2017

Decomposition Algorithms in Stochastic Integer Programming: Applications and Computations.

Babak Saleck Pay

DECOMPOSITION ALGORITHMS IN STOCHASTIC INTEGER PROGRAMMING: APPLICATIONS AND COMPUTATIONS.

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Systems Modeling and Analysis) at Virginia Commonwealth University.

by

BABAK SALECK PAY

Director: Dr. Yongjia Song, Assistant Professor

Department of Statistical Sciences and Operations Research

Virginia Commonwealth University

Richmond, Virginia

July, 2017

Acknowledgements

I would like to express my deepest gratitude to all people who helped me during this journey.

First, I really thank my dear family, Baba, Moman, Abji and Madar (RIP). They were a true supporters and were there whenever I needed help. There is a strong emotional bound among us which neither time nor distance can break.

Second, I want to thank my dear adviser Dr. Yongjia Song. During the four years of PhD, I have learned a lot from him, either in his classrooms or in our research meeting. He helped me generously in every aspect of this job.

I also want to thank Dr. Jason Merrick for trusting and granting me the PhD position.

I want to express my gratitude to the other members of my committee: Dr. J Paul Brooks, Dr. José Dulá, and Dr. Qin Wang. They are great scholars in their communities and I am very proud to have this unique opportunity to work with them.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Abstract

DECOMPOSITION ALGORITHMS IN STOCHASTIC INTEGER PROGRAMMING: APPLICATIONS AND COMPUTATIONS.

By Babak Saleck Pay

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2017.

Director: Dr. Yongjia Song, Assistant Professor

Department of Statistical Sciences and Operations Research

In this dissertation we focus on two main topics. Under the first topic, we develop a new framework for stochastic network interdiction problem to address ambiguity in the defender risk preferences. The second topic is dedicated to computational studies of two-stage stochastic integer programs. More specifically, we consider two cases. First, we develop some solution methods for two-stage stochastic integer programs with continuous recourse; second, we study some computational strategies for two-stage stochastic integer programs with integer recourse.

We study a class of stochastic network interdiction problems where the defender has incomplete (ambiguous) preferences. Specifically, we focus on the shortest path network interdiction modeled as a Stackelberg game, where the defender (leader) makes an interdiction decision first, then the attacker (follower) selects a shortest path after the observation of random arc costs and interdiction effects in the network. We take a decision-analytic perspective in addressing probabilistic risk over network parameters, assuming that the defender's risk preferences over exogenously given probabilities can be summarized by the expected utility theory. Although the exact form of the utility function is ambiguous to the defender, we assume that a set of historical data on some pairwise comparisons made by the defender is available, which can be used to restrict the shape of the utility function. We use two different approaches to tackle this problem. The first approach conducts utility estimation and optimization separately, by first finding the best fit for a piecewise linear concave utility function according to the available data, and then optimizing the expected utility. The second approach integrates utility estimation and optimization, by modeling the utility ambiguity under a robust optimization framework following [4] and [44]. We conduct extensive computational experiments to evaluate the performances of these approaches on the stochastic shortest path network interdiction problem.

In third chapter, we propose partition-based decomposition algorithms for solving two-stage stochastic integer program with continuous recourse. The partition-based decomposition method enhance the classical decomposition methods (such as Benders decomposition) by utilizing the inexact cuts (coarse cuts) induced by a scenario partition. Coarse cut generation can be much less expensive than the standard Benders cuts, when the partition size is relatively small compared to the total number of scenarios. We conduct an extensive computational study to illustrate the advantage of the proposed partition-based decomposition algorithms compared with the state-of-the-art approaches.

In chapter four, we concentrate on computational methods for two-stage stochastic integer program with integer recourse. We consider the partition-based relaxation framework integrated with a scenario decomposition algorithm in order to develop strategies which provide a better lower bound on the optimal objective value, within a tight time limit.

CHAPTER 1

INTRODUCTION

1.1 Stochastic Linear/Integer Programming

Stochastic programming has firmly established itself as an indispensible modeling tool and solution approach to deal with optimization problems under uncertainty [10]. Among all the stochastic programming models, two-stage stochastic program is one of the most popular models that has been applied in a variety of application areas, see, e.g., [46, 21, 11, 101, 75, 102, 3]. In two-stage stochastic programs, the first-stage decisions, known as the here-and-now decisions, are made before the realization of the random parameters in the model. On the other hand, the second-stage decisions, known as the wait-and-see decisions, are made after resolutions of the random parameters are observed. The second-stage decisions are meant to compensate for the first-stage decisions, and therefore are also called the *recourse decisions*.

We present a two-stage stochastic linear program as follows:

$$z^* = \min_{x \in X} c^T x + \mathbb{E}[f(x, \xi)], \tag{1.1}$$

where

$$f(x,\xi) = \min\ d^T y \tag{1.2a}$$

$$\text{s.t. } \tilde{T}x + Wy \geq \tilde{h} \tag{1.2b}$$

$$y \in \mathbb{R}^{n_2}. \tag{1.2c}$$

In (1.1), $X \subset \mathbb{R}^{n_1}$ is a non-empty and closed set. In (1.2), the triple $(\Omega, \mathcal{F}, \mathcal{P})$ is a probability space in which $\xi = (\tilde{T}, \tilde{h})$ is defined and the closed set $\Xi \subset \mathbb{R}^D$ is the support for $\mathcal{P}$.

There are two basic assumptions upon which most of the methods in stochastic programming are built. The first assumption is that the probability distribution of ξ is known; but, in most cases, calculating the expected value $\mathbb{E}[f(x,\xi)]$ is computationally very intensive. Moreover, if decision maker accept a good approximate solution, then we might be able to avoid this intense calculation. Hence, we can follow the most common procedure in stochastic programming which is finding a discrete set of realization of random parameters. This set is used to approximate the true underlying probability distribution. The larger the set, the better the approximation. In stochastic programming literature, we call the set defined above as scenario set and the problems that use such a set are called scenario-based problems. The second assumption is the independence of random parameters ξ of decision variable x. See [35] for a class of problems that violate this assumption. Throughout this dissertation, we follow these two assumptions.

To obtain an approximate of $\mathbb{E}[f(x,\xi)]$, we can use Monte Carlo estimation.

Given a set of independent random samples $\xi^1, \xi^2, \cdots \xi^N$, we have:

$$f_N(x) = \frac{\sum_{k=1}^{N} f_k(x)}{|N|}, \tag{1.3}$$

where for each scenario $k \in N$, $f_k(x)$ represents the second-stage value function, which is given by:

$$f_k(x) := \min_{y^k \in \mathbb{R}_+^{n_2}} \left\{ d^\top y^k \ \mid \ T^k x + W y^k \geq h^k \right\}. \tag{1.4}$$

Note that $N = \{1, 2, \ldots, N\}$ is the the set of indices corresponding to each scenario. In model (1.4), $T^k \in \mathbb{R}^{m_2 \times n_1}$, and $h^k \in \mathbb{R}^{m_2}$ could be scenario-specific, while $d \in \mathbb{R}^{n_2}$ and $W \in \mathbb{R}^{m_2 \times n_2}$ are assumed to be fixed for all scenarios $k \in N$. Then, a two-stage stochastic program with a finite set N of equally probable scenarios, motivated by the sample average approximation (SAA) [10], can be formulated as:

$$\min_{x \in X_{LP}} \ c^\top x + \sum_{k=1}^{N} f_k(x), \tag{1.5}$$

where X_{LP} is a linear relaxation of a set given by $X = \left\{ x \in \mathbb{R}_+^{n_1 - p_1} \times \mathbb{Z}_+^{p_1} \mid Ax = b \right\}$, and $n_1 \geq p_1$. Set X does not need to be bounded, however first-stage problem must have a bounded solution.

The assumption that matrix W is fixed in is the so-called *fixed recourse* assumption. Without this assumption, the SAA may be unstable and ill-posed, as shown by Example 2.5 in [84]. The proposed partition-based algorithms depend on the structure of fixed recourse. Formulation (1.5) implicitly assumes that all scenarios happen with the same probability and this probability has been incorporated in the second-stage objective coefficient vector d. For simplicity of presentation, we also assume that the second-stage problem is feasible and bounded for any feasible first-stage solution $x \in X$, a property known as the *relative complete recourse*. However, we note that this assumption is not necessary for the proposed algorithms to work.

Putting problems from two stages together with all the scenarios $k \in N$ yields the so-called *extensive formulation* as follows:

$$\begin{aligned} z^* = \min\ & c^T x + \sum_{k \in N} d^T y^k && \text{(1.6a)} \\ \text{s.t.}\ & Ax = b && \text{(1.6b)} \\ & T^k x + W y^k \geq h^k,\ \forall k \in N, && \text{(1.6c)} \\ & x \in \mathbb{R}_+^{n_1 - p_1} \times \mathbb{Z}_+^{p_1},\ y^k \in \mathbb{R}_+^{n_2},\ \forall k \in N. && \text{(1.6d)} \end{aligned}$$

Although off-the-shelf commercial mixed integer programming (MIP) solvers can be used to directly solve (1.6), this is not an efficient approach since (1.6) is a large-scale MIP when a large number of scenarios is incorporated into the model to characterize uncertainty. Because of this difficulty, decomposition approaches have been studied extensively in the literature that could exploit the special structure of two-stage stochastic programs. In (1.6), the scenario-based constraints are linked together only through the first-stage variables. Therefore, given a fixed first-stage solution, (1.6) decomposes into $|N|$ separate (and smaller) problems, one for each scenario $k \in N$. This decomposability is the main building block of different decomposition methods developed for two-stage (and multi-stage) stochastic programs. The most popular decomposition algorithm applied to solve two-stage stochastic programs is the L-shaped method [94], which applies Benders decomposition [9] to (1.6). Along this line, there is a rich literature on various decomposition methods developed for two-stage stochastic programs such as stochastic decomposition [42], subgradient decomposition [82], regularized decomposition [72], level decomposition [103, 53], inexact bundle method [99, 25], etc. On the other hand, the adaptive partition-based algorithms have recently been proposed to directly mitigate the computational challenge in two-stage stochastic linear programs brought by the large number of scenarios [86]. Under the assumption of fixed recourse, the basic idea of these partition-based algorithms is to partition the scenario set into clusters, and construct a lower approximation for the second stage value function by aggregating constraints and variables for scenarios within the same cluster. This partition is then adaptively refined according to the optimal dual multipliers for each scenario obtained at certain trial points, until it is *sufficient*, i.e., the optimal solution obtained by solving the corresponding lower approximation is optimal to the original problem with all scenarios. Existence of a *sufficient partition* whose size is independent of the number of scenarios is shown by [92]. [92] also combines this scenario partition idea with Bender decomposition and level decomposition, using the concept of on-demand accuracy [25], and yields significant computational improvements.

Depending on whether or not the second-stage variables involve integer restrictions, there are two types of two-stage stochastic integer programs (SIP): SIP with continuous recourse and SIP with integer recourse. In this paper, we focus on the former one, in which case the aforementioned decomposition methods based on the cutting plane approach can still be applied because the second-stage problem is still a linear program (LP). Enhancements of the basic Benders decomposition that exploit the structure given by the first-stage integer variables have been successful recently. In [11], the authors take advantage of the integrality information of the first-stage solution and derive a new class of valid inequalities based on mixed integer rounding. In [12], two strategies that apply split cuts to exploit the integrality of the first-stage variables are proposed to yield strengthened optimality cuts. We note that these enhancements can also be incorporated in the proposed partition-based algorithms in a straightforward way.

1.1.1 Benders decomposition

Benders decomposition [see, e.g., 10] or the L-shaped method [94] is a well-known and widely accepted solution framework for solving two-stage stochastic linear programs. The idea of Benders decomposition is to iteratively approximate the epigraph of function $f_k(x)$, $F_k := \left\{(x, \theta^k) \in \mathbb{R}^{n_1} \times \mathbb{R} \mid \theta^k \geq f_k(x)\right\}, \forall k \in N$, by constructing a piece-wise convex relaxation defined by a set of valid inequalities that are generated during the algorithm. This relaxation, also called the *Benders master problem*, is as follows:

$$\min_{x \in X_{LP}} c^\top x + \sum_{k \in N} \theta^k \tag{1.7a}$$

$$\text{s.t. } \theta^k \geq G^k x + g^k, \ (G^k, g^k) \in \mathcal{G}^k, \ \forall k \in N, \tag{1.7b}$$

where $\mathcal{G}^k$ is a collection of optimality cuts which are valid inequalities that have been generated for each scenario $k \in N$ so far through the algorithm. The Benders master problem is a relaxation of (1.6) in that it contains only a partial set of constraints that are necessary to describe the set F_k. Given an optimal solution $(\hat{x}, \{\hat{\theta}^k\}_{k \in N})$

of the Benders master problem (1.7), the second-stage problem (1.4) is solved for $k \in N$ to generate Benders optimality cuts. Specifically, let $\hat{\lambda}^k$ be the corresponding optimal dual vector for scenario $k \in N$, the Benders optimality cut (1.7b) takes the form:

$$\theta^k \geq (h^k - T^k x)^\top \hat{\lambda}^k. \tag{1.8}$$

An alternative way of applying Benders decomposition to solve (1.6) is to maintain a single variable θ in (1.7) instead of one variable θ^k for each scenario $k \in N$:

$$\min_{x \in X_{LP}} \; c^\top x + \theta \tag{1.9a}$$

$$\text{s.t. } \theta \geq Gx + g, \; (G, g) \in \mathcal{G}, \tag{1.9b}$$

where $\mathcal{G}$ is a collection of the aggregated Benders optimality cuts (known as the L-shaped cuts):

$$\theta \geq \sum_{k \in N} (h^k - T^k x)^\top \hat{\lambda}^k. \tag{1.10}$$

A single cut (1.10), instead of (at most) one cut for each scenario $k \in N$, is generated at each iteration. As a result, Benders decomposition has two well-known variants: single-cut (1.10), and multi-cut (1.8).

While multi-cut Benders decomposition adds more information to the master problem at each iteration, it has to solve potentially a much larger master problem (1.7) compared to the single-cut Benders decomposition, although less iterations are expected for the algorithm to converge. We note that in order to integrate the partition-based approach within the branch-and-cut framework for solving two-stage SIPs with continuous recourse, it is more convenient to work with the single-cut variant of the Benders decomposition (see more details in Section 3.2). For more information about the trade-off between single-cut and multi-cut approach as well as other enhancements to Benders decomposition, see, e.g., [75, 103, 91, 51, 31, 98].

1.1.2 Level decomposition

It is well-known that cutting plane methods such as Benders decomposition suffer from instability [14, Example 8.7]. They might take "large jumps" even in the region that is close to the optimal solution and oscillate around it, which slows down the convergence. Regularization techniques mitigate this inefficiency of cutting plane approaches by keeping the next iteration close to the so-called *stabilization center*, which is usually defined as the incumbent solution encountered during the solution procedure (see, e.g., [43]). Regularization technique for cutting plane approaches can be categorized in two different classes: proximal bundle method [72] and level bundle method [53]. In the context of stochastic programming, proximal bundle method and level bundle method are known as *regularized decomposition* (see, e.g., [73]) and *level decomposition* (see, e.g., [99]), respectively. In this paper we leverage the level decomposition to stabilize the proposed decomposition algorithm. We next briefly review the basic idea of the level method.

Instead of the standard cutting plane model (1.7), the trial point for the next iteration is obtained by solving a quadratic master problem in level decomposition, which models the projection of the stabilization center onto a level set defined by the current cutting plane relaxation and a given level target parameter. By doing this, we keep the next iteration either close to the previous one or close to the incumbent solution depending on how the stabilization center is defined. In practice, the stabilization is usually defined as the incumbent solution, or is kept unchanged from the previous iteration whenever a substantial decrease on the upper bound is not obtained. The projection problem is formulated as the following quadratic program:

$$\min_{x \in X} \frac{1}{2}\|x - \bar{x}\|_2^2 \tag{1.11a}$$

$$\text{s.t. } c^T x + \sum_{k \in N} \theta^k \leq f_{lev} \tag{1.11b}$$

$$\theta^k \geq G^k x + g^k, \ (G^k, g^k) \in \mathcal{G}^k, \ \forall k \in N, \tag{1.11c}$$

where $\bar{x}$ is the stabilization center and f_{lev} is the level target. In the context of two-stage stochastic programming, after obtaining a trial point $\hat{x}$, we solve the scenario-based subproblem (1.4) for each scenario $k \in N$, update the best upper bound z^{ub} so far, and add a new optimality cut (1.11c) (if any violated) to (1.11). Then, we update the level set by setting $f_{lev} = \kappa z^{lb} + (1 - \kappa) z^{ub}$ for some parameter κ and solve the level master problem (1.11). Unlike the standard Benders decomposition, optimal objective value of the level master problem (1.11) does not yield a lower bound for the problem. Instead, an updated lower bound is obtained whenever the level set, i.e., the feasible region of model (1.11), is an empty set, in which case the lower bound is updated to the level target, f_{lev}.

1.1.3 Adaptive partition-based algorithms

We next briefly review the adaptive partition-based algorithms introduced in [86]. Assume that we have an initial partition of the scenarios $\mathcal{N} = \{P_1, P_2, ..., P_L\}$ where

$P_1 \cup P_2 \cup ... \cup P_L = N$ and $P_i \cap P_j = \emptyset \; \forall i, j \in \{1, \ldots, L\}, \; i \neq j$.

$$z^{\mathcal{N}} = \min \; c^T x + \sum_{\mathcal{P} \in \mathcal{N}} d^T y^{\mathcal{P}} \tag{1.12a}$$

$$\text{s.t. } T^{\mathcal{P}} x + W y^{\mathcal{P}} \geq h^{\mathcal{P}} \quad \forall \mathcal{P} \in \mathcal{N}, \tag{1.12b}$$

$$x \in X, \; y^{\mathcal{P}} \in \mathbb{R}_+^{n_2} \quad \forall \mathcal{P} \in \mathcal{N}, \tag{1.12c}$$

where $y^{\mathcal{P}} = \sum_{k \in \mathcal{P}} y^k$, $T^{\mathcal{P}} = \sum_{k \in \mathcal{P}} T^k$, and $h^{\mathcal{P}} = \sum_{k \in \mathcal{P}} h^k$. It is clear that the partition-based problem (1.12) is a relaxation of the original stochastic program (1.6). The goal of the partition-based framework is to identify a partition $\mathcal{N}$ such that the corresponding optimal solution $\hat{x}_{\mathcal{N}}$ of (1.12) either solves (1.6) exactly (in this case, the partition $\mathcal{N}$ is referred to as "sufficient" in [86]) or has the objective value that is sufficiently close to the optimal objective value of the original problem (1.6) according to some user-specified criterion. To achieve this goal, the partition-based framework solves a sequence of problems of the form (1.12) with adaptively "refined" partitions $\mathcal{N}$. A partition $\mathcal{N}'$ is a refinement of partition $\mathcal{N}$, if $\forall P' \in \mathcal{N}', P' \subseteq P$ for some $P \in \mathcal{N}$, and $|\mathcal{N}'| > |\mathcal{N}|$. In [86] and [92], partitions are refined according to the optimal second-stage dual solutions $\hat{\lambda}^k$ for each scenario $k \in N$. This is motivated by the observation that the gap between lower and upper bounds at a given trial solution $\hat{x}$ is caused by the mismatch among these dual solutions [86, Theorem 2.5]. After a partition refinement, the partition-based master problem gives a tighter relaxation of the original stochastic program. Algorithm 1 summarizes the basics steps of adaptive partition-based method.

```
$z^{ub} \leftarrow +\infty$, find an initial scenario partition $\mathcal{N}$
while $gap > \epsilon$ do
    Solve (1.12); obtain $\hat{x}$ and $z^{\mathcal{N}}$
    foreach $k \in N$ do
        Solve subproblem (1.4) with $\hat{x}$ and obtain an optimal dual solution $\hat{\lambda}^k$
    end
    Refine partition $\mathcal{N}$ by further partitioning each component $\mathcal{P} \in \mathcal{N}$ based
    on clustering $\{\hat{\lambda}^k\}_{k \in \mathcal{P}}$
    $z^{ub} \leftarrow \min\left\{z^{ub}, c^\top \hat{x} + \sum_{k \in N} f_k(\hat{x})\right\}$
    $gap \leftarrow \frac{z^{ub} - z^{\mathcal{N}}}{z^{\mathcal{N}}}$
end
```

Algorithm 1: Adaptive Partition-Based Algorithm

It is clear that this framework will always converge in a finite number of iterations, since there exists a trivial partition that is given by the original scenario set itself. The key for the partition-based framework to work well in practice is to develop an adaptive approach to find a small completely sufficient partition in an efficient manner, if any such partition exists. It has been proved in [92, Theorem 1] that there exists a sufficient partition for two-stage stochastic linear programs with fixed recourse, whose size is at most $n_1 - m_1 + |E|$, where n_1 and m_1 are the number of first-stage variables and constraints, respectively, and E is the set of extreme points of the dual polyhedron of (1.4), i.e., $\{\lambda \in \mathbb{R}^{m_2}_+ \mid W^\top \lambda \leq d\}$. Although the number of extreme points of the dual polyhedron is exponentially many, this number $|E|$ is independent of the total number of scenarios $|N|$, and thus may still be small compared to $|N|$. In [92], the adaptive partition-based algorithms are embedded into an overall cutting plane framework by generating coarse optimality cuts corresponding to a scenario partition first, and resorting to the exact Benders optimality (fine) cuts only when it is necessary to do so. In the meantime, the scenario partitions are refined so that these coarse cuts are asymptotically exact as well. This framework generates cuts whose qualities are adaptive to the solution procedure. This idea has also been applied in a level decomposition framework in [92]. More details along this line will be elaborated in Section 3.2.1.

In this work, following [86] and [92] we embed the adaptive partition-based overall cutting plane framework within the branch-and-cut framework for solving two-stage SIPs with continuous recourse. An immediate consequence of this extension is that there is no theoretical guarantee that a sufficient partition of a small size exists. In other words, it is possible that any partition of size less than the scenario size ,$|N|$, may yield an objective value that is strictly less than the true optimal value. However, we may still benefit from the adaptive partition-based framework by generating cheap coarse cuts early in the solution process, and gradually enhancing the cut quality using more efforts, which is adaptive to the solution progress.

1.2 Expected Utility

In this section we briefly review the basic assumptions of expected utility theory when completeness axiom is violated. We do not specifically contribute to the current literature of expected utility models, however, we utilize it in network interdiction model which is the subject of Chapter 2.

Von-Neumann and Morgenstern [95] proved that if a preference relation ($\succeq$)

satisfies a set of properties (axioms), then there exists an increasing utility function such that the decision makers preferences can be represented by expected utility. In other words, there exists a function $u : \mathbb{R} \to \mathbb{R}$, such that for any arbitrary pair of risky decisions X and Y (also called lotteries or gamble), decision maker prefers X to Y if and only if $\mathbb{E}[u(X)] \geq \mathbb{E}[u(Y)]$. The set of axioms are (this is the variant which Wakker listed in his book [96]):

1. $\succeq$ is a weak ordering:

   (a) $\succeq$ is transitive:

   $$X \succeq Y \text{ and } Y \succeq Z \text{ implies } X \succeq Z. \tag{1.13}$$

   (b) $\succeq$ is complete:

   $$\text{For all } X \text{ and } Y, \quad X \succ Y \quad \text{OR} \quad X \prec Y \quad \text{OR} \quad X \sim Y. \tag{1.14}$$

2. Standard-Gamble solvability: Suppose $X = [p : M, 1-p : m]$ is a gamble which yields M with probability p and m with probability $1-p$. Then, for all $m < \alpha < M$, there exists α such that:

   $$\alpha \sim X \tag{1.15}$$

3. Standard-Gamble dominance: Suppose $X = [p : M, 1-p : m]$ and $Y = [q : M, 1-q : m]$ be two different gambles such that $M > m$. For all $p > q$ we have:

   $$X \succ Y \tag{1.16}$$

4. Standard-Gamble consistency: Suppose $X = [p : M, 1-p : m]$, $Y = [\lambda : \alpha, 1-\lambda : C]$, and $Z = [\lambda : X, 1-\lambda : C]$, then we have:

$$\alpha \sim X \text{ implies } Y \sim Z. \tag{1.17}$$

If any of the properties above is not satisfied, then Von-Neumann and Morgenstern expected utility theorem, in its original form, does not hold any more. The violation of axioms is not a hypothetical assumption. Indeed, in real world setting, it is often very difficult to assume that all the axioms are true. Among these axioms, completeness is perhaps the most controversial one [6]. Indeed, Von-Neumann and Morgenstern originally mentioned that it may be more convincing that decision makers cannot decide on all pairs of lotteries. The completeness axiom states that the space of the lotteries is completely ordered. In other words, the decision maker is capable of choosing one lottery over the other one for any pairs of lotteries (or be neutral). As it was mentioned, this is not always the case. For instance, when we are dealing with utility elicitation procedure, we are usually limited by the number of questions we could ask from the decision maker; or, when the decision maker is a group of people, we are usually unable to uniquely determine the preferences over a given set of comparisons.

Despite their note about violation of completeness, Von-Neumann and Morgenstern never mentioned how to modify the theorem to address this issue. Later, in 1962, Aumann proved that when we do not have completeness, Von-Neumann and Morgenstern expected utility still holds, however, instead of a unique utility function, we will have a set of utility functions [6]. In Chapter 2, we propose a model for stochastic network interdiction problem where the available knowledge about the operators is incomplete. Therefore, we have a set of utility function which are aligned with our assumptions about decision maker's risk preferences. We base our model development on the framework proposed by Armbruster and Delage [4] which takes a robust optimization perspective to address the ambiguity on the shape of the utility function.

1.3 Network Interdiction Problem

Network interdiction problems involve two competing sides, a "leader" and a "follower". For the convenience of presentation, we use the word "she" to refer to a leader, and the word "he" to refer to a follower. The follower attempts to meet his desired objective (e.g., minimizing the cost of transporting illegal drugs through the network by picking a shortest path), while the leader modifies the parameters of the network (e.g., the traveling cost of each arc) to make it harder for the follower to achieve his objective, a decision known as interdiction. Depending on the follower's objective, network interdiction problems are classified as shortest-path interdiction, maximum-flow interdiction, maximum-reliability path interdiction, etc.

The network interdiction problem was originally proposed during the Cold War. Rand Corporation researchers sought a means of interdiction planning for former Soviet Union rail roads, in order to interfere rail traffic to Eastern Europe [39]. Readers can refer to [58, 33, 89, 70, 100] for origins of network interdiction models. Since then, researchers have applied network interdiction to various real-world problems by securing systems of great societal and economic importance such as power systems

[76, 77], transportation networks [88, 81, 32], cyber networks [63, 80], etc., as well as disrupting networks of harmful or illegal goods such as drug distribution network [57, 36], nuclear smuggling network [68, 60, 19, 67, 85], human trafficking network [48], etc.

