8-2019

Network Interdiction under Uncertainty

Timothy Holzmann
Clemson University, tim.holzmann@gmail.com

Network Interdiction under Uncertainty

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Industrial Engineering

---

by
Timothy W. Holzmann
August 2019

Abstract

We consider variants to one of the most common network interdiction formulations: the shortest path interdiction problem. This problem involves leader and a follower playing a zero-sum game over a directed network. The leader interdicts a set of arcs, and arc costs increase each time they are interdicted. The follower observes the leader's actions and selects a shortest path in response. The leader's optimal interdiction strategy maximizes the follower's minimum-cost path.

Our first variant allows the follower to improve the network after the interdiction by lowering the costs of some arcs, and the leader is uncertain regarding the follower's cardinality budget restricting the arc improvements. We propose a multiobjective approach for this problem, with each objective corresponding to a different possible improvement budget value. To this end, we also present the modified augmented weighted Tchebychev norm, which can be used to generate a complete efficient set of solutions to a discrete multi-objective optimization problem, and which tends to scale better than competing methods as the number of objectives grows.

In our second variant, the leader selects a policy of randomized interdiction actions, and the follower uses the probability of where interdictions are deployed on the network to select a path having the minimum expected cost. We show that this continuous non-convex problem becomes strongly NP-hard when the cost functions are convex or when they are concave. After formally describing each variant, we present various algorithms for solving them, and we examine the efficacy of all our algorithms on test beds of randomly generated instances.

Dedication

My loving wife, Danielle, is both my rock and my flower: you have been a strong foundation to carry our growing family through a very stressful time, and you always brought a sweet fragrance of love and joy when I was overwhelmed. Thank you for carrying us through . . . We did it!

My first daughter, Mackenzie, is my pride: a blossoming woman of God. You see a world full of flowers, rainbows, and righteousness (and yes, maybe even unicorns)! But with the eyes in your heart that see such goodness; God has also given you strength in your soul to face dark truths and not be overcome. Instead, you shine his kingdom light into the darkness and confound it.

My second daughter, Hannah, is my heart: never short of emotion. Your feelings are so big that they infect everyone; we laugh when you laugh and weep when you weep. You have struggled more than others, but then God just pours more mercy into your heart, and it overflows so that your mouth speaks a gospel of grace. That's when I know that there is no one like our God!

My third daughter, Amelia, is my warmth: full of soft hugs and tender kisses. Your sweet smile has melted me more times than I can count. I feel myself whirling around with so many things to do and think about, and then I turn to see you standing there with your peaceful smile. Your patient compassion will touch many lives with the blessings of heaven.

My son, Titus, is my strength: you never stop trying. Your persistence reminds me to keep going when I think I have nothing more to give. As you relentlessly chased me around the dining room, I pray you never tire of running after Jesus – even when you're going in circles. Renew your hope, and you will find that really he's chasing you and now lifts you up on eagles' wings.

My fourth daughter, Katriel, is my peace: thank you for reminding me that I only need to love those around me. When I cradle you sleeping in my arms, I know that God, by his grace, has given me something better than any academic accolade. This dissertation is a great achievement, but my children are a greater one, and you are the crown God has placed on our family!

Acknowledgments

Praise God, from whom all blessings flow! I have been blessed abundantly in this work by the many friends and family around me who have aided me in this research.

I first wish to thank my advisor, Dr. Cole Smith: Working with you is both an honor and humbling experience. I am amazed at your pulchritude in explaining many IE concepts such as LP, Benders' cuts, bi-level programming, RLT, McCormick constraints, etc. Thank you for teaching me (using words only some of the time) what it means to be a professor and researcher. Thank you also to my committee members for your support, critiques, and insights. You have shown me a glimpse into a much larger world of research.

I am also indebted to countless others: faculty, staff, administrators, the university food workers, the janitorial staff of Freeman Hall, the computer network engineers, the librarians, and many others – I am amazed at how many people have come together to make enrich this educational experience. Each one has touched my time here, and I thank you for your dedicated service.

More personally, my parents and in-laws provided encouragement throughout these years, and the members of Eternal Shepherd Lutheran Church enfolded our family into their community and loved us. But what lifts me from the deepest valleys, inspires me to the loftiest hopes, and carries me through marathon periods of no apparent progress are my wife and children, to whom this dissertation is dedicated.

Finally, to acknowledge my research sponsors: this work was supported by the Air Force Office of Scientific Research [grant number FA9550-12-1-0353]; and the Office of Naval Research [grant numbers N00014-13-1-0036, N00014-17-1-2421]. I am also grateful to Artelys for providing an HPC license of their software for the study in Chapter 5. The views expressed in this dissertation are those of the author and do not necessarily reflect the official policy or position of the Air Force, the Department of Defense or the U.S. Government.

Table of Contents

Title Page . . . . **i**

Abstract . . . . **ii**

Dedication . . . . **iii**

Acknowledgments . . . . **iv**

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

List of Figures . . . . **viii**

1 Introduction **1**
- 1.1 Background and Literature Review 1
- 1.2 Contribution and Motivation 4
- 1.3 Dissertation Organization 6

2 Modified Augmented Weighted Tchebychev Scalarizations **7**
- 2.1 Chapter Introduction 7
- 2.2 Algorithm 13
- 2.3 Algorithm Proof 16
- 2.4 Example 19
- 2.5 Computational Study 22
- 2.6 Conclusion 30

3 The Shortest Path Interdiction Problem with Arc Improvement Recourse . . . **31**
- 3.1 Problem description 31
- 3.2 Transforming the SPIP-I into a SPIP instance 35
- 3.3 Recursion algorithm for series-parallel graphs 37
- 3.4 Improved recursive algorithm 46
- 3.5 Computational study 51
- 3.6 Conclusion 59

4 The shortest Path Interdiction Problem with Randomized Strategies **60**
- 4.1 Problem Statement and Background 60
- 4.2 The Convex Case 64
- 4.3 The Concave Case 71
- 4.4 Sample Average Approximation for General SPIP-RS 78
- 4.5 Computational Study 80
- 4.6 Conclusion and Future Work 88

Appendices **89**

A Algorithms 90

Bibliography 102

List of Tables

2.1 $|Z_N|$ for instances in our testbed. 22
2.2 Summary of computation times. All times are in seconds and significant to the nearest second. 25
2.3 Number of IPs solved for each algorithm on completed problem instances in our testbed. "N/A" indicates the algorithm failed to complete a sufficient number of problem instances to compute the summary statistic. 27
2.4 Running times (in seconds) and number of IPs solved by BOX-NBR along with respective average percent improvement over BOX. 28

3.1 Cost functions, $c_{ij}(\mathbf{x}^L, \mathbf{x}^F)$, for arcs in Figure 3.1. 34
3.2 Some interdiction strategies on the graph from Figure 3.1, recourse paths, and costs. 34
3.3 Efficient interdiction strategies on G^{11} and its subgraphs, G^{10} and G^5, recourse paths, and costs. 41
3.4 Efficient interdiction strategies on graphs of G^{11}, G^{20}, and G^{21}, $b^F = 0$ recourse paths, and objective vector. Recourse path when $b^F = 1$ is $s \to a_1 \to b_1 \to t$ in all cases. . . 42
3.5 Iteration 2 strategies for $G^{21} = G$ from Example 3.5. Since G^{20} is not updated in iteration 2, these strategies and the associated $\overline{\mathbf{z}^G}(\mathbf{x})$ values derive from the $X_E(G^{11}, \kappa^L)$ sets given in Tables 3.3 and 3.4. 50
3.6 Density by $|\mathcal{A}|$ and $|\mathcal{N}|$. 52
3.7 Algorithm running time summaries for SPG portion of the computational study. . . 54
3.8 Comparison of running times for Algorithms 3.1, 3.2, and 3.3 with p large ($p = n/2$). Columns "n" and "m" column give the number of nodes and arcs in each instance, respectively. 58

4.1 Arc costs for Figure 4.1. 61
4.2 Expected path costs given X^1. 61
4.3 Expected path costs given X^2. 62
4.4 Costs for arcs in Figure 4.2. 67
4.5 Comparison of SAA (with $|\widehat{\Omega}| = 50$ scenarios) and Knitro on SPIP-RS instances with 10 nodes and 27 arcs. 82
4.6 Summary statistics for Knitro with the lazy approach. 83
4.7 Branch-and-bound study results. "NL" indicates the non-lazy approach, in contrast to the lazy one described in Section 4.5.1. 86
4.8 Comparing objectives from Knitro and the approximation algorithm of Section 4.3.2 on convex instances. The percentage column is given by dividing the approximation objective by the Knitro objective. 87

List of Figures

2.1 Level curves for $\|\mathbf{z}\|_{\epsilon}^{\mathbf{w}} = 1$ (dashed) and $\|\mathbf{z}\|^{\mathbf{w},\epsilon} = 1$ (solid). 13
2.2 Illustration of Algorithm 2.1 iterations for Section 2.4 example problem 21
2.3 Average time for Gurobi to solve one IP sub-problem in our testbed. Each line represents an average of solver times over all IP sub-problems for one problem instance. 25
2.4 Performance profiles for algorithms. 26
2.5 Performance profiles for algorithms by p (logarithmic time axis). 26
2.6 Performance profiles for BOX-NBR (previous algorithm profiles are given as dotted lines for comparison). 28
2.7 Performance profiles for BOX-NBR by p (logarithmic time axis, previous algorithm profiles are given for comparison). 29

3.1 Graph for Example 3.1. Solid arcs are candidates for interdiction, and dotted arcs are candidates for improvement. Arc costs are given by B_{ij} $(\pm\delta_{ij})$ where B_{ij} is the arc cost without interdiction or improvement and $\pm\delta_{ij}$ is the increase or decrease in cost due to a single interdiction or improvement. Additional interdictions or improvements have no further effect. 33
3.2 Example of converting $G = (\mathcal{N}, \mathcal{A})$ into $\bar{G} = (\bar{\mathcal{N}}, \bar{\mathcal{A}})$. 36
3.3 Examples of basic SPG graphs and series-parallel configurations. 38
3.4 Binary decomposition tree, $\mathcal{T}$, for graph in Figure 3.1. Nodes are annotated $n_{\#}$ (`Type`), where `Type` is P = parallel, S = series, or L = leaf. 40
3.5 Decomposition tree for Example 3.5. Nodes needing updates each iteration are highlighted. 50
3.6 Number of `MOGA` iterations for line 3 of Algorithm 3.1 (see Algorithm 2.1). 53
3.7 Mean running time of `SolveSPIP` (averaged over all calls per instance). 54
3.8 Running times for Algorithm 3.1 on general graphs. 55
3.9 Batch 3 results by $|\mathcal{A}|$. 56
3.10 Log-log plot of running times for Algorithms 3.1, 3.2, and 3.3 on 100 SPIP-I instances with SPGs. 56
3.11 Three instances from the computational study where Algorithm 3.1 is faster than Algorithm 3.2. Nodes s and t (large gray dots) tend to be close in the graph. 57
3.12 Three instances from the computational study where Algorithm 3.2 is faster than Algorithm 3.1. Nodes s and t (large gray dots) tend to be relatively distanced in the graph. 58

4.1 Example 1 graph. 61
4.2 Gadgets used in proof of Theorem 4.3. 67
4.3 Effect of the lazy approach on the objective. "X" markers indicate non-lazy approach timed-out before converging. 83
4.4 Effect of the lazy approach on running time. Instances with $n = 80$ nodes are excluded. 84
4.5 Running time (in seconds) for Knitro (using the lazy approach) by number of graph nodes (n). 84

A.1 Algorithm 2.1 search functions separated into three modules. 90

Chapter 1

Introduction

First conceived in the 1960s [19, 93], network interdiction problems (NIPs) were developed to respond to strategists concerns about worst-case scenarios of how an adversary (e.g., USSR) might attack road or railway networks. For this application, modelers placed the adversary as the interdicting agent and public organizations as the followers, and the assumed the interdictor had full access to the network information. In contrast, more recent NIP applications examine how public organizations might disrupt the networks and activities of criminals. Examples include interdiction of nuclear smuggling [58], improvement of border protection [82], and combating human trafficking [47]. In these formulations, the public organization is the leader and the social disruptors are the followers. This reversal of roles warrants reconsideration of the underlying assumptions inherent in NIP formulations.

1.1 Background and Literature Review

Smith et al. [76] present an introductory summary on network interdiction problems and review of the relevant literature, and the seminal work of Wood [94] provides a general expanded treatment of network interdiction problems, including the common network interdiction problem we consider in this dissertation: the shortest path interdiction problem (SPIP). In the SPIP a leading agent (the *leader* or *interdictor*) and a following agent (the *follower*) interact in a maximin game on a network, $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. We denote the size of this graph by $n = |\mathcal{N}|$ and $m = |\mathcal{A}|$. The leader selects an interdiction strategy, subject to a cardinality constraint, b, on the number of total number of interdictions. We represent the leader's interdiction decision by the vector $\mathbf{x} \in \{0, \ldots, b\}^m$, where each component x_a denotes the number of interdictions on arc $a \in \mathcal{A}$. Each successive interdiction on arc $a \in \mathcal{A}$ raises the arc cost according to a nondecreasing function $c_a : \{0, \ldots, b\} \to \mathbb{R}_+$. The follower observes the interdiction strategy and chooses a shortest path through the interdicted network from a source node $s \in \mathcal{N}$ to a destination node, $t \in \mathcal{N}$. The follower's decision is given by the vector $\mathbf{y} \in \{0,1\}^m$, where $y_a = 1$ if arc $a \in \mathcal{A}$ is on the follower's path, and $y_a = 0$ otherwise. The SPIP may be formulated as a bi-level mathematical program:

$$
\begin{aligned}
\max \quad & z(\mathbf{x}) && && \text{(1.1a)}\\
\text{subject to:} \quad & \sum_{a \in \mathcal{A}} x_a \le b && && \text{(1.1b)}\\
& x_a \in \mathbb{Z}_+ && \forall a \in \mathcal{A}, && \text{(1.1c)}\\
\text{where } z(\mathbf{x}) = \min \quad & \sum_{a \in \mathcal{A}} c_a(x_a) y_a && && \text{(1.1d)}\\
\text{subject to:} \quad & \sum_{\substack{j \in \mathcal{N}: \\ (i,j) \in \mathcal{A}}} y_{ij} - \sum_{\substack{j \in \mathcal{N}: \\ (j,i) \in \mathcal{A}}} y_{ji} = \begin{cases} 1 & i = s \\ -1 & i = t \\ 0 & \text{otherwise} \end{cases} && \forall i \in \mathcal{N} && \text{(1.1e)}\\
& y_{ij} \in \mathbb{Z}_+ && \forall (i,j) \in \mathcal{A}, && \text{(1.1f)}
\end{aligned}
$$

where $\mathbb{Z}_+$ denotes the non-negative integers. The leader's outer problem, given in (1.1a)–(1.1c), selects an interdiction strategy, while (1.1d)–(1.1f) capture the follower's inner problem of finding a shortest path. Constraint (1.1b) limits the number of interdicted arcs, and (1.1e) are the standard flow constraints for the follower's shortest path problem. Since the coefficient matrix for (1.1e) is totally unimodular, we may relax the integrality constraints in (1.1f), resulting in a linear program for the follower's shortest path, and taking the dual of the follower's inner problem, we can collapse (1.1) into the following equivalent mixed integer linear program:

$$
\begin{aligned}
\max \quad & d_s && \text{(1.2a)}\\
\text{subject to:} \quad & d_i - d_j \le c_a(x_a) && \text{(1.2b)}\\
& \sum_{a \in \mathcal{A}} x_a \le b && \text{(1.2c)}
\end{aligned}
$$

$$d_t = 0 \tag{1.2d}$$

$$x_a \in \mathbb{Z}_+ \qquad \forall a \in \mathcal{A}, \tag{1.2e}$$

where each d_i is the dual variable corresponding to the i^{th} flow constraint in (1.1e) and represents the shortest path length from node i to node t. The assignment $d_t = 0$ in (1.2d) gives an arbitrary assignment for the extra d_t variable since (1.1e) includes one redundant constraint. The SPIP is NP-hard [3], and the leading approaches for solving it use cutting-plane algorithms [36, 50].

In this dissertation, we introduce uncertainty in the determistic models of (1.1) and (1.2). Multiple works examine stochastic variations of the SPIP; these may generally be grouped into three cases:

1. **Symmetric uncertainty**: Here the leader and follower lack the same information. For example [14, 37, 63] consider scenarios with uncertain success or effectiveness of the interdiction actions, and [29, 31] model a NIP with uncertain network topology.

2. **Asymmetric uncertainty**: Asymmetric access to information may benefit either the leader or the follower. For example, [5, 72, 82] examine uncertain NIPs where the follower lacks some of the arc cost information. In each of these articles, the follower is an evader attempting to penetrate the network but lacks some awareness of the interdictor's actions or their effect on the network. In contrast, [10, 30, 51] present formulations where it is the leader who lacks access to arc cost information. All three of these studies present multi-stage NIPs, where each stage evolves a new interdiction strategy until the leader gains enough information to converge to a stable strategy. Also [77] presents a formulation where the arc costs are stochastic, but the follower adopts a "wait and see" strategy, effectively placing the leader at an information disadvantage; they present the leader's problem using a robust optimization formulation.

3. **Uncertain Strategies**: In some NIP scenarios, such as the case of deterring fare evasion on public transit [2, 9, 15], the leader selects a randomized interdiction strategy, and the follower observes the probabilities of various interdiction locations and then selects a preferred path. Alternatively, in modeling the NIP as a simultaneous game, [90] show that a Nash equilibrium is attained when both the leader and follower adopt mixed strategies.

Two common approaches to handling uncertainty include robust optimization (RO) models [7] and stochastic programming (SP) models [8, 40]. Numerous studies [32, 35, 42, 43, 59, 64, 73] explore the possibility of expressing RO and SP techniques using multiobjective formulations, but application of multiobjective formulations to uncertain problems are extremely limited in practice. In particular, no studies to our knowledge use multiobjective methods to handle uncertainty in NIPs (though [69, 70, 71] consider deterministic multiobjective NIPs). We find this approach is useful, however, for in one of our models, provided the number of uncertain scenarios is limited.

1.2 Contribution and Motivation

In this dissertation, we contribute two new SPIP variants that exhibit uncertainty. For each variant, we introduce the problem, examine its properties, propose various algorithms to exploit the properties, and compare the algorithms in computational studies. In addition to these new SPIP variants, we also contribute a new algorithm for generating Pareto solutions for discrete multiobjective problems.

Our contribution of a multiobjective generating algorithm is vital to tractably handling problems with uncertainty using multiobjective techniques. Multiobjective formulations of uncertain problems require a separate objective for each possible scenario, and current techniques for generating nondominated solutions to multiobjective exhibit exponential growth in running times as the number of objectives grows. Thus, the set of possible scenarios must be kept extremely small, and hence the applications for multiobjective approaches to uncertain problems are relatively limited. We contribute a new algorithm, based on the augmented weighted Tchebychev norm, that tends to scale better (with respect to the number of objectives) than other leading techniques in generating a complete set of non-dominated solutions. This property allows us to use tractable multiobjective methods in presenting and evaluating trade-offs in our first variant of the SPIP.

Our first variant is the SPIP with arc improvements (SPIP-I). It exhibits information asymmetry that benefits the follower, but differs from previous studies where the (unknown) true costs are fixed. Rather, the SPIP-I allows that (in the vernacular) "the enemy gets a vote," by modeling the follower as an adaptive agent, who observes the leader's actions and reacts to mitigate the interdiction strategy. In this variant, the follower is permitted to lower arc costs (i.e., "improve arcs") after observing the interdictions, subject to a budget constraint that is unknown to the the leader. We adopt a multiobjective approach to this problem, seeking to generate a complete set of Pareto-optimal interdiction strategies; that is, for each such strategy it is not possible to improve the objective for one given value of the follower's interdiction budget without degrading the objective for some other value of the follower's interdiction budget. Arc improvements encompass a variety of situations that arise in network interdiction analysis. For example, the leader may suspect the follower has some hidden arcs and considers a worst-case event in which the follower might use them to avoid interdicted arcs. Another example is that the follower may commit additional resources after the interdiction to reduce the cost of using an arc, perhaps as a nonlinear function of expenditures in improvements (see, e.g., [52]). Both of these scenarios may be represented using our framework of arc improvements with the interdictor uncertain of the improvement budget.

Our second variation is the shortest path interdiction problem with randomized strategies (SPIP-RS). This variant resembles the models of [2, 9, 15] in that the leader can select a *randomized* interdiction strategy and the follower must select a path knowing only the probability distribution specified in the leader's interdiction strategy (and not the true interdiction locations). The follower's goal is to select a path having minimum *expected* cost, given the leader's (stochastic) interdiction strategy. The SPIP-RS is motivated by real-life applications where randomized interdiction strategies can be gainfully employed. This situation arises frequently in security applications, for which true interdiction resources are dispersed among decoys. For instance, detection devices such as remote cameras may be placed at border crossings, only some of which are monitored due to personnel limits. (Note the equivalence between maximizing a minimum-cost path and minimizing a maximum-evasion-probability path for an evader; see, e.g., [63].) The evader may obtain data to ascertain evasion probabilities in the presence of cameras, but the interdictor can reasonably randomize its monitoring activity and hide the day-to-day assignment of interdiction resources from the evader. Unlike previous studies that assume affine cost functions, we consider the complexity of the SPIP-RS when the cost functions are nonlinear. Returning to the border protection scenario, concave costs might occur if mounting and monitoring additional cameras at a location yields diminishing returns in detection probability. We also examine the case of convex cost functions, such as when monitoring agents work in teams and are more effective as a group than they are as individuals. Lunday and Sherali [52] provide a thorough discussion of such synergies among interdiction resources, leading to nonlinear interdiction models such as the one we study here.

1.3 Dissertation Organization

Chapter 2 introduces our multiobjective approach based on the modified augmented weighted Tchebychev (MAWT) norm. We present the MAWT algorithm and demonstrate its computations with an example. Our computational study shows that the MAWT algorithm performs comparatively to other leading algorithms on problems with few objectives, and it scales better as the number of objectives increases.

After introducing our MAWT algorithm, we proceed to apply it as we consider the SPIP-I in Chapter 3. We introduce the problem, give a formulation based on the SPIP formulation of (1.1), and use an example to demonstrate the value of a multiobjective approach. We then present three algorithms for solving the SPIP-I and compare their performance in a computational study. We conclude Chapter 3 with ideas for future research on the SPIP-I.

Chapter 4 presents our study of the SPIP-RS. As in the previous chapter, we introduce the problem, formulate it based on (1.2), and give an example. We prove that this problem is NP-complete and present several algorithms for solving it. We provide computation results from an experiment and conclude with areas of future research.

Chapter 2

Modified Augmented Weighted Tchebychev Scalarizations

2.1 Chapter Introduction

In this chapter we examine discrete multi-objective optimization problems in which there can exist any finite number of objective functions. The goal of this study is to provide an algorithm that generates a complete set of efficient solutions to the problem without user intervention. Section 2.1.1 presents concepts and notation used in this chapter. Section 2.1.2 discusses literature most closely related to our study. Section 2.1.3 states our primary contributions and gives the organization of the chapter.

2.1.1 Preliminaries

Consider the multi-objective problem:

$$\begin{aligned} \min\ & [z_1(\mathbf{x}), \ldots, z_p(\mathbf{x})] \\ \text{subject to: } & \mathbf{x} \in X, \end{aligned}$$

where $p \in \mathbb{N}$, $X \subseteq \mathbb{R}^n$, and $z_k : X \to \mathbb{R}$ for each $k \in P = \{1, \ldots, p\}$. We call X the *decision set* and $\mathbb{R}^n$ the *decision space*. For any $X' \subseteq X$, let $\mathbf{z}(X') = \bigcup_{\mathbf{x} \in X'} \mathbf{z}(\mathbf{x})$ be the image set of X' under the vector-valued function $\mathbf{z}(\mathbf{x}) = [z_1(\mathbf{x}), \ldots, z_p(\mathbf{x})]$. We call $Z = \mathbf{z}(X)$ the *solution set* and $\mathbb{R}^p$ the *objective space* or *solution space*.

We apply the standard Pareto preference ordering to the vectors in $\mathbb{R}^p$, as given in the following definitions.

Definition 1. Let $\mathbf{z}^a, \mathbf{z}^b \in \mathbb{R}^p$. Then

$$\mathbf{z}^a \leqq \mathbf{z}^b \Leftrightarrow z_k^a \leq z_k^b,\ \forall k \in P. \text{ We say } \mathbf{z}^a \textit{ weakly dominates } \mathbf{z}^b.$$
$$\mathbf{z}^a \leq \mathbf{z}^b \Leftrightarrow \mathbf{z}^a \leqq \mathbf{z}^b \text{ and } \mathbf{z}^a \neq \mathbf{z}^b. \text{ We say } \mathbf{z}^a \textit{ dominates } \mathbf{z}^b.$$
$$\mathbf{z}^a < \mathbf{z}^b \Leftrightarrow z_k^a < z_k^b,\ \forall k \in P. \text{ We say } \mathbf{z}^a \textit{ strongly dominates } \mathbf{z}^b.$$

Definition 2. $\mathbf{z} \in Z$ is

$$\textit{non-dominated} \Leftrightarrow \nexists \mathbf{z}' \in Z : \mathbf{z}' \leq \mathbf{z}, \text{ and}$$
$$\textit{weakly non-dominated} \Leftrightarrow \nexists \mathbf{z}' \in Z : \mathbf{z}' < \mathbf{z}.$$

Definition 3. The non-dominated set is $Z_N = \{\mathbf{z} \in Z : \mathbf{z} \text{ is non-dominated}\}$.

The necessary and sufficient conditions for our algorithm to operate correctly are given in Assumptions 2.1 and 2.2. However, since both conditions refer to Z_N, which is unknown at the outset, it may be difficult to verify that these assumptions hold. Therefore, Lemma 2.1 states a condition on Z that is sufficient for Assumptions 2.1 and 2.2 to hold.

Assumption 2.1. $|Z_N| < \infty$.

Assumption 2.2. $\forall \mathbf{z} \in Z$, there exists $\mathbf{z}^* \in Z_N$ such that $\mathbf{z}^* \leqq \mathbf{z}$.

Lemma 2.1. If $|Z| < \infty$, then Assumptions 2.1 and 2.2 hold.

Proof. Assume $|Z| < \infty$. By construction, $Z_N \subseteq Z$ so $|Z_N| \leq |Z| < \infty$, and Assumption 2.1 holds. To show Assumption 2.2 holds, consider $\mathbf{z}_1 \in Z$. If $\mathbf{z}_1$ is non-dominated, then $\mathbf{z}_1 \in Z_N$. Otherwise it is dominated, and there exists $\mathbf{z}_2 \in Z : \mathbf{z}_2 \leq \mathbf{z}_1$. If $\mathbf{z}_2$ is non-dominated, then $\mathbf{z}_2 \in Z_N$. Otherwise, $\exists \mathbf{z}_3 \in Z : \mathbf{z}_3 \leq \mathbf{z}_2$. Continuing in this manner, we can construct the sequence $\mathbf{z}_1 \geq \mathbf{z}_2 \geq \cdots \geq \mathbf{z}_d$ where $\{\mathbf{z}_i : 1 \leq i \leq d \leq |Z|\} \subseteq Z$ and $\nexists \mathbf{z} \in Z : \mathbf{z} \leq \mathbf{z}_d$. Thus, $\mathbf{z}_d \in Z_N$, and Assumption 2.2 holds. This completes the proof. $\square$

Definition 4. For $Z' \subseteq \mathbb{R}^p$, $\mathbf{z}^{\text{lb}} \in \mathbb{R}^p$ is a *(weak/·/strong) lower bound* if ($\mathbf{z}^{\text{lb}} \leqq \mathbf{z}/\mathbf{z}^{\text{lb}} \leq \mathbf{z}/\mathbf{z}^{\text{lb}} < \mathbf{z}$), $\forall \mathbf{z} \in Z'$.

Definition 5. For $Z' \subseteq \mathbb{R}^p$, $\mathbf{z}^{\text{ub}} \in \mathbb{R}^p$ is a *(weak/·/strong) upper bound* if ($\mathbf{z} \leqq \mathbf{z}^{\text{ub}}/\mathbf{z} \leq \mathbf{z}^{\text{ub}}/\mathbf{z} < \mathbf{z}^{\text{ub}}$), $\forall \mathbf{z} \in Z'$.

Lemma 2.2. (See [83].) Let $Z \subseteq \mathbb{R}^p$. Then $|Z| < \infty \Leftrightarrow Z$ is compact and discrete. □

Corollary 2.2.1. Z_N has a weak lower bound, $\mathbf{z}^{\text{lb}}$, and a strong upper bound, $\mathbf{z}^{\text{ub}}$.

Proof. The result follows from the boundedness of Z_N given by Assumption 2.1 and Lemma 2.2. □

Definition 6. The *step size*, s, over Z_N is given by $\min_{\{\mathbf{z}^a, \mathbf{z}^b \in Z_N,\ k \in P\}} \{|z_k^a - z_k^b| : z_k^a \neq z_k^b\}$.

Lemma 2.3. Let $r = \max_{k \in P}\{z_k^{\text{ub}} - z_k^{\text{lb}}\}$ where $\mathbf{z}^{\text{lb}}$ and $\mathbf{z}^{\text{ub}}$ are weak lower and strong upper bounds on Z_N, respectively, and let s be the step size over Z_N. Then, $|Z_N| > 1 \Rightarrow 0 < s < r$.

Proof. Assume $|Z_N| > 1$. We note $0 < s$, since $s = \min_{\{\mathbf{z}^a, \mathbf{z}^b \in Z_N,\ k \in P\}} \{|z_k^a - z_k^b| : z_k^a \neq z_k^b\} > 0$. To show $s < r$, let $\mathbf{z}^a,\ \mathbf{z}^b \in Z_N$ where $\mathbf{z}^a \neq \mathbf{z}^b$. Then we have some $k \in P$ such that $z_k^a < z_k^b$, which gives us:

$$\begin{aligned} & z_k^{\text{lb}} + s \leq z_k^a + s \leq z_k^b < z_k^{\text{ub}} \\ \Rightarrow\ & z_k^{\text{lb}} + s < z_k^{\text{ub}} \\ & \Leftrightarrow s < z_k^{\text{ub}} - z_k^{\text{lb}} \leq r. \end{aligned}$$

Thus, $0 < s < r$, and the proof is complete. □

Definition 7. A point $\mathbf{z}^* \in Z$ is *k-preferred* for some $k \in P$ if $z_k^* \leq z_k,\ \forall \mathbf{z} \in Z$. A set $Z' \subseteq Z$ of preferred points is a *complete set of preferred points* if $\forall k \in P$, $\exists \mathbf{z}' \in Z'$ such that $\mathbf{z}'$ is k-preferred.

In addition to the above definitions for the solution space, we present the following definitions regarding efficiency in the decision space.

Definition 8. Point $\mathbf{x} \in X$ is *efficient* if $\mathbf{z}(\mathbf{x}) \in Z_N$. The set $X_E = \{\mathbf{x} \in X : \mathbf{x} \text{ is efficient}\}$ is the efficient set.

Definition 9. Any subset $X' \subseteq X_E$ such that $\mathbf{z}(X') = Z_N$ is a *complete set of efficient points.*

2.1.2 Literature Review

Hwang and Masud [34] classify techniques to find non-dominated points into three categories, depending on the level of decision maker interaction: a-priori methods, interactive, and a-posteriori (or generating) methods. Our approach falls into the a-posteriori category. These methods seek to generate the full set of non-dominated objectives, and then allow the decision maker (a-posteriori) to select any one among those solutions. Our review of relevant literature examines five families of generating methods.

2.1.2.1 ϵ-Constraint Methods

First presented by [27], the ϵ-constraint method constrains all objectives but one. Those constraints are initially loose, and they are iteratively tightened. At each iteration, the algorithm minimizes the unconstrained objective, subject to the other objectives' constraints. The algorithm terminates when no feasible solutions remain under the tightened constraints.

Several improvements for the ϵ-constraint method have been proposed. Notably, while Haimes et al.'s algorithm may return points that are only weakly non-dominated, [21] develop the elastic ϵ-constraint method, which avoids generating such weak solutions. Their method adds slack variables to the ϵ-constraints and then appends a small (possibly zero) weighted contribution of those slack variables to the objective. Both [28] and [60] improve the algorithm using a weighting scheme that produces a lexicographical ordering for tightening the ϵ-constraints. [48] and [41] suggest methods to reduce the number of search regions, which reduces computation time considerably. Also, [95] provide further improvements via early exits and bouncing steps. Finally, [53] and [56] implement the augmented ϵ-constraint method, combining the elastic method with lexigraphically-ordered searches.

2.1.2.2 Klein and Hannan's Family of Methods

Klein and Hannan [46] develop the dominated region exclusion approach, which iteratively identifies efficient solutions and then adds constraints to the problem that exclude those solutions and the regions dominated by them. Sylva and Crema [84] enhance the method by using a sum of the objective functions, but they observe that the approach is impractical for large non-dominated sets, because the constraint set grows each iteration. Lokman and Koksalan [49] also improve the method with two algorithms. Their first algorithm reduces the number of constraints, thus improving the results in [84], but still suffering for large non-dominated sets. Their second algorithm replaces a single subproblem with many variables and constraints, with multiple subproblems with fewer variables and constraints. Numerical tests suggest this second algorithm seems to overcome the computational limitations of previous efforts to generate Z_N from the previous studies in this family.

2.1.2.3 Two-Phase Methods

The two-phase method was first suggested by [85]. The concept is to find solutions on the convex hull of the non-dominated set by means of a weighted sum of the objective functions. Then a different method, e.g., branch-and-bound as given in [85], is used to find the solutions in the interior of the hull. This algorithm is applied to various types of bi-objective problems, including knapsack [88], minimum spanning tree [68, 78], minimum cost flow [67] and assignment [65, 66, 85]. While all use the weighted sum for the first phase, the authors vary their search algorithms for the second. While most papers are restricted to biobjective problems, the paper [66] demonstrates an application of this method to a problem having more than two objectives.

2.1.2.4 Branch-and-Bound Methods

White [91] conceives of applying branch-and-bound algorithms to multi-objective combinatorial optimization problems. This approach conducts the search in the decision space rather than the solution space. Thus, it is the components of $\mathbf{x} \in X \subseteq \mathbb{R}^n$ that are branched, fathomed, and pruned. The benefit of this approach, as shown in [54, 55] and [87], is that it is capable of generating a full representation of Z_N even for combinatorial multi-objective mixed-integer programs (MOMIPs) where $|Z_N| \not< \infty$. Then [81] compare the performance of a bi-objective branch-and-bound algorithm against a two-phase algorithm on six types of combinatorial MOMIPs, demonstrating that the branch-and-bound method performed better than the two-phase method in five of the six types of problems. Finally, [6] demonstrate a multi-objective branch-and-bound algorithm for general integer problems.

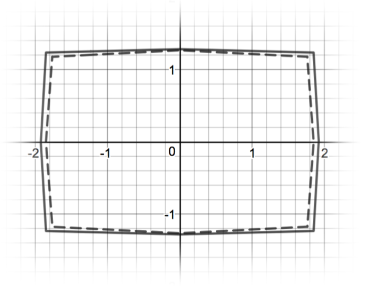
2.1.2.5 Tchebychev Norm Methods

Bowman [11] suggests the application of the Tchebychev norm for finding the non-dominated set of a multi-objective problem. Since the norm is known to guarantee only weakly efficient solutions, [80] propose augmenting the base (Tchebychev) norm with a weighted sum of all the objectives multiplied by some constant coefficient. This augmented norm is applied successfully in interactive methods [1, 79, 92] as well as in generating approximations to the non-dominated set [45, 74]. We are unaware of any complete generating methods using the Tchebychev norm until the work of [17]. Dächert et al. demonstrate that, in general, no single coefficient will generate the entire non-dominated set for a problem instance. However, by adaptively adjusting the coefficient based on the search region, they use the augmented Tchebychev norm to generate Z_N for bi-objective problems. Their results exhibit comparable average computation time to the ϵ-constraint method as implemented by [62], with about half the standard deviation.

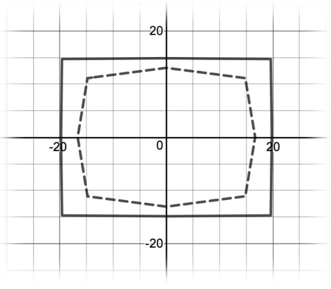
2.1.3 Contribution and Overview

Similar to the work of Dächert et al., our approach employs a variant of the Tchebychev norm, adaptively adjusting the search parameters without human intervention to generate Z_N. Our contribution is to propose a modification to the augmented Tchebychev norm that affords a simpler computation of the coefficient for the augmentation term. We also give an algorithm to generate the non-dominated set for any number of objectives. To our knowledge, this is the first generating method that uses a variant of the Tchebychev norm to generate the non-dominated set for discrete multi-objective optimization problems with any number of objectives. By contrast, the approach in [17] employs the (more traditional) augmented weighted Tchebychev norm with a sophisticated parameter scheme in order to generate a complete efficient set of solutions. The authors additionally provide a rigorous analysis of the parameters employed in the augmented weighted Tchebychev norm. The goal in their work is to find the largest possible augmentation parameter that still guarantees that all non-dominated points can be found, noting that larger values of the augmentation term tend to avoid numerical difficulties in optimization. By contrast, the structure of the variant of the Tchebychev norm that we propose in this chapter will allow us to state a simpler weighting scheme that can be used to automatically generate a complete efficient set of solutions with any number of criteria.

In Section 2.2 we present our proposed modification and our algorithm. We prove the correctness and finiteness of our algorithm in Section 2.3, and give an example in Section 2.4. In Section 2.5 we present computational results. The chapter concludes in Section 2.6 with consideration of future research areas.



(a) $\mathbf{w} = \left[\frac{1}{2}, \frac{3}{4}\right]$ and $\epsilon = 0.04$.



(b) $\mathbf{w} = \left[\frac{1}{20}, \frac{1}{15}\right]$ and $\epsilon = 0.01$.

Figure 2.1: Level curves for $\|\mathbf{z}\|_{\epsilon}^{\mathbf{w}} = 1$ (dashed) and $\|\mathbf{z}\|^{\mathbf{w},\epsilon} = 1$ (solid).

2.2 Algorithm

We provide Algorithm 2.1 in this section, which generates a complete efficient set without requiring user intervention. Our approach uses a modification of the augmented weighted Tchebychev norm [17, 80]. The (original) augmented weighted Tchebychev norm is given by:

$$\|\mathbf{z}\|_{\epsilon}^{\mathbf{w}} = \|\mathbf{z}\|_{\infty}^{\mathbf{w}} + \epsilon\|\mathbf{z}\|_1 = \max_{k\in P}\{w_k|z_k|\} + \epsilon\sum_{k\in P}|z_k|, \tag{2.1}$$

where $\mathbf{w} \geq \mathbf{0}$ and $\epsilon > 0$. Our proposed modification incorporates the same weighting into the augmentation term as is used on the base Tchebychev norm. Thus, the modified augmented weighted Tchebychev (MAWT) norm we propose is:

$$\|\mathbf{z}\|^{\mathbf{w},\epsilon} = \|\mathbf{z}\|_{\infty}^{\mathbf{w}} + \epsilon\|\mathbf{z}\|_1^{\mathbf{w}} = \max_{k\in P}\{w_k|z_k|\} + \epsilon\sum_{k\in P}w_k|z_k|. \tag{2.2}$$

Kaliszewski [38] also examines a modification of the Tchebychev norm similar to (2.2) in some respects; however, there are distinct differences. Also, [39] considers a more general Tchebychev-norm variant which, with the appropriate selection of the parameters, may be reduced to (2.2). These papers aim at interactive (a-posteriori) multi-objective methods and do not provide the contribution that is the focus of this chapter. We distinguish between the norms (2.1) and (2.2) by placing the ϵ parameter in the superscript in (2.2) rather than the subscript as in (2.1). Figure 2.1 compares level curves for (2.1) and (2.2).

In the initialization phase of Algorithm 2.1, we determine a weak lower bound, $\mathbf{z}^{\text{lb}}$, by minimizing each objective individually to give a complete set of preferred points, and selecting their

Algorithm 2.1: Search algorithm using the MAWT norm

```
Input  : $\mathcal{I}$: Multi-objective problem instance, $\min\{[z_1(\mathbf{x}),\dots,z_p(\mathbf{x})] : \mathbf{x} \in X\}$
         $s$: Positive lower bound on single-objective step sizes in $Z$
         $\mathbf{z}^{\text{ub}}$: Strong upper bound for $Z_N$, (i.e., $\mathbf{z}^{\text{ub}} > \mathbf{z},\ \forall z \in Z_N$)
Output : $\widehat{X}_E$: A complete efficient set
         $\widehat{Z}_N$: The non-dominated set
Initialize
1  for $k \in P$ do
2      $z_k^{\text{lb}} \leftarrow \min_{\mathbf{x}\in X} z_k(\mathbf{x})$        // At the end $\mathbf{z}^{\text{lb}} = [z_1^{\text{lb}},\dots,z_p^{\text{lb}}]$
3  Choose $\epsilon \in (0, s/(p(r-s)))$ where $r = \max_{k\in P}\{z_k^{\text{ub}} - z_k^{\text{lb}}\}$  // e.g., $\epsilon \leftarrow s/(2p(r-s))$
4  $\widehat{X}_E \leftarrow \emptyset$, $\widehat{Z}_N \leftarrow \emptyset$, $j \leftarrow 0$; and $L \leftarrow \{\mathbf{z}^{\text{ub}}\}$
Main Loop
5  while $L \neq \emptyset$ do
6      $j \leftarrow j+1$
7      Choose $\mathbf{z}^{(j)} \in L$; $L \leftarrow L \setminus \{\mathbf{z}^{(j)}\}$
8      for $k \in P$ do
9          $w_k \leftarrow 1/\left(\max\left\{s, z_k^{(j)} - z_k^{\text{lb}}\right\}\right)$        // At the end $\mathbf{w} = [w_1,\dots,w_p]$
10     $g(\mathbf{z}(\mathbf{x})) \leftarrow \|\mathbf{z}(\mathbf{x}) - \mathbf{z}^{\text{lb}}\|^{\mathbf{w},\epsilon}$
11     $\mathbf{x}^{(j)} \leftarrow \text{argmin}_{\mathbf{x}\in X}\, g(\mathbf{z}(\mathbf{x}))$
12     if $g(\mathbf{z}(\mathbf{x}^{(j)})) < 1$ then
13         if $\mathbf{z}(\mathbf{x}^{(j)}) \notin \widehat{Z}_N$ then $\widehat{X}_E \leftarrow \widehat{X}_E \cup \{\mathbf{x}^{(j)}\}$ and $\widehat{Z}_N \leftarrow \widehat{Z}_N \cup \{\mathbf{z}(\mathbf{x}^{(j)})\}$.
14         for $k \in P$ do
15             if $z_k(\mathbf{x}^{(j)}) > z_k^{\text{lb}}$ then
16                 $\mathbf{z}^{(j),-k} \leftarrow [z_1^{(j)},\dots,z_{k-1}^{(j)}, z_k(\mathbf{x}^{(j)}), z_{k+1}^{(j)},\dots,z_p^{(j)}]$
17                 $L \leftarrow L \cup \{\mathbf{z}^{(j),-k}\}$
Terminate
18 return $\widehat{X}_E, \widehat{Z}_N$
```

respective preferred objective values (lines 1–2). Thus $\mathbf{z}^{\text{lb}} \leqq \mathbf{z} < \mathbf{z}^{\text{ub}},\ \forall \mathbf{z} \in Z_N$, where $\mathbf{z}^{\text{ub}}$ is a strong upper bound over Z_N given as input to the algorithm. We choose not to compute $\mathbf{z}^{\text{ub}}$ because it is generally non-trivial to solve for a upper bound on Z_N [20]. However, a strong upper bound for Z_N is often apparent for specific problem classes (e.g., $(-1, ..., -1)$ for a multi-objective max flow problem).

