8-2017

Exact Algorithms for Mixed-Integer Multilevel Programming Problems

Leonardo Lozano
Clemson University, llozano@g.clemson.edu

Exact Algorithms for Mixed-Integer Multilevel Programming Problems

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
Leonardo Lozano
August 2017

Abstract

We examine multistage optimization problems, in which one or more decision makers solve a sequence of interdependent optimization problems. In each stage the corresponding decision maker determines values for a set of variables, which in turn parameterizes the subsequent problem by modifying its constraints and objective function. The optimization literature has covered multistage optimization problems in the form of bilevel programs, interdiction problems, robust optimization, and two-stage stochastic programming. One of the main differences among these research areas lies in the relationship between the decision makers. We analyze the case in which the decision makers are self-interested agents seeking to optimize their own objective function (bilevel programming), the case in which the decision makers are opponents working against each other, playing a zero-sum game (interdiction), and the case in which the decision makers are cooperative agents working towards a common goal (two-stage stochastic programming). Traditional exact approaches for solving multistage optimization problems often rely on strong duality either for the purpose of achieving single-level reformulations of the original multistage problems, or for the development of cutting-plane approaches similar to Benders' decomposition. As a result, existing solution approaches usually assume that the last-stage problems are linear or convex, and fail to solve problems for which the last-stage is nonconvex (e.g., because of the presence of discrete variables). We contribute exact finite algorithms for bilevel mixed-integer programs, three-stage defender-attacker-defender problems, and two-stage stochastic programs. Moreover, we do not assume linearity or convexity for the last-stage problem and allow the existence of discrete variables. We demonstrate how our proposed algorithms significantly outperform existing state-of-the-art algorithms. Additionally, we solve for the first time a class of interdiction and fortification problems in which the third-stage problem is $\mathcal{NP}$-hard, opening a venue for new research and applications in the field of (network) interdiction.

Acknowledgments

Professor J. Cole Smith for his advice and mentorship, but above all, for his friendship and for being an outstanding role model. Professor Andrés Medaglia for his unconditional support and encouragement. My advisory committee, professors Warren Adams, Mary E. Kurz, and Amin Khademi. Cindy, Katie, Dani, and Claire for taking me as one of their own throughout these years away from home. My wife and my family for cheering me up every single time.

I am grateful to Dr. Paola Scaparra at University of Kent (UK) and Dr. Paola Cappanera at University of Florence (Italy) for their generosity in sharing with us their code for the computational experiments in Chapter 3. I am also grateful to Dr. Illya Hicks at Rice University for sharing with us the compiler instances for our computational experiments in Chapter 5.

Table of Contents

List of Tables

List of Figures

Chapter 1

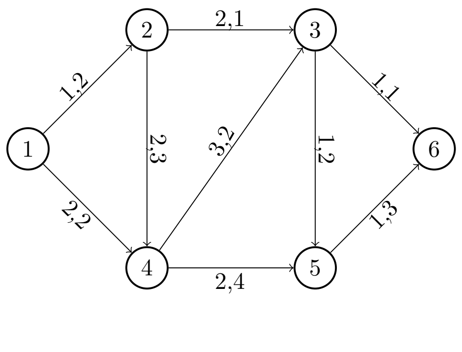
Introduction

The optimization literature has explored multilevel optimization in the form of bilevel programming, interdiction problems, robust optimization, and two-stage stochastic programming. In a multistage setting two decision makers solve a sequence of interdependent optimization problems. Typically, in the first stage an *upper-level* decision maker (*leader*) determines values for its set of variables, while in the second stage a *lower-level* decision maker (*follower*) solves an optimization problem that is parameterized by the leader's first-stage decisions. Bilevel programming deals with the case in which the leader and the follower are self-interested agents seeking to optimize their own objective function. Interdiction and robust optimization focus on the setting in which the agents are adversaries engaged in a zero-sum game. Two-stage stochastic programming studies the case in which the leader and the follower are cooperative agents (or, more frequently when they are the same agent operating at different time stages) who are optimizing an aligned objective function.

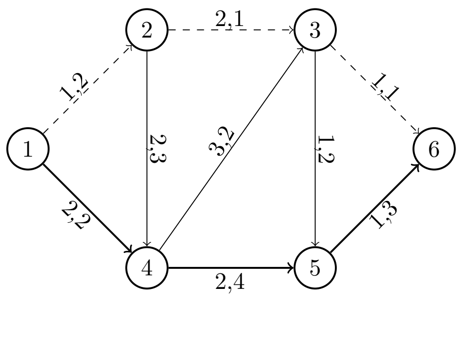
1.1 Background and Contribution

This dissertation presents novel algorithmic approaches to solve three challenging general classes of multistage optimization problems, as well as featured case studies and applications. The first class of problems we study are bilevel programs. In this setting, the leader's first-stage decisions affect the follower's second-stage feasible region and objective, and vice versa. In the first stage the leader selects values for the upper-level variables anticipating that the follower will react in the second stage by solving the lower-level problem to optimality. The follower response could deteriorate the leader's objective or even render the leader's solution infeasible. As a result, the leader's first-stage decisions should impel the follower to respond in a way that is favorable to the leader.

Figure 1.1 illustrates this bilevel setting with a variation of the traditional shortest-path problem. Both the leader and the follower seek a minimum-cost path from node 1 to node 6. The cost of traversing each arc may be different for each agent. Leader and follower costs are displayed alongside the arcs, respectively, in Figure 1.1. If an arc is traversed by both agents, then its cost is increased by 3 units (e.g., due to the effect of congestion). Otherwise, the cost of the arc remains unchanged. For example, if both the leader and the follower traverse arc $(4, 5)$, then the leader incurs a cost of 5 $(2 + 3)$ and the follower incurs a cost of 7 $(4 + 3)$. If only the leader (follower) traverses arc $(4, 5)$, then the cost of using the arc is 2 (4). Figure 1.1b presents an optimal solution to the bilevel shortest-path problem described. Bold arcs represent the leader's first-stage decisions and dashed arcs represent the follower's second-stage response. Note that under this setting an optimal solution is defined as a first-stage solution that maximizes the leader's objective, which is computed using the corresponding follower's optimal response. The leader selects path 1–4–5–6 and



(a)



(b)

Figure 1.1: Bilevel shortest-path problem from node 1 to 6. Leader and follower costs displayed alongside the arcs respectively. Bold arcs represent an optimal leader's path and dashed arcs represent an optimal corresponding follower's response.

the follower selects path 1–2–3–6. The optimal objective function value for the leader is 5 and the optimal follower's objective is 4, since the agents do not use any common arcs. Note that leader's path 1–2–3–6 yields a cost of 4 units and is the optimal solution to the single level shortest-path problem (ignoring the delays caused by the follower's decisions). However, if the leader selects path 1–2–3–6 in the first stage, then the follower response in the second stage would be path 1–4–3–6 and the corresponding leader's objective function value would be 7 $(4 + 3)$ because both agents would use arc $(3, 6)$. On the other hand, the path that the leader selects induces a follower response that avoids using common arcs, yielding the optimal objective value of 5.

The second class of problems we study are interdiction problems. In this setting, the leader (or attacker) and the follower (or defender) play a zero-sum Stackelberg game, also known as an attacker-defender game. The leader selects a set of actions pursuing the sole objective of worsening the follower's objective function value. For instance, if the follower attempts to minimize an objective function, then the leader seeks to maximize the minimum achievable follower's objective. Typically, the leader's actions from the first stage (attack) affect the follower's feasible region and objective. In this context, an attack is not necessarily due to a malicious adversary, but could alternatively represent some bounded worst-case scenario on a system's uncertain failures. We also analyze a three stage version of these games in which the defender is able to protect some assets beforehand, and then the attacker is not allowed to attack protected assets while playing the attacker-defender game described above. These games are known as defender-attacker-defender games.

We illustrate this attacker-defender setting with a variation of the bilevel shortest-path problem described above. As before, both the leader and the follower seek a path from node 1 to node 6. In contrast to the bilevel setting, the leader's objective is now to maximize the follower's minimum-cost path. As before, if an arc is traversed by both agents, then its cost is increased by 3 units. Follower arc costs are displayed alongside the arcs in Figure 1.2. Figure 1.2b depicts an optimal solution to the attacker-defender shortest-path problem described. The leader selects path

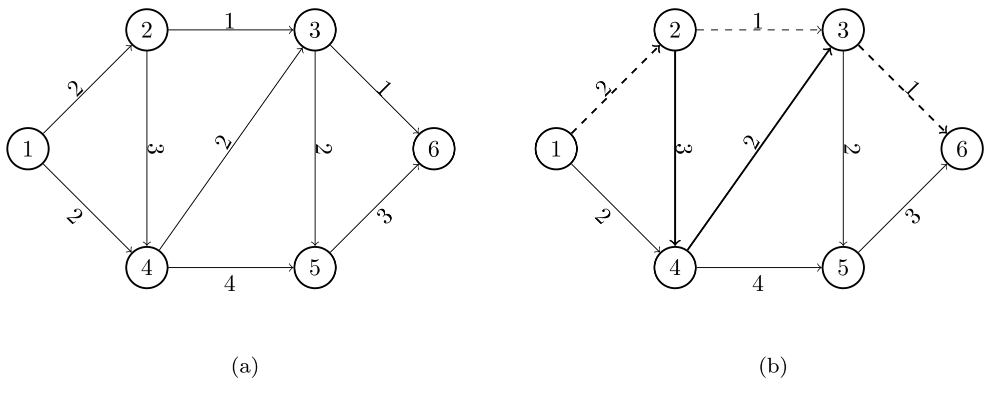


Figure 1.2: Shortest path interdiction problem from node 1 to 6. Follower costs displayed alongside the arcs. Bold arcs represent an optimal leader path and dashed arcs represent an optimal corresponding follower response.

1–2–4–3–6 and the follower responds with path 1–2–3–6, yielding a follower's objective function value of 10 $(4+6)$. Note that the follower's minimum-cost path has a cost of 4 (ignoring the delays caused by the leader's decisions), and the leader is able to increase this minimum cost by 6 units.

The third class of problems we study are two-stage stochastic programming problems. In this setting the leader selects a vector of decisions in the first stage, before the realization of some uncertain parameters. In the second stage (or recourse problem), the follower determines the remaining variable values in response to the first-stage variables and to the realization of the uncertain parameters. The leader and the follower cooperate towards the goal of minimizing the total expected cost.

We modify the bilevel shortest-path problem described above to illustrate this two-stage stochastic programming setting. In this case, the cost of the arcs is uncertain and both the leader and the follower seek a unique path from node 1 to node 6 that minimizes the expected travel cost. In the first stage, the leader selects an initial path considering a deterministic fixed cost (displayed outside the parentheses in Figure 1.3) and two equiprobable variable arc cost scenarios (a low-cost and a high-cost scenario displayed inside the parentheses in Figure 1.3). In the second stage, after the realization of the uncertain variable arc cost, the follower is allowed to reevaluate part of the first-stage path by performing at most one detour. We define a detour from node i to node j as any path i–k–j such that $k \neq i$ and $k \neq j$. For instance, if the leader selects path 1–4–5–6 in the first stage, the follower is allowed to select paths 1–2–4–5–6, 1–4–3–5–6, and 1–4–5–6. The optimal

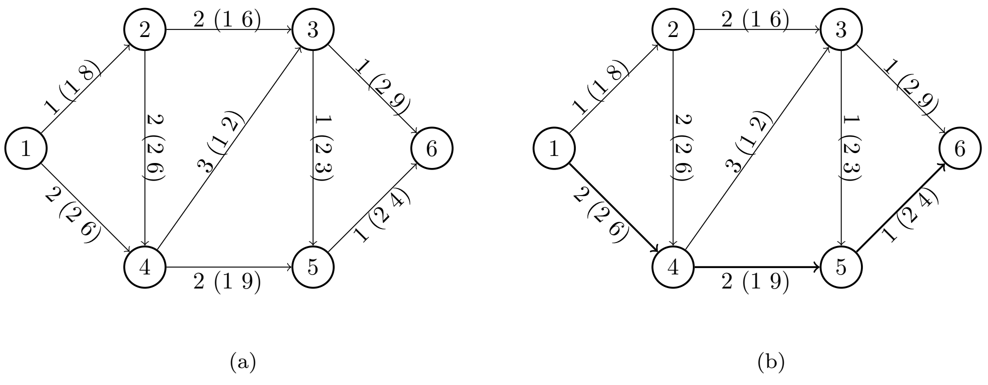


Figure 1.3: Shortest-path problem from node 1 to 6 having arc cost uncertainty. Fixed costs displayed outside the parentheses and two cost scenarios displayed inside the parentheses. Bold arcs represent an optimal first-stage path.

first-stage solution is path 1–4–5–6. Under the low-cost scenario, the follower does not change the first-stage path, yielding a cost of 10 $(4 + 3 + 3)$ units. Under the high-cost scenario, the follower selects path 1–4–3–5–6, yielding a cost of 22 $(8 + 5 + 4 + 5)$ units, in contrast to the 24 units of cost corresponding to path 1–4–5–6 under the high-cost scenario. The total expected travel cost is 16.

Exact solution techniques for multistage optimization problems often rely on strong dual formulations to the second-stage problem. A prevalent approach in the literature reformulates the multistage optimization problem as a single-level problem by appending the lower-level problem Karush-Kuhn-Tucker or Fritz-John optimality conditions to the upper-level problem or by taking the dual of the second-stage problem to combine the first- and second-stage problems. The resulting combined single-level problem is often a bilinear program, which is usually transformed into a mixed-integer program by means of linearization techniques.

Another approach to solve multistage optimization problems is based on cutting-plane algorithms related to Benders' decomposition. The main idea of Benders' decomposition approaches is to formulate an equivalent single-level master problem, which is a projection of the original problem onto the space of the first-stage variables. This problem is then solved by a cutting-plane algorithm, which derives cuts from subproblems that are obtained by fixing the first-stage variables. Benders' decomposition algorithms have proven to be specially successful for problems in which the second stage is a linear program. The special structure of the value function of a linear program, which is convex and piecewise linear, facilitates its approximation by means of cuts derived from the dual of the second-stage problem.

Since existing exact solution approaches for multistage optimization problems from the literature usually rely on strong duality, they often require the second-stage problem to be linear or convex. These approaches usually fail to solve multistage optimization problems for which the second-stage problem is nonconvex (e.g., due to the presence of discrete decision variables) because of the lack of polynomial-sized strong dual formulations. We contribute new exact algorithms for three classes of challenging multistage optimization problems, in which we do not assume that the second-stage problem is linear or convex and allow the existence of discrete variables in both the first- and second-stage problems. We demonstrate that, in some cases, our approach not only solves more general classes of multistage optimization problems than previous approaches from the literature but also outperforms traditional approaches on multistage problems for which the second-stage problem is convex.

1.2 Literature Review

We now present developments in bilevel programming, interdiction, and integer stochastic programs. Bilevel programming has multiple applications in areas such as traffic systems [Brotcorne et al., 2001, Dempe and Zemkoho, 2012, Labbé et al., 1998, Migdalas, 1995], resource allocation [Xu et al., 2012], natural gas market regulation [Dempe et al., 2005, 2011, Kalashnikov et al., 2010], waste management [Xu and Wei, 2012], bioengineering [Burgard et al., 2003], and chemical engineering [Bollas et al., 2009, Mitsos et al., 2009a,b]. Two problems in bilevel mixed-integer programming include a nonlinear procurement problem [Prince et al., 2013a] and a competitive product introduction game [Hemmati and Smith, 2016]. Prince et al. [2013a] examine the case in which two firms seek to purchase, at minimum cost, a product from various capacitated suppliers. The lower-level problem is converted to a linear program, whose optimality conditions are represented using constraints that enforce strong duality. This paper is notable in that one version of the problem allows the leader to suboptimize its decisions by some parameter δ in order to maximize the minimum follower objective. The competitive set covering problem of Hemmati and Smith [2016] considers a linear bilevel mixed-integer program, which is inspired by product introduction games. Their approach to solving the problem is based on an exponential-size reformulation of the problem that is amenable to solution via a cutting-plane algorithm.

Existing algorithms in the literature focus primarily on continuous bilevel programs. Bard [1983], Candler and Townsley [1982], and Tuy et al. [1993] propose algorithms based on extreme point enumeration. A single-level reformulation of the problem, obtained by appending the lower-level problem Karush-Kuhn-Tucker or Fritz-John optimality conditions to the upper-level problem is studied by Dempe and Zemkoho [2013, 2014], Dempe and Franke [2015], Hansen et al. [1992], Shi et al. [2005], and Shi et al. [2006]. Tsoukalas et al. [2009] develop an exact algorithm for nonconvex bilevel problems. The algorithm performs a binary search over the leader's objective values using an "oracle" that decides whether a target objective value is achievable or not. Mitsos et al. [2008] propose an optimal-value-function reformulation for solving nonlinear bilevel problems having nonconvex functions in both the upper and lower level. Dempe et al. [2007] and Dempe and Pilecka [2015] study necessary optimality conditions for the optimistic formulation, while Dempe et al. [2014] study conditions for the pessimistic formulation. Wiesemann et al. [2013] study the computational complexity of pessimistic bilevel problems and devise an ϵ-approximation.

