each $k=1,\dots,K$, we compute a scenario subproblem given by

$$\beta_k^{\mathrm{DD2}}=\min_x\left\{(\alpha_k-\delta_k)\hat\lambda^\top x+\hat q_k f_k(x): x\in X\right\}. \tag{4.13}$$

If $\hat\phi>\sum_{k=1}^K\beta_k^{\mathrm{DD2}}$, we add an optimality cut

$$\phi\leq\sum\nolimits_{k=1}^{K}\left((\alpha_k-\delta_k)\lambda^\top\hat x^k+q_k f_k(\hat x^k)\right) \tag{4.14}$$

where $\hat{x}^k$ represents an optimal solution to the subproblem (4.13), and re-iterate to solve the master problem (4.12). We terminate this cutting-plane subroutine when $\hat{\phi} \leq \sum_{k=1}^{K} \beta_k^{\mathrm{DD2}}$ which implies that constraints (4.12b) are satisfied and $\hat{\phi}$ equals to $\max_\lambda\{g(\lambda)\}$. We then update the lower bound ℓ with $\hat{\phi}$.

Throughout the cutting-plane iterations, the following relation holds:

$$\sum_{k=1}^{K} \beta_k^{\mathrm{DD2}} \leq \max_{\lambda}\{g(\lambda)\} \leq \hat{\phi}.$$

Therefore, even before attaining $\max_\lambda\{g(\lambda)\}$, we can update the lower bound ℓ with $\sum_{k=1}^{K} \beta_k^{\mathrm{DD2}}$ in every cutting-plane iteration, which may close the gap between ℓ and u earlier.

Remark 3. *For any $\hat{\lambda}$ and nonnegative $\hat{q}$, the subproblem* (4.13) *must be bounded, because* ***Scen***(k) *is bounded by assumption and $x^k \in X \subseteq \{0,1\}^d$.*

Algorithm 4.2 gives the steps of this cutting-plane subroutine. The new decomposition algorithm runs this subroutine in every bound-and-cut iteration, which generates valid lower bounds for updating ℓ but also feasible solutions (note that every scenario subproblem (4.13) shares the same feasible region with the original problem) to update u. The corresponding procedure is deduced from replacing steps 4–8 in Algorithm 4.1 by Algorithm 4.2. To avoid the cutting-plane subroutine taking excessively long time to converge, we set an upper limit to the number of cutting-plane iterations when implementing Algorithm 4.2.

Algorithm 4.2 The cutting-plane subroutine in DD2 for updating the lower bound

1: $\hat{\lambda} \leftarrow 0, \hat{q} \leftarrow p, \hat{\phi} \leftarrow +\infty$
2: **repeat**
3: **for** $k = 1, \ldots, K$ **do**
4: solve (4.13) to attain the optimal objective value β_k^{DD2} and the optimal solution $\hat{x}^k$
5: $S \leftarrow S \cup \{\hat{x}^k\}$
6: **end for**
7: $\ell \leftarrow \max\{\ell, \sum_{k=1}^{K} \beta_k^{\mathrm{DD2}}\}$
8: add cut (4.14) to the master problem (4.12)
9: solve the master problem and attain an optimal solution $(\hat{\phi}, \hat{\lambda}, \hat{q})$
10: **until** $\hat{\phi} \leq \sum_{k=1}^{K} \beta_k^{\mathrm{DD2}}$

4.2.5 DD3 by Using a Subgradient Method

In this method, we approach the Lagrangian dual $\max_\lambda\{g(\lambda)\}$ by a subgradient method for updating the lower bound. It forms a decomposition algorithm similar to Algorithm 3.1 that we propose for chance-constrained 0-1 programs in Chapter 3. As mentioned, since the subgradient subroutine tends to take long time and tightens the lower bound very little, we explore a higher-dimensional NAC:

$$\sum_{k=1}^{K} \alpha_k x^k = x^i \quad \forall i = 1,\dots,K, \tag{4.15}$$

which are K copies of (4.5) with the right-hand sides varying from x^1 to x^K. We associate each of these constraints with multiplier $q_i\lambda^i$, where q_i is the probability mass value of the unknown distribution from the uncertainty set $\mathcal{Q}_\rho(p)$ and λ^i is a d-dimensional variable, for all $i = 1,\dots,K$. Following similar steps for obtaining the function $g(\lambda)$ in (4.7), we formulate a relaxation of the model with the new NAC (4.15) as:

$$\begin{aligned} g(\lambda^1,\dots,\lambda^K) &:= \max_{q\in\mathcal{Q}_\rho(p)} \min_{x^1,\dots,x^K\in X} \left\{ \sum_{i=1}^{K} q_i \left((\lambda^i)^\top \left(\sum_{k=1}^{K} q_k x^k - x^i \right) + f_i(x^i) \right) \right\} \\ &= \max_{q\in\mathcal{Q}_\rho(p)} \min_{x^1,\dots,x^K\in X} \left\{ \sum_{k=1}^{K} q_k \left(f_k(x^k) - \left(\lambda^k\right)^\top x^k \right) + \left(\sum_{k=1}^{K} q_k \lambda^k \right)^\top \left(\sum_{k=1}^{K} q_k x^k \right) \right\}. \end{aligned} \tag{4.16}$$

We consider a polyhedron set $A(\lambda^1,\dots,\lambda^K) = \left\{ q : \sum_{k=1}^{K} \lambda^k q_k = 0 \right\}$. A lower approximation of $g(\lambda^1,\dots,\lambda^K)$ that has a decomposable inner minimization problem is given by

$$\underline{g}(\lambda^1,\dots,\lambda^K) := \max_{q\in\mathcal{Q}_\rho(p)\cap A(\lambda^1,\dots,\lambda^K)} \min_{x^1,\dots,x^K\in X} \left\{ \sum_{k=1}^{K} q_k \left(f_k(x^k) - \left(\lambda^k\right)^\top x^k \right) \right\} \tag{4.17}$$

$$= \max_{q\in\mathcal{Q}_\rho(p)\cap A(\lambda^1,\dots,\lambda^K)} \left\{ \sum_{k=1}^{K} \left(q_k \min_{x^k\in X} \left\{ f_k(x^k) - \left(\lambda^k\right)^\top x^k \right\} \right) \right\}. \tag{4.18}$$

which we compute in two steps as follows.

- *Step (i):* for each $k = 1,\dots,K$, solve a scenario subproblem:

$$\beta_k^{\mathrm{DD3}} = \min_{x\in X} \left\{ f_k(x) - (\lambda^k)^\top x \right\} \tag{4.19}$$

  which is bounded following the same explanation in Remark 3.

- *Step (ii):* solve a maximization problem over the continuous variable q:

$$\max_{q} \left\{ \sum_{k=1}^{K} \beta_k^{\mathrm{DD3}} q_k : q \in \mathcal{Q}_\rho(p) \cap A(\lambda^1, \ldots, \lambda^K) \right\}. \tag{4.20}$$

The value of $\underline{g}(\lambda^1, \ldots, \lambda^K)$ for any given multiples $\lambda^1, \ldots, \lambda^K$, will provide a valid lower bound. We strive to obtain the best lower bound by maximizing $\underline{g}(\lambda^1, \ldots, \lambda^K)$ via a subgradient method. We repeat the steps of computing $\underline{g}(\lambda^1, \ldots, \lambda^K)$, and then updating λ^k for $k = 1, \ldots, K$ by using the subgradient $-\hat{q}_k \hat{x}^k$ (where $\hat{x}^k$ is an optimal solution to the kth subproblem (4.19)) and the Polyak rule for computing step size. We present the steps of this subgradient subroutine in Algorithm 4.3. We deduce the new algorithm by replacing steps 4–8 in Algorithm 4.1 with Algorithm 4.3.

Algorithm 4.3 The subgradient subroutine in DD3 for updating the lower bound

1: $\lambda^k \leftarrow 0, \ \forall k = 1, \ldots, K$
2: **repeat**
3: **for** $k = 1, \ldots, K$ **do**
4: solve scenario subproblem (4.19) to attain the optimal objective value β_k^{DD3} and the optimal solution $\hat{x}^k$.
5: $S \leftarrow S \cup \{\hat{x}^k\}$
6: **end for**
7: solve (4.20) to attain an optimal solution $\hat{q}$ and the optimal objective value $\sum_{k=1}^{K} \beta_k^{\mathrm{DD3}} \hat{q}_k$.
8: $\ell \leftarrow \max\{\ell, \sum_{k=1}^{K} \beta_k^{\mathrm{DD3}} \hat{q}_k\}$
9: **for** $k = 1, \ldots, K$ **do**
10: update λ^k by using the subgradient $-\hat{q}_k \hat{x}^k$.
11: **end for**
12: **until** achieving the general stop criteria for the subgradient method

4.3 Distributionally Robust Variants

We consider that the probability distribution p of the uncertainty ξ is not exhaustively known. Instead, an uncertainty set $\mathcal{P} \subseteq \mathcal{M}$ of p, consisting of all possible distributions, is available. We let $\overline{\rho}(f(x, \xi))$ be the worst-case risk outcome for any $p \in \mathcal{P}$. We define set

$$\mathcal{D}_\rho(\mathcal{P}) := \left\{ q : q \in \mathcal{Q}_\rho(p), \forall p \in \mathcal{P} \right\}. \tag{4.21}$$

We consider a distributionally robust risk-averse program as

$$\min_{x\in X} \overline{\rho}(f(x,\xi)) = \min_{x\in X} \max_{q\in\mathcal{D}_\rho(\mathcal{P})} \left\{ \sum_{k=1}^{K} q_k f_k(x) \right\} \tag{4.22}$$

Comparing (4.22) with (4.4), we have $\mathcal{Q}_\rho(p)$ in the latter replaced by $\mathcal{D}_\rho(\mathcal{P})$ in the former, under the ambiguity of p. Therefore, we can easily adapt the aforementioned algorithms to solving (4.22).

4.4 Parallel Implementation Schemes

In this section, we explore parallel computing schemes for implementing DD1, DD2, and DD3 in a distributed framework. DD1 has a simpler iterative structure than DD2 and DD3. (The latter two run two loops and take potentially multiple iterations for updating the lower bound in each round.) As a result, DD1 is easier to be parallelized, and the parallel programs can scale better. In Section 4.5.2, we show that DD1 outperforms the other two algorithms even when the steps in each algorithm are implemented in a series. Thus, we focus on develop parallel schemes for DD1 and present a general scheme for DD2 and DD3 in Appendix C.1.