There are many different variants of the network interdiction problem. We focus on the shortest path network interdiction problem, rooting our basic formulation in the model of [45]. Given a graph $G = (V, A)$, the inner problem corresponds to the follower finding a shortest path from a source node s to a target node t, after first having observed the leader's interdiction decision; the outer problem models the leader's interdiction decision in order to maximize the length of the shortest path chosen by the follower, given a limited budget for the interdiction. This max-min problem can be formulated as:

$$\max_{x \in X} \min_{y} \sum_{a \in A} (c_a + x_a d_a) y_a \tag{1.18a}$$

$$\text{s.t.} \quad \sum_{a \in FS(m)} y_a - \sum_{a \in RS(m)} y_a = \begin{cases} 1 & \text{if } m = s \\ -1 & \text{if } m = t \\ 0 & \text{o.w.} \end{cases} \tag{1.18b}$$

$$y_a \geq 0, \ \forall a \in A, \tag{1.18c}$$

where c_a is the traveling cost of arc a and d_a is the interdiction effect on arc a, $\forall a \in A$, $FS(m)$ is the set of arcs that leave node m, $RS(m)$ is the set of arcs that enter node m, $\forall m \in V$, and $X = \{x \in \{0,1\}^{|A|} \mid r^\top x \leq r_0\}$ is the budget constraint set, where r is the interdiction cost vector and r_0 is the available budget for the leader. Constraints (1.18b) are the flow-balance constraints for the shortest path problem. We can turn this max-min model into a mixed-integer program (MIP) model by fixing x, taking the dual of the inner problem, and then releasing x, yielding [45]:

$$\max_{x,\pi} \pi_t - \pi_s \tag{1.19a}$$

$$\text{s.t. } \pi_n - \pi_m - d_a x_a \le c_a, \ \forall a = (m,n) \in A \tag{1.19b}$$

$$\pi_s = 0 \tag{1.19c}$$

$$x \in X, \tag{1.19d}$$

where π_m is the dual variable corresponding to constraint (1.18b) for node $m \in V$.

Stochastic variants of the network interdiction problem have also been studied extensively in the literature. In stochastic network interdiction problems, a variety of uncertain sources of risk can emerge, for example, the leader may not necessarily know which two nodes are the source node s and target node t between which the follower wants to travel; the cost and interdiction effect of each arc could be uncertain as well, so that the leader may not know how much the cost of each arc will change after a particular interdiction plan is applied [59]. For the maximum-flow network interdiction problem, uncertainty may also be apparent in the capacities of arcs [22, 23]. [8, 61, 90] study the case where the leader and the follower may have different perceptions about the parameters of the network. [41, 40] study models in which the exact configuration of the network is not known and we only have a set of possible configurations; problems of this sort can appear in computer, defense, and drug transportation networks. Decision makers' risk preferences have also been incorporated into network interdiction models; for example, recently, [87] study the shortest path interdiction problem when the interdictor's risk aversion is modeled by a chance constraint. We consider a stochastic shortest path interdiction problem with uncertainty in the traveling cost, the interdiction effect on each arc, and, as explained in detail below, ambiguity in decision-makers' risk preferences.

We assume that the leader initially interdicts some arcs; the follower then observes the leader's interdiction decision, the resolution of the random cost and interdiction effect of each arc, and finally solves a deterministic shortest path problem. This is the same as the sequence of decisions given in [59]. Under this assumption, the leader contends with probabilistic risk because she makes her decision before the resolution of network-parameter uncertainty. Therefore, the decision could lead to desirable or undesirable outcomes, depending on the realizations of the network parameters. We develop our model to take explicit account of the leader's risk preferences and ambiguity in her knowledge of those preferences. The follower deals with a deterministic optimization problem in this case, and therefore no utility function is needed to model his decision. We suppose that there exists a utility function which can be used to summarize the leader's risk preferences, but that the leader is unsure which utility function is the true one, although she has some knowledge that constrains the shape of her true utility function. It may seem less natural to argue that the leader does not know her own utility function. However, people are often indecisive about expressing a clear opinion between risky alternatives, particularly when stakes are outlandish and difficult to imagine concretely, and many prominent utility theorists have identified completeness as particularly problematic for both descriptive and normative analysis. As [5] argued in his seminal work, for example, completeness "is perhaps the most questionable of all the assumptions of utility theory." This is particularly true when the leader consists of a group of decision makers. These considerations motivate us to explicitly model the partial knowledge that decision-makers have about their true utility functions. First, we assume that a set of pairwise comparisons between given pairs of gambles is available, for example, by observing past decisions made, or by discussing acceptable comparisons in the committee, etc. In addition, we suppose that the decision makers are willing to make further, common conjectures about their utility functions, e.g., that they are monotonically nondecreasing (with respect to path costs) and concave [62, 18]. These important characteristics of real-world decision-making have not been addressed by standard network interdiction literature, to the best of our knowledge.

As discussed above, the ambiguity about the true utility function is modeled by a set of constraints on the form of the utility function according to the available historical data and some common assumptions. One way to deal with this ambiguity is to find a utility function that best fits the available knowledge according to some criterion. Alternatively, one could deal with the optimization problem under the ambiguity about the utility function in a robust optimization fashion, i.e., the leader maximizes her worst-case utility [4, 44]. The special case in which the set of possible utility functions is a singleton corresponds to exact or complete knowledge of the true utility function [18]. Incorporating the incomplete knowledge theme in network interdiction problem was also addressed in [15]. Nevertheless, there are some fundamental differences between our contribution and what authors proposed in [15]. In their work, the leader has incomplete knowledge about the structure of the network and its parameters (e.g., traveling cost) and she learns more about the network (in a sequential manner) through a set of interactions with the follower. On the contrary, we assume that both the leader and the follower have complete knowledge about the topology of the network. Moreover, the leader's uncertainty about the network parameters is represented as a random variable with a known probability distribution (source of uncertainty) and the incomplete knowledge that we consider is toward the shape of her utility function.

CHAPTER 2

STOCHASTIC NETWORK INTERDICTION WITH INCOMPLETE PREFERENCE

2.1 Introduction

A common theme in research in defense and homeland security is prescribing an optimal decision to the defender given the adaptation of the attacker [16]. A good example of this set-up is the important problem of network interdiction, where the attacker seeks to reach a target and the defender allocates resources to make this as hard as possible (see, e.g., [60, 16, 85]). Given a network of possible paths of any complexity, this requires an optimization for the path taken by the attacker and an optimization for the defender to allocate available resources to interdict the attacker's most preferred paths [100].

Given the nature of this problem, either party may not have complete information about their opponent or even the underlying network, so the research on this area has concentrated on stochastic network interdiction [34, 22, 23, 41, 61, 8, 59]. In the field of decision analysis, the presence of uncertainties in a prescriptive setting requires the consideration of risk preferences [47, 27]. Such considerations have recently been studied in network interdiction with a risk averse defender [87]. However, neither party will have complete information about the other's risk preferences and even their own risk preferences. Previous decisions made by either the attacker or the defender will reveal some information about their risk preferences, but this information may not be complete [64].

In this chapter, we focus on a situation in stochastic network interdiction when the defender's incomplete preference information can be revealed from historical data on decisions that they have made in the past. We propose two different approaches to tackle this problem. The first approach follows the traditional decision-analytic perspective, which separates the utility learning and optimization. Specifically, we first fit a piecewise linear concave utility function according to the information revealed from historical data, and then optimize the interdiction decision that maximizes the expected utility using this utility function. The second approach integrates utility estimation and optimization by modeling the utility ambiguity under a robust optimization framework, following [4] and [44]. Instead of pursuing the utility function that best fits the historical data, this approach exploits information from the historical data and formulates constraints on the form of the utility function, and then finds an optimal interdiction solution for the worst case utility among all feasible utility functions. The approaches proposed in this chapter forge an interdisciplinary connection between the areas of network interdiction and decision analysis. We use synthetic data to model empirically developed knowledge about the decision-makers' preferences. The models we present are in this respect intended to be data-driven, an advantage made possible precisely because of their interdisciplinary nature.

The organization of this chapter is as follows. In Section 2.2.1 we discuss the groundwork of expected utility model with incomplete knowledge and introduce the required notations which will be used throughout the chapter. In section2.2.2 we review the classical way of fitting a utility function. In our case, we find the best utility function which fit the incomplete knowledge we have about the decision maker. In Section 2.3 we discuss the robust expected utility model and then present two proposed approaches for stochastic network interdiction problem: robust utility model and robust certainty equivalent model. Detailed numerical results and discussion around the performance of different approaches are presented in section 2.4. We introduce an extension to the proposed approaches in Section 2.5.

In Section 2.2 we describe the proposed approaches. In Section 2.4, we present numerical results for the two different approaches that we study on stochastic shortest path interdiction instances. We discuss some extensions of the proposed approaches in Section 2.5 and then we close with some concluding thoughts in Section 2.6.

2.2 Stochastic Shortest Path Interdiction with Incomplete Defender's Preference

In this section we first present some preliminaries on the expected utility theory and incomplete preference. Then we introduce two different approaches that we propose to solve the stochastic shortest path interdiction problem with incomplete defender's preference. The first approach first fits a piecewise utility function given the incomplete knowledge about the defender's preference exploited from historical data, and incorporates this utility function in the context of optimizing the expected utility. The second approach models the optimization problem with incomplete preference under a robust optimization framework motivated by [4, 44]. In particular, we consider the robust expected utility model and the robust certainty equivalent model, and present how they specialize into the stochastic shortest path network interdiction problem with incomplete defender's preference.

2.2.1 Preliminaries: expected utility and incomplete preference

In this section we introduce the foundations of the proposed approaches for stochastic shortest path interdiction where the leader has incomplete preference. We start by formulating the expected utility maximization problem as a stochastic program, and then describe how the ambiguity about the utility function is modeled based on the available historical data.

Stochastic programming is a classic method for solving problems with uncertain parameters (considered as random variables) generated from known probability distributions. In practice, a finite number of samples are drawn from the joint distribution of uncertain parameters, resulting in a sample average approximation (SAA) of the problem, which is in turn solved in lieu of the original problem; in this respect, our models are applicable even to distributions with infinite support, so long as the SAA can be used to generate a reasonable approximation to the underlying problem [10]. Let $h(x, \xi_k)$ be the random return (ξ is the vector of random parameters and x is the vector of decision variables) for scenario k, corresponding to a particular realization ξ_k of the original problem's uncertain parameters; the objective function of the SAA problem is $\mathbb{E}[h(x, \xi)] = \sum_{k \in K} p_k h(x, \xi_k)$, where K is the finite set of scenarios and $p_k = \mathbb{P}(\xi = \xi_k)$, $\forall k \in K$. We use the expected utility theory to model decision-makers' risk preferences, as previously described. With this in mind, if we substitute the decision-maker's utility function u for the random return h, the following stochastic program maximizes the decision maker's expected utility:

$$\max_{x \in X} \ \mathbb{E}[u(h(x,\xi))], \tag{2.1}$$

where x is the decision vector, ξ is the vector of uncertain parameters of the problem, and $h(x,\xi)$ is the return of decision x, contingent on the random vector ξ. In the following, we assume that our decision-maker does not know the exact form of the utility function $u(\cdot)$, but we wish to model that our decision-maker has some partial knowledge which constrains the true shape of the utility function $u(\cdot)$.

We characterize incomplete knowledge of the utility function by a set U, which consists of all possible non-decreasing concave utility functions consistent with decision-makers' prior knowledge of $u(\cdot)$ according to historical data on a set of pairwise comparisons made by the decision-makers. Formally, set U can be characterized as follows:

$$U_2 = \{u : u \text{ is non-decreasing and concave}\} \tag{2.2a}$$

$$U_n = \{u : \mathbb{E}[u(W_0)] - \mathbb{E}[u(Y_0)] = 1\} \tag{2.2b}$$

$$U_a = \{u : \mathbb{E}[u(W_\ell)] \geq \mathbb{E}[u(Y_\ell)], \forall \ell = 1, ..., L\}, \tag{2.2c}$$

where $W_0, W_1, \ldots, W_L,\ Y_0, Y_1, \ldots, Y_L$ each represents a different gamble or probabilistic lottery over possible outcomes. The pairs of comparisons between these gambles given in the set U_a reveal knowledge of the true utility function u acquired by observing past choices. In our context, W_ℓ and Y_ℓ are two different interdiction plans. If they appear in U_a, then the decision-maker has observed in the past that under the true utility function W_ℓ is weakly preferred to Y_ℓ. Note that each interdiction plan is a gamble indeed. Because under different realization of random parameters (i.e., different scenarios), we will have different outcome (i.e., length of the shortest path that the follower chooses). Set U_2 expresses that the true utility function exhibits risk aversion. U_n encodes an assumption of scale; if we assume that W_0 is the gamble with the highest expected utility and Y_0 is the gamble with the lowest expected utility among all possible lotteries, then U_n requires that the utility of each other gamble is normalized to the interval $[0, 1]$. For simplicity, we assume that W_0 and Y_0 have deterministic outcomes, representing the largest and lowest outcome among all possible gambles, respectively. We then define U, the set of all possible utility functions consistent with prior knowledge, as the intersection of these three sets. We denote the union of the support of all gambles $\{W_\ell, Y_\ell\}_{\ell=1}^{L}$ as $Z = \bigcup_{\ell \in L} \{\text{supp}(Y_\ell) \cup \text{supp}(W_\ell)\}$, and we index this set Z by $Z = \{\bar{z}_j\}_{j \in T}$.

Before proceeding to the stochastic network interdiction formulation, we summarize what type of information the leader has. As we mentioned before, the leader has a full knowledge about the topology of the network. However, she is uncertain about the traveling cost and the interdiction effect of each arc. This uncertainty will be modeled as a random variable with a known probability distribution. Moreover, leader has an incomplete knowledge about the shape of her utility function. The source of this knowledge is the previous interactions that she had with the follower. This information is provided in the form of pair-wise comparisons of two different interdiction plan (W_ℓ, Y_ℓ). We should mention that, the pairs of (W_ℓ, Y_ℓ) does not need to be limited to the interdiction plans for the current network. Indeed, we can use another types of information obtained from same group of decision makers who has been playing different game on the network (historical data). However, in our specific case, we are modeling the leader side. Therefore, we can access to the leader and follow a common utility elicitation procedure by asking her preference about different pairs of interdiction plans. Since we know that completeness axiom is very likely to be violated, we need a framework which can take this in to account.

2.2.2 Fitting the utility function using piecewise linear concave functions

One of the most natural ideas for optimization with an ambiguous utility function is to first find a utility function of a parametric form that best fits the available incomplete preference information, and then optimize the expected utility using this specific utility function. Utility estimation and optimization are thus separated. In this section, we propose to find such best fit among all piecewise linear concave functions. Theoretically, utility functions of any parametric form can be used in this framework, and we chose the family of piecewise linear concave functions out of their computational convenience in both fitting and optimization.

Since the utility function under consideration is piecewise linear, its form totally depends on the utility value $\alpha_j, j \in T$ at each realization $\bar{z}_j, j \in T$. Therefore, the expected utility for each gamble W_ℓ and Y_ℓ is $\sum_{j \in T} \mathbb{P}[W_\ell = \bar{z}_j]\alpha_j$ and $\sum_{j \in T} \mathbb{P}[Y_\ell = \bar{z}_j]\alpha_j$, respectively. The set of α_j's are compatible with the available knowledge according to (2.2), if there exists a vector $\beta \in \mathbb{R}_+^T$ such that:

$$\sum_{j \in T} (\mathbb{P}[W_0 = \bar{z}_j]\alpha_j - \mathbb{P}[Y_0 = \bar{z}_j]\alpha_j) = 1 \tag{2.3a}$$

$$\sum_{j \in T} (\mathbb{P}[W_\ell = \bar{z}_j]\alpha_j - \mathbb{P}[Y_\ell = \bar{z}_j]\alpha_j) \geq 0, \ \forall \ell = 1, 2, \ldots, L \tag{2.3b}$$

$$(\alpha_{j+1} - \alpha_j) \geq \beta_j(\bar{z}_{j+1} - \bar{z}_j), \ \forall j = 0, 1, \ldots, T \tag{2.3c}$$

$$(\alpha_{j+1} - \alpha_j) \leq \beta_{j+1}(\bar{z}_{j+1} - \bar{z}_j), \ \forall j = 0, 1, \ldots, T - 1 \tag{2.3d}$$

$$\alpha \in \mathbb{R}_+^{T+1}, \tag{2.3e}$$

where constraint (2.3c) and (2.3d) enforce the value of α so that the corresponding utility function is non-decreasing and concave.

To find the utility function that best fits the available knowledge on the incomplete preference among all piecewise linear concave utility functions, we first compute an upper bound $\bar{\alpha}_j$ and a lower bound $\underline{\alpha}_j$ on the utility value α_j at each realization $\bar{z}_j, j = 0, 1, \ldots, T$, as long as they satisfy (2.3). Since the true utility at each realization $\bar{z}_j$ must be between these two bounds, we compute the average of these two bounds, $\tilde{u}_j = 0.5(\bar{\alpha}_j + \underline{\alpha}_j), \ \forall j = 1, \ldots, T$, and find a piecewise linear concave utility function whose values at realizations $\bar{z}_j, j = 1, \ldots, T$ are the closest to $\tilde{u}_j, j = 1, \ldots, T$. This approach is also used, e.g., in [4] as a benchmark to compare with their proposed robust utility framework. Specifically, these two bounds $\bar{\alpha}_j$ and $\underline{\alpha}_j$ can be calculated by solving $2 \times T$ linear programs: $\bar{\alpha}_j = \max\{\alpha_j \mid (2.3)\}$ and $\underline{\alpha}_j = \min\{\alpha_j \mid (2.3)\}$ for $j = 1, 2, \ldots, T$. Then the best fit can be found by solving the following quadratic program:

$$\min \left\{ \sum_{j \in T} (\alpha_j - \tilde{u}_j)^2 \mid \alpha \text{ satisfies (2.3)} \right\}. \tag{2.4}$$

Given an optimal solution of (2.4), the piecewise linear utility function that will be used in the optimization can be represented as: $u(z) = \min_{i \in I}\{v_i z + w_i\}$, where I is the number of pieces of this function. This utility function is then plugged into the stochastic program (2.1) to maximize the expected utility:

$$\max_{x \in X} \frac{1}{K} \sum_{k \in K} \min_{i \in I} \{v_i(h(x, \xi_k)) + w_i\}, \tag{2.5}$$

where $h(x, \xi^k)$ is the length of the shortest path that the follower chooses in each scenario $k \in K$. Note that $h(x, \xi^k)$ can also be seen as a piecewise linear concave function of x, we introduce a new variable θ^k for each scenario $k \in K$ and reformulate (2.5) as follows:

$$\max \sum_{k \in K} \frac{1}{K} \theta^k \tag{2.6a}$$

$$\text{s.t. } \theta^k \leq v_i \left(\sum_{a \in P} (\tilde{c}_a^k + \tilde{d}_a^k x_a) \right) + w_i \quad \forall P \in \mathcal{P}, i \in I, k \in K \tag{2.6b}$$

$$x \in X, \tag{2.6c}$$

where $\mathcal{P}$ is the set of all s-t paths in the network. To solve model (2.6) we can use a branch-and-cut algorithm. In model (2.6), constraints (2.6b) for each piece $i \in I$ and each scenario $k \in K$ should be satisfied for all s-t paths in the network. However, there are exponentially many such paths, which makes it inefficient or even impractical to include them in the model all at once. This motivates us to solve model (2.6) with an initial set of paths $\mathcal{P}' \subseteq \mathcal{P}$ (possibly $\mathcal{P}' = \emptyset$) using a branch-and-bound approach. Whenever an integral relaxation solution ($\hat{x} \in \{0,1\}^{|A|}$) is obtained during the branch-and-bound tree, we solve a shortest path problem for each scenario k with $c_a^k + d_a^k \hat{x}_a$ being the cost of each arc a, and obtain an s-t path $\hat{P}$. We then check if any constraint (2.6b) for this solution $\hat{x}$ and scenario k is violated with $\hat{P}$ being the s-t path used in the constraint. If so, we add this constraint as a lazy constraint to the MIP solver to cut off solution $\hat{x}$.

Model (2.6) with a large number of pieces $|I|$ could be computationally demanding to solve. In fact, I could be as large as $T-1$ in the worst case, resulting in $(T-1) \times K$ constraints for each path $P \in \mathcal{P}$ in model (2.6). However, we observed in our experiments that among these $T-1$ pieces, a large number of them could be eliminated if their corresponding slopes are close enough to each other (more details are provided in Section 2.4). A simple heuristic can be devised to reduce the number of pieces to, e.g., a constant $N' < T-1$ by tuning a threshold δ, where consecutive pieces are merged if their slopes differ by no more than δ. However, this does not guarantee that the best fitted utility function among all piecewise linear concave functions with at most N' pieces can be found in this way. To do so, we propose the following formulation instead:

$$\min \sum_{j \in T} (\alpha_j - \tilde{u}_j)^2 \tag{2.7a}$$

$$\text{s.t. } \exists \beta \in \mathbb{R}^T_+ \text{ s.t. } (\alpha, \beta) \text{ satisfies (2.3)} \tag{2.7b}$$

$$\alpha_j = \min_{i=1,\ldots,N'} \{v_i \bar{z}_j + w_i\}, \ \forall j \in T \tag{2.7c}$$

$$v_i \geq 0, \ i = 1, \ldots, N'. \tag{2.7d}$$

To handle constraint (2.7c) for each $j \in T$, we introduce a new binary variable for each piece of the utility function and replace (2.7c) by the following set of constraints:

$$\alpha_j \leq v_i \bar{z}_j + w_i \quad \forall i \in N, j \in T \tag{2.8a}$$

$$\alpha_j \geq (v_i \bar{z}_j + w_i) - M(1 - t_{ij}) \quad \forall i \in N, j \in T \tag{2.8b}$$

$$\sum_{i \in \{1,\ldots,N'\}} t_{ij} = 1 \quad \forall j \in T \tag{2.8c}$$

$$t_{ij} \in \{0, 1\}, \ \forall i \in \{1, \ldots, N'\}, \ j \in T, \tag{2.8d}$$

where M is an upper bound for all pieces within range $[Y_0, W_0]$. A valid value for M can be set as, e.g., $\frac{1}{\delta^*} W_0 + 1$ where δ^* is the smallest element in the set $\Delta = \{\bar{z}_{j+1} - \bar{z}_j \mid j = 1, \ldots, T - 1\}$, since the slope of any piece is bounded by $\frac{1}{\delta^*}$. Formulation (2.8) can possibly be improved, e.g., using disjunctive programming [7], however, as we indicate in Section 2.4, solving (2.8) to get the best fit piecewise linear concave function with up to N' pieces is not a bottleneck in the entire procedure.

2.3 Robust expected utility framework with incomplete preference

In this section, we propose an alternative approach to address the incomplete leader's preference by following the robust expected utility framework rooted in [4, 44]. To do so, we first explain the main results of [4, 44], and then adapt these ideas into the stochastic shortest path interdiction setting.

We model the worst-case utility by modifying problem (2.1) to take an inner infimum over the functional space of possible utility functions:

$$\max_{x \in X} \inf_{u \in U} \mathbb{E}\left[u\left(\min\left\{\sum_{a \in P}(\tilde{c}_a + \tilde{d}_a x_a) \mid P \in \mathcal{P}\right\}\right)\right], \tag{2.9}$$

It will be useful for us to focus on the inner optimization problem here independently, as there is considerable development required to explain how [4, 44] approach optimization over the set of functions U. First, we partition the set of utility functions by their values at points $z_j \in Z, j \in T$. For any given α this yields a set of all those utility functions consistent with these utilities for each possible outcome of our gambles, $U(\alpha) := \{u : u(z_j) = \alpha_j\}$; for each such α, we will need to further restrict our attention to those utility functions also in U_n, U_a, and U_2. Thus we reformulate the inner optimization in (2.9) as:

$$\min_{\alpha} \mathbb{E}\left[u\left(\min\left\{\sum_{a \in P}(\tilde{c}_a + \tilde{d}_a x_a) \mid P \in \mathcal{P}\right\}\right)\right] \tag{2.10a}$$

$$\text{s.t. } U(\alpha) \cap U_2 \neq \varnothing \tag{2.10b}$$

$$U(\alpha) \subseteq U_a \tag{2.10c}$$

$$U(\alpha) \subseteq U_n. \tag{2.10d}$$

There are infinitely many utility functions satisfying these constraints. In particular, for each consecutive pair of points z_j and z_{j+1} in our gambles' support, we may generate an infinite number of lines by varying α_j and α_{j+1}. However, by the definition of U_2 our utility functions must also be non-decreasing, so $\alpha_j \leq \alpha_{j+1}$ for each such pair, and we seek to find the worst-case utility consistent with these assumptions. Hence, if we call the optimum utility of point z as $u^*(z)$ (optimum regarding worst-case utility):

$$u^*(z) = \min_{v \geq 0, w} vz + w \tag{2.11a}$$

$$\text{s.t } v\bar{z}_j + w \geq \alpha_j, \ \forall j = 0, 1, \ldots, T. \tag{2.11b}$$

Model (2.11) provides a tractable linear programming (LP) representation for the concave piecewise linear program formed by connecting each (z_j, α_j) to its successor (z_{j+1}, α_{j+1}) by a line segment; at each z, this piecewise linear function gives the minimal function value consistent with the vector α and our structural assumptions on U. If we now substitute z by $h(x, \xi)$ for every scenario, and use constraints (2.3) to model (2.10b) to (2.10d), the following LP model gives the worst-case utility for a fixed $x \in X$:

$$\min_{\alpha,\beta,v,w} \sum_{k \in K} p_k (v_k \left(\min \left\{ \sum_{a \in P} (\tilde{c}_a + \tilde{d}_a x_a) \mid P \in \mathcal{P} \right\} \right) + w_k) \tag{2.12a}$$

$$\text{s.t. } \bar{z}_j v_k + w_k \geq \alpha_j, \ \forall k = 1, 2, \ldots, K, \ \forall j = 0, 1, \ldots, T \tag{2.12b}$$

$$\exists \beta \in \mathbb{R}^T_+ \text{ s.t. } (\alpha, \beta) \text{ satisfies (2.3)} \tag{2.12c}$$

$$v \in \mathbb{R}^K_+. \tag{2.12d}$$

Remark. If $\{\underline{\alpha}_j\}_{j=1}^T$ satisfies (2.3) (recall that $\underline{\alpha}_j$ is defined in Section 2.2.2), then the piecewise linear utility function defined by $\{\underline{\alpha}_j\}_{j=1}^T$ is identical to the worst-case utility in (2.9).