For bilevel mixed-integer linear programs (BMILPs) under the optimistic assumption, Bard and Moore [1992] propose a binary search procedure that fixes the leader variables to uncover feasible solutions while iteratively improving a bound on the objective function value. They assume that all variables are binary and that there are no constraints in the upper-level problem. Moore and Bard [1990] propose a basic implicit enumeration scheme that solves small problems having up to 40 variables, assuming that the lower-level variables do not appear in the upper-level constraints. DeNegre and Ralphs [2009] extend this idea and propose a branch-and-cut algorithm for the case in which all variables are integer. They report computational experiments on a set of interdiction problems (with sizes ranging from 20 to 34 variables) in which the lower-level problem is a binary knapsack problem. Saharidis and Ierapetritou [2009] propose a Benders-decomposition-based algorithm. Their approach iteratively solves a subproblem (SP) obtained by fixing the integer variables to a feasible value and a restricted master problem (RMP), which is a relaxation of the original bilevel problem. In each iteration of the algorithm, the SP generates a cut for the RMP, which converges to an optimal solution assuming that the leader has control over all the integer variables in both the upper- and lower-level problems. Mitsos [2010] solves nonconvex bilevel problems using an optimal-value-function reformulation to obtain a sequence of nondecreasing lower bounds, which converge to the optimal objective value. Xu and Wang [2014] propose a branch-and-bound algorithm that solves a series of mixed-integer linear programs, branching on the contribution of the first-stage variables to the lower-level constraints to generate multiple branches at each node. They assume that all leader variables are integer-valued and that the contribution of the first-stage variables to the lower-level constraints is integer valued. Their work is also notable for reporting a comprehensive computational study, on instances with sizes ranging from 20 to 920 variables.

Another line of research in BMILPs is based on parametric programming, which attempts to represent the follower's optimal variable values as a function of the leader's variables. Algorithms in this area are proposed by Domínguez and Pistikopoulos [2010] for the (mixed-)integer case and by Köppe et al. [2010] for a linear case in which the leader variables are continuous and the follower variables are integer. These works adopt the optimistic assumption and present computational results over small-sized example problems. Faisca et al. [2007] propose a parametric programming approach that transforms the original bilevel problem into a set of quadratic, linear, or mixed-integer linear programming problems.

Fanghänel and Dempe [2009] study optimality conditions for bilevel programs with continuous upper-level and discrete lower-level problems. See also the work of Dempe et al. [2015] for a recent survey on bilevel programming.

Special cases of bilevel programs include interdiction problems, in which the leader seeks to minimize the follower's objective, and robust optimization problems in which the follower seeks to minimize the leader's objective [Wood, 1993]. Interdiction problems have multiple applications in areas such as military and homeland security operations [Brown et al., 2005a, 2006, 2009, Morton et al., 2007, Pan et al., 2003, Washburn and Wood, 1995], facility protection [Church and Scaparra, 2007, Church et al., 2004, Scaparra and Church, 2008a,b], survivable network design [Smith et al., 2007], and power grid protection [Salmerón et al., 2004, 2009]. At a more abstract level, interdiction problems can often be modeled as games that take place over networks having well-studied recourse problems. Some of these network interdiction problems include shortest path [Bayrak and Bailey, 2008, Cappanera and Scaparra, 2011, Fulkerson and Harding, 1977, Golden, 1978, Held and Woodruff, 2005, Held et al., 2005, Israeli and Wood, 2002], maximum flow [Cormican et al., 1998, Royset and Wood, 2007, Wollmer, 1964, Wood, 1993], and multicommodity flow [Lim and Smith, 2007] studies.

Of particular interest in this dissertation are previous studies on defender-attacker-defender problems that fit within the problem framework studied in Chapter 3. Church and Scaparra [2007] consider fortification decisions for the interdiction median problem with fortification (IMF), which arises in the context of facility protection. They reformulate the three-level problem into a single-level mixed-integer programming problem (MIP) by explicitly enumerating all possible attack plans. If the number of attack plans is not too large, then the resulting MIP can be solved via commercial branch-and-bound software. Scaparra and Church [2008b] extend this idea by reformulating the problem as a single-level maximal covering problem with precedence constraints. They propose a heuristic algorithm for finding upper and lower bounds, which they use to reduce the size of the original model. Scaparra and Church [2008a] formulate the IMF as a bilevel programming problem and solve it with a specialized implicit enumeration algorithm that efficiently solves the lower-level interdiction problem. This approach was extended by Cappanera and Scaparra [2011] for the allocation of protective resources in a shortest-path network.

Another line of research in defender-attacker-defender problems focuses on duality as a mechanism for formulating interdiction problems. Brown et al. [2006] study the problem of protecting critical components in an electric power grid. Their approach combines the second- and third-stage problems by taking the dual of the third-stage problem, and solves the resulting problem using a special Benders' decomposition algorithm in which the subproblem is an MIP. Smith et al. [2007] take a similar approach for the survivable network design problem. They rely on the dual of the third-stage problem to combine the second- and third-stage problems into a disjointly constrained bilinear program, which is then transformed into a MIP by applying a standard linearization technique. The resulting bilevel problem is solved with a cutting-plane approach. Prince et al. [2013b] followed these ideas for a three-stage procurement optimization problem under uncertainty. They transform the third-stage (nonconvex) procurement problem into a large-scale shortest path problem, which can then be solved by the foregoing strategies. Because the MIP is too large to solve using standard approaches, the authors propose a scaling approach to quickly obtain optimal MIP solutions. For a comprehensive literature review on interdiction problems, see [Brown et al., 2005b, Smith, 2010, Smith and Lim, 2008].

The Prince et al. [2013b] study is notable in that its recourse problem is nonconvex. The authors obviate this nonconvexity by formulating an equivalent linear programming model that is pseudopolynomial in size. There are relatively few studies that regard interdiction problems having more general nonconvex recourse problems. One example of such study is considered by Yen et al. [2014], who provide an exact approach for solving two-stage interdiction problems having mixed-integer recourse variables and (general) integer interdiction variables. Their approach is based on the dualization of a convex restriction of the recourse problem, which is iteratively enlarged as their algorithm converges to an optimal solution. Their approach is capable of solving relatively modest-sized problems to optimality.

Regarding stochastic programs having integer variables, an important line of research focuses on extending L-shaped or Benders' methods, which are among the most successful algorithms for stochastic linear programs. Laporte and Louveaux [1993] propose a cutting-plane approach for problems having binary first-stage variables. They solve the recourse problem by branch-and-bound and derive first-stage cuts from the objective function values of the recourse problems. These cuts prevent the algorithm from exploring the same first-stage solution twice, thus guaranteeing finite termination. Laporte et al. [2002] implement this method for the capacitated vehicle routing problem with stochastic demands and solve instances having up to 100 customers. This approach is generalized by Carøe and Tind [1998] for mixed-integer first-stage variables. They propose a Benders' decomposition algorithm based on general duality theory. Their approach derives feasibility and optimality cuts from dual variables, which are obtained at the termination of a cutting-plane or branch-and-bound algorithm used for solving the recourse problem.

Sherali and Fraticelli [2002] propose a Benders' decomposition method for problems in which first- and second-stage variables are binary. Their algorithm generates partial descriptions of the convex hull of the recourse problem based on the Reformulation-Linearization Technique (RLT) of Sherali and Adams [1999]. Sen and Higle [2005] use an alternative convexification process based on disjunctive programming (see also related work in [Ntaimo and Sen, 2005, Sen and Sherali, 2006]). Sherali and Smith [2009] consider stochastic programs involving binary first-stage variables and second-stage variables that include both continuous decision variables and binary risk variables. The risk variables are specially structured so that the second-stage problem can be convexified using the RLT.

Gade et al. [2014] propose a Benders' decomposition algorithm for problems in which the first-stage variables are binary and the second-stage variables are general integers. Their decomposition algorithm approximates the recourse problem using Gomory cuts, which are parametrized by the first-stage decisions. The resulting master problem is a linear integer program. Their work is notable for providing an approximation of the expected recourse function by piecewise linear functions of the first-stage variables.

Another line of research in stochastic integer programs focuses on branch-and-bound algorithms. Carøe and Schultz [1999] present a branch-and-bound algorithm in which a Lagrangian dual is solved at each node of the tree. They reformulate the problem by introducing copies of the first-stage variables and obtain a Lagrangian relaxation with respect to non-anticipativity constraints, which require the copies of the first-stage variables to be equal. Finite termination is guaranteed if the first-stage variables are integer, or for an ϵ-optimal termination condition. Ahmed et al. [2004] propose an algorithm for two-stage stochastic integer programs that exploits structural properties of the recourse problem value function. Their scheme develops a branching strategy that guarantees finite termination even if the first-stage problem allows continuous variables.

Other approaches to solve stochastic integer programs include enumeration algorithms [Schultz et al., 1998], convex approximations [Haneveld et al., 2006, van der Vlerk, 2004], branch-and-fix [Alonso-Ayuso et al., 2003, Escudero et al., 2010], and branch-and-price [Lulli and Sen, 2004], among others. We refer the reader to [Haneveld and van der Vlerk, 1999, Schultz, 2003, Sen, 2005] for surveys on stochastic mixed-integer programming.

Because BDDs are relevant to our approach described in Chapter 5, we briefly cover some of the related developments in this area of research. BDDs were first introduced as graphical representations of a Boolean function in the context of circuit design and formal verification [Akers, 1978, Lee, 1959]. From an optimization perspective BDDs have been used in the generation of valid inequalities for integer programs [Becker et al., 2005], vertex and facet enumeration for combinatorial problems [Behle and Eisenbrand, 2007], and a general branch-and-bound solver for discrete optimization problems [Bergman et al., 2016]. A generalization of BDDs known as multivalued decision diagrams is used for solving sequencing problems [Cire and van Hoeve, 2013] and multidimensional bin packing [Kell and van Hoeve, 2013]. The relationship between BDDs and dynamic programming is explored in [Hooker, 2013].

1.3 Organization of the Dissertation

Chapter 2 examines bilevel mixed-integer programs whose constraints and objective functions depend on both upper- and lower-level variables. The class of problems we consider allows for nonlinear terms to appear in both the constraints and the objective functions, requires all upper-level variables to be integer, and allows a subset of the lower-level variables to be integer. This class of bilevel problems is difficult to solve because the upper-level feasible region is defined in part by optimality conditions governing the lower-level variables, which are difficult to characterize because of the nonconvexity of the follower problem. We contribute an exact finite algorithm for these problems based on an optimal-value-function reformulation. We demonstrate how this algorithm can be tailored to accommodate either optimistic or pessimistic assumptions on the follower behavior. Computational experiments demonstrate that our approach outperforms a state-of-the-art algorithm for solving bilevel mixed-integer linear programs.

Chapter 3 examines a class of three-stage sequential defender-attacker-defender problems. In these problems the defender first selects a subset of assets to protect, the attacker next damages a subset of unprotected assets in the interdiction stage, after which the defender optimizes a recourse problem over the surviving assets. These problems are notoriously difficult to optimize, and almost always require the recourse problem to be a convex optimization problem. Our contribution is a new approach to solving defender-attacker-defender problems. We require all variables in the first two stages to be binary-valued, but allow the recourse problem to take any form. The proposed framework focuses on solving the interdiction problem by restricting the defender to select a recourse decision from a sample of feasible vectors. The algorithm then iteratively refines the sample to force finite convergence to an optimal solution. We demonstrate that our algorithm not only solves interdiction problems involving NP-hard recourse problems within reasonable computational limits, but it also solves shortest path fortification and interdiction problems more efficiently than state-of-the-art algorithms tailored for that problem, finding optimal solutions to real-road networks having up to 300,000 nodes and over 1,000,000 arcs.

Chapter 4 solves a defender-attacker-defender problem over a traveling salesman problem (TSP), which is a well-known $\mathcal{NP}$-hard problem that seeks a minimum-cost Hamiltonian cycle (tour) over a graph [Flood, 1956, Lawler et al., 1985]. We present the traveling salesman problem with interdiction and fortification (TSPIF). In the first stage (*fortification*), the defender fortifies a subset of arcs. In the second stage (*attack*), an attacker interdicts a subset of unprotected arcs, thus increasing their cost. In the third stage (*recourse*), the defender solves a TSP defined using the costs resulting from the attack stage. Apart from defense applications, the TSPIF arises as an alternative conservative approach to modeling routing problems under uncertainty, in which the road travel times may not be known in advance due to congestion effects [Pillac et al., 2013]. Our proposed approach employs the exact approach proposed in Chapter 3 augmented with a TSP restriction phase to accelerate the convergence of the algorithm. Our computational results show success for the first time in optimally solving defender-attacker-defender TSP problems.

Chapter 5 considers a class of two-stage stochastic integer programming problems with binary variables appearing in both stages. The special class of problems has a set-covering structure in the second stage, where both first- and second-stage variables can be used to satisfy those constraints. Our approach seeks to uncover strong dual formulations to the second-stage problems by transforming them into dynamic programming (DP) problems parameterized by first-stage variables. We demonstrate how these DPs can be formed by use of binary decision diagrams (BDDs), which are layered directed acyclic graphs in which arcs correspond to assigning values to binary variables. Using BDDs we reformulate the second-stage problem as a shortest-path problem in which arcs availabilities are given as a function of the first-stage variables. This representation allows us to parameterize our optimal DP solutions as a function of the first-stage variables, which then yield traditional Benders inequalities that can be strengthened based on observations regarding the structure of the resulting BDDs. Moreover, we limit the size of the resulting BDDs by employing

concepts related to the minimization of branchwidth on hypergraphs. We demonstrate the efficacy of our approach on a set of stochastic vertex cover problems.

Chapter 2

A Value-Function-Based Exact Approach for the Bilevel Mixed-Integer Programming Problem

2.1 Problem Statement

We study a class of problems known as bilevel mixed-integer programs (BMIPs). These problems are modeled as two-level, two-player Stackelberg games, in which two decision makers sequentially solve interdependent problems that optimize different objective functions. In the first stage an *upper-level* decision maker (*leader*) determines values for its set of variables, while in the second stage a *lower-level* decision maker (*follower*) solves an optimization problem that is parametrized by the leader decisions. Formally, let $\mathbf{x} \in \mathcal{H}^x$ be an n_1-dimensional vector of variables controlled by the leader and $\mathbf{y} \in \mathcal{H}^y$ be an n_2-dimensional vector of variables controlled by the follower, where host set $\mathcal{H}^x = \{\mathbf{x} \mid \mathbf{x} \geq \mathbf{0};\ x_i \in \mathbb{Z},\ \forall i \in \mathcal{I} \subseteq \{1, \ldots, n_1\}\}$ and $\mathcal{H}^y = \{\mathbf{y} \mid \mathbf{y} \geq \mathbf{0};\ y_j \in \mathbb{Z},\ \forall j \in \mathcal{J} \subseteq \{1, \ldots, n_2\}\}$. Let ϕ^l, ϕ^f, g_j^k, and h_j^k be continuous (and possibly nonconvex) functions defined over $\mathcal{H}^x \times \mathcal{H}^y$, for $k = 1, 2$ and $j = 1, \ldots, m_k$. The BMIP can be formally stated as:

$$z^* = \max_{\mathbf{x},\mathbf{y}} \ \phi^l(\mathbf{x}, \mathbf{y}) \tag{2.1a}$$

$$\text{s.t.} \ g_j^1(\mathbf{x}) + h_j^1(\mathbf{y}) \leq b_j^1 \qquad \forall j = 1, \ldots, m_1 \tag{2.1b}$$

$$\mathbf{y} \in \underset{\mathbf{y}^f}{\operatorname{argmax}} \{\phi^f(\mathbf{x}, \mathbf{y}^f) \,|\, g_j^2(\mathbf{x}) + h_j^2(\mathbf{y}^f) \leq b_j^2, \forall j = 1, \ldots, m_2; \, \mathbf{y}^f \in \mathcal{H}^y\} \tag{2.1c}$$

$$\mathbf{x} \in \mathcal{H}^x, \tag{2.1d}$$

where dummy variables $\mathbf{y}^f$ replace $\mathbf{y}$ in the lower-level problem. Note that if the lower-level problem has alternative optimal solutions, then the follower will select a $\hat{\mathbf{y}}$ that maximizes ϕ^l, thus benefiting the leader. This is known as the *optimistic* formulation of the problem. We also consider a *pessimistic* formulation in Section 2.4, in which the follower seeks to worsen the leader's objective among all alternative optimal solutions to its own problem.

Our proposed approach requires the following assumptions:

- Assumption 1: Both the upper- and lower-level feasible regions are compact sets. This assumption, in conjunction with the continuity of ϕ^l and ϕ^f, ensures the existence of global optimal solutions for all optimization problems solved by our approach.

- Assumption 2: $g_j^2(\mathbf{x})$ is integer-valued for all $\mathbf{x} \in \mathcal{H}^x$, $j = 1, \ldots, m_2$. This assumption guarantees the exactness of our algorithm. Remark 1 in Section 2.2.2 discusses the implications of this assumption in more detail.

- Assumption 3: All leader variables are integer-valued. This assumption guarantees the finite termination of our algorithm as demonstrated by Proposition 4 in Section 2.2.3.

Additionally, our approach necessitates the repeated solution of subproblems stemming from problem (2.1), which we describe in Section 2.2.2. Because the algorithm proposed in this chapter is general, we do not prescribe tailored methods for solving individual classes of subproblems that might arise. We instead assume that those subproblems are solved by appropriate algorithms available in the literature. Of course, certain classes of nonlinear programs resist solution in practice by any known algorithm, and therefore an implicit assumption must also be made that the subproblems stemming from (2.1) are practically solvable.

Our proposed algorithm relies on establishing a partial enumeration of follower solutions.

Unlike the case of interdiction problems, though, restricting the follower to select from a set of sampled solutions does not result in an upper bound on the leader's problem. Therefore, one of the primary challenges we address regards the development of lower- and upper-bounding mechanisms based on a sampling scheme for the follower. We then contribute variable fixing and inequality generation schemes that accelerate the convergence of our algorithm, which we show runs up to 17 times faster than a state-of-the-art approach for BMILPs proposed by Xu and Wang [2014] over test instances from the literature. Finally, we contribute a modification to our approach that accommodates the so-called pessimistic assumption for BMIPs, and illustrate on a competitive scheduling problem why implicitly treating the objective as a nonlinear function is important in obtaining good computational results.

The remainder of this chapter is organized as follows. Section 2.2 presents the sampling-based algorithm and establishes the finite convergence of our approach to an optimal solution. Section 2.3 devises strengthening strategies for our mathematical formulations. Section 2.4 extends our algorithm to accommodate the pessimistic assumption. Section 2.5 presents a featured study on competitive scheduling. Finally, Section 2.6 presents our computational experiments.