4.4.1 Overview

In a serial implementation of DD1/DD2/DD3, subproblems like **Scen**$(\cdot)$ and **Eval**$(\cdot)$ are solved one by one. If there are multiple processes, we can spread the subproblems and place a barrier to synchronize all the processes which then exchange results for updating bounds and cuts before reiteration. We refer to this scheme as *Basic Parallel (BP)* and present the details in Section 4.4.2. However, waiting caused by barriers and high intensity of communication may compromise parallel efficiency. In Section 4.4.3, we introduce a master-worker scheme that dedicates one process to consolidate information. In Section 4.4.4, we introduce another master-worker scheme that avoids barriers. We refer to these two schemes as *Master-worker Parallel with Barriers (MWB)* and *Master-worker Parallel without Barriers (MWN)*, respectively.

Figure 4.4.1 shows a schematic view of each mentioned computing scheme. Squares of each color represent parallel tasks, e.g., **Scen**$(\cdot)$ or **Eval**$(\cdot)$. Red bars denote the barriers. Note that BP and MWB are "push" systems where we pre-assign computing jobs to processes. Specifically, given tasks $1,\ldots,J$ (with no prior knowledge about their time complexity), we assign them to N processes in a round-robin manner, such that the n^{th} process receives a subset

$$\Omega_n^{J,N} := \{j \in \{1,\ldots,J\} : (j-1) \bmod N = (n-1)\} \tag{4.23}$$

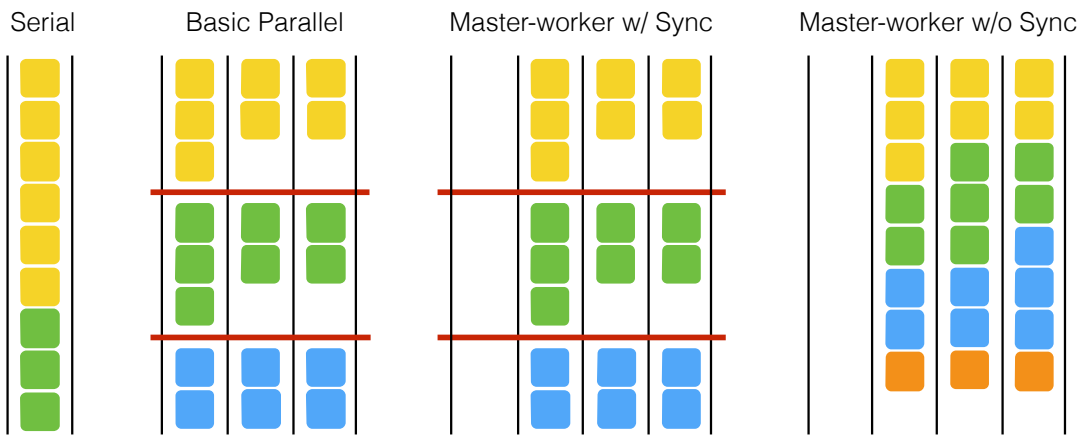

Figure 4.1: An overview of implementation schemes for dual decomposition algorithms

of tasks. In contrast, MWN is a "pull" system, where jobs are kept in a queue, and each is waiting to be solved whenever a process becomes available.

4.4.2 Basic Parallel

We present BP in Algorithm 4.4, which assigns the K scenarios across the N processes such that Process n (named "Proc^n") receives a subset $\Omega_n^{K,N}$ of scenarios, for $n = 1,\ldots,N$. Each process solves **Scen**(k) for every assigned scenario k, and evaluates its optimal solution. We use the evaluation result to update the local upper bound u_n in Proc^n. We then let all the processes share their results through some collective operation (e.g., AllToAll in MPI (Message Passing Interface) in Pacheco, 1997), which, in general, communicates faster than point-to-point sending/receiving messages, but implies a barrier such that all the processes must reach the point before they can begin communication. In practice, barriers cause waiting.

The BP scheme has another drawback as evaluating repeated solutions from different scenario subproblems. In each process, we can do a local check between steps 5 and 6 to avoid evaluating some $\hat{x}^k$ that we have encountered earlier in the loop. However, we cannot avoid re-evaluation if the same solution occurs in different processes, which is very likely in our computation.

4.4.3 Master-worker Parallel with Barriers

We let Proc^N be the master (process) consolidating solutions and updating the bounds. The rest $N-1$ processes are workers sharing the computation of **Scen**$(\cdot)$ and **Eval**$(\cdot)$. Every Proc^n $(n \neq N)$ solves **Scen**(k) for all $k \in \Omega_n^{K,N-1}$, and sends the results to the master. The master stores β_k and collects non-repeated $\hat{x}^k$ in a solution list $\mathcal{S}$. (Note that $\mathcal{S}$ is different from the solution set S, which

Algorithm 4.4 The BP Scheme at Proc^n ($n \in \{1,\ldots,N\}$)

$u \leftarrow +\infty$, $\ell \leftarrow -\infty$
repeat
  Initialize the local upper bound $u_n \leftarrow +\infty$.
  for $k \in \Omega_n^{K,N}$ **do**
    $(\beta_k, \hat{x}^k) \leftarrow$ **Scen**(k)
    $u_n \leftarrow \min\{u_n, \textbf{Eval}(\hat{x}^k)\}$
    $S \leftarrow S \cup \{\hat{x}^k\}$
  end for
  pass $\{(\beta_k, \hat{x}^k) : k \in \Omega_n^{K,N}\}$ and u_n to all the other processes
  $\ell \leftarrow \max\{\ell, \textbf{Cont}(\beta)\}$
  $u \leftarrow \min\{u, u_1, \ldots, u_N\}$
  $X \leftarrow X \setminus S$.
until $u - \ell < \epsilon$

collects solutions that are not necessarily ordered, whereas $\mathcal{S}$ arranges the candidate solutions in a certain order.) Once it has attained the results from all K scenarios, it broadcasts the list $\mathcal{S}$. Every worker receives the whole list (for adding no-good cuts), and shares the evaluation of all solutions in the list $\mathcal{S}$. Specifically, we define $\mathcal{S}_i$ as the i^{th} solution in $\mathcal{S}$. Every Proc^n ($n \neq N$) evaluates $\mathcal{S}_i$ for all $i \in \Omega_n^{|\mathcal{S}|,N-1}$, and sends the results to the master. The master, after broadcasting $\mathcal{S}$, updates the bounds and forces all processes to terminate once it detects a sufficiently small gap. We present the algorithmic steps of a worker process in Algorithm 4.5 and of the master in Algorithm 4.6, where u' denotes a temporary, valid lower bound obtained at each worker and $\mathcal{S} + \hat{x}^k$ means appending solution $\hat{x}^k$ to the tail of the list $\mathcal{S}$.

Algorithm 4.5 The MWB Scheme at Proc^n ($n \in \{1,\ldots,N-1\}$) (worker)

loop
  for $k \in \Omega_n^{K,N-1}$ **do**
    $(\beta_k, \hat{x}^k) \leftarrow$ **Scen**(k)
    send $(\beta_k, \hat{x}^k)$ to Proc^N
  end for
  gather $\mathcal{S}$ from Proc^N
  for $i \in \Omega_n^{|\mathcal{S}|,N-1}$ **do**
    $u' \leftarrow$ **Eval**$(\mathcal{S}_i)$
    send u' to Proc^N
  end for
  $X \leftarrow X \setminus \mathcal{S}$
end loop

The master collecting and broadcasting solutions avoids evaluating duplicated solutions. In terms of communication, we use asynchronous "send/receive", so that the process that sends the message does not wait for the reception of the message, but proceeds with the succeeding steps,

Algorithm 4.6 The MWB Scheme at Proc^N (master)

1: **repeat**
2: $u \leftarrow +\infty$, $\ell \leftarrow -\infty$
3: $\mathcal{S} \leftarrow \emptyset$
4: **for** K times **do**
5: receive $(\beta_k, \hat{x}^k)$ from Proc^n
6: **if** $\hat{x}^k \notin \mathcal{S}$ **then**
7: $\mathcal{S} \leftarrow \mathcal{S} + \hat{x}^k$
8: **end if**
9: **end for**
10: broadcast $\mathcal{S}$ to $\text{Proc}^1, \ldots, \text{Proc}^{N-1}$
11: $\ell \leftarrow \max\{\ell, \textbf{Cont}(\beta)\}$
12: **for** $|\mathcal{S}|$ times **do**
13: receive u' from Proc^n
14: $u \leftarrow \min\{u, u'\}$
15: **end for**
16: **until** $u - \ell < \epsilon$
17: terminate all processes

which allows for more parallelism. However, this requires buffers to store the data in transit. In each iteration of MWB, transiting data contains at most K d-dimensional binary vectors and K real numbers, which is a fairly modest amount. In step 10 of Algorithm 4.6 (or step 6 of Algorithm 4.5), to send the solution list $\mathcal{S}$ from the master to the worker, we use broadcast, which, again, is a collective operation implying a barrier across all the processes.

4.4.4 Master-worker Parallel without Barriers

In BP and MWB, all of the processes go through iterations synchronously due to the barriers implied by collective communication steps. Therefore, we want to design a new scheme that avoids these barriers. At the same time, we make it a pull system for better load balance.

Formally, the master keeps a queue of idle workers as $\mathtt{Q}_{\mathtt{proc}}$, and a queue of subproblems to solve as $\mathtt{Q}_{\mathtt{job}}$. As long as both queues are non-empty, it repeatedly pops out from each a worker and a job, then assigns the job to the worker. It then waits to receive any result from the workers. Once the master hears from some worker, it adds the worker to $\mathtt{Q}_{\mathtt{proc}}$, processes the received results, and creates new jobs if needed. Algorithm 4.7 explains the steps. In particular, steps 1, 5 and 9 are detailed in Algorithms 4.10, 4.9 and 4.8, respectively.