This model is solvable only for a given x. If we consider a fixed x, then the distribution $h(x, \xi)$ is determined, and so we can find the corresponding expected utility. To find the leader's optimal $x \in X$, we need to combine our outer decision problem with the framework we have just developed for minimizing over the set U. Because the decision problem is a maximization problem, we first need to change the optimization sense of problem (2.12). To this end, we can fix the value of x, take the dual of the problem (2.12), and then release x. We introduce the following sets of dual variables: $\mu_{k,j},\ k = 1, 2, \ldots, K, j = 0, 1, \ldots, T$ for constraint (2.12b), $\nu_\ell,\ \forall \ell = 1, 2, \ldots, L$ for constraint (2.3b), $\lambda_j^{(1)},\ \forall j = 0, 1, \ldots, T-1$ for constraint (2.3c), and $\lambda_j^{(2)},\ \forall j = 0, 1, \ldots, T-1$ for constraint (2.3d). The combined model can now be stated as:

$$\max_{x\in X,\mu,\nu,\lambda^{(1)},\lambda^{(2)}} \nu_0 \tag{2.13a}$$

$$\text{s.t.} \sum_{j\in T} \frac{\bar{z}_j}{p_k}\mu_{k,j} \le \min\left\{\sum_{a\in P}(\tilde{c}_a + \tilde{d}_a x_a) \mid P \in \mathcal{P}\right\},\ \forall k \in 1,2,\ldots,K \tag{2.13b}$$

$$\sum_{k\in K}\mu_{k,j} - \left[\mathbb{P}(W_0 = \bar{z}_j) - \mathbb{P}(Y_0 = \bar{z}_j)\right]\nu_0 - \sum_{\ell\in L}\left[\mathbb{P}(W_\ell = \bar{z}_j) - \mathbb{P}(Y_\ell = \bar{z}_j)\right]\nu_\ell$$
$$+ (\lambda_j^{(1)} - \lambda_{j-1}^{(1)}) - (\lambda_j^{(2)} - \lambda_{j-1}^{(2)}) \ge 0,\ \forall j = 0,1,\ldots,T \tag{2.13c}$$

$$\lambda_j^{(2)}(\bar{z}_{j+1} - \bar{z}_j) - \lambda_{j-1}^{(1)}(\bar{z}_j - \bar{z}_{j-1}) \le 0,\ \forall j = 0,1,\ldots,T-1 \tag{2.13d}$$

$$\sum_{j\in T}\mu_{k,j} = p_k,\ \forall k = 1,2,\ldots,K \tag{2.13e}$$

$$\mu \in \mathbb{R}_+^K \times \mathbb{R}_+^{T+1}, \nu \in \mathbb{R}_+^{L+1}, \lambda^{(1)} \in \mathbb{R}_+^T, \lambda^{(2)} \in \mathbb{R}_+^T. \tag{2.13f}$$

If $h(\cdot,\xi)$ is a concave function of x, the feasible set defined by constraint (2.13b) is convex. When all constraints are linear in the decision variables, we can use an LP or MIP solver (depending on whether or not the decision vector x involves any integer variables) to solve this model and find an optimal $x \in X$ and the associated worst-case utility.

2.3.1 A robust utility model for stochastic shortest path interdiction with incomplete defender's preference

In this section, we develop a model in which the leader's risk preference is ambiguous and the follower makes his decision after observing the realization of the random variables (wait-and-see). This model is particularly useful for modeling situations in which the leader represents a group of decision makers with different risk preferences, for modeling leader for whom standard decision-analytic elicitation techniques require preference comparisons that the leader finds difficult to concretely imagine, or simply for evaluating the sensitivity of standard models to the completeness assumption. In this case, the leader confronts a risky decision but the follower faces only a deterministic shortest path problem. Because the leader's preferences are incomplete, we use the previously developed model (2.13) to represent optimization under incomplete preferences. The objective is to find an interdiction plan by considering the leader's worst-case utility. The value of each interdiction plan, that is, the value of the shortest path that the follower chooses given that plan, depends on uncertain parameters (cost and interdiction effect of each arc). From the perspective of the leader, each interdiction plan can be seen as a gamble. Therefore, in this model, the pairwise comparisons are for different interdiction plans.

In constraint (2.13b), $h(x, \xi)$ is the value of the shortest path that follower chooses contingent on the random scenario value ξ given an interdiction plan x. Let set $\mathcal{P}$ be the set of all s-t paths in the network, so that $h(x, \xi) = h(x, \tilde{c}, \tilde{d}) = \min \left\{ \sum_{a \in P} (\tilde{c}_a + \tilde{d}_a x_a) \mid P \in \mathcal{P} \right\}$. Constraint (2.13b) can then be reformulated as: $\sum_{j \in T} \bar{z}_j \mu_{k,j} \leq p_k \sum_{a \in P} (\tilde{c}_a^k + \tilde{d}_a^k x_a)$, $\forall P \in \mathcal{P}$, $\forall k = 1, 2, \ldots, K$, which gives the following MIP formulation for the stochastic network interdiction problem with a wait-and-see follower and incomplete knowledge about the leader's preferences:

$$\max_{x \in X, \mu, \nu, \lambda^{(1)}, \lambda^{(2)}} \nu_0 \tag{2.14a}$$

$$\text{s.t.} \sum_{j \in T} \bar{z}_j \mu_{k,j} \leq p_k \sum_{a \in P} (\tilde{c}_a^k + \tilde{d}_a^k x_a), \ \forall P \in \mathcal{P}, \ \forall k = 1, 2, \ldots, K \tag{2.14b}$$

$$(2.13c) - -(2.13f). \tag{2.14c}$$

Similar to solving model (2.6), we can add constraints (2.14b) in a delayed constraint generation manner and solve (2.14) using a branch-and-cut algorithm. For the sake of computational comparison (presented in Appendix) we also consider a common practice in robust optimization to handle constraint 2.14b. Note that constraint 2.14b can be written as (for $x = \hat{x}$):

$$\frac{1}{p_k} \sum_{j \in T} \bar{z}_j \mu_{k,j} \leq \min \left\{ \sum_{a \in A} (\tilde{c}_a^k + \tilde{d}_a^k \hat{x_a}) y_a, \ y \in Y \right\}, \ \forall k = 1, 2, \ldots, K. \tag{2.15}$$

Let $\pi_i^k, \ i \in V$ be the dual variable corresponding to each constraint in Y, then we can replace constraint (2.14b) with the following two constraints:

$$\frac{1}{p_k} \sum_{j \in T} \bar{z}_j \mu_{k,j} \leq \pi_t^k, \ \forall k = 1, 2, \ldots, K \tag{2.16}$$

$$\pi_n^k - \pi_m^k \leq \tilde{c}_a^k + \tilde{d}_a^k x_a, \ \forall a = (m, n) \in A, \ \forall k = 1, 2, \ldots, K. \tag{2.17}$$

Same approach could be applied to the model (2.6), however because of the similarity in the structure and acceptable computational performance of model (2.6), we only compare this formulation for robust expected utility model.

2.3.2 A robust certainty equivalent model for stochastic shortest path interdiction with incomplete defender's preference

The performance of the robust utility model (2.14) depends on the amount of information that can be exploited from historical data on pairwise comparisons made in the past. When we do not have enough information, the corresponding worst-case utility may not well represent the decision maker's actual risk preference. In the extreme case, for example, when only a very small number of comparisons are incorporated, then the worst-case utility function may just be a linear function, making it a risk neutral model. Indeed, it is shown in [4] that model (2.14) is inclined towards a risk neutral model when a large amount of ambiguity exists. In this case, the robust certainty equivalent model, also proposed and studied in [4], is shown to be more desired.

Given a utility function $u(\cdot)$, the certainty equivalent of a gamble $h(x, \xi)$ is the largest sure amount such that the decision maker is indifferent between this sure amount and the gamble. In other words, the certainty equivalent corresponding to the optimal interdiction plan $\hat{x}$ is given by $\sup\{C \mid \mathbb{E}[u(h(\hat{x}, \xi))] \geq u(C)\}$. When the information on the preference is incomplete, i.e., the utility function is only known to belong to an ambiguity set U, the robust certainty equivalent can be defined in a robust optimization setting as:

$$\sup\{C \mid \mathbb{E}[u(h(\hat{x}, \xi))] \geq u(C), \ \forall u \in U\}. \tag{2.18}$$

Recall that U is defined in (2.2), to check whether a constant C is qualified as a candidate for the robust certainty equivalent, one could instead solve:

$$\max_{x\in X}\min_{u\in U}\ \{\mathbb{E}[u(h(x,\xi))]-u(C)\}\,, \tag{2.19}$$

and check if the optimal objective value is nonnegative. It is clear that model (2.19) is almost identical to model (2.9), one could apply the same approach used for solving (2.13) to solve model (2.19) with a fixed C. Since the certainty equivalent is a scalar, one could solve (2.18) by finding the largest qualified C using a bisection procedure. We note that C can only take values between Y_0 and W_0, the smallest and largest outcome from any gamble.

Here, we summarize the pros and cons of each model presented in this section and try to provide a guideline on how to choose the right model given a specific situation. The proposed models could be separated to two different groups: one which separates the utility estimation and optimization phase and another one which integrates the utility estimation and optimization in a robust manner in order to optimally deploy the available information. As it will be discussed in section 2.4, the first group of methods are very straightforward to implement and computationally easy. However, if we have inadequate amount of information on decision maker's risk preferences (a considerable amount of ambiguity is involved), then the first group of methods will perform very poorly in capturing the risk aversion. This is the situation where we prefer robust certainty equivalent model because of its strength in capturing risk aversion under high level of ambiguity. When we have enough information regarding the shape of the utility function, then we choose robust expected utility model if computational time is not a limiting issue for us, otherwise the first group of methods are preferred.

2.4 Computational Experiments

In this section we evaluate the models described in the previous section through a set of test instances that are randomly generated. We first provide details on how our random instances are generated. Then we present and discuss the computational results for various models that we consider.

2.4.1 Instance generation

Recall that all the proposed approaches require a set of comparisons between pairs of gambles as inout. We assume that the results of these comparisons should be consistent with some underlying (true) utility function. Our instance generation procedure is primarily directed towards constructing a set of gamble pairs with this property. For simplicity, all the network instances that we generate in our experiments are grid networks of various sizes. The instance generation procedure consists of the following steps:

1. Select the size of the grid network and assign the cost and interdiction effect for each arc of the network in each scenario. The costs and interdiction effects are randomly generated according to a uniform distribution $U(15, 30)$.

2. Randomly generate a set of feasible interdiction plans using a given budget ratio ρ, so that $r_0 = |A| \times \rho$. We assume that the interdiction costs for all arcs are identical, i.e., $r_a = 1,\ \forall a \in A$ (recall that r_0, r are defined in (1.18)). We consider three different budget ratios $\rho \in \{0.5, 0.6, 0.7\}$. For each ratio, we

randomly generate 20 different interdiction plans.

3. Calculate the outcome corresponding to each decision (i.e., the shortest path value for each interdiction plan) for each scenario and refer to the values of these outcomes as “attributes”.

4. Assign a utility value for each attribute using an underlying non-decreasing concave utility function, which can be interpreted as the “true utility”. In our experiments we use $u(z) = a(tan^{-1}(cz))^d + b$ with $c = 0.009$ and $d = 0.01$, and parameters a and b are determined based on two constraints: $u(Y_0) = 0$ and $u(W_0) = 1$.

5. Calculate the expected utility values of the generated interdiction plans:

   5.1. For each budget ratio, calculate the expected utility of each plan.

   5.2. Calculate the difference of expected utilities between every two plans and sort these differences in an ascending order.

6. Given a fixed number of pairwise comparisons to be included in any model, we choose pairs of gamble comparisons corresponding to the largest differences in terms of the expected utility values evenly from each budget ratio.

2.4.2 Computational settings

We implement all models considered in Python 3.4 and we use Gurobi 6.5.1 as the LP and MIP solver to solve them. For each network instance and scenario size, we generate five replications and report the average results. We leave all Gurobi parameters at their default settings except for the time limit, number of threads, and MIPGap. We impose a time limit of 3600 seconds on each experiment, set the number of threads to one, and set the MIPGap (ending optimality gap) parameter to 10^{-3}. Valid inequalities in models (2.6) and (2.14) are added as lazy constraints in Gurobi. We use package *Networkx* [37] for graph-related algorithms (e.g., for solving the shortest path problems) and Numpy [97] for array and matrix computations. All experiments are conducted on a Linux workstation with four 3.00 GHz processors and 6 Gb memory.

2.4.3 Comparison between the robust expected utility model and the robust certainty equivalent model

In this section we present and discuss the performances of the following models:

1. The risk neutral model (RNM), where the interdictor simply maximize the expected value of the stochastic shortest interdicted path, i.e.,

$$\max_{x \in X} \ \mathbb{E}[h(x, \xi)].$$

2. The robust expected utility model (REU) presented in (2.14) in Section 2.3.1.

3. The robust certainty equivalent model (RCE) presented in (2.18) in Section 2.3.2.

4. Fitting the utility function using piecewise linear concave functions with up to $T - 1$ pieces (recall that T is the cardinality of the support of all random gambles involved in the set of pairwise gamble comparisons), which we call

the "full piecewise linear fitting" (FPM), presented in model (2.4). Given a solution of (2.4), we use parameter $\delta = 10^{-4}$ so that consecutive pieces are merged together if the differences between their slopes are less than δ.

5. Fitting the utility function using piecewise linear concave functions with up to $N' = 5$ pieces, which we call the "partial piecewise linear fitting" (PPM), presented in model (2.7).

We first evaluate whether or not the robust models (REU and RCE) could successfully capture the risk aversion of the decision maker, if the amount of preference information is little, which happens when the number of pairwise comparisons recorded in the historical data is small. We use a well-known risk measure, the Conditional Value at Risk (CVaR), to quantify the level of risk aversion of the corresponding solution. Given a parameter α, CVaR is defined as:

$$CVaR_{\alpha} = \mathbb{E}[X \mid X \leq VaR_{\alpha}(X)], \tag{2.20a}$$

where,

$$VaR_{\alpha} = \max\{c : \mathbb{P}(X \geq c) \geq 1 - \alpha\}, \tag{2.21a}$$

which represents the $\alpha\%$ worst outcome, i.e., the realization of the shortest interdicted path value is larger than this value with probability at least $1 - \alpha$. In the context of shortest path interdiction, X in 2.20a and 2.21 is the length of the shortest path that the follower choose after observing the leader's plan.

Table 1 reports the corresponding CVaR values for optimal plans obtained from different models with 100 scenarios, a fixed budget ratio (0.4) and two different levels of α (0.01 and 0.1). For the robust expected utility model, we consider two different situations: one with 20 pairwise gamble comparisons and the other with only three pairwise gamble comparisons. To some extent, the number of pairwise comparison included in the model represents the amount of incomplete preference information. We should mention that to emphasize the unfavorable effects caused by the lack of information, the three pairwise comparisons are chosen as the ones with the smallest differences in their expected utility values (please refer to Step 6 of the instance generation procedure in Section 2.4.1).

Table 1. Comparison of the solution behavior for the risk neutrial model (RNM), robust expected utility model (REU) with 20 and 3 pairwise comparisons, and robust certainty equivalent model (RCE) with 3 pairwise gamble comparisons using CVaR.

	$\alpha = 0.01$					$\alpha = 0.1$				
Instances	RNM	REU (L = 3)	REU (L = 20)	RCE (L = 3)	RCE (L = 20)	RNM	REU (L = 3)	REU (L = 20)	RCE (L = 3)	RCE (L = 20)
3×3	93.30	93.30	106.4	109.20	106.4	114.47	114.47	122.97	121.56	122.97
4×4	158.90	158.90	174.40	181.13	176.20	183.71	183.71	192.53	192.93	197.05
5×5	234.10	234.10	248.40	264.13	251.90	260.13	260.13	271.18	272.65	275.38
6×6	310.63	310.63	321.63	342.33	320.63	341.28	341.28	352.93	350.39	351.00
7×7	381.60	380.10	397.30	426.55	396.43	416.82	420.40	427.96	432.57	430.49

Table 2. Comparison of solution time for REU and RCE with $L = 20$

Instances	REU (L = 20)	RCE (L = 20)
3×3	1.30	58.46
4×4	3.99	196.66
5×5	10.75	410.08
6×6	223.05	16573.62
7×7	3604.83	20827.93

As we can observe in Table 1, solutions from the risk neutral model (RNM) and REU ($L = 3$) behave almost the same in terms of the CVaR value. On the other hand, we see that the robust certainty equivalent model can capture the risk aversion quite well, which yields similar CVaR values as REU with a larger number of pairwise gamble comparisons. For REU and RCE when $L = 20$, CVaR values are very similar. However, as we mentioned earlier, the computational time of RCE is intense, as reported in Table 2. Hence, when we do not have a large amount of ambiguity, REU is a preferred approach. As an illustrative example, we focus on the tail distribution (with $\alpha = 0.1$) of the stochastic shortest interdicted path values for optimal solutions yielded by various models for a particular instance in Figure 1. Note that we expect a risk-averse decision to stay away from extremely bad outcomes, i.e., the ones that correspond to very small shortest path values, which should results in a larger CVaR value. This is indeed what we can observe in Figure 1, since the realizations in the left tail of REU ($L = 20$) and RCE are clearly larger than RNM and REU ($L = 3$). This confirms the observation made by [4] that when there is not enough information that can be exploited from the set of pairwise gamble comparisons, e.g., when too few comparisons are available, the worst-case utility function given by the robust expected utility model behaves like a linear function, which corresponds to a risk neutral model.

2.4.4 Comparison between the robust expected utility model and the best fitted piecewise linear concave utility model

We next compare the performances of the robust expected utility model and the best fitted piecewise linear concave utility model in terms of how the resulting utility functions resemble the true underlying utility function. If we base our comparison

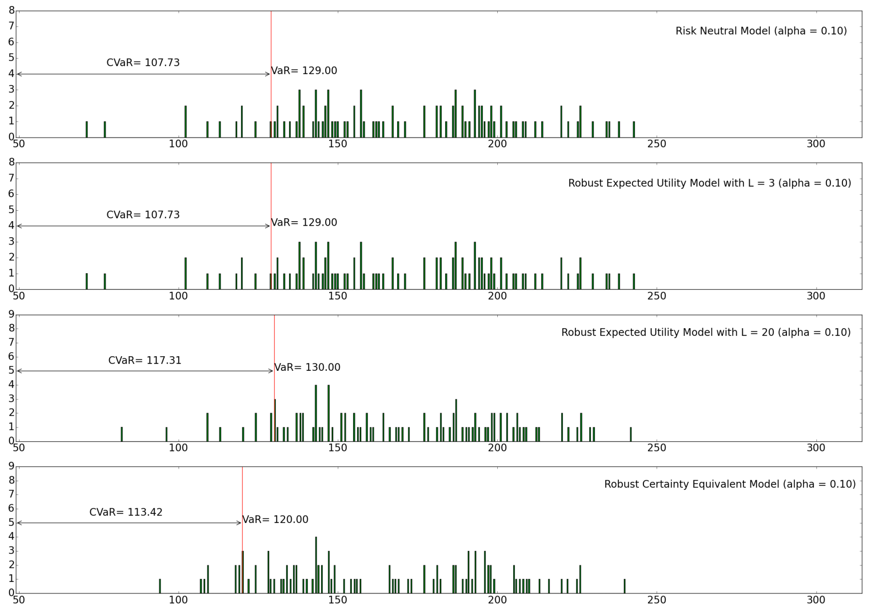

Fig. 1. Illustration on the tail distributions of the optimal solutions obtained by model RNM, REU with $L = 3$ and $L = 20$, and RCE with $L = 3$ for a 3×3 grid network instance with 100 scenarios and budget ratio 0.4.

using a measure of overall distance (e.g., the average squared error) from the true utility function, we can observe from Table 3 that the fitted piecewise linear concave utility functions perform better than the worst-case utility function from the robust expected utility model. We also see that the performance of FPM and PPM are very similar, meaning that a small number of pieces are sufficient to yield a good fit of

Table 3. The average squared error between the resulting utility function from each model (REU, FPM and PPM) and the true utility function. All instances considered in this table have 100 scenarios and budget ratio 0.4.

Instances	Average Squared Error		
	REU (L=20)	FPM	PPM
3×3	0.0323	0.0083	0.0088
4×4	0.0286	0.0092	0.0099
5×5	0.0245	0.0074	0.0083
6×6	0.0341	0.0049	0.0077
7×7	0.0409	0.0223	0.0288

the utility function. However, there is a key difference between these two types of approaches: For FPM and PPM, utility estimation is separated from optimization, so that the optimal solution does not have any effect on the fitted utility function. On the contrary, in the robust expected utility model, the resulting worst case utility function depends on the distribution of the stochastic shortest interdicted path value corresponding to the optimal solution (note that the shortest path value is indeed a random variable). In Figure 2, the barchart between the two vertical dashed lines gives the distribution of the random outcome of the shortest interdicted path value with respect to the optimal interdiction plan (the length of the shortest path for each scenario). Focusing on the realization of this random variable, we see that the worst case utility in this region is almost parallel to the true utility function, which captures the true risk aversion locally in the area of interest. On the other hand, we see that the fitted function from the FPM model does not resemble the curvature of the true utility function in that local area as accurate as the worst-case utility function, although over the entire region, this function fits the true utility function better.

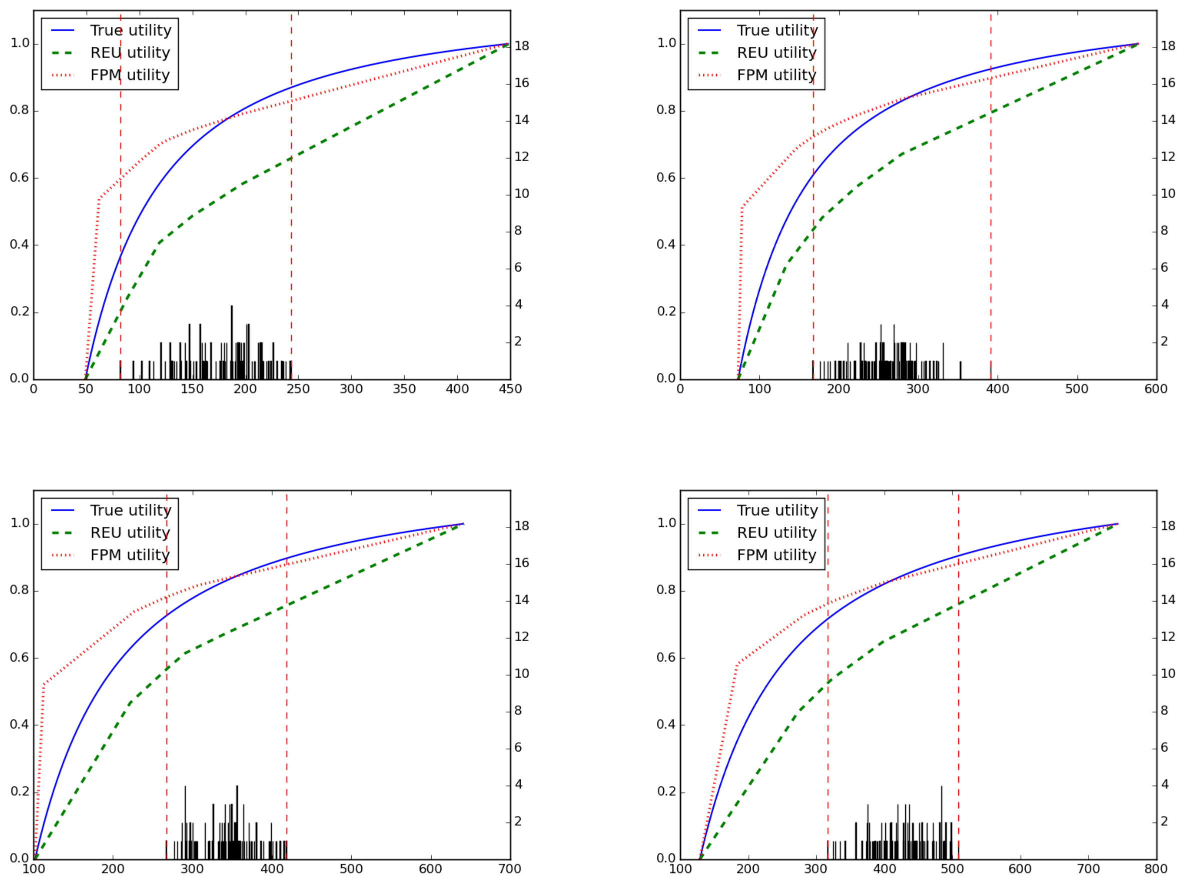

Fig. 2. An illustration of the true utility function, the worst-case utility function resulting from the REU model, and the fitted utility function from the FPM model for four grid network instances: 3×3 with interdiction budget $r_0 = 6$ (top left), 4×4 with interdiction budget $r_0 = 10$ (top right), 5×5 with interdiction budget $r_0 = 16$ (bottom left) and 6×6 with $r_0 = 34$ (bottom right) with 100 scenarios and 20 pairwise comparisons.

2.4.5 Computational performances

In this section we present the computational performances of the proposed approaches. In all tables presented below, we use the following labels:

- **Instance**: Size of the grid network in the form of $n \times n$, e.g., a 3×3 grid network has nine nodes and 12 arcs.
- K: Number of scenarios. We test for three different scenario sizes: $K = 100$, 500, and 1000.
- **Time**: If the instance is solved to optimality within the time limit, we report the computational time (in seconds), otherwise, we report the average ending optimality gap (A) and the number of instances that were solved to optimality within the time limit (among the five replications) in the form of $A(B)$.
- **# Nodes**: Total number of nodes explored in the branch-and-bound tree.
- **RootGap**: The relative optimality gap between the best objective value obtained (U^*) and the relaxation bound obtained after exploring the root node of the branch-and-bound tree (U_R), i.e., $\frac{U^* - U_R}{U_R}$.

Table 4. Average computational time (including the time spent on fitting the utility function for approach FPM and PPM), number of nodes explored in the branch-and-bound tree, and the root optimality gap for three approaches REU, FPM and PPM for grid network instances with various network sizes, interdiction budgets, and scenario sizes.

			Time			# Nodes			RootGap (%)		
K	Instance	Budget	REU	FPM	PPM	REU	FPM	PPM	REU	FPM	PPM
100	3×3	6	2.77	10.85	7.44	38	30	23	2.42	0.94	0.64
	4×4	12	2.14	21.29	16.17	35	18	15	0.99	0.33	0.18
	5×5	20	21.79	69.97	27.17	1920	1671	1690	1.73	0.68	0.47
	6×6	30	94.95	156.11	33.18	7385	1605	1988	1.60	0.71	0.77
	7×7	42	1.16% (0)	2241.75	114.96	1017	33183	3313	1.53	1.10	1.07
500	3×3	6	18.40	44.15	22.28	36	30	34	3.72	1.10	1.34
	4×4	12	23.72	54.59	32.16	41	28	26.2	0.75	0.33	0.31
	5×5	20	158.86	548.68	71.25	3225	3049	3119	1.00	0.59	0.60
	6×6	30	545.03	674.66	89.77	9566	2074	2296	1.00	0.60	00.65
	7×7	42	0.54% (0)	0.24% (0)	2495.81	49542	71644	110563	2.21	1.14	1.37
1000	3×3	6	68.95	52.4	21.75	41	32	33	4.08	0.91	0.91
	4×4	12	57.88	85.16	39.68	48	29	30	1.23	0.40	0.43
	5×5	20	920.89	1257.17	133.16	3228	3331	3426	2.00	0.64	0.58
	6×6	30	1178.01	1209.10	153	4811	2241	1892	1.36	0.64	0.61
	7×7	42	0.77% (0)	0.28% (0)	0.19% (0)	16978	6388	38646	2.87	1.07	1.33

Table 5. Average computational time for two approaches, REU and PPM, on sets of pairwise gamble comparisons of different sizes (20, 50, and 100, respectively).