2.2 A Sampling-Based Exact Algorithm

The proposed algorithm employs a BMIP relaxation that considers disjunctive constraints, which are generated from a subset or *sample* of feasible follower responses. Our algorithm iteratively solves this relaxation to obtain an upper bound on the BMIP; uncovers bilevel feasible solutions to obtain lower bounds; and enlarges the current sample to potentially obtain tighter upper bounds at subsequent iterations. The algorithm stops once it proves global optimality of the current incumbent solution. Section 2.2.1 presents relevant definitions and notation. Section 2.2.2 describes the proposed BMIP relaxation, while Section 2.2.3 presents our algorithm and proves its finite convergence. Section 2.2.4 discusses our sampling approach. Finally, Section 2.2.5 discusses the case in which the objectives and constraints are linear.

2.2.1 Definitions and Notation

Let

$$\mathcal{X}(\mathbf{y}) = \{\mathbf{x} \mid g^1_j(\mathbf{x}) \le b^1_j - h^1_j(\mathbf{y}),\ \forall j = 1,\ldots,m_1;\ \mathbf{x} \in \mathcal{H}^x\} \tag{2.2}$$

be the region defined by the leader constraints for any fixed follower decision vector $\mathbf{y}$ and

$$\mathcal{Y}(\mathbf{x}) = \{\mathbf{y} \mid h^2_j(\mathbf{y}) \le b^2_j - g^2_j(\mathbf{x}),\ \forall j = 1,\ldots,m_2;\ \mathbf{y} \in \mathcal{H}^y\} \tag{2.3}$$

be the region defined by the follower constraints for a fixed leader decision vector $\mathbf{x}$. Define

$$\Omega = \{(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in \mathcal{X}(\mathbf{y}),\ \mathbf{y} \in \mathcal{Y}(\mathbf{x})\} \tag{2.4}$$

as the region obtained by relaxing the optimality requirement for the follower variables in formulation (2.1), and define

$$\Omega(\mathcal{X}) = \{\mathbf{x} \mid \exists \mathbf{y} \text{ such that } (\mathbf{x},\mathbf{y}) \in \Omega\} \tag{2.5}$$

as the projection of Ω onto the leader decision space. Next, let

$$\Psi(\mathbf{x}) = \operatorname{argmax}\{\phi^f(\mathbf{x},\mathbf{y}) \mid \mathbf{y} \in \mathcal{Y}(\mathbf{x})\} \tag{2.6}$$

be the follower *rational reaction set* for each leader solution $\mathbf{x}$, i.e., the set of all follower solutions that satisfy (2.1c) for a given value of $\mathbf{x}$. Finally, define

$$\mathcal{Y} = \bigcup_{\mathbf{x} \in \Omega(\mathcal{X})} \mathcal{Y}(\mathbf{x}) \tag{2.7}$$

as the set of all feasible follower responses.

An $(\mathbf{x},\mathbf{y})$-solution is called *bilevel feasible* if $\mathbf{x} \in \mathcal{X}(\mathbf{y})$ and $\mathbf{y} \in \Psi(\mathbf{x})$. The BMIP can now be restated as

$$z^* = \max_{(\mathbf{x},\mathbf{y})}\{\phi^l(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in \mathcal{X}(\mathbf{y}),\ \mathbf{y} \in \Psi(\mathbf{x})\}. \tag{2.8}$$

Note that if $\Psi(\mathbf{x})$ is not a singleton, then the follower will select a $\hat{\mathbf{y}} \in \Psi(\mathbf{x})$ that maximizes $\phi^l(\mathbf{x}, \mathbf{y})$, because of our so-called optimistic assumption that the follower breaks ties in favor of the leader. We extend our algorithm for a pessimistic formulation of the problem in Section 2.4.

The single-level optimization problem obtained by relaxing the optimality requirement (2.1c) for the follower variables is called the *high point problem* (HPP). This problem is stated as

$$z^{\text{HPP}} = \max_{(\mathbf{x},\mathbf{y}) \in \Omega} \{\phi^l(\mathbf{x}, \mathbf{y})\}. \tag{2.9}$$

An optimal solution to the HPP yields a valid upper bound on z^*.

2.2.2 Solving the BMIP

We propose a single-level optimal-value-function reformulation for the BMIP [Ye and Zhu, 1995, Ye, 2006, Mitsos et al., 2008]. This formulation is based on the following result, which is adapted from Lemma 2.1 in Ye [2006] for the problem we consider in this chapter.

Proposition 1. *A solution $(\mathbf{x}, \mathbf{y}) \in \Omega$ is bilevel feasible if and only if $\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}})$ for every $\hat{\mathbf{y}} \in \mathcal{Y}(\mathbf{x})$.*

Proof Assume that $(\mathbf{x}, \mathbf{y})$ is bilevel feasible and suppose by contradiction that there exists a follower solution $\hat{\mathbf{y}} \in \mathcal{Y}(\mathbf{x})$ such that $\phi^f(\mathbf{x}, \mathbf{y}) < \phi^f(\mathbf{x}, \hat{\mathbf{y}})$. Then $\mathbf{y} \notin \Psi(\mathbf{x})$, which contradicts the assumption that $(\mathbf{x}, \mathbf{y})$ is bilevel feasible. Now assume that $\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}})$ for all $\hat{\mathbf{y}} \in \mathcal{Y}(\mathbf{x})$. This implies that $\mathbf{y} \in \Psi(\mathbf{x})$. Moreover, $\mathbf{x} \in \mathcal{X}(\mathbf{y})$ since $(\mathbf{x}, \mathbf{y}) \in \Omega$, and so $(\mathbf{x}, \mathbf{y})$ is bilevel feasible. This completes the proof. ■

This proposition implies that a reformulation may allow the leader to control both the $\mathbf{x}$- and $\mathbf{y}$-variables (as in the HPP), while formulating disjunctive constraints that enforce bilevel feasibility. These constraints require that for every $\hat{\mathbf{y}} \in \mathcal{Y}$, the leader must either select an $(\mathbf{x}, \mathbf{y})$ such that $\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}})$, or *block* follower solution $\hat{\mathbf{y}}$ by selecting an $\mathbf{x}$ such that $\hat{\mathbf{y}} \notin \mathcal{Y}(\mathbf{x})$. Note that if $\hat{\mathbf{y}} \notin \mathcal{Y}(\mathbf{x})$ for some $(\mathbf{x}, \hat{\mathbf{y}})$, then $g_j^2(\mathbf{x}) + h_j^2(\hat{\mathbf{y}}) > b_j^2$ must hold for at least one $j \in \{1, \ldots, m_2\}$, implying that $\hat{\mathbf{y}}$ is blocked by constraint j. The leader must therefore satisfy at least one constraint of the form $g_j^2(\mathbf{x}) > b_j^2 - h_j^2(\hat{\mathbf{y}})$ to block follower solution $\hat{\mathbf{y}}$.

Proposition 2. *Define $\gamma_{\hat{\mathbf{y}}j} = \lfloor b_j^2 - h_j^2(\hat{\mathbf{y}}) \rfloor + 1$ for every $\hat{\mathbf{y}} \in \mathcal{Y}$, $j = 1, \ldots, m_2$. The leader blocks solution $\hat{\mathbf{y}} \in \mathcal{Y}$ by constraint j if and only if $g_j^2(\mathbf{x}) \geq \gamma_{\hat{\mathbf{y}}j}$.*

Proof Note that $g_j^2(\mathbf{x})$ is integer-valued by Assumption 2. If $g_j^2(\mathbf{x}) \geq \gamma_{\hat{\mathbf{y}}j}$, then because $\gamma_{\hat{\mathbf{y}}j} > b_j^2 - h_j^2(\hat{\mathbf{y}})$, solution $(\mathbf{x}, \hat{\mathbf{y}})$ violates follower constraint j. If $g_j^2(\mathbf{x}) < \gamma_{\hat{\mathbf{y}}j}$, then $g_j^2(\mathbf{x}) \leq \gamma_{\hat{\mathbf{y}}j} - 1 = \lfloor b_j^2 - h_j^2(\hat{\mathbf{y}})\rfloor \leq b_j^2 - h_j^2(\hat{\mathbf{y}})$. Thus, $g_j^2(\mathbf{x}) + h_j^2(\hat{\mathbf{y}}) \leq b_j^2$ and solution $(\mathbf{x}, \hat{\mathbf{y}})$ does not violate follower constraint j. This completes the proof. ■

Remark 1. *If we relax Assumption 2, thus allowing $g_j^2(\mathbf{x})$ to take on fractional values, then Proposition 2 could be adjusted by setting $\gamma_{\hat{\mathbf{y}}j} = b_j^2 - h_j^2(\hat{\mathbf{y}})$ and requiring that $g_j^2(\mathbf{x}) > \gamma_{\hat{\mathbf{y}}j}$. However, these strict inequalities can lead to open feasible sets, which further complicates the optimization model. We omit the analysis of this case by imposing Assumption 2. However, we discuss how to address a similar complication that arises in the solution of the pessimistic case in Section 2.4. The strategies for the pessimistic case can in turn be used for problems that do not satisfy Assumption 2.*

Define $\mathcal{B}(\hat{\mathbf{y}}, \mathcal{Y}) = \{(\mathbf{y}', q) \mid \gamma_{\mathbf{y}'q} \geq \gamma_{\hat{\mathbf{y}}q},\ \mathbf{y}' \in \mathcal{Y},\ q = 1, \ldots, m_2\}$. This set represents all ordered pairs $(\mathbf{y}', q)$, such that if $\mathbf{x}$ blocks follower solution $\mathbf{y}' \in \mathcal{Y}$ by constraint q, then $\mathbf{x}$ also blocks solution $\hat{\mathbf{y}}$ by constraint q. Also, define binary variables $w_{\hat{\mathbf{y}}j} = 1$ if constraint j blocks solution $\hat{\mathbf{y}}$; if $w_{\hat{\mathbf{y}}j} = 0$, then constraint j may or may not block $\hat{\mathbf{y}}$. We formulate the Extended High Point Problem (EHPP), where the M-values are large constants whose values we discuss subsequently.

$$\max_{(\mathbf{x},\mathbf{y})} \quad \phi^l(\mathbf{x}, \mathbf{y}) \tag{2.10a}$$

$$\text{s.t.} \quad g_j^2(\mathbf{x}) \geq -M_j^1 + \sum_{\hat{\mathbf{y}} \in \mathcal{Y}} (M_j^1 + \gamma_{\hat{\mathbf{y}}j}) w_{\hat{\mathbf{y}}j} \qquad \forall j = 1, \ldots, m_2 \tag{2.10b}$$

$$\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}}) - M_{\hat{\mathbf{y}}}^2 \sum_{(\mathbf{y}', q) \in \mathcal{B}(\hat{\mathbf{y}}, \mathcal{Y})} w_{\mathbf{y}'q} \qquad \forall \hat{\mathbf{y}} \in \mathcal{Y} \tag{2.10c}$$

$$(\mathbf{x}, \mathbf{y}) \in \Omega \tag{2.10d}$$

$$w_{\hat{\mathbf{y}}j} \in \{0, 1\} \qquad \forall \hat{\mathbf{y}} \in \mathcal{Y},\ j = 1, \ldots, m_2. \tag{2.10e}$$

The objective function (2.10a) maximizes the leader's objective. Constraints (2.10b) and (2.10c) utilize binary variables to enforce the bilevel feasibility condition established in Proposition 1. In particular, constraints (2.10b) ensure that for every follower constraint j, $w_{\hat{\mathbf{y}}j} = 0$ for all $\hat{\mathbf{y}} \in \mathcal{Y}$ such that $g_j^2(\mathbf{x}) < \gamma_{\hat{\mathbf{y}}j}$. In fact, there exists an optimal solution such that $w_{\hat{\mathbf{y}}j} = 1$ for $\hat{\mathbf{y}} \in \underset{\mathbf{y}' \in \mathcal{Y}}{\operatorname{argmax}}\{\gamma_{\mathbf{y}'j} \mid g_j^2(\mathbf{x}) \geq \gamma_{\mathbf{y}'j}\}$; if no such index exists, then $w_{\hat{\mathbf{y}}j} = 0,\ \forall \hat{\mathbf{y}} \in \mathcal{Y}$. Constraints (2.10c) then imply that the leader must select an $(\mathbf{x}, \mathbf{y})$ such that $\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}})$ for every $\hat{\mathbf{y}} \in \mathcal{Y}$, unless $\hat{\mathbf{y}}$ has been blocked. Constraints (2.10d) enforce both the upper- and lower-level constraints, and constraints (2.10e) restrict the $\mathbf{w}$-variables to be binary-valued.

The M-values in EHPP must be defined to be sufficiently large so that that (2.10b) and (2.10c) are valid. It is vital for model tightness and computational precision to define the smallest valid parameter values possible, and so we specify how those parameters should be defined in Section 2.6.5. Practically speaking, if the parameters obtained for the M-values are still large enough to cause numerical instability, then an alternative methodology that does not require the use of these M-values would become necessary.

Proposition 3. *The EHPP is equivalent to the BMIP.*

Proof We first consider a feasible solution $(\mathbf{x}, \mathbf{y}, \mathbf{w})$ to the EHPP, and show that $(\mathbf{x}, \mathbf{y})$ is feasible to the BMIP as formulated in (2.8). Constraints (2.10d) ensure that $\mathbf{x} \in \mathcal{X}(\mathbf{y})$ and $\mathbf{y} \in \mathcal{Y}(\mathbf{x})$. To show that $\mathbf{y} \in \Psi(\mathbf{x})$, note that constraints (2.10b) and (2.10c) enforce the bilevel feasibility condition established in Proposition 1. Thus $(\mathbf{x}, \mathbf{y})$ is feasible to the BMIP.

Now we show that for every solution $(\mathbf{x}, \mathbf{y})$ that is feasible to the BMIP, there exists a $\mathbf{w}$ such that $(\mathbf{x}, \mathbf{y}, \mathbf{w})$ is feasible to the EHPP. For each $j = 1, \ldots, m_2$, identify a solution

$$\hat{\mathbf{y}}_j \in \underset{\mathbf{y}' \in \mathcal{Y}}{\operatorname{argmax}} \{\gamma_{\mathbf{y}'j} \mid g_j^2(\mathbf{x}) \geq \gamma_{\mathbf{y}'j}\}.$$

Define $\mathbf{w}$ by setting

$$w_{\hat{\mathbf{y}}_j j} = 1 \text{ and } w_{\hat{\mathbf{y}} j} = 0, \ \forall \hat{\mathbf{y}} \in \mathcal{Y} \setminus \{\hat{\mathbf{y}}_j\}. \tag{2.11}$$

Clearly, this solution satisfies (2.10b), (2.10d), and (2.10e). Assume by contradiction that $(\mathbf{x}, \mathbf{y}, \mathbf{w})$ violates at least one constraint (2.10c). Then, there exists a $\hat{\mathbf{y}} \in \mathcal{Y}$ such that $\phi^f(\mathbf{x}, \mathbf{y}) < \phi^f(\mathbf{x}, \hat{\mathbf{y}})$ and $w_{\mathbf{y}'q} = 0, \ \forall (\mathbf{y}', q) \in \mathcal{B}(\hat{\mathbf{y}}, \mathcal{Y})$, which by Proposition 2 implies that $\hat{\mathbf{y}} \in \mathcal{Y}(\mathbf{x})$. Because of Proposition 1, this contradicts the assumption that $(\mathbf{x}, \mathbf{y})$ is feasible for the BMIP.

Because the objective function for both problems is the same, any feasible EHPP solution corresponds to a BMIP solution having the same objective, and vice versa. This completes the proof. ■

Solving the EHPP requires the enumeration of all solutions in $\mathcal{Y}$, which could be an exponential-size or infinite set. Therefore, we define a Relaxed Extended High Point Problem (REHPP), which only includes a subset $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$ of follower responses. Formally, problem REHPP($\hat{\mathcal{Y}}$)

is defined exactly as EHPP, except that $\mathcal{Y}$ is replaced by $\hat{\mathcal{Y}}$ throughout. Define $z(\hat{\mathcal{Y}})$ as the optimal objective function value to REHPP($\hat{\mathcal{Y}}$).

Lemma 1. *For any $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$, $z(\hat{\mathcal{Y}}) \geq z^*$.*

Proof Problem REHPP($\hat{\mathcal{Y}}$) is a relaxation of the EHPP because it only considers a subset of disjunctive constraints (2.10b) and (2.10c). Since the EHPP is equivalent to the BMIP by Proposition 3, REHPP($\hat{\mathcal{Y}}$) is in turn a relaxation for the BMIP, for any $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$. ■

Remark 2. *It is instructive to note the relationship between our approach and some similar approaches in the literature. First, we compare our approach with the sample-based relaxation proposed by Mitsos et al. [2008] and Mitsos [2010]. The latter works define a sample of pairs $(\hat{\mathcal{X}}, \hat{\mathbf{y}})$ where $\hat{\mathcal{X}} \subset \Omega(\mathcal{X})$ is a subset of leader solutions such that $\hat{\mathbf{y}} \in \mathcal{Y}(\hat{\mathbf{x}})$ for all $\hat{\mathbf{x}} \in \hat{\mathcal{X}}$. For each pair in the sample, they use auxiliary binary variables to enforce the following logical constraint:*

$$\mathbf{x} \in \hat{\mathcal{X}} \Rightarrow \phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}}). \tag{2.12}$$

Note that for a sample of feasible follower solutions $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$, constraints (2.10b) and (2.10c) enforce $\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}})$ for every $\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$ and every $\mathbf{x} \in \Omega(\mathcal{X})$ such that $\hat{\mathbf{y}} \in \mathcal{Y}(\mathbf{x})$. This is equivalent to imposing logical constraint (2.12) on a sample of pairs $(\hat{\mathcal{X}}, \hat{\mathbf{y}})$ in which $\hat{\mathcal{X}}$ contains all leader solutions $\mathbf{x} \in \Omega(\mathcal{X})$ for which $\hat{\mathbf{y}} \in \mathcal{Y}(\mathbf{x})$, rather than a subset of such leader solutions.