The master is still keeping the bounds and the list $\mathcal{S}$ of vertices. Upon receiving a result of **Eval**$(\cdot)$, i.e., u', the master uses it to update the upper bound. Upon receiving a result of **Scen**(k), i.e., $(\beta_k, \hat{x}^k)$, the master resolves **Cont**(β) which takes in the new value of β_k, to update the lower

Algorithm 4.7 The MWN Scheme at Proc^N (worker)

```
 1: initialization
 2: repeat
 3:    while Q_proc ≠ ∅ and Q_job ≠ ∅ do
 4:       n ← pop(Q_proc), j ← pop(Q_job)
 5:       assign j to Proc^n
 6:    end while
 7:    receive r from Proc^n
 8:    Q_proc ← Q_proc + n
 9:    process r (and create jobs to Q_job)
10: until u − ℓ < ϵ
11: terminate all processes
```

bound. If $\hat{x}^k$ is new, the master appends it to $\mathcal{S}$ and creates a job for evaluating its objective value. In addition, every reception of **Scen**(k)'s result triggers creating a new job for solving **Scen**(k) for reiteration.

Algorithm 4.8 Master result processing subroutine (step 9 of Algorithm 4.7)

```
 1: switch r:
 2:    case u′ :
 3:       u ← min{u, u′}
 4:    case (β_k, x̂^k) :
 5:       ℓ ← max{ℓ, Cont(β)}
 6:       if x̂^k ∉ S then
 7:          S ← S + x̂^k
 8:          Q_job ← Q_job + Eval(x̂^k)
 9:       end if
10:       Q_job ← Q_job + Scen(k)
11: end switch
```

To avoid barriers, the system has no centralized step for informing the workers about all of the explored vertices, but rather use point-to-point communication between the master and any individual worker to communicate about jobs and results. So that the workers know what cuts to add to the subproblems, we let the master keep, for each worker n, an index $\tau(n) \in \{0,\ldots,|\mathcal{S}|\}$ pointing to the end of a sublist of $\mathcal{S}$ which the worker has been given in order to generate cuts. For example, if $\tau(1) = 3$ and $\tau(2) = 5$, it means Proc^1 and Proc^2 have generated cuts to exclude vertices $\mathcal{S}_1,\ldots,\mathcal{S}_3$, and $\mathcal{S}_1,\ldots,\mathcal{S}_5$, respectively. Once it is time for Proc^n to update cuts, the master sends $\mathcal{S}_{\tau(n)+1},\ldots,\mathcal{S}_{|\mathcal{S}|}$, and then increases $\tau(n)$ to $|\mathcal{S}|$. Note that additional cuts affect **Scen**$(\cdot)$ but not **Eval**$(\cdot)$. The master therefore sends these vertices along with the jobs of **Scen**$(\cdot)$. Algorithm 4.9 gives the job assignment subroutine with these details. Algorithm 4.10 gives the initialization steps.

Algorithm 4.9 Master job assignment subroutine (step 5 of Algorithm 4.7)

switch j**:**
  case Eval$(\hat{x})$ **:**
    send (**Eval**$(\hat{x})$) to Proc^n
  case Scen(k) **:**
    $S \leftarrow \{\mathcal{S}_i : i = \tau(n)+1, \ldots, |\mathcal{S}|\}$
    $\tau(n) \leftarrow |\mathcal{S}|$
    send $(S, \textbf{Scen}(k))$ to Proc^n
end switch

Algorithm 4.10 Master initialization subroutine (step 1 of Algorithm 4.7)

$u \leftarrow +\infty$, $\ell \leftarrow -\infty$
$\mathcal{S} \leftarrow \emptyset$
$\tau(1), \ldots, \tau(N-1) \leftarrow 0$
$\mathtt{Q}_{\mathtt{proc}} \leftarrow \langle 1, \ldots, N-1 \rangle$
$\mathtt{Q}_{\mathtt{job}} \leftarrow \langle \textbf{Scen}(1), \ldots, \textbf{Scen}(K) \rangle$

On the other hand, the workers receive tasks from the master, work on them and send the results back to the master. Algorithm 4.11 gives the steps.

Algorithm 4.11 The MWN Scheme at Proc^n ($n \in \{1, \ldots, N-1\}$)

loop
  receive j from Proc^N
  switch j**:**
    case (**Eval**$(\hat{x})$) **:**
      $u' \leftarrow$ **Eval**$(\hat{x})$
      send u' to Proc^N
    case $(S, \textbf{Scen}(k))$ **:**
      $X \leftarrow X \setminus S$
      $(\beta_k, \hat{x}^k) \leftarrow$ **Scen**(k)
  end switch
end loop

Figure 4.2 summarizes MWN with a schematic view for both the master and the workers.

A major difference between MWN and schemes BP and MWB is that the workers are asynchronous in adding cuts. Given the cuts that Proc^n has generated, we denote the feasible region as X_n. Then each **Scen**(k) that Proc^n solves, precisely, is:

$$\beta'_k(n_k) = \min\{f_k(x) : x \in X_{n_k}\},$$

where $n_k \in \{1, \ldots, N-1\}$ specifies the worker to which the scenario subproblem **Scen**(k) is assigned.

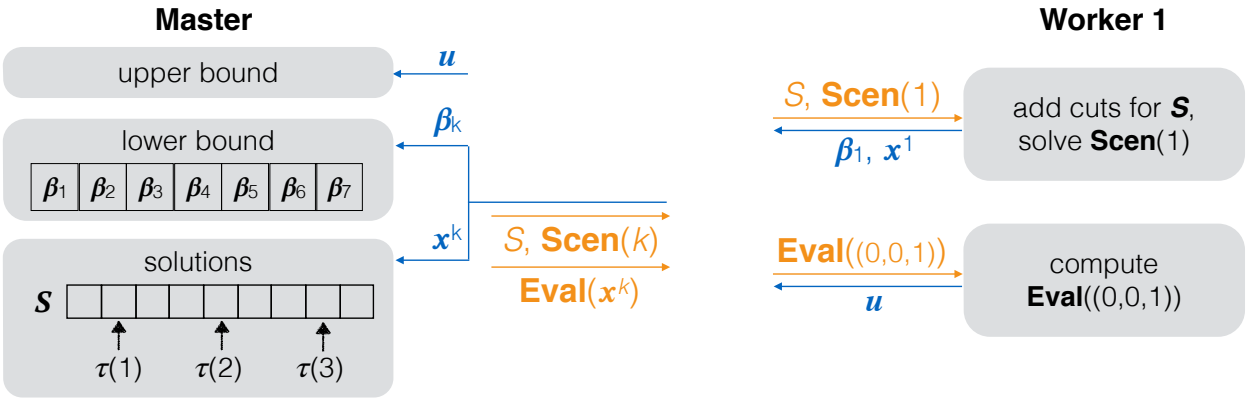

Figure 4.2: A schematic view of MWN

A lower bound obtained in this case is

$$\tilde{\ell} = \mathbf{Cont}((\beta'_1(n_1),\ldots,\beta'_K(n_K))).$$

To justify that the algorithm still works with this lower bound, we want to show that $\tilde{\ell}$ will not cross with the upper bound u, when u is not yet optimal. We define $\overline{X} = X_{n_1} \cap \cdots \cap X_{n_K}$, and for $k = 1,\ldots,K$, consider

$$\overline{\beta}_k = \min\{f_k(x) : x \in \overline{X}\}.$$

By definition, $\overline{\beta}_k \geq \beta'_k(n_k)$ for $k = 1,\ldots,K$. When u is suboptimal, any optimal solution x^* must have not been explored yet (i.e., $x^* \notin \mathcal{S}$), and thus $x^* \in X_{n_k}$ for $k = 1,\ldots,K$ which implies that $x^* \in \overline{X}$. Therefore, we also have $\overline{\beta}_k \leq f_k(x^*)$ for $k = 1,\ldots,K$. Therefore,

$$u > \mathbf{Cont}((f_1(x^*),\ldots,f_K(x^*))) \geq \mathbf{Cont}((\overline{\beta}_1,\ldots,\overline{\beta}_K)) \geq \mathbf{Cont}((\beta'_1(n_1),\ldots,\beta'_K(n_K))) = \tilde{\ell}.$$

4.5 Computational Results

We implement parallelization by OpenMPI 1.6 (Gabriel et al., 2004), and perform the computation on Flux HPC cluster, University of Michigan. We use as many as 21 compute nodes; each compute node has twelve 2.67 GHz Intel Xeon X5650 processes and 48GB RAM. All involved optimization models (including subproblems as a result of decomposition) are solved by CPLEX 12.6 via ILOG Concert Technology. We set a runtime limit of six hours for computing each instance.

4.5.1 Instances and Experimental Setup

We extract two sets of instances from SIPLIB (Ahmed et al., 2015a):

- Stochastic server location problem (SSLP) from a telecommunication application studied by Ntaimo and Sen (2005): The first stage decides where to place servers out of n locations, and the second stage satisfies uncertain demand from m clients. The cost function is given by

$$f(x,\xi) = \gamma^\top x + \min_y \{\theta_1^\top y + \theta_2^\top z : W_1 y + W_2 z \geq r(\xi) - Tx,\ y \in \{0,1\}^{n\times m},\ z \in \mathbb{R}^n_+\}$$

  and the feasible region is $x \in X = \{0,1\}^n$. We use four instances with $n = 10$ and $m = 50$. They contain 50, 100, 500 and 1000 scenarios, respectively.

- Stochastic multiple 0-1 knapsack problem (SMKP) studied by Angulo et al. (2014a): The first and the second stages are multiple 0-1 knapsack problems with n and m entities, respectively. The cost function is:

$$f(x,\xi) = \gamma^\top x + \min_y \{\theta(\xi)^\top y : Wy \geq r - Tx,\ y \in \{0,1\}^m\}$$

  and the feasible region is $X = \{x \in \{0,1\}^n : Ax \geq w\}$ with linear constraints $Ax \geq w$. The original dataset contains 30 instances each containing 20 scenarios. To examine how problem scale affects the algorithm performance, we select the first instance and propagate it to attain instances with 40, 80, and 160 scenarios. To propagate a K-scenario dataset to K'-scenario, we first include the original K scenarios. Next, we repeat for $K' - K$ times, generate random numbers $\sigma_1, \ldots, \sigma_K$, and then create a scenario k such that $\theta(\xi^k)$ is a weighted average of the original $\theta(\xi^1), \ldots, \theta(\xi^K)$ with weighing coefficients given by normalized $\sigma_1, \ldots, \sigma_K$, i.e.,

$$\theta(\xi^k) = \frac{\sum_{k'=1}^K \sigma_{k'} \theta(\xi^{k'})}{\sum_{k'=1}^K \sigma_{k'}}.$$

Table 4.1 presents the scales and the performance of LP relaxations for these instances. Specifically, *#_var* and *#_constr* give the numbers of binary variables and constraints, respectively, in the corresponding problems. SMKP has pure binary problems in both stages. SSLP has a binary first-stage, and a mixed-binary second-stage problem, which involves 10 continuous variables in each scenario. Both sets contain four instances with varying numbers of scenarios. In every instance, all the scenarios are equal likely, i.e., $p_1 = \cdots = p_K = 1/K$. Regarding the number of variables and constraints, SSLP has a larger second stage than the first stage, whereas SMKP has the opposite. The last section of Table 4.1 lists the computational time (in seconds) of the LP relaxation (see *total_time*) and the gaps between the optimal objective values of the LP relaxation and the original binary program (see *tightness*). As the number of scenarios increases, we observe a trend of longer computations and weaker LP relaxations.