Next, with $\mathbf{z}^{\text{lb}}$ and $\mathbf{z}^{\text{ub}}$, we solve $r = \max_{k \in P}\{z_k^{\text{ub}} - z_k^{\text{lb}}\}$, and we select a parameter ϵ such that:

$$0 < \epsilon < \frac{s}{p(r-s)}, \tag{2.3}$$

where $s > 0$ is a lower bound on the step size (given as an input parameter). Recall from Lemma 2.3 that either $s < r$, or $|Z_N| \leq 1$ (moreover, $|Z_N| = 0 \Rightarrow X = \emptyset$ by Assumption 2.2, and $|Z_N| = 1 \Rightarrow Z_N = \{\mathbf{z}^{\text{lb}}\}$ by construction of $\mathbf{z}^{\text{lb}}$). Finally we initialize our lists of efficient and non-dominated objectives $\widehat{X}_E$ and $\widehat{Z}_N$ as empty sets, our iteration counter $j = 0$, and list of search regions, L, to contain a single entry.

Each iteration j of the main loop begins in line 5 by choosing a vector, $\mathbf{z}^{(j)} \in L$, and removing it from L. The vector $\mathbf{z}^{(j)}$ uniquely defines our search region for that iteration, $B^{(j)} = \{\mathbf{z} \in \mathbb{R}^p : \mathbf{z}^{\text{lb}} \leqq \mathbf{z} < \mathbf{z}^{(j)}\}$, a hyperbox with weak lower and strong upper bounds given by the points $\mathbf{z}^{\text{lb}}$ and $\mathbf{z}^{(j)}$, respectively. To determine whether $B^{(j)} \cap Z_N = \emptyset$, we will use (2.2), for which we require parameters $\mathbf{w}$ and ϵ. We have the ϵ parameter from line 3. To find $\mathbf{w}$, we consider the side-lengths of $B^{(j)}$, given by $(z_k^{(j)} - z_k^{\text{lb}})$ for each $k \in P$. Then we take:

$$\mathbf{w} = \left[\frac{1}{\max\{s, z_1^{(j)} - z_1^{\text{lb}}\}}, \ldots, \frac{1}{\max\{s, z_p^{(j)} - z_p^{\text{lb}}\}}\right]. \tag{2.4}$$

Intuitively, equation (2.4) scales the side lengths to unit values, with the caveat that all side lengths are at least s to prevent division by zero.

In line 10, we use the MAWT norm of (2.2) to define our scalarizing function, $g : \mathbb{R}^p \to \mathbb{R}$, and in line 11 we use g to solve

$$\min_{\mathbf{x} \in X}\ g(\mathbf{z}(\mathbf{x})) = \|\mathbf{z}(\mathbf{x}) - \mathbf{z}^{\text{lb}}\|^{\mathbf{w},\epsilon}, \tag{2.5}$$

where ϵ and $\mathbf{w}$ are given by (2.3) and (2.4), respectively. Note that (2.5) is feasible as long as $X \neq \emptyset$ due to Assumption 2.2, and it is bounded because the objective is non-negative. If the solution, $\mathbf{x}^{(j)}$, has objective value $g(\mathbf{z}(\mathbf{x}^{(j)})) < 1$, then we claim $\mathbf{z}(\mathbf{x}^{(j)}) \in B^{(j)} \cap Z_N$. In that case, line 13 adds the solutions $\mathbf{x}^{(j)}$ and $\mathbf{z}(\mathbf{x}^{(j)})$ to $\widehat{X}_E$ and $\widehat{Z}_N$, respectively, unless $\mathbf{z}(\mathbf{x}^{(j)})$ already belongs to $\widehat{Z}_N$. Lines 14–17 then spawn "child" upper bounds, $\{\mathbf{z}^{(j),-k} \in \mathbb{R}^p : \mathbf{z}^{(j),-k} = [z_1^{(j)}, \ldots, z_{k-1}^{(j)}, z_k(\mathbf{x}^{(j)}), z_{k+1}^{(j)}, \ldots, z_p^{(j)}], k \in P\}$ and add them to L. Otherwise, if $g(\mathbf{z}(\mathbf{x}^{(j)})) \geq 1$, we claim $B^{(j)} \cap Z_N = \emptyset$, and Algorithm 2.1 proceeds to the next iteration. We prove both of the foregoing claims in Section 2.3.

Algorithm 2.1 terminates when no remaining vectors are in L, and we claim (and prove in Section 2.3) that $\widehat{X}_E$ is a complete efficient set.

2.3 Algorithm Proof

We begin this section by demonstrating that Algorithm 2.1 generates only efficient solutions.

Proposition 2.1. The solution, $\mathbf{x}^{(j)}$, obtained in line 11 of Algorithm 2.1 is efficient.

Proof. Let $\mathbf{x}^{(j)}$ optimize (2.5) with ϵ and $\mathbf{w}$ given as in lines 3 and 9, respectively. By contradiction, assume $\exists \mathbf{z} \in Z$ such that $\mathbf{z} \leq \mathbf{z}(\mathbf{x}^{(j)})$. Then

$$\begin{aligned}
\|\mathbf{z}(\mathbf{x}^{(j)}) - \mathbf{z}^{\text{lb}}\|^{\mathbf{w},\epsilon} &= \max_{k\in P} \left\{ w_k(z_k(\mathbf{x}^{(j)}) - z_k^{\text{lb}}) \right\} + \epsilon \sum_{k\in P} w_k(z_k(\mathbf{x}^{(j)}) - z_k^{\text{lb}}) \\
&> \max_{k\in P} \left\{ w_k(z_k - z_k^{\text{lb}}) \right\} + \epsilon \sum_{k\in P} w_k(z_k - z_k^{\text{lb}}) \\
&= \|\mathbf{z} - \mathbf{z}^{\text{lb}}\|^{\mathbf{w},\epsilon},
\end{aligned}$$

which contradicts the optimality of $\mathbf{x}^{(j)}$. This completes the proof. □

Remark 1. The inequality in the proof of Proposition 2.1 is valid because $\mathbf{z}^{\text{lb}} \leq \mathbf{z}(\mathbf{x}^{(j)})$. However, if $\epsilon w_k(z_k(\mathbf{x}^{(j)}) - z_k^{\text{lb}})$ becomes very small for all $k \in P$, then round-off errors may allow a non-dominated solution to be returned. If we assume $s < r/2$ and $p \geq 2$, then recalling $\epsilon < s/(p(r-s))$ from (2.3), we can mitigate the potential for numerical issues by maintaining ϵ^2 to be sufficiently large, since $w_k(z_k(\mathbf{x}^{(j)}) - z_k^{\text{lb}}) \geq s/r \geq s/(p(r-s)) > \epsilon$. We therefore suggest monitoring the condition that $\epsilon s/r$ should be significantly larger than machine epsilon, and since $\epsilon s/r < s^2/(pr^2)$, this also bounds the granularity achievable using the MAWT norm.

The following theorem states that the evaluation of line 12 properly determines whether $B^{(j)} \cap Z_N = \emptyset$.

Theorem 2.4. Let $\mathbf{z}^{\text{lb}} < \mathbf{z}^{(j)} \leqq \mathbf{z}^{\text{ub}}$ and $\mathbf{x} \in X$. Then $\mathbf{z}(\mathbf{x}) \in B^{(j)}$ if and only if $g(\mathbf{z}(\mathbf{x})) < 1$.

Proof. Assume $g(\mathbf{f}(\mathbf{x})) < 1$. We know $\mathbf{z}^{\text{lb}} \leqq \mathbf{f}(\mathbf{x})$ by our construction of $\mathbf{z}^{\text{lb}}$, so we have:

$$\begin{aligned}
1 &> \max_{k \in P} \left\{ \frac{(z_k(\mathbf{x}) - z_k^{\text{lb}})}{(z_k^{(j)} - z_k^{\text{lb}})} \right\} + \epsilon \sum_{k \in P} \frac{(z_k(\mathbf{x}) - z_k^{\text{lb}})}{(z_k^{(j)} - z_k^{\text{lb}})} \\
&\geq \max_{k \in P} \left\{ \frac{(z_k(\mathbf{x}) - z_k^{\text{lb}})}{(z_k^{(j)} - z_k^{\text{lb}})} \right\} \\
\Rightarrow z_k^{(j)} - z_k^{\text{lb}} &> z_k(\mathbf{x}) - z_k^{\text{lb}} \quad \forall k \in P \\
&\Leftrightarrow \mathbf{z}^{(j)} > \mathbf{z}(\mathbf{x}), \text{ which guarantees } \mathbf{z}(\mathbf{x}) \in B^{(j)}.
\end{aligned}$$

Conversely, assume $\mathbf{z}(\mathbf{x}) \in B^{(j)}$ for some $\mathbf{x} \in X$. Then, since $\mathbf{z}(\mathbf{x}) \in B^{(j)}$ we know $\mathbf{z}(\mathbf{x}) < \mathbf{z}^{(j)}$. This gives us $\mathbf{z}^{\text{lb}} \leqq \mathbf{z}(\mathbf{x}) < \mathbf{z}^{(j)} \leqq \mathbf{z}^{\text{ub}}$, and so

$$\begin{aligned}
g(\mathbf{z}(\mathbf{x})) &= \max_{k \in P} \left\{ \frac{(z_k(\mathbf{x}) - z_k^{\text{lb}})}{(z_k^{(j)} - z_k^{\text{lb}})} \right\} + \epsilon \sum_{k \in P} \frac{(z_k(\mathbf{x}) - z_k^{\text{lb}})}{(z_k^{(j)} - z_k^{\text{lb}})} \\
&\leq \max_{k \in P} \left\{ \frac{(z_k^{(j)} - z_k^{\text{lb}}) - s}{(z_k^{(j)} - z_k^{\text{lb}})} \right\} + \epsilon \sum_{k \in P} \frac{(z_k^{(j)} - z_k^{\text{lb}}) - s}{(z_k^{(j)} - z_k^{\text{lb}})} \\
&\leq \max_{k \in P} \left\{ \frac{(z_k^{\text{ub}} - z_k^{\text{lb}}) - s}{(z_k^{\text{ub}} - z_k^{\text{lb}})} \right\} + \epsilon \sum_{k \in P} \frac{(z_k^{\text{ub}} - z_k^{\text{lb}}) - s}{(z_k^{\text{ub}} - z_k^{\text{lb}})} \\
&\leq \frac{r-s}{r} + \epsilon \, \frac{p(r-s)}{r} \\
&< \frac{r-s}{r} + \left(\frac{s}{p(r-s)} \right) \frac{p(r-s)}{r} = \frac{r-s}{r} + \frac{s}{r} = 1.
\end{aligned}$$

Thus $g(\mathbf{z}(\mathbf{x})) < 1$, which completes the proof. □

Corollary 2.4.1. If $g(\mathbf{z}(\mathbf{x}^{(j)})) < 1$ then $\mathbf{z}(\mathbf{x}^{(j)}) \in B^{(j)} \cap Z_N$. Otherwise $B^{(j)} \cap Z_N = \emptyset$.

Proof. We note that $g(\mathbf{z}(\mathbf{x}^{(j)})) \geq 1 \Leftrightarrow g(\mathbf{z}(\mathbf{x})) \geq 1, \ \forall \mathbf{x} \in X$. Proposition 2.1 and Theorem 2.4 then establish the corollary. □

Our next result considers how Algorithm 2.1 continues the search over $(B^{(j)} \cap Z_N) \setminus \{\mathbf{f}(\mathbf{x}^{(j)})\}$.

Lemma 2.5. Let $\mathbf{z}^{\text{lb}} < \mathbf{z}^{(j)} \leqq \mathbf{z}^{\text{ub}}$ for some iteration j, and $\mathbf{z} \in (B^{(j)} \cap Z_N) \setminus \{\mathbf{z}(\mathbf{x}^{(j)})\}$. Then $\exists \mathbf{z}^{(j),-k} = [z_1^{(j)}, \ldots, z_{k-1}^{(j)}, z_k(\mathbf{x}^{(j)}), z_{k+1}^{(j)}, \ldots, z_p^{(j)}]$ for some $k \in P$, such that $\mathbf{z}^{\text{lb}} < \mathbf{z}^{(j),-k} \leq \mathbf{z}^{\text{ub}}$ and

$\mathbf{z}^{(j),-k}$ is added to L in line 17.

Proof. Because $\mathbf{z} \in B^{(j)}$, we have $\mathbf{z} < \mathbf{z}^{(j)}$, and $\mathbf{z} \in Z_N$ implies $\exists k \in P$ such that $z_k^{\text{lb}} \leq z_k < z_k(\mathbf{x}^{(j)}) < z_k^{(j)}$. This satisfies the condition of line 15, so the point $\mathbf{z}^{(j),-k}$ is added to L. Moreover, $\mathbf{z}^{\text{lb}} \leq \mathbf{z} < \mathbf{z}^{(j),-k} \leq \mathbf{z}^{(j)} \leqq \mathbf{z}^{\text{ub}}$, and the proof is complete. □

As a result of Lemma 2.5, when a new efficient solution is identified in the algorithm, we generate "child" upper bounds of the form $\mathbf{z}^{(j),-k}$ and add them to L. We skip in line 17 any k for which $\mathbf{z}(\mathbf{x}^{(j)})$ is a k-preferred point, because that child, $\mathbf{z}^{(j')} = \mathbf{z}^{(j),-k}$ yields an empty corresponding search region, $B^{(j')} = \emptyset$. The requirement that $\mathbf{z}^{\text{lb}} < \mathbf{z}^{(j),-k} \leq \mathbf{z}^{\text{ub}}$ provides the sufficient conditions to apply Theorem 2.4 when the child is chosen in a future iteration.

We conclude this section with arguments on the finiteness and completeness of the returned sets $\widehat{Z}_N$ and $\widehat{X}_E$.

Proposition 2.2. Algorithm 2.1 terminates finitely, having the number of iterations bounded by $O(p^{|Z_N|})$.

Proof. Consider a tree graph structure, where each node corresponds to one iteration. The tree branches correspond to the child upper bounds $z^{(j),-k}$, for $k \in P$. Each node may have up to p branches; thus, the number of nodes at level of i of the tree is no more than p^i. To bound the tree depth, consider one $\mathbf{z}^{(j),-k} = \mathbf{z}^{(j')}$, and its associated search region $B^{(j')}$. Note $\mathbf{z}^{(j')} \leq \mathbf{z}^{(j)} \Rightarrow B^{(j')} \subset B^{(j)}$. Moreover, $\mathbf{z}(\mathbf{x}^{(j)}) \not< \mathbf{z}^{(j')} \Rightarrow \mathbf{z}(\mathbf{x}^{(j)}) \notin B^{(j')} \Rightarrow (B^{(j+1)} \cap Z_N) \subset (B^{(j)} \cap Z_N)$. Therefore, on every path from the root, the search regions diminish by at least one non-dominated point at each node. Hence, no path from the root node to a leaf can contain more than $|Z_N| + 1$ nodes, and so the tree depth is bounded by $|Z_N| + 1$. The total number of nodes in the full tree is then bounded by $\sum_{i=0}^{|Z_N|} p^i$, i.e., $O(p^{|Z_N|})$ nodes. This completes the proof. □

Remark 2. The bounds generated in Proposition 2.2 are loose and do not reflect the state-of-the-art. This is due to the simple decomposition of the search elements in Algorithm 2.1. As we find in our computational experiments, Algorithm 2.1 solves many more IPs than other recent generating algorithms. Better results may be obtained using more advanced decomposition schemes such as those of [44] and [18]. We discuss these improvements further in Section 2.5.3. The decomposition proposed in Algorithm 2.1, however, is simpler for our initial exposition and sufficient for the proof of finiteness.

Theorem 2.6. At termination, Algorithm 2.1 returns $\widehat{Z}_N = Z_N$.

Proof. To show $\widehat{Z}_N \subseteq Z_N$, note that only solutions to (2.5) are added to $\widehat{Z}_N$ in line 13, and by Proposition 2.1, those solutions are non-dominated. To show that $Z_N \subseteq \widehat{Z}_N$, consider some $\mathbf{z} \in Z_N$, and assume that $\mathbf{z} \in B^{(j)}$ for some $j \in \mathbb{N}$. Since $B^{(j)} \cap Z_N \neq \emptyset$, by Corollary 2.4.1, $g(\mathbf{z}(\mathbf{x}^{(j)})) < 1$, meeting the condition of line 12. Now we have two cases:

Case 1: $\mathbf{z} = \mathbf{z}(\mathbf{x}^{(j)})$: Then line 13 guarantees $\mathbf{z} \in \widehat{Z}_N$.

Case 2: $\mathbf{z} \neq \mathbf{z}(\mathbf{x}^{(j)})$: Then from Lemma 2.5, we know some $\mathbf{z}^{(j')} = \mathbf{z}^{(j),-k}$ is added to L in line 17, where $\mathbf{z} < \mathbf{z}^{(j')} \leq \mathbf{z}^{\text{ub}} \Rightarrow \mathbf{z} \in B^{(j')}$ and the algorithm continues.

Observe from these cases that Algorithm 2.1 cannot terminate unless $\mathbf{z} = \mathbf{z}(\mathbf{x}^{(j)})$ for some j. Then because Proposition 2.2 shows the algorithm must terminate, we have $\mathbf{z} \in \widehat{Z}_N$ at termination, and the proof is complete. $\square$

Note that every time an element $\mathbf{z}(\mathbf{x}^{(j)})$ is added to $\widehat{Z}_N$ in line 13, its pre-image $\mathbf{x}^{(j)}$ is also added to $\widehat{X}_E$, which guarantees by Theorem 2.6 that $\widehat{X}_E$ is a complete efficient set at termination.

2.4 Example

In this section, we illustrate Algorithm 2.1 on the following bounded, non-convex bi-objective problem:

$$\begin{aligned}
\min \quad & [z_1(\mathbf{x}) = x_1,\ z_2(\mathbf{x}) = x_2] \\
\text{subject to:} \quad & (x_1, x_2) \in (X_1 \cup X_2) \cap \mathbb{Z}^2, \text{ where} \\
& X_1 = \left\{ (x_1, x_2) \in \mathbb{R}^2 : \sqrt{\frac{(x_1 - 7)^2}{3} + \frac{(x_2 - 2)^2}{1.5}} \leq 1 \right\} \\
& X_2 = \left\{ (x_1, x_2) \in \mathbb{R}^2 : \frac{3}{2}x_1 + x_2 \geq 9,\ -\frac{2}{5}x_1 - x_2 \geq -8,\ x_2 \geq 4 \right\}.
\end{aligned}$$

A lower bound on the step size is $s = 1$. A strict upper bound on Z_N is $\mathbf{z}^{\text{ub}} = [11, 8]$.

In Figure 2.2 we show the regions X_1 and X_2 as shaded. The point $\mathbf{z}^{\text{lb}}$ is marked with a star. Dominated points are dull circles, non-dominated points are bold circles, and the non-dominated solution to (2.5) in each iteration, $\mathbf{x}^{(j)}$, is a filled circle. Also for each iteration, the upper bound, $\mathbf{z}^{(j)}$, is a filled diamond, and the child upper bounds spawned from that iteration, $\mathbf{z}^{(j),-k}$, are empty diamonds. Finally, for each iteration we show the level curves $\{\mathbf{z} \in \mathbb{R}^p : g(\mathbf{z}) = g(\mathbf{z}(\mathbf{x}^{(j)}))\}$ (solid line) and $\{\mathbf{z} \in \mathbb{R}^p : g(\mathbf{z}) = 1\}$ (dashed line).

Initialization: (See Figure 2.2a.) We initialize by first solving for the lower bound, $\mathbf{z}^{\text{lb}} = (2, 1)$. Then, we choose $\epsilon = 0.05 < 1/16 = s/p(r - s)$. Finally, we initialize $L = \{(11, 8)\}$.

Iteration 1: (See Figure 2.2b.) We choose $\mathbf{z}^{(1)} = (11, 8)$ and remove this vector from L. Solving (2.5), we get $\mathbf{x}^{(1)} = \mathbf{z}(\mathbf{x}^{(1)}) = (4, 2)$ with $g(\mathbf{z}(\mathbf{x}^{(1)})) \approx 0.2405$. Since $g(\mathbf{z}(\mathbf{x}^{(1)})) < 1$, we add $\mathbf{x}^{(1)}$ to our solution set, so that $\widehat{X}_E = \widehat{Z}_N = \{(4, 2)\}$. Next, we find the child upper bounds $\mathbf{z}^{(1),-1} = (4, 8)$ and $\mathbf{z}^{(1),-2} = (11, 2)$ and add them to $L = \{(4, 8), (11, 2)\}$.

Iteration 2: (See Figure 2.2c.) We choose $\mathbf{z}^{(2)} = (4, 8)$ and remove this vector from L. Solving (2.5), we get $\mathbf{x}^{(2)} = \mathbf{z}(\mathbf{x}^{(2)}) = (3, 5)$ with $g(\mathbf{z}(\mathbf{x}^{(2)})) = 0.625$. Since $g(\mathbf{z}(\mathbf{x}^{(2)})) < 1$, we add $\mathbf{x}^{(2)}$ to our solution set, so that $\widehat{X}_E = \widehat{Z}_N = \{(3, 5), (4, 2)\}$. Next, we find the child upper bounds $\mathbf{z}^{(2),-1} = (3, 8)$ and $\mathbf{z}^{(2),-2} = (4, 5)$ and add them to $L = \{(11, 2), (3, 8), (4, 5)\}$.

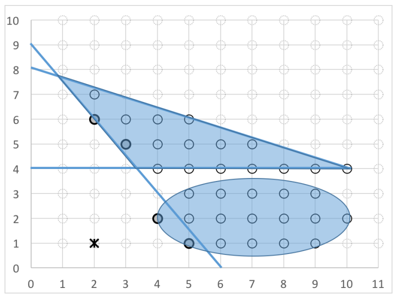
Iteration 3: (See Figure 2.2d.) We choose $\mathbf{z}^{(3)} = (3, 8)$ and remove this vector from L. Solving (2.5), we get $\mathbf{x}^{(3)} = \mathbf{z}(\mathbf{x}^{(3)}) = (2, 6)$ with $g(\mathbf{z}(\mathbf{x}^{(3)})) = 0.75$. Since $g(\mathbf{z}(\mathbf{x}^{(3)})) < 1$, we add $\mathbf{x}^{(3)}$ to our solution set, so that $\widehat{X}_E = \widehat{Z}_N = \{(2, 6), (3, 5), (4, 2)\}$. Note $\mathbf{z}(\mathbf{x}^{(3)})$ is 1-preferred because $z_1(\mathbf{x}^{(3)}) = z_1^{\text{lb}}$, so we do not generate a 1-child for $\mathbf{z}^{(3)}$. However, we still add the 2-child, $\mathbf{z}^{(3),-2} = (3, 6)$, to $L = \{(11, 2), (4, 5), (3, 6)\}$.

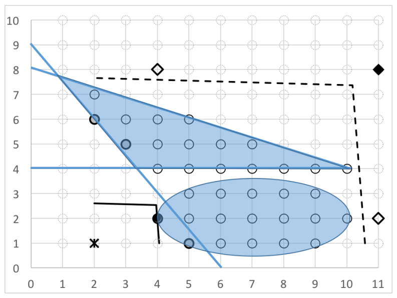
Iteration 4: (See Figure 2.2e.) We choose $\mathbf{z}^{(4)} = (3, 6)$ and remove this vector from L. Solving (2.5), we get $\mathbf{x}^{(4)} = \mathbf{z}(\mathbf{x}^{(4)}) = (2, 6)$ with $g(\mathbf{z}(\mathbf{x}^{(4)})) = 1.05$. Since $g(\mathbf{z}(\mathbf{x}^{(4)})) > 1$, we move on to the next iteration, selecting a new upper bound from $L = \{(11, 2), (4, 5)\}$.

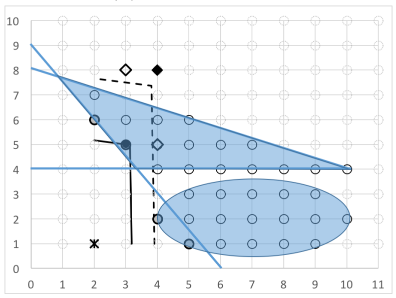
Iteration 5: (See Figure 2.2f.) We choose $\mathbf{z}^{(5)} = (4, 5)$ and remove this vector from L. Solving (2.5), we get $\mathbf{x}^{(5)} = \mathbf{z}(\mathbf{x}^{(5)}) = (4, 2)$ with $g(\mathbf{z}(\mathbf{x}^{(5)})) = 1.0625$. Since $g(\mathbf{z}(\mathbf{x}^{(5)})) > 1$, we move on to the next iteration, selecting a new upper bound from $L = \{(11, 2)\}$.

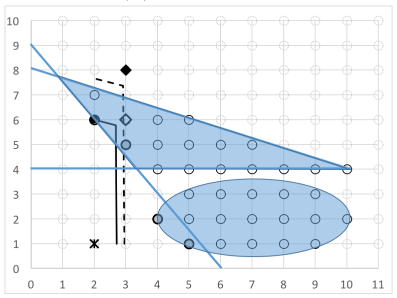
Iteration 6: (See Figure 2.2g.) We choose $\mathbf{z}^{(6)} = (11, 2)$ and remove this vector from L. Solving (2.5), we get $\mathbf{x}^{(6)} = \mathbf{z}(\mathbf{x}^{(6)}) = (5, 1)$ with $g(\mathbf{z}(\mathbf{x}^{(6)})) = 0.35$. Since $g(\mathbf{z}(\mathbf{x}^{(6)})) < 1$, we add $\mathbf{x}^{(6)}$ to our solution set, so that $\widehat{X}_E = \widehat{Z}_N = \{(2, 6), (3, 5), (4, 2), (5, 1)\}$. Because $z_2(\mathbf{x}^{(6)}) = z_2^{\text{lb}}$, we add only $\mathbf{z}^{(6),-1} = (5, 2)$ to $L = \{(5, 2)\}$.

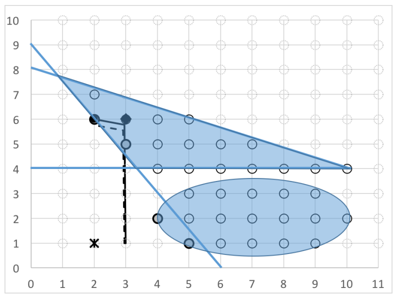
Iteration 7: (See Figure 2.2h.) We choose $\mathbf{z}^{(7)} = (5, 2)$ and remove this vector from L. Solving (2.5), we get $\mathbf{x}^{(7)} = \mathbf{z}(\mathbf{x}^{(7)}) = (5, 1)$ with $g(\mathbf{z}(\mathbf{x}^{(7)})) = 1.05$. Since $g(\mathbf{z}(\mathbf{x}^{(7)})) > 1$ and L is empty, the algorithm terminates, returning $\widehat{X}_E = \widehat{Z}_N = \{(2, 6), (3, 5), (4, 2), (5, 1)\}$.

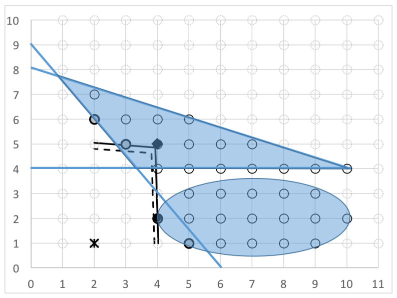

(a) Initialization

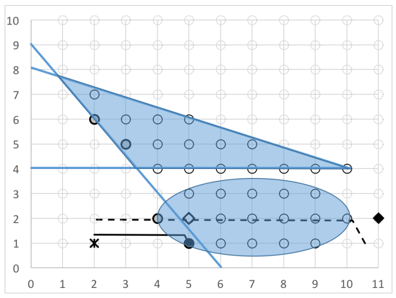

(b) Iteration 1


(c) Iteration 2


(d) Iteration 3


(e) Iteration 4


(f) Iteration 5


(g) Iteration 6

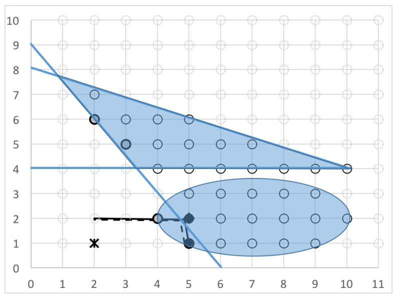

(h) Iteration 7

Figure 2.2: Illustration of Algorithm 2.1 iterations for Section 2.4 example problem

Table 2.1: $|Z_N|$ for instances in our testbed.

		Instance number									
p	n	1	2	3	4	5	6	7	8	9	10
3	10 (KP)	10	19	18	22	6	27	15	14	8	14
	25 (AP)	20	22	12	10	10	22	11	6	9	15
4	8 (KP)	16	22	12	13	54	17	21	11	19	13
	9 (AP)	3	3	3	2	6	5	5	4	6	4
	10 (KP)	27	45	22	57	47	77	15	31	32	64
	20 (KP)	110	160	664	611	482	708	451	328	197	372
	25 (AP)	31	26	43	25	35	23	32	28	16	17
	64(AP)	185	212	292	389	264	310	362	190	254	141
	100 (AP)	1,359	1,186	1,043	310	524	546	999	1,378	500	837
5	10 (KP)	76	96	57	175	48	86	72	49	82	59
	25 (AP)	24	36	73	60	52	35	81	18	35	41
6	10 (KP)	101	115	97	166	153	91	124	102	45	86
	25 (AP)	67	66	49	76	38	75	48	50	46	67

2.5 Computational Study

2.5.1 Implementation

We conducted an empirical study to compare the efficiency of searches using the MAWT norm versus those performed by recently published algorithms. For this study, we refer to our implementation of Algorithm 2.1 as "BOX." For comparison, we implemented the ϵ-constraint algorithm of [41], which we refer to as "KS," and we implemented the recursive ϵ-constraint algorithm of [61], which we refer to as "OBM." Our implementations were written in Python 2.7, and we used Gurobi 7.5 to solve the scalarized sub-problems, $\min_{\mathbf{x}\in X} g(\mathbf{z}(\mathbf{x}))$, in each iteration.

For the study we ran 60 multi-objective cardinality-constrained knapsack and 70 multi-objective assignment problem instances. The number of non-dominated objectives for these 130 instances is given in Table 2.1.

We generated the knapsack problems by varying the number of binary decision variables $n \in \{8, 10, 20\}$ while holding the number of objectives $p = 4$, and then we varied the number of objectives $p \in \{3, 5, 6\}$ while holding decision variables $n = 10$. For each of the six (n, p) combinations we generated 10 instances, giving 60 instances in total. Our knapsack problems were formulated as:

$$\begin{array}{rl} \min & [-\mathbf{w}_1^T\mathbf{x}, \ldots, -\mathbf{w}_p^T\mathbf{x}] \\ \text{subject to:} & \displaystyle\sum_{i=1}^{n} x_i \leq n/2 \end{array}$$

$$\mathbf{x} \in \{0,1\}^n,$$

where each $\mathbf{w}_k \in \{0,\dots,10\}^n$ for $k \in P$ was randomly generated using a discrete uniform distribution. To initialize BOX, we used $\mathbf{z}^{\text{ub}} = \mathbf{e} = [1,\dots,1]$, and $s = 1$, and we chose $\epsilon = 1/pr < s/(p(r-s))$. Observing that the minimum possible objective in any component is $-10n/2$, our ϵ parameter was smallest on the instances where $p = 4$, $n = 20$, which give $\epsilon \geq 1/(4 * (-1 - (-10 * 10))) = 1/396$. Thus, we easily avoided numerical stability issues for our knapsack problems.

For our assignment problems we used complete bipartite graphs having node set $N_1 \cup N_2$, with $|N_1| = |N_2|$. We varied the number of nodes over $|N_1| \in \{3,5,8,10\}$ while holding $p = 4$. Since the number of arcs in a complete bipartite graph is given by $|N_1|^2$, this varied the number of decision variables over $n \in \{9,25,64,100\}$ with $p = 4$. We then varied the number of objectives $p \in \{3,5,6\}$ while holding $n = 25$ (i.e., $|N_1| = 5$). Again, for each (n,p) combination, we generated 10 instances, resulting in a total of 70 instances. Our assignment problems were formulated as:

$$\begin{aligned}
\min \quad & [\mathbf{w}_1^T\mathbf{x},\dots,\mathbf{w}_p^T\mathbf{x}] \\
\text{subject to :} \quad & \sum_{j \in N_2} x_{ij} = 1 \quad \forall i \in N_1 \\
& \sum_{i \in N_1} x_{ij} = 1 \quad \forall j \in N_2 \\
& x_{ij} \in \{0,1\} \quad \forall (i,j) \in N_1 \times N_2,
\end{aligned}$$

where each $\mathbf{w}_k \in \{0,\dots,10\}^n$ for $k \in P$ was randomly generated using a discrete uniform distribution. For the assignment problems, our upper bound input was $\mathbf{z}^{\text{ub}} = 10np\mathbf{e}$, and $s = 1$. As before, we chose $\epsilon = 1/pr$. Since zero is a lower bound on any component objective, we can form a lower theoretical bound on ϵ over our test suite using instances where $p = 6$ and $n = 10$, which give $\epsilon \geq 1/(6 * (10 * 10 * 6 - 0)) = 1/3600$, which again was large enough to ensure that we did not encounter numerical stability issues.

We executed each algorithm five times on each instance and recorded data on the fastest total computational time observed. The study was run on an Intel Core i5 1.8GHz processor with 4GB of 1600MHz DDR3 RAM. We imposed an 1800 second time limit, after which the algorithm terminated and returned the partial non-dominated set, $\widehat{Z}_N$, obtained at termination.

2.5.2 Computational Results

The first measure of merit in our experiment was how quickly Gurobi solved the integer programming (IP) sub-problems for each algorithm. Since KS and OBM employed lexicographic preferences to find each non-dominated point, each IP required multiple calls to Gurobi: OBM required p calls per IP, and KS required two. We scored these multiple calls as solving a single IP, summing Gurobi's solving time over the multiple calls to get the total time to solve the IP sub-problem. Our experiment indicates that Gurobi consistently solved KS's sub-problems faster than either BOX's or OBM's (see Figure 2.3). The times were comparable, however, with differences usually less than one order of magnitude. Usually, Gurobi solved OBM's sub-problems in less time than it did for BOX, but as n grew large this trend reversed. Gurobi's time to solve BOX's sub-problems became increasingly comparatively slower than the other two as p increased.

Surprisingly, however, the faster computation of sub-problems did not always translate to faster time to termination (see Table 2.2). Figure 2.4 shows the performance profiles for each algorithm, depicting the cumulative percentage of problems solved by each algorithm within time limits appearing on the horizontal axis. Figure 2.4a depicts these performance profiles on a linear time axis, which demonstrates that our proposed BOX algorithm outperforms both KS and OBM over the aggregated set of test instances we generated. However, this plot hides the behavior of the algorithms when they terminate relatively quickly. Figure 2.4b shows the performance profiles on a logarithmic time axis instead, revealing that these both outperform BOX on instances that require about ten seconds or less of computational time. Figure 2.5 provides further insight to this trend, showing that BOX underperformed when $p \leq 4$, but tended to be faster when $p \geq 5$.

2.5.3 Running Time Improvement

Our study demonstrated that using the MAWT norm as implemented in Algorithm 2.1 is comparable with two recently published algorithms and that it shows improved performance as the number of objectives increases. However, in comparing the number of IPs solved (see Table 2.3), it is clear that the BOX algorithm still has not reached its fullest potential, since it solves many more IPs than necessary (see Remark 2). Klamtroth et al. [44] describe two techniques to avoid such redundant searches: redundancy elimination (RE) and redundancy avoidance (RA). Upon inspection, it may be seen that both KS and OBM implement RE variants. Dächert et al. provide an RA routine

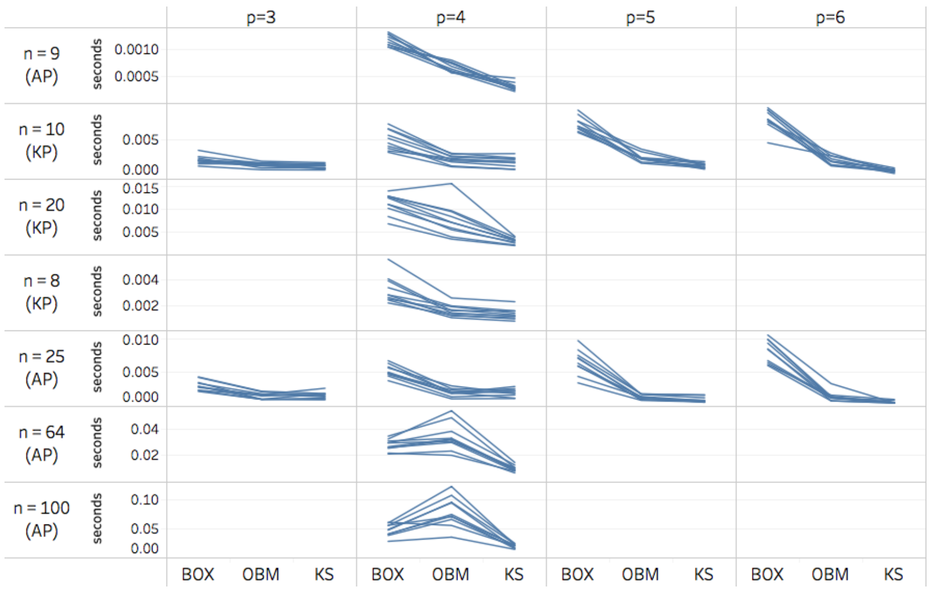
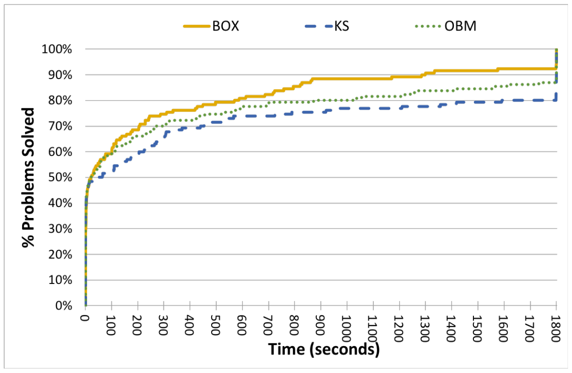


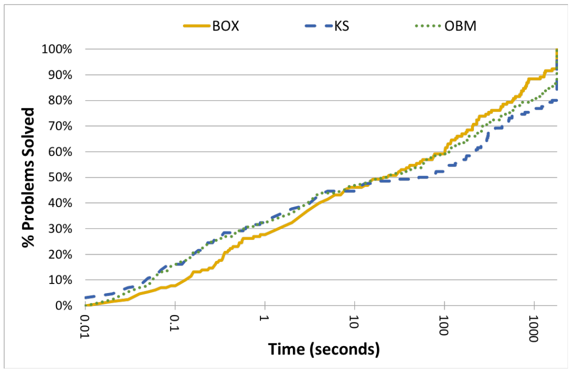
Figure 2.3: Average time for Gurobi to solve one IP sub-problem in our testbed. Each line represents an average of solver times over all IP sub-problems for one problem instance.

Table 2.2: Summary of computation times. All times are in seconds and significant to the nearest second.

		BOX			KS			OBM		
p	n	Min	Median	Max	Min	Median	Max	Min	Median	Max
3	10 (KP)	< 0.5	< 0.5	1	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
	25 (AP)	< 0.5	< 0.5	1	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
4	8 (KP)	< 0.5	1	8	< 0.5	< 0.5	10	< 0.5	< 0.5	2
	9 (AP)	0.51	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
	10 (KP)	1	4	15	< 0.5	2	13	< 0.5	2	13
	20 (KP)	23	392	867	15	281	474	16	116	572
	25 (AP)	1	4	9	1	2	4	< 0.5	2	8
	64(AP)	204	790	1800	134	262	790	44	206	457
	100 (AP)	862	1,800	1,800	307	1,800	1,800	531	1,530	1,800
5	10 (KP)	8	55	286	27	213	1,356	12	82	899
	25 (AP)	2	36	241	5	109	1800	3	85	873
6	10 (KP)	20	236	1,576	920	1,800	1,800	55	1,766	1,800
	25 (AP)	75	184	1,299	1,800	1,800	1,800	261	1,800	1,800

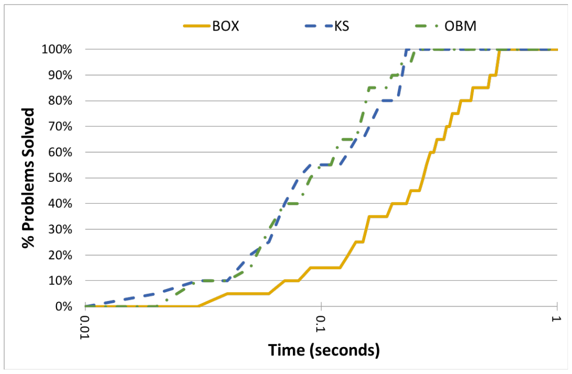


(a) Performance profiles with linear time axis.