Next, Xu and Wang [2014] obtain upper bounds for the BMILP also based on Proposition 1. However, instead of using a value-function reformulation, their branch-and-bound algorithm generates $m_2 + 1$ branches for a given follower solution $\hat{\mathbf{y}}$. The first m_2 branches correspond to blocking $\hat{\mathbf{y}}$ by each one of the follower's constraints while the last branch imposes the following constraint:

$$\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}}). \tag{2.13}$$

As a result, their approach does not require additional binary variables or the calculation and usage of M-values, which can lead to computational difficulties. However, the approach taken in this chapter avoids the need to create $m_2 + 1$ branches at each node, and permits the use of additional algorithmic tools and model extensions covered in the remainder of this chapter.

2.2.3 Algorithm and Convergence

Algorithm 1 presents our proposed approach for solving BMIPs. This algorithm starts in line 2 by selecting a sample of feasible follower responses $\hat{\mathcal{Y}}^1 \subseteq \mathcal{Y}$. As we will discuss in Section 2.2.4, the only requirement we impose on $\hat{\mathcal{Y}}^1$ for our algorithm to terminate with an optimal solution is that $\hat{\mathcal{Y}}^1 \subseteq \mathcal{Y}$. The main while-loop (line 3) is executed until optimality of the current incumbent is proven or the problem is declared infeasible. Inside this loop, line 5 checks if REHPP($\hat{\mathcal{Y}}^i$) is infeasible. If so, then line 6 terminates the algorithm. Otherwise, line 8 solves REHPP($\hat{\mathcal{Y}}^i$) over the current sample $\hat{\mathcal{Y}}^i$, obtains an optimal solution $(\mathbf{x}^i, \mathbf{y}_l^i)$, and sets the new upper bound equal to the optimal objective function of this relaxed problem. Note that $(\mathbf{x}^i, \mathbf{y}_l^i)$ may not be bilevel feasible, because $\mathbf{y}_l^i$ need not belong to $\Psi(\mathbf{x}^i)$. Line 9 then finds an optimal follower response $\mathbf{y}_f^i \in \Psi(\mathbf{x}^i)$ by solving $\max_{\mathbf{y}}\{\phi^f(\mathbf{x}^i, \mathbf{y}) \mid \mathbf{y} \in \mathcal{Y}(\mathbf{x}^i)\}$. Line 10 defines the sample at the next iteration as the solutions in the previous sample along with $\mathbf{y}_f^i$. Line 11 checks if $(\mathbf{x}^i, \mathbf{y}_l^i)$ is bilevel feasible by testing if $\phi^f(\mathbf{x}^i, \mathbf{y}_l^i) = \phi^f(\mathbf{x}^i, \mathbf{y}_f^i)$; if so, then $\mathbf{y}_l^i$ and $\mathbf{y}_f^i$ must both belong to $\Psi(\mathbf{x}^i)$. If $\mathbf{y}_l^i \in \Psi(\mathbf{x}^i)$, then $(\mathbf{x}^i, \mathbf{y}_l^i)$ becomes the new incumbent solution in line 12; also, the lower and upper bounds match, and the algorithm will terminate with optimal solution $(\mathbf{x}^i, \mathbf{y}_l^i)$. If $(\mathbf{x}^i, \mathbf{y}_l^i)$ is not bilevel feasible, then line 13 determines whether $(\mathbf{x}^i, \mathbf{y}_f^i)$ is bilevel feasible by checking if $\mathbf{x}^i \in \mathcal{X}(\mathbf{y}_f^i)$, and if this solution improves the current lower bound by checking if $\phi^l(\mathbf{x}^i, \mathbf{y}_f^i) > LB_{i-1}$. If both conditions are satisfied, then the lower bound and incumbent solutions are updated accordingly in line 14. If no new lower bound is found at iteration i, then line 16 sets $LB_i = LB_{i-1}$. Line 20 returns an optimal solution. Proposition 4 states the correctness and finiteness of the proposed algorithm.

Algorithm 1 An Exact Algorithm for the BMIP

```
1: Initialize $UB_0 = \infty$, $LB_0 = -\infty$, and set counter $i = 0$
2: Select an initial subset of follower responses $\hat{\mathcal{Y}}^1 \subseteq \mathcal{Y}$    ▷ *see Section 2.2.4*
3: while $UB_i > LB_i$ do
4:     Set $i = i + 1$
5:     if REHPP($\hat{\mathcal{Y}}^i$) is infeasible then
6:         Terminate; the original BMIP instance is infeasible    ▷ *see Remark 3*
7:     else
8:         Obtain an optimal solution $(\mathbf{x}^i, \mathbf{y}^i_l)$ to REHPP($\hat{\mathcal{Y}}^i$), and set $UB_i = z(\hat{\mathcal{Y}}^i)$
9:         Obtain an optimal follower response $\mathbf{y}^i_f \in \Psi(\mathbf{x}^i)$
10:        Set $\hat{\mathcal{Y}}^{i+1} = \hat{\mathcal{Y}}^i \cup \{\mathbf{y}^i_f\}$
11:        if $\phi^f(\mathbf{x}^i, \mathbf{y}^i_l) = \phi^f(\mathbf{x}^i, \mathbf{y}^i_f)$ then
12:            Update $LB_i = UB_i$ and the incumbent solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \leftarrow (\mathbf{x}^i, \mathbf{y}^i_l)$
13:        else if $\mathbf{x}^i \in \mathcal{X}(\mathbf{y}^i_f)$ and $\phi^l(\mathbf{x}^i, \mathbf{y}^i_f) > LB_{i-1}$ then
14:            Update $LB_i = \phi^l(\mathbf{x}^i, \mathbf{y}^i_f)$ and the incumbent solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \leftarrow (\mathbf{x}^i, \mathbf{y}^i_f)$
15:        else
16:            Set $LB_i = LB_{i-1}$
17:        end if
18:    end if
19: end while
20: Return $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$
```

Proposition 4. *Algorithm 1 terminates finitely with an optimal solution.*

Proof Suppose that Algorithm 1 completes $|\Omega(\mathcal{X})| + 1$ iterations of the while loop beginning at line 3. Then, by the finiteness of $\Omega(\mathcal{X})$, there must be two iterations i and k, $1 \leq i < k \leq |\Omega(\mathcal{X})| + 1$, such that $\mathbf{x}^i = \mathbf{x}^k$. At line 10 of iteration i, the algorithm includes a follower response $\mathbf{y}^i_f \in \Psi(\mathbf{x}^i)$ into $\hat{\mathcal{Y}}^{i+1}$. Furthermore, because $\hat{\mathcal{Y}}^{i+1} \subseteq \hat{\mathcal{Y}}^k$, we have that $\mathbf{y}^i_f \in \hat{\mathcal{Y}}^k$. Since $\mathbf{y}^i_f \in \mathcal{Y}(\mathbf{x}^i) = \mathcal{Y}(\mathbf{x}^k)$ and $\mathbf{y}^i_f \in \hat{\mathcal{Y}}^k$, constraints (2.10c) guarantee that $\phi^f(\mathbf{x}^k, \mathbf{y}^k_l) \geq \phi^f(\mathbf{x}^k, \mathbf{y}^i_f)$, implying that $\mathbf{y}^k_l \in \Psi(\mathbf{x}^k)$. Therefore, Algorithm 1 reaches line 12 at iteration k, and terminates with solution $(\mathbf{x}^k, \mathbf{y}^k_l)$ after iteration k. Because the algorithm terminates in no more than $|\Omega(\mathcal{X})| + 1$ iterations with a bilevel feasible solution whose objective value equals an upper bound on z^*, Algorithm 1 terminates finitely with a global optimal solution. ■

Observe that the proof of Proposition 4 relies on the finiteness of the set $\Omega(\mathcal{X})$. If we allow the upper-level variables to be continuous, then $\Omega(\mathcal{X})$ could be an infinite set, and we can no longer claim finite convergence of Algorithm 1.

Remark 3. *If the original BMIP is infeasible, then a similar argument as the one used in Proposition 4 proves that Algorithm 1 detects infeasibility in line 5 after no more than $|\Omega(\mathcal{X})|$ iterations. To see* *this, suppose that the BMIP is infeasible. The proof of Proposition 4 implies that if two iterations $i < k$ are encountered such that $\mathbf{x}^i = \mathbf{x}^k$, the algorithm finds a bilevel feasible solution at iteration k. Therefore, Algorithm 1 yields distinct vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots$, and so the REHPP becomes infeasible after no more than $|\Omega(\mathcal{X})|$ iterations.*

2.2.4 The Sampling Scheme

We propose a sampling scheme that leverages information obtained in the HPP branch-and-bound tree to generate an initial set of solutions $\hat{\mathcal{Y}}^1$. The intuition behind this idea is that the HPP is equivalent to REHPP($\emptyset$). Thus, starting Algorithm 1 with $\hat{\mathcal{Y}}^1 = \emptyset$ would generate an HPP solution in the first iteration. If the follower's response $\mathbf{y}_f^1$ in line 6 makes the HPP solution $\mathbf{x}^1$ infeasible, or if the lower bound due to $(\mathbf{x}^1, \mathbf{y}_f^1)$ is sufficiently poor, then the second iteration of Algorithm 1 amounts to searching for the second-best HPP solution. This process may continue for several iterations, with Algorithm 1 generating a sequence $\mathbf{x}^1, \mathbf{x}^2, \ldots$ of solutions that appear as feasible solutions in the HPP branch-and-bound tree.

Our proposed sampling scheme is conducted within the branch-and-bound solution of the HPP. Our sampling scheme analyzes every node that yields a feasible HPP solution, $(\mathbf{x}, \mathbf{y}_l)$, and finds an optimal follower response $\mathbf{y}_f \in \Psi(\mathbf{x})$. If $\mathbf{y}_l$ belongs to $\Psi(\mathbf{x})$, then $(\mathbf{x}, \mathbf{y}_l)$ is bilevel feasible: The procedure adds $\mathbf{y}_l$ to the sample and updates lower bound, $\underline{z}$, accordingly. If $(\mathbf{x}, \mathbf{y}_l)$ is not bilevel feasible, then the procedure adds $\mathbf{y}_f$ to the sample. Furthermore, if $(\mathbf{x}, \mathbf{y}_f)$ is bilevel feasible, then the procedure updates the lower bound. We terminate the sampling procedure if the best upper bound from the branch-and-bound tree is less than or equal to $\underline{z}$ or if a maximum initial sample size limit is exceeded.

Note that our proposed sampling scheme differs from a standard approach that simply collects integer leader solutions generated in the branch-and-bound search and then adds the corresponding follower responses to the sample. Specifically, our sampling procedure only updates the lower bound from the branch-and-bound tree when bilevel feasible solutions are found. As a result, fewer promising leader solutions are fathomed, which in turn leads to the generation of larger and more relevant initial samples. Preliminary computational experiments show that using our sampling scheme reduces the execution time of Algorithm 1 by a factor of three compared to the standard approach.

Collecting a large initial sample using our sampling scheme benefits our proposed algorithm, especially if $\hat{\mathcal{Y}}$ is chosen so that an optimal follower solution will typically belong to $\hat{\mathcal{Y}}$. However, if $|\hat{\mathcal{Y}}|$ is too large, then REHPP($\hat{\mathcal{Y}}$) will be large as well, and may potentially be too difficult to solve. Therefore, it is prudent to eliminate solutions in $\hat{\mathcal{Y}}$ that will not likely appear as optimal follower solutions in the course of Algorithm 1. Toward that purpose, we define the concept of dominated and alternative solutions.

Definition 1. *Consider solutions* $\mathbf{y}, \mathbf{y}' \in \mathcal{Y}$. *If* $\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \mathbf{y}')$ *for all* $\mathbf{x} \in \Omega(\mathcal{X})$ *and* $h^2_j(\mathbf{y}) \leq h^2_j(\mathbf{y}')$ *for all* $j = 1, \ldots, m_2$*, with at least one inequality being strict (among both sets of inequalities), then* $\mathbf{y}$ *dominates* $\mathbf{y}'$.

Definition 2. *Consider solutions* $\mathbf{y}, \mathbf{y}' \in \mathcal{Y}$ *such that* $\mathbf{y} \neq \mathbf{y}'$. *If* $\phi^f(\mathbf{x}, \mathbf{y}) = \phi^f(\mathbf{x}, \mathbf{y}')$ *for all* $\mathbf{x} \in \Omega(\mathcal{X})$ *and* $h^2_j(\mathbf{y}) = h^2_j(\mathbf{y}')$ *for all* $j = 1, \ldots, m_2$*, then* $\mathbf{y}$ *and* $\mathbf{y}'$ *are said to be alternative solutions.*

Proposition 5 shows that for any sample $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$, removing dominated solutions does not change the optimal objective function value of REHPP($\hat{\mathcal{Y}}$).

Proposition 5. *Consider any sample* $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$ *and let* $\bar{\mathcal{Y}} = \{\mathbf{y}' \in \hat{\mathcal{Y}} \mid \mathbf{y}$ *does not dominate* $\mathbf{y}', \forall \mathbf{y} \in \hat{\mathcal{Y}}\}$. *Problem REHPP*($\bar{\mathcal{Y}}$) *is equivalent to REHPP*($\hat{\mathcal{Y}}$).

Proof For any pair of solutions $\mathbf{y}$ and $\mathbf{y}'$, if $\mathbf{y}$ dominates $\mathbf{y}'$, then $\gamma_{\mathbf{y}j} \geq \gamma_{\mathbf{y}'j}$, $\forall j = 1, \ldots, m_2$. This implies that if $\mathbf{y}$ dominates $\mathbf{y}'$, then constraint (2.10c) associated with $\mathbf{y}'$ is implied by the corresponding constraint associated with $\mathbf{y}$. All disjunctive constraints (2.10c) associated with solutions in $\hat{\mathcal{Y}} \setminus \bar{\mathcal{Y}}$ are redundant, and so any feasible solution to REHPP($\bar{\mathcal{Y}}$) is feasible to REHPP($\hat{\mathcal{Y}}$) and vice versa. Because the objective function for these problems is not affected by the sample choice, this completes the proof. ■

Remark 4. *The same argument used in Proposition 5 proves that in a sample* $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$ *containing a pair of alternative solutions* $\mathbf{y}$ *and* $\mathbf{y}'$*, one can be removed without changing the optimal objective function value of REHPP*($\hat{\mathcal{Y}}$).

Based on these results, before solving REHPP($\hat{\mathcal{Y}}$) we remove dominated solutions from $\hat{\mathcal{Y}}$ and retain only one solution out of a set of multiple alternative solutions, if any exist.

2.2.5 The Linear Case

Of particular interest are BMILPs in which the objectives and constraints are linear. The BMILP can be formally stated as:

$$\max_{\mathbf{x},\mathbf{y}} \quad \mathbf{cx} + \mathbf{d}^1\mathbf{y} \tag{2.14a}$$

$$\text{s.t.} \quad \mathbf{A}^1\mathbf{x} + \mathbf{B}^1\mathbf{y} \leq \mathbf{b}^1 \tag{2.14b}$$

$$\mathbf{y} \in \underset{\mathbf{y}^f}{\operatorname{argmax}}\{\mathbf{d}^2\mathbf{y}^f \mid \mathbf{A}^2\mathbf{x} + \mathbf{B}^2\mathbf{y}^f \leq \mathbf{b}^2;\ \mathbf{y}^f \in \mathcal{H}^y\} \tag{2.14c}$$

$$\mathbf{x} \in \mathcal{H}^x. \tag{2.14d}$$

The upper-level problem has m_1 constraints, the lower-level problem has m_2 constraints, and the coefficient matrices have conforming dimensions. For our algorithm to finitely converge to an optimal solution we must only assume that all leader variables are integer-valued, the coefficients in constraint matrix $\mathbf{A}^2$ are integers, and that the $\mathbf{x}$- and $\mathbf{y}$-variables are bounded. Also, because the objective functions are separable, dominance relationships become simpler in this case. Definition 3 specifies the concept of dominance for BMILPs.

Definition 3. *Let $\boldsymbol{\beta}_j$ be the jth row of constraint matrix $\mathbf{B}^2$. Consider solutions $\mathbf{y}, \mathbf{y}' \in \mathcal{Y}$. If $\mathbf{d}^2\mathbf{y} \geq \mathbf{d}^2\mathbf{y}'$ and $\boldsymbol{\beta}_j^2\hat{\mathbf{y}} \leq \boldsymbol{\beta}_j^2\mathbf{y}'$ for all $j = 1, \ldots, m_2$, with at least one inequality being strict (among both sets of inequalities), then $\mathbf{y}$ dominates $\mathbf{y}'$.*

2.3 Strengthening the REHPP Formulation

Solving REHPP($\hat{\mathcal{Y}}$) is the most computationally expensive step of Algorithm 1. Accordingly, we examine strategies that accelerate the solution of REHPP($\hat{\mathcal{Y}}$). Section 2.3.1 seeks to fix $\mathbf{w}$-variable values *a priori*, while Section 2.3.2 identifies *supervalid inequalities* (SVIs) for the problem. SVIs potentially cut off integer solutions, but ensure that not all optimal solutions are eliminated [Israeli and Wood, 2002].

The proposed acceleration strategies utilize lower bounds on z^*. Note that in Algorithm 1, at the beginning of any iteration $k > 1$, we have a lower bound LB_{k-1} on z^*. At iteration $k = 1$ we leverage our proposed sampling scheme by setting $LB_0 = \underline{z}$. Henceforth, we will refer to these bounds at a given iteration as LB and denote by $\hat{\mathcal{Y}}$ a sample of follower solutions in $\mathcal{Y}$, with the iteration index omitted.