Table 4.1: Scales of different instances and performance of their LP relaxations

		SSLP				SMKP			
		_50	_100	_500	_1000	_20	_40	_80	_160
1st-stage	#_var	10	10	10	10	240	240	240	240
	#_constr	1	1	1	1	50	50	50	50
2nd-stage	#_scen (K)	50	100	500	1000	20	40	80	160
	#_var (per scen)	500	500	500	500	120	120	120	120
	#_constr (per scen)	60	60	60	60	5	5	5	5
LP relaxation	total_time	0.48	2.35	59.29	45.19	0.043	0.071	0.069	0.229
	tightness	24.8%	25.8%	24.3%	27.9%	0.36%	2.99%	3.15%	5.53%

For the coherent risk measures, we choose conditional value-at-risk (CVaR), i.e., $\rho(\cdot) = \text{CVaR}_\alpha(\cdot)$, and set the reliability parameter α to 0.9. The uncertainty set in the dual representation of CVaR contains only linear constraints (see Shapiro and Ahmed, 2004), given by

$$\mathcal{Q}_\rho(p) = \left\{ (q_1, \ldots, q_K) : \sum_{k=1}^{K} q_k = 1,\ 0 \le q_k \le p_k/(1-\alpha),\ \forall k = 1, \ldots, K \right\}.$$

In this case, model (4.4) becomes a minimax linear integer program, and the maximization problems **Cont**(β), (4.12a)—(4.12c), and (4.20), which emerge from the decomposition algorithms, are all linear programs.

For the benchmark of the dual decomposition algorithms, we use the model of CVaR first proposed by Rockafellar et al. (2002):

$$\text{CVaR}_\alpha(f(x,\xi)) = \min_{\eta} \left\{ \eta + \frac{1}{1-\alpha} \sum_{k=1}^{K} p_k \left[f_k(x) - \eta \right]^+ : \eta \in \mathbb{R} \right\},$$

and solve the considered risk-averse problem with an MIP:

$$\min_{x,\eta,v_1,\ldots,v_K} \left\{ \eta + \frac{1}{1-\alpha} \sum_{k=1}^{K} p_k v_k : v_k \ge f_k(x) - \eta,\ v_k \ge 0,\ \forall k = 1, \ldots, K,\ x \in X \right\}, \tag{4.24}$$

.

4.5.2 Results of Serial Implementation

The *default* serial scheme directly calls an off-the-shelf solver to optimize model (4.24). Table 4.2 compares the runtime of different approaches solved in one process, reported in *total_time*. We break down total time to show the total time for solving scenario subproblems (i.e., **Scen**(·) in DD1, (4.13) in DD2 and (4.19) in DD3), reported in *scen_time*, and the time for evaluating solutions (i.e.,

Table 4.2: Time (in seconds) and iteration counts in the serial implementation

		SSLP				SMKP			
		_50	_100	_500	_1000	_20	_40	_80	_160
default	**total_time**	**195**	**201**	**(100%)**	**(100%)**	**299**	**(0.09%)**	**(0.11%)**	**(0.16%)**
DD1	**total_time**	**248**	**502**	**4663**	**12750**	**2692**	**9866**	**11249**	**18774**
	scen_time	68	98	110	4857	2688	9853	11218	18704
	eval_time	180	404	4552	7893	3	13	31	70
	#_iter	4	3	2	2	2	1	1	2
DD2	**total_time**	**1276**	**2570**	**(10%)**	**(16%)**	**(0.02%)**	**(0.01%)**	**(0.02%)**	**(0.02%)**
	scen_time	651	1734	-	-	-	-	-	-
	eval_time	625	836	-	-	-	-	-	-
	#_iter	4	3	-	-	-	-	-	-
DD3	**total_time**	**415**	**602**	**7231**	**(9%)**	**3496**	**9080**	**(0.01%)**	**(0.01%)**
	scen_time	164	184	1066	-	3491	9056	-	-
	eval_time	251	418	6165	-	5	24	-	-
	#_iter	4	3	2	-	2	1	-	-

Eval$(\cdot)$), reported in *eval_time*. The number of iterations needed for the upper and lower bounds to converge, are also given in *#_iter*. For DD2 and DD3 which contain two nested loops, this refers to the count of the outer iterations. If an instance is not solved to optimum within the runtime limit of six hours, we present the optimality gap in parentheses in *total_time* and omit the other computational details.

Compared with default, the dual decomposition methods are slower when the number of scenarios in the instances is relatively small (e.g., SSLP_50, SSLP_100, and SMKP_20). However, as the number of scenarios increases, they demonstrate better time efficiency: (i) the optimality gaps on SSLP_500 and SSLP_1000 are, respectively, 10% and 16% under DD2, which are much better than the two 100%s under the default scheme; (ii) DD3 yields an even better optimality gap of 9% on SSLP_1000, and can solve SSLP_500 in two hours; and (iii) DD1 is even more efficient, solving the two instances in 4663 seconds and 12750 seconds, respectively. We observe a similar trend of computational efficacy in solving modest and large instances of SMKP (_40, _60, and _80) as

$$\text{default} < \text{DD2} < \text{DD3} < \text{DD1}.$$

As mentioned, both DD2 and DD3 contain an extra inner loop for improving the lower bound without shrinking the feasible region. However, by comparing *#_iter* of DD2/DD3 with DD1, we observe that this inner loop does not dramatically reduce the number of iterations in the outer loop. In contrast, the extra layer of loop results in more scenario subproblems to solve and more solutions to evaluate. Therefore, DD2 and DD3 are slower than DD1.

In every run of the dual decomposition algorithms, we observe that *scen_time* and *eval_time*

add up to *total_time*, which suggests that the other steps are time-wise negligible. We conclude that parallelism for solving the scenario subproblems and evaluating vertices significantly affect computational time.

4.5.3 Results of Parallel Algorithms

We capture the parallel time as the walltime that elapses from the start of the parallel program to the end. For each run we compute *speedup* as the ratio of the serial time to the parallel time. We test the three schemes introduced in Section 4.4 on both sets of SSLP and SMKP instances, and show how the results of speedup change as we vary the number of processes in Figure 4.3.

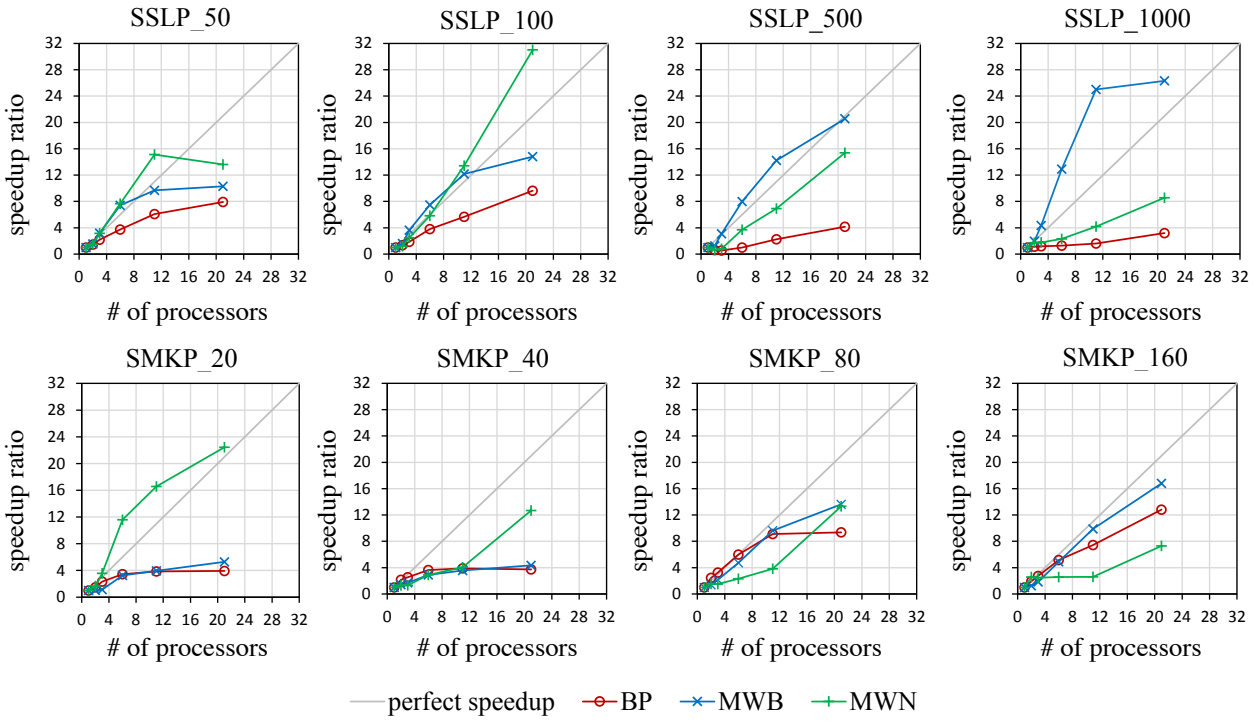

Figure 4.3: Speedup versus number of processes for implementing DD1

In Figure 4.3, each subfigure presents the results on one particular instance. We scale them up equally, so the main diagonal (in grey) gives the trending for a perfect parallelism where speedup is equal to the number of processes. Each background grid cell has a height of 5 (unit-less) and a width of 5 processes. We use 2, 3, 6, 11, and 21 processes. The red, blue, and green curves are results of BP, MWB, and MWN, respectively.