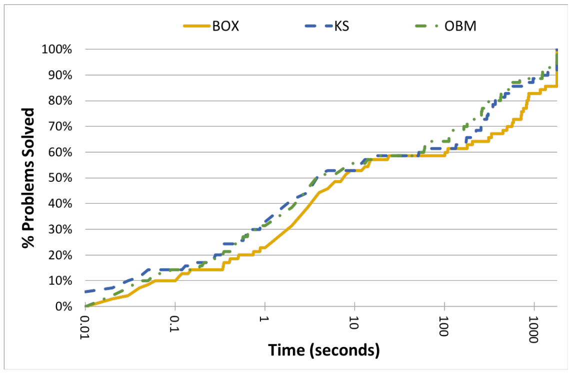


(b) Performance profiles with logarithmic time axis.

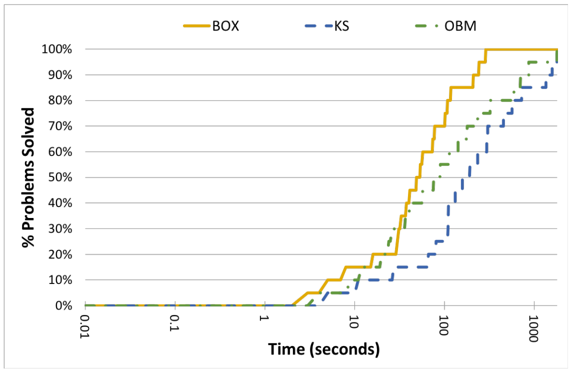
Figure 2.4: Performance profiles for algorithms.



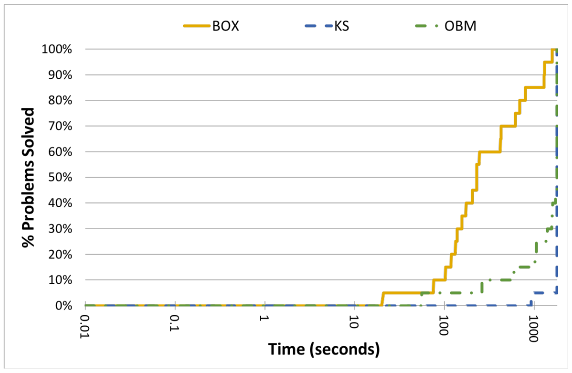
(a) Performance profiles for $p = 3$.



(b) Performance profiles for $p = 4$.



(c) Performance profiles for $p = 5$.



(d) Performance profiles for $p = 6$.

Figure 2.5: Performance profiles for algorithms by p (logarithmic time axis).

Table 2.3: Number of IPs solved for each algorithm on completed problem instances in our testbed. “N/A” indicates the algorithm failed to complete a sufficient number of problem instances to compute the summary statistic.

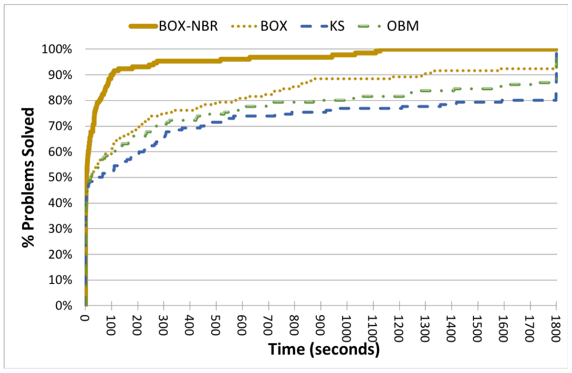
		BOX			KS			OBM		
p	n	Min	Median	Max	Min	Median	Max	Min	Median	Max
3	10 (KP)	14	69	138	8	35	66	12	34	54
	25 (AP)	19	57	106	12	28	66	14	25	43
4	8 (KP)	91	185	1,064	36	69	414	51	91	252
	9 (AP)	7	21	64	5	14	27	9	18	32
	10 (KP)	235	572	1,758	57	169	497	77	186	399
	20 (KP)	2,522	24,815	44,057	403	2,335	3,586	462	1,605	2,751
	25 (AP)	214	449	962	85	182	306	90	185	301
	64(AP)	5,936	17,844	27,755	1,226	1,999	3,420	786	1,534	2,127
	100 (AP)	17,265	N/A	N/A	1,480	4,336	4,918	1,815	2,822	3,430
5	10 (KP)	1,036	6,666	26,855	370	998	3,302	585	1,412	3,152
	25 (AP)	419	3,781	19,242	159	735	2,973	294	1,296	4,082
6	10 (KP)	3,189	26,131	147,968	745	N/A	N/A	1,117	6,891	7,632
	25 (AP)	9,091	17,944	96,117	N/A	N/A	N/A	2,799	5,306	6,560

for managing search regions using a neighbor relationship between the search region upper bounds. Dächert et al. prove that if all solutions are in general position (i.e., no two solutions have objectives equal in any component) their routine complexity is bounded in the order of the size of the upper bounding set over Z_N. In our testbed, the general position assumption does not hold, and while Dächert et al. describe how to modify their routine for the non-general-position case, they provide no complexity results for such a scenario. We implemented the list management routine of Dächert et al., still using the MAWT norm to identify non-dominated objectives in the search regions. We call resulting algorithm “BOX-NBR.”

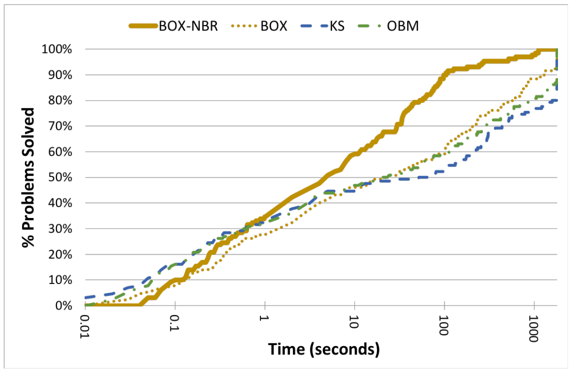
After implementing these improvements, we tested BOX-NBR on the same instances from our computational study. The results demonstrate marked improvement over the base routine (see Table 2.4). Over the entire testbed, BOX-NBR exhibited an average of 64.0% improvement over BOX running times; moreover, the running time reductions tended to increase as p increased. Overall, BOX-NBR significantly outperformed all three of the previous algorithms over the comprehensive testbed, but was slower than KS and OBM on instances taking less than 1 second (see Figure 2.6). As before, the strength of the MAWT norm becomes most apparent as the number of objectives is increased (see Figure 2.7). Notably, of the four algorithms tested, BOX-NBR was the only one to successfully solve all testbed instances within 1800 seconds. Nevertheless, we also observe that BOX-NBR still required more IP sub-problems than OBM on a number of problems (see Table 2.4).

Table 2.4: Running times (in seconds) and number of IPs solved by BOX-NBR along with respective average percent improvement over BOX.

		Running Time				IP's Solved			
p	n	Min	Median	Max	% Improved	Min	Median	Max	% Improved
3	10 (KP)	< 0.5	< 0.5	< 0.5	-19.5%	29	51	90	-19.7%
	25 (AP)	< 0.5	< 0.5	< 0.5	-30.9%	25	44	78	-25.3%
4	8 (KP)	< 0.5	< 0.5	3	-55.7%	67	98	321	-57.6%
	9 (AP)	< 0.5	< 0.5	< 0.5	+8.3%	28	36	49	+25.8%
	10 (KP)	< 0.5	2	5	-65.5%	88	212	450	-70.8%
	20 (KP)	8	51	104	-87.0%	563	2,549	4,571	-89.2%
	25 (AP)	1	1	3	-65.7%	96	166	243	-69.0%
	64(AP)	36	88	180	-89.4%	792	1,583	2,766	-91.8%
	100 (AP)	110	572	1,128	-67.0%	1,815	5,844	9,017	-82.8%
5	10 (KP)	1	5	20	-88.6%	479	825	2,934	-88.9%
	25 (AP)	1	5	20	-88.6%	143	424	1,196	-90.9%
6	10 (KP)	5	35	89	-91.5%	546	2,724	6,097	-93.7%
	25 (AP)	9	23	70	-93.1%	751	1,593	3,843	-94.7%



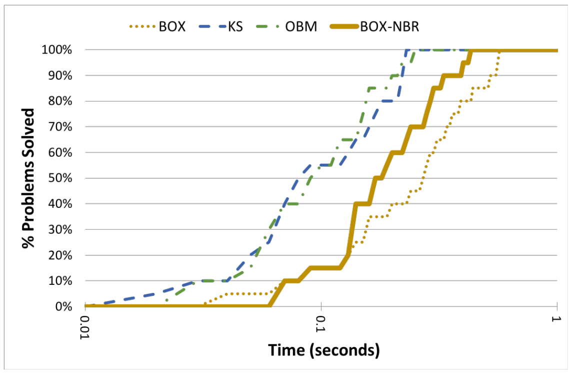
(a) Performance profiles with linear time axis.



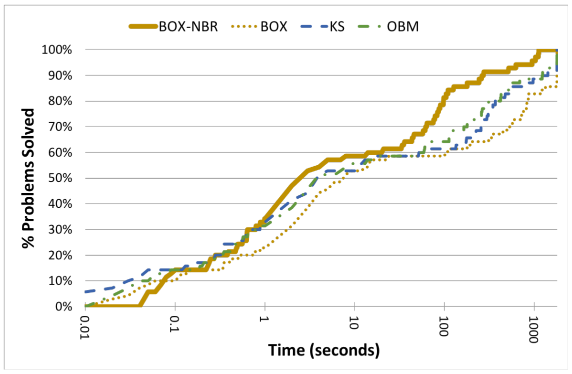
(b) Performance profiles with logarithmic time axis.

Figure 2.6: Performance profiles for BOX-NBR (previous algorithm profiles are given as dotted lines for comparison).

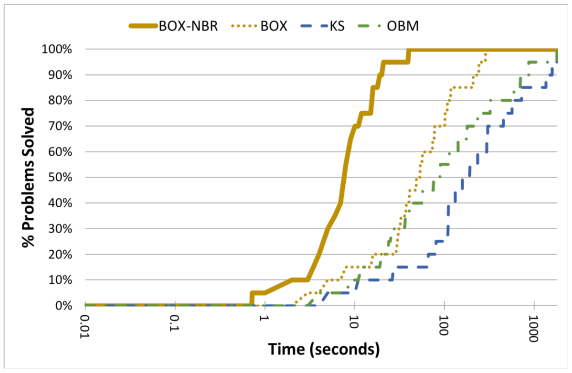
This is likely because the OBM routine, as an ϵ-constraint method, conducts its search in $\mathbb{R}^{p-1}$ space rather than $\mathbb{R}^p$, and it is reasonable to expect that fewer search regions are required for a full search in the lower-dimensional space.



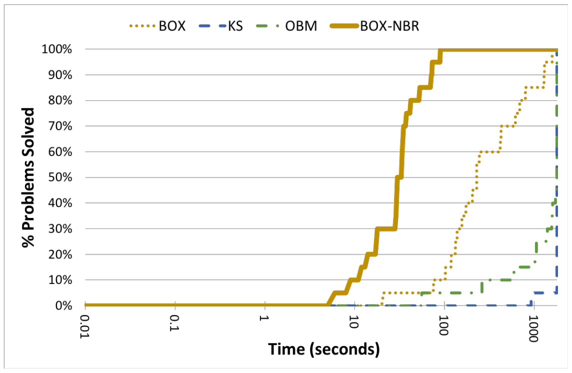
(a) Performance profiles for $p = 3$.



(b) Performance profiles for $p = 4$.



(c) Performance profiles for $p = 5$.



(d) Performance profiles for $p = 6$.

Figure 2.7: Performance profiles for BOX-NBR by p (logarithmic time axis, previous algorithm profiles are given for comparison).

2.6 Conclusion

We contribute the MAWT norm and an associated algorithm, which, under fairly general assumptions, generates the entire non-dominated set for a multi-objective optimization problem having any number of objectives. Numerical experiments suggest that the MAWT norm yields scalarized sub-problems that may be more difficult to solve than comparable ϵ-constraint sub-problems. However, the algorithm still produced comparable computational results to two recently published algorithms. In particular, our computational investigation showed that the proposed BOX algorithm is preferable to the ϵ-constraint algorithms when the number of objectives was relatively large (more than four). However, our testbed includes only a limited set of problems, none of which are NP–hard in the single linear objective case. We also consider an improvement to our algorithm leveraging the work of [18] and [44] and show significant reductions in running time by avoiding redundant searches.

Our observations motivate further computational study of various scalarization-based generating methods, including ϵ-constraint methods, Klein and Hannan's family of methods, and the MAWT norm. Each of these methods can be integrated with advanced approaches for managing the list of search regions and avoiding redundant searches. The integration of various scalarization methods with the diverse list management and redundancy mitigation schemes in the literature warrants further attention. Future studies may also examine the performance of these combinations on a variety of problem classes, including nonlinear, non-convex, and ill-conditioned problems.

Chapter 3

The Shortest Path Interdiction Problem with Arc Improvement Recourse

3.1 Problem description

The SPIP with arc improvement (SPIP-I) is a SPIP variant where the follower has an opportunity to improve arcs, resulting in lower arc costs. That is, instead of the usual arc cost functions, $c_a : \{0, \dots, b\} \to \mathbb{R}_+$, the SPIP-I uses the non-negative cost function, $c_a(x_a^L, x_a^F)$, where $x_a^L, x_a^F \in \mathbb{Z}_+$ represent the leader's and follower's respective efforts to interdict or improve arc a. The marginal cost functions are non-decreasing over x_a^L and non-increasing over x_a^F. The follower's objective is to select an improvement strategy, represented by variables $\mathbf{x}^F$, and a path, represented by variables $\mathbf{y}$, that minimize the cost to travel from a source node, $s \in \mathcal{N}$, to a sink node, $t \in \mathcal{N}$. The leader's objective is to select an interdiction strategy, $\mathbf{x}^L$, that maximizes the follower's minimum cost. The leader's interdiction effort is constrained by $\sum_{a \in \mathcal{A}} x_a^L \leq b^L$, and similarly the total amount of improvement is constrained by $\sum_{a \in \mathcal{A}} x_a^F \leq b^F$, where b^L and b^F are non-negative integers. As we will later show, the cardinality restriction on the *leader's* budget can easily be generalized as a knapsack constraint, but significant computational difficulties can result by changing the *follower's* budget constraint to the general knapsack form.

As in the traditional SPIP formulation, each player acts with full knowledge of the other's actions, taking turns in three sequential moves as follows:

Move 1: The leader chooses an interdiction strategy, $\mathbf{x}^L \in \mathbb{Z}_+^{|\mathcal{A}|}$, with $\sum_{(i,j)\in\mathcal{A}} x_{ij}^L \leq b^L$.

Move 2: The follower chooses an improvement strategy, $\mathbf{x}^F \in \mathbb{Z}_+^{|\mathcal{A}|}$, with $\sum_{(i,j)\in\mathcal{A}} x_{ij}^F \leq b^F$.

Move 3: Finally, the follower selects a shortest path from s to t, represented by $\mathbf{y} \in \{0,1\}^{|\mathcal{A}|}$, where $y_{ij} = 1$ if arc (i,j) is on the path, and $y_{ij} = 0$ otherwise.

Both players are aware of all data and previous decisions in the SPIP-I, with the exception that the leader is unaware of the improvement budget, b^F. Here, the leader is only aware that $b^F \in \{0,\dots,p\}$ for some $p \in \mathbb{Z}_+$. Parameterizing the objective function by b^F, we formulate the single-objective SPIP-I as:

$$
\begin{aligned}
&\max_{\mathbf{x}^L} && z_{b^F}(\mathbf{x}^L) && && (3.1a)\\
&\text{subject to:} && \sum_{a\in\mathcal{A}} x_a^L \leq b^L && && (3.1b)\\
& && x_a^L \in \mathbb{Z}_+ && \forall a \in \mathcal{A}, && (3.1c)\\
\text{where } z_{b^F}(\mathbf{x}^L) = &\min_{(\mathbf{x}^F,\mathbf{y})} && \sum_{a\in\mathcal{A}} c_a(x_a^L, x_a^F) y_a && && (3.1d)\\
&\text{subject to:} && \sum_{\substack{j\in\mathcal{N}:\\(i,j)\in\mathcal{A}}} y_{ij} - \sum_{\substack{j\in\mathcal{N}:\\(j,i)\in\mathcal{A}}} y_{ji} = \begin{cases} 1 & i = s\\ -1 & i = t\\ 0 & \text{otherwise}\end{cases} && \forall i \in \mathcal{N} && (3.1e)\\
& && \sum_{a\in\mathcal{A}} x_a^F \leq b^F && && (3.1f)\\
& && y_a \in \{0,1\} && \forall a \in \mathcal{A} && (3.1g)\\
& && x_a^F \in \mathbb{Z}_+ && \forall a \in \mathcal{A}. && (3.1h)
\end{aligned}
$$

Formulation (3.1) is equivalent to the SPIP formulation from (1.1), with the addition of constraints (3.1f) and (3.1h), which restrict the follower's improvements.

The leader's ideal interdiction strategy would be simultaneously optimal for all values $b^F \in \{0,\dots,p\}$. However, this is not always possible (see Example 3.1). Often an interdiction strategy that is relatively effective for one value of b^F may be relatively ineffective for another value,

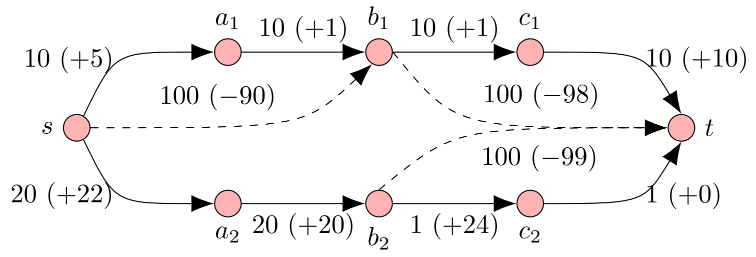


Figure 3.1: Graph for Example 3.1. Solid arcs are candidates for interdiction, and dotted arcs are candidates for improvement. Arc costs are given by B_{ij} $(\pm\delta_{ij})$ where B_{ij} is the arc cost without interdiction or improvement and $\pm\delta_{ij}$ is the increase or decrease in cost due to a single interdiction or improvement. Additional interdictions or improvements have no further effect.

and vice-versa. A robust approach would consider only the case where $b^F = p$, but this approach may miss certain strategies that yield near-optimal degradation in the case that $b^F = p$ while giving significant improvement in the case that $b^F < p$. In contrast, our multi-objective approach creates a separate objective function for each $b^F \in \{0, \dots, p\}$ and seeks to generate a set of strategies where, for any strategy, it is not possible to improve one objective without degrading another. This allows us to consider the full range of trade-off options, and remains computationally feasible when the range of possible values for b^F is limited. Our multiobjective SPIP-I is defined as:

$$\begin{aligned} \max_{\mathbf{x}^L} \quad & [z_0(\mathbf{x}^L), \dots, z_p(\mathbf{x}^L)] \\ \text{subject to: } & \text{(3.1b)–(3.1c)}, \end{aligned} \tag{3.2}$$

where $z_{b^F}(\mathbf{x}^L)$ is defined as in (3.1d)–(3.1h). Let $X = \{\mathbf{x}^L \in \mathbb{Z}_+^{|\mathcal{A}|} : \sum_{(i,j)\in\mathcal{A}} x_{ij}^L \leq b^L\}$, and let $Z = \bigcup_{\mathbf{x}^L \in X} \{\mathbf{z}(\mathbf{x}^L)\} \subset \mathbb{R}^{p+1}$. Throughout this chapter, except where specified otherwise, we refer by default to SPIP-I as the multiobjective version of the problem, and we seek to find a complete set of non-dominated solutions $\mathbf{z}(\mathbf{x}) \subseteq Z_N$ (using the definitions of efficiency and non-dominance from the previous chapter).

Example 3.1. Throughout this chapter, we illustrate the SPIP-I by the graph given in Figure 3.1, with the arc cost functions, $c_{ij}(\mathbf{x}^L, \mathbf{x}^F)$, given in Table 3.1. We set $b^L = 3$ and $p = 1$ (i.e., $b^F \in \{0, 1\}$).

The cost functions in this example effectively partition the arcs into two sets: candidates for improvement and candidates for interdiction. Expending improvement budget on candidates for interdiction will have no effect, nor will expending interdiction budget on candidates for improvement.

Table 3.1: Cost functions, $c_{ij}(\mathbf{x}^L, \mathbf{x}^F)$, for arcs in Figure 3.1.

Candidates for interdiction			Candidates for improvement		
(i,j)	$x^L_{ij} = 0$, $x^F_{ij} \in \mathbb{Z}_+$	$x^L_{ij} \geq 1$, $x^F_{ij} \in \mathbb{Z}_+$	(i,j)	$x^L_{ij} \in \mathbb{Z}_+$, $x^F_{ij} = 0$	$x^L_{ij} \in \mathbb{Z}_+$, $x^F_{ij} \geq 1$
(s,a_1)	10	15	(s,b_1)	100	10
(s,a_2)	20	42	(b_1,t)	100	2
(a_1,b_1)	10	11	(b_2,t)	100	1
(a_2,b_2)	20	40			
(b_1,c_1)	10	11			
(b_2,c_2)	1	25			
(c_1,t)	10	20			
(c_2,t)	1	1			

Table 3.2: Some interdiction strategies on the graph from Figure 3.1, recourse paths, and costs.

$\mathbf{x}^L$	**y for** $b^F = 0$	$\mathbf{z}(\mathbf{x}^L)$	**Domination**
$\mathbf{x}^1 \equiv \{(s,a_1),(a_2,b_2),(c_1,t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(55,27)$	Non-dominated
$\mathbf{x}^2 \equiv \{(s,a_1),(s,a_2),(c_1,t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(55,27)$	Non-dominated
$\mathbf{x}^3 \equiv \{(s,a_1),(b_2,c_2),(c_1,t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(55,27)$	Non-dominated
$\mathbf{x}^4 \equiv \{(s,a_1),(s,a_2),(a_1,b_1)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(46,28)$	Non-dominated
$\mathbf{x}^5 \equiv \{(s,a_1),(a_1,b_1),(a_2,b_2)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(46,28)$	Non-dominated
$\mathbf{x}^6 \equiv \{(s,a_1),(a_1,b_1),(b_2,c_2)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(46,28)$	Non-dominated
$\mathbf{x}^7 \equiv \{(s,a_1),(b_1,c_1),(c_1,t)\}$	$s \to a_2 \to b_2 \to c_2 \to t$	$(42,27)$	$\mathbf{z}(\mathbf{x}^4) > \mathbf{z}(\mathbf{x}^7)$
$\mathbf{x}^8 \equiv \{(s,a_1),(a_1,b_1)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(42,28)$	$\mathbf{z}(\mathbf{x}^4) \geq \mathbf{z}(\mathbf{x}^8)$

Moreover, expending one unit of either the interdiction or improvement budget on any arc reduces the marginal effect for additional expenditures to zero. Thus, each interdiction and improvement choice is essentially binary. Since $\mathbf{x}^L, \mathbf{x}^F \in \{0,1\}^{|\mathcal{A}|}$, rather than giving the interdiction and improvement strategies in vector form, we present them using their equivalent incidence sets, $\mathbf{x}^L \equiv \{(i,j) \in \mathcal{A} : x^L_{ij} = 1\}$ and $\mathbf{x}^F \equiv \{(i,j) \in \mathcal{A} : x^F_{ij} = 1\}$. Also, rather than giving the path $\mathbf{y}$ in its vector form, we present it as a sequence of nodes $s \to \cdots \to t$. We will hold these conventions in our other examples as well.

Table 3.2 gives several interdiction strategies and the corresponding optimal follower recourse paths for $b^F = 0$ and the costs for both cases. For $b^F = 1$, the optimal follower recourse path is $s \to a_1 \to b_1 \to t$, regardless of the leader's interdiction strategy. If the leader wishes to optimize primarily for the case of $b^F = 0$, breaking ties by optimizing for the case of $b^F = 1$, then the optimal interdiction will be one of $\{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3\}$. However, if the leader optimizes for $b^F = 1$ first, breaking ties for the $b^F = 0$ case, then one of $\{\mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^6\}$ is optimal. All other feasible strategies are dominated. For example, $\mathbf{z}(\mathbf{x}^7)$ is strongly dominated, and $\mathbf{z}(\mathbf{x}^8)$ is dominated. The non-dominated set is $Z_{\mathcal{N}} = \{[55,27],[46,28]\}$. The efficient set is $\mathbf{X}_E = \{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^6\}$. A complete efficient set is given by $X_{CE} = \{\mathbf{x}^a, \mathbf{x}^b\}$ where $\mathbf{x}^a \in \{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3\}$ and $\mathbf{x}^b \in \{\mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^6\}$.

In addition to demonstrating the concepts of non-dominance and efficiency in the context of the SPIP-I, this example also motivates the benefit of the proposed multiobjective approach. In this example, if the leader assumed that $b^F = 1$, then the leader would select one of $\{\mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^6, \mathbf{x}^8\}$. Solution $\mathbf{x}^8$ is dominated by the first three, though, and would not be generated by our multiobjective approach. Furthermore, the multiobjective approach also uncovers efficient strategies that raise the follower's cost by 9 units when $b^F = 0$, while sacrificing only one unit in the case that $b^F = 1$. $\square$

The remainder of this chapter focuses on solving the SPIP-I. Section 3.2 presents an algorithm to generate a complete set of efficient solutions for (3.2). For the case in which G is a series-parallel graph, we provide an alternative approach to generate a complete set of efficient solutions in Section 3.3. Section 3.4 revises the approach in Section 3.3 by avoiding algorithmic steps over portions of the graph that the follower does not use. We present computational results of all three algorithms in Section 3.5 and finally conclude our chapter in Section 3.6.

3.2 Transforming the SPIP-I into a SPIP instance

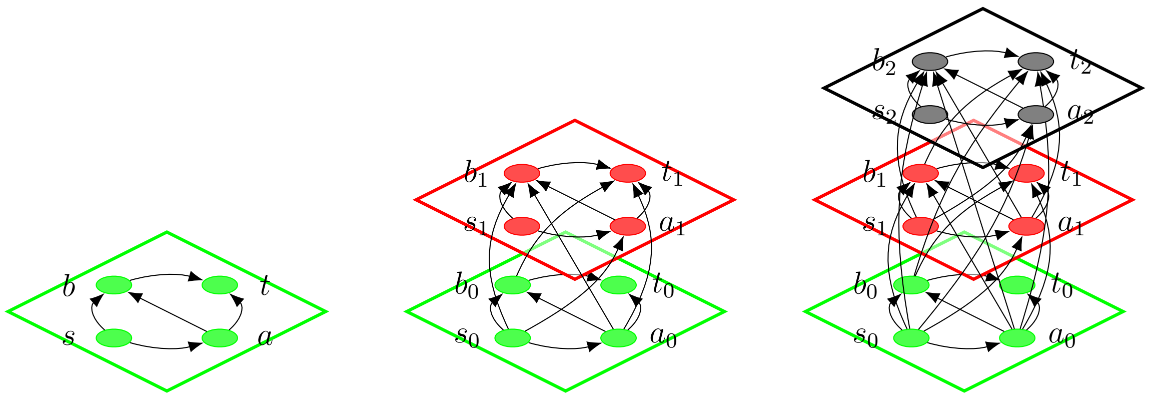
In this section we present an algorithm for generating a complete set of efficient solutions to the SPIP-I for general instances. This method reformulates the problem as a (multiobjective) SPIP instance, removing the complicating decision variables, $\mathbf{x}^F$. To transform a SPIP-I instance into a SPIP instance, we create an instance of SPIP having a graph $\bar{G} = (\bar{\mathcal{N}}, \bar{\mathcal{A}})$ (see Figure 3.2), where

- $\bar{\mathcal{N}} = \{n_h : n \in \mathcal{N}, h \in \{0, \dots, p\}\}$, and
- $\bar{\mathcal{A}} = \{(i_h, j_{h+\ell}) : (i,j) \in \mathcal{A},\ h \in \{0, \dots, p\},\ \ell \in \{0, \dots, p-h\}\}$.

Thus, $\bar{G}$ contains $p+1$ "layers," where for each $h \in \{0, \dots, p\}$, the set $\{n_h \in \bar{\mathcal{N}}\}$ constitutes the nodes in one layer of $\bar{G}$. For every arc $(i,j) \in \mathcal{A}$, we have multiple corresponding arcs, $(i_h, j_{h+\ell}) \in \bar{\mathcal{A}}$, within each layer, and from each layer to every higher layer. Selecting arc $(i_h, j_{h+\ell}) \in \bar{\mathcal{A}}$ represents the event where the follower uses arc $(i,j) \in \mathcal{A}$, expending h units of the improvement budget on the $s \to i$ path and an additional ℓ units on arc (i,j).

We restate (3.2) as the following multiobjective SPIP on $\bar{G}$:

$$\max_{\mathbf{x}^L} \quad [z_0(\mathbf{x}^L), \dots, z_p(\mathbf{x}^L)] \tag{3.3a}$$



(a) $G = (\mathcal{N}, \mathcal{A})$ (b) $\bar{G} = (\bar{\mathcal{N}}, \bar{\mathcal{A}})$ for $p = 1$ (c) $\bar{G} = (\bar{\mathcal{N}}, \bar{\mathcal{A}})$ for $p = 2$

Figure 3.2: Example of converting $G = (\mathcal{N}, \mathcal{A})$ into $\bar{G} = (\bar{\mathcal{N}}, \bar{\mathcal{A}})$.

$$\text{subject to:} \quad \sum_{(i,j)\in\mathcal{A}} x^L_{ij} \le b^L \tag{3.3b}$$

$$x^L_{ij} \in \mathbb{Z}_+ \qquad \forall (i,j) \in \mathcal{A}, \tag{3.3c}$$

$$\text{where } z_{b^F}(\mathbf{x}^L) = \min_{\bar{\mathbf{y}}} \sum_{(i_h, j_{h+\ell})\in\bar{A}} c_{ij}(x^L_{ij}, \ell)\bar{y}_{i_h j_{h+\ell}} \tag{3.3d}$$

$$\text{subject to:} \quad \sum_{\substack{j\in\bar{\mathcal{N}}:\\ (i,j)\in\bar{A}}} \bar{y}_{ij} - \sum_{\substack{j\in\bar{\mathcal{N}}:\\ (j,i)\in\bar{A}}} \bar{y}_{ji} = \begin{cases} 1, & i = s_0 \\ -1, & i = t_{b^F} \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in \bar{\mathcal{N}} \tag{3.3e}$$

$$\bar{y}_{ij} \ge 0 \qquad \forall (i,j) \in \bar{\mathcal{A}}. \tag{3.3f}$$

Note that unlike (3.1d), the objective function in (3.3d) is linear, because the choice of variable x^F_{ij} is implicit in the new arc-flow variable, $\bar{y}_{(i_h, j_{h+\ell})}$. The cost of removing the complicating discrete variable $\mathbf{x}^F$ and linearizing the inner objective is captured in the larger graph size, since $\bar{G}$ has $O(p)$ times the number of nodes and $O(p^2)$ times the number of arcs as G. The formulation of (3.3) may be further reduced to a single-level multiobjective mixed integer linear program by taking the dual of (3.3d)–(3.3f) and combining the two maximization optimization problems (see [76]).

Our proposed method to solving the general SPIP-I is captured in Algorithm 3.1. Algorithm 3.1 accepts as input a graph, G, and a collection of associated cost functions, $\{c_{ij}\}_{(i,j)\in\mathcal{A}}$. We assume the parameters b^L and p are implicit in $\{c_{ij}\}_{(i,j)\in\mathcal{A}}$. The algorithm returns a complete efficient set of interdiction strategies. The algorithm uses two auxiliary routines: `Layerize` and `MOGA`. `Layerize`

Algorithm 3.1: `SolveXE1`$(G, \{c_{ij}\}_{(i,j)\in\mathcal{A}})$

```
Input        : $G$: A directed graph
               $\{c_{ij}\}_{(i,j)\in\mathcal{A}}$: Cost functions for each $(i,j) \in \mathcal{A}$, where $c_{ij} : \mathbb{Z}^2_+ \to \mathbb{R}_+$
Output       : $X_{CE}(G)$: A complete set of efficient interdiction strategies
Dependencies : Layerize$(G, p)$: Returns a graph, $\bar{G}$ (see Figure 3.2)
               MOGA$(P)$: Multiobjective generating algorithm. Returns efficient
               solutions to $P$
1 $\bar{G} \leftarrow$ Layerize$(G, p)$
2 $P \leftarrow$ (3.3) formulated using $\bar{G}$
3 $\widehat{X}_{CE} \leftarrow$ MOGA$(P)$
4 return $\widehat{X}_{CE}$
```

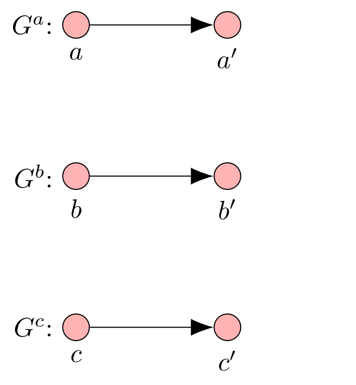
accepts as input a graph, G, and a parameter, p, and it returns the layered graph, $\bar{G}$, described previously. The auxiliary routine `MOGA` is a multiobjective generating algorithm that accepts as input a multiobjective problem, P, and returns a complete set of efficient solutions to P. There are a variety of algorithms that could fill the role of `MOGA`, including ϵ-constraint [46, 61], two-phase [66, 85], and branch-and-bound methods [55, 81, 87]. Our implementation (see Algorithm 2.1 in Appendix A.2) is based on the modified augmented weighted Tchebychev (MAWT) norm in Chapter 2, and tends to perform better than competing methods when $p > 2$. The following lemma holds due to the equivalence of formulations (3.2) and (3.3) and is stated without proof for brevity.

Lemma 3.1. Algorithm 3.1 returns a complete set of efficient interdiction strategies.

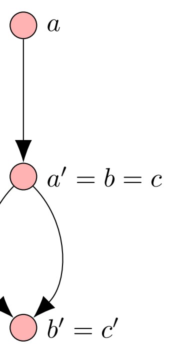
3.3 Recursion algorithm for series-parallel graphs

In this section, we present an alternative algorithm to generate efficient solutions to the SPIP-I on series-parallel graphs (SPGs) (see [86]). SPGs are applicable in a variety of scenarios (e.g., precedence constraints in sequencing and scheduling [57]), and they have a useful decomposition structure that may be exploited to solve the SPIP-I. Specifically, our method leverages the recursive decomposition of an SPG to derive efficient solutions for the composed graph from the efficient solutions for the subgraphs. Thus, while this algorithm pertains only to SPGs, it avoids the solving of mixed integer programs on a larger graph as in Section 3.2. To begin we establish the following notational conventions:

- Henceforth we do not need to explicitly model the improvement strategy, $\mathbf{x}^F$. For notational convenience, therefore, we omit the L superscript on the interdiction strategy.



(a) Three basic SPGS:SPIPI. G^a, G^b, G^c



(b) The composed SPG: $G^a \otimes (G^b \oplus G^c)$

Figure 3.3: Examples of basic SPG graphs and series-parallel configurations.

- When $G^a = (\mathcal{N}^a, \mathcal{A}^a)$ and $G^b = (\mathcal{N}^b, \mathcal{A}^b)$ are non-intersecting (i.e., $\mathcal{N}^a \cap \mathcal{N}^b = \emptyset$) SPGs, then $G^a \oplus G^b$ and $G^a \otimes G^b$ denote G^a and G^b in parallel and series constructions, respectively (see Figure 3.3).

- With a slight abuse in notation, when composing $G = G^a \oplus G^b$ or $G = G^a \otimes G^b$, we represent subgraph interdiction strategies $\mathbf{x}^a \in \mathbb{Z}_+^{|\mathcal{A}^a|} \times \{0\}^{|\mathcal{A}^b|}$ on G^a and $\mathbf{x}^b \in \{0\}^{|\mathcal{A}^a|} \times \mathbb{Z}_+^{|\mathcal{A}^b|}$ on G^b, while denoting the composition of these strategies on G as $\mathbf{x} = \mathbf{x}^a + \mathbf{x}^b \in \mathbb{Z}_+^{|\mathcal{A}^a \cup \mathcal{A}^b|}$.

- We superscript our objective, $\mathbf{z}^G(\mathbf{x})$, to highlight that $\mathbf{z}$ is a vector of shortest path lengths in graph G when the leader uses interdiction strategy $\mathbf{x}$. As before, each element of $\mathbf{z}$ corresponds to a different improvement budget for the follower.

- $X(G, \kappa^L) = \{\mathbf{x} \in \mathbb{Z}_+^{|\mathcal{A}|} : \sum_{(i,j)\in\mathcal{A}} x_{ij} = \kappa^L\}$ denotes the set of feasible interdiction strategies on G that expend an interdiction budget of $\kappa^L \leq b^L$ arcs.

- $X_E(G, \kappa^L)$ (similarly, $X_{CE}(G, \kappa^L)$) is the efficient set (similarly, a complete efficient set) of strategies over $X(G, \kappa^L)$.

We now develop the mathematical foundation for our recursive algorithm. Our first lemma argues that efficiency in the subgraph strategies is a necessary condition for efficiency on a series-composed graph.

Lemma 3.2. If $\mathbf{x} \in X_E(G^a \otimes G^b, \kappa^L)$, then $\mathbf{x} = \mathbf{x}^a + \mathbf{x}^b$ where $\mathbf{x}^a \in X_E(G^a, \kappa_a^L)$ and $\mathbf{x}^b \in X_E(G^b, \kappa^L - \kappa_a^L)$ with $\kappa_a^L = \sum_{(i,j)\in\mathcal{A}} x_{ij}^a$.

Proof. Assume by contradiction that $\mathbf{x}^a \notin X_E(G^a, \kappa_a^L)$, and so $\mathbf{z}^{G^a}(\mathbf{x}') \geq \mathbf{z}^{G^a}(\mathbf{x}^a)$ for some $\mathbf{x}' \in X(G^a, \kappa_a^L)$. Then

$$\begin{aligned}
\mathbf{z}(\mathbf{x}' + \mathbf{x}^b) &= \left[z_0^{G^a}(\mathbf{x}') + z_0^{G^b}(\mathbf{x}^b), \ldots, z_p^{G^a}(\mathbf{x}') + z_p^{G^b}(\mathbf{x}^b)\right] \\
&\geq \left[z_0^{G^a}(\mathbf{x}^a) + z_0^{G^b}(\mathbf{x}^b), \ldots, z_p^{G^a}(\mathbf{x}^a) + z_p^{G^b}(\mathbf{x}^b)\right] \\
&= \mathbf{z}(\mathbf{x}),
\end{aligned}$$

which contradicts the efficiency of $\mathbf{x}$. Thus, we know $\mathbf{x}^a \in X_E(G^a, \kappa_a^L)$. A similar argument shows $\mathbf{x}^b \in X_E(G^b, \kappa^L - \kappa_a^L)$, and the proof is complete. □

Example 3.2. We continue Example 3.1. Figure 3.4 gives a binary decomposition tree, $\mathcal{T}$, of the graph in Figure 3.1. Each node, $n_\# \in \mathcal{T}$, corresponds to a subgraph of Figure 3.1 induced by the arcs mapped from the leaves of the subtree rooted at $n_\#$. We refer to this subgraph as $G^\#$. Each branch node in $\mathcal{T}$ is labeled in Figure 3.4 as `Type` S (series) or P (parallel). Leaves are labeled with `Type` L. The root of the tree is n_{21}, with $G^{21} = G$.

Consider G^{11} as a series composition of the two subgraphs: G^5, induced by the arc set $\{(s, a_1), (a_1, b_1), (s, b_1)\}$, and G^{10}, induced by the arc set $\{(b_1, c_1), (c_1, t), (b_1, t)\}$. Table 3.3 gives efficient strategies on the subgraphs G^5, G^{10}, and G^{11}. Note that for each $\kappa^L = 0, \ldots, 3$, the efficient strategies in $X_E(G^{11}, \kappa^L)$ are composed of efficient strategies on G^5 and G^{10}. For example, $\mathbf{x} \equiv \{(s, a_1), (a_1, b_1), (c_1, t)\} \in X_E(G^{11}, 3)$ is composed of $\mathbf{x}^a \equiv \{(s, a_1), (a_1, b_1)\} \in X_E(G^5, 2)$ and $\mathbf{x}^b \equiv \{(c_1, t)\} \in X_E(G^{10}, 1)$. Conversely, not every feasible pairing of efficient strategies from G^5 and G^{10} creates an efficient strategy on G^{11}. For example, combining $\mathbf{x}^a \equiv \{(s, a_1)\} \in X_E(G^5, 1)$ and $\mathbf{x}^b \equiv \{(b_1, c_1), (c_1, t)\} \in X_E(G^{10}, 2)$ gives $\mathbf{x} = \mathbf{x}^a + \mathbf{x}^b \equiv \{(s, a_1), (b_1, c_1), (c_1, t)\} \notin X_E(G^{11}, 3)$. □

By contrast, for parallel-composed graphs, it is possible to create an efficient solution by composing a non-efficient subgraph strategy with an efficient subgraph strategy. However, the following lemma shows that every efficient strategy is *equivalent* to one that *is* composed entirely of efficient subgraph strategies.

Lemma 3.3. Let $\mathbf{x} \in X_E(G^a \oplus G^b, \kappa^L)$. Then there exist efficient strategies $\bar{\mathbf{x}}^a \in X_E(G^a, \kappa_a^L)$ and $\bar{\mathbf{x}}^b \in X_E(G^b, \kappa^L - \kappa_a^L)$, such that $\mathbf{z}^{G^a \oplus G^b}(\mathbf{x}) = \mathbf{z}^{G^a \oplus G^b}(\bar{\mathbf{x}}^a + \bar{\mathbf{x}}^b)$.