2.3.1 Fixing Variables

We propose a strategy that fixes some $w_{\hat{\mathbf{y}}j} = 0$ whenever it is impossible to set $w_{\hat{\mathbf{y}}j} = 1$ in an optimal solution. We start by solving the following continuous optimization problem for every $j = 1, \ldots, m_2$:

$$u_j = \max \quad g_j^2(\mathbf{x}) \tag{2.15a}$$

$$\text{s.t.} \quad g_i^1(\mathbf{x}) + h_i^1(\mathbf{y}) \le b_i^1 \qquad \forall i = 1, \ldots, m_1 \tag{2.15b}$$

$$g_i^2(\mathbf{x}) + h_i^2(\mathbf{y}) \le b_i^2 \qquad \forall i = 1, \ldots, m_2 \tag{2.15c}$$

$$\phi^l(\mathbf{x}, \mathbf{y}) \ge LB \tag{2.15d}$$

$$\mathbf{x}, \mathbf{y} \ge \mathbf{0}. \tag{2.15e}$$

Lemma 2. *If $\gamma_{\hat{\mathbf{y}}j} > u_j$ for a solution $\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$ and a lower-level constraint $j \in \{1, \ldots, m_2\}$, then $w_{\hat{\mathbf{y}}j} = 0$ for any optimal REHPP($\hat{\mathcal{Y}}$) solution.*

Proof Any optimal REHPP($\hat{\mathcal{Y}}$) solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{w}})$ satisfies $g_j^2(\bar{\mathbf{x}}) \le u_j$. This inequality holds because constraints (2.15b)–(2.15d) enforce the upper- and lower-level constraints while ensuring that the leader's objective value is greater than or equal to the lower bound. If $\gamma_{\hat{\mathbf{y}}j} > u_j$, then any optimal solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{w}})$ to REHPP($\hat{\mathcal{Y}}$) satisfies $g_j^2(\bar{\mathbf{x}}) < \gamma_{\hat{\mathbf{y}}j}$, which implies that $\bar{w}_{\hat{\mathbf{y}}j} = 0$. This completes the proof. ■

2.3.2 Supervalid Inequalities

Our first set of SVIs is based on the following observation. In the continuous relaxation of REHPP($\hat{\mathcal{Y}}$), constraints (2.10b) tend to be active. This occurs because for any fixed value of $\mathbf{x}$, optimization forces the $\mathbf{w}$-values to be as large as possible in order to decrease the right-hand side of constraints (2.10c). Depending on the M-values defined, this can lead to solutions with several $\mathbf{w}$-variables taking fractional values, resulting in a weak continuous relaxation and a large branch-and-bound tree. To mitigate the extent to which $\mathbf{w}$-variables can be fractional in this manner, Lemma 3 states the following condition.

Lemma 3. *There exists an optimal solution to REHPP($\hat{\mathcal{Y}}$) in which*

$$\sum_{\hat{\mathbf{y}} \in \hat{\mathcal{Y}}} w_{\hat{\mathbf{y}}j} \leq 1 \quad \forall j = 1, \ldots, m_2. \tag{2.16}$$

Proof Consider an optimal solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{w}})$ to REHPP($\hat{\mathcal{Y}}$) in which $\sum_{\hat{\mathbf{y}} \in \hat{\mathcal{Y}}} \bar{w}_{\hat{\mathbf{y}}j} > 1$ for some $j \in \{1, \ldots, m_2\}$. An alternative optimal solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \mathbf{w}')$ that satisfies (2.16) exists by defining $\mathbf{w}'$ as in Equation (2.11). ■

Our second set of SVIs is based on the idea of identifying follower solutions that must be blocked in an optimal solution to REHPP($\hat{\mathcal{Y}}$). We first obtain an upper bound, u^f, on the maximum objective achievable by the follower, while insisting that the leader's objective value is at least LB.

$$u^f = \max \quad \phi^f(\mathbf{x}, \mathbf{y}) \tag{2.17a}$$

$$\text{s.t.} \quad \phi^l(\mathbf{x}, \mathbf{y}) \geq LB \tag{2.17b}$$

$$(\mathbf{x}, \mathbf{y}) \in \Omega. \tag{2.17c}$$

Lemma 4. *Any follower solution $\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$ such that $\phi^f(\mathbf{x}, \hat{\mathbf{y}}) > u^f$, $\forall \mathbf{x} \in \Omega(\mathcal{X})$, must be blocked in an optimal REHPP($\hat{\mathcal{Y}}$) solution.*

Proof Suppose by contradiction that there exists an optimal REHPP($\hat{\mathcal{Y}}$) solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{w}})$ that does not block $\hat{\mathbf{y}}$, i.e., $\bar{w}_{\mathbf{y}'q} = 0$, $\forall (\mathbf{y}', q) \in \mathcal{B}(\hat{\mathbf{y}}, \hat{\mathcal{Y}})$. By constraints (2.10c) and the assumption of the lemma, we have that $\phi^f(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \geq \phi^f(\bar{\mathbf{x}}, \hat{\mathbf{y}}) > u^f$, which implies that $\phi^l(\bar{\mathbf{x}}, \bar{\mathbf{y}}) < LB$. This contradicts the optimality of $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{w}})$. ■

Based on Lemma 4, let $\hat{\mathcal{Y}}^b = \{\hat{\mathbf{y}} \in \hat{\mathcal{Y}} \mid \phi^f(\mathbf{x}, \hat{\mathbf{y}}) > u^f, \forall \mathbf{x} \in \Omega(\mathcal{X})\}$ be the subset of follower solutions from the sample that must be blocked. Our second set of proposed SVIs requires the leader to block any follower solution $\hat{\mathbf{y}} \in \hat{\mathcal{Y}}^b$:

$$\sum_{(\mathbf{y}', q) \in \mathcal{B}(\hat{\mathbf{y}}, \hat{\mathcal{Y}})} w_{\mathbf{y}'q} \geq 1 \quad \forall \hat{\mathbf{y}} \in \hat{\mathcal{Y}}^b. \tag{2.18}$$

Our third set of SVIs uses upper bounds on $\phi^l(\mathbf{x}, \mathbf{y})$, obtained under the assumption that a given solution $\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$ is not being blocked. To obtain the desired bounds, we solve for every

$\hat{\mathbf{y}} \in \hat{\mathcal{Y}} \setminus \hat{\mathcal{Y}}^b$ the following optimization problem:

$$
\begin{aligned}
u^l_{\hat{\mathbf{y}}} = \max \quad & \phi^l(\mathbf{x}, \mathbf{y}) && && (2.19a)\\
\text{s.t.} \quad & g^1_j(\mathbf{x}) + h^1_j(\mathbf{y}) \leq b^1_j && \forall j = 1, \ldots, m_1 && (2.19b)\\
& g^2_j(\mathbf{x}) + h^2_j(\mathbf{y}) \leq b^2_j && \forall j = 1, \ldots, m_2 && (2.19c)\\
& g^2_j(\mathbf{x}) \leq b^2_j - h^2_j(\hat{\mathbf{y}}) && \forall j = 1, \ldots, m_2 && (2.19d)\\
& \mathbf{x}, \mathbf{y} \geq \mathbf{0}. && && (2.19e)
\end{aligned}
$$

Constraints (2.19d) ensure that follower solution $\hat{\mathbf{y}}$ is not blocked by $\mathbf{x}$. Let $\bar{u}$ be the upper bound on $\phi^l(\mathbf{x}, \mathbf{y})$ obtained by solving model (2.19) without constraints (2.19d), i.e., without enforcing that $\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$ is not being blocked. Note that $\bar{u} \geq u^l_{\hat{\mathbf{y}}}$ for all $\hat{\mathbf{y}} \in \hat{\mathcal{Y}} \setminus \hat{\mathcal{Y}}^b$.

Lemma 5. *All optimal solutions to REHPP($\hat{\mathcal{Y}}$) satisfy the following inequalities:*

$$
\phi^l(\mathbf{x}, \mathbf{y}) \leq u^l_{\hat{\mathbf{y}}} + (\bar{u} - u^l_{\hat{\mathbf{y}}}) \sum_{(\mathbf{y}', q) \in \mathcal{B}(\hat{\mathbf{y}}, \hat{\mathcal{Y}})} w_{\mathbf{y}'q} \quad \forall \hat{\mathbf{y}} \in \hat{\mathcal{Y}} \setminus \hat{\mathcal{Y}}^b. \tag{2.20}
$$

Proof Consider any optimal REHPP($\hat{\mathcal{Y}}$) solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{w}})$. If $\hat{\mathbf{y}}$ is not blocked, then $\bar{w}_{\mathbf{y}'q} = 0,\ \forall (\mathbf{y}', q) \in \mathcal{B}(\hat{\mathbf{y}}, \hat{\mathcal{Y}})$, and inequalities (2.20) enforce $\phi^l(\mathbf{x}, \mathbf{y}) \leq u^l_{\hat{\mathbf{y}}}$. If $\hat{\mathbf{y}}$ is blocked, then

$$
\sum_{(\mathbf{y}', q) \in \mathcal{B}(\hat{\mathbf{y}}, \hat{\mathcal{Y}})} w_{\mathbf{y}'q} \geq 1
$$

and the corresponding inequality (2.20) becomes redundant. In both cases, $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{w}})$ satisfies inequalities (2.20). ■

Our fourth set of SVIs investigates the complement of inequalities (2.20). Here, we identify the maximum attainable value of $\phi^l(\mathbf{x}, \mathbf{y})$ when the leader blocks some solution $\hat{\mathbf{y}}$ by constraint j. We compute this value, $u_{\hat{\mathbf{y}}j}$, by solving the following optimization problem for every $\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$ and $j = 1, \ldots, m_2$:

$$
\begin{aligned}
u_{\hat{\mathbf{y}}j} = \max \quad & \phi^l(\mathbf{x}, \mathbf{y}) && && (2.21a)\\
\text{s.t.} \quad & g^1_i(\mathbf{x}) + h^1_i(\mathbf{y}) \leq b^1_i && \forall i = 1, \ldots, m_1 && (2.21b)\\
& g^2_i(\mathbf{x}) + h^2_i(\mathbf{y}) \leq b^2_i && \forall i = 1, \ldots, m_2 && (2.21c)
\end{aligned}
$$

$$g_j^2(\mathbf{x}) \geq \gamma_{\hat{\mathbf{y}}j} \tag{2.21d}$$

$$\mathbf{x}, \mathbf{y} \geq \mathbf{0}. \tag{2.21e}$$

These continuous optimization problems provide the desired upper bound on $\phi^l(\mathbf{x}, \mathbf{y})$, because constraints (2.21b)–(2.21c) enforce the upper- and lower-level constraints, while (2.21d) requires the leader to block solution $\hat{\mathbf{y}}$ by constraint j. Lemma 6 states our fourth set of SVIs.

Lemma 6. *There exists an optimal solution to REHPP($\hat{\mathcal{Y}}$) that satisfies the following inequalities:*

$$\phi^l(\mathbf{x}, \mathbf{y}) \leq \bar{u} - \sum_{\hat{\mathbf{y}} \in \hat{\mathcal{Y}}} (\bar{u} - u_{\hat{\mathbf{y}}j}) w_{\hat{\mathbf{y}}j} \quad \forall j = 1, \ldots, m_2. \tag{2.22}$$

Proof Consider an optimal REHPP($\hat{\mathcal{Y}}$) solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{w}})$ and some $j \in \{1, \ldots, m_2\}$. If

$$\sum_{\hat{\mathbf{y}} \in \hat{\mathcal{Y}}} \bar{w}_{\hat{\mathbf{y}}j} = 0,$$

then the corresponding inequality (2.22) becomes redundant. If

$$\sum_{\hat{\mathbf{y}} \in \hat{\mathcal{Y}}} \bar{w}_{\hat{\mathbf{y}}j} = 1,$$

then the corresponding inequality (2.22) imposes an upper bound $u_{\hat{\mathbf{y}}j}$ on $\phi^l(\mathbf{x}, \mathbf{y})$. In both cases, $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{w}})$ satisfies inequalities (2.22). Because Lemma 3 guarantees the existence of an optimal solution in which

$$\sum_{\hat{\mathbf{y}} \in \hat{\mathcal{Y}}} \bar{w}_{\hat{\mathbf{y}}j} \leq 1,$$

this completes the proof. ■

2.4 Extension to the Pessimistic Formulation

In the pessimistic formulation the follower chooses a $\hat{\mathbf{y}} \in \Psi(\mathbf{x})$ that makes $\mathbf{x}$ infeasible if such a $\hat{\mathbf{y}}$ exists. If no $\hat{\mathbf{y}} \in \Psi(\mathbf{x})$ exists for which $\mathbf{x} \notin \mathcal{X}(\hat{\mathbf{y}})$, then the follower selects a $\hat{\mathbf{y}} \in \Psi(\mathbf{x})$ that minimizes $\phi^l(\mathbf{x}, \mathbf{y})$ instead, thus seeking to minimize the leader's objective among all alternative optimal solutions to its own problem. Define

$$\Psi^p(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{y}\in\Psi(\mathbf{x})}\{\phi^l(\mathbf{x},\mathbf{y})\} \tag{2.23}$$

as the *rational pessimistic reaction set* and

$$\Psi^b(\mathbf{x}) = \{\mathbf{y} \in \Psi(\mathbf{x}) \mid \exists j \text{ such that } g_j^1(\mathbf{x}) + h_j^1(\mathbf{y}) > b_j^1\} \tag{2.24}$$

as the subset of follower solutions in $\Psi(\mathbf{x})$ that block leader solution $\mathbf{x}$. Under the pessimistic assumption, a solution $(\mathbf{x},\mathbf{y}) \in \Omega$ is said to be bilevel feasible if $(\mathbf{x},\mathbf{y}) \in \Omega^p$, where

$$\Omega^p = \{(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in \mathcal{X}(\mathbf{y}),\ \mathbf{y} \in \Psi^p(\mathbf{x}),\ \Psi^b(\mathbf{x}) = \emptyset\}. \tag{2.25}$$

We initially assume that there is a parameter $\delta > 0$ such that $|\phi^f(\mathbf{x},\mathbf{y}') - \phi^f(\mathbf{x},\mathbf{y})| \in \{0, [\delta,\infty)\}$ for all $\mathbf{x} \in \Omega(\mathcal{X})$ and $\mathbf{y},\mathbf{y}' \in \Psi^p(\mathbf{x})$. (Remark 5 below handles the case in which no such δ is known.) Clearly, $\delta = 1$ for problems in which ϕ^f only takes integer values. Alternatively, one could regard δ as a value such that if $|\phi^f(\mathbf{x},\mathbf{y}') - \phi^f(\mathbf{x},\mathbf{y})| < \delta$, then the follower regards $\mathbf{y}$ and $\mathbf{y}'$ as being essentially alternative optimal solutions. This notion slightly extends the concept of a pessimistic follower by allowing the follower to suboptimize, choosing a solution whose objective function value is strictly within δ of optimal in order to minimize the leader's objective.

We now extend Proposition 1 to accommodate the pessimistic formulation.

Proposition 6. *Solution $(\mathbf{x},\mathbf{y}) \in \Omega$ belongs to Ω^p if and only if: (i) $\phi^f(\mathbf{x},\mathbf{y}) \geq \phi^f(\mathbf{x},\hat{\mathbf{y}})$ for every $\hat{\mathbf{y}} \in \mathcal{Y}(\mathbf{x})$; (ii) $\phi^l(\mathbf{x},\mathbf{y}) \leq \phi^l(\mathbf{x},\hat{\mathbf{y}})$ for all $\hat{\mathbf{y}} \in \Psi(\mathbf{x})$; and (iii) $g_j^1(\mathbf{x}) + h_j^1(\hat{\mathbf{y}}) \leq b_j^1$ for every $\hat{\mathbf{y}} \in \Psi(\mathbf{x})$, $j = 1,\ldots,m_1$.*

Proof Assume that $(\mathbf{x},\mathbf{y}) \in \Omega^p$. Proposition 1 proves that statement (i) holds true. For statement (ii) suppose by contradiction that there exists a follower solution $\hat{\mathbf{y}} \in \Psi(\mathbf{x})$ such that $\phi^l(\mathbf{x},\mathbf{y}) > \phi^l(\mathbf{x},\hat{\mathbf{y}})$. Then $\mathbf{y} \notin \Psi^p(\mathbf{x})$, which contradicts the assumption that $(\mathbf{x},\mathbf{y}) \in \Omega^p$. For statement (iii) suppose by contradiction that there exists a follower solution $\hat{\mathbf{y}} \in \Psi(\mathbf{x})$ and an upper-level constraint $j \in \{1,\ldots,m_1\}$ such that $g_j^1(\mathbf{x}) + h_j^1(\hat{\mathbf{y}}) > b_j^1$. Then $\hat{\mathbf{y}} \in \Psi^b(\mathbf{x})$, i.e., $\Psi^b(\mathbf{x}) \neq \emptyset$, which contradicts the assumption that $(\mathbf{x},\mathbf{y}) \in \Omega^p$.