We observe many super-linear speedups, i.e., points located above the diagonal line, since our choice of parallelization can cause the total workload to abate from the serial implementation. For example, the serial program has spent 2 hours going through some iterations, and then comes

Table 4.3: Number of evaluated x-solutions

		SSLP				SMKP			
scheme	num of processes (N)	_50	_100	_500	_1000	_20	_40	_80	_160
BP	2	139	183	259	172	9	14	22	34
	3	155	214	489	221	11	16	24	42
	6	170	247	701	324	15	21	30	52
	11	184	266	892	882	17	27	37	62
	21	189	283	1095	1151	20	34	48	74
MWB	2	117	136	240	187	8	12	15	23
	3	117	136	240	187	8	12	15	23
	6	117	136	240	188	8	12	15	23
	11	117	134	240	188	8	12	15	23
	21	117	136	240	189	8	12	15	23

to an iteration with three outstanding jobs (e.g., **Scen**(k) or **Eval**($\hat{x}$)), that respectively take 1, 8, and 1 hours. The result of the third job leads to convergence. The serial total time is thus $2 + (1 + 8 + 1) = 12$ hours. However, if we have three processes each taking care of one job, the process responsible for the third job will detect convergence immediately after hour 1 and promptly terminate all processes. With two extra processes, the iterations that previously took 2 hours also become shorter. Therefore, the parallel total time is shorter than $2 + 1 = 3$ hours. In this case, we obtain a speedup ≥ 4 with only three processes.

Figure 4.3 shows other interesting results. On every SMKP instance, we observe a crossover of the red (BP) and blue (MWB) curves: the red always starts climbing sharper, but flatten out earlier and crossed with the blue. As we know, BP has one more process sharing the computation than MWB. This is an important advantage when N is still small, which explains why the red line climbing faster in the beginning. This advantage, however, fades as we increase N. In contrast, the advantage of MWB that it avoids reevaluating duplicate solutions stands out. This can be verified by comparing the number of solutions BP and MWB evaluate, reported in Table 4.3. For MWB, less time on evaluation and thus faster, which is then reflected as the blue curve climbing higher when N is large. For SSLP, BP scales worse than MWB, and the discrepancy widens as N increases, because BP needs more time to evaluate duplicated solutions which increases along with the increase of N.

Figure 4.3 also shows that when the number of scenarios is small (e.g., SSLP_50, SSLP_100, SMKP_20, and SMKP_40), MWB suffers from load imbalance, whereas MWN achieves good speedup due to the deployed pull mechanism. As the number of scenarios increases, there are more subproblems to solve and more solutions to evaluate. MWN has to communicate more to dispatch jobs, which in turn compromises speed. In contrast, when there are more scenarios, i.e., more jobs, MWB can shuffle around between barriers, thus alleviating load imbalance. This is why

MWB is faster than MWN on SSLP_500, SSLP_100, SMKP_80, and SMKP_160.

In addition to speedup, we also analyze communication time, which is the total time elapse from the completion of the predecessor of a communication step (e.g., send, receive, broadcast, etc.) to the start of its successor. Therefore, it consists of the time on transiting data (the net communication time) and the time on waiting to receive or broadcast data (the idle time). In practice, compared to the latter, the former is almost negligible due to the modest quantity of transiting data. We therefore can view the communication time as the idle time. We capture the communication time of each process as comm(n). Figure 4.4 plots the total communication time, i.e.,

$$\sum_{n=1}^{N} \text{comm}(n)$$

(in second) against the number of processes.

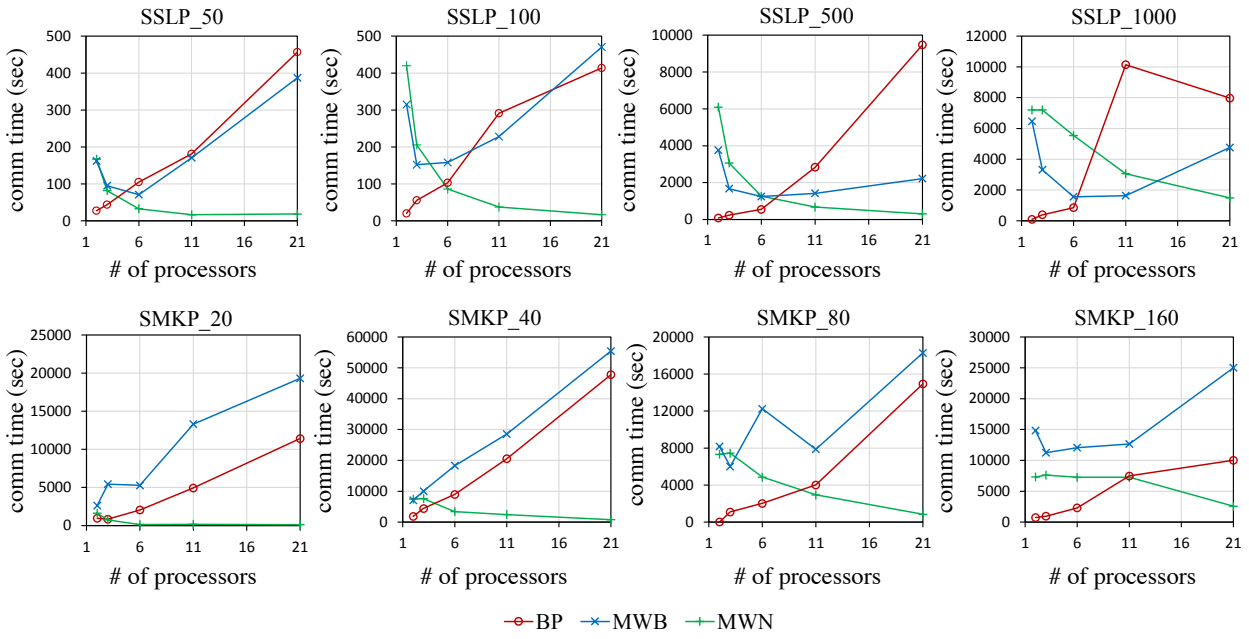

Figure 4.4: Communication time vs Number of processes (N)

Table 4.4 presents the percentage of communication time contributed by the master, i.e.,

$$\frac{\text{comm}(N)}{\sum_{n=1}^{N} \text{comm}(n)}.$$

We derive the following observations.

- As seen in Figure 4.4, the curves for BP, MWB and MWN follow notable trends of increasing, increasing and decreasing, respectively, although some MWB curves have one or two

Table 4.4: The percentage of communication time contributed by the master

		SSLP				SMKP			
scheme	N	_50	_100	_500	_1000	_20	_40	_80	_160
MWB	2	100%	100%	100%	100%	100%	100%	100%	100%
	3	82%	91%	91%	89%	83%	50%	50%	85%
	6	47%	43%	47%	63%	26%	20%	20%	29%
	11	15%	18%	23%	31%	11%	10%	10%	11%
	21	6%	7%	10%	10%	5%	5%	5%	5%
MWN	2	100%	100%	100%	100%	100%	100%	100%	100%
	3	100%	100%	100%	100%	100%	100%	100%	100%
	6	100%	100%	100%	100%	100%	100%	100%	100%
	11	99%	99%	100%	100%	100%	100%	100%	100%
	21	98%	97%	100%	100%	95%	99%	99%	100%

saddle points.

- In BP, there are only collective communication steps which imply barriers. Except for the small amount of time on transiting data, the communication time is composed of waiting time at barriers, which increases along with the increase of N.

- In MWN, the master is devoted to consolidating results from the workers. The more workers, the less time the master waits for results. We also see from Table 4.4 that nearly all the communication time is contributed by the master (which implies that the workers are well utilized for computation thanks to the pull mechanism), which explains the increasing of MWN curves in Figure 4.4.

- In MWB, although it has a similar master-worker structure, it contains a broadcasting step involving all the processes and barrier wait come along. As the number of processes increases, the communication time of the master shortens; however, the communication time of the workers increase in a larger magnitude. This addresses the drop in the master communication-time percentage along with the increase of N in Table 4.4, but also the trend of the MWB curves in Figure 4.4.

Finally, we summarize the runtime of all tested schemes in Table 4.5. For each instance, we give the shortest serial time and the shortest parallel time in boldface. We observe in all instances that the fastest parallel scheme outperforms the fastest serial scheme. Among the parallel schemes, MWN performs better for small instances, whereas MWB performs better for large instances.

Table 4.5: Computational time of all tested schemes

Scheme		N	SSLP				SMKP			
			_50	_100	_500	_1000	_20	_40	_80	_160
Serial	default	1	**195**	**201**	(100%)	(100%)	**300**	(0.09%)	(0.11%)	(0.16%)
	DD1	1	248	502	**4663**	**12750**	2692	9866	**11249**	**18774**
	DD2	1	1276	2570	(10%)	(16%)	(0.02%)	(0.01%)	(0.02%)	(0.02%)
	DD3	1	415	602	7231	(9%)	3496	**9080**	(0.11%)	(0.11%)
Parallel	BP	2	170	377	7633	11412	1808	4560	4630	9779
		3	113	271	8514	10781	1162	3897	3462	6877
		6	66	132	4694	9844	782	2711	1879	3659
		11	41	89	2074	7870	696	2539	1233	2521
		21	31	52	1122	3991	685	2648	1202	1464
	MWB	2	162	315	3758	6462	2641	7046	8162	14830
		3	78	138	1514	2938	2400	5834	5251	10297
		6	33	67	584	987	825	3418	2375	3840
		11	26	41	327	510	684	2773	1169	1903
		21	24	34	**227**	**484**	509	2269	**826**	**1117**
	MWN	2	166	418	7200	7304	1625	7481	7322	7312
		3	79	203	5783	7224	755	7582	7449	7615
		6	32	86	1260	5540	232	3411	4860	7263
		11	16	37	674	3058	162	2444	2943	7258
		21	**18**	**16**	303	1489	**120**	**777**	844	2572

4.5.4 Results of Stochastic Mean-Risk Programs

To investigate how much the risk measure contributes to computational difficulty, we consider a mean-risk variant of the considered risk-averse problem which aims to minimize a weighted average of the original risk and an expectation as:

$$\min\{w \cdot \rho(f(x,\xi)) + (1-w) \cdot \mathbb{E}[f(x,\xi)] : x \in X\}$$

where w is a constant between 0 and 1. The extended formulation (formerly model (4.4), given the generic coherent risk measure ρ) now becomes:

$$\min_{x \in X} \max_{q \in \mathcal{Q}_\rho(p)} \left\{ \sum_{k=1}^{K} (wq_k + (1-w)p_k) f_k(x) \right\}$$

to which all the proposed algorithms can easily adapt.