Proof. Define $\mathbf{x}^a \in \mathbb{Z}^{|\mathcal{A}^a|} \times \{0\}^{|\mathcal{A}^b|}$ and $\mathbf{x}^b \in \{0\}^{|\mathcal{A}^a|} \times \mathbb{Z}^{|\mathcal{A}^b|}$ such that $\mathbf{x} = \mathbf{x}^a + \mathbf{x}^b$, and set $\kappa_a^L = \sum_{(i,j)\in\mathcal{A}} x_{ij}^a$. Let $\bar{\mathbf{x}}^a \in X_E(G^a, \kappa_a^L)$ and $\bar{\mathbf{x}}^b \in X_E(G^b, \kappa^L - \kappa_a^L)$, giving $\mathbf{z}^{G^a}(\bar{\mathbf{x}}^a) \geqq \mathbf{z}^{G^a}(\mathbf{x}^a)$ and

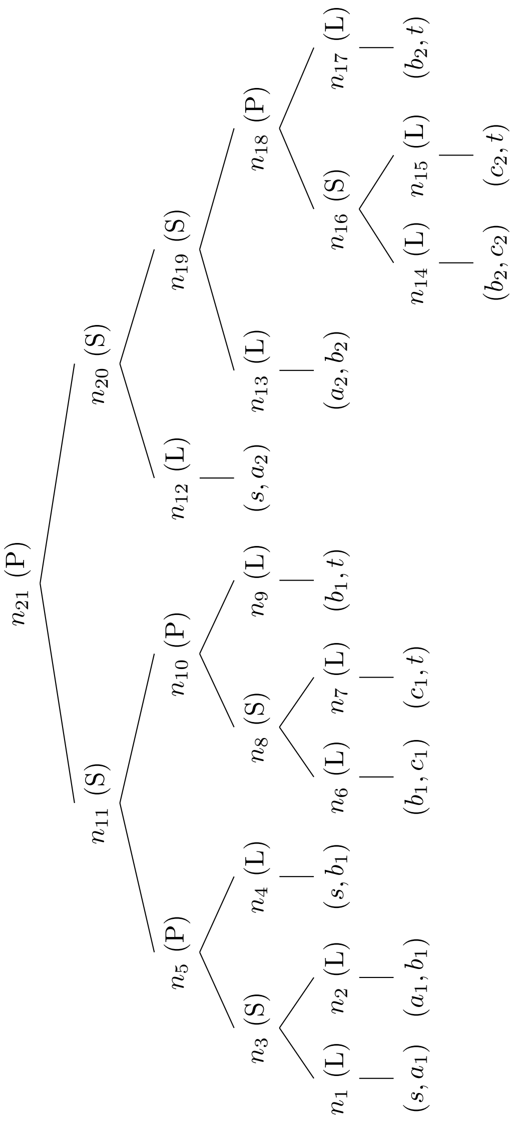


Figure 3.4: Binary decomposition tree, $\mathcal{T}$, for graph in Figure 3.1. Nodes are annotated $n_{\#}$ (`Type`), where `Type` is P = parallel, S = series, or L = leaf.

Table 3.3: Efficient interdiction strategies on G^{11} and its subgraphs, G^{10} and G^5, recourse paths, and costs.

Graph	**Set**	**Strategy**	$b^F = 0$ **path** $(z_0(\mathbf{x}))$	$b^F = 1$ **path** $(z_1(\mathbf{x}))$
G^5	$X_E(G^5, 2)$	$\{(s, a_1), (a_1, b_1)\}$	$s \to a_1 \to b_1$ (26)	$s \to b_1$ (10)
	$X_E(G^5, 1)$	$\{(s, a_1)\}$	$s \to a_1 \to b_1$ (25)	$s \to b_1$ (10)
	$X_E(G^5, 0)$	$\emptyset$	$s \to a_1 \to b_1$ (20)	$s \to b_1$ (10)
G^{10}	$X_E(G^{10}, 2)$	$\{(b_1, c_1), (c_1, t)\}$	$b_1 \to c_1 \to t$ (31)	$b_1 \to t$ (2)
	$X_E(G^{10}, 1)$	$\{(c_1, t)\}$	$b_1 \to c_1 \to t$ (30)	$b_1 \to t$ (2)
	$X_E(G^{10}, 0)$	$\emptyset$	$b_1 \to c_1 \to t$ (20)	$b_1 \to t$ (2)
G^{11}	$X_E(G^{11}, 3)$	$\{(s, a_1), (a_1, b_1), (c_1, t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$ (56)	$s \to a_1 \to b_1 \to t$ (28)
	$X_E(G^{11}, 2)$	$\{(s, a_1), (a_1, b_1)\}$	$s \to a_1 \to b_1 \to c_1 \to t$ (46)	$s \to a_1 \to b_1 \to t$ (28)
		$\{(s, a_1), (c_1, t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$ (55)	$s \to a_1 \to b_1 \to t$ (27)
	$X_E(G^{11}, 1)$	$\{(s, a_1)\}$	$s \to a_1 \to b_1 \to c_1 \to t$ (45)	$s \to a_1 \to b_1 \to t$ (27)
		$\{(c_1, t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$ (50)	$s \to a_1 \to b_1 \to t$ (22)
	$X_E(G^{11}, 0)$	$\emptyset$	$s \to a_1 \to b_1 \to c_1 \to t$ (40)	$s \to a_1 \to b_1 \to t$ (22)

$\mathbf{z}^{G^b}(\bar{\mathbf{x}}^b) \geqq \mathbf{z}^{G^b}(\mathbf{x}^b)$. We then find:

$$\begin{aligned}
z_{b^F}^{G^a \oplus G^b}(\mathbf{x}) &= \min\{z_{b^F}^{G^a}(\mathbf{x}^a), z_{b^F}^{G^b}(\mathbf{x}^b)\} \quad \forall b^F \in \{0, \dots, p\} \\
&\leq \min\{z_{b^F}^{G^a}(\bar{\mathbf{x}}^a), z_{b^F}^{G^b}(\bar{\mathbf{x}}^b)\} \quad \forall b^F \in \{0, \dots, p\} \\
&= z_{b^F}^{G^a \oplus G^b}(\bar{\mathbf{x}}^a + \bar{\mathbf{x}}^b) \quad \forall b^F \in \{0, \dots, p\} \\
\Rightarrow \mathbf{z}^{G^a \oplus G^b}(\mathbf{x}) &\leqq \mathbf{z}^{G^a \oplus G^b}(\bar{\mathbf{x}}^a + \bar{\mathbf{x}}^b).
\end{aligned}$$

However, by the efficiency of $\mathbf{x}$, we know $\mathbf{z}^{G^a \oplus G^b}(\mathbf{x}) \not\leq \mathbf{z}^{G^a \oplus G^b}(\bar{\mathbf{x}}^a + \bar{\mathbf{x}}^b)$, so $\mathbf{z}^{G^a \oplus G^b}(\mathbf{x}) = \mathbf{z}^{G^a \oplus G^b}(\bar{\mathbf{x}}^a + \bar{\mathbf{x}}^b)$. □

Example 3.3. Continuing from Example 3.1, consider the graph $G = G^{21} = G^{11} \oplus G^{20}$. Table 3.4 describes all efficient strategies for G^{11}, G^{20}, and G^{21}. As in the series case, not every pairing of efficient strategies on G^{11} and G^{20} generates an efficient strategy for G^{21}. For example, $\mathbf{x}^a \equiv \{(s, a_1)\} \in X_E(G^{11}, 1)$ and $\mathbf{x}^b \equiv \{(s, a_2), (a_2, b_2)\} \in X_E(G^{20}, 2)$ combine to form $\mathbf{x}^a + \mathbf{x}^b \equiv \{(s, a_1), (s, a_2), (a_2, b_2)\} \notin X_E(G^{21}, 3)$. Unlike the series case, the converse also fails. In this scenario, there are four efficient interdiction strategies in $X_E(G^{21}, 3)$ that are derived by combining efficient strategies from G^{11} and G^{20} (namely, $\mathbf{x}^2$, $\mathbf{x}^3$, $\mathbf{x}^4$, and $\mathbf{x}^6$) and two efficient strategies (namely, $\mathbf{x}^1$ and $\mathbf{x}^5$) that combine an efficient strategy from $X_E(G^{11}, 2)$ with a non-efficient interdiction strategy on G^{20}. However, $\mathbf{z}^G(\mathbf{x}^2) = \mathbf{z}^G(\mathbf{x}^1)$ and $\mathbf{z}^G(\mathbf{x}^4) = \mathbf{z}^G(\mathbf{x}^5)$ are equivalent strategies composed of efficient subgraph strategies as described in Lemma 3.3. □

Table 3.4: Efficient interdiction strategies on graphs of G^{11}, G^{20}, and G^{21}, $b^F = 0$ recourse paths, and objective vector. Recourse path when $b^F = 1$ is $s \to a_1 \to b_1 \to t$ in all cases.

Graph	**Set**	**Strategy**	$b^F = 0$ **path**	$\mathbf{z}(\mathbf{x})$)
G^{11}	$X_E(G^{11}, 3)$	$\{(s, a_1), (a_1, b_1), (c_1, t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(56, 28)$
	$X_E(G^{11}, 2)$	$\{(s, a_1), (a_1, b_1)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(46, 28)$
		$\{(s, a_1), (c_1, t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(55, 27)$
	$X_E(G^{11}, 1)$	$\{(s, a_1)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(45, 27)$
		$\{(c_1, t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(50, 22)$
	$X_E(G^{11}, 0)$	$\emptyset$	$s \to a_1 \to b_1 \to c_1 \to t$	$(40, 22)$
G^{20}	$X_E(G^{20}, 3)$	$\{(s, a_2), (a_2, b_2), (b_2, c_2)\}$	$s \to a_2 \to b_2 \to c_2 \to t$	$(108, 83)$
	$X_E(G^{20}, 2)$	$\{(s, a_2), (a_2, b_2)\}$	$s \to a_2 \to b_2 \to c_2 \to t$	$(84, 83)$
		$\{(s, a_2), (b_2, c_2)\}$	$s \to a_2 \to b_2 \to c_2 \to t$	$(88, 42)$
	$X_E(G^{20}, 1)$	$\{(s, a_2)\}$	$s \to a_2 \to b_2 \to c_2 \to t$	$(64, 63)$
		$\{(b_2, c_2)\}$	$s \to a_2 \to b_2 \to c_2 \to t$	$(66, 41)$
	$X_E(G^{20}, 0)$	$\emptyset$	$s \to a_2 \to b_2 \to c_2 \to t$	$(42, 41)$
G^{21}	$X_E(G^{21}, 3)$	$\mathbf{x}^1 \equiv \{(s, a_1), (a_2, b_2), (c_1, t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(55, 27)$
		$\mathbf{x}^2 \equiv \{(s, a_1), (s, a_2), (c_1, t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(55, 27)$
		$\mathbf{x}^3 \equiv \{(s, a_1), (b_2, c_2), (c_1, t)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(55, 27)$
		$\mathbf{x}^4 \equiv \{(s, a_1), (s, a_2), (a_1, b_1)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(46, 28)$
		$\mathbf{x}^5 \equiv \{(s, a_1), (a_1, b_1), (a_2, b_2)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(46, 28)$
		$\mathbf{x}^6 \equiv \{(s, a_1), (a_1, b_1), (b_2, c_2)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(46, 28)$
	$X_E(G^{21}, 2)$	$\{(s, a_1), (s, a_2)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(45, 27)$
		$\{(s, a_1), (b_2, c_2)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(45, 27)$
		$\{(s, a_1), (a_2, b_2)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(45, 27)$
		$\{(c_1, t), (s, a_2)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(50, 22)$
		$\{(c_1, t), (a_2, b_2)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(50, 22)$
		$\{(c_1, t), (b_2, c_2)\}$	$s \to a_1 \to b_1 \to c_1 \to t$	$(50, 22)$
	$X_E(G^{21}, 1)$	$\{(s, a_1)\}$	$s \to a_2 \to b_2 \to c_2 \to t$	$(42, 27)$
	$X_E(G^{21}, 0)$	$\emptyset$	$s \to a_1 \to b_1 \to c_1 \to t$	$(40, 22)$

Corollary 3.3.1. Let $X_{CE}^{G^a}$ and $X_{CE}^{G^b}$ be complete efficient sets on SPGs G^a and G^b, respectively. If $G = G^a \oplus G^b$ or $G = G^a \otimes G^b$, then there exists $X_{CE}^{G} \subseteq \{\mathbf{x}^a + \mathbf{x}^b : \mathbf{x}^a \in X_{CE}^{G^a},\ \mathbf{x}^b \in X_{CE}^{G^b}\}$.

Proof. Let $\mathbf{z}^G(\bar{\mathbf{x}})$ be a non-dominated objective. If $G = G^a \otimes G^b$ or $G = G^a \oplus G^b$, then by Lemma 3.2 or Lemma 3.3, respectively, we have $\mathbf{z}^G(\bar{\mathbf{x}}) = \mathbf{z}^G(\bar{\mathbf{x}}^a + \bar{\mathbf{x}}^b)$ for some $\bar{\mathbf{x}}^a \in X_E^{G^a}$ and $\bar{\mathbf{x}}^b \in X_E^{G^b}$. If $\bar{\mathbf{x}}^a \notin X_{CE}^{G^a}$ or $\bar{\mathbf{x}}^b \notin X_{CE}^{G^b}$, then since $X_{CE}^{G^a}$ and $X_{CE}^{G^b}$ are complete efficient sets, we can find $\mathbf{x}^a \in X_{CE}^{G^a}$ or $\mathbf{x}^b \in X_{CE}^{G^b}$ with $\mathbf{z}^{G^a}(\bar{\mathbf{x}}^a) = \mathbf{z}^{G^a}(\mathbf{x}^a)$ and $\mathbf{z}^{G^b}(\bar{\mathbf{x}}^b) = \mathbf{z}^{G^b}(\mathbf{x}^b)$. Thus, $\exists \mathbf{x} \in \{\mathbf{x}^a + \mathbf{x}^b : \mathbf{x}^a \in X_{CE}^{G^a},\ \mathbf{x}^b \in X_{CE}^{G^b}\}$ such that $\mathbf{z}^G(\mathbf{x}) = \mathbf{z}^G(\mathbf{x}^a + \mathbf{x}^b) = \mathbf{z}^G(\bar{\mathbf{x}}^a + \bar{\mathbf{x}}^b) = \mathbf{z}^G(\bar{\mathbf{x}})$. $\square$

We now present Algorithm 3.2 to solve the SPIP-I on SPGs. The algorithm accepts as input an SPG, G, and a collection of cost functions, $\{c_{ij}\}_{(i,j)\in\mathcal{A}}$, and it returns a complete efficient set of interdiction strategies, $X_{CE}(G, b^L)$. It does this by means of a recursive routine, `SP`, given in lines 4–26. Algorithm 3.2 requires several auxiliary routines: `DecompTree`, `Children`, and `Type`. `DecompTree` generates a tree, $\mathcal{T}$, representing the binary decomposition of G, and returns the tree root node, r (see Appendix A.2.3). The `Children` routine returns the child nodes of a branch node in $\mathcal{T}$, or the arc in G represented by a leaf node in $\mathcal{T}$.

The outer processing of Algorithm 3.2 is given in lines 1–3. Line 1 generates a decomposition tree, $\mathcal{T}$, and receives the root, r. Line 2 then calls the subroutine `SP` on the tree root, receiving a collection of sets of efficient interdiction strategies on G (one for each $\kappa^L \in \{0, \ldots, b^L\}$). The algorithm then terminates on line 3 by returning the complete set of efficient strategies that expend b^L of the interdiction budget.

The key functionality of Algorithm 3.2 lies in the recursive subroutine `SP`, which takes as input a node, $n \in \mathcal{T}$, and returns a collection, $\{X_{CE}(G^n, \kappa^L)\}_{\kappa^L \in \{0,\ldots,b^L\}}$. The base case for `SP`$(n)$, handled in lines 4–8, has n a leaf, and G^n contains a single arc. The arc is identified in line 5 by calling `Children`(n). Line 6 gives for each $\kappa^L = 0, \ldots, b^L$ the interdiction strategy expending κ^L interdictions on that arc. The associated objective vector of costs to travel the subgraph G^n for each possible $\kappa^F \in \{0, \ldots, p\}$ is computed in line 7. Line 8 returns these results.

In the recursion case (lines 9–26), `SP` begins calling `Children`(n) and receives nodes c^a and c^b, corresponding to subgraphs G^a and G^b, respectively. Then lines 11–12 recursively identify efficient strategies on G^a and G^b. Next, line 13 initializes a loop to obtain $\widehat{X}_{CE}(G^n, \kappa^L)$ for each κ^L. The algorithm initializes $\widehat{X}_{CE}(G^n, \kappa^L)$ as an empty set in line 14. Then lines 15–16 consider every possible distribution of κ^L interdictions between the subgraphs, G^a and G^b, by κ^a and κ^b, respectively. Lines 17–21 combine efficient subgraph interdiction strategies, $\mathbf{x}^a$ and $\mathbf{x}^b$, to make $\mathbf{x}^a + \mathbf{x}^b$, and they compute $\mathbf{z}^{G^n}(\mathbf{x}^a + \mathbf{x}^b)$ by a sequence of minimization evaluations on $\mathbf{z}^{G^a}(\mathbf{x}^a)$ and $\mathbf{z}^{G^b}(\mathbf{x}^b)$. Those evaluations depend on whether G^n is a parallel or series composition of G^a and G^b. The strategy is added to the set of candidate efficient vectors in line 22. Lines 23–24 evaluate each pair of candidate efficient vectors to see if any are dominated by another. The redundant and dominated ones are removed by line 25, leaving a complete set of efficient vectors. After generating $\widehat{X}_{CE}(G^n, \kappa^L)$ for each $\kappa^L \in \{0, \ldots, b^L\}$, Algorithm 3.2 returns the collection of these sets.

Proposition 3.1. Each $\widehat{X}_{CE}(G, \kappa^L)$ returned by Algorithm 3.2 is a complete efficient set of interdiction strategies expending total interdiction effort κ^L on graph G.

Proof. In the base case, G^n contains one arc. Trivially, then $X(G^n, \kappa^L) = X_E(G^n, \kappa^L) = \widehat{X}_{CE}(G^n, \kappa^L)$ for each $\kappa^L = 0, \ldots, p$, so that line 8 returns a complete efficient set. In the recursion case, Corollary 3.3.1 provides that the set $\widehat{X}_{CE}(G^n, \kappa^L)$, composed in lines 15–22 from complete efficient sets of sub-graph strategies, contains a complete efficient set. Inefficient strategies are removed by the inspection subroutine of lines 23–25, leaving a complete efficient set of interdiction strategies on the composed graph. $\square$

Algorithm 3.2: `SolveXE2`$(G,\ \{c_{ij}\}_{(i,j)\in\mathcal{A}})$

```
Input      : $G$: A series-parallel graph
             $\{c_{ij}\}_{(i,j)\in\mathcal{A}}$: Collection of arc cost functions
Output     : $X_{CE}$: A complete efficient set of interdiction strategies
Dependency : DecompTree($G$): Decomposes $G$ into tree, and returns the tree root, $r$
             Children($n$): returns the children nodes of $n$
             Type($n$): returns the type (S, P, or L) of node $n$
1  $r \leftarrow$ DecompTree($G$)
2  $\left\{\left\{(\mathbf{x},\mathbf{z}(\mathbf{x})) : \mathbf{x}\in\widehat{X}_{CE}(G,\kappa^L)\right\}\right\}_{\kappa^L=0}^{b^L} \leftarrow$ SP($r$)
3  return $\widehat{X}_{CE}(G,b^L)$

   SP($n$)
4  if Type($n$) = "L" then                                                      // Base Case
5      $a \leftarrow$ Children($n$)
6      $\mathbf{x}_{\kappa^L} \leftarrow [0,\ldots,0,\kappa^L,0,\ldots,0]$ for each $\kappa^L = 0,\ldots,b^L$        // $a^{\text{th}}$ term is $\kappa^L$
7      $\mathbf{z}(\mathbf{x}_{\kappa^L}) \leftarrow [c_a(\kappa^L,0),\ldots,c_a(\kappa^L,p)]$ for each $\kappa^L = 0,\ldots,b^L$
8      return $\{\{(\mathbf{x}_{\kappa^L},\mathbf{z}(\mathbf{x}_{\kappa^L}))\}\}_{\kappa^L=0}^{b^L}$
9  else                                                                        // Recursion Case
10     $(c^a,c^b) \leftarrow$ Children($n$)
11     $\{\{(\mathbf{x}^a,\mathbf{z}^{G^a}(\mathbf{x}^a)) : \mathbf{x}^a\in X_E(G^a,\kappa^L)\}\}_{\kappa^L=0}^{b^L} \leftarrow$ SP($c^a$)
12     $\left\{\{(\mathbf{x}^b,\mathbf{z}^{G^b}(\mathbf{x}^b)) : \mathbf{x}^b\in X_E(G^b,\kappa^L)\}\right\}_{\kappa^L=0}^{b^L} \leftarrow$ SP($c^b$)
13     for $\kappa^L = 0,\ldots,b^L$ do
14         $\widehat{X}_{CE}(G^n,\kappa^L) \leftarrow \emptyset$
15         for $\kappa^a = 0,\ldots,\kappa^L$ do
16             $\kappa^b = \kappa^L - \kappa^a$
17             for $\mathbf{x}^a \in X_E(G^a,\kappa^a)$ AND $\mathbf{x}^b \in X_E(G^b,\kappa^b)$ do
18                 if Type($n$) = "P" then
19                     $\mathbf{z}^{G^n}(\mathbf{x}^a+\mathbf{x}^b) \leftarrow$
                           $\left[\min\left\{z_0^{G^a}(\mathbf{x}^a), z_0^{G^b}(\mathbf{x}^b)\right\}, \min\left\{z_1^{G^a}(\mathbf{x}^a), z_1^{G^b}(\mathbf{x}^b)\right\}, \ldots,\right.$
                                   $\left.\min\left\{z_p^{G^a}(\mathbf{x}^a), z_p^{G^b}(\mathbf{x}^b)\right\}\right]$
20                 else                                                    // Type($n$) = "S"
21                     $\mathbf{z}^{G^n}(\mathbf{x}^a+\mathbf{x}^b) \leftarrow$
                           $\left[\left(z_0^{G^a}(\mathbf{x}^a) + z_0^{G^b}(\mathbf{x}^b)\right), \min_{\kappa\in\{0,1\}}\left\{z_\kappa^{G^a}(\mathbf{x}^a) + z_{1-\kappa}^{G^b}(\mathbf{x}^b)\right\}, \ldots,\right.$
                                   $\left.\min_{\kappa\in\{0,\ldots,p\}}\left\{z_\kappa^{G^a}(\mathbf{x}^a) + z_{p-\kappa}^{G^b}(\mathbf{x}^b)\right\}\right]$
22                 $\widehat{X}_{CE}(G^n,\kappa^L) \leftarrow \widehat{X}_E(G^n,\kappa^L) \cup \{(\mathbf{x}^a+\mathbf{x}^b)\}$
23         for $\mathbf{x}^a,\mathbf{x}^b \in \widehat{X}_{CE}(G^n,\kappa^L)$ do
24             if $\mathbf{z}^{G^n}(\mathbf{x}^a) \geqq \mathbf{z}^{G^n}(\mathbf{x}^b)$ then
25                 $\widehat{X}_{CE}(G^n,\kappa^L) \leftarrow \widehat{X}_{CE}(G^n,\kappa^L) \setminus \{\mathbf{x}^b\}$
26     return $\left\{\left\{(\mathbf{x},\mathbf{z}(\mathbf{x})) : \mathbf{x}\in\widehat{X}_{CE}(G^n,\kappa^L)\right\}\right\}_{\kappa^L=0}^{b^L}$
```

3.4 Improved recursive algorithm

A weakness of Algorithm 3.2 is that it generates non-dominated strategies on all parallel-composed subgraphs, regardless of whether arcs in these subgraphs are potentially desirable to the follower.

Example 3.4. Continuing Example 3.1, Table 3.3 shows that regardless of the interdiction strategy on G^{11}, the follower never chooses to traverse arc (s, b_1). Thus, when Algorithm 3.2 parses node n_4 in Figure 3.4, it gains no useful information in generating the complete set of efficient interdiction strategies. In general, as the number of unused arcs increases, the amount of effort required to execute Algorithm 3.2 can in the worst case grow exponentially. $\square$

In this section we present Algorithm 3.3, which composes efficient interdiction strategies in a manner similar to Algorithm 3.2, but parses only over a restricted portion of G. Throughout the course of the algorithm, we will track a set of *used arcs* in G, which are those arcs the follower has used in constructing short paths. Interdiction strategies are computed over a restricted graph that contains only used arcs. We then compute the objective vector $\mathbf{z}^G(\mathbf{x})$ by fixing $\mathbf{x}$ and finding the follower's shortest $s_0 \to t_h$ paths for each $h \in \{0, \ldots, p\}$ in the layered graph, $\bar{G}$, given $\mathbf{x}$. The algorithm examines these paths for new arcs, and if it finds any then it updates the list of *used arcs* and finds a new interdiction strategy. The key to Algorithm 3.3 lies in finding these new interdiction strategies by updating information from the previous iteration rather than recomputing all interdiction strategies from scratch.

As in Algorithms 3.1 and 3.2, Algorithm 3.3 takes as input a graph and a collection of cost functions and returns a complete efficient set of interdiction strategies. Additional auxiliary routines required by Algorithm 3.3 are `Parent`, `Dijkstra`, `Arcs`, and `SP*`. `Parent` is the inverse of `Children`, returning the parent of a node $n \in \mathcal{T}$, or when given an arc in G, it returns the corresponding leaf in $\mathcal{T}$. `Dijkstra` accepts a graph, G, a source node, s, a set of sink nodes, T, and a vector of arc costs, $\mathbf{c}$, and it returns a collection of shortest $s \to t$ paths, $\{\pi_t\}_{t \in T}$. `Arcs` returns the set of arcs on path π. Finally the routine `SP*` is a variant of `SP` from Algorithm 3.2 where the recursive calls of lines 11 and 12 are replaced with a call to `GetEff`, the routine given in lines 22–25 of Algorithm 3.3.

Algorithm 3.3 maintains a list `UsedArcs`. Also for each node in $\mathcal{T}$ that is an ancestor of an arc in `UsedArcs`, the algorithm maintains a collection of interdiction strategies for that node. The collection consists of sets that, assuming that the follower is limited to arcs in `UsedArcs`,

Algorithm 3.3: `SPTree`$(G, \{c_{ij}\}_{(i,j)\in\mathcal{A}})$

```
Input      : $G$: A series-parallel graph
Output     : $X_{CE}$: A complete efficient set of interdiction strategies
Dependency : DecompTree($G$): Performs a tree decomposition for $G$, and returns the
             root, $r$
             Layerize($G, p$): Returns a graph, $\bar{G}$, having $p+1$ layers (see Figure 3.2)
             Parent($n$): Returns the parent node of $n$ in $T$
             Dijkstra($G, s, T, \mathbf{c}$): Returns the collection $\{\pi_t\}_{t\in T}$ where each $\pi_t$ is a
             shortest $s \to t$ path on graph $G$, given arc costs $\mathbf{c} \in \mathbb{R}_+^{|\mathcal{A}|}$
             Arcs($\pi$): Returns the set of arcs along path $\pi$
             SP*($n$): Lines 4–26 of Algorithm 3.2, with lines 11–12 replace by call to
             GetEff

   Main Routine
1  $(r) \leftarrow$ DecompTree($G$)
2  $\bar{G} \leftarrow$ Layerize($G, p$)
3  for $n \in \mathcal{T} \setminus \{r\}$ do
4      $\widehat{X}_{CE}(G^n, 0) \leftarrow \{\mathbf{0}\}$; $\overline{\mathbf{z}^{G^n}}(\mathbf{0}) \leftarrow [\infty, \ldots, \infty]$
5      $\widehat{X}_{CE}(G^n, \kappa^L) \leftarrow \emptyset$ for $\kappa^L = 1, \ldots, b^L$
6      NeedsUpdate($n$) $\leftarrow$ False
7  UsedArcs $\leftarrow \emptyset$; NeedsUpdate($r$) $\leftarrow$ True
8  while NeedsUpdate($r$) do
9      $\left\{\{(\mathbf{x}, \overline{\mathbf{z}^{G}}(\mathbf{x}) : \mathbf{x} \in \widehat{X}_{CE}(G, \kappa^L)\}\right\}_{\kappa^L=0}^{b^L} \leftarrow$ GetEff($r$)
10     for $\mathbf{x} \in \bigcup_{\kappa^L=0}^{b^L} \widehat{X}_{CE}(G, \kappa^L)$ do
11         for $(i_h, j_{h+\ell}) \in \bar{\mathcal{A}}$ do
12             $c_{(i_h, j_{h+\ell})} \leftarrow c_{ij}(x_{ij}, \ell)$          // At the end, $\mathbf{c} \in \mathbb{R}^{|\bar{\mathcal{A}}|}$
13         $\{\pi_h\}_{h=0}^{p} \leftarrow$ Dijkstra($\bar{G}, s_0, \{t_h\}_{h=0}^{p}, \mathbf{c}$)
14         for $a \in \bigcup_{\pi=\pi_0,\ldots,\pi_p}$ Arcs($\pi$) do
15             if $a \notin$ UsedArcs then
16                 UsedArcs $\leftarrow$ UsedArcs $\cup \{a\}$
17                 $n \leftarrow$ Parent($a$)
18                 while NeedsUpdate($n$) = False AND $n \neq \emptyset$ do
19                     NeedsUpdate($n$) $\leftarrow$ True
20                     $n \leftarrow$ Parent($n$)          // Parent($r$) = $\emptyset$
21 return $\widehat{X}_{CE}(G, b^L)$

   GetEff($n$)
22 if NeedsUpdate($n$) then
23     NeedsUpdate(n) $\leftarrow$ False
24     $\left\{\{(\mathbf{x}, \overline{\mathbf{z}^{G^n}}(\mathbf{x}) : \mathbf{x} \in \widehat{X}_{CE}(G^n, \kappa^L)\}\right\}_{\kappa^L=0}^{b^L} \leftarrow$ SP*($n$)
25 return $\left\{\{(\mathbf{x}, \overline{\mathbf{z}^{G^n}}(\mathbf{x}) : \mathbf{x} \in \widehat{X}_{CE}(G^n, \kappa^L)\}\right\}_{\kappa^L=0}^{b^L}$
```

compose complete efficient sets of strategies, expending κ^L units of the interdiction budget, for each $\kappa^L \in \{0, \ldots, b^L\}$. We denote this collection by $\{X_{CE}(G^n, \kappa^L)\}_{\kappa^L=0}^{p}$. Initially `UsedArcs` $= \emptyset$, so for each $n \in \mathcal{T}$, the collection of strategies $\{\widehat{X}_{CE}(G^n, \kappa^L)\}_{\kappa^L=0}^{b^L}$ has elements:

$$\widehat{X}_{CE}(G^n, \kappa^L) = \begin{cases} \{\mathbf{0}\}, & \kappa^L = 0 \\ \emptyset, & 1 \leq \kappa^L \leq b^L, \end{cases}$$

where $\mathbf{0}$ indicates a strategy of expending no interdictions on any arc. Finally, for the `SP*` routine, we associate with each strategy an upper bound on the objective. Initially, this upper bound is $\overline{\mathbf{z}^{G^n}}(\mathbf{0}) = [\infty, \ldots, \infty]$. At each iteration, if a node is not flagged for updating, `GetEff` returns the incumbent collection of strategies for G^n and the associated objective upper bounds, thus avoiding redundant computations within the decomposition tree.

Algorithm 3.3 begins the initialization process by generating the decomposition tree, $\mathcal{T}$, in line 1, receiving the tree root, r, in return, and then constructing the layered graph, $\bar{G}$ in line 2. Then for every node except the root, lines 3–6 establish the initial set of strategies and set flag `NeedsUpdate`$(n) =$ `False`. The final initialization step creates the empty set `UsedArcs`, and it flags the root for needing an update.

The main loop of Algorithm 3.3 is lines 8–20, which continue to iterate as long as the root needs updating. The main loop begins by calling the routine, `GetEff`(r), given in lines 22–25. Beginning at the root, this routine checks if a node n needs updating. If n does need updating, then Algorithm 3.3 calls `SP*`, storing and returning the result, but if it does not, then `GetEff` simply returns the incumbent collection of interdiction strategies for G^n without calling `SP*`. In this way, `GetEff` executes `SP*` only on nodes needing updates, and after the recursion is complete all nodes are properly updated for the set `UsedArcs`. After receiving a collection of interdiction strategies for $G^r = G$ in line 9, Algorithm 3.3 proceeds in lines 10–20 to check whether non-updated arcs might provide more favorable recourse paths for the follower. Line 10 initiates a loop through all of the latest strategies. Lines 11–13 construct a set of arc costs in $\bar{G}$ that correspond to the strategy and find a corresponding set of shortest $s_0 \to t_h$ path in $\bar{G}$ for each $h \in \{0, \ldots, p\}$. Finally, lines 14–20 iterate through all arcs in all shortest paths, and if any arc was previously unused, then it and all its ancestors (including the root) are flagged for updating. If no new arcs are found, then no nodes are flagged for updating, and since the root does not need updating, the algorithm terminates in line 21 by returning the complete efficient set of strategies that expends an interdiction budget of b^L.

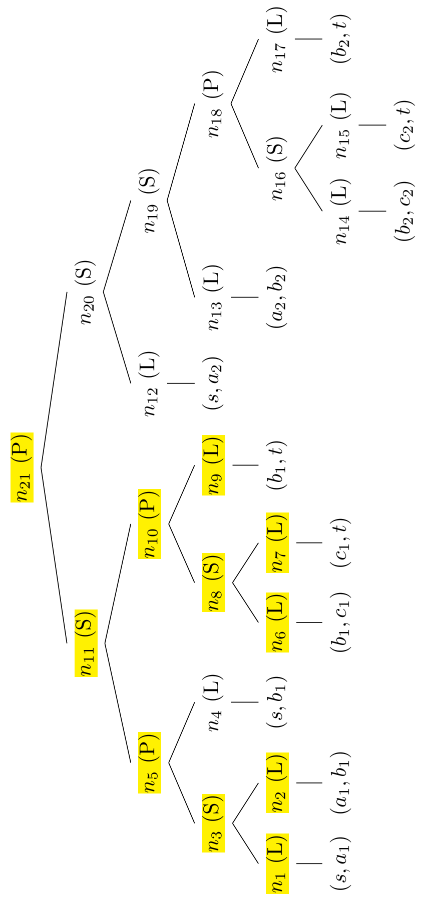
Recall from Example 3.4 that Algorithm 3.2 generates a complete efficient set of strategies by parsing the entire tree in Figure 3.4. We now examine how Algorithm 3.3 would process the problem of Example 3.4 differently.

Example 3.5. Algorithm 3.3 begins by generating $\mathcal{T}$, $\bar{G}$, and setting the initially empty sets of strategies for every node in $\mathcal{T} \setminus \{r\}$. Initially, `UsedArcs` is empty, and only the root needs updating. After initialization the algorithm enters the main **while** loop because the root needs updating.

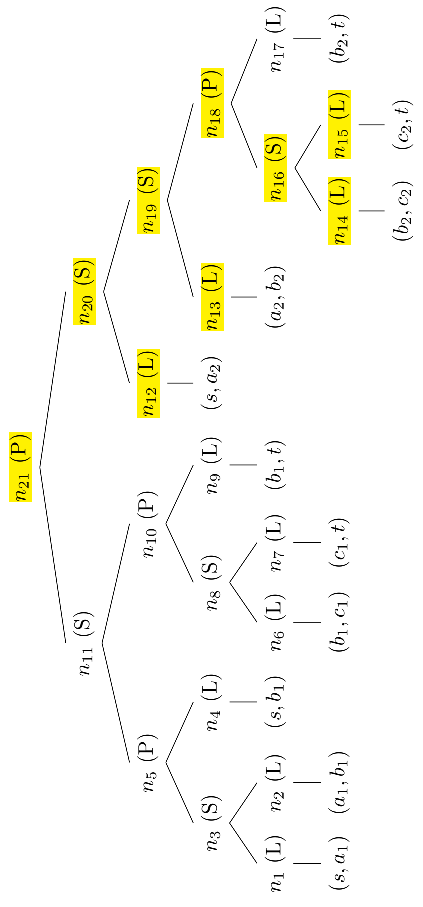
Since the root requires updating, Algorithm 3.3 enters the first loop. In `GetEff`(r), it calls `SP*`(r), and since neither child of r requires updating, it returns the incumbent solutions, $X_{CE}(G^{11}, 0) = X_{CE}(G^{20}, 0) = \{\mathbf{0}\}$, without calling `SP*`$(n_{11})$ or `SP*`(n_{20}). `SP*`(r) combines these results to get $\{X_{CE}(G^{21}, 0)\} = \{\mathbf{0}\}$ with the corresponding objective $\overline{\mathbf{z}^{G^{21}}}(\mathbf{0}) = [\infty, \ldots, \infty]$. Next, line 10 iterates through each strategy in the set $\bigcup_{\kappa^L=0}^{b^L} \widehat{X}_{CE}(G, \kappa^L) = \{\mathbf{0}\}$. After lines 11–12 construct the appropriate vector of arc costs, $\mathbf{c}$, line 13 finds the shortest $s_0 \to t_h$ paths in $\bar{G}$ given no interdictions. From Table 3.4 in the $X_E(G^{21}, 0)$ row, `Dijkstra` returns $\pi_0 \equiv s \to a_1 \to b_1 \to c_1 \to t$ with cost 40 and $\pi_1 \equiv s \to a_1 \to b_1 \to t$ with cost 22. Since `UsedArcs` is empty, lines 14–20 adds each arc in these paths to `UsedArcs`, flagging all the ancestor nodes to each one for updating in the process.

The root needs updating, so Algorithm 3.3 enters the main **while** loop for a second iteration. Proceeding to the `GetEff` routine, `GetEff`(n_{21}) calls `SP*`(n_{21}), which recursively calls `GetEff`(n_{11}) and `GetEff`(n_{20}). Since n_{20} does not need updating, `GetEff` immediately returns the incumbent collection, $\{X_{CE}(G^{20}, \kappa^L)\}_{\kappa^L=0}^{b^L}$, without calling `SP*`$(n_{20})$. However, n_{11} does need updating, so `SP*`(n_{11}) is called, and the recursion process continues. Table 3.5 outlines the strategies for $G^{21} = G$ at the end of the recursion process. These are returned to Algorithm 3.3, which then validates them against a set of shortest paths from the `Dijkstra` routine (also given in Table 3.5). Of the arcs on these paths, $\{(s, a_1), (a_1, b_1), (b_1, c_1), (b_1, t), (c_1, t)\} \subseteq$ `UsedArcs`, so lines 14–20 only add (s, a_2), (a_2, b_2), (b_2, c_2), and (c_2, t) to `UsedArcs`, flagging leaf nodes n_{12}, n_{13}, n_{14}, and n_{15}, respectively, and their ancestors for updating.

The root needs updating, so a third iteration is performed. The `GetEff` routine calls `SP*`(n_{21}), which then finds child nodes n_{11} and n_{20}. Node n_{11} does not need updating, so `GetEff`(n_{11}) returns the incumbent solutions from iteration 2, given in Table 3.5. However, n_{20} does require updating, so `GetEff`(n_{20}) calls `SP*`(n_{20}) and the recursion continues. In the end, `GetEff`(n_{20}) returns the



(a) Iteration 2 updates are highlighted.



(b) Iteration 3 updates are highlighted.

Figure 3.5: Decomposition tree for Example 3.5. Nodes needing updates each iteration are highlighted.

Table 3.5: Iteration 2 strategies for $G^{21} = G$ from Example 3.5. Since G^{20} is not updated in iteration 2, these strategies and the associated $\overline{\mathbf{z}^G}(\mathbf{x})$ values derive from the $X_E(G^{11}, \kappa^L)$ sets given in Tables 3.3 and 3.4.

Set	**Strategy**	$\overline{\mathbf{z}^G}(\mathbf{x})$	$b^F = 0$ **path** ($z_0^G(\mathbf{x})$)	$b^F = 1$ **path** ($z_1^G(\mathbf{x})$)
$\widehat{X}_{CE}(G, 3)$	$\{(s, a_1), (a_1, b_1), (c_1, t)\}$	$[56, 28]$	$s \to a_2 \to b_2 \to c_2 \to t$ (42)	$s \to a_1 \to b_1 \to t$ (28)
$\widehat{X}_{CE}(G, 2)$	$\{(s, a_1), (a_1, b_1)\}$	$[46, 28]$	$s \to a_2 \to b_2 \to c_2 \to t$ (42)	$s \to a_1 \to b_1 \to t$ (28)
	$\{(s, a_1), (c_1, t)\}$	$[55, 27]$	$s \to a_2 \to b_2 \to c_2 \to t$ (42)	$s \to a_1 \to b_1 \to t$ (27)
$\widehat{X}_{CE}(G, 1)$	$\{(s, a_1)\}$	$[45, 27]$	$s \to a_2 \to b_2 \to c_2 \to t$ (42)	$s \to a_1 \to b_1 \to t$ (27)
	$\{(c_1, t)\}$	$[50, 22]$	$s \to a_2 \to b_2 \to c_2 \to t$ (42)	$s \to a_1 \to b_1 \to t$ (22)
$\widehat{X}_{CE}(G, 0)$	$\emptyset$	$[40, 22]$	$s \to a_1 \to b_1 \to c_1 \to t$ (40)	$s \to a_1 \to b_1 \to t$ (22)

set of strategies in the G^{20} row of Table 3.4. When `SP*`(r) computes the parallel composition of the strategies of nodes G^{11} and G^{20}, it finds a true complete set of efficient interdiction strategies (a subset of those in the G^{21} row of Table 3.4). The `GetEff`(G) routine returns the collection of strategies to Algorithm 3.3, which then validates them against the shortest paths from the `Dijkstra` routine. The paths are as given in Table 3.4. All arcs on these paths were used in either iteration 1 or 2, and no nodes are flagged for updating.

Since the root does not need updating, the algorithm terminates, returning $\widehat{X}_{CE}(G, b^L)$ after three iterations. It never processed leaf nodes n_4 and n_7 (corresponding to (s, b_1) and (b_2, t), respectively), since none of the paths use these arcs under any of the evaluated interdiction strategies. All other nodes in the tree were processed by `SP*` exactly once, with the exception of the root node, n_{21}, which was processed three times. $\square$

3.5 Computational study

We divide our computational study into two portions: a general graph portion and a portion dedicated to SPGs. The general graph portion examines the difficulty of solving SPIP-I instances on various randomly generated graphs. The SPG portion compares the performance of Algorithms 3.1, 3.2, and 3.3 using a common set of problem instances. In Section 3.5.1 we present our methodology for the study, and Sections 3.5.2 and 3.5.3 examine the results of the general graph and SPG portions, respectively.

3.5.1 Methodology

For this study, we implemented Algorithms 3.1, 3.2, and 3.3 in Python 2.7, and we used iGraph's Python library [16] to generate the SPIP-I instances. All algorithms were run on an Intel Core i5 1.8 GHz processor with 4 GB of 1600 MHz DDR3 RAM. In computing the running time for each algorithm, we ignored the time spent loading the instance graph and the time used by the `Layerize` routine (line 1 of Algorithm 3.1 and line 2 of Algorithm 3.3).