			REU			PPM		
K	Instance	Budget	20	50	100	20	50	100
100	3×3	6	2.77	1.77	2.45	7.44	12.10	15.30
	4×4	12	2.14	3.85	4.04	16.17	24.94	25.85
	5×5	20	21.79	46.69	48.76	27.17	33.74	45.78
500	3×3	6	18.40	19.04	39.67	22.28	21.02	22.53
	4×4	12	23.72	36.90	45.03	32.16	33.69	36.96
	5×5	20	158.86	226.61	271.28	71.25	63.65	68.28
1000	3×3	6	68.95	72.97	92.12	21.75	31.30	26.37
	4×4	12	57.88	95.28	657.41	39.68	51.14	46.87
	5×5	20	920.89	1332.38	1419.86	133.16	140.29	145.80

In Table 4, we report the average computational time (including the time spent on fitting the utility function for approaches FPM and PPM), number of nodes explored in the branch-and-bound tree, and the root optimality gap for three approaches REU, FPM and PPM for grid network instances with various network sizes, interdiction budgets, and scenario sizes. From Table 4, we observe that FPM takes more time than both PPM and REU in almost all the test instances. REU yields a better performance in terms of computational time for small network instances with a small number of scenarios, however, PPM gives the best performance in general, particularly for larger network instances. We also see that a larger scenario set has more impact on REU than PPM. Finally, we see that all three approaches yield a relatively tight root relaxation bound.

In Table 5, we report the average computational time for REU and PPM for sets of pairwise gamble comparisons of various sizes. From 5, we observe that the computational performance of PPM is almost independent of the number of pairwise comparisons that are incorporated to the model. In contrast, REU takes significantly more time when a larger number of gamble comparisons are incorporated.

In the Appendix, we report the full computational results for the three approaches, REU, FPM and PPM with various interdiction budget ratios in Table 19 and Table 21. We also report in Table 22 a detailed computational time profile for approach PPM on three different phases (calculating the upper and lower bounds $\bar{u}_j, \underline{u}_j, j \in T$, fitting the utility function, and optimization using the utility function).

2.4.6 Summary of computational experiments

According to our computational experiments using stochastic shortest path interdiction problems with incomplete defender's preference, we conclude that there are two important factors to consider when choosing the appropriate approach: the amount of preference information (encoded by the set of pairwise gamble comparisons) and the computational budget. When the number of pairwise gamble comparisons is small, we see that the robust certainty equivalent model (RCE) captures the risk aversion of the decision maker better than the robust expected utility model (REU). When the number of pairwise gamble comparisons is sufficient, the robust expected utility model performs better than the approach that separates utility fitting and optimization (FPM and PPM), as it better captures the local risk aversion behavior of the true utility around the outcomes corresponding to the underlying optimal interdiction plan. However, it comes at the cost of increased computational time. Therefore, REU is a preferred approach when the computational budget is not tight. Otherwise, one would choose to fit a piecewise linear concave utility function with a fixed (small) number of pieces first, and then do optimization using the best fitted function.

2.5 Extensions

In this section we discuss a possible extension of the proposed models, to the case when the follower also makes his decision before the realization of uncertain parameters, i.e., in a “here-and-now” setting. Network interdiction problem with a “here-and-now” follower has been studied by [52], where the authors show that different follower's risk preferences may have a significant impact on the leader's optimal decision. We assume that the follower knows his own utility function; the leader is ambiguous about the follower's risk preference, but has access to some data (in the form of pairwise comparisons of gambles made by the follower in the past) that can help her gain possibly incomplete information about it.

One way to incorporate this incomplete information about the follower's preference is to consider a set U of utility functions of some parametric form that are consistent with the pairwise gamble comparisons. We consider formulation (2.22) to model this situation. The lowest level problem is the expected utility maximization problem for the follower, given a fixed utility function. For a given interdiction decision x, since the leader is unsure about the follower's utility function, she considers the worst case (for her) among all possible utility functions that belong to the ambiguity set constructed using her incomplete information about the follower's preference. Lastly, the outermost problem optimizes the leader's interdiction decision.

$$\min_{x\in X}\ \max_{u\in U}\ \max_{y\in Y}\ \mathbb{E}\left[u\left(\sum_{a\in A}(\tilde{c}_a+\tilde{d}_a x_a)y_a\right)\right], \tag{2.22}$$

where $Y=\left\{y\in\mathbb{R}_+^{|A|}\mid(1.18\text{b}),(1.18\text{c})\right\}$.

Model (2.22) is challenging to solve. Instead of considering the worst case utility function among U, one may settle with a sample approximation by sampling from this set of utility functions (which belong to a finite dimensional space, since a particular parametric form (such as piecewise linear functions) is assumed), and consider the worst case utility function among all samples. Given a set of utility functions $\{u_j\}_{j\in J}$ sampled from the entire set U of utility functions that are compatible with the leader's incomplete knowledge about the follower, this problem can be formulated as follows:

$$\min_{x\in X}\ \max_{j\in J}\ \max_{y\in Y}\ \mathbb{E}\left[u_j\left(\sum_{a\in A}(\tilde{c}_a+\tilde{d}_a x_a)y_a\right)\right]. \tag{2.23}$$

Suppose we consider the family of piecewise linear functions with a fixed number of pieces N, similar to model (2.5), given a fixed interdiction plan x, the worst case expected utility can be calculated as:

$$\max_{j\in J} \sum_{k\in K} \frac{1}{K}\, \theta_k^j \tag{2.24a}$$

$$\text{s.t. } \theta_k^j \le v_i^j \left[\sum_{a\in A}(c_a^k + d_a^k x_a) y_a\right] + w_i^j, \ \ \forall j \in J, \forall k \in K,\ \forall i \in \{1,\dots,N\} \tag{2.24b}$$

$$\sum_{a\in FS(m)} y_a - \sum_{a\in RS(m)} y_a = \begin{cases} 1 & \text{if } m = s \\ -1 & \text{if } m = t \\ 0 & \text{o.w.} \end{cases} \tag{2.24c}$$

$$v_i^j \ge 0,\ \forall i \in \{1,\dots,N\},\ \forall j \in J,\ y_a \in \{0,1\},\ \forall a \in A. \tag{2.24d}$$

Therefore, the defender's problem (2.23) can be formulated as:

$$\min_{x\in X} \max_{j\in J} \left\{\sum_{k\in K} \frac{1}{K}\, \theta_k^j \mid \text{(2.24b) to (2.24d)}\right\}. \tag{2.25}$$

Model (2.25) can be seen as a bilevel min-max optimization problem in which the lower level problem depends on the higher level solution. Here we describe a possible solution method with which we can solve some small instances. The solution method is based on the integer L-shaped method developed to solve two-stage stochastic integer programs [49].

Given a fixed integer feasible solution $\hat{x}$ in the upper level, we can solve $|J|$ separate subproblems (2.24) in the lower level for each utility function $j \in J$ and obtain the corresponding expected utility value $Q^j(\hat{x})$. Assume that η_0 is an overall lower bound for the optimal objective value. Then the following integer L-shaped cut can be generated and added to the upper level problem:

$$\eta \geq (Q^j(\hat{x}) - \eta_0) \left(\sum_{i \in S(\hat{x})} x_i - \sum_{i \notin S(\hat{x})} x_i - |S(\hat{x})| + 1 \right) + \eta_0,$$

where $S(\hat{x}) = \{a \in A \mid \hat{x}_a = 1\}$.

We should mention that, even though it is easy to implement integer L-shaped method, integer L-shaped cuts are known to be very weak, and should be embedded with some additional structure-exploiting cuts that are problem specific. In order to be able to solve medium- and large-scale instances, more advanced techniques are needed for solving this problem.

2.6 Concluding Remarks

We studied the stochastic shortest path interdiction problem with a wait-and-see follower when the leader has incomplete knowledge about her risk preference. We took two different approaches to model this problem. In the first approach, we fit a piecewise linear concave utility function according to the incomplete knowledge about the leader's true utility function through a set of pairwise gamble comparisons. Then we used this best-fitted utility function to formulate an expected utility maximization problem. In the second approach, we integrated the incomplete knowledge into the expected utility maximization problem by adapting the robust expected utility framework proposed by [4, 44], which combines utility estimation and optimization into a single model. We also developed a branch-and-cut algorithm to solve both of the proposed models. We conducted extensive computational experiments on both approaches, and we found that the robust expected utility model performs better in terms of resembling the true utility function locally at the support of the random outcomes associated with the corresponding optimal solution. However, this model could be very time-consuming to solve in large-scale instances. On the other hand, solving the expected utility maximization problem with a best-fitted piecewise linear concave utility function with a small number of pieces is more computationally convenient. As a possible future work, we could extend the proposed models to the situation where the follower also needs to make a here-and-now decision. Therefore, the leader needs to make an optimal interdiction plan with an incomplete knowledge about the follower's preference.

CHAPTER 3

PARTITION-BASED DECOMPOSITION ALGORITHMS FOR TWO-STAGE STOCHASTIC INTEGER PROGRAMS WITH CONTINUOUS RECOURSE

3.1 Introduction

In this chapter, we propose adaptive partition-based decomposition algorithms for two-stage SIP with integer variables only in the first stage. The core of the proposed methods is based on the theoretical and computational results established for two-stage stochastic linear programs reported in [86, 92]. We show that the success of partition-based approaches for two-stage stochastic linear programs with fixed recourse can be extended to the two-stage SIP with continuous recourse to some extent. In particular, the overall cutting plane approach and level decomposition using inexact cuts induced by scenario partitions can be successfully integrated into the branch-and-cut algorithm used to solve the integer programming master problem. However, in contrast to the linear case, where it has been proved that there exists a sufficient partition whose size is independent of the number of scenarios used in the model, we do not have such theoretical guarantee when the first-stage problem involves integer variables. (The proof of [92][Theorem 1] highly depends on the polyhedral property of the extensive formulation (1.6).) Therefore, using the same refinement rule suggested by [86, 92] based on the optimal dual multiplier for each scenario may result in a very large partition in a few iterations, making the entire algorithm less interesting. To address this issue, we propose a heuristic strategy for doing partition refinement that maintains a small partition in the first few iterations and then gradually increases the size of the partition, with the guidance of the optimal dual multipliers.

The contributions of this chapter can be summarized as follows. First, this is the first time to the best of our knowledge that the computational performances of the adaptive partition-based decomposition algorithms for two-stage SIP are reported. To make this happen, we integrate the partition-based overall cutting plane algorithm and level decomposition with the branch-and-cut algorithm for solving the integer programming master problem. In addition, we provide empirical guidance on how to perform partition refinement when it is not necessarily guaranteed that a small sufficient partition exists.

The rest of the chapter is organized as follows. In Section 3.2, we describe how the partition-based overall cutting plane algorithm is integrated with the branch-and-cut algorithm, and how the partition-based level decomposition is customized for solving two-stage SIP with continuous recourse. We present our computational experiment results in Section 3.3, and give concluding remarks in Section 3.4.

3.2 Partition-Based Decomposition Algorithms for Two-Stage Stochastic Integer Programs with Continuous Recourse

In this section we propose two types of adaptive partition-based decomposition algorithms for solving two-stage SIP with continuous recourse. The proposed algorithms integrate the partition-based framework established for two-stage stochastic linear programs [86, 92] with the branch-and-cut algorithm for solving the integer programming first-stage master problem.

3.2.1 A partition-based branch-and-cut algorithm

The partition-based branch-and-cut algorithm embeds the partition-based overall cutting plane algorithm for two-stage stochastic linear programs with fixed recourse [92] into the branch-and-cut framework. The key idea is to generate coarse cuts corresponding to a scenario partition first, and resort to the exact Benders (fine) cuts only when it is necessary. When the partition size is small compared to the total number of scenarios, i.e., $|\mathcal{N}| \ll |N|$, this potentially could accelerate the cutting plane generation procedure by starting with coarse cuts that can be generated efficiently, and only using the more expensive fine cuts when they are necessary to close the optimality gap. When the fine cuts are generated, since the second-stage subproblem is solved for each scenario $k \in N$, one could use the corresponding optimal dual multipliers to refine the current partition. Because of these partition refinements, the coarse cuts are asymptotically exact and their strengths are adaptive to the solution procedure. Specifically, given a partition $\mathcal{N}$ and a first-stage solution $\hat{x}$, the partition-based subproblem for each component $\mathcal{P} \in \mathcal{N}$ is defined as:

$$f_{\mathcal{P}}(\hat{x}) = \min_{y^{\mathcal{P}} \in \mathbb{R}^{n_2}_{+}} \left\{ d^{\top} y^{\mathcal{P}} \;\mid\; W y^{\mathcal{P}} \geq h^{\mathcal{P}} - T^{\mathcal{P}} x \right\}, \tag{3.1}$$

where $T^{\mathcal{P}} := \sum_{k \in \mathcal{P}} T^k$ and $h^{\mathcal{P}} := \sum_{k \in \mathcal{P}} h^k$. A partition-based coarse cut can then be generated by using the optimal dual multipliers $\hat{\lambda}^{\mathcal{P}}$ corresponding to the partition-based subproblem for each component $\mathcal{P} \in \mathcal{N}$:

$$\theta \geq \sum_{\mathcal{P} \in \mathcal{N}} (\hat{\lambda}^{\mathcal{P}})^{\top} (\bar{h}^{\mathcal{P}} - \bar{T}^{\mathcal{P}} \hat{x}). \tag{3.2}$$

It is clear that cuts (3.2) are valid optimality cuts for (1.6), since they can be seen as Benders (fine) cuts for (1.12), which is a relaxation of (1.6). Compared with the standard Benders decomposition, where a fine cut that involves solving all N scenario-based subproblems needs to be generated throughout the algorithm, partition-based coarse cuts provide a much cheaper way of generating valid cuts, at least during the early stages of the algorithm. This could be very useful especially in the first few iterations of the algorithm, since the initial solutions usually are far from the optimal solution, and it does not worth the effort of generating exact cuts based on these solutions. The whole framework can be seen as a special case of the so-called "on-demand accuracy" [99, 25, 24], where more accurate cuts are generated when they are mostly needed.

We implement the partition-based branch-and-cut algorithm using the aggregated form of Benders cuts (1.10) for the sake of convenience. Indeed, if a disaggregated form is used, one needs to define a separate variable $\theta^{\mathcal{P}}$ for each component $\mathcal{P}$ of partition $\mathcal{N}$. Since partitions are adaptively refined, one may have to introduce a variable θ^k for each scenario $k \in N$ and replace $\theta^{\mathcal{P}}$ by $\sum_{k \in \mathcal{P}} \theta^k$, in order to keep all the generated cuts in the same solution space, so that a single branch-and-bound tree is maintained. Otherwise, one has to maintain one separate branch-and-bound tree for each partition $\mathcal{N}$, which may be computationally inefficient.

Algorithm 2 presents the partition-based branch-and-cut algorithm for two-stage

SIP with continuous recourse in details. For the convenience of presentation, we assume that the first stage feasible set $X \subseteq \{0,1\}^{n_1}$. At the root node of the branch-and-bound tree, the algorithm performs the same way as the overall cutting plane method with coarse and fine cuts presented in [92] for two-stage stochastic linear programs. During the branch-and-bound tree, we only attempt to generate cuts at integer feasible solutions. In our experiments, we have also tested on variants of this algorithm where cuts are also added at fractional relaxation solutions obtained during the branch-and-bound tree beyond the root node. However, these variants did not lead to an improved computational performance. In Algorithm 2, for each node t in the branch-and-bound tree, $N_0(t)$ represents the set of variables fixed to zero, and $N_1(t)$ represents the set of variables fixed to one. In our default setting, the partition refinement is performed following [86, Algorithm 2], which can be done as follows:

1. Given a partition $\mathcal{N}$, consider a component $\mathcal{P} \in \mathcal{N}$ and a set of optimal dual multipliers $\hat{\lambda}^k,\ \forall k \in \mathcal{P}$.
2. Let $\{K^1, K^2, ...K^M\}$ be a partition of $\mathcal{P}$ such that $\|\hat{\lambda}^k - \hat{\lambda}^{k'}\| \leq \delta,\ \forall k, k' \in K^m, m = 1, 2, ..., M$ for some threshold $\delta > 0$.
3. Remove partition $\mathcal{P}$ from $\mathcal{N}$ and add components $K^1, K^2, ...K^M$ to $\mathcal{N}$.

This refinement strategy is motivated by [86, Theorem 2.5]. In practice, this refinement strategy tends to yield a large partition after a few refinements, especially when a small sufficient partition does not necessarily exists (which is the case for two-stage SIP with continuous recourse). In our computational experiments, we also experiment with other refinement strategies that have stronger controls on the partition size.

During the main loop of Algorithm 2, we first choose a node from the set of open nodes to process in step one. In step two, we first solve the master problem (1.9) with the integrality constraints relaxed, and obtain a first-stage relaxation solution ($(\hat{x}, \hat{\theta})$) and a lower bound for that node (lb). If the node lower bound is greater than the overall upper bound (z^{ub}), we leave this node and choose another open node to process (if there exists any). Otherwise, we update the overall lower bound (z^{lb}). The second part in step two is to attempt to generate either a coarse cut or a fine cut. Note that, if we are at the root node, these cuts are added as long as there exists any that is violated by the relaxation solution. Beyond the root node, we only add cuts at integer feasible relaxation solutions, and if a fractional relaxation solution is obtained, we go to step three and do branching. If there is no violated cut, we update the overall upper bound (z^{ub}) and choose another open node to process. Otherwise, we add a cut (coarse or fine) and repeat the process for the current node.

In addition to the partition-based coarse optimality cuts and the Benders (fine) optimality cuts, one could potentially add other valid inequalities to strengthen the relaxation of the underlying integer program during the branch-and-bound tree. In particular, valid inequalities that exploit the integrality of the first-stage variables, such as the ones shown by [12, 11], can be integrated into the algorithm in the cut generation loop.

```
Let $t \leftarrow 0$, $N_0(0) \leftarrow \emptyset$, $N_1(0) \leftarrow \emptyset$, OPEN $\leftarrow \{0\}$, $z^{lb} \leftarrow -\infty$, $z^{ub} \leftarrow +\infty$, $\gamma^c < 0$, $\gamma^f < 0$, and choose an
initial partition, e.g., $\mathcal{N} = \{1, 2, \ldots, |N|\}$
while $OPEN \neq \emptyset$ do
    step 1: Choose $t \in$ OPEN; OPEN $\leftarrow$ OPEN $\setminus\{l\}$.
    step 2: Process node $l$. Set $ADDEDCUT \leftarrow TRUE$.
    repeat
        Solve the master problem (1.9) (with integrality constraints relaxed), obtain a relaxation
        solution $(\hat{x}, \hat{\theta})$ and the node lower bound $lb$.
        if $lb > z^{ub}$ or (1.9) is infeasible then
            Go to step 1.
        else
            Update the overall lower bound $z^{lb}$.
            if $\hat{x} \in \{0,1\}^{n_1}$ or $t = 0$ then
                foreach $\mathcal{P} \in \mathcal{N}$ do
                    Solve the dual of (3.1) and obtain an optimal dual solution $\hat{\lambda}^P$.
                end
                if $\hat{\theta} - \sum_{\mathcal{P}\in\mathcal{N}} (h^{\mathcal{P}} - T^P \hat{x})^\top \hat{\lambda}^{\mathcal{P}} < \gamma^c |\hat{\theta}|$ then
                    Add $\theta \geq \sum_{\mathcal{P}\in\mathcal{N}} (h^{\mathcal{P}} - T^{\mathcal{P}} x)^\top \hat{\lambda}^{\mathcal{P}}$ to model (1.9).
                else
                    foreach $k \in N$ do
                        Solve the dual of (1.4) and obtain an optimal dual solution $\hat{\lambda}^k$.
                    end
                    Refine partition $\mathcal{N}$ according to some refinement strategy.
                    if $\hat{\theta} - \sum_{k\in N} (h^k - T^k \hat{x})^\top \hat{\lambda}^k < \gamma^f |\hat{\theta}|$ then
                        Add $\theta \geq \sum_{k\in N} (h^k - T^k x)^\top \hat{\lambda}^k$ to model (1.9).
                    else
                        Set ADDEDCUT $\leftarrow$ FALSE.
                        if $t > 0$ then
                            $z^{ub} \leftarrow \min\{lb, z^{ub}\}$
                        end
                    end
                end
            end
            Additional valid inequalities could be added here.
        end
    until $ADDEDCUT = FALSE$;
    step 3: Branching if necessary
    if $lb < z^{ub}$ then
        Choose $i$ such that $\hat{x}_i \in (0, 1)$.
        Set $N_0(t+1) \leftarrow N_0(l) \cup \{i\}, N_1(t+1) \leftarrow N_1(l)$, $N_0(t+2) \leftarrow N_0(l), N_1(t+2) \leftarrow N_1(l) \cup \{i\}$,
        and $t \leftarrow t + 2$.
        OPEN $\leftarrow$ OPEN $\cup \{t+1, t+2\}$.
    end
end
```

Algorithm 2: Partition-based branch-and-cut algorithm for solving two-stage stochastic integer programs with continuous recourse.

3.2.2 Stabilizing the partition-based decomposition algorithm

In this section, we integrate the adaptive partition-based algorithm with level decomposition, aiming at stabilizing the overall cutting plane method with partition-based coarse cuts and Benders (fine) cuts. We note that such an integration has been done for the case of two-stage stochastic linear programs in [92]. Similar as the standard level decomposition briefly reviewed in Section 1.1.2, the level master problem projects a stabilization center $\hat{x}$ onto a level set constructed by the current cutting plane approximation of the objective function and a certain level target f_{lev}. This cutting plane approximation includes the partition-based coarse cuts and Benders (fine) cuts. Specifically, the level master problem is given by:

$$\bar{z} = \min_{x \in X} \|x - \hat{x}\|_1 \tag{3.3a}$$

$$\text{s.t. } (c^T + \alpha_\ell^{av})x + \beta_\ell^{av} \le f_{lev}, \ \forall \ell \in \mathcal{L} \tag{3.3b}$$

$$(c^T + \alpha_m)x + \beta_m \le f_{lev}, \ \forall m \in \mathcal{M} \tag{3.3c}$$

where

$$\alpha_\ell^{av} = -\sum_{\mathcal{P} \in \mathcal{N}_\ell} (\hat{\lambda}_\ell^{\mathcal{P}})^\top T^{\mathcal{P}}, \ \beta_\ell^{av} = \sum_{\mathcal{P} \in \mathcal{N}_\ell} (h^{\mathcal{P}})^\top \hat{\lambda}_\ell^{\mathcal{P}} \tag{3.4}$$

$$\alpha_m = -\sum_{k \in N} (\hat{\lambda}_m^k)^\top T^k, \ \beta_m = \sum_{k \in N} (h^k)^\top \hat{\lambda}_m^k, \tag{3.5}$$

and $\mathcal{L}$ and $\mathcal{M}$ collect all the coarse and fine cuts generated so far, respectively. We note that the objective function of the level master problem (3.3) is defined using L_1-norm instead of the typical choice of L_2-norm. This is because the L_1-norm could be linearized easily so that problem (3.3) remains an MIP, which can be better handled by the commercial MIP solvers.

Instead of embedding the partition-based level decomposition within the branch-and-cut framework, we solve the level master problem (3.3) to optimality at each iteration. Integrating the level decomposition within the branch-and-bound tree can be a challenging task since in the level master problem (3.3), both the objective function and the constraints may be different in each iteration.

Algorithm 3 is divided into two main loops depending on the type of cuts that we generate to build the cutting plane relaxation. We start the algorithm with a first-stage solution ($\hat{x}$) and a lower bound (z^{lb}) of the original problem (1.5) (which can be obtained by solving the mean value problem of (1.5)). In step one, after updating the level parameter (f_{lev}), we solve the level master problem (3.3) with $\hat{x}$ as the stabilization center. If the projection model is infeasible, then f_{lev} is a valid lower bound, we then update the lower bound as well as the level parameter and repeat the process with the same partition. Otherwise, we add a coarse cut based on the solution x^{t+1} obtained from the level master problem (3.3) and update $\bar{z}$. We change the stabilization center only if a significant improvement in $\bar{z}$ is observed (which is controlled by κ_f). Whenever the relative gap between lower bound (z^{lb}) and upper bound estimated in the coarse cut loop ($\bar{z}$), given by gap^c, is small enough, we go to step two. In step two, we add a fine cut (3.3c) (by solving the second-stage problems for all scenarios), refine the current partition, update the current upper bound, and use the incumbent solution as the stabilization center in the next iteration.

In the next section, we elaborate more on implementation details regarding the algorithms developed in this section.

```
$t \leftarrow 0$, $\kappa_f, \kappa \in (0,1)$, $\bar{z}^t \leftarrow +\infty, z^{ub} \leftarrow \infty$.
Choose an initial $\hat{x}^t \in X$, obtain $z^{lb}$, e.g., from solving the mean-value problem.
Choose an initial partition $\mathcal{N}$, and set tolerance parameters $\epsilon^c \geq \epsilon^f > 0$. Calculate
gap$^c \leftarrow \frac{\bar{z}^t - z^{lb}}{|z^{lb}|}$, gap$^f \leftarrow \frac{z^{ub} - z^{lb}}{|z^{lb}|}$.
repeat
    step 1: Coarse cut loop
    repeat
        Set $f_{lev} = \kappa\, z^{lb} + (1-\kappa)\bar{z}^t$.
        Solve model (3.3).
        if model (3.3) is INFEASIBLE then
            $z^{lb} \leftarrow f_{lev}$, $\bar{z}^{t+1} \leftarrow \bar{z}^t$, $\hat{x}^{t+1} \leftarrow \hat{x}^t$, $t \leftarrow t+1$.
        else
            Obtain $x^{t+1}$ from projection model (3.3), and add a coarse cut to
            model (3.3).
            $\bar{z}^{t+1} \leftarrow \min\left\{\bar{z}^t, c^\top x^{t+1} + \sum_{\mathcal{P} \in \mathcal{N}} (h^{\mathcal{P}} - T^{\mathcal{P}} x^{t+1})^\top \hat{\lambda}^{\mathcal{P}}\right\}$,
            gap$^c \leftarrow \frac{\bar{z}^{t+1} - z^{lb}}{|z^{lb}|}$.
            if $\bar{z}^{t+1} < \bar{z}^t - \kappa_f(\bar{z}^t - f_{lev})$ then
                $\hat{x}^{t+1} \leftarrow x^{t+1}$.
            else
                $\hat{x}^{t+1} \leftarrow \hat{x}^t$.
            end
        end
    until gap$^c < \epsilon^c$;
    step 2: Fine cut loop
    Add a fine cut to model (3.3).
    Set $z^{ub} \leftarrow \min\left\{z^{ub}, c^\top \hat{x}^{t+1} + \sum_{k \in N} (h^k - T^k \hat{x}^{t+1})^\top \hat{\lambda}^k\right\}$, $\bar{z}^{t+1} \leftarrow z^{ub}$,
    gap$^f \leftarrow \frac{z^{ub} - z^{lb}}{|z^{lb}|}$, and $t \leftarrow t+1$.
until gap$^f < \epsilon^f$;
```
Algorithm 3: Partition-based level decomposition for solving two-stage stochastic integer programs with continuous recourse.