Table 4.6 compares the runtime of the risk-averse problem (i.e., $w = 1$) with its expectation-based counterpart (i.e., $w = 0$) under MWB. Noting that they are very close on most of the instances, we conclude that the proposed algorithm can solve the risk-averse problem with any coherent risk measure, as a transitional expectation-based stochastic program.

Table 4.6: Results of mean-risk model variants and its expectation counterpart under MWB

		SSLP_50		SSLP_100		SSLP_500		SSLP_1000	
	N	$w=1$	$w=0$	$w=1$	$w=0$	$w=1$	$w=0$	$w=1$	$w=0$
time	2	6	6	13	13	162	162	315	344
	3	3	3	6	6	78	76	138	138
	6	1	1	3	3	33	33	67	65
	11	1	1	2	2	26	25	41	48
	21	1	1	2	2	**24**	29	**34**	34
optimal obj	-	-253	-365	-248	-355	-246	-350	-250	-352

		SMKP_20		SMKP_40		SMKP_80		SMKP_160	
	N	$w=1$	$w=0$	$w=1$	$w=0$	$w=1$	$w=0$	$w=1$	$w=0$
time	2	3758	3814	6462	11930	2641	3651	7046	8706
	3	1514	1525	2938	5995	2400	3448	5834	7257
	6	584	583	987	2110	825	983	3418	4082
	11	327	332	510	964	684	846	2773	3656
	21	227	186	484	864	509	602	2269	3071
optimal obj	-	9357	9043	9550	9187	9475	9194	9572	9251

Table 4.7: Computational results of distributionally robust risk-averse problem under MWB

		SSLP_100			SSLP_1000			SMKP_80		
	N	$\upsilon=0$	$\upsilon=0.3$	$\upsilon=0.6$	$\upsilon=0$	$\upsilon=0.3$	$\upsilon=0.6$	$\upsilon=0$	$\upsilon=0.3$	$\upsilon=0.6$
time	2	315	311	311	6462	8016	7210	8162	8326	8565
	3	138	137	137	2938	2950	2866	5251	5444	5122
	6	67	70	70	987	992	977	2375	2475	2474
	11	41	42	42	510	641	640	1169	1245	1243
	21	34	34	34	484	660	681	826	796	799
optimal obj	-	-248	-245	-243	-250	-244	-239	9475	9484	9493

4.5.5 Results of Distributionally Robust Variants

As demonstrated in Section 4.3, the distributionally robust risk-averse problem can be viewed as a special case of the considered problem as long as we replace $\mathcal{Q}_\rho(p)$ with $\mathcal{D}_\rho(\mathcal{P})$. Here, we test solving a specific class of distributionally robust risk-averse problem with

$$\mathcal{P} = \left\{ p : \sum_{k=1}^{K} p_k = 1,\ (1-\upsilon)/K \le p_k \le (1+\upsilon)/K,\ \forall k \right\}$$

where υ is a constant between 0 and 1. We vary $\upsilon = 0.3$ and 0.6 on three instances (i.e., SSLP_100, SSLP_1000, and SMKP_80), and use MWB to solve each case. Table 4.7 shows the results. We also show the results of the corresponding run, given singleton-$\mathcal{P}$ (i.e., $\upsilon = 0.0$) as the reference.

We observe that the change in runtime as υ varies is very small. This is reasonable because the structure of $\mathcal{P}$ only affects the computation of **Cont**(β) in step 11 of Algorithm 4.6, which in this case is a simple LP with negligible computational time.

4.6 Concluding Remarks

In this chapter, we transformed a class of risk-averse 0-1 stochastic programs into risk-neutral minimax programs by exploiting the dual representation of coherent risk measures. We then developed three dual decomposition algorithms to solve the reformulations, following the common procedure of iteratively evaluating and cutting off candidate solutions discovered from scenario subproblems, until the gap between the upper and lower bounds closed. The three algorithms were different in how they recover lower bounds. DD1 used the functional values of the Lagrangian relaxation at zero, which was a weaker lower bound, although much easier to compute. DD2 and DD3 used the value of the Lagrangian dual, which was computed through a cutting-plane subroutine and a subgradient subroutine, respectively. Both subroutines were by themselves an iterative procedure, so DD2 and DD3 were double-loop algorithms. Our computation results suggested the following:

1. Unlike directly solving the risk-averse 0-1 program (e.g., the default scheme in Section 4.5.2), the dual decomposition algorithms, although slower in some small instances, exhibit larger speed advantages along with the increased number of scenarios.

2. The speeds of the three dual decomposition algorithms for the test instances considered are as follows:

   $$\text{DD2} < \text{DD3} < \text{DD1},$$

   which suggests that adjusting the dual multiplier through the cutting-plane method (DD2) or the cutting-plane method (DD3) is not effective in producing tighter lower bounds. In contrast, the extra loop in these two subroutines considerably increases the number of subproblems to solve, and significantly lengths the runtimes. Therefore, DDI is preferable due to its speed, simplicity, and amenability to parallelism.

To further speedup, we introduced three parallel schemes for DD1. In BP, all of the processes synchronously shared the computation of subproblems. In MWB, with the objective to avoid reevaluating duplicate solutions and abate computational workload, we dedicated one process, termed as the master, to collecting solutions and remove duplicates, which, however, resulted in one less process sharing the computation. This scheme was partially synchronous as we still used collective communication to pass solutions/cuts. In MWN, to avoid barriers we used an asynchronous cut-adding strategy that only required point-to-point communication between the master

and individual workers. Moreover, we deployed a "pull" mechanism in assigning subproblems to processes, which was a better load balancer than the "push" mechanism in the previous two schemes. Our computation results suggested the following:

1. Parallelism for DD1 has near-linear speedup, and in some cases even super-linear speedup due to the early detection of convergence achieved from spreading out the computation of subproblems.

2. When the number of processes is small, BP performs well. However, as it becomes relatively large, the performance is significantly compromised by the waiting caused by barriers.

3. MWB and MWN are quite complementary. When the number of scenarios is still small, MWB suffers from load imbalance, but MWN achieves good speedup due to the deployed pull mechanism. As the number of scenarios increases, the communication for job dispatching in MWN starts to compromise the performance, whereas MWB, which alleviates load imbalance, starts to perform well.

We also tested the algorithms on a mean-risk variant and a DR variant of the considered problem. The results suggested that:

1. Compared to the traditional risk-neutral stochastic programs, the risk-averse 0-1 stochastic program is not computationally more difficult.

2. All of the proposed algorithms solve the DR variant of the considered problem. Moreover, the shape and the size of the uncertainty set in a DR model have insignificant effects on the runtimes.

APPENDIX A

Appendix for Chapter 2

A.1 Model (2.14) is a relaxation of model (2.9)

Proof. Consider any feasible solution (x, y, z, s, δ) to (2.9). In each open OR j, we can recover the sequence of assigned surgeries based on $\hat{z}^j$ as $i_1, \ldots, i_{A_j}$, where A_j denotes the number of assigned surgeries. Then, for any scenario $\omega \in \Omega$,

$$\sum_{i \in \mathcal{S}} \xi_i^\omega y_{ji} = \sum_{k=1}^{A_j} \xi_{i_k}^\omega = \xi_{i_1}^\omega + \sum_{k=2}^{A_j} \xi_{i_k}^\omega \leq \xi_{i_1}^\omega + \sum_{k=2}^{A_j} (\delta_k^{j\omega} - \delta_{k-1}^{j\omega}) = \xi_{i_1}^\omega + \delta_{A_j}^{j\omega} - \delta_1^{j\omega} \leq \delta_{A_j}^{j\omega}$$

This implies that $\sum_{i \in \mathcal{S}} \xi_i^\omega y_{ji} \leq T_j$ is dominated by $\delta_{A_j}^{j\omega} \leq T_j$ for any $j \in \mathcal{R}$. □

APPENDIX B

Appendix for Chapter 3

B.1 Proof of Proposition 3.1

Proof. Consider some $x^* \in \mathcal{P}^* \setminus E$. There must exist some $z^* \in \{0,1\}^K$ satisfying (3.2c) such that (x^*, z^*) is feasible to (3.5). Construct $(\hat{x}^k, \hat{z}_k,\ \forall k = 1,\dots,K)$, where $\hat{x}^k = x^*,\ \hat{z}_k = z^*_k,\ \forall k \in \Omega$. It is clear that it is feasible to the minimization problem in the definition of $g(\rho, \lambda, S)$. Therefore,

$$\begin{aligned} g(\rho,\lambda,S) &\leq \sum_{k=1}^{K} f(\hat{x}^k)/K + (\alpha_k - \delta_k)\lambda^\top \hat{x}^k + \rho\left(\sum_{k=1}^{K} \hat{z}_k - K'\right) \\ &= f(x^*) + \rho\left(\sum_{k=1}^{K} z^*_k - K'\right) \\ &\leq f(x^*), \end{aligned}$$

which completes the proof that $g(\rho, \lambda, S)$ is a valid lower bound. □

APPENDIX C

Appendix for Chapter 4

C.1 Parallel Algorithm for DD2 and DD3

Unlike DD1, both DD2 and DD3 contain an inner loop that improves ℓ by adjusting objective-function parameters as opposed to shrinking the feasible region. Let Λ_k represent the concatenation of parameters that affect the scenario-k subproblem and vary in the inner loop. Let $h(\cdot)$ represent the function that maps $(\beta_k^{\mathrm{DD}i}, \hat{x}^k, \Lambda_k)_{k=1}^K$ to a tentative lower bound, and $r((\beta_k^{\mathrm{DD}i}, \hat{x}^k, \Lambda_k)_{k=1}^K) \geq 0$ represent the stop condition of the inner loop. We redescribe the double-loop structure as a single loop, and present a general parallel scheme for DD2 and DD3 in Algorithm C.1.