For the general graph portion of our study we generated a total of 980 instances in three batches. Our first batch included 540 SPIP-I instances, where we set $|\mathcal{N}| = 100$ and varied arc densities over values in the set $\{10\%, 20\%, \ldots, 90\%\}$ (where the maximum number of arcs in the graph is given by $|\mathcal{N}| \cdot (|\mathcal{N}| - 1)$) and p over the set $\{1, \ldots, 6\}$. For each (density, p) pair, we generated

Table 3.6: Density by $|\mathcal{A}|$ and $|\mathcal{N}|$.

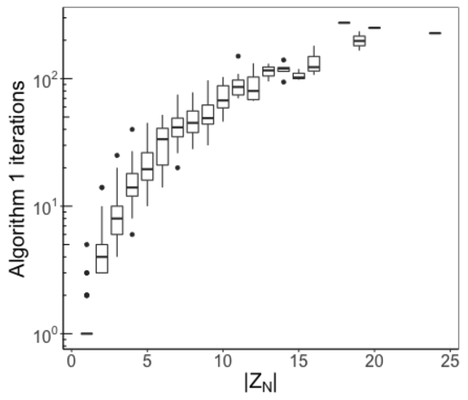
$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{A}\rvert$					
	3960	4950	5940	6930	7920	8910
100	40%	50%	60%	70%	80%	90%
150	17.7%	22.1%	26.6%	31.0%	35%	39.9%
200	9.9%	12.4%	14.9%	17.4%	19.9%	22.4%
250	6.4%	8.0%	9.5%	11.1%	12.7%	14.3%

10 Erős-Rényi random graphs [22], giving a total of 540 instances. The first batch allows us to inspect the running time dependency on $|Z_{\mathcal{N}}|$ and p. For the second batch, we generated an additional 200 instances, holding $p = 4$ constant, varying $|\mathcal{N}| \in \{100, 150, 200, 250\}$ and arc densities over the set $\{30\%, 40\%, 50\%, 60\%, 70\%\}$, and again generating 10 instances per (density, $|\mathcal{N}|$) pair. We used the second batch of instances to examine the running time dependence on $|\mathcal{N}|$. Finally, we generated 240 instances with $p = 4$ constant, and varying $|\mathcal{N}| \in \{100, 150, 200, 250\}$ and $|\mathcal{A}|$ over the values given in Table 3.6. This third batch allowed us to examine the running time dependence on $|\mathcal{A}|$ and the arc density. We set $b^L = 4$ for all instances. For the arc cost functions in each of the 980 instances, we populated (for each arc) a $b^L \times p$ matrix with random integers between 0 and $(10 \cdot \max\{b^L, p\})$. Then we sorted the matrices row-wise in decreasing order, column-wise in increasing order, and again row-wise in decreasing order. This resulted in a $b^L \times p$ matrix of integers that was non-increasing in each row and non-decreasing in each column.

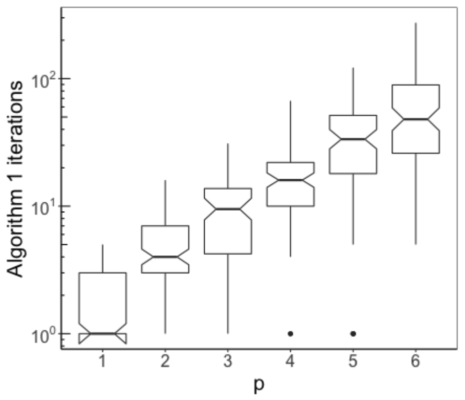
For the comparison of Algorithms 3.1, 3.2, and 3.3 on SPGs, we randomly generated 100 SPGs, fifty graphs each having 100 and 200 nodes. For the 100-node graphs, we used $b^L = 3$ and $p = 4$. For the 200-node graphs we used $b^L = 6$ and $p = 8$. Finally, to show the difference in how the algorithms scale, we generated 10 additional SPGs with 100 nodes, $b^L = 10$, and $p = 50$. The arc costs for SPGs were generated in the same manner as in the random graphs.

3.5.2 Results from the general graph study

We first note that $|Z_{\mathcal{N}}| > 1$ for over 90% of our randomly-generated general graph instances, giving empirical evidence that our multiobjective approach provides value over alternatives such as robust optimization. Algorithm 3.1's average running time to generate a complete efficient set over all 980 instances was 38.69 seconds, with a minimum value of 0.81 seconds and a maximum of 454.83 seconds. These times were influenced by various instance parameters. We display these dependencies in the box-and-whiskers plots of Figures 3.6–3.9. The boxes give the 25th–75th quantile range; the



(a) Batch 1 iterations by $|Z_{\mathcal{N}}|$.



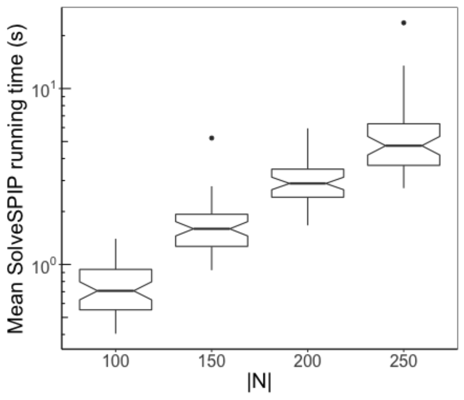
(b) Batch 1 iterations by p.

Figure 3.6: Number of `MOGA` iterations for line 3 of Algorithm 3.1 (see Algorithm 2.1).

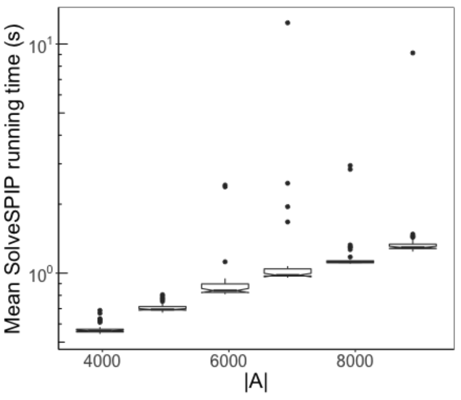
mid-line gives the median value; and the "whiskers" give 1.5 $\times$ inter-quartile range. The notches (when present) give the 95% confidence intervals on the median value, and dots are outliers.

The computational time required by Algorithm 3.1 depends on the number of points in the upper bounding set for the Pareto frontier and the mean time to solve SPIP subproblems on $\bar{G}$. The number of upper bounding points is driven by $|Z_{\mathcal{N}}|$ and p (see Figure 3.6), which corresponds to the result of [12] that bounds the number of upper bounding pounts by $\mathcal{O}(|Z_{\mathcal{N}}|^{\lfloor p/2 \rfloor})$. Meanwhile, the running time of each subproblem is predominantly determined by the time required to execute `SolveSPIP` in Algorithm 2.1, and that time is driven by $|\mathcal{N}|$ and $|\mathcal{A}|$ (see Figure 3.7). Figure 3.8 displays the effect of these parameters on the overall instance running time of Algorithm 3.1. Theoretically and empirically, arc densities do not have a strong influence on the running time of instances in our study (see Figure 3.8e).

Figure 3.8d depicts a decline in the median running time for those instances with the greatest number of arcs. This result is unexpected; however, the overlapping notches indicate that this trend is not significant when we consider the 95% confidence interval on the medians. Moreover, Figure 3.7b indicates a consistent exponential growth in the mean `SolveSPIP` running time with respect to $|\mathcal{A}|$. Thus, we must conclude that this decrease in the median running time for Algorithm 3.1 depicted in Figure 3.8d is a result of fewer iterations within the `MOGA` routine (see Figure 3.9a). Such reduction in the number of iterations may be due to a smaller median $|Z_{\mathcal{N}}|$ (see Figure 3.9b), combined with unusually structured arrangements of $|Z_{\mathcal{N}}|$ in $\mathbb{R}^{p+1}$ that reduce the cardinality of the upper bound sets over $Z_{\mathcal{N}}$.



(a) Batch 2 mean `SPIPSolver` running times by $|\mathcal{N}|$.



(b) Batch 3 mean `SPIPSolver` running times by $|\mathcal{A}|$.

Figure 3.7: Mean running time of `SolveSPIP` (averaged over all calls per instance).

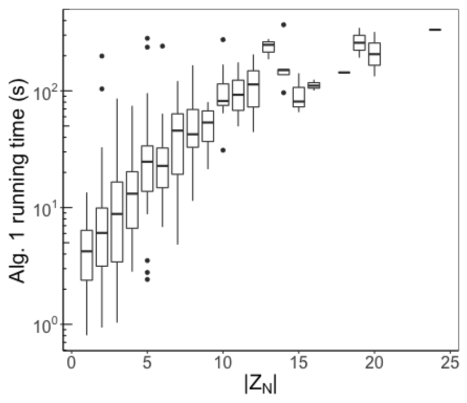
Table 3.7: Algorithm running time summaries for SPG portion of the computational study.

	# of	Algorithm 3.1			Algorithm 3.2			Algorithm 3.3		
Category	Instances	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
$\|Z_{\mathcal{N}}\| = 1$	56	0.04	0.15	1.04	0.60	0.12	13.65	0.01	0.12	0.65
$\|Z_{\mathcal{N}}\| = 2$	24	0.17	0.59	2.90	0.54	2.02	4.81	0.02	0.23	0.60
$\|Z_{\mathcal{N}}\| = 3$	9	0.33	2.63	8.40	0.74	4.66	11.06	0.08	0.49	1.22
$\|Z_{\mathcal{N}}\| = 4$	3	2.13	4.11	7.84	3.78	4.32	5.02	0.30	0.47	0.64
$\|Z_{\mathcal{N}}\| = 5$	3	0.85	1.37	1.70	0.68	1.74	3.79	0.16	0.21	0.24
$\|Z_{\mathcal{N}}\| = 6$	2	6.42	20.88	35.33	4.51	5.01	5.51	0.45	0.71	0.97
$\|Z_{\mathcal{N}}\| = 8$	1	10.68	10.68	10.68	3.59	3.59	3.59	0.83	0.83	0.83
$\|Z_{\mathcal{N}}\| = 10$	2	15.73	46.12	76.51	3.94	4.07	4.20	0.57	0.95	1.34
$\|\mathcal{N}\| = 100$	50	0.04	0.21	1.70	0.54	0.72	1.23	0.01	0.10	0.35
$\|\mathcal{N}\| = 200$	50	0.11	3.94	76.51	3.18	4.80	13.65	0.03	0.36	1.34

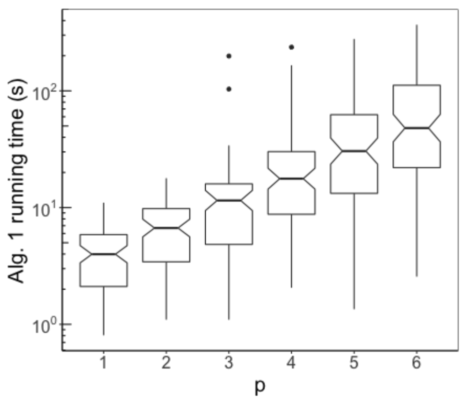
3.5.3 Results from the SPG study

Table 3.7 gives summaries of the running times for Algorithms 3.1, 3.2, and 3.3 among the 100 SPIP-I instances on SPGs, and Figure 3.10 compares these times on log-log plots. The data demonstrates that, among the three algorithms, Algorithm 3.3 consistently performed the best. In Table 3.7 Algorithm 3.3 dominates in every summary statistic for all classes of instances. Figure 3.10b demonstrates that Algorithm 3.1 is faster than Algorithm 3.3 in only a few instances, and in those cases the difference is negligible.

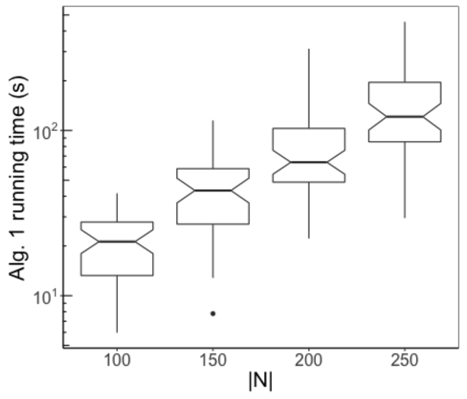
The weakness of Algorithm 3.2 discussed in Example 3.4 is seen by examining the comparative performance of Algorithms 3.1 and 3.2 along with the graph structures. As expected, Algorithm 3.1 is faster than Algorithm 3.2 when nodes s and t are relatively close in the graph, there are just a few



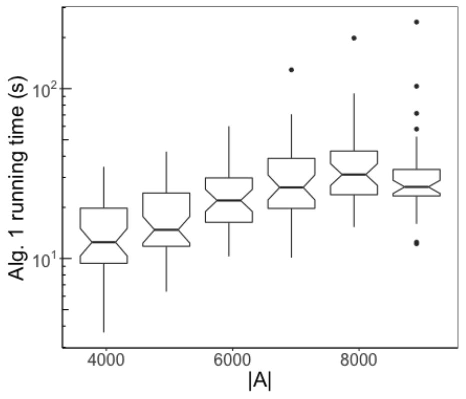
(a) Batch 1 running times by $|Z_{\mathcal{N}}|$.



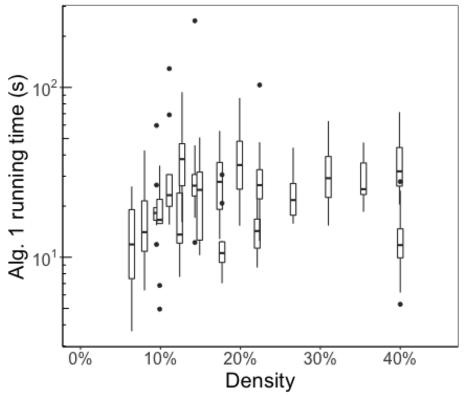
(b) Batch 1 running times by p.



(c) Batch 2 running times by $|\mathcal{N}|$.

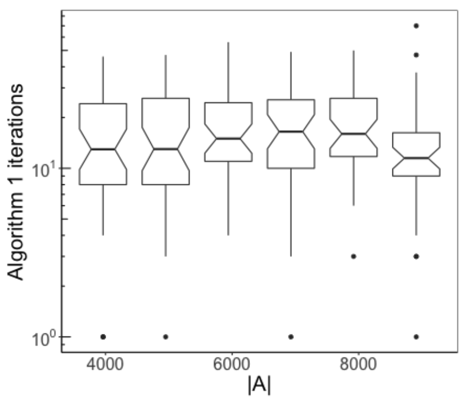


(d) Batch 3 running times by $|\mathcal{A}|$.

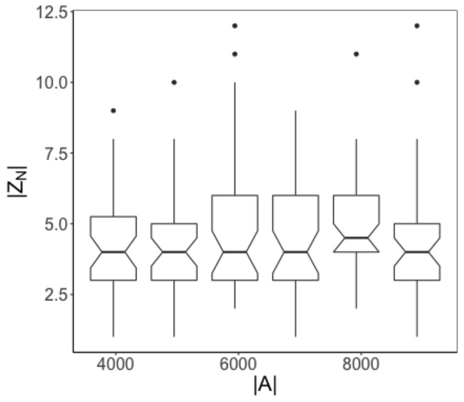


(e) Batch 3 running times by arc density.

Figure 3.8: Running times for Algorithm 3.1 on general graphs.

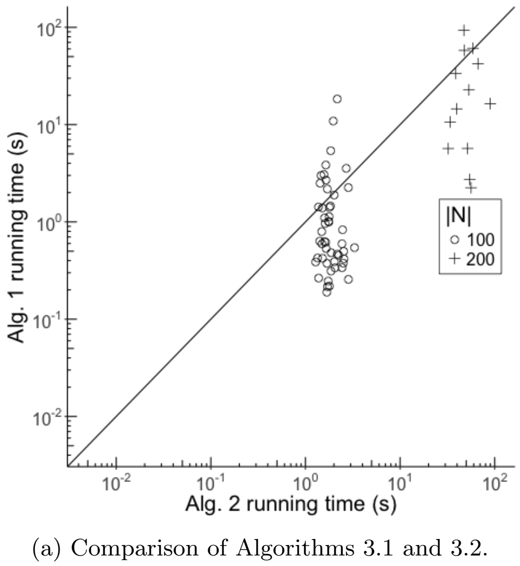


(a) Batch 3 `MOGA` iterations by $|\mathcal{A}|$.

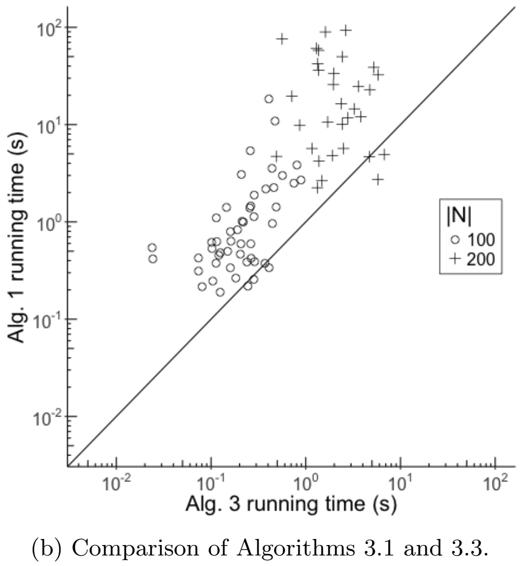


(b) Batch 3 $|Z_{\mathcal{N}}|$ by $|\mathcal{A}|$.

Figure 3.9: Batch 3 results by $|\mathcal{A}|$.

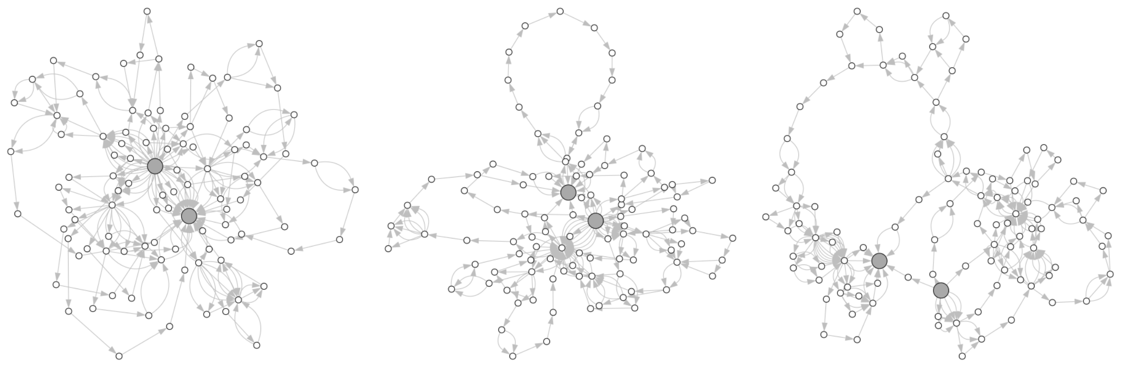


(a) Comparison of Algorithms 3.1 and 3.2.



(b) Comparison of Algorithms 3.1 and 3.3.

Figure 3.10: Log-log plot of running times for Algorithms 3.1, 3.2, and 3.3 on 100 SPIP-I instances with SPGs.



(a) Instance #9 ($|\mathcal{N}| = 100$). (b) Instance #20 ($|\mathcal{N}| = 100$). (c) Instance #24 ($|\mathcal{N}| = 100$).

Figure 3.11: Three instances from the computational study where Algorithm 3.1 is faster than Algorithm 3.2. Nodes s and t (large gray dots) tend to be close in the graph.

short paths between them, and the rest of the arcs form long paths that are not desirable for the follower (see Figure 3.11). The comparatively poor performance of Algorithm 3.2 on these graphs is exacerbated by the efficiences of the row-generating techniques [36, 50] that we use in our Algorithm 3.1 implementation (see Appendix A.2.2). These techniques allow Algorithm 3.1 to ignore paths that are not preferred, directing computational effort to the fewer short paths.

In contrast, Algorithm 3.2 is faster when the nodes s and t are more separated in the graph, the preferred path is not as evident, and the undesirable paths have fewer arcs (see Figure 3.12). In such graphs, $|Z_{\mathcal{N}}|$ also tends to be larger, and Table 3.7 shows that Algorithm 3.2 tended to be faster when $|Z_{\mathcal{N}}| > 5$. This relative dominance by $|Z_{\mathcal{N}}|$ is because Algorithm 3.1's running time is more sensitive to $|Z_{\mathcal{N}}|$ than $|\mathcal{A}|$ (see Figures 3.8a and 3.8d); whereas, from Table 3.7 and Figure 3.10a, Algorithm 3.2's running time is insensitive to $|Z_{\mathcal{N}}|$ and is primarily determined by $|\mathcal{N}|$. Theoretically, this dependence derives from the fact that $|\mathcal{A}| < |\mathcal{N}|^2$ determines the maximum size of the decomposition tree parsed by Algorithm 3.2. Moreover, given the common generation techniques used to produce the SPGs, we find $|\mathcal{A}|$ varied little for a given $|\mathcal{N}|$, which explains why Algorithm 3.2's running time had low variance for a given $|\mathcal{N}|$, as seen in Figure 3.10a.

These comparative strengths of Algorithms 3.1 and 3.2 both contribute to the dominant performance of Algorithm 3.3. When $|Z_{\mathcal{N}}|$ is small, Algorithm 3.3's use of the `UsedArcs` paradigm, similar to the row-generation techniques used in our Algorithm 3.1 implementation (see Appendix A.2.2), restricts parsing effort to the few arcs contained in the follower's optimal recourse paths. Thus, Algorithm 3.3's running time approximates that of Algorithm 3.1 when $|Z_{\mathcal{N}}|$ is small. Alternatively,

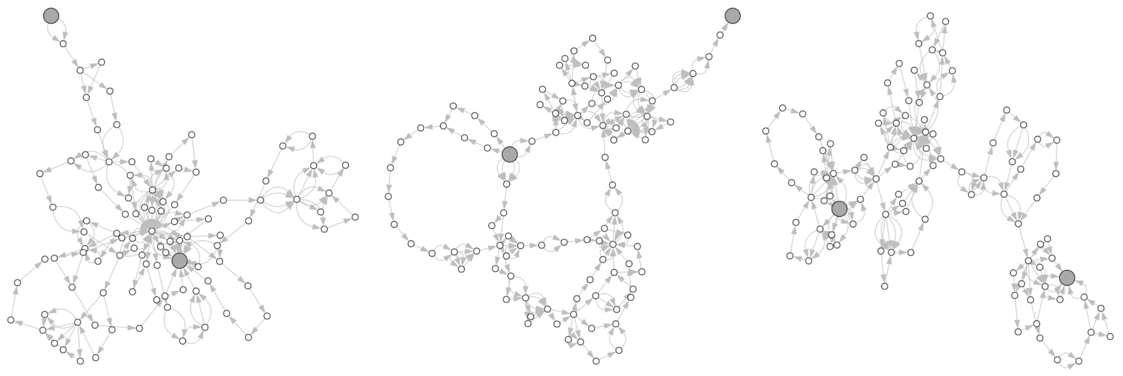

(a) Instance #12 ($|\mathcal{N}| = 100$) (b) Instance #19 ($|\mathcal{N}| = 100$) (c) Instance #22 ($|\mathcal{N}| = 100$)

Figure 3.12: Three instances from the computational study where Algorithm 3.2 is faster than Algorithm 3.1. Nodes s and t (large gray dots) tend to be relatively distanced in the graph.

Table 3.8: Comparison of running times for Algorithms 3.1, 3.2, and 3.3 with p large ($p = n/2$). Columns "n" and "m" column give the number of nodes and arcs in each instance, respectively.

Instance #	n	m	$\lvert Z_{\mathcal{N}}\rvert$	Running Times (s) Alg. 1	Alg. 2	Alg. 3
1	100	197	38	≥ 1800	≥ 1800	196.2
2	100	192	7	≥ 1800	≥ 1800	18.7
3	100	202	34	≥ 1800	446.1	65.8
4	100	204	unk	≥ 1800	≥ 1800	≥ 1800
5	100	195	13	≥ 1800	≥ 1800	147.9
6	100	186	27	≥ 1800	≥ 1800	492.7
7	100	212	15	≥ 1800	≥ 1800	14.9
8	100	177	19	≥ 1800	1364.7	183.8
9	100	198	42	≥ 1800	540.8	32.2
10	100	211	80	≥ 1800	≥ 1800	25.9

when $|Z_{\mathcal{N}}|$ is large, the number of arcs available (and useful) for recourse paths remains bounded. In these instances Algorithm 3.3's running time tends to depend on $|\mathcal{A}|$, similarly to Algorithm 3.2, and avoids a running time bound of $\mathcal{O}(|Z_{\mathcal{N}}|^{\lfloor p/2 \rfloor})$, as in Algorithm 3.1.

Finally, we compared Algorithms 3.1, 3.2, and 3.3 on ten SPGs, using $n = 100$, $b^L = 10$ and $p = 50 = n/2$. Because of the random generation methods, the number of arcs, m, in each graph varied (see Table 3.8). The intent of these instances to examine how the three algorithms scale. Each algorithm was given up to 1800 seconds to solve each of the ten problems. The results are described in Table 3.8. Notably, Algorithm 3.1 failed to solve any of the problems within 1800 seconds, and Algorithm 3.2 only solved three. Algorithm 3.3 solved nine of the ten problems within 1800 seconds.

3.6 Conclusion

The SPIP-I presents an inaugural examination of NIPs where both the leader and the follower take actions to affect arc costs. In the SPIP-I, we consider the problem faced by a leader who has uncertainty regarding the follower's capabilities. When the set of scenarios is relatively limited, a multiobjective problem formulation enables generation of a complete set of efficient interdiction strategies allowing trade-off comparisons between efficient strategies. While general-purpose generating methods such as Algorithm 3.1 are necessary to produce a complete set of efficient strategies on general graphs, the tailored approach given by Algorithm 3.3 performs better on the special class of series-parallel graphs.

The SPIP-I affords many opportunities for additional research. Future research will examine extensions of Algorithm 3.3 to other graph structures. These extensions would permit us to use better-scaling algorithms to a broader range of applications, including supply chain, road, power, and computer networks. Also, the SPIP-I may be extended to other problems in a variety of Stackelberg games, including fortification-by-obfuscation problems. In such problems an actor broadcasts a false set of arc costs that optimally persuades the interdictor to abandon an optimal interdiction strategy. In contrast to classic defender-attacker-defender problems, where the actor's first-stage actions explicitly prevent interdictions, these games would implicitly defend the network by inducing suboptimal interdiction decisions.

Chapter 4

The shortest Path Interdiction Problem with Randomized Strategies

4.1 Problem Statement and Background

The Shortest Path Interdiction Problem with Randomized Strategies (SPIP-RS) is similar to the SPIP. It is a maximin Stackelberg game, where the leader and follower play over a directed network, $G = (\mathcal{N}, \mathcal{A})$. The leader performs exactly $b \in \mathbb{Z}_+$ interdiction actions on the arcs, possibly interdicting some arcs multiple times. The cost for the follower to use arc $a \in \mathcal{A}$, if that arc has been interdicted τ times, is given by $c_a(\tau)$. Defining $\mathcal{H} = \{1, \ldots, b\}$, we assume that $c_a : \{0\} \cup \mathcal{H} \to \mathbb{R}_+$ is non-decreasing.

Unlike the SPIP, however, the SPIP-RS allows the leader to adopt a randomized interdiction strategy. We define Bernoulli random variables χ_{ha} that equal 1 if interdiction h is deployed on arc a and 0 otherwise, $\forall h \in \mathcal{H}$ and $a \in \mathcal{A}$. Since each interdiction action is deployed on exactly one arc, these random variables are not independent. The leader's decision space is $\mathfrak{X} = \{X \in [0,1]^{b \times m} : \sum_{a \in \mathcal{A}} x_{ha} = 1,\ \forall h \in \mathcal{H}\}$, where the components of X are chosen so that $\mathcal{P}(\chi_{ha} = 1) = x_{ha}$. The follower observes X, knows the cost functions for each arc, and seeks a path from node $s \in \mathcal{N}$ to node $t \in \mathcal{N}$ with the minimum expected cost.

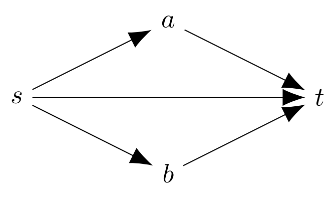


Figure 4.1: Example 1 graph.

Table 4.1: Arc costs for Figure 4.1.

Arc (i,j)	$c_{ij}(0)$	$c_{ij}(1)$	$c_{ij}(2)$
(s,a)	2.5	3.5	5
(s,b)	3.3	4.9	5.7
(s,t)	3.9	4.8	11.1
(a,t)	2	3.5	5.2
(b,t)	2.4	3.2	5.6

Denote the columns of decision matrix X by $\mathbf{x}_a$ for $a \in \mathcal{A}$. Throughout, we refer to $\mathrm{E}\left(c_a(\tau)|\mathbf{x}_a\right)$ as the expected cost of arc $a \in \mathcal{A}$, given vector $\mathbf{x}_a$, where $\tau = \sum_{h\in\mathcal{H}} \chi_{ha}$ acts as a random variable corresponding to the number of times arc a is interdicted given $\mathbf{x}_a$. The leader selects a stochastic interdiction strategy, $X \in \mathfrak{X}$, that maximizes the follower's minimum expected cost. Modifying (1.2), we formulate the following mathematical optimization model for the SPIP-RS:

$$\max_{X\in\mathfrak{X}} \quad d_s \tag{4.1a}$$

$$\text{s.t.} \quad d_i - d_j \leq \mathrm{E}(c_{ij}(\tau)|\mathbf{x}_{ij}) \qquad \forall (i,j) \in \mathcal{A}, \tag{4.1b}$$

$$d_t = 0. \tag{4.1c}$$

Note the change in the right-hand side of (4.1b), so that d_i represents the shortest expected distance from node i to node t.

Example 4.1. Consider the network displayed in Figure 4.1 with cost functions given in Table 4.1. Our example consists of two cases, both using a budget of $b = 2$ interdictions. In the first case we adopt an optimal deterministic interdiction strategy. In the second case we randomize the interdiction strategy, which yields a larger objective for the SPIP-RS than the Case 1 solution.

Case 1: Strategy X^1 is an optimal deterministic interdiction strategy, with expected costs given in Table 4.2. The follower opts to use arc $s \to t$ at an expected cost of 4.8.

$$X^1 = \begin{array}{c} \begin{array}{ccccc} \scriptstyle (s,a) & \scriptstyle (s,b) & \scriptstyle (s,t) & \scriptstyle (a,t) & \scriptstyle (b,t) \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Table 4.2: Expected path costs given X^1.

Path	Expected cost
$s \to a \to t$	$3.5 + 2 = 5.5$
$s \to b \to t$	$3.3 + 2.4 = 5.7$
$s \to t$	4.8

Case 2: The stochastic interdiction strategy X^2 applies a positive probability of interdicting each arc. The expected path costs given X^2 are given in Table 4.3. The follower now chooses path $s \to a \to t$ with expected cost 5.4285.

$$X^2 = \begin{array}{c} \begin{array}{ccccc} \scriptstyle (s,a) & \scriptstyle (s,b) & \scriptstyle (s,t) & \scriptstyle (a,t) & \scriptstyle (b,t) \end{array} \\ \left[\begin{array}{ccccc} 0.3 & 0.05 & 0.5 & 0.1 & 0.05 \\ 0.35 & 0.05 & 0.5 & 0.05 & 0.05 \end{array}\right] \end{array}$$

Table 4.3: Expected path costs given X^2.

Path	Expected cost
$s \to a \to t$	$= 5.4285$
$s \to b \to t$	$= 6.5228$
$s \to t$	$= 6.15$

□

In this chapter we examine the complexity of SPIP-RS under different assumptions regarding the cost functions. In many classic SPIP formulations and in the more recent work of [2, 9, 15] the cost functions are affine. If this property holds, then $\mathrm{E}(c_a(\tau)|\mathbf{x}_a) = c_a(\sum_{h \in \mathcal{H}} x_{ha})$ for all $a \in \mathcal{A}$, and model (4.1) is a linear program. Thus, SPIP-RS is polynomially solvable in this case (as shown by [2, 24]). For, the more general case in which the cost functions are non-linear, we use the following definitions for convexity and concavity of discrete cost functions: c_a is a *discrete-convex* (or *-concave*) *function* if $c_a(t) - 2c_a(t+1) + c_a(t+2) \geq 0$ (or ≤ 0) for all $t \in \{0, \dots, b-2\}$. Since we only use discrete cost functions, for brevity we will simply refer to these properties as convexity or concavity.

To perform our complexity analysis, we will examine the decision problem associated with the SPIP-RS:

DSPIP-RS

Input: Given the following inputs:

- Digraph, $G = (\mathcal{N}, \mathcal{A})$ with source $s \in \mathcal{N}$ and destination $t \in \mathcal{N}$
- Number of interdictions, $b \in \mathbb{Z}_+$
- Discrete, non-decreasing arc cost functions, c_a, for each $a \in \mathcal{A}$
- Threshold, $z^* \geq 0$

Question: Does there exist $(X, \mathbf{d}) \in \mathfrak{X} \times \mathbb{R}^n$ feasible to (4.1b)–(4.1c) with $d_s \geq z^*$?

We first show that DSPIP-RS is in NP, i.e., given an interdiction strategy X, show in polynomial time that the follower's shortest path is indeed of length at least z^*. Given the expected arc costs, this is accomplished using Dijkstra's algorithm; however, the randomized strategies of the SPIP-RS make computing an arc's expected cost non-trivial. Solving for $\mathrm{E}(c_a(\sigma_a)|\mathbf{x}_a)$ with non-linear cost functions by enumerating the scenarios requires an exponential number of computations. We instead present an algorithm to generate this value in polynomial time using a recursive technique.

Let $h \in \{0\} \cup \mathcal{H}$ and define $\Pi^h(\mathbf{x}_a, t)$ as the probability that arc $a \in \mathcal{A}$ has been interdicted exactly t times among interdictions $\{1, \dots, h\}$, given $\mathbf{x}_a$. Define $\Pi^0(\mathbf{x}_a, 0) = 1$. Thus,

$$\Pi^h(\mathbf{x}_a, t) = \begin{cases} 1, & j = 0, t = 0 \\ \mathcal{P}(\sum_{h=1}^{j} \chi_h = t | \mathbf{x}_a) & 0 < h \leq b, \\ 0, & \text{otherwise.} \end{cases}$$

Then we have the recursive formula for $h \in \mathcal{H}$:

$$\Pi^h(\mathbf{x}_a, t) = x_{ha}\Pi^{h-1}(\mathbf{x}_a, t-1) + (1 - x_{ha})\Pi^{h-1}(\mathbf{x}_a, t). \tag{4.2}$$

Using (4.2) we can compute $\Pi^h(\mathbf{x}_a, 0), \dots, \Pi^h(\mathbf{x}_a, b)$, in increasing order of $h = 0, \dots, b$ with $\mathcal{O}(b^2)$ operations. The final expected cost of arc $a \in \mathcal{A}$ is computed using $\mathcal{O}(b)$ additional operations via the standard expectation formula:

$$\mathrm{E}(c_a(\tau)|\mathbf{x}_a) = \sum_{\tau=0}^{b} c_a(\tau)\Pi^b(\mathbf{x}_a, \tau). \tag{4.3}$$

We demonstrate how the recursion given by (4.2) may be used to compute $\Pi(\mathbf{x}_a, \sigma)$ in $\mathcal{O}(b^2)$ computations in Example 4.2 below.

Example 4.2. We reconsider Case 2 from Example 1, showing the steps to compute $\Pi(\mathbf{x}_{(s,t)}, \sigma)$ in $\mathcal{O}(b^2)$ time.

$$\Pi = \begin{array}{c} \\ \sigma{=}0 \\ \sigma{=}1 \\ \sigma{=}2 \end{array} \begin{array}{c} \begin{array}{ccc} j{=}0 & j{=}1 & j{=}2 \end{array} \\ \begin{bmatrix} \Pi^0(\mathbf{x}_{(s,t)}, 0) & \Pi^1(\mathbf{x}_{(s,t)}, 0) & \Pi^2(\mathbf{x}_{(s,t)}, 0) \\ \Pi^0(\mathbf{x}_{(s,t)}, 1) & \Pi^1(\mathbf{x}_{(s,t)}, 1) & \Pi^2(\mathbf{x}_{(s,t)}, 1) \\ \Pi^0(\mathbf{x}_{(s,t)}, 2) & \Pi^1(\mathbf{x}_{(s,t)}, 2) & \Pi^2(\mathbf{x}_{(s,t)}, 2) \end{bmatrix} \end{array}$$

$$= \begin{bmatrix} 1 & x_{1,(s,t)}\Pi^0(\mathbf{x}_{(s,t)}, 0) + (1 - x_{1,(s,t)})\Pi^0(\mathbf{x}_{(s,t)}, -1) & \Pi^2(\mathbf{x}_{(s,t)}, 0) \\ 0 & x_{1,(s,t)}\Pi^0(\mathbf{x}_{(s,t)}, 1) + (1 - x_{1,(s,t)})\Pi^0(\mathbf{x}_{(s,t)}, 0) & \Pi^2(\mathbf{x}_{(s,t)}, 1) \\ 0 & x_{1,(s,t)}\Pi^0(\mathbf{x}_{(s,t)}, 2) + (1 - x_{1,(s,t)})\Pi^0(\mathbf{x}_{(s,t)}, 1) & \Pi^2(\mathbf{x}_{(s,t)}, 2) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & (50\%)(1)+(50\%)(0) & x_{2,(s,t)}\Pi^1(\mathbf{x}_{(s,t)},0)+(1-x_{2,(s,t)})\Pi^1(\mathbf{x}_{(s,t)},-1) \\ 0 & (50\%)(0)+(50\%)(1) & x_{2,(s,t)}\Pi^1(\mathbf{x}_{(s,t)},1)+(1-x_{2,(s,t)})\Pi^1(\mathbf{x}_{(s,t)},0) \\ 0 & (50\%)(0)+(50\%)(0) & x_{2,(s,t)}\Pi^1(\mathbf{x}_{(s,t)},2)+(1-x_{2,(s,t)})\Pi^1(\mathbf{x}_{(s,t)},1) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 50\% & (50\%)(50\%)+(50\%)(0) \\ 0 & 50\% & (50\%)(50\%)+(50\%)(50\%) \\ 0 & 0 & (50\%)(0)+(50\%)(50\%) \end{bmatrix} = \begin{bmatrix} 1 & 50\% & 25\% \\ 0 & 50\% & 50\% \\ 0 & 0 & 25\% \end{bmatrix}.$$

The final column gives the probabilities of the levels of interdiction on arc (s,t), and using (4.3), the expected cost is given as

$$\mathrm{E}(c_a(\sigma_a)|P) = \sum_{\sigma=0}^{b} c_a(\sigma)\Pi(\mathbf{p}_a,\sigma) = \sum_{\sigma=0}^{b} c_a(\sigma)\Pi^2(\mathbf{p}_a,\sigma) = 3.9(25\%) + 4.8(50\%) + 11.1(25\%) = 6.15.$$

□

Lemma 4.1. DSPIP-RS belongs to NP.

Proof. Given a certificate $(X, \mathbf{d})$ we can compute the expected costs, $\mathrm{E}(c_a(\tau)|\mathbf{x}_a)$, $\forall a \in \mathcal{A}$, in polynomial time. Then, a polynomial-time verification step can verify the objective and feasibility of $(X, \mathbf{d})$. □

The remainder of this chapter is organized as follows. In Section 4.2 we show that DSPIP-RS is NP-complete when all costs functions are convex, and provide a spatial branch-and-bound approach to solving the SPIP-RS. In Section 4.3 we examine the case in which all cost functions are concave. We prove that DSPIP-RS is NP-hard in this case as well, and we present an efficient approximation algorithm. We then provide a sample average approximation approach for the SPIP-RS under general cost functions in Section 4.4. Section 4.5 describes the methods and results of our computational study to examine the performance of our algorithms. Finally, we conclude with future research directions in Section 4.6.

4.2 The Convex Case

In this section we focus on the convex SPIP-RS, i.e., the SPIP-RS in which c_a is convex for all $a \in \mathcal{A}$. We show that the convex SPIP-RS is NP-hard in Section 4.2.1, and provide a tailored algorithm to solve this problem in Section 4.2.2.

4.2.1 Complexity

We begin this section by establishing a property of optimal solutions to the convex SPIP-RS.

Lemma 4.2. Let c_a be convex for each $a \in \mathcal{A}$. There exists a vector $\mathbf{t} \in \mathbb{R}^m$: $\mathbf{t} \geq \mathbf{0}$ and $\sum_{a=1}^{m} t_a = b$ such that

$$X^*(\mathbf{t}) = \begin{bmatrix} {}^{t_1}\!/_b & {}^{t_2}\!/_b & \cdots & {}^{t_m}\!/_b \\ \vdots & \vdots & \ddots & \vdots \\ {}^{t_1}\!/_b & {}^{t_2}\!/_b & \cdots & {}^{t_m}\!/_b \end{bmatrix} \tag{4.4}$$

and a corresponding vector $\mathbf{d}$, such that $(X^*(\mathbf{t}), \mathbf{d})$ is an optimal solution to (4.1).

Proof. Consider an optimal solution $(X, \mathbf{d})$ and define $t_a = \sum_{h \in \mathcal{H}} x_{ha}$, $\forall a \in \mathcal{A}$. We show that $(X^*(\mathbf{t}), \mathbf{d}^*)$ is an alternative optimal solution to (4.1). To accomplish this, it is sufficient to show that $\mathrm{E}(c_a(\tau)|\mathbf{x}_a^*(\mathbf{t})) \geq \mathrm{E}(c_a(\tau)|\mathbf{x}_a)$ for each $a \in \mathcal{A}$, since that implies $d_a^* \geq d_a$ by (4.1b).