3.3 Computational Experiments

In this section we present and discuss the computational results obtained by the proposed adaptive partition-based algorithms for two-stage SIP with continuous recourse on several different test instances: sslp [2], storm [54], and cap [12] (see also [55]). We note that some of these instances involve integer variables in the second stage, and therefore do not have continuous recourse structure. In these cases, we relax the integrality constraints in the second stage problem. Also, the set of "storm" instances does not have integer variables in the first stage in the original problem, and we impose integrality constraints on these first stage variables to make them our test instances. We describe the problem profiles of the test instances in Table 6. Except for the "cap" instances, where we directly use the instances available from [12], we randomly generate our test instances with various sample sizes. Specifically, for "storm" instances, we estimate the sample mean $\hat{\mu}$ and sample variance $\hat{\sigma}^2$ of each random variable according to the scenario information contained in the original data files [54], and generate random samples from a normal distribution given by $N(\hat{\mu}, \frac{2}{3}\hat{\sigma})$; for "sslp" instances, since the realizations of the random variables are all 0's and 1's, we generate random samples from a binomial distribution using parameter p estimated from the scenario information contained in the original data files [2]. For each instance and scenario size, we generate five replications, and report the average results of these five replications.

Table 6. Profiles of test instances ($|N|$ is the number of scenarios, n_1 and m_1 are the numbers of first-stage variables and constraints, respectively, and n_2 and m_2 are the numbers of second-stage variables and constraints, respectively).

Instances	$\|N\|$	n_1	n_2	m_1	m_2
sslp (5-25)	$\{5000, 10000, 20000\}$	5	130	1	30
sslp (10-50)	$\{5000, 10000, 20000\}$	10	510	1	60
sslp (15-45)	$\{5000, 10000, 20000\}$	15	690	1	60
Storm	$\{500, 1000, 5000\}$	121	1259	185	528
cap (10)	$\{100, 500, 1000\}$	25	1250	1	75
cap (11)	$\{100, 500, 1000\}$	50	2500	1	100

3.3.1 Computational setup

All the experiments were conducted on a Linux workstation with 3.0 GHz Intel CPU and 6 GB of RAM. We used the commercial solver Gurobi 7.0.1 to implement all the algorithms in our computations. We modified the following Gurobi parameters and left the rest as the default setting: we set the number of threads to one, the time limit as one hour (3600 seconds), the integer feasibility tolerance parameter to be 10^{-5}, the optimality gap threshold (MIPGap) as 10^{-4}, and we turned on the *PreCrush* parameter. All algorithms are implemented in Python 3.5 and we used Numpy [93] to do all the matrix operations.

3.3.2 Implementation details

We implemented the proposed adaptive partition-based decomposition algorithms presented in Algorithm 2 (labeled as "Partition-B&C") and Algorithm 3 (labeled as "Partition-Level"). To implement Algorithm 2 and the classical Benders decomposition, we maintained a single branch-and-bound tree and use the Callback utility in Gurobi to add cuts. Gurobi provides two ways to add a cut: cbCut (user cuts) and cbLazy (lazy constraints), where user cuts are known as valid inequalities that are added to strengthen the problem formulation, whereas the lazy constraints are the constraints that are necessary to ensure the validity of the formulation. At the root node, we used cbCut for adding cuts at fractional relaxation solutions and cbLazy for integer feasible relaxation solutions. Beyond the root node, we only used cbLazy in our implementation, since we only added cuts at integer relaxation solutions. Furthermore, in our implementation, we called the cut generation routine only when Gurobi found a new incumbent solution. We only called the refinement routine in Algorithm 2 within the root node, since we found in our experiments that performing refinements beyond the root node led to a worse computational performance. We set the parameters in the proposed partition-based decomposition algorithms as follows. We set the cut violation thresholds to be $\gamma^c = 10^{-2}$ (coarse cuts) and $\gamma^f = 10^{-8}$ (fine cuts) in Algorithm 2 and Algorithm 3, and the refinement threshold to be $\delta = 10^{-5}$. In the partition-based level decomposition (Algorithm 3), we set the optimality gap thresholds to be $\epsilon^c = \epsilon^f = 10^{-4}$, and level parameters $\kappa = \kappa^f = 0.3$.

In addition, we have implemented the following algorithms in our experiments for the sake of comparison:

- **Extensive**: Solve the extensive formulation (1.6) by Gurobi using the default setting.
- **Benders**: Apply single-cut version of Benders decomposition (presented in Section 1.1.1).
- **Level**: Apply standard level decomposition (presented in Section 1.1.2).
- **Partition-Naive**: Instead of solving the partition-based master problem (1.9) in a branch-and-cut manner, we solve each master problem (1.9) with respect to a partition $\mathcal{N}$ to optimality (as an MIP), and then refine the partition and add cuts with respect to the corresponding optimal solution. This follows the original idea of applying partition-based algorithm to solve two-stage stochastic linear programs [86].

We use the following abbreviations for labels of columns in the tables presented below:

- **T**: Average computational time when the time limit is not hit for all five replications. Otherwise, we report in this column using format $A\%(B)$, where $A\%$ is the average optimality gap and B is the number of instances solved to optimality within the time limit among the five replications.
- **N**: Average number of explored nodes in the branch-and-bound tree.
- **RGap**: Average root optimality gap $\frac{U^*-R}{U^*}$, where U^* is the final objective value and R is the lower bound obtained after the root node is processed.

- **Cut**: Average number of Benders optimality cuts added during the algorithm.
- **FCut**,**CCut**: Average number of fine cuts and coarse cuts added during the algorithm, respectively.
- $|\mathcal{N}|$: Average final partition size.

Table 7. Computational results for algorithms “Extensive”, “Benders” and “Level”.

Instances	$\lvert N \rvert$	Extensive			Benders				Level	
		T	N	RGap	T	N	RGap	Cut	T	Cut
sslp (5-25)	5000	818.23	29	23.18 %	59.27	20	39.91 %	18	16.01	20
sslp (5-25)	10000	9.81 % (1)	27	23.19 %	123.69	18	39.17 %	18	31.76	20
sslp (5-25)	20000	> 100 % (0)	0	> 100 %	249.77	19	39.14 %	17	63.67	20
sslp (10-50)	5000	12.66 % (0)	31	12.43 %	1612.04	557	29.91 %	196	345.26	181
sslp (10-50)	10000	> 100 % (0)	0	> 100 %	3208.92	572	26.55 %	193	688.65	180
sslp (10-50)	20000	> 100 % (0)	0	> 100 %	6739.56	577	29.88 %	199	1434.69	181
sslp (15-45)	5000	> 100 % (0)	0	> 100 %	1467.78	986	47.23 %	118	349.64	84
sslp (15-45)	10000	> 100 % (0)	0	> 100 %	2865.11	988	47.22 %	115	717.5	86
sslp (15-45)	20000	> 100 % (0)	0	> 100 %	5818.78	989	47.23 %	115	1444.01	84
Storm	500	68.17	15	0.04 %	0.72 (0)	2137	23.77 %	379	28.07	19
Storm	1000	223.98	19	0.04 %	916.83	564	5.21 %	90	36.57	14
Storm	5000	M†	M	M	0.12 % (2)	875	5.22 %	74	206.62	16
cap (10)	100	98.58	1	0.07 %	8.91	20	17.66 %	26	1.21	24
cap (10)	500	1.03 % (0)	8	1.69 %	44.35	22	5.13 %	26	5.28	23
cap (10)	1000	1.72 % (0)	0	1.76 %	90.35	16	2.04 %	26	10.75	23
cap (11)	100	0.04 % (2)	79	0.05 %	0.33% (0)	96411	0.35 %	5054	0.23 % (0)	1729
cap (11)	500	0.08 % (0)	0	0.08 %	0.06% (0)	13926	0.06 %	1072	0.15 % (0)	2086
cap (11)	1000	0.04 % (0)	0	0.05 %	0.03% (0)	5471	0.03 %	540	0.07 % (0)	1317

‘†’: a memory issue is encountered.

Table 8. Computational results for three variants of the adaptive partition-based decomposition algorithms, “Partition-B&C” (Algorithm 2), “Partition-Naive” and “Partition-Level” (Algorithm 3).

Instances	$\|N\|$	Partition-B&C			Partition-Naive	Partition-Level
		T	N	RGap	T	T
sslp (5-25)	5000	4.69	16	28.75 %	1.42	1.49
sslp (5-25)	10000	11.71	17	29.20 %	3.61	3.59
sslp (5-25)	20000	19.17	20	27.81 %	6.44	6.54
sslp (10-50)	5000	25.01	568	32.44 %	23.97	15.14
sslp (10-50)	10000	33.81	581	26.04 %	30.83	19.45
sslp (10-50)	20000	48.92	583	32.32 %	38.65	25.12
sslp (15-45)	5000	82.41	1481	44.37 %	113.71	74.98
sslp (15-45)	10000	133.81	1458	44.73 %	189.57	153.99
sslp (15-45)	20000	212.99	1488	44.82 %	303.79	245.78
Storm	500	250.23	1079	50.66 %	59.85	15.17
Storm	1000	224.52	907	5.38 %	118.84	21.09
Strom	5000	1079.07	912	5.37 %	705.9	85.08
cap (10)	100	0.98	21	12.51%	1.09	0.25
cap (10)	500	3.27	21	3.13 %	2.27	0.43
cap (10)	1000	4.17	15	1.16 %	3.14	0.71
cap (11)	100	0.6% (0)	222363	0.65%	>100%	>100%
cap (11)	500	0.1% (0)	137232	0.1%	>100%	>100%
cap (11)	1000	0.02% (0)	210760	0.03%	>100%	>100%

Table 9. Number of cuts (coarse and fine) and the final partition size for three variants of the adaptive partition-based decomposition algorithms, “Partition-B&C” (Algorithm 2), “Partition-Naive” and “Partition-Level” (Algorithm 3).

Instances	$\|N\|$	Partition-B&C			Partition-Naive			Partition-Level		
		FCut	CCut	$\|\mathcal{N}\|$	FCut	CCut	$\|\mathcal{N}\|$	FCut	CCut	$\|\mathcal{N}\|$
sslp (5-25)	5000	2	19	90	1	16	42	1	22	43
sslp (5-25)	10000	3	19	219	2	16	81	2	22	81
sslp (5-25)	20000	3	19	227	2	16	84	2	22	85
sslp (10-50)	5000	1	199	282	1	230	286	1	178	286
sslp (10-50)	10000	1	196	333	1	234	356	1	183	348
sslp (10-50)	20000	1	201	381	1	234	411	1	184	402
sslp (15-45)	5000	3	174	570	3	147	3068	3	139	2522
sslp (15-45)	10000	4	167	786	3	149	5413	3	133	5419
sslp (15-45)	20000	4	164	892	3	147	9404	3	131	9402
Storm	500	67	7	242	23	8	498	2	32	240
Storm	1000	70	16	1	24	8	994	2	34	347
Strom	5000	68	16	1	23	8	4981	2	29	1366
cap (10)	100	3	26	50	2	24	42	1	25	40
cap (10)	500	4	26	130	3	20	103	1	22	96
cap (10)	1000	2	26	130	3	19	123	1	22	109
cap (11)	100	10	11620	100	0	2419	1	0	313	1
cap (11)	500	13	5775	339	0	4686	1	0	257	1
cap (11)	1000	38	5848	246	0	6004	1	0	370	1

Table 10. Computational result comparisons between full refinement and partial refinement strategies for variant "Partition-B&C".

Instances	$\|N\|$	Full Refinement			Partial Refinement		
		T	N	$\|\mathcal{N}\|$	T	N	$\|\mathcal{N}\|$
sslp (5-25)	5000	4.69	16	90	6.02	16	44
sslp (5-25)	10000	11.71	17	219	23.17	18	71
sslp (5-25)	20000	19.17	20	227	58.22	20	70
sslp (10-50)	5000	25.01	568	282	30.13	582	293
sslp (10-50)	10000	33.81	581	333	50.21	586	350
sslp (10-50)	20000	48.92	583	381	103.24	583	402
sslp (15-45)	5000	82.41	1481	570	93.11	1471	596
sslp (15-45)	10000	133.81	1458	786	184.73	1483	834
sslp (15-45)	20000	212.99	1488	892	381.96	1544	968
Storm	500	250.23	1079	242	188.48	1752	53
Storm	1000	224.52	907	1	242.07	907	1
Strom	5000	1079.07	912	1	1097.14	919	1
cap (10)	100	0.98	21	50	0.8	17	47
cap (10)	500	3.27	21	130	2.9	25	109
cap (10)	1000	4.17	15	130	3.66	19	125
cap (11)	100	0.6 % (0)	222363	100	0.4 % (0)	225927	98
cap (11)	500	0.1 % (0)	137232	339	0.08 % (0)	113580	300
cap (11)	1000	0.02 % (0)	210760	246	0.02 % (0)	209139	187

Comparing the results in Table 7 and Table 8, we can see that the partition-based algorithms “Partition-B&C”, “Partition-Naive” and “Partition-Level” outperform the existing algorithms that do not incorporate the idea of adaptive partitions. In particular, this can be seen clearly from pairwise comparison between algorithms “Benders” and “Partition-B&C”, and algorithms “Level” and “Partition-Level”. Comparing the performances of the three partition-based algorithms shown in Table 8, we see that the performance of algorithm “Partition-Level” dominates the performance of “Partition-Naive” in most instances, indicating the advantage of stabilization. In addition, we see that Algorithm “Partition-B&C” and “Partition-Level” are competitive in the “sslp” instances, and compared to “Partition-Level”, algorithm “Partition-B&C” performs significantly better for the “cap” instances, and worse for the “Storm” instances. This phenomenon can be explained from results shown in Table 9, which reports the number of cuts (coarse and fine) and the final partition size for the proposed partition-based decomposition algorithms. In fact, we see that the reason why algorithm “Partition-B&C” is outperformed in the “Storm” instances is that too many fine cuts were generated compared to the other two algorithms. This is due to the fact that we only perform refinements within the root node. We can also see that algorithm “Partition-B&C” usually ends with a smaller partition than the other two algorithms. This is not surprising since we do not perform partition refinement beyond the root node for algorithm “Partition-B&C”. We note that algorithms “Partition-Level” and “Partition-Naive” did not reach the refinement step within the time limit for the “cap (11)” instances, so that no fine cuts were added to the model and the final partition size is 1 for these instances. Because of this, we do not include results for this instance in the experiment on heuristic refinement rules presented in Section 3.3.3. As a summary, although we see significant advantages of integrating the adaptive partition into decomposition algorithms, the performances of different partition-based algorithms vary case by case, and both algorithms "Partition-B&C" and "Partition-Level" are competitive according to our computational results.

3.3.3 Heuristic refinement strategies

We consider two types of heuristic refinement strategies, aiming at a stronger control on the partition size after a partition refinement. The first heuristic refinement strategy applies a refinement procedure which we call "partial refinement", to avoid solving all the second-stage subproblems immediately after a coarse cut fails to yield any violation, but do so in a sequential manner. The second heuristic refinement strategy maintains a decreasing sequence for the refinement parameter δ, instead of a fixed parameter throughout the algorithm.

The first heuristic refinement strategy is motivated by the concern in our implementation that, whenever a coarse cut generated is not violated by the relaxation solution, we perform a "full" refinement on the current partition by solving all the second-stage subproblems for all scenarios, using the corresponding dual optimal solutions. A more conservative idea is to perform a partial refinement of the partition, by sequentially solving the second-stage subproblems with respect to scenarios only within each component of the current partition, until the resulting cut is violated by the current relaxation solution. Therefore, if not all components of a partition is refined, the resulting cut is a mixture of fine cut and coarse cut, which we refer to as a “semi-coarse cut”. It is clear that the semi-coarse cut can be seen as the weakest cut (generated in this fashion) that is necessary for the algorithm to move on to a new iteration. We show our experiment results on this partial refinement strategy using algorithm “Partition-B&C” in Table 10. We see from Table 10 that doing partial refinement instead of full refinement does not yield improvement in terms of computational time. One possible explanation is that although the increase in partition size can be better controlled by the partial refinement strategy, the semi-coarse cuts are relatively weak, which may lead to more cut generation iterations.

The second heuristic refinement strategy is motivated by the concern in the simple refinement strategy described in Section 3.2.1 that, a fixed small threshold $\delta = 10^{-5}$ could yield a very large partition after the first few refinements. We next consider a heuristic refinement strategy to impose more controls on the allowable increase in the partition size using an adaptive threshold value. Specifically, we start with a larger threshold and then gradually decrease it to 10^{-5} during the solution process, using a sequence $\frac{1}{n^\tau}$, where n is the number of times that a fine cut is added, and τ controls the speed how the threshold δ decreases (we set $\tau = 3, 6,$ and 10 in our experiments). In the case of algorithm “Partition-B&C” (Algorithm 2), we also need to determine how far beyond the root node that we are allowed to refine the partitions (in contrast to the previous implementation where refinement is only performed within the root node). This is because we may leave the root node with a very small partition due to this heuristic refinement strategy, making the coarse cuts too weak to be useful after the root node. To do so, we set a limit on the node count up to which we perform this heuristic refinement strategy. In our experiments we set this limit to 100. Table 11 and 12 show the computational performances of this heuristic refinement strategy for algorithm “Partition-B&C” and “Partition-Level” (We skipped algorithm “Partition-Naive” since its performance is dominated by “Partition-Level”), respectively.

Table 11. Computational results for algorithm “Partition-B&C” using a heuristic refinement strategy.

Instances	$\|N\|$	Partition-B&C		Partition-B&C ($\tau = 3$)		Partition-B&C ($\tau = 6$)		Partition-B&C ($\tau = 10$)	
		T	$\|\mathcal{N}\|$	T	$\|\mathcal{N}\|$	T	$\|\mathcal{N}\|$	T	$\|\mathcal{N}\|$
sslp (5-25)	5000	4.69	90	2.27	34	2.85	59	2.95	63
sslp (5-25)	10000	11.71	219	10.26	70	12.35	198	11.44	251
sslp (5-25)	20000	19.17	227	3.96	2	18.74	171	20.87	258
sslp (10-50)	5000	25.01	282	410.06	4162	202.98	2530	111.87	1193
sslp (10-50)	10000	33.81	333	341.69	3119	595.97	7228	41.53	331
sslp (10-50)	20000	48.92	381	1589.24	14071	1295.53	14182	64.40	367
sslp (15-45)	5000	82.41	570	458.29	2723	424.39	4247	77.49	453
sslp (15-45)	10000	133.81	786	876.91	5812	985.55	8343	189.08	1382
sslp (15-45)	20000	212.99	892	2927.56	16649	1611.09	11889	238.38	864
Storm	500	250.22	242	62.61	495	60.93	495	61.01	495
Storm	1000	224.52	1	287.54	218	286.25	218	212.87	218
Strom	5000	1079.07	1	1482.50	1045	1688.47	1610	1553.74	1610
cap (10)	100	0.98	50	0.46	13	0.94	40	0.83	40
cap (10)	500	3.27	130	0.44	4	1.69	68	4.07	130
cap (10)	1000	4.17	130	0.76	3	0.83	36	2.07	88
cap (11)	100	0.6% (0)	100	0.12% (0)	16	0.3% (0)	64	0.3% (0)	100
cap (11)	500	0.1% (0)	339	0.009% (1)	6	0.01% (1)	66	0.05% (0)	356
cap (11)	1000	0.02% (0)	246	0.007% (2)	7	0.01% (2)	71	0.01% (1)	189

Table 12. Computational results for algorithm “Partition-Level” using a heuristic refinement strategy.

Instances	K	Partition-Level		Partition-Level ($\tau = 3$)		Partition-Level ($\tau = 6$)		Partition-Level ($\tau = 10$)	
		T	$\lvert\mathcal{N}\rvert$	T	$\lvert\mathcal{N}\rvert$	T	$\lvert\mathcal{N}\rvert$	T	$\lvert\mathcal{N}\rvert$
sslp (5-25)	5000	1.49	43	1.76	8	1.45	27	1.43	33
sslp (5-25)	10000	3.59	81	3.22	5	3.21	35	3.56	66
sslp (5-25)	20000	6.54	85	6.32	3	6.20	28	6.46	72
sslp (10-50)	5000	15.14	286	11.99	2	12.01	6	11.96	9
sslp (10-50)	10000	19.45	348	15.17	2	15.03	2	14.93	3
sslp (10-50)	20000	25.12	402	20.89	2	20.70	12	20.24	2
sslp (15-45)	5000	74.98	2522	19.83	5	23.58	339	60.01	2480
sslp (15-45)	10000	153.99	5419	35.39	5	36.40	146	98.41	3941
sslp (15-45)	20000	245.78	9402	66.15	5	67.69	151	148.15	6097
Storm	500	15.17	240	14.57	240	14.77	240	15.08	233
Storm	1000	21.09	347	21.94	374	22.37	374	22.11	374.4
Storm	5000	85.08	1366	52.39	723	84.60	1366	84.69	1366
cap (10)	100	0.25	40	0.24	3	0.25	25	0.26	29
cap (10)	500	0.43	96	0.42	4	0.40	44	0.44	91
cap (10)	1000	0.71	109	0.75	3	0.67	36	0.66	62

We see from Table 11 and 12 that the heuristic refinement strategy is more useful for algorithm “Partition-Level”, in the sense that both computational time and the partition size decreases significantly, especially for the “sslp” instances. Heuristic refinement does not yield any improvement for algorithm “Partition-B&C” in most cases, especially when τ is relatively small. However, we see that this strategy makes it possible to solve some of the hardest instances in our test set, “cap(11)”, to optimality within the time limit. These promising results motivate further research on developing effective heuristic refinement strategies.

3.4 Conclusion

In this chapter, we proposed several adaptive partition-based decomposition algorithms for solving two-stage stochastic integer programs with continuous recourse, and performed an extensive computational study of the proposed algorithms on a diversified set of test instances. Computational results indicate that the proposed partition-based algorithms yield a better performance than the state-of-the-art decomposition approaches. We also proposed several heuristic partition refinement strategies to control the partition size during the algorithm, which is potentially useful when a sufficient partition does not necessarily exist, as in the case of two-stage stochastic integer programs with continuous recourse. The proposed algorithms have a key limitation that the second stage problems should contain continuous variables only. A natural extension of this work is to consider the more challenging case when the second stage also involves some integer variables. In this case, we expect to embed the adaptive partition-based algorithms into lower bounding techniques such as dual decomposition.

CHAPTER 4

PARTITION-BASED DUAL DECOMPOSITION FOR TWO-STAGE STOCHASTIC INTEGER PROGRAMS WITH INTEGER RECOURSE

4.1 Introduction

In this chapter, we relax our assumptions of stochastic integer programs (SIP) from chapter 3 to the case where both first-stage and second-stage variables are allowed to be integer. This assumption introduces a higher level of complexity to the problem since every subproblem now is a mixed integer program (MIP), which usually is very challenging to solve. Moreover, we cannot adopt the decomposition methods from two-stage stochastic linear program (in the manner we did in chapter 3) due to the integrality constraints on second-stage variables. Our focus in this chapter is to utilize partition-based relaxation framework and dual decomposition (DD) to improve the lower bound for the optimal objective value of an SIP within a time limit. The proposed method could be used instead of DD, whenever the time limit does not allow to obtain a lower bound from DD. We also modify this method to obtain a bound close to DD with scenario grouping (GDD), while improve the computational time. For ease of use, we refer to the proposed method as partition-based dual decomposition (PDD).

SIPs are applied in broad application areas such as energy planning [38, 66], vehicle routing [50], scheduling [28], and many others. Despite their numerous applications, solving SIPs is a very challenging task, unless they have special structures to exploit [83, 10]. This difficulty is due to the integrality constraints on the second-stage variables. In order to evaluate the second-stage cost for a fixed first-stage solution in a given scenario, we need to solve an MIP which is generally NP-hard [1].

Although there are different methods in the literature especially designed for SIPs with some specific structures (e.g., [10]), the more general SIPs are still very difficult to solve. Scenario decomposition methods, (DD [17], progressive hedging [71]) have been developed to tackle SIPs with integer recourse. These methods are based on the Lagrangian relaxation of extensive model. One of the decomposition methods that has been extensively studied is DD [17, 56], on which we base our proposed framework. In this method, similar to other scenario decomposition methods, we dualize the nonanticipativity constraints in order to obtain a Lagrangian relaxation problem which is decomposable for each scenario, so each scenario-based subproblem could be solved independently. Since the Lagrangian dual problem could be formulated as a convex optimization problem, a cutting-plane method can be applied to solve the Lagrangian dual problem. It is well known that this dual problem gives a lower bound on the optimal objective value of the original SIP which is better than one obtained from the linear programming (LP) relaxation.

It was mentioned in [17] that to further improve this bound, we can group the scenarios together and form subproblems based on a group of scenarios instead of a single scenario. Different aspects of scenario grouping for SIP with integer recourse have been studied in the literature [74, 78, 30, 29, 26]. For the specific case of two-stage mixed 0–1 models, an improvement in lower bound using scenario groups for different scenario decomposition algorithms was reported in [29]. Their work was further studied in [30] for multistage mixed 0–1 problems. In a recent work [74], authors formulated an MIP to optimally assign scenarios to groups in order to maximize the improvement we gain in lower bound from grouping scenarios. Another line of work in scenario grouping for SIP is studied in [78]. They generated all possible groups of scenarios with a fixed cardinality and then for each group they solve corresponding group subproblem (original SIP which is limited to those scenarios in the group). Then, the lower bound is given by taking average over the scenario groups. They proved that this bound is monotonically non-decreasing as cardinality of the group increases. This framework was later extended to the multi-stage case in [79, 13]. Authors in [13] provided a theoretical hierarchy of bounds by enumerating all possible partitions of scenarios with subsets of nearly equal sizes. In practice, they showed that only a sample of partition is enough to obtain a good bound. In our proposed method, we consider the bound obtained from only one partition of scenarios. Moreover, we do not control the size of each group in a partition.

Before proceeding to the contribution of this study, it is worthwhile to define a few terms which we will use throughout the chapter: we refer to any subset of the original scenario set as **group**; we refer to a set of mutually-exclusive groups which covers the original scenario set as **partition**; we refer to the Lagrangian dual problem for partition-based relaxation of a two-stage SIP as **partition-based Lagrangian dual problem**, we refer to a subproblem which contains all the scenario dependent constraints indexed by a specific group as **group subproblem**; we refer to the partition-based relaxation of a group subproblem (which is formed by aggregating all the scenario dependent constraints in that group) as **partition-based subproblem**. The contribution of this study is focused on developing some computational strategies in order to obtain a relatively better bound than those obtained from DD and GDD, when the computational time is very limited. We will empirically show that with a very tight time limit, one can obtain a very good lower bound by solving partition-based relaxation of group subproblem. We develop two strategies for this purpose. In the first strategy, to which we refer as "One-phase PDD", we solve a partition-based Lagrangian dual problem for a given partition of scenarios. In this method, a subproblem is defined based on a group of scenarios (similar to GDD); however, instead of considering all the scenarios within the group separately, we aggregate them (by adding the rows of scenario dependent constraints indexed by that specific group). Hence, the partition-based subproblem is a relaxation of the corresponding subproblem in GDD. In the second strategy, which we call "Two-phase PDD", we start with partition-based Lagrangian dual problem. When first phase is done, we collect some intermediate information to utilize in the second phase. The second phase is a GDD corresponding the the partition which was used to formulate the partition-based Lagrangian dual problem.