Now assume that (i), (ii), and (iii) hold. Statement (i) implies that $\mathbf{y} \in \Psi(\mathbf{x})$, while (ii) implies that $\mathbf{y} \in \Psi^p(\mathbf{x})$, and (iii) implies that $\Psi^b(\mathbf{x}) = \emptyset$. Moreover, $\mathbf{x} \in \mathcal{X}(\mathbf{y})$ since $(\mathbf{x}, \mathbf{y}) \in \Omega$, and so $(\mathbf{x}, \mathbf{y}) \in \Omega^p$. This completes the proof. ■

Define a binary variable $v_{\hat{\mathbf{y}}}$ corresponding to every $\hat{\mathbf{y}} \in \mathcal{Y}$ such that $v_{\hat{\mathbf{y}}} = 0$ if $\phi^f(\mathbf{x}, \mathbf{y}) > \phi^f(\mathbf{x}, \hat{\mathbf{y}})$. If $v_{\hat{\mathbf{y}}} = 1$, then $\phi^f(\mathbf{x}, \mathbf{y})$ may or may not be greater than $\phi^f(\mathbf{x}, \hat{\mathbf{y}})$. We formulate a Pessimistic Extended High Point Problem (PEHPP) where the M-values are large constants whose values we discuss in Section 2.6.5.

$$\begin{aligned}
\max_{(\mathbf{x},\mathbf{y})} \quad & \phi^l(\mathbf{x}, \mathbf{y}) && && (2.26a)\\
\text{s.t.} \quad & g_j^2(\mathbf{x}) \geq -M_j^1 + \sum_{\hat{\mathbf{y}} \in \mathcal{Y}} (M_j^1 + \gamma_{\hat{\mathbf{y}}j}) w_{\hat{\mathbf{y}}j} && \forall j = 1, \ldots, m_2 && (2.26b)\\
& \phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}}) - M_{\hat{\mathbf{y}}}^2 \sum_{(\mathbf{y}', q) \in \mathcal{B}(\hat{\mathbf{y}}, \mathcal{Y})} w_{\mathbf{y}'q} && \forall \hat{\mathbf{y}} \in \mathcal{Y} && (2.26c)\\
& \phi^f(\mathbf{x}, \mathbf{y}) + M_{\hat{\mathbf{y}}}^3 v_{\hat{\mathbf{y}}} \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}}) + \delta && \forall \hat{\mathbf{y}} \in \mathcal{Y} && (2.26d)\\
& \phi^l(\mathbf{x}, \mathbf{y}) \leq \phi^l(\mathbf{x}, \hat{\mathbf{y}}) + M_{\hat{\mathbf{y}}}^4 \sum_{(\mathbf{y}', q) \in \mathcal{B}(\hat{\mathbf{y}}, \mathcal{Y})} w_{\mathbf{y}'q} + M_{\hat{\mathbf{y}}}^4 (1 - v_{\hat{\mathbf{y}}}) && \forall \hat{\mathbf{y}} \in \mathcal{Y} && (2.26e)\\
& g_j^1(\mathbf{x}) + h_j^1(\hat{\mathbf{y}}) \leq b_j^1 + M_{\hat{\mathbf{y}}j}^5 \sum_{(\mathbf{y}', q) \in \mathcal{B}(\hat{\mathbf{y}}, \mathcal{Y})} w_{\mathbf{y}'q} + M_{\hat{\mathbf{y}}j}^5 (1 - v_{\hat{\mathbf{y}}}) && \forall \hat{\mathbf{y}} \in \mathcal{Y},\ j = 1, \ldots, m_1 && (2.26f)\\
& (\mathbf{x}, \mathbf{y}) \in \Omega && && (2.26g)\\
& w_{\hat{\mathbf{y}}j} \in \{0, 1\} && \forall \hat{\mathbf{y}} \in \mathcal{Y},\ j = 1, \ldots, m_2 && (2.26h)\\
& v_{\hat{\mathbf{y}}} \in \{0, 1\} && \forall \hat{\mathbf{y}} \in \mathcal{Y}. && (2.26i)
\end{aligned}$$

As in the EHPP, the objective function (2.26a) maximizes the leader's objective. Constraints (2.26b)–(2.26i) utilize binary variables $\mathbf{w}$ and $\mathbf{v}$ to enforce the bilevel feasibility conditions established in Proposition 6. In particular, constraints (2.26b) define the $\mathbf{w}$-variables and constraints (2.26c) ensure that $\mathbf{y} \in \Psi(\mathbf{x})$. Constraints (2.26d) define the $\mathbf{v}$-variables. Constraints (2.26e) then imply that for every $\hat{\mathbf{y}} \in \mathcal{Y}$, the leader must select an $(\mathbf{x}, \mathbf{y})$ such that $\phi^l(\mathbf{x}, \mathbf{y}) \leq \phi^l(\mathbf{x}, \hat{\mathbf{y}})$ unless $\hat{\mathbf{y}}$ has been blocked or $\hat{\mathbf{y}} \notin \Psi(\mathbf{x})$. Constraints (2.26f) ensure that $\Psi^b(\mathbf{x}) = \emptyset$ by requiring that $g_j^1(\mathbf{x}) + h_j^1(\hat{\mathbf{y}}) \leq b_j^1$ unless $\hat{\mathbf{y}}$ is blocked or $\hat{\mathbf{y}} \notin \Psi(\mathbf{x})$. Constraints (2.26g) enforce the upper- and lower-level constraints. Finally, constraints (2.26h) and (2.26i) restrict the $\mathbf{w}$-variables and $\mathbf{v}$-variables to be binary-valued, respectively. A similar argument to the one used in Proposition 3 proves that the PEHPP is equivalent to the pessimistic formulation of the BMIP.

We define a Relaxed Pessimistic Extended High Point Problem (RPEHPP), which only considers a subset $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$ of follower responses. Problem RPEHPP($\hat{\mathcal{Y}}$) is defined exactly as PEHPP, except that $\mathcal{Y}$ is replaced by $\hat{\mathcal{Y}}$ throughout. Define $z^p(\hat{\mathcal{Y}})$ as the optimal objective function value to RPEHPP($\hat{\mathcal{Y}}$) and note that $z(\hat{\mathcal{Y}}) \geq z^p(\hat{\mathcal{Y}})$ for any $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$.

Algorithm 2 presents a two-phase approach that extends Algorithm 1 to solve the pessimistic formulation of the problem. Phase one starts in line 2 by solving the optimistic version of the problem using Algorithm 1. We record all leader solutions considered throughout the execution of Algorithm 1, and place them into set $\hat{\mathcal{X}}$. Lines 3–12 generate the initial sample of follower solutions $\hat{\mathcal{Y}}^1$ by finding an optimal pessimistic follower response $\hat{\mathbf{y}}$ to every $\hat{\mathbf{x}} \in \hat{\mathcal{X}}$.

To obtain a pessimistic follower response for a given leader solution $\mathbf{x}^i$, we first compute the optimal follower's objective value by solving $z_f(\mathbf{x}^i) = \max_{\mathbf{y}}\{\phi^f(\mathbf{x}^i, \mathbf{y}) \mid \mathbf{y} \in \mathcal{Y}(\mathbf{x}^i)\}$.

Next, we find a follower response $\mathbf{y} \in \Psi^b(\mathbf{x}^i)$ or establish that $\Psi^b(\mathbf{x}^i) = \emptyset$. Define a continuous variable s_j and a binary variable q_j for every upper-level constraint $j = 1, \ldots, m_1$ such that $s_j = 0$ if $(\mathbf{x}^i, \mathbf{y})$ satisfies constraint j. We formulate the following problem to determine if a solution $\mathbf{y} \in \Psi^b(\mathbf{x}^i)$ exists.

$$\max_{\mathbf{y},\mathbf{s},\mathbf{q}} \quad \sum_{j=1}^{m_1} s_j \tag{2.27a}$$

$$\text{s.t.} \quad \phi^f(\mathbf{x}^i, \mathbf{y}) = z_f(\mathbf{x}^i) \tag{2.27b}$$

$$s_j \leq h_j^1(\mathbf{y}) - \left(b_j^1 - g_j^1(\mathbf{x}^i)\right) + M_j^6 q_j \qquad \forall j = 1, \ldots, m_1 \tag{2.27c}$$

$$s_j \leq (1 - q_j) \qquad \forall j = 1, \ldots, m_1 \tag{2.27d}$$

$$\mathbf{y} \in \mathcal{Y}(\mathbf{x}^i) \tag{2.27e}$$

$$q_j \in \{0, 1\} \qquad \forall j = 1, \ldots, m_1 \tag{2.27f}$$

$$\mathbf{s} \geq \mathbf{0}. \tag{2.27g}$$

The objective function (2.27a) takes a positive value if and only if $\Psi^b(\mathbf{x}^i) \neq \emptyset$. Constraints (2.27b) guarantee that $\mathbf{y} \in \Psi(\mathbf{x}^i)$. Constraints (2.27c)–(2.27d) ensure that if $(\mathbf{x}^i, \mathbf{y})$ satisfies constraint j, then $s_j = 0$. If $(\mathbf{x}^i, \mathbf{y})$ violates constraint j, then optimization ensures that $s_j = \min\left\{h_j^1(\mathbf{y}) - \left(b_j^1 - g_j^1(\mathbf{x}^i)\right), 1\right\}$, which is a strictly positive value. Constraints (2.27e) impose the lower-level constraints. Constraints (2.27f) restrict the $\mathbf{q}$-variables to be binary-valued and constraints (2.27g) require the $\mathbf{s}$-variables the to be nonnegative.

Finally, if problem (2.27) has an optimal objective function value equal to zero, we select $\mathbf{y} \in \Psi^p(\mathbf{x}^i)$ by solving:

$$\min_{\mathbf{y}} \quad \phi^l(\mathbf{x}^i, \mathbf{y}) \tag{2.28a}$$

$$\text{s.t.} \quad \phi^f(\mathbf{x}^i, \mathbf{y}) = z_f(\mathbf{x}^i) \tag{2.28b}$$

$$\mathbf{y} \in \mathcal{Y}(\mathbf{x}^i). \tag{2.28c}$$

The objective function (2.28a) minimizes the leader's objective. Constraints (2.28b) ensure that $\mathbf{y} \in \Psi(\mathbf{x}^i)$ while constraints (2.28c) impose the lower-level constraints.

Algorithm 2 Exact Algorithm for the Pessimistic Variation of the BMIP

```
 1: Initialize $LB_0 = -\infty$, $\hat{\mathcal{Y}}^1 = \emptyset$, and counter $i = 0$          ▷ *Begin phase one*
 2: Solve $UB_0 = \max_{(\mathbf{x},\mathbf{y})}\{\phi^l(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in \mathcal{X}(\mathbf{y}),\ \mathbf{y} \in \Psi(\mathbf{x})\}$ using Algorithm 1 and let $\hat{\mathcal{X}}$ be the set of all
    leader solutions explored
 3: for $\hat{\mathbf{x}} \in \hat{\mathcal{X}}$ do
 4:     if $\Psi^b(\hat{\mathbf{x}}) \neq \emptyset$ then
 5:         Obtain a follower response $\hat{\mathbf{y}} \in \Psi^b(\hat{\mathbf{x}})$ and add it into $\hat{\mathcal{Y}}^1$
 6:     else
 7:         Obtain a follower response $\hat{\mathbf{y}} \in \Psi^p(\hat{\mathbf{x}})$ and add it into $\hat{\mathcal{Y}}^1$
 8:         if $\phi^l(\hat{\mathbf{x}},\hat{\mathbf{y}}) > LB_0$ then
 9:             Update $LB_0 = \phi^l(\hat{\mathbf{x}},\hat{\mathbf{y}})$ and the incumbent solution $(\bar{\mathbf{x}},\bar{\mathbf{y}}) \leftarrow (\hat{\mathbf{x}},\hat{\mathbf{y}})$
10:         end if
11:     end if
12: end for
13: while $UB_i > LB_i$ do                                                                 ▷ *Begin phase two*
14:     Set $i = i + 1$ and $LB_i = LB_{i-1}$
15:     if RPEHPP($\hat{\mathcal{Y}}^i$) is infeasible then
16:         Terminate; the original BMIP instance is infeasible
17:     else
18:         Obtain an optimal solution $(\mathbf{x}^i, \mathbf{y}^i_l)$ to RPEHPP($\hat{\mathcal{Y}}^i$), and set $UB_i = z^p(\hat{\mathcal{Y}}^i)$
19:         if $\Psi^b(\mathbf{x}^i) \neq \emptyset$ then
20:             Obtain a follower response $\mathbf{y}^i_f \in \Psi^b(\mathbf{x}^i)$
21:         else
22:             Obtain a follower response $\mathbf{y}^i_f \in \Psi^p(\mathbf{x}^i)$
23:             if $\phi^f(\mathbf{x}^i,\mathbf{y}^i_l) = \phi^f(\mathbf{x}^i,\mathbf{y}^i_f)$ and $\phi^l(\mathbf{x}^i,\mathbf{y}^i_l) = \phi^l(\mathbf{x}^i,\mathbf{y}^i_f)$ then
24:                 Update $LB_i = UB_i$ and the incumbent solution $(\bar{\mathbf{x}},\bar{\mathbf{y}}) \leftarrow (\mathbf{x}^i,\mathbf{y}^i_l)$
25:             else if $\phi^l(\mathbf{x}^i,\mathbf{y}^i_f) > LB_{i-1}$ then
26:                 Update $LB_i = \phi^l(\mathbf{x}^i,\mathbf{y}^i_f)$ and the incumbent solution $(\bar{\mathbf{x}},\bar{\mathbf{y}}) \leftarrow (\mathbf{x}^i,\mathbf{y}^i_f)$
27:             end if
28:         end if
29:         Set $\hat{\mathcal{Y}}^{i+1} = \hat{\mathcal{Y}}^i \cup \{\mathbf{y}^i_f\}$
30:     end if
31: end while
32: Return $(\bar{\mathbf{x}},\bar{\mathbf{y}})$
```

Phase two (lines 13–31) is a straightforward extension of Algorithm 1 in which the REHPP($\mathcal{Y}^i$) is replaced by the RPEHPP($\mathcal{Y}^i$) and follower responses to leader solutions $\mathbf{x}^i$ are obtained from $\Psi^b(\mathbf{x}^i)$ or $\Psi^p(\mathbf{x}^i)$. Note that if the optimistic optimal solution found with Algorithm 1 in line 2 also solves the problem under the pessimistic assumption, then $LB_0 = UB_0$ after line 12 and Algorithm 2 terminates without executing phase two.

A similar argument to the one used in Proposition 4 proves that Algorithm 2 terminates finitely with an optimal solution.

Definitions 4 and 5 provide the concept of dominance and alternative solutions for the pessimistic formulation.

Definition 4. *Consider solutions* $\mathbf{y}, \mathbf{y}' \in \mathcal{Y}$. *If* $\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \mathbf{y}')$ *for all* $\mathbf{x} \in \Omega(\mathcal{X})$, $h_j^2(\mathbf{y}) \leq h_j^2(\mathbf{y}')$ *for all* $j = 1, \ldots, m_2$, $\phi^l(\mathbf{x}, \mathbf{y}) \leq \phi^l(\mathbf{x}, \mathbf{y}')$ *for all* $\mathbf{x} \in \Omega(\mathcal{X})$, *and* $h_j^1(\mathbf{y}) \geq h_j^1(\mathbf{y}')$ *for all* $j = 1, \ldots, m_1$, *with at least one inequality being strict (among all sets of inequalities), then* $\mathbf{y}$ *dominates* $\mathbf{y}'$.

Definition 5. *Consider solutions* $\mathbf{y}, \mathbf{y}' \in \mathcal{Y}$ *such that* $\mathbf{y} \neq \mathbf{y}'$. *If* $\phi^f(\mathbf{x}, \mathbf{y}) = \phi^f(\mathbf{x}, \mathbf{y}')$ *for all* $\mathbf{x} \in \Omega(\mathcal{X})$, $h_j^2(\mathbf{y}) = h_j^2(\mathbf{y}')$ *for all* $j = 1, \ldots, m_2$, $\phi^l(\mathbf{x}, \mathbf{y}) = \phi^l(\mathbf{x}, \mathbf{y}')$ *for all* $\mathbf{x} \in \Omega(\mathcal{X})$, *and* $h_j^1(\mathbf{y}) = h_j^1(\mathbf{y}')$ *for all* $j = 1, \ldots, m_1$, *then* $\mathbf{y}$ *and* $\mathbf{y}'$ *are said to be alternative solutions.*

Definition 4 requires several conditions in order to establish dominance between two follower solutions. Accordingly, dominance relationships are less likely to occur in the pessimistic formulation. A similar argument to the one used in Proposition 5 proves that for any sample $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$, removing dominated solutions does not change the optimal objective function value of RPEHPP($\hat{\mathcal{Y}}$).