Let $a \in \mathcal{A}$. If $\mathbf{x}_a = \mathbf{x}_a^*(\mathbf{t})$ $(= [t_a/b, \ldots, t_a/b])$, then the result holds by equality. Otherwise, suppose that $\mathbf{x}_a \neq \mathbf{x}_a^*(\mathbf{t})$, which also implies that $0 < t_a < b$. Assume (without loss of generality) that $x_{ba} > x_{(b-1)a}$. Let $\boldsymbol{\delta} = [0, \ldots, 0, \epsilon, -\epsilon]$ for some $0 < \epsilon < x_{ba} - x_{(b-1)a}$. Then we can expand $\mathrm{E}(c_a(\tau)|\mathbf{x}_a + \boldsymbol{\delta})$ using two recursion steps from (4.2), to obtain the following expression where $\Pi^h(\mathbf{x}_a, -1) = \Pi^h(\mathbf{x}_a, -2) = 0$ for all h:

$$\begin{aligned} \mathrm{E}(c_a(\tau)|\mathbf{x}_a + \boldsymbol{\delta}) = \sum_{\tau=0}^{b} c_a(\tau)\Big(& (x_{ba} - \epsilon)(x_{(b-1)a} + \epsilon)\Pi^{b-2}(\mathbf{x}_a, \tau - 2) \\ & + (1 - (x_{ba} - \epsilon))(x_{(b-1)a} + \epsilon)\Pi^{b-2}(\mathbf{x}_a, \tau - 1) \\ & + (x_{ba} - \epsilon)(1 - (x_{(b-1)a} + \epsilon))\Pi^{b-2}(\mathbf{x}_a, \tau - 1) \\ & + (1 - (x_{ba} - \epsilon))(1 - (x_{(b-1)a} + \epsilon))\Pi^{b-2}(\mathbf{x}_a, \tau)\Big). \end{aligned} \tag{4.5}$$

Applying the same recursion to $\mathrm{E}(c_a(\tau)|\mathbf{x}_a)$ yields the same expression as (4.5), but with $\epsilon = 0$.

With these, we have the following:

$$\begin{aligned} & \mathrm{E}(c_a(\tau)|\mathbf{x}_a + \boldsymbol{\delta}) - \mathrm{E}(c_a(\tau)|\mathbf{x}_a) \\ & \quad = \sum_{\tau=0}^{b} c_a(\tau)\Big(\Big(\epsilon x_{ba} - \epsilon x_{(b-1)a} - \epsilon^2\Big)\Pi^{b-2}(\mathbf{x}_a, \tau - 2) \\ & \qquad\qquad + \Big(\epsilon x_{(b-1)a} - \epsilon x_{ba} + \epsilon^2 + \epsilon\Big)\Pi^{b-2}(\mathbf{x}_a, \tau - 1) \end{aligned}$$

$$+ \Big(\epsilon x_{(b-1)a} - \epsilon x_{ba} + \epsilon^2 - \epsilon\Big)\Pi^{b-2}(\mathbf{x}_a, \tau - 1)$$

$$+ \Big(\epsilon x_{ba} - \epsilon x_{(b-1)a} - \epsilon^2\Big)\Pi^{b-2}(\mathbf{x}_a, \tau)\Big)$$

$$= \big(\epsilon(x_{ba} - x_{(b-1)a}) - \epsilon^2\big) \sum_{\tau=0}^{b} c_a(\tau)\Big(\Pi^{b-2}(\mathbf{x}_a, \tau - 2) - 2\Pi^{b-2}(\mathbf{x}_a, \tau - 1) + \Pi^{b-2}(\mathbf{x}_a, \tau)\Big)$$

$$= \big(\epsilon(x_{ba} - x_{(b-1)a}) - \epsilon^2\big) \sum_{\tau=0}^{b-2} \Pi^{b-2}(\mathbf{x}_a, \tau)\Big(c_a(\tau) - 2c_a(\tau+1) + c_a(\tau+2)\Big) \geq 0.$$

The inequality holds because $0 < \epsilon < x_{ba} - x_{(b-1)a} \leq 1$, implying that $\epsilon(x_{ba} - x_{(b-1)a}) - \epsilon^2 > 0$, and because c_a is convex, implying that $c_a(\tau) - 2c_a(\tau+1) + c_a(\tau+2) \geq 0$ for all $\tau = 0, \ldots, b-2$. Repeating this process, we modify X until the interdiction matrix coincides with $X^*(\mathbf{t})$, which must also be an alternative optimal solution. This completes the proof. $\square$

Lemma 4.2 implies that if an optimal expected number of interdictions, t_a, is known for each arc $a \in \mathcal{A}$, then $X^*(\mathbf{t})$ gives an optimal variable assignment for the SPIP-RS, with the expected cost of arc a being $\sum_{\tau=0}^{b} c_a(\tau)\binom{b}{\tau}\left({}^{t_a}/_b\right)^\tau \left(1 - {}^{t_a}/_b\right)^{b-\tau}$. Thus, model (4.1) simplifies to

$$\max \quad d_s \tag{4.6a}$$

$$\text{s.t.} \quad d_i - d_j \leq \sum_{\tau=0}^{b} c_{ij}(\tau)\binom{b}{\tau}\left({}^{t_{ij}}/_b\right)^\tau \left(1 - {}^{t_{ij}}/_b\right)^{b-\tau} \qquad \forall (i,j) \in \mathcal{A}, \tag{4.6b}$$

$$d_t = 0 \tag{4.6c}$$

$$\sum_{(i,j)\in\mathcal{A}} t_{ij} = b \tag{4.6d}$$

$$\mathbf{t} \geq \mathbf{0}. \tag{4.6e}$$

We can now prove the hardness of DSPIP-RS for the convex case.

Theorem 4.3. DSPIP-RS is NP-complete, even when c_a is convex for all $a \in \mathcal{A}$.

Proof. We employ a reduction from VERTEX COVER to DSPIP-RS modified for (4.6), i.e., the question becomes "$\exists(\mathbf{t}, \mathbf{d}) \in \mathbb{R}^m \times \mathbb{R}^n$ feasible to (4.6b)–(4.6e) such that $d_s \geq z^*$?" The VERTEX COVER problem is strongly NP-complete [25] and is stated as:

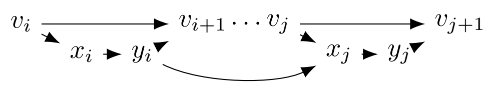


Figure 4.2: Gadgets used in proof of Theorem 4.3.

Table 4.4: Costs for arcs in Figure 4.2.

Arc (a)	$c_a(\tau)$, $\tau \in \{0, \ldots, b-1\}$	$c_a(b)$
(v_i, v_{i+1})	1	1
(v_i, x_i)	1	1
(x_i, y_i)	0	$2r^b$
(y_i, v_{i+1})	0	0
(y_i, x_j)	$j-i-1$	$j-i-1$

VERTEX COVER

Input: Given the following inputs:

- An undirected graph, $G = (V, E)$
- Threshold, $r \in \{1, \ldots, |V|\}$

Question: Does there exist a vertex cover for G of size at most r, i.e., $\exists U \subseteq V$ s.t. $|U| \leq r$ and $U \cap \{i, j\} \neq \emptyset$ for all $(i, j) \in E$?

Given a VERTEX COVER instance, let $|V| = n$. Our graph for the DSPIP-RS instance, $G' = (\mathcal{N}', \mathcal{A}')$, is composed of n "gadgets." These gadgets are similar to those used by [4] for their complexity analysis of the most vital arcs problem. Gadget i consists of nodes $\{v_i, x_i, y_i, v_{i+1}\}$ and arcs $\{(v_i, x_i), (x_i, y_i), (y_i, v_{i+1}), (v_i, v_{i+1})\}$. Also there are "shortcut" arcs between gadgets: $\{(y_i, x_j) : (i, j) \in E; \ i < j\}$. Figure 4.2 illustrates gadgets for two nodes, i and j. The follower's source node is $s = v_1$ and the destination is $t = v_{n+1}$. We set the number of interdictions

$$b = \left\lfloor \log_{r/r+0.5} \left(\frac{1}{4}\right) \right\rfloor = \left\lfloor \frac{\ln\left(1/4\right)}{\ln\left(r/(r+0.5)\right)} \right\rfloor,$$

and assign cost functions for $i \in V$ and $(i, j) \in E$ with $i < j$ according to Table 4.4. Observe that for any $a \in \mathcal{A}'$, c_a is non-decreasing and convex. Finally, our cost threshold for the DSPIP-RS is $z^* = n$. It is evident that G' can be constructed in polynomial time. Also, all parameters other than $c_{(x_i,y_i)}(b) = 2r^b$ are clearly polynomial in size; moreover, since $b \leq 3r + 1$ for $r \geq 0$, the encoding size for these costs are also polynomial as well. We now show that the VERTEX COVER instance has a solution if and only if there exists a feasible solution $(\mathbf{t}, \mathbf{d})$ to (4.6) having $d_s \geq n$.

Assume that $U \subseteq V$ is a vertex cover for G of size $|U| = r$ (we assume equality without loss of generality). For each $i \in U$, we set $t_{(x_i,y_i)} = b/r$, and we set $t_a = 0$ for all other $a \in \mathcal{A}'$ (so that $\mathbf{t}$ satisfies (4.6d) and (4.6e)). Lastly, we set the components of $\mathbf{d}$ as follows:

- $d_{v_i} = n - i + 1$ for $i \in \{1, \ldots, n+1\}$,

- $d_{x_i} = n - i + 1$ for $i \in \{1, \ldots, n\} \cap U$ and $d_{x_i} = n - i$ for $i \in \{1, \ldots, n\} \setminus U$,
- $d_{y_i} = n - i - 1$ for $i \in \{1, \ldots, n\} \cap U$, and $d_{y_i} = n - i$ for $i \in \{1, \ldots, n\} \setminus U$.

We note that (4.6c) is satisfied since $d_t = d_{v_{n+1}} = n - (n + 1) + 1 = 0$. Next, we show that the solution $(\mathbf{t}, \mathbf{d})$ satisfies (4.6b) by examining the left-hand side (LHS) and right-hand side (RHS) of (4.6b) for every arc in $\mathcal{A}'$.

Arc (v_i, v_{i+1}) *for* $i \in \{1, \ldots, n\}$. The LHS of (4.6b) for this arc is $d_{v_i} - d_{v_{i+1}} = (n-i+1)-(n-i) = 1$, while the RHS is 1 regardless of $\mathbf{t}$. Thus, constraint (4.6b) is satisfied.

Arc (v_i, x_i) *for* $i \in \{1, \ldots, n\}$. Again, $d_{v_i} = n - i + 1$. If $i \in U$ then $d_{x_i} = n - i + 1$; otherwise, $d_{x_i} = n - i$. The LHS of (4.6b) is therefore either 0 or 1. The RHS is 1 regardless of $\mathbf{t}$, thus satisfying (4.6b).

Arc (x_i, y_i) *for* $i \in \{1, \ldots, n\}$. There are two cases to consider.

Case 1: $i \in U$, implying that $t_{(x_i,y_i)} = {}^{b}\!/_{r}$. The LHS of (4.6b) is $d_{x_i} - d_{y_i} = (n - i + 1) - (n - i - 1) = 2$, while the RHS evaluates to $2r^b({}^{b}\!/_{r})^b \geq 2$.

Case 2: $i \notin U$, implying that $t_{(x_i,y_i)} = 0$. The LHS of (4.6b) is $d_{x_i} - d_{y_i} = (n-i)-(n-i) = 0$, while the LHS is 0 because $t_{(x_i,y_i)} = 0$.

In either case, constraint (4.6b) is satisfied.

Arc (y_i, v_{i+1}) *for* $i \in \{1, \ldots, n\}$. We set d_{y_i} to either $n - i - 1$ or $n - i$, and $d_{v_{i+1}} = n - i$. The LHS of (4.6b) is either 0 or -1, while the RHS is 0 regardless of $\mathbf{t}$, thus satisfying (4.6b).

Arc (y_i, x_j) *for* $(i, j) \in E$. Because $(i, j) \in E$ and U is a cover, then either i or j (or both) belong to U. If both i and j belong to U, then the LHS of (4.6b) is $d_{y_i} - d_{x_j} = (n-i-1)-(n-j+1) = j-i-2$. If only i belongs to U, then the LHS is $(n - i - 1) - (n - j) = j - i - 1$, and if only j belongs to U, then the LHS is $(n - i) - (n - j + 1) = j - i - 1$. The maximum value that the LHS can take is $j - i - 1$; furthermore, regardless of $\mathbf{t}$, the RHS of (4.6b) for this arc is $j - i - 1$, so the constraint is satisfied.

Having demonstrated that $(\mathbf{t}, \mathbf{d})$ satisfies (4.6b)–(4.6e), the first direction of our proof is completed by showing $d_s = d_{v_1} = n - 1 + 1 = n$, implying that the transformed DSPIP-RS instance also has a solution.

Conversely, assume there exists a solution $(\mathbf{t}, \mathbf{d})$ feasible to (4.6) with $d_s \geq n$. We construct $U = \{i \in V : t_{(x_i,y_i)} \geq \left(b \sqrt[b]{1/4}\right)/r\}$. To show that U is a cover, assume by contradiction that $(i,j) \in E$ is not covered by U; i.e., $t_{(x_i,y_i)}, t_{(x_j,y_j)} < \left(b \sqrt[b]{1/4}\right)/r$. We show that there exists a path from s to t that constrains $d_{v_s} < n$, which leads to the desired contradiction. From repeated application of (4.6b) we have

$$\begin{aligned}
d_{v_{j+1}} &\leq d_{v_{j+2}} + 1 \leq d_{v_{j+3}} + 2 \leq \cdots \leq d_{v_{n+1}} + n - j = n - j \\
\Rightarrow d_{y_j} &\leq d_{v_{j+1}} + \sum_{\tau=0}^{b} c_{(y_j,v_{j+1})}(\tau) \binom{b}{\tau} \left(t_{(y_j,v_{j+1})}/b\right)^\tau = (n-j) + 0 = n - j \\
\Rightarrow d_{x_j} &\leq d_{y_j} + \sum_{\tau=0}^{b} c_{(x_j,y_j)}(\tau) \binom{b}{\tau} \left(t_{(x_j,y_j)}/b\right)^\tau = d_{y_j} + c_{(x_j,y_j)}(b) \left(t_{(x_j,y_j)}/b\right)^b \\
&< (n-j) + \left(2r^b\right) \left(\frac{b\sqrt[b]{1/4}}{br}\right)^b = n - j + 1/2 \\
\Rightarrow d_{y_i} &\leq d_{x_j} + \sum_{\tau=0}^{b} c_{(x_j,y_j)}(\tau) \binom{b}{\tau} \left(t_{(y_i,x_j)}/b\right)^\tau = d_{x_j} + c_{(y_i,x_j)}(0) \binom{b}{0} (0/b)^0 \\
&< (n - j + 1/2) + (j - i - 1) = n - i - 1/2 \\
\Rightarrow d_{x_i} &\leq d_{y_i} + \sum_{\tau=0}^{b} c_{(x_i,y_i)}(\tau) \binom{b}{\tau} \left(t_{(x_i,y_i)}/b\right)^\tau = d_{y_i} + c_{(x_i,y_i)}(b) \binom{b}{b} \left(t_{(x_i,y_i)}/b\right)^b \\
&< (n - i - 1/2) + \left(2r^b\right) \left(\frac{b\sqrt[b]{1/4}}{br}\right)^b = n - i \\
\Rightarrow d_{v_i} &\leq d_{x_i} + \sum_{\tau=0}^{b} c_{(v_i,x_i)}(\tau) \binom{b}{\tau} \left(t_{(v_i,x_i)}/b\right)^\tau = d_{x_i} + c_{(v_i,x_i)}(0) \binom{b}{0} (0/b)^0 \\
&< (n-i) + 1 = n - i + 1 \\
\Rightarrow d_s = d_{v_1} &\leq d_{v_2} + 1 \leq \cdots \leq d_{v_i} + (i-1) \\
&< (n - i + 1) + (i - 1) = n,
\end{aligned}$$

which contradicts our assumption that $d_s \geq n$. Thus, we conclude that U is a cover. It remains to show that $|U| \leq r$. Note from (4.6d) that $|U|$ is no more than b divided by the threshold interdiction level required to place a node in U, i.e.,

$$|U| \leq \frac{b}{\left(b\sqrt[b]{1/4}\right)/r}. \tag{4.7}$$

Moreover, since $|U|$ is integer, we can take the floor of the RHS of (4.7), yielding with a slight rearrangement $|U| \leq \lfloor r/(\sqrt[b]{1/4}) \rfloor$. Recalling $b = \lfloor \log_{r/r+0.5}(1/4) \rfloor \Rightarrow r/(r+0.5) \leq \sqrt[b]{1/4}$, we then obtain

$$|U| \leq \left\lfloor \frac{r}{\sqrt[b]{1/4}} \right\rfloor \leq \lfloor (r+0.5) \rfloor = r.$$

Because DSPIP-RS belongs to NP as shown in Lemma 4.1, this completes the proof. □

4.2.2 Spatial Branch-and-Bound Algorithm for the Convex Case

Our proposed algorithm for the convex SPIP-RS leverages the following result regarding the RHS of (4.6b).

Proposition 4.1. If c_a is convex, then the function $f(t_a) = \sum_{\tau=0}^{b} c_{ij}(\tau)\binom{b}{\tau}(t_a/b)^{\tau}(1 - t_a/b)^{b-\tau}$ from (4.6b) is convex.

Proof. Let $t_a \in [0, b]$, and for notational ease, let $\hat{t}_a = t_a/b$. Define $\mathbf{x}_a = [\hat{t}_a, \ldots, \hat{t}_a]$. Note that the first two derivatives of $f(\hat{t}_a)$ are:

$$\begin{aligned}
f'\left(\hat{t}_a\right) &= \sum_{\tau=0}^{b} c_a(\tau)\binom{b}{\tau}\Big((\tau)\left(\hat{t}_a\right)^{\tau-1}\left(1-\hat{t}_a\right)^{b-\tau} - (b-\tau)\left(\hat{t}_a\right)^{\tau}\left(1-\hat{t}_a\right)^{b-\tau-1}\Big)\\
\Rightarrow f''\left(\hat{t}_a\right) &= \sum_{\tau=0}^{b} c_a(\tau)\binom{b}{\tau}\Big((\tau)(\tau-1)\left(\hat{t}_a\right)^{\tau-2}\left(1-\hat{t}_a\right)^{b-\tau}\\
&\qquad - 2(\tau)(b-\tau)\left(\hat{t}_a\right)^{\tau-1}\left(1-\hat{t}_a\right)^{b-\tau-1}\\
&\qquad + (b-\tau)(b-\tau-1)\left(\hat{t}_a\right)^{\tau}\left(1-\hat{t}_a\right)^{b-\tau-2}\Big).
\end{aligned}$$

Now regrouping the summed form for $f''(\hat{t}_a)$, we have

$$\begin{aligned}
f''(\hat{t}_a) = \sum_{\tau=0}^{b-2}\left(\hat{t}_a\right)^{\tau}\left(1-\hat{t}_a\right)^{b-\tau-2}\Bigg(& c_a(\tau+2)\left[(\tau+2)(\tau+1)\binom{b}{\tau+2}\right]\\
& - 2c_a(\tau+1)\left[(\tau+1)(b-\tau-1)\binom{b}{\tau+1}\right]\\
& + c_a(\tau)\left[(b-\tau)(b-\tau-1)\binom{b}{\tau}\right]\Bigg).
\end{aligned}$$

Observing that

$$(\tau+1)(\tau+2)\binom{b}{\tau+2} = (\tau+1)(b-\tau-1)\binom{b}{\tau+1} = (b-\tau)(b-\tau-1)\binom{b}{\tau},$$

we can simplify $f''(\hat{t}_a)$ as:

$$f''(\hat{t}_a) = \sum_{\tau=0}^{b-2} (c_a(\tau+2) - 2c_a(\tau+1) + c_a(\tau)) \left(\frac{b!}{\tau!(b-\tau-2)!}\right) \left(\hat{t}_a\right)^{\tau} \left(1-\hat{t}_a\right)^{b-\tau-2} \geq 0,$$

where the inequality holds by convexity of c_a and the fact that $0 \leq \hat{t}_a \leq 1$. $\square$

By Proposition 4.1, formulation (4.6) is a non-convex problem. Appendix A.3.1 provides a spatial branch-and-bound [23] algorithm to solve the SPIP-RS. It solves over the convex hull of (4.6b)–(4.6e), and then repeatedly branches and tightens each subproblem relaxation to find a solution whose objective value is arbitrarily close to the true optimal objective.

4.3 The Concave Case

In the previous section, we optimized over aggregate variables $t_1, \ldots, t_m$, where $t_a = \sum_{h=1}^{b} x_{ha}$ represents the expected number of interdiction actions taken on arc a. For the case in which all c_a are convex, finding this optimal set of $\mathbf{t}$-values is NP-hard; however, once this is done, an optimal solution, $X^*(\mathbf{t}) \in \mathfrak{X}$, is easily generated by dividing t_a equally among the b rows. By contrast, when the c_a functions are all concave, the problem of finding a preferred vector $\mathbf{t}$ is polynomially solvable, whereas the problem of mapping a solution $\mathbf{t}$ to an optimal matrix $X \in \mathfrak{X}$ becomes NP-hard. We prove this result in Section 4.3.1, and then in Section 4.3.2, we provide a polynomial-time $(1 - {}^1\!/\mathrm{e})$-approximation algorithm.

4.3.1 Complexity Proof

Similar to Section 4.2.1 we begin by proving a column property of the ideal solution to the concave SPIP-RS.

Lemma 4.4. Consider a concave function c_a for some $a \in \mathcal{A}$, and consider an interdiction solution vector $\mathbf{x}_a$ for arc a that contains two fractional components h' and h'' such that $0 < x_{h'a} \leq x_{h''a} < 1$.

Define $\overline{\boldsymbol{\delta}} \in \mathbb{R}^b$ to be a vector of all zeros except for $\overline{\delta}_{h'} = -\epsilon$ and $\overline{\delta}_{h''} = \epsilon$, where $\epsilon = \min\{x_{h'a}, 1-x_{h''a}\}$. Then $\mathrm{E}\left(c_a(\tau)\,|\mathbf{x}_a + \overline{\boldsymbol{\delta}}\right) \geq \mathrm{E}\left(c_a(\tau)\,|\mathbf{x}_a\right)$.

Proof. As in the proof of Lemma 4.2, assume (by reindexing rows if necessary) that $h' = b-1$ and $h'' = b$. Then $\overline{\boldsymbol{\delta}} = -\boldsymbol{\delta}$ as defined in the proof of Lemma 4.2, and therefore:

$$\begin{aligned}&\mathrm{E}(c_a(\tau)|\mathbf{x}_a + \overline{\boldsymbol{\delta}}) - \mathrm{E}(c_a(\tau)|\mathbf{x}_a)\\ &\qquad = \left(\epsilon(x_{(b-1)a} - x_{ba}) - \epsilon^2\right)\sum_{\tau=0}^{b-2}\Pi^{b-2}(\mathbf{x}_a,\tau)\Big(c_a(\tau) - 2c_a(\tau+1) + c_a(\tau+2)\Big) \geq 0,\end{aligned}$$

where the derivation of this term mirrors that in Lemma 4.2 by negating the direction vector $\boldsymbol{\delta}$. The inequality now stems from the fact that $x_{(b-1)a} - x_{ba} < 0 < \epsilon$; hence, $\epsilon(x_{(b-1)a} - x_{ba}) - \epsilon^2 < 0$. Also, each term in the summation is non-positive since all $\Pi^{b-2}(\widehat{\mathbf{x}}_a, \tau) \geq 0$ and $c_a(\tau) - 2c_a(\tau+1) + c_a(\tau+2) \leq 0$ by the concavity of c_a. This completes the proof. □

Lemma 4.4 implies that given a vector of preferred column sums, $\mathbf{t}^*$, we would ideally select some $X(\mathbf{t}^*) \in \mathfrak{X}$ such that each column contains at most one fractional component. The challenge would be to allocate fractional components of the columns to rows in a way such that the sum of components in every row equals one (recalling the condition of $\mathfrak{X}$ that requires $\sum_{a\in\mathcal{A}} x_{ha} = 1,\ \forall h \in \mathcal{H}$). However, this allocation is not always feasible, and there may exist a uniquely optimal solution in which some columns have multiple fractional values. This observation motivates our proof of the following theorem.

Theorem 4.5. DSPIP-RS is NP-complete, even when c_a is concave for all $a \in \mathcal{A}$.

Proof. We prove the result with a reduction from 3 PARTITION (3PART), which is NP-complete in the strong sense [25], and is stated as follows.

3PART

Input: Given a set of positive integers, $S = \{s_1, \ldots, s_m\}$ where

- $m = 3q$ for some $q \in \mathbb{Z}_+$
- $\sum_{a=1}^{m} s_a = qB$ for some $B \in \mathbb{Z}_+$
- $B/4 < s_a < B/2$ for each $s_a \in S$

Question: Does there exist a partitioning of S into three-element sets, $S_1, \ldots, S_q$, such that the sum of elements in each set is B?

Given an 3PART instance we construct a reduction to DSPIP-RS as follows. Our directed graph $G = (\mathcal{N}, \mathcal{A})$ has two vertices, $\mathcal{N} = \{s, t\}$, and m parallel arcs, $\mathcal{A} = \{a_1, \ldots, a_m\}$. A similar transformation to a network having no parallel arcs is easy to obtain by splitting arcs; we use a multigraph here for ease of presentation. The leader's interdiction budget is q. Each arc a has the concave cost function,

$$c_a(t) = \begin{cases} 0 & t = 0 \\ B/s_a & t = 1, \ldots, b. \end{cases}$$

Finally, the threshold is $z^* = 1$. Observing that this reduction is polynomial time, we proceed to show that our 3PART instance has a solution if and only if the transformed DSPIP-RS instance has a solution.

Let $S_1, \ldots, S_q$ be a true certificate for 3PART. For each $h = 1, \ldots, q$, examine each $s_a \in S$ and set $x_{ha} = s_a/B$ if $s_a \in S_h$ and $x_{ha} = 0$ otherwise. Then for each $h = \mathcal{H}$, we have $\sum_{a \in \mathcal{A}} x_{ha} = \sum_{s_a \in S_h} s_a/B = B/B = 1$, so that $X \in \mathfrak{X}$. Further, since $S_1, \ldots, S_q$ is a partition, there is a unique $S_h \ni s_a$; hence, each column of X has a unique non-zero component $x_{ha} = s_a/B$. Thus $\mathrm{E}(c_a(\sigma)|\mathbf{x}_a) = (1 - {}^{s_a}\!/_{B})c_a(0) + ({}^{s_a}\!/_{B})c_a(1) = 1$ for each $a \in \mathcal{A}$, and $(X, [1, 0])$ is a true certificate for DSPIP-RS.

Conversely, assume that the DSPIP-RS instance has a true certificate, $(X, [1, 0])$. We first prove that $\sum_{h \in \mathcal{H}} x_{ha} = s_a/B$ for each $a \in \mathcal{A}$.

The $\geq$ direction: Assume by contradiction that $\sum_{h \in \mathcal{H}} x_{ha} < s_a/B$ for some arc $a \in \mathcal{A}$. If $\mathbf{x}_a$ has a single positive component, $x_{h^*a} > 0$, then from (4.3):

$$\mathrm{E}(c_a(\tau)|\mathbf{x}_a) = c_a(0)(1 - x_{h^*a}) + c_a(1)x_{h^*a} < 0(1 - {}^{s_a}\!/_{B}) + {}^{B}\!/_{s_a}({}^{s_a}\!/_{B}) = 1. \tag{4.8}$$

Alternatively, if $\mathbf{x}_a$ has at least two fractional components, then Lemma 4.4 bounds $\mathrm{E}(c_a(\tau_a)|\mathbf{x}_a)$ by the result in (4.8), which is less than 1. In either case $\mathrm{E}(c_a(\tau_a)|\mathbf{x}_a) < 1$, contradicting our certificate.

The $\leq$ direction: This follows directly from the three linear inequalities:

$$\sum_{h \in \mathcal{H}} \sum_{a \in \mathcal{A}} x_{ha} \leq \sum_{a=1}^{m} s_a/B; \qquad \sum_{h \in \mathcal{H}} x_{ha} \geq s_a/B; \qquad \text{and } x_{ha} \geq 0.$$

Next, noting that $\sum_{h\in\mathcal{H}} x_{ha} = {}^{s_a}/_{B}$ for all arcs $a \in \mathcal{A}$, we have that $\mathrm{E}(c_a(\tau_a)|\mathbf{x}_a) = 1$ only if each column $\mathbf{x}_a$ has exactly one positive component. Thus, setting $S_h = \{a \in \mathcal{A} : x_{ha} > 0\}$ for each $h \in \mathcal{H}$ yields a partition of S. Moreover, for each $h \in \mathcal{H}$,

$$X \in \mathfrak{X} \Longrightarrow \sum_{a\in\mathcal{A}:x_{ha}>0} s_a/B = 1 \Longrightarrow \sum_{s_a\in S_h} s_a = B.$$

Finally, $|S_h| = 3$ for each $h = 1, \ldots, b$ because ${}^{B}/_{4} < s_a < {}^{B}/_{2}$. Thus, $S_1, \ldots, S_b$ is indeed a true certificate for 3PART. Thus, the concave DSPIP-RS is NP-hard, and by Lemma 4.1 the DSPIP-RS is NP-complete. $\square$

4.3.2 Polynomial-time approximation algorithm for concave SPIP-RS

We present a polynomial-time approximation algorithm for the concave SPIP-RS. This algorithm first identifies preferred column sums $\mathbf{t}^* \geq \mathbf{0} : \sum_{i=1}^{m} t_i^* = b$, and then uses that vector to construct a heuristic solution $X^{\mathrm{H}}(\mathbf{t}) \in \mathfrak{X}$. We will show that our heuristic objective is within a factor of at least $1 - {}^{1}/_{\mathrm{e}} \approx 63.2\%$ of the optimal objective.

We begin our heuristic by finding a set of preferred column sums, $\mathbf{t}^*$, by solving the linear program:

$$\max \quad d_s \tag{4.9a}$$

$$\text{s.t.}\; d_i - d_j \leq (t_{ij} - \tau)(c_{ij}(\tau+1) - c_{ij}(\tau)) + c_{ij}(\tau) \qquad \forall (i,j) \in \mathcal{A},\; \tau \in \{0, \ldots, b-1\} \tag{4.9b}$$

$$d_t = 0 \tag{4.9c}$$

$$\sum_{a\in\mathcal{A}} t_a = b \tag{4.9d}$$

$$t_a \geq 0 \qquad \forall a \in \mathcal{A}. \tag{4.9e}$$

Observe that (4.9a) and (4.9c) are equivalent to (4.1a) and (4.1c), respectively, and the RHS of constraint (4.1b) is replaced in (4.9b) by the linear functions defining facets of the piecewise-linear function c_{ij}. From the definition of (discrete) concavity, it is straightforward to show that these functions correspond to upper bound values for $\mathrm{E}(c_{ij}(\tau)|\mathbf{x}_a)$. Furthermore, the column-wise interdiction policy defined by $\mathbf{t}$ subject to constraints (4.9d)–(4.9e) relaxes the row-sum constraints of $X \in \mathfrak{X}$ by aggregating them. Thus (4.9) is a relaxation of (4.1).

Given $\mathbf{t}^*$ optimal to (4.9), we construct a heuristic solution that allocates integer portions of each t_a^* to individual rows and then spreads the remaining fractional portions evenly across the remaining rows. Let $b' = \sum_{a=1}^{m} \lfloor t_a^* \rfloor$ and $b'' = b - b'$. The integer portion of the heuristic solution generated from $\mathbf{t}^*$ is given by the $b' \times m$ submatrix:

$$X^{\text{Int}}(\mathbf{t}^*) = \begin{array}{c} \\ \text{Interdiction } 1 \\ \vdots \\ \text{Interdiction } \lfloor t_1 \rfloor \\ \text{Interdiction } \lfloor t_1 \rfloor{+}1 \\ \vdots \\ \text{Interdiction } \lfloor t_1 \rfloor{+}\lfloor t_2 \rfloor \\ \vdots \\ \text{Interdiction } b' \end{array} \begin{array}{c} \begin{array}{cccc} \text{Arc } 1 & \text{Arc } 2 & \cdots & \text{Arc } m \end{array} \\ \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \end{array}.$$

Letting $\widehat{t_a} = t_a - \lfloor t_a \rfloor$ for each $a \in \mathcal{A}$, the full heuristic solution is given by:

$$X^{\text{H}}(\mathbf{t}^*) = \begin{array}{c} 1 \\ \vdots \\ b' \\ b'{+}1 \\ \vdots \\ b \end{array} \left[\begin{array}{ccc} & X^{\text{Int}}(\mathbf{t}^*) & \\ \hline \widehat{t_1^*}/b'' & \cdots & \widehat{t_m^*}/b'' \\ \vdots & \ddots & \vdots \\ \widehat{t_1^*}/b'' & \cdots & \widehat{t_m^*}/b'' \end{array} \right].$$

The complexity of computing $X^{\text{H}}(\mathbf{t}^*)$ is proportional to the complexity of solving the linear program (4.9), which is polynomial in the size of the SPIP-RS instance.

To establish the desired approximation result, we now define two more solutions:

$$X^{\mathrm{LB}}(\mathbf{t}^*) = \begin{array}{c} {\scriptstyle 1} \\ \vdots \\ {\scriptstyle b'} \\ \hline {\scriptstyle b'+1} \\ \vdots \\ {\scriptstyle b} \end{array} \left[\begin{array}{c} \\ X^{\mathrm{Int}}(\mathbf{t}^*) \\ \\ \hline \\ \mathbf{0} \\ \\ \end{array}\right], \quad \text{and } X^{\mathrm{UB}}(\mathbf{t}^*) = \begin{array}{c} {\scriptstyle 1} \\ \vdots \\ {\scriptstyle b'} \\ \hline {\scriptstyle b'+1} \\ \hline {\scriptstyle b'+2} \\ \vdots \\ {\scriptstyle b} \end{array} \left[\begin{array}{c} \\ X^{\mathrm{Int}}(\mathbf{t}^*) \\ \\ \hline \widehat{\boldsymbol{t}^*}^T \\ \hline \\ \mathbf{0} \\ \\ \end{array}\right].$$

Although neither $X^{\mathrm{LB}}(\mathbf{t})$ nor $X^{\mathrm{UB}}(\mathbf{t})$ necessarily belong to $\mathfrak{X}$, since they need not satisfy the row sum constraints, we use their corresponding objectives to bound the optimal objective. Defining $z(X)$ to be the objective function value of (4.1) given a fixed interdiction matrix X, we get the following objective bounds.

Proposition 4.2. Let X^* be an optimal solution to (4.1) for some graph G with concave cost functions, c_a $\forall a \in A$, and let $\mathbf{t}^*$ be an optimal solution to (4.9). Then $z(X^{\mathrm{LB}}(\mathbf{t}^*)) \leq z(X^{\mathrm{H}}(\mathbf{t}^*)) \leq z(X^*) \leq z(X^{\mathrm{UB}}(\mathbf{t}^*))$.

Proof. We have $z(X^{\mathrm{LB}}(\mathbf{t}^*)) \leq z(X^{\mathrm{H}}(\mathbf{t}^*))$ since $X^{\mathrm{LB}}(\mathbf{t}^*) \leq X^{\mathrm{H}}(\mathbf{t}^*)$ and all cost functions are non-decreasing. The next inequality $z(X^{\mathrm{H}}(\mathbf{t}^*)) \leq z(X^*)$ holds since $X^{\mathrm{H}}(\mathbf{t}^*) \in \mathfrak{X}$ and X^* is optimal. The final inequality is a result from Lemma 4.4 and the optimality of $\mathbf{t}^*$ for (4.9), which is a relaxation of (4.1). $\square$

We now state an approximation result that guarantees how well the heuristic solution performs relative to optimality. We measure the performance of $z(X^{\mathrm{H}}(\mathbf{t}^*)) - z(X^{\mathrm{LB}}(\mathbf{t}^*))$ relative to $z(X^*) - z(X^{\mathrm{LB}}(\mathbf{t}^*))$. Note that if $\mathbf{t}^* \in [0,1)^m$, then $X^{\mathrm{LB}}(\mathbf{t}^*) = [0]^{b\times m}$ (the zero matrix), and otherwise $X^{\mathrm{LB}}(\mathbf{t}^*)$ has at least one positive element. Thus, using $z(X^{\mathrm{LB}}(\mathbf{t}^*))$ as a baseline gives a result that is at least as strong as comparing $z(X^{\mathrm{H}}(\mathbf{t}^*)) - z([0])$ to $z(X^*) - z([0])$ (i.e., a baseline of the shortest-path objective with no interdiction action), which, in turn, is stronger than comparing $z(X^{\mathrm{H}}(\mathbf{t}^*))$ to $z(X^*)$ (i.e., a baseline of zero).

Theorem 4.6. $z(X^{\mathrm{H}}(\mathbf{t}^*)) - z(X^{\mathrm{LB}}(\mathbf{t}^*)) > (1 - {}^1\!/\mathrm{e})(z(X^*) - z(X^{\mathrm{LB}}(\mathbf{t}^*)))$, and this bound is tight.

Proof. If $\mathbf{t}^* \in \mathbb{Z}^m$, then $X^{\mathrm{LB}}(\mathbf{t}^*) = X^{\mathrm{UB}}(\mathbf{t}^*)$, $X^{\mathrm{H}}(\mathbf{t}^*)$ is optimal from Proposition 4.2, and the result follows. Assume then $\mathbf{t}^* \notin \mathbb{Z}^m$, and by implication, $b'' = b - b' \geq 1$. It suffices to prove that $\mathrm{E}(c_a(\tau)|\mathbf{x}_a^{\mathrm{H}}) - \mathrm{E}(c_a(\tau)|\mathbf{x}_a^{\mathrm{LB}}) > (1 - {}^1\!/\mathrm{e})(\mathrm{E}(c_a(\tau)|\mathbf{x}_a^{\mathrm{UB}}) - \mathrm{E}(c_a(\tau)|\mathbf{x}_a^{\mathrm{LB}}))$, $\forall a \in \mathcal{A}$, where $\mathbf{x}_a^{\mathrm{H}}$, $\mathbf{x}_a^{\mathrm{LB}}$, and $\mathbf{x}_a^{\mathrm{UB}}$ denote the a^{th} column of $X^{\mathrm{H}}(\mathbf{t}^*)$, $X^{\mathrm{LB}}(\mathbf{t}^*)$, and $X^{\mathrm{UB}}(\mathbf{t}^*)$, respectively. This result guarantees that $z(X^{\mathrm{H}}(\mathbf{t}^*)) - z(X^{\mathrm{LB}}(\mathbf{t}^*)) > (1 - {}^1\!/\mathrm{e})(z(X^{\mathrm{UB}}(\mathbf{t}^*)) - z(X^{\mathrm{LB}}(\mathbf{t}^*)))$, and the theorem follows by Proposition 4.2. Let $a \in \mathcal{A}$. We note that

$$\begin{aligned}
&\mathrm{E}(c_a(\tau)|\mathbf{x}_a^{\mathrm{LB}}) = c_a(\lfloor t_a^* \rfloor),\\
&\mathrm{E}(c_a(\tau)|\mathbf{x}_a^{\mathrm{UB}}) = c_a(\lfloor t_a^* \rfloor) + \widehat{t}_a(c_a(\lfloor t_a^* \rfloor + 1) - c_a(\lfloor t_a^* \rfloor)), \text{ and}\\
&\mathrm{E}(c_a(\tau)|\mathbf{x}_a^{\mathrm{H}}) = c_a(\lfloor t_a^* \rfloor) + \sum_{\tau=1}^{b''}(c(\tau + \lfloor t_a^* \rfloor) - c_a(\lfloor t_a^* \rfloor))\binom{b''}{\tau}\left({}^{\widehat{t}_a}\!/\!{}_{b''}\right)^{\tau}\left(1 - {}^{\widehat{t}_a}\!/\!{}_{b''}\right)^{b''-\tau}.
\end{aligned}$$

Using these values, we proceed with our algebraic reduction:

$$\begin{aligned}
\mathrm{E}(c_a(\tau)|\mathbf{x}_a^{\mathrm{H}}) - \mathrm{E}(c_a(\tau)|\mathbf{x}_a^{\mathrm{LB}}) &= \Bigg(\sum_{\tau=1}^{b''}(c(\tau + \lfloor t_a^* \rfloor) - c_a(\lfloor t_a^* \rfloor))\binom{b''}{\tau}\left({}^{\widehat{t}_a}\!/\!{}_{b''}\right)^{\tau}\left(1 - {}^{\widehat{t}_a}\!/\!{}_{b''}\right)^{b''-\tau}\\
&\qquad + c_a(\lfloor t_a^* \rfloor)\Bigg) - (c_a(\lfloor t_a^* \rfloor))\\
&\geq (c_a(1 + \lfloor t_a^* \rfloor) - c_a(\lfloor t_a^* \rfloor))\sum_{\tau=1}^{b''}\binom{b''}{\tau}\left(\frac{\widehat{t}_a}{b''}\right)^{\tau}\left(1 - \frac{\widehat{t}_a}{b''}\right)^{b''-\tau} && \text{(4.10a)}\\
&= (c_a(1 + \lfloor t_a^* \rfloor) - c_a(\lfloor t_a^* \rfloor))\left(1 - \left(1 - \frac{\widehat{t}_a}{b''}\right)^{b''}\right)\\
&> (c_a(1 + \lfloor t_a^* \rfloor) - c_a(\lfloor t_a^* \rfloor))\left(1 - \lim_{b''\to\infty}\left(1 - \frac{\widehat{t}_a}{b''}\right)^{b''}\right) && \text{(4.10b)}\\
&= (c_a(1 + \lfloor t_a^* \rfloor) - c_a(\lfloor t_a^* \rfloor))\left(1 - \frac{1}{\mathrm{e}^{\widehat{t}_a}}\right)\\
&\geq (c_a(1 + \lfloor t_a^* \rfloor) - c_a(\lfloor t_a^* \rfloor))\left(1 - \frac{1}{\mathrm{e}}\right)(\widehat{t}_a) && \text{(4.10c)}\\
&= (1 - {}^1\!/\mathrm{e})\Big(\widehat{t}_a\big(c_a(\lfloor t_a^* \rfloor + 1) - c_a(\lfloor t_a^* \rfloor)\big) + c_a(\lfloor t_a^* \rfloor) - c_a(\lfloor t_a^* \rfloor)\Big)\\
&= (1 - {}^1\!/\mathrm{e})\Big(\mathrm{E}\big(c_a(\tau)|\mathbf{x}_a^{\mathrm{UB}}\big) - \mathrm{E}\big(c_a(\tau)|\mathbf{x}_a^{\mathrm{LB}}\big)\Big).
\end{aligned}$$

The inequality in (4.10a) holds by noting that in the summation, $c_a(\tau + \lfloor t_a^* \rfloor) \geq c_a(1 + \lfloor t_a^* \rfloor)$ since c_a is non-decreasing. The inequality in (4.10b) holds because $(1 - \widehat{t}_a/b'')^{b''}$ is increasing in b''. (This fact and the inequality in (4.10c) are both proven using calculus arguments that we omit for brevity.) The inequality in (4.10a) becomes tight when $c_a(1+\lfloor t_a^*\rfloor) = c_a(\tau + \lfloor t_a^*\rfloor)$ for $\tau > 1$; inequality (4.10b) holds in the limit as b'' becomes arbitrarily large; and inequality (4.10c) becomes tight either when $\widehat{t}_a = 0$, or in the limit as $\widehat{t}_a$ tends towards 1. Therefore, the ratio $1 - {}^1\!/_{\mathrm{e}}$ is tight. $\square$

4.4 Sample Average Approximation for General SPIP-RS

In this section we briefly describe a sample average approximation (SAA) based formulation to find near-optimal solutions to the SPIP-RS. The method solves stochastic programs of the form $\min \mathrm{E}_{\Omega} g(\mathbf{x}, \omega)$, where the random variable $\omega \in \Omega$ is independent of $\mathbf{x}$. SAA uses Monte Carlo simulation to draw an independent and identically distributed set of scenarios, $\widehat{\Omega} \subset \Omega$. The sampling must be independent of the decision variables. The method then approximates an optimal solution by minimizing the sample average: $\sum_{\omega \in \widehat{\Omega}} g(\mathbf{x}, \omega)/|\widehat{\Omega}|$. Shapiro et al. [75] show that the standard error for the SAA solution is given by $\mathcal{O}(|\widehat{\Omega}|^{-1/2})$.