Algorithm C.1 The parallel scheme for DD2 and DD3 at Proc^n ($n \in \{1, \ldots, N\}$)

1: $u \leftarrow +\infty$, $\ell \leftarrow -\infty$
2: initialize $\Lambda_1, \ldots, \Lambda_K$
3: **repeat**
4: $\quad u_n \leftarrow +\infty$
5: $\quad$ **for** $k \in \Omega_n^{K,N}$ **do**
6: $\quad\quad (\beta_k^{\mathrm{DD}i}, \hat{x}^k) \leftarrow$ a scenario-k-based subproblem parameterized by Λ_k
7: $\quad\quad u_n \leftarrow \min\{u_n, \mathbf{Eval}(\hat{x}^k)\}$
8: $\quad$ **end for**
9: $\quad$ pass $\{(\beta_k^{\mathrm{DD}i}, \hat{x}^k) : k \in \Omega_n^{K,N}\}$ and u_n to all the other processes
10: $\quad \ell \leftarrow \max\{\ell, h((\beta_k^{\mathrm{DD}i}, \hat{x}^k, \Lambda_k)_{k=1}^K)\}$
11: $\quad u \leftarrow \min\{u, u_1, \ldots, u_N\}$
12: $\quad$ **if** $r((\beta_k^{\mathrm{DD}i}, \hat{x}^k, \Lambda_k)_{k=1}^K) < 0$ **then**
13: $\quad\quad$ update $\Lambda_1, \ldots, \Lambda_K$
14: $\quad$ **else**
15: $\quad\quad X \leftarrow X \setminus S$
16: $\quad\quad$ reset $\Lambda_1, \ldots, \Lambda_K$
17: $\quad$ **end if**
18: **until** $u - \ell < \epsilon$

This algorithm is similar to BP, but the difference is that it gives two options at the end of each iteration. If $r((\beta_k^{\mathrm{DD}i}, \hat{x}^k, \Lambda_k)_{k=1}^K) < 0$, we update $\Lambda_1, \ldots, \Lambda_K$ (i.e., step 13) which corresponds

to proceeding to the next iteration of the inner loop in the double-loop algorithm. Otherwise, we shrink the feasible region and reset $\Lambda_1, \ldots, \Lambda_K$ (i.e., steps 15,16) which corresponds to closing the inner loop and proceeding to the next iteration of the outer loop.

BIBLIOGRAPHY

Ahmed, S. (2006). Convexity and decomposition of mean-risk stochastic programs. *Mathematical Programming*, 106(3):433–446.

Ahmed, S. (2013). A scenario decomposition algorithm for 0-1 stochastic programs. *Operations Research Letters*, 41(6):565–569.

Ahmed, S., Garcia, R., Kong, N., Ntaimo, L., Parija, G., Qiu, F., and Sen., S. (2015a). SIPLIB: a stochastic integer programming test library. http://www2.isye.gatech.edu/~sahmed/siplib/.

Ahmed, S., Luedtke, J., Song, Y., and Xie, W. (2015b). Nonanticipative duality, relaxations, and formulations for chance-constrained stochastic programs. Available at Optimization-Online http://www.optimization-online.org/DB_HTML/2014/07/4447.html.

Ahmed, S., Tawarmalani, M., and Sahinidis, N. V. (2004). A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, 100(2):355–377.

Angulo, G., Ahmed, S., and Dey, S. S. (2014a). Improving the integer L-shaped method.

Angulo, G., Ahmed, S., Dey, S. S., and Kaibel, V. (2014b). Forbidden vertices. *Mathematics of Operations Research*, 40(2):350–360.

Artzner, P., Delbaen, F., Eber, J.-M., and Heath, D. (1999). Coherent measures of risk. *Mathematical finance*, 9(3):203–228.

Ben-Tal, A., den Hertog, D., De Waegenaere, A., Melenberg, B., and Rennen, G. (2013). Robust solutions of optimization problems affected by uncertain probabilities. *Management Science*, 59(2):341–357.

Beraldi, P. and Bruni, M. E. (2010). An exact approach for solving integer problems under probabilistic constraints with random technology matrix. *Annals of operations research*, 177(1):127–137.

Birge, J. R., Donohue, C. J., Holmes, D. F., and Svintsitski, O. G. (1996). A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs. *Mathematical Programming*, 75(2):327–352.

Birge, J. R. and Louveaux, F. (2011). *Introduction to stochastic programming*. Springer Science & Business Media.

Calafiore, G. and El Ghaoui, L. (2006). On distributionally robust chance-constrained linear programs. *Journal of Optimization Theory and Applications*, 130(1):1–22.

Cardoen, B., Demeulemeester, E., and Beliën, J. (2010). Operating room planning and scheduling: A literature review. *European Journal of Operational Research*, 201(3):921–932.

Carøe, C. C. and Schultz, R. (1998). *A two-stage stochastic program for unit commitment under uncertainty in a hydro-thermal power system*. ZIB.

Carøe, C. C. and Schultz, R. (1999). Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24(1):37–45.

Carøe, C. C. and Tind, J. (1998). L-shaped decomposition of two-stage stochastic programs with integer recourse. *Mathematical Programming*, 83(1-3):451–464.

Collado, R. A., Papp, D., and Ruszczyński, A. (2012). Scenario decomposition of risk-averse multistage stochastic programming problems. *Annals of Operations Research*, 200(1):147–170.

Cooper, W. L., Homem-de Mello, T., and Kleywegt, A. J. (2006). Models of the spiral-down effect in revenue management. *Operations Research*, 54(5):968–987.

Crainic, T. G., Fu, X., Gendreau, M., Rei, W., and Wallace, S. W. (2011). Progressive hedging-based metaheuristics for stochastic network design. *Networks*, 58(2):114–124.

Crainic, T. G., Hewitt, M., and Rei, W. (2014). Scenario grouping in a progressive hedging-based meta-heuristic for stochastic network design. *Computers & Operations Research*, 43:90–99.

Delage, E. and Ye, Y. (2010). Distributionally robust optimization under moment uncertainty with application to data-driven problems. *Operations Research*, 58(3):595–612.

Dentcheva, D. and Römisch, W. (2004). Duality gaps in nonconvex stochastic optimization. *Mathematical Programming*, 101(3):515–535.

Denton, B., Viapiano, J., and Vogl, A. (2007). Optimization of surgery sequencing and scheduling decisions under uncertainty. *Health care management science*, 10(1):13–24.

Denton, B. T. and Gupta, D. (2003). A sequential bounding approach for optimal appointment scheduling. *IIE Transactions*, 35(11):1003–1016.

Denton, B. T., Miller, A. J., Balasubramanian, H. J., and Huschka, T. R. (2010). Optimal allocation of surgery blocks to operating rooms under uncertainty. *Operations Research*, 58(4):802–816.

Erdoğan, E. and Iyengar, G. (2006). Ambiguous chance constrained problems and robust optimization. *Mathematical Programming*, 107(1-2):37–61.

Erdogan, S. A. and Denton, B. T. (2011). Surgery planning and scheduling. In Cochran, J., Cox, L., Keskinocak, P., Kharoufeh, J., and Smith, J., editors, *Wiley Encyclopedia of Operations Research and Management Science*. Wiley Online Library.

Escudero, L. F., Kamesam, P. V., King, A. J., and Wets, R. J. (1993). Production planning via scenario modelling. *Annals of Operations Research*, 43(6):309–335.

Fernandez, A. (1995). The optimal solution to the resource-constrained project scheduling problem with stochastic task durations. *Unpublished doctoral dissertation, University of Central Florida*.

Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., et al. (2004). Open mpi: Goals, concept, and design of a next generation mpi implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 97–104. Springer.

Gade, D., Küçükyavuz, S., and Sen, S. (2014). Decomposition algorithms with parametric gomory cuts for two-stage stochastic integer programs. *Mathematical Programming*, 144(1-2):39–64.

Ghaoui, L. E., Oks, M., and Oustry, F. (2003). Worst-case value-at-risk and robust portfolio optimization: A conic programming approach. *Operations Research*, 51(4):543–556.

Goel, V. and Grossmann, I. E. (2006). A class of stochastic programs with decision dependent uncertainty. *Mathematical programming*, 108(2-3):355–394.

Goh, M., Lim, J. Y., and Meng, F. (2007). A stochastic model for risk management in global supply chain networks. *European Journal of Operational Research*, 182(1):164–173.

Gul, S., Denton, B. T., Fowler, J. W., and Huschka, T. R. (2011). Bi-criteria scheduling of surgical services for an outpatient procedure center. *Production and Operations Management*, 20(3):406–417.

Günlük, O. and Pochet, Y. (2001). Mixing mixed-integer inequalities. *Mathematical Programming*, 90(3):429–457.

Gupta, D. (2007). Surgical suites' operations management. *Production and Operations Management*, 16(6):689–700.

Hayes, L. J., O'Brien-Pallas, L., Duffield, C., Shamian, J., Buchan, J., Hughes, F., Laschinger, H. K. S., North, N., and Stone, P. W. (2006). Nurse turnover: A literature review. *International Journal of Nursing Studies*, 43(2):237–263.

Higle, J. L. and Sen, S. (1991). Stochastic decomposition: An algorithm for two-stage linear programs with recourse. *Mathematics of operations research*, 16(3):650–669.

Jiang, R. and Guan, Y. (2015). Data-driven chance constrained stochastic program. To appear in *Mathematical Programming*. Available at Optimization-Online: http://www.optimization-online.org/DB_FILE/2013/09/4044.pdf.

Kleywegt, A., Shapiro, A., and Homem-de Mello, T. (2002). The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502.

Kong, N., Schaefer, A. J., and Hunsaker, B. (2006). Two-stage integer programs with stochastic right-hand sides: A superadditive dual approach. *Mathematical Programming*, 108(2-3):275–296.

Kong, Q., Lee, C.-Y., Teo, C.-P., and Zheng, Z. (2013). Scheduling arrivals to a stochastic service delivery system using copositive cones. *Operations Research*, 61(3):711–726.

Kouwenberg, R. (2001). Scenario generation and stochastic programming models for asset liability management. *European Journal of Operational Research*, 134(2):279–292.

Küçükyavuz, S. (2012). On mixing sets arising in chance-constrained programming. *Mathematical Programming*, 132(1-2):31–56.

Langer, A., Venkataraman, R., Palekar, U., Kale, L. V., and Baker, S. (2012). Performance optimization of a parallel, two stage stochastic linear program: The military aircraft allocation problem. In *Proceedings of the 18th International Conference on Parallel and Distributed Systems (ICPADS 2012). To Appear, Singapore*.

Laporte, G. and Louveaux, F. V. (1993). The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 13(3):133–142.