In the following we clarify the main methodological difference between the proposed framework in this chapter and one studied in the previous chapter. For SIPs with continuous recourse, we did not work with a new problem formulation, instead we propose strategies to accelerate the Benders decomposition framework. We estimate $f(x) = \min_{y_k}\{\sum_{k\in N} d^T y^k \mid T^k x + W y^k \leq h^k,\ \forall k \in N\}$ using a weaker version of Benders cut (i.e., coarse cut). To obtain a coarse cut, we form a partition of scenarios. For each element in this partition, we solve the corresponding partition-based subproblem, generate a Benders optimality cut, and add it to the master model. Since the cut is not obtained by solving every single scenario in that element, it is weaker than the original Benders optimality cut. For SIPs with integer recourse, we form a partition over the scenario set, and define a new (relaxed) model based on this partition. So, in the first case, the contribution is on solution methodology, utilizing coarse Benders optimality cut defined using scenario partitions. specifically generating new form of cuts. In the second case, the contribution is to define a new formulation in order to obtain a lower bound.

The rest of the chapter is organized as follows. In Section 4.2 we review DD method as well as a regularized cutting-plane method to solve. Next, we explain the scenario grouping problem and its partition-based relaxation. Two computational strategies are presented in Section 4.3. In Section 4.4 we discuss the computational experiments and their numerical results. Some conclusion remarks are discussed in Section 4.5

4.2 Problem Setting and Background

In this section, we review DD. We then discuss the scenario grouping problem in Section 4.2.2 and its partition-based relaxation form in Section 4.2.3.

4.2.1 Dual Decomposition

Given a finite set of scenarios $N = \{1, 2, \dots, |N|\}$ and using sample average approximation, we can formulate the two-stage stochastic integer program with integer recourse as:

$$
\begin{aligned}
z^* = \min_{x, y^1, \dots, y^{|N|}} \ & c^\top x + \frac{1}{|N|} \sum_{k=1}^{|N|} d^\top y^k && \text{(4.1a)} \\
\text{s.t. } & Ax \leq b, && \text{(4.1b)} \\
& T^k x + W y^k \leq h^k, \quad \forall k \in N && \text{(4.1c)} \\
& x \in X \subset \mathbb{R}^{n_1}, y^k \in Y \subset \mathbb{R}^{n_2}, \forall k \in N. && \text{(4.1d)}
\end{aligned}
$$

In model (4.1), $c \in \mathbb{R}^{n_1}$, $d \in \mathbb{R}^{n_2}$, and $A \in \mathbb{R}^{m_1 \times n_1}$. In constraint (4.1c), $T^k \in \mathbb{R}^{m_2 \times n_1}$ is called the technology matrix and could be scenario dependent; $W \in \mathbb{R}^{m_2 \times n_2}$ is the recourse matrix and we assume it is fixed for all scenarios. We further assume that sets X and Y impose integrality constraints on first-stage variables x and second-stage variables y, respectively. For every $k \in N$, we define the following set:

$$
S^k := \left\{ (x, y^k) : Ax \leq b, T^k x + W y^k \leq h^k, x \in X, y^k \in Y \right\}. \tag{4.2}
$$

If we define a copy of first-stage variable for each scenario, then an alternative formulation (variable-split [56]) for (4.1) is:

$$z^* = \min_{\substack{x,x^1,\dots,x^{|N|}\\ y^1,\dots,y^{|N|}}} \sum_{k=1}^{|N|} \frac{1}{|N|}(c^\top x^k + d^\top y^k) \tag{4.3a}$$

$$\text{s.t. } x^k - x = 0,\ \forall k \in N, \tag{4.3b}$$

$$(x^k, y^k) \in S^k, \forall k \in N. \tag{4.3c}$$

Constraints (4.3b) are called the "nonanticipativity" constraints which enforce all copies of first-stage variables to be equal. Relaxing constraints (4.3b) by introducing a dual vector $\lambda^k \in \mathbb{R}^{n_1}$, $\forall k \in N$, the corresponding Lagrangian relaxation of (4.3) is:

$$D(\lambda^1,\dots,\lambda^{|N|}) := \min_{\substack{x,x^1,\dots,x^{|N|}\\ y^1,\dots,y^{|N|}}} \left\{ \sum_{k=1}^{|N|} \left(\frac{1}{|N|}(c^\top x^k + d^\top y^k) + (\lambda^k)^\top x^k \right) - \sum_{k=1}^{|N|} (\lambda^k)^\top x : (x^k, y^k) \in S^k, \forall k \in N \right\}. \tag{4.4}$$

Because there is no constraint on x, we impose the condition $\sum_{k=1}^{|N|} \lambda^k = 0$ to make (4.4) bounded. In the Lagrangian dual problem, for each arbitrary $\lambda^1, \dots, \lambda^{|N|}$ such that $\sum_{k=1}^{|N|} \lambda^k = 0$, (4.4) provides a lower bound on the optimal objective value of (4.1): $D(\lambda^1, \dots, \lambda^{|N|}) \leq z^*$. Hence, the Lagrangian dual problem is to find the best such bound by solving the following problem:

$$z_{LD} = \max_{\lambda^1,\dots,\lambda^{|N|}} \left\{ \sum_{k=1}^{|N|} D_k(\lambda^k) : \sum_{k=1}^{|N|} \lambda^k = 0 \right\}, \tag{4.5}$$

where

$$D_k(\lambda^k) = \min_{x^k,y^k} \left\{ \frac{1}{|N|}(c^\top x^k + d^\top y^k) + (\lambda^k)^\top x^k : (x^k, y^k) \in S^k \right\}. \tag{4.6}$$

The objective function of problem (4.5) is a piecewise linear concave function

[56]. Therefore, we can apply a cutting-plane method to solve it. We state this method in Algorithm 4 which was proposed in [17], and further studied in [56]. The cutting-plane method overestimates $\sum_{k=1}^{|N|} D_k(\lambda^k)$ by solving the following *master problem* in each iteration:

$$\max_{\substack{\theta^1,\ldots,\theta^{|N|}\\ \lambda^1,\ldots,\lambda^{|N|}}} \sum_{k=1}^{|N|} \theta^k \tag{4.7a}$$

$$\text{s.t.} \sum_{k=1}^{|N|} \lambda^k = 0, \tag{4.7b}$$

$$\theta^k \le D_k(\hat{\lambda}^k) + (\hat{x}^k)^\top (\lambda^k - \hat{\lambda}^k),\ \forall \hat{\lambda}^k \in I, \forall k \in N, \tag{4.7c}$$

where I is a set that contains all encountered $\hat{\lambda}^k$ during the course of the algorithm and $\hat{x}^k$ is the corresponding solution of subproblem (4.6) for $\lambda^k = \hat{\lambda}^k$.

```
Set a convergence tolerance $\epsilon$ ;
Set $I \leftarrow \emptyset$, $\hat{\lambda}^k \leftarrow 0, \forall k \in N$ ;
Solve (4.6), $\forall k \in N$, save optimal value $D_k(\hat{\lambda}^k)$ and solution $\hat{x}^k$ ;
repeat
    Add $\theta^k \le D_k(\hat{\lambda}^k) + (\hat{x}^k)^\top(\lambda^k - \hat{\lambda}^k), \forall k \in N$ to the model (4.7) if it is violated
    by $(\hat{\theta}^k, \hat{\lambda}^k)$ ;
    Solve (4.7), save $\hat{\theta}^k$ and $\hat{\lambda}^k$ ;
    Set $I \leftarrow I \cup \{\hat{\lambda}^k\}$ ;
    Solve (4.6), $\forall k \in N$, save optimal value $D_k(\hat{\lambda}^k)$ and solution $\hat{x}^k$ ;
    Set $gap \leftarrow \frac{\sum_{k=1}^N \hat{\theta}^k - \sum_{k=1}^N D_k(\hat{\lambda}^k)}{|\sum_{k=1}^N \theta^k|}$ ;
until $gap \le \epsilon$;
```

Algorithm 4: Cutting-plane algorithm

Algorithm 4 is not very efficient in its original form. Indeed, it is well known that cutting-plane algorithms suffer from instability in general; in a sense that they might take large jumps when they are close to the optimal solution. To alleviate this shortcoming, various regularization methods have been studied in the literature such as proximal bundle method [72] and level method [53]. We adapt the level method in order to stabilize the cutting-plane method in Algorithm 4. The core idea of the level method is to define a level set based on the current cutting-plane model and a level parameter. Then, we obtain the next trial point by projecting the current stabilization center (which usually is the incumbent solution) onto the level set, which keeps the next trial point as close as possible to the stabilization center. Instead of solving a master model of the form (4.7), in level method we solve the following *projection problem*:

$$\min_{\substack{\theta^1,\dots,\theta^{|N|}\\ \lambda^1,\dots,\lambda^{|N|}}} \sum_{k=1}^{|N|} |\lambda^k - \bar{\lambda}^k| \tag{4.8a}$$

$$\text{s.t. } \sum_{k=1}^{|N|} \lambda^k = 0, \tag{4.8b}$$

$$\theta^k \leq D_k(\hat{\lambda}^k) + (\hat{x}^k)^\top(\lambda^k - \hat{\lambda}^k),\ \forall \hat{\lambda}^k \in I, \forall k \in N, \tag{4.8c}$$

$$\sum_{k=1}^{|N|} \theta^k \geq lev, \tag{4.8d}$$

where $lev = \kappa\ lb + (1-\kappa)ub,\ \kappa \in (0,1)$, and $\bar{\lambda}$ is the current stabilization center. lb and ub are lower and upper bound on z^*, respectively. When model (4.8) is infeasible (i.e., the level target cannot be met), then lev is a valid upper bound. We can update upper bound and re-solve the projection model. When model (4.8) is feasible, then we solve all the subproblems as before, add cuts (4.8c) to model (4.8), and update the lower bound. If the lower bound is improved, then we also update the stabilization center $\bar{\lambda}$. This method is summarized in Algorithm 5.

```
Set a convergence tolerance $\epsilon$ ;
Set $I \leftarrow \emptyset$, $\bar{\lambda}^k \leftarrow 0, \hat{\lambda}^k \leftarrow 0, \forall k \in N$, $\kappa \in (0,1)$, $status \leftarrow$ FEASIBLE ;
Initialize $lb$ and $ub$ ;
Solve (4.6), $\forall k \in N$, save optimal value $D_k(\hat{\lambda}^k)$ and solution $\hat{x}^k$ ;
repeat
    Add $\theta^k \leq D_k(\hat{\lambda}^k) + (\hat{x}^k)^\top(\lambda^k - \hat{\lambda}^k), \forall k \in N$ to the model (4.8) ;
    Solve (4.8) ;
    if model (4.8) is INFEASIBLE then
        $status \leftarrow$ INFEASIBLE ;
    end
    while status is INFEASIBLE do
        $ub \leftarrow lev$ ;
        $lev \leftarrow \kappa lb + (1-\kappa)ub$;
        Solve (4.8) ;
        if model (4.8) is FEASIBLE then
            $status \leftarrow$ FEASIBLE ;
        end
    end
    Save $\hat{\theta}^k$ and $\hat{\lambda}^k$ as optimal solution of model (4.8) ;
    Set $I \leftarrow I \cup \{\hat{\lambda}^k\}$ ;
    Solve (4.6), $\forall k \in N$, save optimal value $D_k(\hat{\lambda}^k)$ and solution $\hat{x}^k$ ;
    if $lb \leq \sum_{k=1}^N D_k(\hat{\lambda}^k)$ then
        $lb \leftarrow \sum_{k=1}^N D_k(\hat{\lambda}^k)$ ;
        $lev \leftarrow \kappa lb + (1-\kappa)ub$ ;
        $\bar{\lambda}^k \leftarrow \hat{\lambda}^k$ ;
    end
until $gap \leq \epsilon$;
```
Algorithm 5: Level method for cutting-plane algorihtm.

4.2.2 Dual Decomposition with Scenario Grouping

Let $\mathcal{P}$ be a partition on the scenario set N, i.e., $\mathcal{P} = \{P_1, P_2, \ldots, P_L\}$, $P_1 \cup P_2 \cup \cdots \cup P_L = N$ and $P_i \cap P_j = \emptyset, \forall P_i, P_j \in \mathcal{P}, i \neq j)$. Then, we can write model (4.1) with respect to partition $\mathcal{P}$:

$$z^* = \min_{x, y^1, \ldots, y^{|N|}} \; c^\top x + \frac{1}{|N|} \sum_{\ell=1}^{L} \sum_{k \in P_\ell} d^\top y^k \tag{4.9a}$$

$$\text{s.t. } (x, y^k) \in S^k, \; \forall k \in P_\ell, \; \ell = 1, 2, \ldots, L. \tag{4.9b}$$

By introducing a copy of variable x for each group and considering the nonanticipativity constraints only among different groups, we have the following formulation for (4.9):

$$z^* = \min_{\substack{x, x^1, \ldots, x^L \\ y^1, \ldots, y^{|N|}}} \; \frac{1}{L} \sum_{\ell=1}^{L} c^\top x^\ell + \frac{1}{|N|} \sum_{\ell=1}^{L} \sum_{k \in P_\ell} d^\top y^k \tag{4.10a}$$

$$\text{s.t. } x^\ell - x = 0, \; \ell = 1, \ldots, L, \tag{4.10b}$$

$$(x^\ell, y^k) \in S^k, \forall k \in P_\ell, \; \ell = 1, 2, \ldots, L. \tag{4.10c}$$

Introducing vector of Lagrangian multiplier λ^ℓ for the nonanticipativity constraints (4.10b) in each group $\ell = 1, \ldots, L$,the Lagrangian relaxation of (4.10) is given by:

$$D(\lambda^1, \ldots, \lambda^L) = \min_{\substack{x^1, \ldots, x^L \\ y^1, \ldots, y^L}} \left\{ \left(\frac{1}{L} \sum_{\ell=1}^{L} c^\top x^\ell + \sum_{\ell=1}^{L} \left[\frac{1}{|N|} \sum_{k \in P_\ell} d^\top y^k + (\lambda^\ell)^\top x^\ell \right] \right) : (x^\ell, y^k) \in S^k, \; \forall k \in P_\ell, \; \ell = 1, \ldots, L \right\}. \tag{4.11}$$

Then, we define the Lagrangian dual problem for (4.10) with respect to $\mathcal{P}$ as:

$$z_{LD}^{\mathcal{P}} = \max_{\lambda^1, \ldots, \lambda^L} \left\{ \sum_{\ell=1}^{L} D_\ell(\lambda^\ell) : \sum_{\ell=1}^{L} \lambda^\ell = 0 \right\}, \tag{4.12}$$

where

$$D_\ell(\lambda^\ell) = \min_{x^\ell, \{y^k\}_{k\in P_\ell}} \left\{ \frac{1}{L} c^\top x^\ell + \frac{1}{|N|} \sum_{k\in P_\ell} d^\top y^k + (\lambda^\ell)^\top x^\ell : (x^\ell, y^k) \in S^k,\ \forall k \in P_\ell \right\}. \tag{4.13}$$

Note, because of implicit nonanticipativity constraint among scenarios within the group, $z_{LD} \leq z_{LD}^{\mathcal{P}}$.

Since model (4.12) has the same property as model (4.5), we use the level method described in Algorithm 5 to solve it. To this end, instead of solving model (4.8), we solve the the following projection problem with respect to partition $\mathcal{P}$:

$$\min_{\substack{\theta^1,\ldots,\theta^L\\ \lambda^1,\ldots,\lambda^L}} \sum_{\ell=1}^{L} |\lambda^\ell - \bar{\lambda}^\ell| \tag{4.14a}$$

$$\text{s.t. } \sum_{\ell=1}^{L} \lambda^\ell = 0, \tag{4.14b}$$

$$\theta^\ell \leq D_\ell(\hat{\lambda}^\ell) + (\hat{x}^\ell)^\top(\lambda^\ell - \hat{\lambda}^\ell),\ \forall \hat{\lambda}^\ell \in I,\ \forall \ell = 1, \ldots, L, \tag{4.14c}$$

$$\sum_{\ell=1}^{L} \theta^\ell \geq lev. \tag{4.14d}$$

In this case, in order to generate a cut, we solve group subproblems of the form (4.13) for each group in $\mathcal{P}$ instead of (4.6).

4.2.3 Partition-based Dual Decomposition

We first present PDD. Then, we prove that PDD cannot provide a better bound than DD. We also show as a result of this proof, the refinement procedure could improve the bound.

Let $\mathcal{P}$ be a partition of scenario set N. We define set $\bar{S}^\ell$:

$$\bar{S}^\ell := \left\{(x, y^\ell) : Ax \leq b, T^\ell x + W y^\ell \leq h^\ell, x \in X, y^\ell \in Y\right\}, \tag{4.15}$$

where $T^\ell = \sum_{k \in P_\ell} T^k$, and $h^\ell = \sum_{k \in P_\ell} h^k$. Corresponding to partition $\mathcal{P}$, we define the following partition-based relaxation of the original two-stage SIP:

$$\min_{\substack{x,x^1,\ldots,x^L\\ y^1,\ldots,y^L}} \frac{1}{L}\sum_{\ell=1}^{L} c^\top x^\ell + \frac{1}{|N|}\sum_{\ell=1}^{L} d^T y^\ell \tag{4.16a}$$

$$\text{s.t. } x^\ell - x = 0, \ \ell = 1, \ldots, L, \tag{4.16b}$$

$$(x^\ell, y^\ell) \in \bar{S}^\ell, \ \ell = 1, \ldots, L. \tag{4.16c}$$

By introducing a vector of Lagrangian multipliers λ^ℓ for nonanticipativity constraints (4.16b), the Lagrangian relaxation for (4.16) is given by:

$$\bar{D}(\lambda^1, \ldots, \lambda^L) = \min_{\substack{x^1,\ldots,x^L\\ y^1,\ldots,y^L}} \left\{\sum_{\ell=1}^{L}\left(\frac{1}{L}c^\top x^\ell + \frac{1}{N}d^\top y^\ell + (\lambda^\ell)^\top x^\ell\right) : (x^\ell, y^\ell) \in \bar{S}^\ell, \ \ell = 1, \ldots, L\right\}. \tag{4.17}$$

Then, the Lagrangian dual problem for (4.16) is:

$$\bar{z}_{LD}^{\mathcal{P}} = \max_{\lambda^1,\ldots,\lambda^L} \left\{\sum_{\ell=1}^{L} \bar{D}_\ell(\lambda^\ell) : \sum_{\ell=1}^{L} \lambda^\ell = 0\right\}, \tag{4.18}$$

where,

$$\bar{D}_\ell(\lambda^\ell) = \min_{x^\ell, y^\ell} \left\{\frac{1}{L}c^\top x^\ell + \frac{1}{N}d^\top y^\ell + (\lambda^\ell)^\top x^\ell : (x^\ell, y^\ell) \in \bar{S}^\ell\right\}. \tag{4.19}$$

Similar to (4.12), we can apply Algorithm 5 to solve (4.18). Instead of solving master model (4.8), we solve projection problem (4.14) and to generate a cut, we need to solve a subproblem of the form (4.19) instead of (4.6).

To present our result on the relation between z_{LD} and $\bar{z}_{LD}^{\mathcal{P}}$, we first restate

Proposition 1 in [56] which is the primal characterization of Lagrangian dual problem. Then, we prove that $\bar{z}^{\mathcal{P}}_{LD} \leq z_{LD}$.

Proposition 1. *The optimal value z_{LD} of the Lagrangian dual* (4.5) *equals the optimal value of*

$$\min_{\substack{x^1,\ldots,x^{|N|}\\ y^1,\ldots,y^{|N|}}} \sum_{k=1}^{|N|} \frac{1}{|N|}(c^T x^k + d^T y^k) \tag{4.20}$$

$$s.t.\ (x^k, y^k) \in conv(S^k),\ \forall k \in N, \tag{4.21}$$

$$x^k - x = 0,\ \forall k \in N, \tag{4.22}$$

where $conv(\cdot)$ is the convex hull of mixed integer set.

The proof could be found in integer programming text books (e.g., [20, 65]).

Proposition 2. *For any partition $\mathcal{P}$ on N, $\bar{z}^{\mathcal{P}}_{LD} \leq z_{LD}$.*

Proof. In this proof, we use the primal characterization of (4.5) and (4.18). Note that, in optimality, we want nonanticipativity constraints (4.22) be satisfied. Hence, if we substitute x for all x^k in (4.3) and for all x^ℓ in (4.16), we have the following two LPs:

$$z_{LD} = \min_{x,y^1,\ldots,y^{|N|}} c^T x + \sum_{k=1}^{N} \frac{1}{|N|} d^T y^k \tag{4.23a}$$

$$\text{s.t. } (x, y^k) \in \text{conv}(S^k), \forall k \in N, \tag{4.23b}$$

$$\hat{z}_{LD}^{\mathcal{P}} = \min_{x,y^1,\ldots,y^L} \; c^T x + \sum_{\ell=1}^{L} \frac{1}{|N|} d^T y^\ell \tag{4.24a}$$

$$\text{s.t. } (x, y^\ell) \in \text{conv}(\bar{S}^\ell), \forall \ell = 1, \ldots, L, \tag{4.24b}$$

where $y_\ell = \sum_{k \in P_\ell} y^k$. It suffices to show that both models (4.23) and (4.24) have the same objective function, and the feasible region in (4.23) is a subset of feasible region in (4.24). Therefore, $\hat{z}_{LD}^{\mathcal{P}}$ cannot be larger than z_{LD}.

Assume $(x^*, \{(y^k)^*\}_{k=1}^{|N|})$ is an optimal solution to (4.23). Because $(x^*, (y^k)^*) \in \text{conv}(S^k)$, we have $(y^k)^* \in \text{conv}(S^k(x^*))$, where

$$S^k(x^*) = \left\{ y^k : W y^k \leq h^k - T^k x^* \;, y^k \in Y \right\}. \tag{4.25}$$

We also define

$$S^\ell(x^*) = \left\{ (y^k)_{k \in P_\ell} \in \mathbb{R}^{|P_\ell| \times n_2} : W(\sum_{k \in P_\ell} y^k) \leq \sum_{k \in P_\ell} (h^k - T^k x^*) \;, y^k \in Y, \forall k \in P_\ell \right\}. \tag{4.26}$$

We know that the following is true:

$$\times_{k \in P_\ell} \text{conv}(S^k(x^*)) = \text{conv}(\times_{k \in P_\ell} S^k(x^*)). \tag{4.27}$$

We also know that:

$$\times_{k \in \mathcal{P}_\ell} S^k(x^*) \subseteq S^\ell(x^*). \tag{4.28}$$

By applying conv$(\cdot)$ to the both sides of (4.28):

$$\text{conv}(\times_{k\in\mathcal{P}_\ell} S^k(x^*)) \subseteq \text{conv}(S^\ell(x^*)), \tag{4.29}$$

therefore:

$$\{(y^k)^*\}_{k\in\mathcal{P}_\ell} \in \times_{k\in\mathcal{P}_\ell}\text{conv}(S^k(x^*)) \Rightarrow \{(y^k)^*\}_{k\in\mathcal{P}_\ell} \in \text{conv}(\bar{S}^\ell(x^*)). \tag{4.30}$$

We re-write model (4.24) in the same space as model (4.23):

$$\hat{z}^{\mathcal{P}}_{LD} = \min_{x,y^1,\ldots,y^{|N|}} c^T x + \sum_{\ell=1}^{L}\sum_{k\in\mathcal{P}_\ell}\frac{1}{N}d^T y^k \tag{4.31a}$$

$$\text{s.t. } (x,(y^k)_{k\in\mathcal{P}_\ell}) \in \text{conv}\left(\left\{(x,(y^k)_{k\in\mathcal{P}_\ell}) : Ax \le b, T^\ell x + W\sum_{k\in\mathcal{P}_\ell} y^k \le h^\ell, y^k \in Y, k\in\mathcal{P}_\ell\right\}\right). \tag{4.31b}$$

Proof is completed because models (4.31) and (4.23) have the same objective function and the optimal solution of model (4.23) is a feasible solution of (4.31). □

Proposition 2 also justifies the refinement procedure to further improve the bound. Assume we refine partition $\mathcal{P}$ to obtain partition $\mathcal{P}^2$. Then by applying Proposition 2 (assume that $N = \mathcal{P}^2$), we have $z^{\mathcal{P}}_{LD} \le z^{\mathcal{P}^2}_{LD}$.

4.3 Partition-based Relaxation Strategies for SIP

We utilize the results from previous section to propose two different computational strategies to obtain a lower bound on the optimal objective value of an SIP. In the rest of the chapter by "PDD procedure", we refer to applying a cutting-plane method (e.g., Algorithm 5) to solve the Lagrangian dual problem for partition-based relaxation, i.e., problem (4.18); by “GDD procedure” we refer to applying a cutting-plane method to solve the Lagrangian dual problem with scenario grouping, i.e., problem (4.12) for a given partition $\mathcal{P}$.

4.3.1 Strategy 1: One-phase PDD

Based on Proposition 2, choosing PDD over DD is not justifiable in theory, since the latter always give a better bound. However, from a practical point of view, PDD could be more promising than DD. Compared to PDD, DD is computationally more demanding, since it solves more MIP subproblems in each iteration if ($L << |N|$). This could prevent DD to provide a good bound when time limit is tight.

The strategy that we propose starts with applying the PDD procedure for a given partition $\mathcal{P}$. Next, we can either stop and report the current lower bound, or refine $\mathcal{P}$ and repeat the procedure to improve the bound. One important shortcoming of this method is its inability to transfer the generated cuts from one phase to the next. Note that, the dimension of projection problem (4.14) is $L(1+n_1)$. Since L increases when we move from one phase to the next, the dimension of (4.14) changes. Therefore, the cuts are not transferable between phases. One way to handle this issue is to discard all the generated cuts and apply the PDD procedure to the new partition. However, we lose all the available information from the previous phase. We propose an approach which allows us to partially employ the available information, by using the active cuts from the final iteration of the first phase of PDD to initialize the new master model. The initialization happens as follows.

Assume $\mathcal{A} = \{(r,\ell) : \text{if subproblem } \ell \text{ at iteration } r \text{ generates the current active cut}\}$. Let $\hat{\lambda}_r^\ell$ be the dual multiplier vector corresponding to each element in $\mathcal{A}$. Further assume that after refining partition $\mathcal{P}$, we obtain a new partition $\mathcal{P}^2$ with L^2 elements ($L^2 > L$) and for each $P_\ell \in \mathcal{P}$ we have $P_\ell = P_{\ell_1} \cup P_{\ell_2}, \cdots \cup P_{\ell_{L'}}$, $P_{\ell_j} \in \mathcal{P}^2$, $j = 1, \ldots, L'$ (i.e., we have L' new subproblem derived from subproblem ℓ in previous phase). To generate the initial cuts, we only need to solve these new L' subproblem of the form (4.19). In order to maintain the condition $\sum_{\ell=1}^{L^2} \lambda^\ell = 0$, we choose the new $\hat{\lambda}^{\ell_j}$ (dual multiplier for subproblem ℓ_j derived from previous subproblem ℓ) proportional to its partition size, which is $\hat{\lambda}^{\ell_j} = \frac{\hat{\lambda}_r^\ell \times |P_{\ell_j}|}{|P_\ell|}$. To complete the initialization phase, we repeat this procedure for all active cuts.