The SPIP-RS exhibits decision-dependent uncertainty (see [26, 89]), since the random variables represent interdiction locations, which depend on the decision variables X. To provide an independent scenario generation, therefore, we will generate scenario vectors using a uniform 0–1 distribution, $\boldsymbol{\omega}_s \in (0,1)^b$ for $s \in \mathcal{S}$, where $\mathcal{S}$ is an indexing set of scenarios. Then for each interdiction $h \in \mathcal{H}$ in each scenario $s \in \mathcal{S}$, we will assign interdiction h to arc a if and only if $\sum_{a'=1}^{a-1} x_{ha'} \leq \omega_{hs} < \sum_{a'=1}^{a} x_{ha'}$. To accommodate the desired strict inequality in our optimization problems, we define $\Psi_h = \cup_{s \in \mathcal{S}} \{\omega_{hs}\} \cup \{0, 1\}$ for all $h \in \mathcal{H}$, and we set $2\epsilon_h$ to be the smallest difference between any pair of values in Ψ_h, for each $h \in \mathcal{H}$. We ensure in our random vector generation process that each value in Ψ_h is distinct (by resampling if necessary).

To formulate our SAA-based formulation as a mixed-integer linear program, we introduce (with a slight abuse of notation) auxiliary binary variables χ_{has} that equal 1 if and only if interdiction $h \in \mathcal{H}$ is applied to arc $a \in \mathcal{A}$ in scenario $s \in \mathcal{S}$, and binary auxiliary variables $I_{\tau as}$ that equal 1 if and only if exactly $\tau \in \mathcal{H} \cup \{0\}$ interdictions occur on arc $a \in \mathcal{A}$ in scenario $s \in \mathcal{S}$. Our SAA mixed-integer linear program is:

$$\max_{X \in \mathfrak{X}} \quad d_s \tag{4.11a}$$

$$\text{s.t. } \chi_{has} \le 1 + \sum_{a'=1}^{a} x_{ha'} - \omega_{hs} - \epsilon_h \qquad \forall (h,a,s) \in \mathcal{H} \times \mathcal{A} \times \mathcal{S} \tag{4.11b}$$

$$\chi_{has} \le \omega_{hs} + \sum_{a'=a}^{m} x_{ha'} \qquad \forall (h,a,s) \in \mathcal{H} \times \mathcal{A} \times \mathcal{S} \tag{4.11c}$$

$$\sum_{\tau \in \{0\} \cup \mathcal{H}} I_{\tau as} = 1 \qquad \forall (a,s) \in \mathcal{A} \times \mathcal{S} \tag{4.11d}$$

$$\sum_{h \in \mathcal{H}} \chi_{has} = \sum_{\tau \in \{0\} \cup \mathcal{H}} \tau I_{\tau as} \qquad \forall (a,s) \in \mathcal{A} \times \mathcal{S} \tag{4.11e}$$

$$d_i - d_j \le \frac{1}{|\mathcal{S}|} \left(\sum_{s \in \mathcal{S}} \sum_{\tau \in \{0\} \cup \mathcal{H}} c_a(\tau) I_{\tau as} \right) \qquad \forall a = (i,j) \in \mathcal{A} \tag{4.11f}$$

$$d_t = 0 \tag{4.11g}$$

$$\chi_{has} \in \{0,1\} \qquad \forall (h,a,s) \in \mathcal{H} \times \mathcal{A} \times \mathcal{S} \tag{4.11h}$$

$$I_{\tau as} \in \{0,1\} \qquad \forall (\tau,a,s) \in \{0\} \cup \mathcal{H} \times \mathcal{A} \times \mathcal{S}. \tag{4.11i}$$

Objective (4.11a) is the same as (4.1a). Constraints (4.11b) and (4.11c) define the χ-variables so that $\chi_{has} = 1$ only if $\sum_{a'=1}^{a-1} x_{ha'} \le \omega_{hs} < \sum_{a'=1}^{a} x_{ha'}$. Note that the maximization objective induces $\chi_{has} = 1$ whenever $\sum_{a'=1}^{a-1} x_{ha'} \le \omega_{hs} < \sum_{a'=1}^{a} x_{ha'}$. Constraints (4.11d) and (4.11e) force $I_{\tau as}$ to their desired values. Constraints (4.11f) correspond to (4.1b), replacing the true expected arc cost with the sample average arc cost on the RHS, and constraint (4.11g) is equivalent to (4.1c). Finally, constraints (4.11h)–(4.11i) state integrality constraints on the χ- and I-variables. However, with the following lemma we can relax the integrality constraints on the I-variables corresponding to any discrete-concave function c_a.

Lemma 4.7. If c_a is discrete-concave, then integrality constraints (4.11i) can equivalently be relaxed to $I_{\tau as} \ge 0$, $\forall \tau \in \mathcal{H} \cup \{0\},\ s \in \mathcal{S}$.

Proof. Let $a = (i,j) \in \mathcal{A}$ with c_a discrete-concave. Examine a variant of formulation (4.11) in which constraints (4.11i) are relaxed to $I_{\tau as} \ge 0$, $\forall \tau \in \mathcal{H} \cup \{0\},\ s \in \mathcal{S}$. We show that there exists an optimal solution whose I-variables, I', are integer.

To begin, suppose an optimal solution to this relaxed formulation has I-variable values given by I^*. If I^* is integer, then we are done. Otherwise, there exists some $s \in \mathcal{S}$ so that the vector

$[I^*_{0as}, I^*_{1as}, \ldots, I^*_{bas}] \notin \mathbb{Z}^{b+1}$. Observe from (4.11e) that $\sum_{\tau=0}^{b} \tau I^*_{\tau as} \in \{0, \ldots, b\} \subset \mathbb{Z}$. So setting

$$I'_{\tau as} = \begin{cases} 1 & \text{if } \tau = \sum_{\tau'=0}^{b} \tau' I^*_{\tau' as} \\ 0 & \text{otherwise,} \end{cases} \qquad \forall \tau \in \mathcal{H} \cup \{0\},$$

we have $[I'_{0as}, I'_{1as}, \ldots, I'_{bas}] \in \mathbb{Z}^{b+1}$. Repeating this for each $s \in \mathcal{S}$ we obtain I' integer-valued. To prove this solution is optimal, note that (4.11d) and (4.11f) bound $d_i - d_j$ above by summing convex combinations of $\{c_a(0), \ldots, c_a(b)\}$ with respective multipliers $I_{0as}, \ldots, I_{bas}$. The result then follows by the discrete-concavity of c_a: $\sum_{\tau=0}^{b} c_a(\tau) I'_{\tau as} = c_a(\sum_{\tau=0}^{b} \tau I'_{\tau as}) \geq \sum_{\tau=0}^{b} c_a(\tau) I^*_{\tau as}$. $\square$

4.5 Computational Study

Our computational study focuses on the following questions. Each question is addressed in a separate subsection.

Question 1: What strategy is most efficient in solving general SPIP-RS instances of varying sizes?

Question 2: How well does the branch-and-bound algorithm described in Section 4.2.2 perform on convex instances?

Question 3: How well does the approximation heuristic of Section 4.3.2 perform on concave instances?

For this study, we coded each algorithm in Python v. 2.7. All runs were performed on Clemson University's Palmetto Cluster; each instance was run on a separate Intel Xeon E5-2670 (2.50GHz) core. Collectively, the cores shared 60Gb of memory. For each run, we recorded computational time, termination status, and solution quality information. The running time for each algorithm was capped at two hours (wall-time), at which point the algorithm was terminated and partial results were collected.

Our test bed consisted of 75 randomly generated instances: 15 each having 10, 20, 30, 50, and 80 nodes. For all instances, we used an interdiction budget of $b = 4$. For each instance, we used igraph's [16] Erdős-Renyí module [22] to randomly generate arcs with 30% arc density (i.e., 27, 114, 261, 735, and 1896 arcs, respectively). The 15 instances for each graph size were divided into five convex instances (`Cvx`), five concave instances (`Ccv`), and five instances having general (i.e., unspecified) cost functions (`Gen`). We randomly generated convex and concave arc cost functions using $b + 1$ random integers in $\{0, \dots, 10b\}$, sorted in increasing (for convex) or decreasing (for concave) order. Denoting the sorted integers for arc a by $\{\delta_0, \delta_1, \dots, \delta_b\}$, we assigned

$$c_a(\tau) = \begin{cases} \delta_0, & \tau = 0 \\ c_a(\tau - 1) + \delta_\tau, & \tau \in \mathcal{H}. \end{cases}$$

For the instances with general cost functions, we simply sorted the random numbers in increasing order and assigned the function values $c_a(\tau) = \delta_\tau$, $\forall \tau \in \mathcal{H} \cup \{0\}$.

For baseline comparison, we used Artelys Knitro v. 11.1 [13] as a general-purpose non-linear solver (using the software's Python API). Knitro primarily uses ascent-based methods, hence its results are not guaranteed to be optimal. We allowed Knitro to automatically determine both gradient and Hessian information.

4.5.1 SAA and Knitro comparison

First we examine the performance of the SAA algorithm of Section 4.4. We compared Gurobi v. 7.5 solving (4.11) and Knitro solving (4.1). We used $|\mathcal{S}| = 50$ scenarios.

Initially, we compared these algorithms on the 10-node instances. Despite the small instance sizes, the SAA formulation could only be solved to optimality in three of the 15 instances within the time limit (see Table 4.5). The presence of $\mathcal{O}(bm|\mathcal{S}|)$ binary variables appears to make formulation (4.11) exceedingly difficult to solve. In contrast, Knitro was able to meet convergence criteria on all 10-node instances within 20 seconds. However, without solution quality information from the SAA model we could not assess the quality of solutions provided by Knitro.

To obtain better results, therefore, we implemented a lazy approach to constructing our formulations, motivated by similar strategies for the SPIP [36, 50]. This approach optimizes SPIP-RS instances over a subgraph of the original graph, expanding the subgraph only as needed (see Appendix A.3.2 for details). We repeated the previous computational experiment and report the results in Table 4.5. The first column lists the instance name. The next two columns report the objective of the best solution obtained by the end of each original algorithm at the time it terminated, while the fourth and fifth columns provide the computational times required by those two algorithms. The sixth, seventh, ninth, and tenth columns provide analogous results for the lazy approach. The eighth column

Table 4.5: Comparison of SAA (with $|\widehat{\Omega}| = 50$ scenarios) and Knitro on SPIP-RS instances with 10 nodes and 27 arcs.

Instance	Original Algorithms				"Lazy" Approach				
	Objective		Time (sec)		Objective		z_{UB}	Time (sec)	
	SAA	*Knitro*	*SAA*	*Knitro*	*SAA*	*Knitro*		*SAA*	*Knitro*
Cvx_$n10$_$i1$	87.74	94.00	> 7200	8.26	6.00	94.00	140.14	> 7200	0.24
Cvx_$n10$_$i2$	35.69	40.72	> 7200	9.91	13.00	40.72	57.38	> 7200	0.82
Cvx_$n10$_$i3$	74.77	76.00	3501	13.29	74.04	76.00	76.56	0.56	0.60
Cvx_$n10$_$i4$	38.59	45.81	> 7200	11.64	8.00	45.81	126.86	> 7200	1.11
Cvx_$n10$_$i5$	138.14	143.00	> 7200	7.92	142.49	143.00	144.68	63.92	0.18
Ccv_$n10$_$i1$	75.00	75.00	0.51	13.58	75.00	75.00	75.00	0.03	0.18
Ccv_$n10$_$i2$	77.92	79.00	0.35	11.03	79.00	79.00	79.00	0.03	0.15
Ccv_$n10$_$i3$	90.22	97.00	> 7200	15.39	69.00	97.00	129.62	> 7200	2.31
Ccv_$n10$_$i4$	122.51	130.75	> 7200	19.69	103.00	130.74	143.76	> 7200	0.47
Ccv_$n10$_$i5$	128.77	134.04	> 7200	12.77	113.19	134.04	151.11	> 7200	2.32
Gen_$n10$_$i1$	27.96	28.91	> 7200	11.85	10.00	28.42	33.00	> 7200	2.56
Gen_$n10$_$i2$	15.12	16.66	> 7200	6.11	4.00	16.66	27.24	> 7200	0.76
Gen_$n10$_$i3$	21.16	23.60	> 7200	6.20	17.00	23.60	25.02	> 7200	0.34
Gen_$n10$_$i4$	31.49	37.87	> 7200	10.12	14.00	37.89	76.94	> 7200	6.18
Gen_$n10$_$i5$	30.57	33.23	> 7200	8.00	23.00	32.24	38.76	> 7200	0.42

reports values for z_{UB}, which is the best (i.e., minimal) upper bound on the objective, as reported by Gurobi, among all of the lazy iterations. In two instances, Cvx_$n10$_$i1$ and Gen_$n10$_$i4$, Gurobi failed to converge on even the first lazy iteration, which implies that the z_{UB} values corresponding to those instances may be excessively large. With the lazy approach, SAA was able to solve one additional instance within two hours, but still failed to meet the convergence criteria for the remaining 11 instances. In contrast, Knitro returned a result in less than two seconds in most cases when using the lazy approach. Furthermore, since Gurobi was able to converge in the first few iterations of the lazy approach on most instances, that allowed us to obtain valid upper bounds and provides additional confidence that Knitro's solutions are near-optimal.

Based on the results from our smallest instances, we only used the Knitro solver for the remaining 60 instances, and we examined its performance with and without the lazy approach on all 75 instances. Our study confirms that with respect to objective quality, the lazy approach and non-lazy approaches both yielded approximately equal objectives (see Figure 4.3). The notable exception was that the lazy approach appeared to become trapped in locally optimal solutions for our 80-node instances with general cost functions, since it met convergence criteria with an objective that was significantly worse than the non-lazy variant's objective. These are the points above the diagonal in Figure 4.3. With regard to computational time (see Figure 4.4), the lazy approach consistently

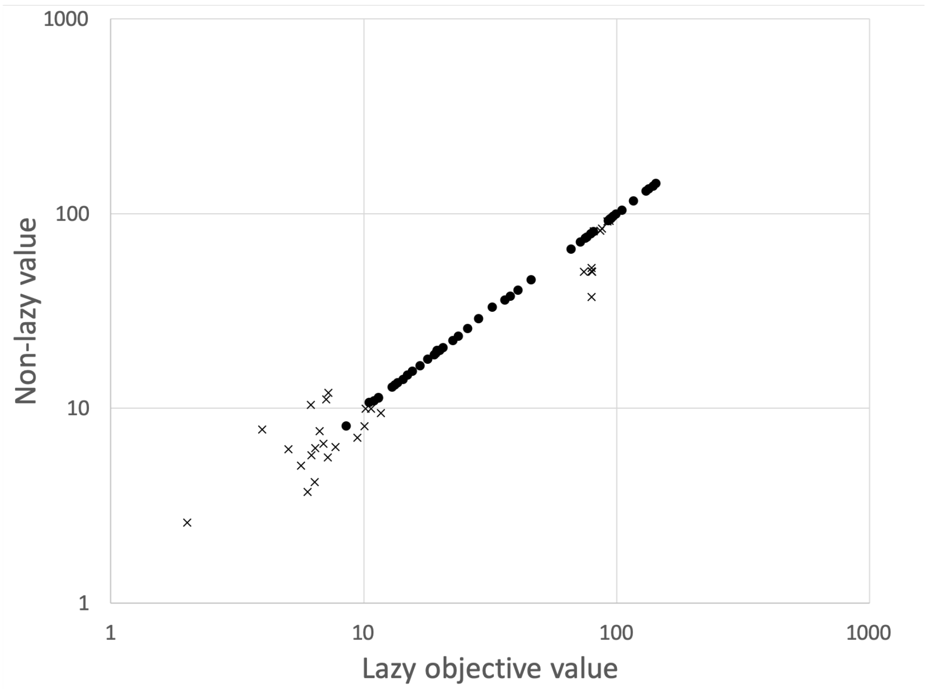


Figure 4.3: Effect of the lazy approach on the objective. "X" markers indicate non-lazy approach timed-out before converging.

Table 4.6: Summary statistics for Knitro with the lazy approach.

# of	Running time		
nodes	Min	Mean	Max
10	0.01	1.05	5.06
20	0.62	21.81	64.70
30	2.79	143.71	473.60
50	8.48	1061.18	2587.63
80	54.14	3237.02	> 7200

terminated in a similar timeframe or faster than the non-lazy approach. For the larger instances with 50 and 80 nodes the non-lazy variant timed out on all 30 instances, whereas the lazy approach successfully converged for all 50-node instances within two hours, and 12 of 15 instances having 80 nodes (for ease of display, the results from the 80-node instances are not depicted in Figure 4.4). Finally, in Figure 4.5 and Table 4.6, we display the lazy variant's running time on all 75 instances and provide summary statistics, respectively. The algorithm exhibits exponential growth in solver times, as expected, with a slight tapering effect due to the fact that the lazy approach ignores a significant number of arcs in the network.

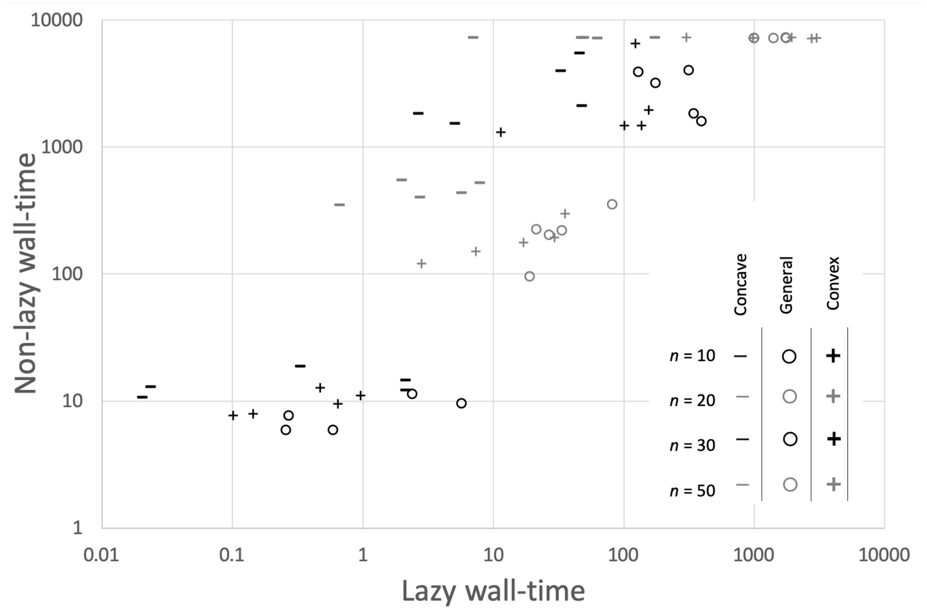


Figure 4.4: Effect of the lazy approach on running time. Instances with $n = 80$ nodes are excluded.

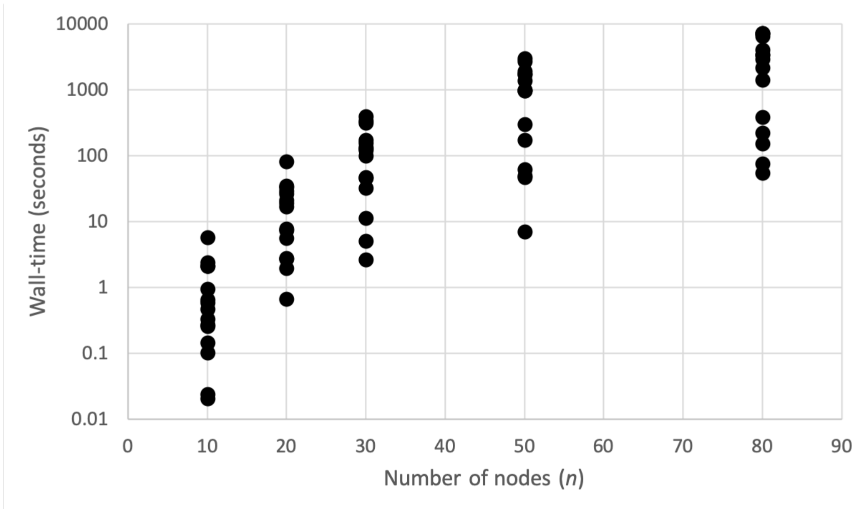


Figure 4.5: Running time (in seconds) for Knitro (using the lazy approach) by number of graph nodes (n).

4.5.2 Branch-and-Bound Performance on Convex Problems

We examined the performance of the branch-and-bound algorithm from Section 4.2.2 on our convex test bed instances, comparing the computational time and solution quality to that provided by Knitro. We examined the performance of the branch-and-bound algorithm both with and without the lazy approach described in Section 4.5.1. For our optimality tolerance we used $\epsilon = 0.01$. The results are shown in Table 4.7. In the columns showing the number of nodes in the branch-and-bound tree, for the lazy approach we include both the total number of nodes over all iterations and the number of nodes in the iteration having the largest tree.

Both the lazy and non-lazy variants were able to solve all of the 10- and 20-node instances to the specified optimality tolerance within two hours. In most of these instances, the lazy approach was faster: Usually it took on the order of half the time as the non-lazy approach. The notable exception is the `Convex`_$n20$_$i4$ instance, where the lazy approach took about 20% longer than the non-lazy approach. This occurred because the subgraphs in the last seven lazy iterations each had only one arc more than the previous iteration's subgraph; thus, these iterations each took similar (significant) effort to solve. Both the lazy and non-lazy approaches solved three of the five 30-node instances in under two hours, and with one exception, both failed to finish in two hours on all 50- and 80-node instances. The exception was instance `Cvx`_$n50$_$i4$, where the lazy approach finished but the non-lazy timed-out. In every case where both branch-and-bound variants timed out, the non-lazy approach identified a better incumbent solution than the lazy variant.

In most instances, the quality of the solutions from Knitro and the branch-and-bound algorithms are identical. This again validates the strength of Knitro's ascent approach, since it found an optimal solution much faster than the exact method. However, the strength of the branch-and-bound algorithm is seen in the `Convex`_$n20$_$i3$ instance, where the exact branch-and-bound algorithm provides a significantly better solution. Thus, for small- and medium-sized instances, which are complex enough such that Knitro occasionally returns a significantly suboptimal solution, the branch-and-bound solver can tractably find a true optimal solution.

4.5.3 Quality of the Approximation Heuristic on Concave Problems

The polynomial-time heuristic algorithm described in Section 4.3.2 terminated on all instances in our test set in less than one second. Recall from Section 4.3.2 that the heuristic must provide

Table 4.7: Branch-and-bound study results. "NL" indicates the non-lazy approach, in contrast to the lazy one described in Section 4.5.1.

Instance	Running Time			Objective Value			Spatial B-&-B Tree Size		
	Knitro	B-&-B		Knitro	B-&-B		NL	Lazy	
	Lazy	NL	Lazy	Lazy	NL	Lazy		Total	Largest
Cvx_n10_i1	0.14	0.08	0.02	94.00	94.00	94.00	4	7	4
Cvx_n10_i2	0.64	0.21	0.08	41.72	41.72	41.72	19	26	19
Cvx_n10_i3	0.47	0.05	< 0.005	76.00	76.00	76.00	1	1	1
Cvx_n10_i4	0.96	0.31	0.14	45.81	45.81	45.81	31	39	28
Cvx_n10_i5	0.10	5.05	< 0.005	143.00	143.00	143.00	1	1	1
Cvx_n20_i1	29.28	33.68	14.54	20.54	20.53	20.53	1039	2357	1004
Cvx_n20_i2	35.41	33.05	8.24	25.76	25.76	25.76	948	1462	822
Cvx_n20_i3	2.80	5.64	0.29	26.01	55.84	55.84	22	63	27
Cvx_n20_i4	7.32	984.02	1173.18	11.39	13.37	13.37	38311	143019	37522
Cvx_n20_i5	17.06	538.42	328.57	10.94	10.95	10.95	16824	41688	13960
Cvx_n30_i1	155.55	> 7200	> 7200	12.90	12.58	8.99	75419	182820	145952
Cvx_n30_i2	100.88	> 7200	> 7200	13.59	13.34	9.00	74352	264589	114923
Cvx_n30_i3	122.81	505.48	209.49	17.89	17.89	17.89	7468	24418	8443
Cvx_n30_i4	136.31	1341.32	1273.20	13.23	13.53	13.53	18136	108242	36341
Cvx_n30_i5	11.44	100.05	61.94	14.86	14.86	14.86	1524	8945	3115
Cvx_n50_i1	2758.40	> 7200	> 7200	6.22	5.77	2.00	31443	255469	81608
Cvx_n50_i2	973.36	> 7200	> 7200	10.06	10.04	8.00	31178	189538	109524
Cvx_n50_i3	3022.38	> 7200	> 7200	7.73	7.68	5.00	31535	210228	107418
Cvx_n50_i4	1930.19	> 7200	> 7200	6.95	6.48	2.75	30998	210189	145877
Cvx_n50_i5	301.10	> 7200	2117.56	10.69	10.68	10.69	31664	130567	43234
Cvx_n80_i1	4093.03	> 7200	> 7200	7.21	7.18	4.00	11363	219805	82761
Cvx_n80_i2	3411.52	> 7200	> 7200	6.40	6.11	2.00	11339	158842	103030
Cvx_n80_i3	6529.53	> 7200	> 7200	5.04	5.70	3.00	11416	192366	89935
Cvx_n80_i4	> 7200	> 7200	> 7200	2.00	4.79	2.00	11331	282376	72163
Cvx_n80_i5	> 7200	> 7200	> 7200	6.00	6.41	3.00	11204	231004	89807

an objective within $1 - {}^{1}/_{e} \approx 63.2\%$ of the true optimum. We compare in Table 4.8 the quality of the solutions from the approximation algorithm to those obtained by Knitro on each of the concave instances in our test bed. For every instance, the approximation heuristic is within approximately 95% of the Knitro's result, and in one instance (Ccv_$n20$_$i1$) the approximation heuristic found a better solution than Knitro.

Table 4.8: Comparing objectives from Knitro and the approximation algorithm of Section 4.3.2 on convex instances. The percentage column is given by dividing the approximation objective by the Knitro objective.

Instance	Objective		Percentage
	Knitro	*Approx*	
Ccv_$n10$_$i1$	75.00	75.00	100.0%
Ccv_$n10$_$i2$	79.00	79.00	100.0%
Ccv_$n10$_$i3$	97.00	92.04	94.9%
Ccv_$n10$_$i4$	130.74	130.74	100.0%
Ccv_$n10$_$i5$	134.04	134.04	100.0%
Ccv_$n20$_$i1$	80.79	80.96	100.2%
Ccv_$n20$_$i2$	139.00	136.52	98.2%
Ccv_$n20$_$i3$	99.44	96.82	97.5%
Ccv_$n20$_$i4$	95.27	92.82	97.4%
Ccv_$n20$_$i5$	66.10	66.10	100.0%
Ccv_$n30$_$i1$	116.32	115.04	98.9%
Ccv_$n30$_$i2$	92.44	90.17	97.5%
Ccv_$n30$_$i3$	104.71	104.32	99.6%
Ccv_$n30$_$i4$	71.66	71.66	100.0%
Ccv_$n30$_$i5$	94.27	90.85	96.4%
Ccv_$n50$_$i1$	94.12	89.78	95.4%
Ccv_$n50$_$i2$	86.25	85.83	99.5%
Ccv_$n50$_$i3$	81.24	79.89	98.3%
Ccv_$n50$_$i4$	87.55	84.19	96.2%
Ccv_$n50$_$i5$	91.97	90.18	98.1%
Ccv_$n80$_$i1$	79.61	76.43	96.0%
Ccv_$n80$_$i1$	80.23	76.34	95.1%
Ccv_$n80$_$i1$	79.62	77.78	97.7%
Ccv_$n80$_$i1$	74.44	73.72	99.0%
Ccv_$n80$_$i1$	79.61	78.11	98.1%

Given the quality of these solutions, we also examined whether warm-starting Knitro with an initial solution from the approximation algorithm improves the computational performance of Knitro. However, preliminary computational experiments showed no benefit to providing this warm start.

4.6 Conclusion and Future Work

In this chapter we examine the SPIP-RS in which multiple interdiction actions impact arc cost functions in a potentially non-linear fashion. When these cost functions are all linear, classical analysis reduces this problem to a linear program. When these cost functions are non-linear, the problem can be formulated as a continuous non-convex optimization problem. However, the SPIP-RS becomes NP-hard in the strong sense in this case, even if the cost functions are all concave, or if they are all convex. We then provide algorithms to solve special cases of the SPIP-RS. For the convex SPIP-RS, we prescribe a spatial branch-and-bound algorithm that returns a solution that is within a specified absolute optimality tolerance. For the concave SPIP-RS, we provide a polynomial-time $(1 - 1/e)$-approximation algorithm, whose empirical performance indicates that the optimality gap is far tighter than the worst-case gap. Finally, we present an SAA-based linear mixed-integer programming model that can be solved to provide an approximate solution to the SPIP-RS. However, computational results show that simply employing a non-linear optimization solver on the original non-linear model for the SPIP-RS tends to provide better results with far less computational effort than solving the SAA-based formulation using a mixed-integer programming solver.

Further research on the SPIP-RS might characterize problem complexity and approximability results under special cases of the problem. These cases may regard assumptions on the number of non-linear arc cost functions, conditions regarding the shape of these functions, or restrictions on the topology of the network itself. Additionally, future research questions may consider variants of the SPIP-RS itself. For instance, one may examine the case in which the follower uses a robust approach to selecting a preferred path, possibly using value-at-risk computations, instead of the shortest expected-cost. Yet another variation of this problem can extend this two-stage game into a three-stage game, where the agent selecting a path can fortify arcs before the interdicting agent can begin the two-stage SPIP-RS problem studied here.

Appendices

Appendix A Algorithms

A.1 MAWT Algorithms

In this appendix we present pseudo-code for the BOX and KS implementations used in our computational study described in Section 2.5. For our implementation, the functionality for each algorithm is separated among three modules, using an object-oriented framework common to both algorithms. Such a separation was not apparent for OBM, since that algorithm uses a recursive structure, as opposed to the looping structure of BOX and KS. Figure A.1 provides a graphical summary of Algorithm 1's search process and the data flows among the modules (initialization processes are not depicted). Module A.1 is the controller, which directs the iterative search in Z and stores non-dominated solutions as they are discovered. The list of search regions is maintained by Module A.2, and the searches are performed by Module A.3. Algorithm 2 uses a similar division of functions.

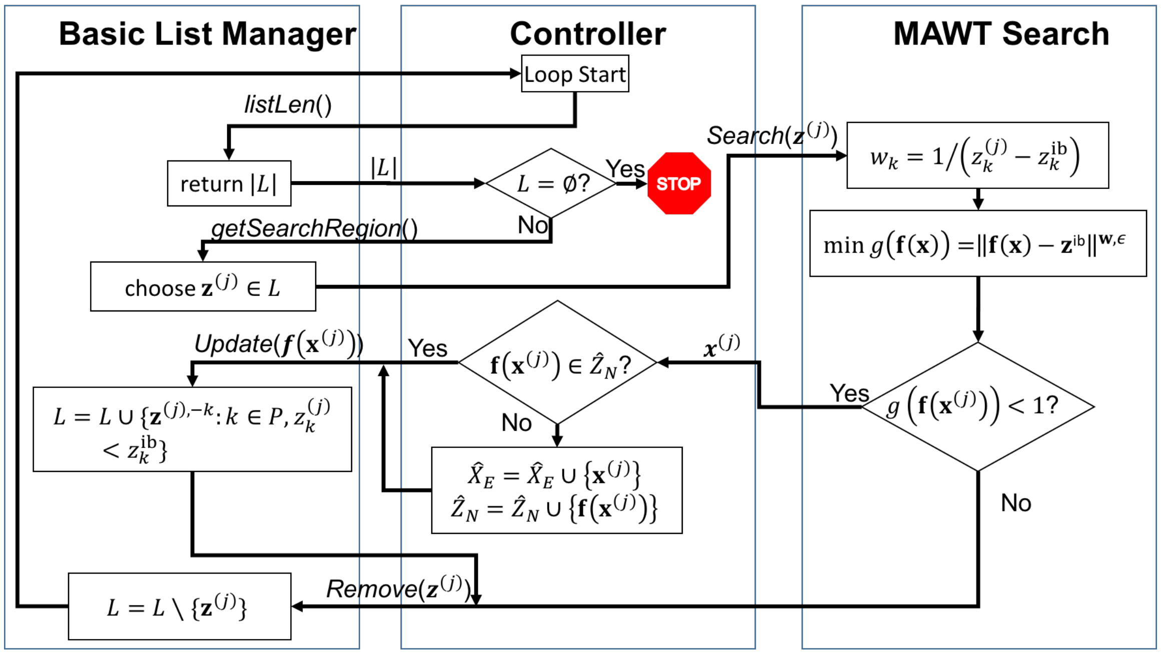

Basic List Manager
Controller
MAWT Search
Loop Start
listLen()
return $|L|$
$|L|$
$L = \emptyset$?
Yes
STOP
No
getSearchRegion()
choose $\mathbf{z}^{(j)} \in L$
Search($\mathbf{z}^{(j)}$)
$w_k = 1/\left(z_k^{(j)} - z_k^{\text{ib}}\right)$
$\min g(\mathbf{f}(\mathbf{x})) = \|\mathbf{f}(\mathbf{x}) - \mathbf{z}^{\text{ib}}\|^{\mathbf{w},\epsilon}$
Update($\boldsymbol{f}(\mathbf{x}^{(j)})$)
Yes
$\mathbf{f}(\mathbf{x}^{(j)}) \in \hat{Z}_N$?
$\boldsymbol{x}^{(j)}$
Yes
$g\left(\mathbf{f}(\mathbf{x}^{(j)})\right) < 1$?
$L = L \cup \{\mathbf{z}^{(j),-k} : k \in P, z_k^{(j)} < z_k^{\text{ib}}\}$
No
$\hat{X}_E = \hat{X}_E \cup \{\mathbf{x}^{(j)}\}$
$\hat{Z}_N = \hat{Z}_N \cup \{\mathbf{f}(\mathbf{x}^{(j)})\}$
No
Remove($\mathbf{z}^{(j)}$)
$L = L \setminus \{\mathbf{z}^{(j)}\}$

Figure A.1: Algorithm 2.1 search functions separated into three modules.

The modular design depicted in Figure A.1 provides several benefits. First, it allows the algorithm to be easily reconfigured by varying the combination of search and list management routines. For example, we used the modular design to implement Dächert et al.'s neighbor routine as our list manager for the tests in Section 2.5.3. Depending on search and list management routines, the resulting algorithm could also implement other ϵ-constraint methods, Tchebychev search methods, and dominated region exclusion methods. Secondly, the modular design allows us to easily examine the effects of the search algorithm separate from the management of L, and control the influence of exogenous variables, such as timing commands by using the common controller. In our study, the timers were managed by the control module. The controller started a timer, then called a command from either the list manager or search routine, and stopped the timer when the result was returned. By using a common controller module, we limit the variance due to the controller's coding and we can easily examine how the algorithms perform updating and searching routines separately.

In Algorithm 1 we give pseudocode in an object-oriented framework that is equivalent to Algorithm 2.1. The objects are initialized by calling the Controller module's *init* procedure. Inputs to the *init* function are the same as the inputs to Algorithm 2.1. In turn, the Controller defines the list manager and search objects by calling their respective *init* procedures. In this pseudocode we assume the initialize procedures are class methods which return the created object. The main algorithm is then run by calling the Controller's *FindNonDom* procedure.

Algorithm 2 presents pseudocode forKS, our object-oriented implementation of Kirlik and Sayin's ϵ-constraint algorithm [41]. It uses essentially the same controller module as Algorithm 1. Modules A.5 and A.6 function as the search and list manager modules, respectively.

Module A.1: *Controller*

```
1  Procedure init($\mathcal{I}, s, \mathbf{z}^{\text{ub}}$):
2      for $k \in P$ do
3          $z_k^{\text{lb}} \leftarrow \min\{z_k(\mathbf{x}) : x \in X\}$
4      ListMgr $\leftarrow$
           BasicListMgr.init($\mathbf{z}^{\text{ub}}$, $\mathbf{z}^{\text{lb}}$)
5      Search $\leftarrow$
           MAWTSrch.init($\mathbf{z}^{\text{ub}}, \mathbf{z}^{\text{lb}}, s$)
6      $\widehat{X}_E \leftarrow \emptyset$; $\widehat{Z}_N \leftarrow \emptyset$; $j \leftarrow 0$
7  Procedure FindNonDom():
8      while ListMgr.listLen() $> 0$ do
9          $j \leftarrow j + 1$
10         $\mathbf{z}^{(j)} \leftarrow$ListMgr.getSearchRegion()

11         $\mathbf{x}^{(j)} \leftarrow$Search.search($\mathbf{z}^{(j)}$)
12         if $\mathbf{x}^{(j)} =$ Null then
13             ListMgr.remove($\mathbf{z}^{(j)}$)
14         else
15             $\widehat{X}_E \leftarrow \widehat{X}_E \cup \{\mathbf{x}^{(j)}\}$
16             $\widehat{Z}_N \leftarrow \widehat{Z}_N \cup \{\mathbf{f}(\mathbf{x}^{(j)})\}$
17             ListMgr.update($\mathbf{z}^{(j)}, \mathbf{f}(\mathbf{x}^{(j)})$)
18     return $\widehat{X}_E$, $\widehat{Z}_N$
```

Module A.2: *BasicListMgr*

```
1  Procedure init($\mathbf{z}^{\text{ub}}, \mathbf{z}^{\text{lb}}$):
2      $L \leftarrow \{\mathbf{z}^{\text{ub}}\}$
3  Procedure getSearchRegion():
4      return $\mathbf{z}^{(j)} \in L$
5  Procedure listLen():
6      return $|L|$
7  Procedure remove($\mathbf{z}^{(j)}$):
8      $L \leftarrow L \setminus \{\mathbf{z}^{(j)}\}$
9  Procedure update($\mathbf{z}^{(j)}, \mathbf{f}(\mathbf{x}^{(j)})$):
10     remove($\mathbf{z}^{(j)}$)
11     $L \leftarrow L \cup \{\mathbf{z}^{(j),-k} : k \in P, z_k^{(j)} <$
           $z_k^{\text{lb}}\}$
```

Module A.3: *MAWTSrch*

```
1  Procedure init($\mathbf{z}^{\text{ub}}, \mathbf{z}^{\text{lb}}, s$):
2      $r \leftarrow \max_{k \in P}\{z_k^{\text{ub}} - z_k^{\text{lb}}\}$
3      $\epsilon \leftarrow (0, s/(2p(r - s)))$
4  Procedure search($\mathbf{z}^{(j)}$):
5      for $k \in P$ do $w_k \leftarrow 1/(z_k^{(j)} - z_k^{\text{lb}})$
6      $(\mathbf{x}^*, g(\mathbf{f}(\mathbf{x}^*)) \leftarrow$
           $\min_{\mathbf{x} \in X}\{\|\mathbf{z}(\mathbf{x}) - \mathbf{z}^{\text{lb}}\|^{\mathbf{w},\epsilon}\}$
7      if $g(\mathbf{z}(\mathbf{x}^*)) < 1$ then
8          return $\mathbf{x}^*$
9      else
10         return Null
```

Algorithm 1: BOX: An object-oriented version of Algorithm 2.1.