Lee, J. (2003). Cropped cubes. *Journal of combinatorial optimization*, 7(2):169–178.

Linderoth, J. and Wright, S. (2003). Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24(2-3):207–250.

Liu, X., Küçükyavuz, S., and Luedtke, J. (2015). Decomposition algorithms for two-stage chance-constrained programs. To appear in *Mathematical Programming*.

Lubin, M., Martin, K., Petra, C. G., and Sandıkçı, B. (2013). On parallelizing dual decomposition in stochastic integer programming. *Operations Research Letters*, 41(3):252–258.

Luedtke, J. (2014). A branch-and-cut decomposition algorithm for solving chance-constrained mathematical programs with finite support. *Mathematical Programming*, 146(1-2):219–244.

Luedtke, J. and Ahmed, S. (2008). A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization*, 19(2):674–699.

Luedtke, J., Ahmed, S., and Nemhauser, G. (2010). An integer programming approach for linear programs with probabilistic constraints. *Mathematical Programming*, 122(2):247–272.

Mak, H.-Y., Rong, Y., and Zhang, J. (2015). Appointment scheduling with limited distributional information. *Management Science*, 61(2):316–334.

Markowitz, H. M. (1968). *Portfolio selection: Efficient diversification of investments*, volume 16. Yale university press.

Miller, N. and Ruszczyński, A. (2011). Risk-averse two-stage stochastic linear programming: modeling and decomposition. *Operations Research*, 59(1):125–132.

Min, D. and Yih, Y. (2010). Scheduling elective surgery under uncertainty and downstream capacity constraints. *European Journal of Operational Research*, 206(3):642–652.

Nielsen, S. S. and Zenios, S. A. (1997). Scalable parallel benders decomposition for stochastic linear programming. *Parallel Computing*, 23(8):1069–1088.

Ntaimo, L. and Sen, S. (2005). The million-variable "march" for stochastic combinatorial optimization. *Journal of Global Optimization*, 32:385–400.

Nursing Solutions, Inc. (2013). 2013 National Healthcare and RN Retention Report. http://www.nsinursingsolutions.com/Files/assets/library/retention-institute/NationalHealthcareRNRetentionReport2013.pdf.

Pacheco, P. S. (1997). *Parallel programming with MPI*. Morgan Kaufmann.

Pagnoncelli, B., Ahmed, S., and Shapiro, A. (2009). Sample average approximation method for chance constrained programming: theory and applications. *Journal of Optimization Theory and Applications*, 142(2):399–416.

Pardo, L. (2005). *Statistical Inference Based On Divergence Measures*. CRC Press.

Peeta, S., Salman, F. S., Gunnec, D., and Viswanath, K. (2010). Pre-disaster investment decisions for strengthening a highway network. *Computers & Operations Research*, 37(10):1708–1719.

Pflug, G. C. and Römisch, W. (2007). *Modeling, measuring and managing risk*, volume 20. World Scientific.

Pollard, D. (1997). Distances and affinities between measures. Available at: http://www.stat.yale.edu/~pollard/Books/Asymptopia/Metrics.pdf.

Polyak, B. (1977). Subgradient methods: A survey of soviet research. In *Nonsmooth optimization: Proceedings of the IIASA workshop March*, pages 5–30.

Popescu, I. (2005). A semidefinite programming approach to optimal-moment bounds for convex classes of distributions. *Mathematics of Operations Research*, 30(3):632–657.

Qiu, F., Ahmed, S., Dey, S. S., and Wolsey, L. A. (2014). Covering linear programming with violations. *INFORMS Journal on Computing*.

Rockafellar, R. T. and Uryasev, S. (2000). Optimization of conditional Value-at-Risk. *Journal of Risk*, 2(3):21–42.

Rockafellar, R. T. and Uryasev, S. (2002). Conditional value-at-risk for general loss distributions. *Journal of banking & finance*, 26(7):1443–1471.

Rockafellar, R. T., Uryasev, S., and Zabarankin, M. (2006). Generalized deviations in risk analysis. *Finance and Stochastics*, 10(1):51–74.

Rockafellar, R. T., Uryasev, S. P., and Zabarankin, M. (2002). Deviation measures in risk analysis and optimization. *University of Florida, Department of Industrial & Systems Engineering Working Paper*, (2002-7).

Rockafellar, R. T. and Wets, R.-B. (1976). Nonanticipativity and l^1-martingales in stochastic optimization problems. In *Stochastic Systems: Modeling, Identification and Optimization II*, pages 170–187. Springer.

Ruszczyński, A. (1993). Parallel decomposition of multistage stochastic programming problems. *Mathematical programming*, 58(1-3):201–228.

Ruszczyński, A. (2013). Advances in risk-averse optimization. In *INFORMS Tutorials in Operations Research*. INFORMS.

Ruszczynski, A. P. and Shapiro, A. (2003). *Stochastic programming*, volume 10. Elsevier Amsterdam.

Ryan, K., Rajan, D., and Ahmed, S. (2015). Scenario decomposition for 0-1 stochastic programs: Improvements and asynchronous implementation. Available at Optimization-Online http://www.optimization-online.org/DB_FILE/2015/11/5201.pdf.

Ryan, S. M., Wets, R. J., Woodruff, D. L., Silva-Monroy, C., and Watson, J.-P. (2013). Toward scalable, parallel progressive hedging for stochastic unit commitment. In *Power and Energy Society General Meeting (PES), 2013 IEEE*, pages 1–5. IEEE.

Salmerón, J. and Apte, A. (2010). Stochastic optimization for natural disaster asset prepositioning. *Production and Operations Management*, 19(5):561–574.

Santoso, T., Ahmed, S., Goetschalckx, M., and Shapiro, A. (2005). A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research*, 167(1):96–115.

Scarf, H., Arrow, K., and Karlin, S. (1958). A min-max solution of an inventory problem. In *Studies in the Mathematical Theory of Inventory and Production*, volume 10, pages 201–209. Stanford University Press, Stanford, CA.

Sen, S. and Higle, J. L. (2005). The c 3 theorem and a d 2 algorithm for large scale stochastic mixed-integer programming: set convexification. *Mathematical Programming*, 104(1):1–20.

Sen, S. and Sherali, H. D. (1985). On the convergence of cutting plane algorithms for a class of nonconvex mathematical programs. *Mathematical Programming*, 31(1):42–56.

Shapiro, A. (1993). Asymptotic behavior of optimal solutions in stochastic programming. *Mathematics of Operations Research*, 18(4):829–845.

Shapiro, A. (2012). Minimax and risk averse multistage stochastic programming. *European Journal of Operational Research*, 219(3):719–726.

Shapiro, A. and Ahmed, S. (2004). On a class of minimax stochastic programs. *SIAM Journal on Optimization*, 14(4):1237–1249.

Shapiro, A., Dentcheva, D., et al. (2014). *Lectures on stochastic programming: modeling and theory*, volume 16. SIAM.

Shapiro, A. and Homem-de Mello, T. (2000). On the rate of convergence of optimal solutions of monte carlo approximations of stochastic programs. *SIAM journal on optimization*, 11(1):70–86.

Shylo, O. V., Prokopyev, O. A., and Schaefer, A. J. (2012). Stochastic operating room scheduling for high-volume specialties under block booking. *INFORMS Journal on Computing*, 25(4):682–692.

Solak, S., Clarke, J.-P. B., Johnson, E. L., and Barnes, E. R. (2010). Optimization of r&d project portfolios under endogenous uncertainty. *European Journal of Operational Research*, 207(1):420–433.

Song, Y. and Luedtke, J. R. (2013). Branch-and-cut approaches for chance-constrained formulations of reliable network design problems. *Mathematical Programming Computation*, 5(4):397–432.

Song, Y., Luedtke, J. R., and Küçükyavuz, S. (2014). Chance-constrained binary packing problems. *INFORMS Journal on Computing*, 26(4):735–747.

Van Parys, B. P., Goulart, P. J., and Kuhn, D. (2015). Generalized gauss inequalities via semidefinite programming. *Mathematical Programming*, pages 1–32.

Van Slyke, R. M. and Wets, R. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663.

Vanden Bosch, P. M. and Dietz, D. C. (2000). Minimizing expected waiting in a medical appointment system. *IIE Transactions*, 32(9):841–848.

Vandenberghe, L., Boyd, S., and Comanor, K. (2007). Generalized Chebyshev bounds via semidefinite programming. *SIAM Review*, 49(1):52.

Wang, J. and Shen, S. (2012). Risk and energy consumption tradeoffs in cloud computing service via stochastic optimization models. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 239–246. IEEE.

Wang, Q., Guan, Y., and Wang, J. (2012). A chance-constrained two-stage stochastic program for unit commitment with uncertain wind power output. *Power Systems, IEEE Transactions on*, 27(1):206–215.

Watson, J.-P., Wets, R. J., and Woodruff, D. L. (2010). Scalable heuristics for a class of chance-constrained stochastic programs. *INFORMS Journal on Computing*, 22(4):543–554.

Watson, J.-P. and Woodruff, D. L. (2011). Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8(4):355–370.

Weiss, E. N. (1990). Models for determining estimated start times and case orderings in hospital operating rooms. *IIE Transactions*, 22(2):143–150.

Wolsey, L. A. and Nemhauser, G. L. (2014). *Integer and combinatorial optimization*. John Wiley & Sons.

Yu, L.-Y., Ji, X.-D., and Wang, S.-Y. (2003). Stochastic programming models in financial optimization: A survey.

Zeng, B., An, Y., and Kuznia, L. (2014). Chance constrained mixed integer program: Bilinear and linear formulations, and benders decomposition. *arXiv preprint arXiv:1403.7875*.

Zhang, M. and Küçükyavuz, S. (2014). Finitely convergent decomposition algorithms for two-stage stochastic pure integer programs. *SIAM Journal on Optimization*, 24(4):1933–1951.

Zhang, M., Küçükyavuz, S., and Goel, S. (2014). A branch-and-cut method for dynamic decision making under joint chance constraints. *Management Science*, 60(5):1317–1333.

Zymler, S., Kuhn, D., and Rustem, B. (2013a). Distributionally robust joint chance constraints with second-order moment information. *Mathematical Programming*, 137(1-2):167–198.

Zymler, S., Kuhn, D., and Rustem, B. (2013b). Worst-case value at risk of nonlinear portfolios. *Management Science*, 59(1):172–188.