We conducted numerical experiments with and without initialization. Empirically, we did not observe any improvement in computational time when we initialize the master model. For that reason, we only report the results for a case where the final size of the partition is fixed beforehand, and we apply PDD procedure only once.

4.3.2 Strategy 2: Two-phase PDD

An alternative to refinement in order to improve the One-phase PDD bound is to disaggregate scenarios within each group. As we discussed in chapter 3, in refinement, we form a larger partition derived from the previous one. Hence, increase the number of subproblems in (4.19). On the contrary, by disaggregation we refer to an operation which disaggregate the aggregated scenarios within each group of the partition, without changing the partition. Depending on the size of the new group, we introduce more copies of variable y in the formulation of subproblem.

As an example, assume $\mathcal{P} = \{\{1,2,3,4\},\{5,6,7\}\}$. Corresponding to $\mathcal{P}$, there are $L = 2$ subproblems with the form (4.19). If we perform refinement, we will obtain a new partition with larger size, e.g., $\mathcal{P}^2 = \{\{1,2\},\{3,4\},\{5\},\{6,7\}\}$. Corresponding to $\mathcal{P}^2$ we have $L = 4$ subproblem with the form (4.19). If we disaggregate the groups, we will have, e.g., $\mathcal{P} = \{\{\{1,2\},\{3,4\}\},\{\{5,6\},\{7\}\}\}$. We still have $L = 2$ subproblems. However, the form of the subproblems change. To introduce the formulation of the new subproblem, let $P_\ell = \{P_{\ell_1} \cup P_{\ell_2} \cdots \cup P_{\ell_{L'}}\}$ be the disaggregation of $P_\ell \in \mathcal{P}$. Then the new formulation is:

$$\hat{D}_\ell(\lambda^\ell) = \min_{x^\ell, y^{\ell_1},\ldots,y^{\ell_{L'}}} \left\{ \frac{1}{L} c^\top x^\ell + \frac{1}{|N|} \sum_{j=1}^{L'} d^\top y^{\ell_j} + (\lambda^\ell)^\top x^\ell : (x^\ell, y^{\ell_j}) \in \bar{S}^{\ell_j}, j = 1,\ldots,L' \right\}. \tag{4.32}$$

After disaggregation, similar to the first strategy, the cuts could not be transferred to the new phase. In this case, the problem is not dimension mismatch. Consider the form of the cut that we generate: $\theta^\ell \leq D_\ell(\hat{\lambda}^\ell) + (\hat{x}^\ell)^\top(\lambda^\ell - \hat{\lambda}^\ell)$. During the first phase, we generate each cut by solving (4.19) and use $\bar{D}_\ell(\hat{\lambda}^\ell)$. After disaggregation, we need to solve (4.32) and use $\hat{D}_\ell(\hat{\lambda}^\ell)$ to generate such a cut. Since $\bar{D}_\ell(\hat{\lambda}^\ell) \leq \hat{D}_\ell(\hat{\lambda}^\ell)$, and we are maximizing $\sum_{\ell=1}^{L} \theta^\ell$ in each iteration, the cuts generated in first phase could be invalid for the next phase. For this strategy, our computational experiments indicate that initializing the master model for the next phase improves the computational time. To initialize the master model for the next phase, for every element in $\mathcal{A}$, we solve the corresponding subproblem (4.32) with $\lambda^\ell = \hat{\lambda}^\ell_r$ and add the generated cut to the new master model. In case solving (4.32) for all elements in $\mathcal{A}$ needs a large amount of time, one can always generate cuts from a feasible solution of (4.32). Indeed, this is the way we generate initial cuts in our implementation. Note, if we fix $x = \hat{x}_r^\ell$, where $\hat{x}_r^\ell$ is a first-stage solution form previous phase obtained from subproblem ℓ at iteration r, we end up with an MIP which is easier to solve.

We provide more implementation related details for the numerical experiments in the next section.

4.4 Computational Experiments

In this section we present some numerical experiments to test the performance of the strategies developed in the previous section. We use two data sets: sslp [2] and cap [55]. We label sslp data set ($|N| = 150, 200$) and cap-10 ($|N| = 50, 70, 100$) as class easy; sslp data set ($|N| = 500, 1000$) and cap-11 ($|N| = 50, 70, 100$) as class hard. We summarize the profile of these two data sets in Table 13.

Table 13. Profile of instances (n_1 and n_2 are number of variables in the first-stage and second-stage; m_1 and m_2 are number of constraints in first-stage and second-stage)

Instances	n_1	n_2	m_1	m_2
sslp-5-25	5	130	1	30
sslp-10-50	10	510	1	60
sslp-10-50	15	690	1	60
cap-10	25	1250	1	75
cap-11	50	2500	1	100

4.4.1 Computational Setting

All the experiments were conducted on a Linux workstation with 24 CPU cores (2.2 GHz) and 40 GB of RAM. We use the commercial solver Gurobi 7.0.2 to solve all MIPs. In Gurobi, we set the number of threads to one. All algorithms are implemented in Python 3.4 and we use Numpy [97] to do all the matrix operations and K-mean algorithm implemented in scikit-learn package [69] to form partition of scenarios. In all experiments, we fix the partition size as a fraction of the size of the original scenario set $L = \gamma \times |N|$ and we consider three different values for γ (0.05, 0.2, and 0.3). All three Lagrangian dual problems ((4.5), (4.12), and (4.18)) are solved by Algorithm 5 with the following parameters' setting: $\epsilon = 10^{-4}$, $\kappa = 0.2$, $lb = -\infty$, and $ub = +\infty$. Optimality gap is calculated by $gap = \frac{z^* - z}{|z^*|}$ where z^* is the optimal objective value of (4.1) and z is a lower bound that a specific method generates. In case z^* is not available, we use an objective value corresponding to a feasible solution of (4.1) (which is the case for all instances in class hard). The percentage of gap which is closed by a proposed method is the relative distance between the optimality gap obtained from that method and one obtained by DD method.

4.4.2 Numerical Results

In the first series of experiments, we compare the results of One-phase PDD and DD with respect to the lower bound which each method generates. For class easy, we impose 60 seconds time limit; for class hard, we impose 3600 seconds time limit.

We report the best lower bound (LB) and number of iterations (r) for the class easy in Table 14 and for the class hard in Table 15 ("$*$" means algorithm converges within the time limit). The numbers inside the parenthesis are the amount of optimality gap which One-phase PDD closes compared to DD. As we observe, the bounds reported under One-phase PDD columns always dominate those under DD columns (for sslp-15-45, DD fails to provide a lower bound). The reason that the number of iterations in DD is smaller compared to One-phase PDD ($\gamma = 0.2, 0.3$) is that DD needs to solve more subproblems in each iteration. Hence, its iterations are computationally more expensive. There is no relation among the partitions in the columns of One-phase PDD (i.e., the larger ones are not the refined partition of smaller ones); therefore, we cannot expect the result of Proposition 2 holds. The same behavior is also observable in Table 15. However, there are instances in which the proposed One-phase PDD could not improve the gap (marked by $-$).

Table 14. Numerical results (lower bound and number of iterations) for class easy instances to compare One-phase PDD versus DD.

Instances	$\|N\|$	DD		One-phase PDD ($\gamma = 0.05$)		One-phase PDD ($\gamma = 0.2$)		One-phase PDD ($\gamma = 0.3$)	
		LB	r	LB	r	LB	r	LB	r
Cap-101	50	16868.62	11	17180.30 (94.68)*	8	17181.04 (94.91)	25	16930.20 (18.71)	30
	70	16859.56	8	17193.06 (94.72)*	11	17196.30 (95.64)	31	16925.15 (18.63)	20
	100	16923.19	6	17264.91 (90.81)*	14	16949.08 (6.88)	25	16934.25 (2.94)	14
Cap-102	50	22116.88	11	22340.68 (86.33)*	9	22344.67 (87.87)*	23	22125.80 (3.44)	18
	70	22074.74	8	22373.74 (90.65)*	11	22280.37 (62.34)	32	22074.38 (-)	21
	100	21988.43	6	22286.10 (89.77)*	15	22027.31 (11.73)	21	22002.16 (4.14)	14
Cap-103	50	26615.79	11	26851.50 (74.46)*	7	26872.86 (81.21)*	22	26885.16 (85.09)	32
	70	26595.98	8	26856.74 (74.92)*	10	26881.51 (82.04)	30	26620.83 (7.14)	22
	100	26811.84	6	27116.51 (78.71)*	14	26818.57 (1.74)	21	26845.66 (8.74)	14
Cap-104	50	34003.63	12	34380.18 (94.44)*	8	34380.50 (94.52)*	23	34326.02 (80.85)	31
	70	34151.26	8	34543.96 (92.04)*	10	34416.94 (62.27)	32	34168.16 (3.96)	21
	100	34128.49	6	34514.02 (86.26)*	16	34146.28 (3.98)	44	34151.41 (5.13)	15
sslp-5-25	150	-149.03	23	-139.23 (98.79)*	11	-139.23 (98.85)*	30	-146.57 (24.87)	35
	200	-146.96	18	-137.35 (98.70)*	16	-143.40 (36.52)	30	-144.19 (28.45)	27
sslp-10-50	150	-360.53	3	-351.84 (65.14)*	6	-354.43 (45.70)	7	-355.95 (34.30)	5
	200	-358.82	2	-350.01 (64.64)*	0	-352.71 (44.84)	5	-354.22 (33.76)	4
sslp-15-45	150	-	0	-276.08 (-)	46	-275.36 (-)	4	-273.89 (-)	4
	200	-	0	-275.49 (-)	14	-276.43 (-)	3	-277.35 (-)	2

Table 15. Numerical results (lower bound and number of iterations) for class hard instances to compare One-phase PDD versus DD.

Instances	$\|N\|$	DD		One-phase PDD ($\gamma = 0.05$)		One-phase PDD ($\gamma = 0.2$)		One-phase PDD ($\gamma = 0.3$)	
		LB	r	LB	r	LB	r	LB	r
Cap-111	50	103939.84	217	103980.32 (21.85)	6010	104056.93 (62.42)	727	104070.81 (69.82)	478
	70	104018.53	145	104085.49 (34.76)*	159	104057.16 (20.06)	77	104052.59 (17.68)	212
	100	104089.71	100	104174.98 (41.54)	1533	104217.42 (62.21)	338	104107.76 (8.79)	202
Cap-112	50	181326.32	232	181312.07 (-)*	94	181421.13 (40.23)	1105	181478.12 (64.41)	517
	70	1814.7.92	197	181473.42 (12.82)*	141	181616.18 (64.37)*	274	181619.97 (65.74)	386
	100	181531.72	143	181533.46 (0.63)	48	181623.72 (33.55)	269	181661.60 (47.36)	280
Cap-113	50	234240.46	234	234264.95 (6.65)*	89	234250.05 (2.60)	67	234388.95 (40.31)	477
	70	234169.15	162	234219.19 (13.85)	3440	234353.32 (50.99)	492	234383.76 (59.42)	354
	100	234056.83	125	234162.97 (38.31)	1557	234182.28 (33.46)	249	234223.72 (44.51)	289
Cap-114	50	356578.31	267	356599.95 (7.53)*	87	356698.10 (41.69)	861	356749.02 (59.42)	734
	70	356483.45	233	356551.00 (25.99)	4655	356641.38 (60.76)	554	356659.22 (67.62)	391
	100	356406.75	188	356488.35 (33.49)	2166	356527.24 (49.45)	576	356467.25 (24.83)	249
sslp-5-25	500	-142.36	348	-131.88 (99.97)*	26	-131.88 (99.97)*	89	-131.88 (99.97)*	150
	1000	-139.60	207	-128.85 (99.97)	47	-128.85 (99.97)*	094	-128.85 (99.97)	298
sslp-10-50	500	-363.23	37	-354.20 (63.16)*	29	-353.43 (68.54)*	109	-359.63 (25.21)	101
	1000	-361.38	19	-351.92 (63.03)*	46	-356.74 (-)	75	-357.69 (24.57)	58
sslp-15-45	500	-273.68	24	-272.49 (5.75)*	67	-276.28 (-)	80	-274.44 (-)	53
	1000	-272.61	12	-268.40 (19.92)*	92	-274.66 (-)	46	-273.89 (-)	32

In the second series of experiments, we test the Two-phase PDD approach on class easy to demonstrate the benefit of warm starting the master model with respect to computational time. We impose four hours time limit on each experiment, and we fully disaggregate each group after the phase one (the second phase is GDD). In Table 16, under Two-phase PDD column, we report the total time and phase 2 time (which also include initialization time). Numbers inside the parenthesis are the number of iterations in phase 2. Under GDD column we report computational time when we apply GDD directly to the initial partition. As we observe, for most instances, the convergence time for Two-phase PDD is less than the convergence time in GDD. This is also evident from number of iterations, since the second phase in Two-phase PDD has smaller number of iterations.

So far in this chapter, all reported computational time were based on solving the subproblems sequentially. However, all the experiments in this study could also be done in a parallel computing platform. For this reason, in the rest of the chapter, we report an optimistic parallel time. In other words, the reported time only accounts for the subproblem with the largest computational time (assuming that we only solve subproblems in parallel). In this series of experiments, we report the performance of Two-phase PDD for class hard. During phase one, we record the dual multiplier corresponding to the best lower bound ($\hat{\lambda}_{best}$). At the end of the phase 1, we fully disaggregate each group and warm start the second phase. We then use $\hat{\lambda}_{best}$ to perform only one iteration of GDD. The reason is that, in our computational experiments, we observed the second phase usually ends with a lower bound either same as or very close to one obtained using $\hat{\lambda}_{best}$. In order to have a fair comparison with GDD, we also perform only one iteration of GDD with initial $\hat{\lambda} = 0$. In our experiments, we consider 24 hours time limit, then report the optimistic parallel time. We also set 900 seconds time limit on first phase and 1200 seconds time limit on each subproblem.

The results of this experiment are reported in Table 17. The numbers inside the parenthesis under Two-phase PDD column are the amount of optimality gap that Two-phase PDD closes. We use "-" to indicate that for a specific instance the optimality gap was not improved. As we observe, for $\gamma = 0.05$, Two-phase PDD is closing the optimality gap in almost all instances. For $\gamma = 0.2$ and $\gamma = 0.3$, there are only a few instances in which the relative gap is improved. In Table 18, we report the computational time for GDD and Two-phase PDD. For Two-phase PDD, we report the total time as well as the amount of time phase 2 spends (which also includes the initialization of master model). The results imply that Two-phase PDD needs more computational time compared to GDD. Nevertheless, it improves the optimality gap.

Table 16. Computational time for Two-phase PDD versus GDD

Instances	$\|N\|$	$\gamma = 0.05$			$\gamma = 0.2$			$\gamma = 0.3$		
		Two-phase PDD		GDD	Two-phase PDD		GDD	Two-phase PDD		GDD
		Phase 2	Total		Phase 2	Total		Phase 2	Total	
Cap-101	50	83.38 (2)	85.77	226.03 (8)	103.09 (5)	136.01	410.66 (26)	127.16 (8)	208.52	457.71 (36)
	70	125.05 (2)	129.35	477.05 (11)	175.95 (7)	237.09	590.59 (32)	192.76 (9)	330.73	850.79 (48)
	100	458.37 (11)	467.59	946.79 (27)	379.25 (10)	499.65	1438.91(44)	440.66 (14)	708.59	1953.33(74)
Cap-102	50	79.60 (2)	82.17	354.90 (9)	117.37 (6)	150.20	333.62 (23)	124.56 (8)	200.03	458.71 (37)
	70	141.85 (2)	146.45	645.19 (11)	194.26 (7)	258.93	712.78 (32)	179.11 (7)	314.33	970.79 (50)
	100	331.71 (5)	342.14	973.73 (14)	321.64 (8)	448.77	1987.85(58)	333.27 (10)	602.19	1655.79(62)
Cap-103	50	229.17 (5)	231.24	297.39 (7)	101.43 (5)	131.22	331.04 (22)	85.07 (5)	149.34	417.80 (32)
	70	143.66 (3)	147.99	444.14 (10)	133.75 (0)	195.80	608.38 (30)	99.58 (7)	237.41	751.90 (46)
	100	285.14 (5)	295.54	674.62 (14)	469.91 (13)	601.15	1280.01 (41)	284.39(4)	551.78	1639.24 (60)
Cap-104	50	115.08 (4)	117.22	199.49 (8)	116.57 (7)	148.66	328.13 (24)	116.10 (8)	181.28	390.85 (35)
	70	150.95 (3)	155.42	509.83 (11)	232.10 (8)	304.40	794.26 (38)	158.30 (6)	291.35	777.28 (45)
	100	382.74 (5)	394.16	892.44 (16)	359.96 (9)	536.10	2064.85 (66)	336.49 (9)	772.62	2449.33 (89)
sslp-5-25	150	61.74 (6)	67.01	73.09 (10)	69.51 (7)	117.77	167.12 (30)	58.53 (22)	130.79	204.50 (40)
	200	90.59 (7)	100.97	121.03 (11)	97.62 (12)	177.26	223.74 (34)	153.14 (10)	81.97	329.19 (51)
sslp-10-50	150	14992.60 (4)	15009.55	15330.88 (3)	1684.67 (7)	2088.99	14799.03 (22)	771.63 (9)	4771.28	15286.44 (46)
	200	14698.46 (3)	14729.73	14996.33 (2)	9795.39 (8)	10378.35	14653.92 (20)	1119.01 (10)	2179.32	14771.73 (59)
sslp-15-45	150	15363.37 (5)	15494.75	15444.50 (5)	13744.14 (7)	15609.81	15493.77 (8)	4295.09 (8)	14482.71	14692.32 (19)
	200	15244.93 (2)	15308.65	14580.95 (3)	11571.22 (4)	14671.29	15078.20 (6)	9683.10 (3)	14475.17	14411.76 (23)

Table 17. Comparison of lower bound for Two-phase PDD versus GDD for class hard

Instances	$\|N\|$	$\gamma = 0.05$		$\gamma = 0.2$		$\gamma = 0.3$	
		GDD	Two-phase PDD	GDD	Two-phase PDD	GDD	Two-phase PDD
Cap-111	50	103988.85	103992.58 (2.69)	104025.00	104109.77 (82.77)	104023.30	104037.43 (13.57)
	70	104075.11	104105.74 (22.52)	104115.70	104115.61 (-)	104103.19	104103.54 (0.32)
	100	104189.46	104210.00 (19.46)	104203.05	104202.56 (-)	104168.09	104167.48 (-)
Cap-112	50	181462.60	181467.78 (5.22)	181449.51	181523.20 (65.50)	181416.25	181444.89 (19.37)
	70	181626.98	181651.60 (28.01)	181549.29	181556.91 (4.60)	181534.63	181534.36 (-)
	100	181704.36	181704.66 (0.29)	181638.83	181638.06 (-)	181618.55	181617.07 (-)
Cap-113	50	234439.20	234442.92 (2.20)	234426.22	234425.67 (-)	234389.41	234388.60 (-)
	70	234356.77	234370.31 (7.80)	234303.77	234303.75 (-)	234300.26	234300.59 (0.15)
	100	234249.57	234292.94 (23.80)	234211.27	234211.58 (0.14)	234168.75	234169.11 (0.14)
Cap-114	50	356743.44	356747.57 (6.30)	356657.50	356756.05 (47.35)	356652.70	356743.90 (42.83)
	70	356620.17	356647.22 (21.95)	356579.54	356579.25 (-)	3565647.88	356550.80 (1.49)
	100	356540.45	356542.13 (1.52)	356493.54	356495.13 (1.01)	356479.85	356481.03 (0.69)
sslp-5-25	500	-134.43	-131.87 (100)	-137.30	-131.88 (99.95)	-138.19	-131.87 (99.99)
	1000	-132.13	-128.84 (99.97)	-134.71	-134.70 (0.00)	-135.81	-135.81 (0.00)
sslp-10-50	500	-350.09	-349.34 (64.67)	-351.21	-353.14 (-)	-354.74	-354.74 (0.00)
	1000	-348.37	-347.09 (64.13)	-349.22	-351.07 (-)	-352.65	-352.65 (0.00)
sslp-15-45	500	-258.16	-258.40 (-)	-261.89	-261.89 (0.00)	-264.03	-264.03 (0.00)
	1000	-258.40	-258.40)(0.00)	-261.53	-261.53 (0.00)	-263.34	-263.34 (-)

Table 18. Comoputational time for Two-phase PDD versus GDD for class hard

Instances	$\|N\|$	$\gamma = 0.05$			$\gamma = 0.2$			$\gamma = 0.3$		
		GDD	Two-phase PDD		GDD	Two-phase PDD		GDD	Two-phase PDD	
			Phase 2	Total		Phase 2	Total		Phase 2	Total
Cap-111	50	1200.04	1224.83	1869.79	44.93	90.26	298.88	36.59	328.57	538.05
	70	1200.04	1277.71	1333.69	131.47	1900.54	2912.42	30.40	1677.87	2204.41
	100	1200.04	1250.25	1540.87	152.16	2492.67	2775.08	48.94	104.07	172.70
Cap-112	50	1200.03	1200.03	1217.69	71.85	116.99	344.87	20.13	299.86	612.23
	70	1200.02	1264.21	1304.18	91.86	722.06	1074.18	40.21	1593.39	1939.81
	100	802.57	2091.09	3366.10	60.71	2345.64	3204.82	433.93	517.97	
Cap-113	50	1200.03	1200.02	1217.48	113.44	504.15	1389.83	69.04	1106.51	1367.83
	70	1200.03	1250.24	1563.67	142.32	1520.91	1844.96	59.90	1365.85	1570.03
	100	1200.02	1250.24	1692.24	285.16	1498.71	1748.27	39.77	99.56	172.68
Cap-114	50	1200.03	1223.24	1238.26	34.59	108.29	577.08	31.06	388.31	613.16
	70	1200.03	1200.02	1600.52	115.56	1369.99	2093.11	17.09	1718.23	2028.11
	100	1061.11	2666.83	3566.06	93.66	1203.23	1527.86	39.92	1869.16	1981.77
sslp-5-25	500	7.02	14.55	18.80	0.74	19.72	33.72	0.37	11.65	30.28
	1000	3.87	35.61	42.68	0.93	14.87	28.51	0.88	14.64	27.76
sslp-10-50	500	1200.04	1242.31	1308.88	1200.01	1246.60	1285.96	700.67	127.70	165.86
	1000	1200.02	1325.29	1437.29	1200.43	1288.82	1316.85	1200.00	1287.20	1305.94
sslp-15-45	500	1200.03	1734.42	2065.77	551.48	780.20	842.46	806.90	1004.19	1103.45
	1000	1200.04	1878.57	1947.09	1200.02	1586.25	1606.80	243.74	1835.72	1880.69

4.5 Conclusion

In this chapter we presented formulation of partition-based Lagrangian relaxation for a two-stage SIP with integer recourse (referred to as PDD). Based on this formulation, we suggested two strategies in order to obtain a lower bound for the original SIP. In the first strategy, called One-phase PDD, we only apply PDD once and report the result. This method is very promising (compared to DD), when we have a very tight time limit. In the second strategy, called Two-phase PDD, we disaggregate all the aggregated scenarios within a group, in order to improve the bound. We also presented a method to partially utilize some information from the first phase, since the cuts could not be transferred between phases. We conducted some computational strategies to evaluate the performance of the proposed methods. One-phase PDD exhibited good results (in terms of improvement on lower bound) in both class easy and hard instances. For Two-phase PDD, we empirically verified that the proposed method could improve the computational time for class easy. For class hard, Two-phase PDD was successful in improving the lower bound when we have a small partition with large groups, but failed for larger partitions. A very natural extension of this work is to conduct a series of experiments on a parallel computing platforms, since we only report a possible optimistic parallel time.

APPENDIX

This Appendix contains some computational results for models developed in Chapter 2.

In Table 19, we present full computational results for three approaches REU, FPM and PPM on our test instances with different interdiction budget ratios (0.4, 0.5, and 0.6). In Table 20, we present computational results for two equivalent formulations for stochastic shortest path interdiction with a robust expected utility model, (2.14) (labeled as REU throughout the paper), and a reformulation of (2.14) by replacing (2.14b) with (2.16) and (2.17), labeled as REU-R. In Table 21, we compare the computational time taken by approaches REU and PPM using instances with various number of pairwise comparisons and interdiction budgets. In Table 22, we report the detailed time profile for the three phases involved in approach PPM: computing the upper bound $\bar{u}_j$ and lower bound $\underline{u}_j$ for $j \in T$, fitting the piecewise linear concave function with a fixed number of pieces, and optimizing the expected utility using this fitted function.

Table 19. Full computational results for three approaches REU, FPM and PPM on instances with different interdiction budget ratios (0.4, 0.5, and 0.6).