Remark 5. *If we do not assume the existence of a minimum follower's objective difference* $\delta > 0$, *then problem PEHPP is no longer equivalent to the pessimistic BMIP, because* (2.26d) *is not valid for* $\delta > 0$, *and is not sufficient to correctly define the* $\mathbf{v}$*-variables if* $\delta = 0$. *Alternatively, even if* δ *exists, it might be difficult to obtain, or its value might be so small as to introduce computational instability in the model (similar to the potential problems encountered in obtaining and using the big-M values). For this case we instead propose a cutting-plane algorithm, described in Algorithm 3.*

Algorithm 3 A Cutting-Plane Algorithm for the Pessimistic Variation of the BMIP

```
1: Set $LB_0 = -\infty$ and $i = 0$. Initialize the set of cutting planes $\mathcal{C} = \emptyset$ and incumbent solution $\bar{\mathbf{x}} = \bar{\mathbf{y}} = \emptyset$.
2: Set $i = i + 1$. Solve the optimistic version of the problem augmented with constraints in $\mathcal{C}$ using Algorithm 1. If the problem is infeasible, then go to Step 5. Otherwise, let $UB_i$ be the optimal objective function value obtained for this problem, and record the optimal leader solution $\hat{\mathbf{x}}$ found.
3: Compute an optimal pessimistic follower response, $\hat{\mathbf{y}}$, given $\hat{\mathbf{x}}$. If $\hat{\mathbf{y}} \notin \Psi^b(\mathbf{x})$, then $\phi^l(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is a lower bound on the pessimistic objective. If $\phi^l(\hat{\mathbf{x}}, \hat{\mathbf{y}}) > LB_{i-1}$, then set $LB_i = \phi^l(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ and update incumbent $(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = (\hat{\mathbf{x}}, \hat{\mathbf{y}})$. Otherwise, set $LB_i = LB_{i-1}$.
4: If $LB_i = UB_i$, then go to Step 5. Otherwise, add a *no-good constraint* set [Balas and Jeroslow, 1972] to $\mathcal{C}$. This constraint set is constructed so that $\hat{\mathbf{x}}$ is the only solution in $\Omega(\mathcal{X})$ that is infeasible to the constraints. Return to Step 2.
5: If the incumbent $(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = (\emptyset, \emptyset)$, then terminate and conclude that the pessimistic BMIP is infeasible. Otherwise, terminate with an optimal pessimistic BMIP solution given by $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$.
```

Two notes conclude this remark. One, the exactness and finiteness of our cutting-plane algorithm then follows from the finiteness of $\Omega(\mathcal{X})$. Two, although the no-good constraint of Balas and Jeroslow [1972] is intended for binary $\mathbf{x}$*-variables, we can accommodate the case of general integer* $\mathbf{x}$*-variables by replacing each variable* x_i *with the expression* $\sum_{j=1}^{k}(2^j)x_i^j$*, where* $x_i^1, \ldots, x_i^k$ *are binary variables, and* k *is a sufficiently large number. The no-good constraints can then be written for the binary* x_i^j*-variables instead of the general* $\mathbf{x}$*-variables.*

2.5 Featured Study on Competitive Scheduling

We consider a competitive single-machine scheduling problem, in which two agents, a leader and a follower, each provide a permutation ordering of their own set of jobs. Both agents have the goal of maximizing the number of jobs that complete on or before their due date. The agents' jobs are placed on the machine by an independent central operator. This operator receives the job permutations selected by the agents, and alternates the jobs sequentially on the machine, starting with the leader's first job, the follower's first job, the leader's second job, and so on with no machine idle time until all jobs are scheduled. Define n as the total number of leader (follower) jobs, p_i^l (p_i^f) as the processing time for leader (follower) job i, and d_i^l (d_i^f) as the due date for leader (follower) job i. A job is considered to be on-time if its completion time is less than or equal to its due date. Let x_{ik} (y_{ik}) be a binary variable that takes a value of 1 if a leader (follower) job $i \in \{1, \ldots, n\}$ is the k^{th} job scheduled by the leader (follower), for $k = 1, \ldots, n$. Define $\phi^l(\mathbf{x}, \mathbf{y})$ ($\phi^f(\mathbf{x}, \mathbf{y})$) as the number of on-time jobs for the leader (follower). We formulate the competitive single-machine scheduling problem under the pessimistic assumption as:

$$\begin{aligned}
\max \quad & \phi^l(\mathbf{x}, \mathbf{y}) && && \text{(2.29a)}\\
\text{s.t.} \quad & \sum_{k=1}^{n} x_{ik} = 1 && \forall i = 1, \ldots, n && \text{(2.29b)}\\
& \sum_{i=1}^{n} x_{ik} = 1 && \forall k = 1, \ldots, n && \text{(2.29c)}\\
& \mathbf{y} \in \Psi^p(\mathbf{x}) && && \text{(2.29d)}\\
& \mathbf{x} \in \{0,1\}^{n \times n}. && && \text{(2.29e)}
\end{aligned}$$

The objective function (2.29a) maximizes the number of the leader's on-time jobs. Constraints (2.29b) ensure that every leader job is assigned to a position while constraints (2.29c) enforce that every position has an assigned job. Constraints (2.29d) require the follower's response to be in the rational pessimistic reaction set, where the pessimistic assumption can easily be relaxed to the optimistic assumption if desired. Constraints (2.29e) restrict the $\mathbf{x}$-variables to be binary-valued. Formally, $\Psi^p(\mathbf{x}) = \underset{\mathbf{y}\in\Psi(\mathbf{x})}{\operatorname{argmin}}\{\phi^l(\mathbf{x},\mathbf{y})\}$, where $\Psi(\mathbf{x})$ is the set of all optimal solutions to:

$$\begin{aligned}
\max \quad & \phi^f(\mathbf{x},\mathbf{y}) && && (2.30\text{a})\\
\text{s.t.} \quad & \sum_{k=1}^{n} y_{ik} = 1 && \forall i = 1,\dots,n && (2.30\text{b})\\
& \sum_{i=1}^{n} y_{ik} = 1 && \forall k = 1,\dots,n && (2.30\text{c})\\
& \mathbf{y} \in \{0,1\}^{n\times n}. && && (2.30\text{d})
\end{aligned}$$

The problem of selecting a follower response $\mathbf{y} \in \Psi^p(\mathbf{x})$ that minimizes $\phi^l(\mathbf{x},\mathbf{y})$ for a given leader solution $\mathbf{x} \in \Omega(\mathcal{X})$ is $\mathcal{NP}$-hard, as we now show.

Denote by FSP the decision version of the follower's subproblem, which seeks a schedule that maximizes the number of on-time follower jobs, while ensuring that no more than τ leader jobs are on time.

Proposition 7. *FSP is $\mathcal{NP}$-complete.*

Proof We show that FSP is $\mathcal{NP}$-hard by using a transformation from EQUIPARTITION (EP) stated as follows: Given $2k$ positive integers $a_1,\dots,a_{2k}$ such that $\sum_{i=1}^{2k} a_i = 2b$, does there exist a set $\mathcal{S} \subseteq \{1,\dots,2k\}$ such that $|\mathcal{S}| = k$ and $\sum_{i\in\mathcal{S}} a_i = b$? To transform an EP instance into an FSP instance, we set the number of jobs $n = 2k$, the leader processing times $p_i^l = 1$ for all $i = 1,\dots,n$, the follower processing times $p_i^f = a_i$ for all $i = 1,\dots,n$, the due dates $d_i^l = d_i^f = b + k$ for all $i = 1,\dots,n$, and the objective target $\tau = k$. We assume that the sum of the smallest $k+1$ integers a_i is greater than b, noting that EP remains $\mathcal{NP}$-complete under this assumption. (The latter claim holds true because any EP instance can be transformed to satisfy this assumption by rescaling the integers a_i.) Note that this transformation is polynomial, and so it is now sufficient to prove that EP has a solution if and only if FSP has a solution.

In the following discussion, note that the leader's jobs are equivalently scheduled in any order. The follower always maximizes its objective by scheduling k jobs on time. This is because after k jobs have been scheduled by both agents, the leader has used k units of processing time on the machine, while the follower has b units available on the machine for processing. The follower can achieve at least k on-time jobs by scheduling the k-smallest-processing-time jobs first, in any order. By assumption, no set of $k+1$ jobs can be scheduled by the follower within its allotted b units of time before the deadline.

Assume that EP has a solution $\mathcal{S}$. We construct a follower solution to the FSP by scheduling the k jobs indexed in $\mathcal{S}$ first and the remaining jobs later. Let C_k^f (C_k^l) denote the completion time for the follower (leader) job scheduled at position k. Because $\sum_{i\in\mathcal{S}} p_i^f = b$ and $p_i^l = 1$ for all $i = 1, \ldots, n$, we get that $C_k^f = b + k$, $C_k^l < b + k$, and $C_{k+1}^l > b + k$. This implies that the leader has exactly k on-time jobs. The follower also obtains k on-time jobs, which is optimal. The solution constructed therefore solves FSP.

Now assume that FSP has a solution. We construct a solution to EP by including in $\mathcal{S}$ the first k jobs scheduled by the follower. Note that $C_k^f \leq b + k$ since the optimal number of follower on-time jobs is equal to k. However $C_k^f < b+k$ is impossible, or else $C_{k+1}^l \leq b+k$, which contradicts the assumption that the leader has no more than k on-time jobs. The fact that $C_k^f = b + k$ implies that $\sum_{i\in\mathcal{S}} a_i = \sum_{i\in\mathcal{S}} p_i^f = b$ because $p_i^l = 1$ for all $i = 1, \ldots, n$. Therefore $\mathcal{S}$ is a solution to EP. ■

We now present the PEHPP for the competitive single-machine scheduling problem. In models (2.29) and (2.30) the leader is not able to block any follower solution and vice versa. This implies that $\Psi^b(\mathbf{x}) = \emptyset$ for all $\mathbf{x} \in \Omega(\mathcal{X})$ and that the $\mathbf{w}$-variables can be removed from the PEHPP, yielding the following model.

$$\max_{(\mathbf{x},\mathbf{y})} \quad \phi^l(\mathbf{x},\mathbf{y}) \tag{2.31a}$$

$$\text{s.t.} \quad \phi^f(\mathbf{x},\mathbf{y}) + v_{\hat{\mathbf{y}}} \geq \phi^f(\mathbf{x},\hat{\mathbf{y}}) + 1 \qquad \forall \hat{\mathbf{y}} \in \mathcal{Y} \tag{2.31b}$$

$$\phi^l(\mathbf{x},\mathbf{y}) \leq \phi^l(\mathbf{x},\hat{\mathbf{y}}) + (n-1)(1 - v_{\hat{\mathbf{y}}}) \qquad \forall \hat{\mathbf{y}} \in \mathcal{Y} \tag{2.31c}$$

$$(\mathbf{x},\mathbf{y}) \in \Omega \tag{2.31d}$$

$$v_{\hat{\mathbf{y}}} \in \{0,1\} \qquad \forall \hat{\mathbf{y}} \in \mathcal{Y}. \tag{2.31e}$$

The objective function (2.31a) maximizes the number of leader on-time jobs. Constraints (2.31b)–(2.31e) enforce the bilevel feasibility conditions established in Proposition 6. Note that (2.31b) implies that $\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}})$, for all $\hat{\mathbf{y}} \in \mathcal{Y}$; furthermore, if $\phi^f(\mathbf{x}, \mathbf{y}) = \phi^f(\mathbf{x}, \hat{\mathbf{y}})$, then $v_{\hat{\mathbf{y}}} = 1$. When $v_{\hat{\mathbf{y}}} = 1$, (2.31c) ensures that $\phi^l(\mathbf{x}, \mathbf{y}) \leq \phi^l(\mathbf{x}, \hat{\mathbf{y}})$ as desired by the pessimistic assumption, and otherwise the constraint becomes redundant.

Models (2.29)–(2.31) are difficult to solve because of the nonlinear functions $\phi^l(\mathbf{x}, \mathbf{y})$ and $\phi^f(\mathbf{x}, \mathbf{y})$ used in those formulations. In order to optimize these problems, we can use any of the various strategies proposed in the literature for single-machine scheduling, such as mixed-integer linear programming (MILP), constraint programming, or other implicit enumeration strategies. We describe a MILP-based approach in Section 2.5.1 for solving these problems along with the RPEHPP($\hat{\mathcal{Y}}$) model. These models require many additional variables and constraints to linearize the model, including several auxiliary binary variables. This approach highlights an important distinction in how we have stated the scheduling problem in this section. Models (2.29)–(2.31) themselves do not employ the additional auxiliary variables used in our MILP formulations, which are only used as tools to yield optimal solutions for models (2.30) and (2.31). Therefore, because the follower is solving (2.30) and the leader is solving (2.31) (although indirectly), no agent can block another agent's solution due to the absence of $\mathbf{x}$ in the constraints for (2.30) and the absence of $\mathbf{y}$ in the constraints for (2.29) (excluding the condition that $\mathbf{y} \in \Psi^p(\mathbf{x})$). As a result, all M-values for this model (see Section 2.6.5 for details) are very small, thus obviating potential issues with numerical instability due to their use.

On the contrary, if we had previously *restricted our analysis to BMILPs*, then our models would necessarily include auxiliary variables to linearize the problems. The leader and follower problems detailed in Section 2.5.1 include (among others) binary variables s_k^l that equal 1 if the leader job in position k will be on time, and 0 otherwise. The leader is therefore forced to declare specifically which jobs will be on time. However, a follower can now force such a solution to become infeasible, simply by identifying a follower schedule that makes a single leader job k late, among those positions for which the leader had set $s_k^l = 1$. This blocking of solutions is artificial, and would require additional iterations of our algorithm in which various leader schedules and vectors of on-time leader jobs are blocked by the follower, and vice versa. This situation is similar to the difficulties faced within robust optimization, in which the presence of auxiliary variables complicates the development of useful robust optimization counterparts [Delage and Iancu, 2015, Gorissen and den Hertog, 2013].

2.5.1 Competitive Scheduling Formulations

We now present MILP formulations that we employ in our algorithm to solve the competitive scheduling problem described in Section 2.5. We assume without loss of generality that p_i^l, p_i^f, d_i^l, and d_i^f are integer-valued for all $i = 1, \dots, n$. To obtain a follower response for a given leader solution $\bar{\mathbf{x}}$, we first compute the follower's optimal number of on-time jobs by solving $z_f(\bar{\mathbf{x}}) = \max_{\mathbf{y}}\{\phi^f(\bar{\mathbf{x}}, \mathbf{y}) \mid \mathbf{y} \in \mathcal{Y}(\bar{\mathbf{x}})\}$. We solve this problem in polynomial time using Moore's earliest due date algorithm [Moore, 1968], which is still valid when considering the fixed processing times for the leader jobs scheduled by $\bar{\mathbf{x}}$. Note that for the optimistic formulation, obtaining $z^f(\bar{\mathbf{x}})$ via Moore's algorithm suffices to identify a follower response $\mathbf{y} \in \Psi(\bar{\mathbf{x}})$. However, obtaining a pessimistic response $\mathbf{y} \in \Psi^p(\bar{\mathbf{x}})$ is $\mathcal{NP}$-hard as already shown.

To find a pessimistic response, the follower seeks a schedule $\mathbf{y} \in \Psi^p(\bar{\mathbf{x}})$ that minimizes the leader's objective among all solutions that schedule $z^f(\bar{\mathbf{x}})$ on-time follower jobs. Let C_k^f (C_k^l) denote the completion time for the follower (leader) job scheduled in position k. Define a binary variable s_k^f for every schedule position $k = 1, \dots, n$, such that $s_k^f = 1$ if C_k^f is less than or equal to the due date of the follower's job scheduled in the k^{th} position. Let binary variable s_k^l be such that $s_k^l = 1$ if C_k^l is less than or equal to the due date of the leader's job scheduled in position k, for every position $k = 1, \dots, n$. We obtain $\mathbf{y} \in \Psi^p(\bar{\mathbf{x}})$ by solving the following problem, where $\bar{T}$-values are large constants whose values we discuss subsequently:

$$\min \quad \sum_{k=1}^{n} s_k^l \tag{2.32a}$$

$$\text{s.t.} \quad \mathbf{y} \in \mathcal{Y}(\bar{\mathbf{x}}) \tag{2.32b}$$

$$\sum_{i=1}^{n}\sum_{q=1}^{k} p_i^l \bar{x}_{iq} + \sum_{i=1}^{n}\sum_{q=1}^{k} p_i^f y_{iq} - \sum_{i=1}^{n} d_i^f y_{ik} \leq T_k^f(1 - s_k^f) \qquad \forall k = 1, \dots, n \tag{2.32c}$$

$$\sum_{k=1}^{n} s_k^f = z^f(\bar{\mathbf{x}}) \tag{2.32d}$$

$$\sum_{i=1}^{n}\sum_{q=1}^{k} p_i^l \bar{x}_{iq} + \sum_{i=1}^{n}\sum_{q=1}^{k-1} p_i^f y_{iq} - \sum_{i=1}^{n} d_i^l \bar{x}_{ik} \geq -\bar{T}_{k\bar{\mathbf{x}}}^l s_k^l + 1 \qquad \forall k = 1, \dots, n \tag{2.32e}$$

$$\mathbf{s}^f, \mathbf{s}^l \in \{0,1\}^n. \tag{2.32f}$$

The objective function (2.32a) minimizes the number of leader on-time jobs. Constraints (2.32b)–(2.32d) enforce optimality restrictions for the follower. In particular, the first two terms on the left-hand side of (2.32c) compute C_k^f, while the third term states the due date of the follower job in position k. If the left-hand side of (2.32c) is positive, then s_k^f is forced to equal 0; otherwise, s_k^f will equal 1 at optimality. Thus, T_k^f must be at least as large as the largest possible value for the left-hand side of (2.32c), i.e., the maximum amount of time by which the k^{th} job scheduled by the follower could be late. To find this value, let U_k^l (U_k^f) be the sum of processing times for the leader's (follower's) k-largest-processing-time jobs. We can then find $i \in \underset{\hat{\imath}\in\{1,\ldots,n\}}{\operatorname{argmax}}\{p_{\hat{\imath}}^f - d_{\hat{\imath}}^f\}$ and set

$$T_k^f = \begin{cases} U_k^l + U_{k-1}^f + p_i^f - d_i^f & \text{if } p_i^f \text{ is not among the} \\ & \text{follower's } (k-1)\text{-largest-processing-time jobs} \\ U_k^l + U_k^f - d_i^f & \text{otherwise.} \end{cases} \qquad \forall k = 1,\ldots,n$$

Constraints (2.32e) define the $\mathbf{s}^l$-variables, and constraints (2.32f) restrict variables to be binary-valued. Observe now that optimality forces s_k^l to its smallest possible value, but (2.32e) forces s_k^l to equal 1 when the leader job in position k is on time. To compute the $\bar{T}$-values let L_k^f be the sum of processing times for the follower's k-smallest-processing-time jobs. We set

$$\bar{T}_{k\bar{\mathbf{x}}}^l = \sum_{i=1}^n d_i^l \bar{x}_{ik} - \sum_{i=1}^n \sum_{q=1}^k p_i^l \bar{x}_{iq} - L_{k-1}^f + 1.$$

Using these $\bar{T}$-values, s_k^l must equal 1 whenever the left-hand side of (2.32e) is nonpositive; moreover, the left-hand side of (2.32e) is never less than $-\bar{T}_{k\bar{\mathbf{x}}}^l s_k^l + 1$, which establishes the validity of that constraint. When the left-hand side of (2.32e) is positive, then that value must be at least one because of our data integrality assumption, and $s_k^l = 0$ at optimality as desired.