Module A.4: *KSController*

```
Procedure $init(P, s, \mathbf{z}^{\text{ub}})$:
    for $k \in P$ do
        $z_k^{\text{lb}} \leftarrow \min\{f_k(\mathbf{x}) : x \in X\}$
    ListMgr $\leftarrow$
      KSListMgr.$init(\mathbf{z}^{\text{ub}},\ \mathbf{z}^{\text{lb}})$
    Search $\leftarrow$ KSSrch.$init(\mathbf{z}^{\text{ub}}, \mathbf{z}^{\text{lb}}, s)$
    $\widehat{X}_E \leftarrow \emptyset$; $\widehat{Z}_N \leftarrow \emptyset$; $j \leftarrow 0$
Procedure $FindNonDom()$:
    See Module A.1
```

Module A.5: *KSSearch*

```
Procedure $init(\mathbf{z}^{\text{ub}}, \mathbf{z}^{\text{lb}}, s)$:
    return
Procedure $run((\mathbf{z}^i, \mathbf{z}^n))$:
    for $k \in \{1, \ldots, p-1\}$ do
        Add constraint $f_k(\mathbf{x}) \leq z_k^n$
    $(\mathbf{x}^*, g(\mathbf{f}(\mathbf{x}^*))) \leftarrow \min_{\mathbf{x} \in X}\{f_p(\mathbf{x})\}$
    if Problem infeasible then
        Remove added constraints
        return Null
    else
        Add constraint $f_p(\mathbf{x}) = f_p(\mathbf{x}^*)$
        $(\mathbf{x}^*, g(\mathbf{f}(\mathbf{x}^*))) \leftarrow$
          $\min_{\mathbf{x} \in X}\{\sum_{k=1}^{p} f_k(\mathbf{x})\}$
        Remove added constraints
        return $\mathbf{x}^*$
```

Module A.6: *KSListMgr*

```
Procedure $init(\mathbf{z}^{\text{ub}}, \mathbf{z}^{\text{lb}})$:
    $L \leftarrow \{([\mathbf{z}_1^{\text{lb}}, \ldots, \mathbf{z}_{p-1}^{\text{lb}}], [\mathbf{z}_1^{\text{ub}}, \ldots, \mathbf{z}_{p-1}^{\text{ub}}])\}$
Procedure $getSearchRegion()$:
    return $(\mathbf{z}^i, \mathbf{z}^n) \leftarrow \text{argmax}_{(\mathbf{z}^i, \mathbf{z}^n) \in L}\{\Pi_{k=1}^{p-1}(z_k^{\text{ub}} - z_k^{\text{lb}})\}$
Procedure $remove([\mathbf{f}_1(\mathbf{x}^{(j)}), \ldots, \mathbf{f}_{p-1}(\mathbf{x}^{(j)})])$:
    for $(\mathbf{i}, \mathbf{n}) \in L$ do
        if $\mathbf{i} \geqq [\mathbf{f}_1(\mathbf{x}^{(j)}), \ldots, \mathbf{f}_{p-1}(\mathbf{x}^{(j)})]$ AND $\mathbf{n} \leqq \mathbf{z}^n$ then    // $\mathbf{z}^n$ is from line 4
            $L \leftarrow L \setminus \{(\mathbf{i}, \mathbf{n})\}$
Procedure $listLen()$:
    return $|L|$
Procedure $update((\mathbf{z}^i, \mathbf{z}^n), \mathbf{f}(\mathbf{x}^{(j)}))$:
    for $(\mathbf{i}, \mathbf{n}) \in L$ do                        // $(\mathbf{i}, \mathbf{n})$ is a candidate parent
        $L \leftarrow L \setminus \{(\mathbf{i}, \mathbf{n})\}$
        $T \leftarrow \{(\mathbf{i}, \mathbf{n})\}$              // $T$ is the container for children of $(\mathbf{i}, \mathbf{n})$
        for $k \in \{1, \ldots, p-1\}$ do                  // Iterate through objectives
            if $i_k < f_k(\mathbf{x}^{(j)}) < n_k$ then             // Check if we need $k$-children
                for $(\mathbf{i}, \mathbf{n}) \in T$ do        // Loop divides $(\mathbf{i}, \mathbf{n}) \in T$ into 2 $k$-children
                    $T \leftarrow T \setminus \{(\mathbf{i}, \mathbf{n})\}$
                    newN $\leftarrow [n_1, \ldots, n_{k-1}, f_k(\mathbf{x}^{(j)}), n_{k+1}, \ldots, n_p]$
                    newI $\leftarrow [i_1, \ldots, i_{k-1}, f_k(\mathbf{x}^{(j)}), i_{k+1}, \ldots, i_p]$
                    $T \leftarrow T \cup \{(\mathbf{i}, \mathbf{newN}), (\mathbf{newI}, \mathbf{n})\}$
        Set $L = L \cup T$
    $remove([f_1(\mathbf{x}), \ldots, f_{p-1}(\mathbf{x})])$
```

Algorithm 2: KS: an object-oriented implementation of Kirlik and Sayin's ϵ-constraint method.

A.2 SPIP-I Algorithms

In Chapter 3 we use the MAWT norm of Chapter 2 to generate a complete set of non-dominated solutions to the SPIP-I. In this section we give two algorithms that tailor Algorithm 2.1 to the SPIP-I.

A.2.1 `GetIdeal`

Algorithm 2.1 finds an ideal point $\mathbf{z}^{\text{lb}} = [z_0(\mathbf{x}_0^*), \dots, z_p(\mathbf{x}_p^*)]$ in lines 1–2, where each $\mathbf{x}_{k^F}^*$ is an optimal solution to the problem of minimizing $z_{k^F}(\mathbf{x})$. A simple implementation for this routine would be to solve (3.1) for each $k^F \in \{0, \dots, p\}$. Algorithm A.3 accelerates this simple routine by leveraging the property that $z_{k^F}(\mathbf{x}_{k^F}^*)$ is non-increasing over k^F.

```
Algorithm A.3: GetIdeal($\bar{G}$)
Input        : $\bar{G}$: A layerized graph
               $\{c_{ij}\}_{(i,j)\in A}$: The collection of costs for arcs in $A$
Output       : $\mathbf{z}^I = [z_0(\mathbf{x}_0^*), \dots, z_p(\mathbf{x}_p^*)]$, where each $\mathbf{x}_{k^F}^*$ is optimal for (3.1)
Dependencies : SolveSPIP($G, \{c_{ij}\}_{(i,j)\in A}, f$): Solves $\min_{\mathbf{x}\in X}\{f(\mathbf{z}(\mathbf{x})) : \text{(3.3b)–(3.3f)}\}$
               returning an optimal solution $\mathbf{x}^*$ and the corresponding objective
               vector $\mathbf{z}(\mathbf{x}^*)$
1  $k^F \leftarrow 0$
2  while $k^F \le p$ do
3      $f(\mathbf{z}) \leftarrow (-z_{k^F})$                                  // $f : \mathbb{R}^{p+1} \to \mathbb{R}$
4      $(\mathbf{x}^*, \mathbf{z}(\mathbf{x}^*)) \leftarrow$ SolveSPIP*($\bar{G}, \{c_{ij}\}_{(i,j)\in A}, f$)
5      $z^* \leftarrow z_{k^F}(\mathbf{x}^*)$
6      while $k^F \le p$ do
7          if $z^* = z_{k^F}(\mathbf{x}^*)$ then
8              $z_{k^F}^I \leftarrow z_{k^F}(\mathbf{x}^*)$
9              $k^F \leftarrow k^F + 1$
10         else
11             break
12 return $\mathbf{z}^I$
```

Algorithm A.3 begins by setting $f(\mathbf{z}) = -z_0$, and using `SolveSPIP` (see Algorithm A.4 from Appendix A.2.2) to find a strategy $\mathbf{x}^*$ that minimizes $f(\mathbf{z})$ (i.e., maximizes z_0). Lines 5–11 are the accelerating portion. First, line 5 selects the k^F component of $\mathbf{z}(\mathbf{x}^*)$ and assigns that value to z^*. For each incremental k^F, if $z^* = z_{k^F}(\mathbf{x}^*)$, then $\mathbf{x}^*$ optimizes z_{k^F}. Lines 6–9 continue to increment k^F and assign $z_{k^F}^I$ component values as long as $\mathbf{x}^*$ is optimal for each z_{k^F}. Once a value for k^F is found for which $\mathbf{x}^*$ is not provably optimal using the acceleration technique, the algorithm breaks out of the inner loop in line 11 and returns to line 2 to solve for a new strategy. When all component values of $\mathbf{z}^I$ are set, the routine terminates, returning $\mathbf{z}^I$.

A.2.2 `SolveSPIP`

The auxiliary routine `SolveSPIP` generalizes row-generating algorithms for single-objective SPIPs [36, 50], allowing them to solve multi-objective SPIPs with scalarized objectives. These row-generating methods iteratively generate a set of short paths (i.e., columns in the follower's inner problem). In each iteration they solve the relaxed master problem of selecting an optimal interdiction strategy, where the follower is restricted to paths in the set of short paths. The optimal strategy is used to establish a new set of arc costs, which are passed again to the subproblem for finding new short paths. The routine terminates when the master (interdiction) and subproblem (shortest-path) solutions' objectives are equal.

Our implementation of `SolveSPIP`, given in Algorithm A.4, extends these methods to solve problems of the form $\min\{g(\mathbf{z}(\mathbf{x}^L)) : (3.3\text{b})\text{–}(3.3\text{f})\}$, where $g : \mathbb{R}^{p+1} \to \mathbb{R}$. Though SPIPs are generally maximization problems, we use the minimization operator, because $g(\mathbf{z}(\mathbf{x}) = \|\mathbf{z}^{\text{ub}} - \mathbf{z}\|$ where $\mathbf{z}^{\text{ub}} \geqq \mathbf{z}(\mathbf{x}),\ \forall \mathbf{x} \in X$, and $\|\cdot\|$ is the MAWT norm (see line 10 of Algorithm 2.1). Since the SPIP-I is a maximization problem, the roles of $\mathbf{z}^{\text{lb}}$ and $\mathbf{z}^{\text{ub}}$ are exchanged from the minimization formulation in Chapter 2, i.e., the leader seeks to minimize the distance between $\mathbf{z}(\mathbf{x})$ and $\mathbf{z}^{\text{ub}}$. Thus, while the multiobjective variant of (3.3) seeks to *maximize* $\mathbf{z}(\mathbf{x}^*)$, Algorithm A.4 achieves this by solving the *minimization* problem: $\min_{\mathbf{x}\in X}\{g(\mathbf{z}(\mathbf{x}))\}$.

Algorithm A.4 starts by establishing an empty set of $s_0 \to t_h$ paths, Π_h, for each $h \in \{0, \ldots, p\}$. Lines 2–3 set a trivial interdiction strategy and upper and lower bounds on the optimal objective. The routine loops until these upper and lower bounds are equal. Each iteration, the routine begins in lines 5–8 by solving the subproblem: setting the arc costs in $\bar{G}$ according to the latest strategy, finding a set of short paths (we used iGraph's [16] implementation of Dijkstra's algorithm), and establishing the upper bound value, f^{UB}, from the vector of path costs. Then line 9 updates the master problem, adding the paths from the subproblem results as new constraints. Line 10 solves the master problem to get a new strategy, $\mathbf{x}^*$, and line 11 stores the objective corresponding to $\mathbf{x}^*$ where the follower is restricted to using paths from the sets $\Pi_0, \ldots, \Pi_p$. Line 12 finishes each iteration by updating the lower bound on the objective, f^{LB}. Once $f^{UB} = f^{LB}$, the routine terminates, returning $\mathbf{x}^*$ and $\mathbf{z}(\mathbf{x}^*)$.

```
Algorithm A.4: SolveSPIP($\bar{G}, \{c_{ij}\}_{(i,j)\in A}, f$)
    Input        : $\bar{G} = (\bar{N}, \bar{A})$: A layered graph
                   $\{c_{ij}\}_{(i,j)\in A} : \mathbb{Z}^2 \to \mathbb{R}$: A collection of cost functions over the base arc
                   set, $A$
                   $f$: A function mapping $\mathbb{R}^{p+1} \to \mathbb{R}$
    Output       : $\mathbf{x}^*$: An interdiction strategy minimizing $f(\mathbf{z}(\mathbf{x}))$
                   $\mathbf{z}(\mathbf{x}^*)$: The objective vector for $\mathbf{x}^*$
    Dependencies : Dijkstra($G, s, T, \mathbf{c}$) $\to \{\pi_t\}_{t\in T}$
                   Arcs($\pi$): Returns the set of arcs along path $\pi$
 1  $\Pi_h \leftarrow \emptyset$ for $h \in \{0, \dots, p\}$
 2  $\mathbf{x}^* \leftarrow \mathbf{0}$
 3  $f^{UB} = \infty$, $f^{LB} = 0$
 4  while $f^{UB} \neq f^{LB}$ do
 5      for $\bar{a} = (i_h, j_{h+\ell}) \in \bar{A}$ do
 6          $\bar{c}_{\bar{a}} \leftarrow c_{ij}(x^*_{ij}, \ell)$                    // At the end $\bar{\mathbf{c}} \in \mathbb{R}^{|\bar{A}|}$
 7      $[\pi_0, \dots, \pi_p] \leftarrow$ Dijkstra($\bar{G}, s, T, \bar{\mathbf{c}}$)
 8      $f^{UB} \leftarrow f\left(\left[\sum_{\bar{a}\in \texttt{Arcs}(\pi_0)} \bar{c}_{\bar{a}}, \dots, \sum_{a\in \texttt{Arcs}(\pi_p)} \bar{c}_{\bar{a}}\right]\right)$
 9      $\Pi_h \leftarrow \Pi_h \cup \{\pi_h\}$ for $h \in \{0, \dots, p\}$
10      $\mathbf{x}^* \in$
          $\text{argmin}_{\mathbf{x}\in X} \left\{ f(\mathbf{z}) : z_{k^F} \leq \sum_{(i_h, j_{h+\ell})\in \text{Arcs}(\pi)} c_{ij}(x_{ij}, \ell),\ \forall k^F \in \{0, \dots, p\},\ \forall \pi \in \Pi_{k^F} \right\}$
11      $\mathbf{z} \leftarrow \left[ \min_{\pi\in\Pi_0} \left\{ \sum_{(i_h, j_{h+\ell})\in \text{Arcs}(\pi)} c_{ij}(x^*_{ij}, \ell) \right\}, \dots, \right.$
              $\left. \min_{\pi\in\Pi_p} \left\{ \sum_{(i_h, j_{h+\ell})\in \text{Arcs}(\pi)} c_{ij}(x^*_{ij}, \ell) \right\} \right]$
12      $f^{LB} \leftarrow f(\mathbf{z})$
13  return $\mathbf{x}^*, \mathbf{z}$
```

A.2.3 `DecompTree`

Our `DecompTree` routine is based on the work of Valdes [86]. We use Python dictionaries to implicitly store the tree. Dictionaries are initialized in Python using brace notation: `MyDict` $\leftarrow$ `{key1:value1, key2:value2}`. After initialization, specific key/value pairs may be added to the dictionary using brackets: `MyDict[key3]` $\leftarrow$ `value3`. Dictionary look-ups also use bracket notation: `MyDict[key1]` returns `value1`.

Algorithm A.5: `DecompTree`(G)

```
Input        : $G = (N, A)$: An SPG
Output       : $r$: The root of the decomposition tree, $\mathcal{T}$
Dependencies :
1  Parent $\leftarrow \{a : n_a$ for $a \in A\}$
2  Children $\leftarrow \{n_a : a$ for $a \in A\}$
3  Type $\leftarrow \{n_a : L$ for $a \in A\}$
4  UnSat $\leftarrow N$
5  while UnSat $\neq \emptyset$ do
6      Choose $n \in$ UnSat; UnSat $\leftarrow$ UnSat $\setminus \{n\}$
7      while $\exists a_1, a_2 \in A : a_1 = a_2 = (v, n)$ do
8          $n_{a_1} \leftarrow$ Parent$[a_1]$; $n_{a_2} \leftarrow$ Parent$[a_2]$
9          $a_{\text{new}} \leftarrow (v, n)$
10         Parent$[a_{\text{new}}] \leftarrow n_{a_{\text{new}}}$
11         $A \leftarrow (A \setminus \{a_1, a_2\}) \cup \{a_{\text{new}}\}$
12         Parent$[n_{a_1}] \leftarrow n_{a_{\text{new}}}$; Parent$[n_{a_2}] \leftarrow n_{a_{\text{new}}}$
13         Children$[n_{a_{\text{new}}}] \leftarrow \{n_{a_1}, n_{a_2}\}$
14         Type$[n_{a_{\text{new}}}] \leftarrow P$
15     while $\exists a_1, a_2 \in A : a_1 = a_2 = (n, w)$ do
16         $n_{a_1} \leftarrow$ Parent$[a_1]$; $n_{a_2} \leftarrow$ Parent$[a_2]$
17         $a_{\text{new}} \leftarrow (v, n)$
18         Parent$[a_{\text{new}}] \leftarrow n_{a_{\text{new}}}$
19         $A \leftarrow (A \setminus \{a_1, a_2\}) \cup \{a_{\text{new}}\}$
20         Parent$[n_{a_1}] \leftarrow n_{a_{\text{new}}}$; Parent$[n_{a_2}] \leftarrow n_{a_{\text{new}}}$
21         Children$[n_{a_{\text{new}}}] \leftarrow \{n_{a_1}, n_{a_2}\}$
22         Type$[n_{a_{\text{new}}}] \leftarrow P$
23     if $\exists! a_1 \in A : a_1 = (v, n)$ AND $\exists! a_2 \in A : a_2 = (n, w)$ then
24         $n_{a_1} \leftarrow$ Parent$[a_1]$; $n_{a_2} \leftarrow$ Parent$[a_2]$
25         $a_{\text{new}} \leftarrow (v, w)$
26         Parent$[a_{\text{new}}] \leftarrow n_{a_{\text{new}}}$
27         $A \leftarrow (A \setminus \{a_1, a_2\}) \cup \{a_{\text{new}}\}$
28         $N \leftarrow N \setminus \{n\}$
29         Parent$[n_{a_1}] \leftarrow n_{a_{\text{new}}}$; Parent$[n_{a_2}] \leftarrow n_{a_{\text{new}}}$
30         Children$[n_{a_{\text{new}}}] \leftarrow \{n_{a_1}, n_{a_2}\}$
31         Type$[n_{a_{\text{new}}}] \leftarrow S$
32         UnSat $\leftarrow$ UnSat $\cup \{v, w\}$
33 return Parent$[(s, t)]$                        // At termination, $A = \{(s, t)\}$
```

The algorithm begins by initializing dictionaries `Parent`, `Children`, and `Type` with leaf nodes mapped to the arcs in G. These dictionaries serve dual purposes: they maintain the tree structure in lieu of a graph library, and they serve as their namesake functions in Algorithms 3.2 and 3.3. Line 3 initializes a set of unsatisfied nodes to include all nodes from G. The main loop continues until all nodes are satisfied. Each iteration selects one unsatisfied node, n, and removes it from the unsatisfied set. Lines 6–10 merge parallel inbound arcs at node n, establishing a new node, $n_{a_{\text{new}}} \in \mathcal{T}$ of type P, and forming appropriate hierarchical relationships in $\mathcal{T}$. Lines 11–15 similarly merge parallel outbound arcs at node n. After merging all parallel inbound and outbound arcs, if n is now a intermediary node, having a single predecessor and a single successor, then the routine performs a series merge on the inbound and outbound arcs, establishing a new tree node, $n_{a_{\text{new}}} \in \mathcal{T}$ of type S. A series reduction requires that the predecessor and successor of the reduced node are categorized as "unsatisfied." When the set of unsatisfied nodes is empty, A contains a single arc, (s, t). The corresponding node is returned as the root node, terminating the routine.

A.3 SPIP-RS Algorithms

A.3.1 Spatial Branch and Bound

Algorithm A.6 states a spatial branch-and-bound algorithm to solve the convex SPIP-RS within an absolute optimality tolerance gap of some given parameter $\epsilon > 0$. In each iteration, the algorithm solves the LP relaxation over the convex hull of (4.6). It then iteratively branches on t_a variables until a solution is available that is provably within ϵ of the true optimal objective.

Lines 1 and 2 initialize a node list for the branch-and-bound tree with a single root node. Each node, v, in the tree is associated with upper and lower bounds on the decision variables $\mathbf{t}$, denoted $\overline{\mathbf{t}^v}$ and $\underline{\mathbf{t}^v}$, respectively. For the root node, r, line 2 sets $\overline{\mathbf{t}^r} = \{b\}^m$ and $\underline{\mathbf{t}^r} = \{0\}^m$. For each node, immediately after setting $\overline{\mathbf{t}^v}$ and $\underline{\mathbf{t}^v}$, we solve the LP relaxation of (4.1) over the region bounded by $\overline{\mathbf{t}^v}$ and $\underline{\mathbf{t}^v}$:

$$P(\overline{\mathbf{t}^v}, \underline{\mathbf{t}^v}): \qquad \max \quad d_s^v \tag{12a}$$

$$\text{s.t. } d_i^v - d_j^v \leq c_{ij}\left(\underline{t_{ij}^v}\right) + \left(\frac{t_{ij}^v - \underline{t_{ij}^v}}{\overline{t_{ij}^v} - \underline{t_{ij}^v}}\right)\left(c_{ij}\left(\overline{t_{ij}^v}\right) - c_{ij}\left(\underline{t_{ij}^v}\right)\right) \quad \forall (i,j) \in \mathcal{A} \tag{12b}$$

$$d_t^v = 0 \tag{12c}$$

$$\underline{t_a^v} \leq t_a^v \leq \overline{t_a^v} \qquad \forall a \in \mathcal{A}. \quad \text{(12d)}$$

Since (12) is a relaxation of the SPIP-RS (from the interdictor's perspective), the optimal objective, d_s^v, is an upper bound on the optimal objective for (4.1) within the same bounds. The initialization portion concludes in line 3 by defining a global lower bound value, `LB`, and setting it to $-\infty$.

After initialization is completed, the algorithm begins the branching and pruning process. Each iteration begins in lines 5–6 by choosing a node having the highest upper bound on the objective, d_s^v, and removing that node from the list of nodes to search. Line 7 checks whether the incumbent solution is within ϵ of d_s^v; if so, then all remaining nodes are fathomed and the algorithm returns the incumbent solution. Otherwise, line 8 checks if this node achieves an objective better than the current best lower bound, using $X^*(\mathbf{t}^v)$ defined as in (4.4) and $z(X^*(\mathbf{t}^v))$ computed using the method of [33] described in Section 4.1. If $z(X^*(\mathbf{t}^v)) > \texttt{LB}$, then this node's objective is better than the incumbent, and lines 9–10 establish this node as the new incumbent. Next, line 11 effectively fathoms this node if $d_s^v \leq \texttt{LB} + \epsilon$. Otherwise, line 12 adds two child nodes. Branching occurs on an arc having the greatest gap between the interpolated cost and true expected cost, chosen in line 13. Lines 14–15 establish two children: One child is restricted by $t_a \leq t_a^*$, and the other is restricted by $t_a \geq t_a^*$. Finally, problem (12) is solved for each child, and the algorithm reiterates. When no unpruned nodes remain in the tree, the algorithm terminates in line 16, returning the incumbent solution.

Algorithm A.6: Spatial Branch and Bound Algorithm

```
Input: $G = (\mathcal{N}, \mathcal{A})$: A directed graph
       $s, t \in \mathcal{N}$: Source and terminal nodes
       $b \in \mathbb{Z}_+$: Interdiction budget
       $\{c_a : \{0, \ldots, b\} \to \mathbb{R}_+\}_{a \in \mathcal{A}}$: Convex cost functions for each $a \in \mathcal{A}$
       $\epsilon > 0$: Optimality tolerance parameter.
Output: $X \in [0,1]^{b \times m}$: An optimal interdiction strategy
1  NodeList ← $\{r\}$
2  $\underline{\mathbf{t}^r} \leftarrow \{0\}^m$; $\overline{\mathbf{t}^r} \leftarrow \{b\}^m$; $(\mathbf{t}^r, \mathbf{d}^r) \leftarrow$ Solve $\left(P(\overline{\mathbf{t}^r}, \underline{\mathbf{t}^r})\right)$
3  LB ← $-\infty$
4  while NodeList $\neq \emptyset$ do
5      $n \leftarrow \arg\max_{k \in \texttt{NodeList}}\{d_s^k\}$
6      NodeList ← NodeList $\setminus \{n\}$
7      if $d_s^v > LB + \epsilon$ then return $X^*(\mathbf{t}^{\text{Incumbent}})$
8      if $z(X^*(\mathbf{t}^v)) > LB$ then
9          Incumbent ← $n$
10         LB ← $z(X^*(\mathbf{t}^v))$
11     if $d_s^v > LB + \epsilon$ then
12         NodeList ← NodeList $\cup \{c_1, c_2\}$
13         $a \leftarrow \arg\max_{\alpha \in \mathcal{A}} \left\{ c_\alpha\left(\underline{t_\alpha^v}\right) + \left(\frac{t_\alpha^v - \underline{t_\alpha^v}}{\overline{t_\alpha^v} - \underline{t_\alpha^v}}\right)\left(c_\alpha\left(\overline{t_\alpha^v}\right) - c_\alpha\left(\underline{t_\alpha^v}\right)\right) - \mathrm{E}\left(c_\alpha(\tau) | \mathbf{x}^*(t_\alpha^v)\right)\right\}$
14         $\underline{\mathbf{t}^{c_1}} \leftarrow \underline{\mathbf{t}^v}$; $\overline{\mathbf{t}^{c_1}} \leftarrow \left[\overline{t_1^v}, \ldots, \overline{t_{a-1}^v}, t_a^*, \overline{t_{a+1}^v}, \ldots, \overline{t_m^v}\right]$; $(\mathbf{t}^{c_1}, \mathbf{d}^{c_1}) \leftarrow$ Solve $\left(P(\overline{\mathbf{t}^{c_1}}, \underline{\mathbf{t}^{c_1}})\right)$
15         $\underline{\mathbf{t}^{c_2}} \leftarrow \left[\underline{t_1^v}, \ldots, \underline{t_{a-1}^v}, t_a^*, \underline{t_{a+1}^v}, \ldots, \underline{t_m^v}\right]$; $\overline{\mathbf{t}^{c_2}} \leftarrow \overline{\mathbf{t}^v}$; $(\mathbf{t}^{c_2}, \mathbf{d}^{c_2}) \leftarrow$ Solve $\left(P(\overline{\mathbf{t}^{c_2}}, \underline{\mathbf{t}^{c_2}})\right)$
16 return $X^*(\mathbf{t}^{\text{Incumbent}})$
```

A.3.2 Lazy Construction

Algorithm A.7 describes a "lazy" approach to implementing SPIP-RS algorithms in this dissertation. In each iteration, the algorithm directs the solver to examine a subgraph of the input graph. Using the returned interdiction strategy, the algorithm then identifies a shortest path on the original graph. If the shortest path is contained in the subgraph examined in the current iteration, then the algorithm terminates. Otherwise, the algorithm adds the arcs from the shortest path to generate a new subgraph, and the algorithm reiterates.

Line 1 initializes an empty subgraph arc set, $\mathcal{A}^0$, and an initial solution, $X^0 = [0]$. Lines 2–8 give the main loop. Each iteration, $k \in \{1, 2, \ldots\}$ begins in lines 3 and 4 by setting a vector, $\mathbf{w}$, of expected arc costs given by the previous iteration's strategy, X^{k-1}. Line 5 finds the follower's shortest expected-cost path on the full graph, G. If the shortest path uses only arcs contained in the previous iteration's subgraph, $\mathcal{A}^{k-1}$, then line 6 terminates the algorithm. Otherwise, line 7 generates a new arc set, $\mathcal{A}^k$, and line 8 solves the SPIP-RS problem on the subgraph induced by $\mathcal{A}^k$ to give an updated solution, X. In our implementation, we allowed the solvers to "warm start" in each iteration with the previous solution, X^{k-1}. Finally, lines 9–10 convert $X \in [0,1]^{b \times |\mathcal{A}^k|}$ to $X^k \in [0,1]^{b \times m}$ by adding zero-columns for arcs not in $\mathcal{A}^k$.

Algorithm A.7: Lazy Algorithm

```
Input: $G = (\mathcal{N}, \mathcal{A})$: A directed graph
       $b \in \mathbb{Z}_+$: Interdiction budget
       $\{c_a : \{0, \ldots, b\} \to \mathbb{R}_+\}_{a \in \mathcal{A}}$: Non-decreasing cost functions for each $a \in \mathcal{A}$
Dependencies: Dijkstra$(G, u, v, \mathbf{w})$: Returns set of arcs representing a shortest
                  $u \to v$ path in $G$, given arc costs $\mathbf{w} \in \mathbb{R}_+^m$
              Solve$(G, b)$: Returns an optimal (or near-optimal) solution to the
                  SPIP-RS represented by $G$ and budget $b$
Output: $X \in [0,1]^{b \times m}$: A (near-)optimal interdiction strategy
1  $\mathcal{A}^0 \leftarrow \{\}$; $X^0 \leftarrow [0]^{b \times m}$
2  for $k \in \{1, 2, \ldots\}$ do
3      for $a \in \mathcal{A}$ do
4          $w_a \leftarrow \sum_{t=0}^{b} c_a(t) \Pi^b(\mathbf{x}_a^{k-1}, t)$
5      $P^k \leftarrow$ Dijkstra$(G, s, t, \mathbf{w})$
6      if $P^k \subseteq \mathcal{A}^{k-1}$ then return $X^{k-1}$
7      $\mathcal{A}^k \leftarrow \mathcal{A}^{k-1} \cup P^k$
8      $X \leftarrow$ Solve$(G(\mathcal{N}, \mathcal{A}^k), b)$
9      for $a \in \mathcal{A}$ do
10         if $a \in \mathcal{A}^k$ then $\mathbf{x}_a^k \leftarrow \mathbf{x}_a$ else $\mathbf{x}_a^k \leftarrow [0]^b$
```

Bibliography

[1] M. Alves and J. Clímaco. An interactive reference point approach for multiobjective mixed-integer programming using branch-and-bound. *European Journal of Operational Research*, 124(3):478–494, 2000.

[2] B. Bahamondes, J. Correa, J. Matuschke, and G. Oriolo. Adaptivity in network interdiction. In S. Rass, B. An, C. Kiekintveld, F. Fang, and S. Schauer, editors, *Decision and Game Theory for Security*, Security and Cryptology (vol. 10575), pages 40–52, Vienna, Austria, 2017. Springer Berlin / Heidelberg.

[3] M. O. Ball, B. L. Golden, and R. V. Vohra. Finding the most vital arcs in a network. *Operations Research Letters*, 8(2):73–76, 1989.

[4] A. Bar-Noy, S. Khuller, and B. Schieber. The complexity of finding most vital arcs and nodes. Technical Report UMIACS-TR-95-96, Univ. of Maryland Institute for Advanced Computer Studies, College Park, MD, 1995.

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

[6] P. Belotti, B. Soylu, and M. M. Wiecek. A branch-and-bound algorithm for biobjective mixed integer programs. *Optimization Online*, 2013.

[7] A. Ben-Tal, L. El-Ghaoui, and A. Nemirovski. *Robust Optimization.* Princeton University Press, Princeton, NJ, 2009.

[8] J. R. Birge and F. V. Louveaux. *Introduction to Stochastic Programming.* Springer, New York, 1997.

[9] R. Borndörfer, J. Buwaya, G. Sagnol, and E. Swarat. Optimizing toll enforcement in transportation networks: a game-theoretic approach. *Electronic Notes in Discrete Mathematics*, 41:253–260, 2013.

[10] J. S. Borrero, O. A. Prokopyev, and D. Sauré. Sequential shortest path interdiction with incomplete information. *Decision Analysis*, 13(1):68–98, 2015.

[11] V. Bowman Jr. On the relationship of the Tchebycheff norm and the efficient frontier of multiple-criteria objectives. In H. Thiriez and S. Zionts, editors, *Multiple Criteria Decision Making: Lecture Notes in Economics and Mathematical Systems*, volume 130, pages 76–86. Springer, Berlin, Heidelberg, 1976.

[12] K. Bringmann. Bringing order to special cases of Klee's measure problem. In K. Chatterjee and J. Sgall, editors, *Mathematical Foundations of Computer Science*, volume 8087, pages 207–218. Springer, Berlin, Heidelberg, 2013.

[13] R. H. Byrd, J. Nocedal, and R. A. Waltz. KNITRO: An integrated package for nonlinear optimization. In G. di Pillo and M. Roma, editors, *Large-Scale Nonlinear Optimization*, pages 35–59. Springer-Verlag, 2006.

[14] K. J. Cormican, D. P. Morton, and R. K. Wood. Stochastic network interdiction. *Operations Research*, 46(2):184–197, 1998.

[15] J. Correa, T. Harks, V. J. C. Kreuzen, and J. Matuschke. Fare evasion in transit networks. *Operations Research*, 65(1):165–183, 2017.

[16] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal of Complex Systems*, 1695, 2006.

[17] K. Dächert, J. Gorski, and K. Klamroth. An augmented weighted Tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems. *Computers and Operations Research*, 39(12):2929–2943, 2012.

[18] K. Dächert, K. Klamroth, R. Lacour, and D. Vanderpooten. Efficient computation of the search region in multi-objective optimization. *European Journal of Operational Research*, 260(3):841–855, 2017.

[19] E. P. Durbin. *An Interdiction Model of Highway Transportation.* Rand Corporation, 1966.

[20] M. Ehrgott. *Multicriteria Optimization.* Springer, Berlin, 2nd edition, 2005.

[21] M. Ehrgott and S. Ruzika. Improved ϵ-constraint method for multiobjective programming. *Journal of Optimization Theory and Applications*, 138(3):375–396, 2008.

[22] P. Erdös and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.

[23] J. E. Falk and R. M. Soland. Algorithm for separable nonconvex programming problems. *Management Science*, 15(9):550–569, 1969.

[24] D. R. Fulkerson and G. C. Harding. Maximizing minimum source-sink path subject to a budget constraint. *Mathematical Programming*, 13(1):116–118, 1977.

[25] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman & Co., Princeton, NJ, 1979.

[26] V. Goel and I. E. Grossmann. A class of stochastic programs with decision dependent uncertainty. *Mathematical Programming*, 106(2):355–394, 2006.

[27] Y. Haimes, L. Lasdon, and D. Wismer. On a bicriterion formulation of the problem of integrated system identification and system optimizations. *IEEE Transactions on Systems, Man, and Cybernetics*, 1:296–297, 1971.

[28] H. W. Hamacher, C. R. Pederson, and S. Ruzika. Finding representative systems for discrete bicriterion optimization problems. *Operations Research Letters*, 35(3):336–344, 2007.

[29] H. Held, R. Hemmecke, and D. L. Woodruff. A decomposition algorithm applied to planning the interdiction of stochastic networks. *Naval Research Logistics*, 52(4):321–328, 2005.

[30] H. Held and D. L. Woodruff. Heuristics for multi-stage interdiction of stochastic networks. *Journal of Heuristics*, 11(6):483–500, 2005.

[31] R. Hemmecke, R. Schultz, and D. L. Woodruff. Interdicting stochastic networks. In D. L. Woodruff, editor, *Network Interdiction and Stochastic Integer Programming*, pages 71–80. Kluwer Academic Publishers, Boston, MA, 2003.

[32] R. Hites, Y. De Smet, N. Risse, M. Salazar-Neumann, and P. Vincke. About the applicability of MCDA to some robustness problems. *European Journal of Operational Research*, 174(1):322–332, 2006.

[33] T. Holzmann and J. C. Smith. Optimizing randomized interdiction strategies in shortest path interdiction problems. In H. Romeijn, A. Schaefer, and R. Thomas, editors, *Proceedings of the 2019 IISE Annual Conference*, 2019.

[34] C.-L. Hwang and A. Masud. Multiple objective decision making – methods and applications: A state-of-the-art survey. In M. Beckman and H. Künzi, editors, *Lecture Notes in Economics and Mathematical Systems*, volume 164. Springer-Verlag, New York, 1979.

[35] D. A. Iancu and N. Trichakis. Pareto efficiency in robust optimization. *Management Science*, 60(1):130–147, 2014.

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

[37] U. Janjarassuk and J. Linderoth. Reformulation and sampling to solve a stochastic network interdiction problem. *Networks*, 52(3):120–132, 2008.

[38] I. Kaliszewski. A modified weighted Tchebycheff metric for multiple objective programming. *Computers and Operations Research*, 14(4):315–323, 1987.

[39] I. Kaliszewski. Using trade-off information in decision-making algorithms. *Computers and Operations Research*, 27(2):161–182, 2000.

[40] P. Kall and S. W. Wallace. *Stochastic Programming.* John Wiley and Sons, Chichester, UK, 1994.

[41] G. Kirlik and S. Sayin. A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232(3):479–488, 2014.

[42] K. Klamroth, E. Köbis, A. Schöbel, and C. Tammer. A unified approach for different concepts of robustness and stochastic programming via non-linear scalarizing functionals. *Optimization*, 62(5):649–671, 2013.

[43] K. Klamroth, E. Köbis, A. Schöbel, and C. Tammer. A unified approach to uncertain optimization. *European Journal of Operational Research*, 260(2):403–420, 2017.

[44] K. Klamroth, R. Lacour, and D. Vanderpooten. On the representation of the search region in multi-objective optimization. *European Journal of Operational Research*, 245(3):767–778, 2015.

[45] K. Klamroth, J. rgen Tind, and M. M. Wiecek. Unbiased approximation in multicriteria optimization. *Mathematical Methods of Operations Research*, 56(3):413–437, 2002.

[46] D. Klein and E. Hannan. An algorithm for the multiple objective linear programming problem. *European Journal of Operational Research*, 9(4):378–385, 1982.

[47] R. A. Konrad, A. C. Trapp, T. M. Palmbach, and J. S. Blom. Overcoming human trafficking via operations research and analytics: Opportunities for methods, models, and applications. *European Journal of Operational Research*, 259(1):733–745, 2017.

[48] M. Laumanns, L. Thiele, and Z. Eckart. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3):932–942, 2006.

[49] B. Lokman and M. Köksalan. Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization*, 57(2):347–365, 2013.

[50] L. Lozano and J. C. Smith. A backward sampling framework for interdiction problems with fortification. *INFORMS Journal on Computing*, 29(1):123–139, 2017.

[51] B. J. Lunday and H. D. Sherali. A dynamic network interdiction problem. *INFORMATICA*, 21(4):553–574, 2010.

[52] B. J. Lunday and H. D. Sherali. Network interdiction to minimize the maximum probability of evasion with synergy between applied resources. *Annals of Operations Research*, 196(1):411–442, 2012.

[53] G. Mavrotas. Effective implementation of the ϵ-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, 213(2):455–465, 2009.

[54] G. Mavrotas and D. Diakoulaki. A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research*, 107(3):530–541, 1998.

[55] G. Mavrotas and D. Diakoulaki. Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming. *Applied Mathematics and Computation*, 171(1):53–71, 2005.

[56] G. Mavrotas and K. Florios. An improved version of the augmented ϵ-constraint method (AUGMECON2) for finding the exact pareto set in multi-objective integer programming problems. *Applied Mathematics and Computation*, 219(18):9652–9669, 2013.

[57] C. L. Monma and J. B. Sidney. Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research*, 4(3):215–224, 1979.

[58] D. P. Morton, F. Pan, and K. J. Saeger. Models for nuclear smuggling interdiction. *IIE Transactions*, 39(1):3–14, 2007.

[59] W. odzimierz Ogryczak. Multiple criteria optimization and decisions under risk. *Control and Cybernetics*, 31(4):975–1003, 2002.

[60] M. Özlen and M. Azizoğlu. Multi-objective integer programming: A general approach for generating all non-dominated solutions. *European Journal of Operational Research*, 199(1):25–35, 2009.

[61] M. Özlen, B. A. Burton, and C. A. G. MacRae. Multi-objective integer programming: An improved recursive algorithm. *Journal of Optimization Theory and Applications*, 160(2):470–482, 2014.

[62] O. Özpeynirci and M. Köksalan. An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Science*, 56(12):2302–2315, 2010.

[63] F. Pan and D. P. Morton. Minimizing a stochastic maximum-reliability path. *Networks*, 52:111–119, 2008.

[64] P. Perny, O. Spanjaard, and L.-X. Storme. A decision-theoretic approach to robust optimization in multivalued graphs. *Annals of Operations Research*, 147(1):317–341, 2006.

[65] A. Przybylski, X. Gandibleux, and M. Ehrgott. Two-phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research*, 185(2):509–533, 2008.

[66] A. Przybylski, X. Gandibleux, and M. Ehrgott. A two-phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optimization*, 7(3):149–165, 2010.

[67] A. Raith and M. Ehrgott. A two-phase algorithm for the biobjective integer minimum cost flow problem. *Computers and Operations Research*, 36(6):1945–1954, 2009.

[68] R. Ramos, S. Alonso, J. Sicilia, and C. Gonzáles. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, 111(3):617–628, 1998.

[69] C. M. Rocco and J. E. Ramirez-Marquez. A bi-objective approach for shortest-path network interdiction. *Computers and Industrial Engineering*, 59(2):232–240, 2010.

[70] C. M. Rocco, J. E. Ramirez-Marquez, and D. E. Salazar. Bi and tri-objective optimization in the deterministic network interdiction problem. *Reliability Engineering and System Safety*, 95(8):887–896, 2010.

[71] J. O. Royset and R. K. Wood. Solving the bi-objective maximum-flow network-interdiction problem. *INFORMS Journal on Computing*, 19(2):175–184, 2007.

[72] J. Salmerón. Deception tactics for network interdiction: A multiobjective approach. *Networks*, 60(1):45–58, 2012.

[73] S. Sayin and P. Kouvelis. The multiobjective discrete optimization problem: a weighted min-max two-stage optimization approach and a bicriteria algorithm. *Management Science*, 51(10):1572–1581, 2005.

[74] B. Schandl, K. Klamroth, and M. M. Wiecek. Norm-based approximation in multicriteria programming. *Computers and Mathematics with Applications*, 44(7):925–942, 2002.

[75] A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures in Stochastic Programming: Modeling and Theory.* SIAM, Philadelphia, PA, 2009.

[76] J. C. Smith, M. Prince, and J. Geunes. Modern network interdiction problems and algorithms. In P. M. Pardalos, D.-Z. Du, and R. Graham, editors, *Handbook of Combinatorial Optimization*, pages 1949–1987. Springer, New York, 2nd edition, 2013.

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

[78] S. Steiner and T. Radzik. Computing all efficient solutions of the biobjective minimum spanning tree problem. *Computers and Operations Research*, 35(1):198–211, 2008.

[79] R. Steuer. *Multiple Criteria Optimization: Theory Computation and Application.* John Wiley and Sons, New York, 1986.

[80] R. Steuer and E. Choo. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26(3):326–344, 1983.

[81] T. Stidsen, K. A. Andersen, and B. Damman. A branch and bound algorithm for a class of biobjective mixed integer programs. *Management Science*, 60(4):1009–1032, 2014.

[82] 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.

[83] W. A. Sutherland. *Introduction to Metric and Topological Spaces.* Oxford University Press, New York, 1975.

[84] J. Sylva and A. Crema. A method for finding the set of non-dominated vectors for multiple objective integer linear programs. *European Journal of Operational Research*, 158(1):46–55, 2004.

[85] E. Ulungu and J. Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3(2):83–104, 1994.

[86] J. Valdes. Parsing flowcharts and series-parallel graphs. Technical Report STAN-CS-78-682, Stanford University, 1978.

[87] T. Vincent, S. Florian, S. Ruzika, and A. Przybylski. Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case. *Computers and Operations Research*, 40(1):498–509, 2013.

[88] M. Visée, J. Teghem, M. Pirlot, and E. Ulungu. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. *Journal of Global Optimization*, 12(2):139–155, 1998.

[89] T. W. Jonsbraten, R. J-b Wets, and D. Woodruff. A class of stochastic programs with decision dependent random elements. *Annals of Operations Research*, 82:83–106, 1998.

[90] A. Washburn and R. K. Wood. Two-person zero-sum games for network interdiction. *Operations Research*, 43(2):243–251, 1995.

[91] D. White. The set of efficient solutions for multipe objective shortest path problems. *Computers and Operations Research*, 9(2):101–107, 1982.

[92] A. Wierzbicki. The use of reference objectives in multiobjective optimization. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making Theory and Application*, pages 468–486. Springer, Berlin, 1980.

[93] R. D. Wollmer. Algorithms for targeting strikes in a lines-of-communication network. *Operations Research*, 18(3):497–515, 1970.

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

[95] W. Zhang and M. Reimann. A simple augmented ϵ-constraint method for multi-objective mathematical integer programming problems. *European Journal of Operational Research*, 234(1):15–24, 2014.