			Time			# Nodes			RootGap (%)		
K	Instance	Budget	REU	FPM	PPM	REU	FPM	PPM	REU	FPM	PPM
100	3 × 3	5	1.31	9.77	7.65	13	10	10	3.45	0.69	0.48
		6	2.77	10.85	7.44	38	30	23	2.42	0.94	0.64
		7	2.77	11.12	7.28	37	35	22	2.03	0.77	0.45
	4 × 4	10	4.01	25.71	16.24	248	203	187	2.52	0.75	0.71
		12	2.14	21.29	16.17	35	18	15	0.99	0.33	0.18
		14	2.57	22.58	19.09	92	86	62	1.84	0.62	0.35
	5 × 5	16	9.51	56.22	21.79	838	900	671	2.22	0.82	0.67
		20	21.79	69.97	27.17	1920	1671	1690	1.73	0.68	0.47
		24	9.87	49.41	26.19	710	805	537	1.12	0.66	0.37
	6 × 6	24	223.04	294.37	33.81	13369	5187	3652	2.62	0.93	0.79
		30	94.95	156.11	33.18	7385	1605	1988	1.60	0.71	0.77
		36	197.45	223.50	38.97	24172	3574	3772	1.80	0.98	0.87
	7 × 7	34	1.01% (0)	2734.33	126.45	1637	33129	38290	1.62	1.36	1.16
		42	1.16% (0)	2241.75	114.96	1017	33183	3313	1.53	1.10	1.07
		50	0.69% (0)	388.47	51.00	1095	5229	8788	1.09	1.04	0.9
500	3 × 3	5	15.54	37.85	22.31	13	11	13	4.83	0.87	0.87
		6	18.40	44.15	22.28	36	30	34	3.72	1.10	1.34
		7	22.85	47.22	22.44	46	40	39	3.21	0.73	0.77
	4 × 4	10	72.60	102.90	33.82	385	331	361	2.87	1.12	1.09
		12	23.72	54.59	32.16	41	28	26.2	0.75	0.33	0.31
		14	27.69	58.02	32.41	127	106	89	1.86	0.39	0.42
	5 × 5	16	91.62	246.15	53.73	1438	1206	1270	2.00	0.91	1.22
		20	158.86	548.68	71.25	3225	3049	3119	1.00	0.59	0.60
		24	140.02	436.59	67.32	3433	2825	3060	1.00	0.61	0.59
	6 × 6	24	620.18	1062.27	117.10	6154	4086	3909	1.00	0.63	0.66
		30	545.03	674.66	89.77	9566	2074	2296	1.00	0.60	00.65
		36	2104.30	2061.56	260.66	47361	8199	10883	1.80	0.76	0.77
	7 × 7	34	0.54% (1)	0.23% (2)	1645.34	47268	8967	61245	3.50	1.19	1.66
		42	0.54% (0)	0.23% (0)	2495.81	49542	8103	110563	2.21	1.14	1.37
		50	0.23% (1)	0.11% (3)	438.44	58127	8346	20899	1.94	1.02	0.99
1000	3 × 3	5	53.73	43.53	21.48	16	10	14	4.39	0.82	0.84
		6	68.95	52.4	21.75	41	32	33	4.08	0.91	0.91
		7	82.98	60.47	22.29	48	44	47	3.43	0.90	0.90
	4 × 4	10	194.87	203.76	49.21	456	379	403	3.27	1.01	1.06
		12	57.88	85.16	39.68	48	29	30	1.23	0.40	0.43
		14	67.52	107.02	40.07	122	122	120	1.85	0.72	0.67
	5 × 5	16	675.91	512.68	71.14	1351	1163	1289	2.00	0.87	0.89
		20	920.89	1257.17	133.16	3228	3331	3426	2.00	0.64	0.58
		24	932.94	1352.48	134.39	4027	3386	3379	2.00	0.60	0.62
	6 × 6	24	1747.63	2171.98	257	5675	3274	4135	2.49	0.89	0.86
		30	1178.01	1209.10	153	4811	2241	1892	1.36	0.64	0.61
		36	0.43% (0)	0.15% (1)	1023	24519	7867	17488	2.00	0.88	0.83
	7 × 7	34	0.8% (0)	0.25% (0)	0.15% (0)	15669	7610	44732	2.37	1.02	1.17
		42	0.77% (0)	0.28% (0)	0.19% (0)	16978	6388	38646	2.87	1.07	1.33
		50	0.33% (0)	0.11% (3)	792.01	24243	6151	11509	1.46	0.73	0.84

Table 20. Full computational results for two formulation REU and REU-R on instances with different interdiction budget ratios (0.4, 0.5, and 0.6).

			Time		# Nodes		RootGap (%)	
K	Instance	Budget	REU	REU-R	REU	REU-R	REU	REU-R
100	3×3	5	1.43	1.36	14	11	4.06	1.29
		6	1.76	1.47	33	21	2.48	1.67
		7	1.93	1.58	43	32	2.17	1.38
	4×4	10	4.15	4.21	242	129	2.22	1.36
		12	2.28	2.61	39	8	0.83	0.18
		14	2.52	2.69	84	34	1.80	0.49
	5×5	16	8.00	10.10	891	386	2.14	1.00
		20	19.51	15.80	1922	744	1.77	0.88
		24	10.56	8.77	1056	333	1.15	0.62
	6×6	24	223.04	153.54	13369	3655	2.62	1.28
		30	94.95	70.33	7385	2025	1.60	0.85
		36	197.45	122.01	24172	2932	1.80	0.91
	7×7	34	1.01 % (0)	0.10 % (4)	1637	23506	1.62	1.26
		42	1.16 % (0)	1759.88	1017	21484	1.53	1.01
		50	0.96 % (0)	484.03	1095	6058	1.09	0.80
500	3×3	5	15.54	38.68	13	15	4.83	1.80
		6	18.40	44.33	36	27	3.72	2.01
		7	22.85	54.02	46	36	3.21	1.65
	4×4	10	72.60	88.84	385	240	2.87	1.92
		12	23.72	36.36	41	15	0.75	0.23
		14	27.69	39.98	127	37	1.86	0.50
	5×5	16	91.62	190.78	1438	451	2.00	0.91
		20	158.86	346.30	3225	1118	1.00	0.86
		24	140.02	324.82	3433	1146	1.00	0.84
	6×6	24	620.18	730.34	6154	1061	1.00	0.84
		30	545.03	1006.18	9566	1631	1.00	0.74
		36	2104.30	0.33 %(0)	47361	3410	1.80	0.96
	7×7	34	0.54 % (1)	1.38 % (0)	47268	2169	3.50	1.28
		42	0.54 % (0)	1.20 % (0)	49542	2019	2.1	1.20
		50	0.23 % (1)	0.89 % (0)	58127	1653	1.94	0.87
1000	3×3	5	53.73	248.00	16	15	4.39	1.66
		6	68.95	312.31	41	27	4.08	1.84
		7	82.98	300.53	48	42	3.43	1.71
	4×4	10	194.87	484.99	456	318	3.27	2.26
		12	57.88	208.90	48	21	1.23	0.32
		14	67.52	242.36	122	50.6	1.85	0.61
	5×5	16	675.91	975.11	1351	473	2.00	0.93
		20	920.89	1806.60	3228	1163	2.00	0.91
		24	932.94	1790.63	4027	1380	2.00	0.91
	6×6	24	1747.63	3186.50	5675	1104	2.49	0.95
		30	1178.01	2875.34	4811	1087	1.36	0.73
		36	0.43 % (0)	0.87 % (0)	24519	600	2.00	1.13
	7×7	34	0.8 % (0)	1.15 % (0)	15669	510	2.37	1.37
		42	0.77 % (0)	1.10 % (0)	16978	510	2.87	1.26
		50	0.33 % (0)	0.77 % (0)	24243	510	1.46	0.94

Table 21. Computational time (in seconds) for two approaches, REU and PPM, with 20, 50, and 100 pairwise gamble comparisons, using different interdiction budget ratios (0.4, 0.5, and 0.6).

			REU			PPM		
K	Instance	Budget	20	50	100	20	50	100
100	3×3	5	1.31	1.65	2.16	7.65	11.77	15.20
		6	2.77	1.77	2.45	7.44	12.10	15.30
		7	2.77	1.70	2.35	7.28	11.74	15.92
	4×4	10	4.31	6.28	7.06	16.24	24.85	25.65
		12	2.14	3.85	4.04	16.17	24.94	25.85
		14	2.57	4.46	4.38	19.09	24.84	25.62
	5×5	16	9.51	26.92	28.93	21.79	33.03	45.20
		20	21.79	46.69	48.76	27.17	33.74	45.78
		24	9.87	23.40	25.04	26.19	32.87	45.35
500	3×3	5	15.54	18.73	35.90	22.31	21.03	22.34
		6	18.40	19.04	39.67	22.28	21.02	22.53
		7	22.85	20.18	45.43	22.44	20.91	22.81
	4×4	10	72.6	104.03	113.48	33.82	35.40	38.57
		12	23.72	36.90	45.03	32.16	33.69	36.96
		14	27.69	41.15	51.86	32.41	34.16	37.11
	5×5	16	91.62	151.70	160.11	53.73	51.63	55.52
		20	158.86	226.61	271.28	71.25	63.65	68.28
		24	140.02	232.90	263.68	67.32	62.75	69.75
1000	3×3	5	53.73	65.35	73.57	21.48	31.39	26.20
		6	68.95	72.97	92.12	21.75	31.30	26.37
		7	82.98	89.99	118.73	22.29	28.56	26.91
	4×4	10	194.78	264.14	788.39	49.21	54.97	60.63
		12	57.88	95.28	651.47	39.68	51.14	46.87
		14	67.52	105.05	641.31	40.07	55.11	48.82
	5×5	16	675.91	651.75	619.03	71.14	85.70	89.75
		20	920.89	1332.38	1419.86	133.16	140.29	145.80
		24	932.94	1600.88	1573.08	134.39	158.12	154.57

Table 22. Detailed time profile (in seconds) for the three phases involved in approach PPM: computing the upper bound $\bar{u}_j$ and lower bound $\underline{u}_j$ for $j \in T$ (Bound), fitting the piecewise linear concave function with a fixed number of pieces (Fitting), and optimize the expected utility using this fitted function (Optimization).

K	Instance	Budget	Optimization	Bound	Fitting
100	3×3	5	0.25	7.09	0.32
		6	0.31	6.74	0.39
		7	0.27	6.65	0.37
	4×4	10	0.54	14.53	1.17
		12	0.37	14.59	1.21
		14	0.44	14.48	1.18
	5×5	16	1.04	19.34	1.41
		20	2.77	22.64	1.75
		24	1.18	22.96	2.05
	6×6	24	7.56	24.08	2.08
		30	5.84	24.98	2.36
		36	9.67	26.69	2.62
	7×7	34	93.61	30.81	2.03
		42	82.50	30.46	2.00
		50	18.26	30.75	2.00
500	3×3	5	1.31	12.02	8.98
		6	1.48	11.86	8.94
		7	1.63	11.87	8.95
	4×4	10	5.66	19.41	8.75
		12	2.83	20.89	8.44
		14	2.84	20.70	8.87
	5×5	16	11.98	24.47	17.28
		20	29.29	24.60	17.36
		24	25.56	24.34	17.42
	6×6	24	83.92	27.03	6.15
		30	55.20	27.89	6.67
		36	223.45	30.87	6.34
	7×7	34	1611.34	32.19	1.81
		42	2461.81	32.22	1.79
		50	404.85	32.07	1.79
1000	3×3	5	2.43	14.07	4.98
		6	2.81	13.91	5.03
		7	3.36	13.94	5.00
	4×4	10	15.65	24.39	9.17
		12	5.55	25.22	8.90
		14	6.91	24.17	9.00
	5×5	16	34.31	26.85	9.98
		20	94.67	28.73	9.77
		24	11969.58	27.37	10.44
	6×6	24	220.82	32.98	2.80
		30	117.04	33.33	2.86
		36	990.4	30.28	2.92
	7×7	34	3600	35.19	2.7
		42	3600	34.68	2.68
		50	754.13	35.18	2.7

Bibliography

[1] Shabbir Ahmed. Two-stage stochastic integer programming: A brief introduction. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[2] Shabbir Ahmed, Renan Garcia, Nan Kong, Lewis Ntaimo, Gyana R Parija, and Feng Qiu. A stochastic integer programming test problem library, 2015. `http://www.isye.gatech.edu/~sahmed/siplib`.

[3] Víctor M Albornoz, Pablo Benario, and Manuel E Rojas. A two-stage stochastic integer programming model for a thermal power system expansion. *International Transactions in Operational Research*, 11(3):243–257, 2004.

[4] Benjamin Armbruster and Erick Delage. Decision making under uncertainty when preference information is incomplete. *Management Science*, 61(1):111–128, 2015.

[5] Robert J Aumann. Utility theory without the completeness axiom. *Econometrica*, 30(3):445–462, 1962.

[6] Robert J Aumann. Utility theory without the completeness axiom. *Econometrica: Journal of the Econometric Society*, pages 445–462, 1962.

[7] Egon Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1):3–44, 1998.

[8] Halil Bayrak and Matthew D Bailey M.D. Shortest path network interdiction with asymmetric information. *Networks*, 52(3):133–140, 2008.

[9] Jacques F Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

[10] John R Birge and Françoise Louveaux. *Introduction to Stochastic Programming*. Springer, 2011.

[11] Merve Bodur and James R Luedtke. Mixed-integer rounding enhanced benders decomposition for multiclass service-system staffing and scheduling with arrival rate uncertainty. *Management Science*, 2016.

[12] Merve Bodur, Sanjeeb Dash, Oktay Günlük, and James Luedtke. Strengthened benders cuts for stochastic integer programs with continuous recourse. *INFORMS Journal on Computing*, 29(1):77–91, 2016.

[13] Natashia Boland, Ilke Bakir, Brian Dandurand, and Alan Erera. Scenario set partition dual bounds for multistage stochastic programming: A hierarchy of bounds and a partition sampling approach. *Optimization Online*, 2016.

[14] Joseph-Frédéric Bonnans, Jean Charles Gilbert, Claude Lemaréchal, and Claudia Sagastizábal. *Numerical Optimization: Theoretical and Practical Aspects*. Universitext, Springer-Verlag, Berlin, 2006.

[15] Juan S Borrero, Oleg A Prokopyev, and Denis Sauré. Sequential shortest path interdiction with incomplete information. *Decision Analysis*, 13(1):68–98, 2015.

[16] Gerald G Brown and Louis Anthony Cox. How probabilistic risk assessment can mislead terrorism risk analysts. *Risk Analysis*, 31(2):196–204, 2011.

[17] Claus C CarøE and Rüdiger Schultz. Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24(1):37–45, 1999.

[18] Robert T Clemen and Terence Reilly. *Making hard decisions with DecisionTools*. Cengage Learning, 2013.

[19] Ricardo A Collado and Dávid Papp. Network interdiction–models, applications, unexplored directions. Technical report, RUTCOR Technical Report RRR 4-2012, Rutgers Center for Operations Research, Rutgers University, 640 Bartholomew Road Piscataway, New Jersey, 2012.

[20] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer programming*, volume 271. Springer, 2014.

[21] Ivan Contreras, Jean-François Cordeau, and Gilbert Laporte. Benders decomposition for large-scale uncapacitated hub location. *Operations research*, 59(6): 1477–1490, 2011.

[22] Kelly James Cormican. *Computational methods for deterministic and stochastic network interdiction problems*. PhD thesis, Naval Postgraduate School, Monterey, California, 1995.

[23] Kelly James Cormican, David P Morton, and R.Kevin Wood. Stochastic network interdiction. *Operations Research*, 46(2):184–197, 1998.

[24] Welington de Oliveira and Claudia Sagastizábal. Level bundle methods for oracles with on-demand accuracy. *Optimization Methods and Software*, 29(6): 1180–1209, 2014.

[25] Welington de Oliveira, Claudia Sagastizábal, and Susana Scheimberg. Inexact bundle methods for two-stage stochastic programming. *SIAM Journal on Optimization*, 21(2):517–544, 2011.

[26] Yan Deng, Shabbir Ahmed, Jon Lee, and Siqian Shen. Scenario grouping and decomposition algorithms for chance-constrained programs.

[27] D. Ellsberg. Risk, ambiguity, and the savage axioms. *Quarterly Journal of Economics*, 75(4):643–669, 1961.

[28] Sebastian Engell, Andreas Märkert, Guido Sand, and Rüdiger Schultz. Aggregated scheduling of a multiproduct batch plant by two-stage stochastic integer programming. *Optimization and Engineering*, 5(3):335–359, 2004.

[29] Laureano F Escudero, M Araceli Garín, Gloria Pérez, and Aitziber Unzueta. Scenario cluster decomposition of the lagrangian dual in two-stage stochastic mixed 0–1 optimization. *Computers & Operations Research*, 40(1):362–377, 2013.

[30] Laureano F Escudero, María Araceli Garín, and Aitziber Unzueta. Cluster lagrangean decomposition in multistage stochastic optimization. *Computers & Operations Research*, 67:48–62, 2016.

[31] Emmanuel Fragniere, Jacek Gondzio, and Jean-Philippe Vial. Building and solving large-scale stochastic programs on an affordable distributed computing system. *Annals of Operations Research*, 99(1-4):167–187, 2000.

[32] Ridvan Gedik, Hugh Medal, Chase Rainwater, Ed A Pohl, and Scott J Mason. Vulnerability assessment and re-routing of freight trains under disruptions: A coal supply chain network application. *Transportation Research Part E: Logistics and Transportation Review*, 71:45–57, 2014.

[33] P.M. Ghare, D.C. Montgomery, and W.C. Turner. Optimal interdiction policy for a flow network. *Naval Research Logistics Quarterly*, 18(1):37–45, 1971.

[34] Itzhak Gilboa and David Schmeidler. Maxmin expected utility with non-unique prior. *Journal of Mathematical Economics*, 18:141–153, 1989.

[35] Vikas Goel and Ignacio E Grossmann. A class of stochastic programs with decision dependent uncertainty. *Mathematical programming*, 108(2):355–394, 2006.

[36] Qingyu Guo, Bo An, Yair Zick, and Chunyan Miao. Optimal interdiction of illegal network flow. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, 2016.

[37] Aric Hagberg, Daniel Schult, and Pieter Swart. Networkx, 2013. `http://networkx.github.io/index.html`.

[38] Willem K Klein Haneveld and Maarten H van der Vlerk. Optimizing electricity

distribution using two-stage integer recourse models. In *Stochastic Optimization: Algorithms and Applications*, pages 137–154. Springer, 2001.

[39] T.E. Harris and F.S. Ross. Fundamentals of a method for evaluating rail network capacities. *Research Memorandum RM-1573*, 1955.

[40] Harald Held, Raymond Hemmecke, and David L Woodruff. A decomposition algorithm applied to planning the interdiction of stochastic networks. *Naval Research Logistics (NRL)*, 52(4):321–328, 2005.

[41] R. Hemmecke, R. Schultz, and D.L. Woodruff. Interdicting stochastic networks with binary interdiction effort. In D.L. Woodruff, editor, *Network interdiction and stochastic integer programming*, pages 69–84. Kluwer Academic Publishers, Boston, MA, 2003.

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

[43] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex analysis and minimization algorithms I: Fundamentals*, volume 305. Springer science & business media, 2013.

[44] Jian Hu and Sanjay Mehrotra. Robust decision making over a set of random targets or risk averse utilities with an application to portfolio optimization. *IIE Transactions*, 47(4):358–372, 2015.

[45] Eitan Israeli and R.Kevin Wood. Shortest-path network interdiction. *Networks*, 40(2):97–111, 2002.

[46] Kibaek Kim and Sanjay Mehrotra. A two-stage stochastic integer programming approach to integrated staffing and scheduling with application to nurse management. *Operations Research*, 63(6):1431–1451, 2015.

[47] Frank H Knight. *Risk, uncertainty, and profit.* Hart, Schaffner & Marx, 1921.

[48] Renata A Konrad, Andrew C Trapp, Timothy M Palmbach, and Jeffrey S Blom. Overcoming human trafficking via operations research and analytics: Opportunities for methods, models, and applications. *European Journal of Operational Research*, 259(2):733–745, 2017.

[49] Gilbert Laporte and François V Louveaux. The integer l-shaped method for stochastic integer programs with complete recourse. *Operations research letters*, 13(3):133–142, 1993.

[50] Gilbert Laporte, Francois Louveaux, and Hélène Mercure. The vehicle routing problem with stochastic travel times. *Transportation science*, 26(3):161–170, 1992.

[51] Jesús M Latorre, Santiago Cerisola, Andrés Ramos, and Rafael Palacios. Analysis of stochastic problem decomposition algorithms in computational grids. *Annals of Operations Research*, 166(1):355–373, 2009.

[52] Xiao Lei, Siqian Shen, and Yongjia Song. Stochastic maximum flow interdiction problems under heterogeneous risk preferences. *Submitted for publication*, 2016.

[53] Claude Lemaréchal, Arkadii Nemirovskii, and Yurii Nesterov. New variants of bundle methods. *Mathematical programming*, 69(1-3):111–147, 1995.

[54] Jeffrey Linderoth, Alexander Shapiro, and Stephen Wright. The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research*, 142:215–241, 2006.

[55] François V Louveaux. Discrete stochastic location models. *Annals of Operations research*, 6(2):21–34, 1986.

[56] Miles Lubin, Kipp Martin, Cosmin G Petra, and Burhaneddin Sandıkçı. On parallelizing dual decomposition in stochastic integer programming. *Operations Research Letters*, 41(3):252–258, 2013.

[57] Ajay Malaviya, Chase Rainwater, and Thomas Sharkey. Multi-period network interdiction problems with applications to city-level drug enforcement. *IIE Transactions*, 44(5):368–380, 2012.

[58] Alan W McMasters and Thomas M Mustin. Optimal interdiction of a supply network. *Naval Research Logistics Quarterly*, 17(3):261–268, 1970.

[59] David P Morton. Stochastic network interdiction. *Wiley Encyclopedia of Operations Research and Management Science*, 2011.

[60] David P Morton and Feng Pan. Using sensors to interdict nuclear material smuggling. In *Proceedings of the IIE Research Conference*, 2005.

[61] David P Morton, Feng Pan, and Kevin J Saeger. Models for nuclear smuggling interdiction. *IIE Transactions*, 39(1):3–14, 2007.

[62] Ishwar Murthy and Sumit Sarkar. Stochastic shortest path problems with piecewise-linear concave utility functions. *Management Science*, 44(11-part-2): 125–136, 1998.

[63] Apurba K Nandi, Hugh R Medal, and Satish Vadlamani. Interdicting attack graphs to protect organizations from cyber attacks: A bi-level defender–attacker model. *Computers & Operations Research*, 75:118–131, 2016.

[64] Robert Nau. The shape of incomplete preferences. *The Annals of Statistics*, 34(5):2430–2448, 2006.

[65] George L Nemhauser and Laurence A Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.

[66] Matthias P Nowak, Rüdiger Schultz, and Markus Westphalen. A stochastic integer programming model for incorporating day-ahead trading of electricity into hydro-thermal unit commitment. *Optimization and Engineering*, 6(2): 163–176, 2005.

[67] Michalopoulos Dennis P, David P Morton, and J. Wesley Barnes. Prioritizing network interdiction of nuclear smuggling. *Stochastic Programming: Applications in Finance, Energy and Logistics, Operations Research/Computer Science Interfaces Series. World Scientific, London*, 2013.

[68] F. Pan, William S Charlton, and David P Morton. A stochastic program for

interdicting smuggled nuclear material. In D.L. Woodruff, editor, *Network interdiction and stochastic integer programming*, pages 1–19. Kluwer Academic Publishers, Boston, MA, 2003.

[69] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[70] C.A. Phillips. The network destruction problem. Technical report, Sandia National Labs., Albuquerque, NM (United States), 1992.

[71] R Tyrrell Rockafellar and Roger J-B Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of operations research*, 16(1): 119–147, 1991.

[72] Andrzej Ruszczyński. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical programming*, 35(3):309–333, 1986.

[73] Andrzej Ruszczyński and Artur Świetanowski. Accelerating the regularized decomposition method for two stage stochastic linear problems. *European Journal of Operational Research*, 101(2):328–342, 1997.

[74] Kevin Ryan, Shabbir Ahmed, Santanu S Dey, and Deepak Rajan. Optimization driven scenario grouping. 2016.

[75] Georgios KD Saharidis, Michel Minoux, and Marianthi G Ierapetritou. Accelerating benders method using covering cut bundle generation. *International Transactions in Operational Research*, 17(2):221–237, 2010.

[76] Javier Salmeron, Kevin Wood, and Ross Baldick. Analysis of electric grid security under terrorist threat. *IEEE Transactions on power systems*, 19(2): 905–912, 2004.

[77] Javier Salmeron, Kevin Wood, and Ross Baldick. Worst-case interdiction analysis of large-scale electric power grids. *IEEE Transactions on power systems*, 24(1):96–104, 2009.

[78] Burhaneddin Sandıkçı, Nan Kong, and Andrew J Schaefer. A hierarchy of bounds for stochastic mixed-integer programs. *Mathematical Programming*, 138(1-2):253–272, 2013.

[79] Burhaneddin Sandikci and Osman Y zaltın. A scalable bounding method for multi-stage stochastic integer programs. 2014.

[80] Anibal Sanjab, Walid Saad, and Tamer Başar. Prospect theory for enhanced cyber-physical security of drone delivery systems: A network interdiction game. *arXiv preprint arXiv:1702.04240*, 2017.

[81] Hassan Sarhadi, David M Tulett, and Manish Verma. An analytical approach to the protection planning of a rail intermodal terminal network. *European Journal of Operational Research*, 257(2):511–525, 2017.

[82] Suvrajeet Sen. Subgradient decomposition and differentiability of the recourse function of a two stage stochastic linear program. *Operations Research Letters*, 13(3):143–148, 1993.

[83] Suvrajeet Sen. Algorithms for stochastic mixed-integer programming models. *Handbooks in operations research and management science*, 12:515–558, 2005.

[84] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczyński. *Lectures on Stochastic Programming: Modeling and Theory.* SIAM, Philadelphia, 2009.

[85] J.C. Smith, M. Prince, and J. Geunes. Modern network interdiction problems and algorithms. In *Handbook of Combinatorial Optimization*, pages 1949–1987. Springer, 2013.

[86] Yongjia Song and James Luedtke. An adaptive partition-based approach for solving two-stage stochastic programs with fixed recourse. *SIAM Journal on Optimization*, 25(3):1344–1367, 2015.

[87] Yongjia Song and Siqian Shen. Risk-averse shortest path interdiction. *INFORMS Journal on Computing*, 28(3):527–539, 2016.

[88] Stefano Starita, M Paola Scaparra, and Jesse R OHanley. A dynamic model for road protection against flooding. *Journal of the Operational Research Society*, 68(1):74–88, 2017.

[89] Roert L Steinrauf. Network interdiction models. Master's thesis, Naval Postgraduate School, Monterey, CA, 1991.

[90] K.M. Sullivan, D.P. Morton, F. Pan, and J.C. Smith. Securing a border under asymmetric information. *Naval Research Logistics*, 61(2):91–100, 2014.

[91] Svyatoslav Trukhanov, Lewis Ntaimo, and Andrew Schaefer. Adaptive multicut aggregation for two-stage stochastic linear programs with recourse. *European Journal of Operational Research*, 206(2):395–406, 2010.

[92] Wim van Ackooij, Welington de Oliveira, and Yongjia Song. An adaptive partition-based level decomposition for solving two-stage stochastic programs with fixed recourse. 2016. Submitted for publication.

[93] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

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

[95] John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton university press, 2007.

[96] Peter P Wakker. *Prospect theory: For risk and ambiguity*. Cambridge university press, 2010.

[97] Stfan van der Walt, S. Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[98] Christian Wolf and Achim Koberstein. Dynamic sequencing and cut consolidation for the parallel hybrid-cut nested l-shaped method. *European Journal of Operational Research*, 230(1):143–156, 2013.

[99] Christian Wolf, Csaba I Fábián, Achim Koberstein, and Leena Suhl. Applying oracles of on-demand accuracy in two-stage stochastic programming–a computational study. *European Journal of Operational Research*, 239(2):437–448, 2014.

[100] R Kevin Wood. Deterministic network interdiction. *Mathematical and Computer Modelling*, 17(2):1–18, 1993.

[101] Fengqi You and Ignacio E Grossmann. Multicut benders decomposition algorithm for process supply chain planning under uncertainty. *Annals of Operations Research*, 210(1):191–211, 2013.

[102] Xiaomei Zhu and Hanif D Sherali. Two-stage workforce planning under demand fluctuations and uncertainty. *Journal of the Operational Research Society*, 60 (1):94–103, 2009.

[103] Victor Zverovich, Csaba I Fábián, Eldon FD Ellison, and Gautam Mitra. A computational study of a solver system for processing two-stage stochastic lps with enhanced benders decomposition. *Mathematical Programming Computation*, 4(3):211–238, 2012.