We now formulate RPEHPP($\hat{\mathcal{Y}}$). We begin by defining T-values that will be used in this model, analogous to those used before. First, let T_k^l be the maximum amount of time by which a leader job scheduled in position k could be late. As before for T_k^f, letting $i \in \underset{\hat{\imath}\in\{1,\ldots,n\}}{\operatorname{argmax}}\{p_{\hat{\imath}}^l - d_{\hat{\imath}}^l\}$, we set

$$T_k^l = \begin{cases} U_{k-1}^l + U_{k-1}^f + p_i^l - d_i^l & \text{if } p_i^l \text{ is not among the} \\ & \text{leader's } (k-1)\text{-largest-processing-time jobs} \\ U_k^l + U_{k-1}^f - d_i^l & \text{otherwise.} \end{cases} \qquad \forall k = 1,\ldots,n$$

Now, define $\left(\bar{T}^f_{k\hat{\mathbf{y}}} - 1\right)$ as the maximum amount of time by which the follower job scheduled in position k by solution $\hat{\mathbf{y}}$ could be early. Let L^l_k be the sum of processing times for the leader's k-smallest-processing-time jobs. For every $\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$ we set

$$\left(\bar{T}^f_{k\hat{\mathbf{y}}} - 1\right) = \sum_{i=1}^{n} d^f_i \hat{y}_{ik} - \sum_{i=1}^{n}\sum_{q=1}^{k} p^f_i \hat{y}_{iq} - L^l_k.$$

Define binary variables $\hat{\mathbf{s}}^l$ such that $\hat{s}^l_{k\hat{\mathbf{y}}} = 1$ if C^l_k, measured with respect to follower solution $\hat{\mathbf{y}}$, is less than or equal to the due date of the leader job scheduled at position k, and $\hat{s}^l_{k\hat{\mathbf{y}}} = 0$ otherwise. Define also binary variables $\hat{\mathbf{s}}^f$ such that $\hat{s}^f_{k\hat{\mathbf{y}}} = 1$ if the job scheduled at position k by follower solution $\hat{\mathbf{y}}$ is on time, and $\hat{s}^f_{k\hat{\mathbf{y}}} = 0$ otherwise. We formulate RPEHPP($\hat{\mathcal{Y}}$) for the competitive scheduling problem as follows.

$$\max \quad \sum_{k=1}^{n} s^l_k \tag{2.33a}$$

$$\text{s.t.} \quad (\mathbf{x}, \mathbf{y}) \in \Omega \tag{2.33b}$$

$$\sum_{i=1}^{n}\sum_{q=1}^{k} p^l_i x_{iq} + \sum_{i=1}^{n}\sum_{q=1}^{k-1} p^f_i y_{iq} - \sum_{i=1}^{n} d^l_i x_{ik} \le T^l_k(1 - s^l_k) \qquad \forall k = 1, \ldots, n \tag{2.33c}$$

$$\sum_{i=1}^{n}\sum_{q=1}^{k} p^l_i x_{iq} + \sum_{i=1}^{n}\sum_{q=1}^{k} p^f_i y_{iq} - \sum_{i=1}^{n} d^f_i y_{ik} \le T^f_k(1 - s^f_k) \qquad \forall k = 1, \ldots, n \tag{2.33d}$$

$$\sum_{i=1}^{n}\sum_{q=1}^{k} p^l_i x_{iq} + \sum_{i=1}^{n}\sum_{q=1}^{k} p^f_i \hat{y}_{iq} - \sum_{i=1}^{n} d^f_i \hat{y}_{ik} \ge -\bar{T}^f_{k\hat{\mathbf{y}}} \hat{s}^f_{k\hat{\mathbf{y}}} + 1 \qquad \forall \hat{\mathbf{y}} \in \hat{\mathcal{Y}},\ k = 1, \ldots, n \tag{2.33e}$$

$$\sum_{k=1}^{n} s^f_k + v_{\hat{\mathbf{y}}} \ge \sum_{k=1}^{n} \hat{s}^f_{k\hat{\mathbf{y}}} + 1 \qquad \forall \hat{\mathbf{y}} \in \hat{\mathcal{Y}} \tag{2.33f}$$

$$\sum_{i=1}^{n}\sum_{q=1}^{k} p^l_i x_{iq} + \sum_{i=1}^{n}\sum_{q=1}^{k-1} p^f_i \hat{y}_{iq} - \sum_{i=1}^{n} d^l_i x_{ik} \le T^l_k(1 - \hat{s}^l_{k\hat{\mathbf{y}}}) \qquad \forall \hat{\mathbf{y}} \in \hat{\mathcal{Y}},\ k = 1, \ldots, n \tag{2.33g}$$

$$\sum_{k=1}^{n} s^l_k \le \sum_{k=1}^{n} \hat{s}^l_{k\hat{\mathbf{y}}} + (n-1)(1 - v_{\hat{\mathbf{y}}}) \qquad \forall \hat{\mathbf{y}} \in \hat{\mathcal{Y}} \tag{2.33h}$$

$$\mathbf{v} \in \{0,1\}^{|\hat{\mathcal{Y}}|} \tag{2.33i}$$

$$\mathbf{s}^l, \mathbf{s}^f \in \{0,1\}^n \tag{2.33j}$$

$$\hat{\mathbf{s}}^l, \hat{\mathbf{s}}^f \in \{0,1\}^{n \times |\hat{\mathcal{Y}}|}. \tag{2.33k}$$

The objective function (2.33a) maximizes the number of leader on-time jobs. Constraints (2.33b) ensure that every job is assigned to a position and every position has an assigned job. Constraints (2.33c) ensure that $s^l_k = 0$ if C^l_k is greater than the due date of the leader job scheduled at position k. Constraints (2.33d) enforce the same condition for follower variables $\mathbf{s}^f$. Constraints (2.33e) define the $\hat{\mathbf{s}}^f$-variables. Constraints (2.33f) define the $\mathbf{v}$-variables and enforce the condition that $\phi^f(\mathbf{x}, \mathbf{y}) \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}})$ for all $\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$. Constraints (2.33g) define the $\hat{\mathbf{s}}^l$-variables. Constraints (2.33h) then enforce that $\phi^l(\mathbf{x}, \mathbf{y}) \leq \phi^l(\mathbf{x}, \hat{\mathbf{y}})$ unless $\hat{\mathbf{y}} \notin \Psi(\mathbf{x})$. Constraints (2.33i)–(2.33k) require all variables to be binary-valued.

2.6 Computational Experiments

In Section 2.6.1 we analyze the effect of the proposed acceleration strategies on the performance of the algorithm. In Section 2.6.2 we compare our algorithm's performance with the current state-of-the-art approach for BMILPs by Xu and Wang [2014]. In Section 2.6.3 we present computational results on the extension to the pessimistic formulation. Finally, in Section 2.6.4 we report the results on the focus application in Section 2.5.

We coded our algorithm in Java, using Eclipse SDK version 4.4.2, and executed the experiments on a machine having an Intel Core i7–3537U CPU (two cores) running at 2.00 GHz with 8 GB of RAM on Windows 8. We solve all optimization problems using CPLEX 12.6. All instances and source code used in this section are available at `http://people.clemson.edu/~jcsmith`.

Our core set of test instances consists of the BMILP testbed provided by Xu and Wang [2014]. This dataset contains 100 instances with sizes ranging from 20 to 920 variables, and 8 to 368 constraints (10 instances for each size). In every instance $n_1 = n_2$, $m_1 = m_2 = 0.4n_1$, and 50% of the follower variables are randomly included in set $\mathcal{I}$. The cost coefficients are random integers uniformly distributed between $[-50, 50]$, the right-hand side coefficients b_1 (b_2) are random integers uniformly distributed between $[30, 130]$ ($[10, 110]$), and all other coefficients are random integers uniformly distributed between $[0, 10]$. Note that both the instances in Xu and Wang [2014] and the competitive set covering instances satisfy the three assumptions listed in Section 2.1, including the boundedness requirement in Assumption 1.

2.6.1 Assessing the Effectiveness of the Proposed Acceleration Strategies

We conducted experiments to measure the effect of the proposed acceleration strategies on the performance of the algorithm. Section 2.6.1.1 discusses the results for the variable fixing procedure, and Section 2.6.1.2 discusses those for the four sets of proposed SVIs.

2.6.1.1 Variable Fixing

We compared the performance of our proposed algorithm with and without the variable fixing technique proposed in Section 2.3.1. After tuning the algorithm parameters, we set a maximum initial sample size limit of $250,000$ follower solutions.

Table 2.1 shows the results for these experiments. The first two columns show the total number of variables ($n = n_1 + n_2$) and constraints ($m = m_1 + m_2$), respectively. Columns 3–6 show the average CPU time in seconds obtained over 10 instances with the same size (Avg) and the largest CPU time obtained over those runs (Max), for both algorithms. Column 7 presents the speedup measured as the ratio between the average execution times reported in columns 3 and 5.

Table 2.1: Assessing the impact of the variable fixing procedure

n	m	No fixing		Variable fixing		Speedup
		Avg	Max	Avg	Max	
20	8	1.5	5.6	1.4	5.5	1.0
120	48	21.1	60.4	16.9	50.7	1.2
220	88	70.3	634.5	60.4	552.7	1.2
320	128	29.0	68.5	20.1	47.0	1.4
420	168	124.4	754.1	74.4	488.6	1.7
520	208	232.5	1034.1	92.8	285.8	2.5
620	248	678.2	1679.3	207.4	561.9	3.3
720	288	861.4	6034.6	199.0	930.5	4.3
820	328	513.5	2042.3	248.6	922.6	2.1
920	368	1193.0	4359.8	465.5	872.1	2.6

All computational times in CPU seconds

Table 2.1 shows that using the variable fixing procedure reduces the average CPU times over all instance sizes. This time reduction is more evident in the larger instances; in particular, the algorithm with variable fixing runs up to 4.3 times faster than the original algorithm for instances with size $n = 720$. The improvement in the maximum CPU times is also consistent across all instance sizes. For the instances with $n = 920$ the worst execution time is reduced from roughly 4000 seconds to under 900 seconds by the variable fixing procedure.

2.6.1.2 SVIs

We compared six different versions of the proposed sampling algorithm, each of which includes the variable fixing technique. The first one does not include any SVIs. The next four versions include one by one each of the proposed SVI sets. The last version includes the first and second SVI sets. Table 2.2 shows the results for these experiments.

Table 2.2: Assessing the effect of the SVIs on the performance of the algorithm

n	m	No SVIs		SVIs 1		SVIs 2		SVIs 3		SVIs 4		SVIs 1 & 2	
		Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
20	8	1.4	5.5	1.5	5.7	1.4	5.3	1.6	5.5	1.7	6.7	1.5	5.6
120	48	16.9	50.7	16.5	51.7	16.7	52.8	18.5	54.9	18.7	55.6	16.3	49.4
220	88	60.4	552.7	59.1	545.7	58.9	540.1	62.0	568.0	61.4	561.2	56.8	521.0
320	128	20.1	47.0	17.8	43.5	19.9	48.1	20.5	48.5	22.4	62.7	17.8	42.8
420	168	74.4	488.6	69.6	463.3	78.5	520.9	87.7	602.8	87.8	565.8	69.1	463.5
520	208	92.8	285.8	78.5	210.7	86.2	254.4	97.2	292.9	126.8	375.9	75.5	186.0
620	248	207.4	561.9	156.1	445.8	204.2	484.7	215.4	562.0	361.3	899.2	152.9	443.5
720	288	199.0	930.5	162.5	739.9	200.2	915.3	195.5	888.8	337.5	1221.0	163.4	728.7
820	328	248.6	922.6	159.6	443.1	273.3	1211.7	265.1	970.9	362.3	874.5	157.9	443.7
920	368	465.5	872.1	280.0	529.9	397.3	763.5	550.4	1475.4	939.4	2651.6	264.4	438.7

Table 2.2 shows that when measuring each set of SVIs individually, the greatest reduction in execution times is achieved by the first set, followed by the second one. Including the third set of SVIs has a slight negative effect on the CPU times on most of the instances, and using the fourth set greatly increases the execution times. These SVIs perform poorly due to the large number of linear programming problems that must be solved to obtain the necessary upper bounds for these SVIs, especially over the larger instances. Using both the first and second sets of SVIs leads to the most efficient version of the algorithm.

2.6.2 Comparison with Xu and Wang [2014]

We conducted a comparison between our algorithm (Sampling) and the branch-and-bound algorithm (BB) by Xu and Wang [2014]. We chose the BB approach due to its impressive performance in the problems discussed by Xu and Wang [2014]. Naturally, for different problem classes, some of the other algorithms cited in Section 2.1 would be appropriate to examine here as well. Because a full computational comparison of a wide array of modern bilevel algorithms is beyond the scope of this study, the goal in this section is to demonstrate the advantages of our proposed algorithm with respect to a current state-of-the-art bilevel programming algorithm.

For completeness, we note that the approach in Xu and Wang [2014] does not place bounds on $\mathbf{y}$, whereas the approach in our algorithm requires these bounds to guarantee convergence. However, the problems in the BMILP testbed allow us to bound the $\mathbf{y}$ vector, allowing a direct comparison of the approaches over these instances.

We coded the BB algorithm in Java and found that the run times obtained with our BB implementation were roughly two times faster than those reported by Xu and Wang [2014], due to our use of a newer version of CPLEX on a faster computer. Table 2.3 presents computational results comparing the two algorithms. For our algorithm we included the proposed variable fixing technique along with the first and second sets of SVIs. Columns 1–6 are defined as before. Column 7 presents the speedup calculated as the ratio between the BB and our algorithm average times.

Table 2.3: Comparing the sampling algorithm to the state-of-the-art algorithm for BMILP

n	m	BB		Sampling		Speedup
		Avg	Max	Avg	Max	
20	8	0.9	2.8	1.5	5.6	0.6
120	48	23.8	83.9	16.3	49.4	1.5
220	88	58.1	368.3	56.8	521.0	1.0
320	128	96.8	218.5	17.8	42.8	5.4
420	168	547.9	3932.4	69.1	463.5	7.9
520	208	547.5	1703.3	75.5	186.0	7.2
620	248	2152.3	6348.0	152.9	443.5	14.1
720	288	1623.1	3214.7	163.4	728.7	9.9
820	328	2178.6	8891.0	157.9	443.7	13.8
920	368	4544.2	9008.3	264.4	438.7	17.2

Table 2.3 shows that our sampling algorithm compares favorably to BB. Both algorithms solve the smaller instances in less than one minute on average. Over the medium-sized instances our sampling algorithm is faster than BB, exhibiting speedups ranging from 5.4 to 7.9. Over the larger instances the difference in performance between the algorithms is greater, and our algorithm achieves speedups ranging from 9.9 to 17.2. Regarding the maximum execution times, the sampling algorithm outperforms BB over almost all instances, except for instances with $n = 20$ and $n = 220$. Moreover, on the larger instances ($n = 820$ and 920) the maximum execution time is reduced by roughly 90%.

2.6.3 Extension to the Pessimistic Formulation

In this section we study our algorithm's performance under the pessimistic assumption. In the original Xu and Wang [2014] testbed, the optimal solution obtained under the optimistic assumption is also optimal for the pessimistic formulation for most of the instances. This behavior is due to the lack of alternative optimal solutions to the follower's problem, given a leader's optimal solution. We modified the testbed to increase the chances of getting alternative optimal solutions in the follower's problem by setting the follower's cost coefficients $d_i^2 = 1$ with probability α, and $d_i^2 = 0$ with probability $1 - \alpha$, for $i = 1, \ldots, n_2$.

Note that for these instances, Algorithm 2 provides heuristic solutions since we cannot establish the existence of a minimum follower's objective function difference $\delta > 0$. On the other hand, the cutting-plane algorithm in Remark 5 finds provable global optimal solutions. We set $\delta = 10^{-3}$ for solving RPEHPP($\hat{\mathcal{Y}}$) within Algorithm 2.

Tables 2.4 and 2.5 compare our algorithm's performance under the optimistic and pessimistic assumptions over the modified testbed with $\alpha = 0.15$ and $\alpha = 0.01$. The "Avg" and "Max" columns are defined as before, while columns "z^*", "z_p^h", and "z_p^*" show the average optimal objective function value for the optimistic formulation, the average heuristic objective function value for the pessimistic formulation with $\delta = 10^{-3}$, and the average optimal objective function value for the pessimistic formulations, respectively. The "# Cuts" column presents the average number of cuts added in the cutting-plane algorithm. The "Gap" column presents the average objective function value gap, calculated as $(z^* - z_p^*)/z^*$.

Table 2.4: Comparing the algorithm performance under the optimistic and pessimistic assumptions when $\alpha = 0.15$

		$\alpha = 0.15$										
n	m	Optimistic			Pessimistic Algorithm 2			Pessimistic cutting-plane				Gap
		Avg	Max	z^*	Avg	Max	z_p^h	Avg	Max	# Cuts	z_p^*	
20	8	1.0	8.0	244.7	1.2	7.2	201.2	170.7	1677.5	170.3	201.2	0.18
120	48	4.0	12.8	186.6	17.7	66.6	173.7	23.4	63.8	15.2	173.7	0.07
220	88	8.5	34.3	166.1	73.1	613.9	157.9	26.8	93.1	7.4	157.9	0.05
320	128	27.9	209.2	149.1	106.6	594.4	136.8	86.4	331.1	10.1	136.8	0.08
420	168	34.1	153.8	141.9	202.1	1186.3	137.5	118.9	501.3	6.4	137.5	0.03
520	208	37.9	161.1	134.8	79.2	296.5	130.5	75.6	327.9	4.7	130.5	0.03
620	248	85.2	205.6	134.5	169.7	627.8	130.8	122.2	368.8	4.6	130.8	0.03
720	288	85.0	195.6	126.7	110.9	296.9	125.2	94.9	248.3	1.7	125.2	0.01
820	328	123.2	318.0	114.1	422.6	1896.2	109.2	292.8	1082.4	6.0	109.2	0.04
920	368	129.1	190.3	112.8	161.3	310.1	112.4	142.9	214.8	0.7	112.4	<0.01

Table 2.5: Comparing the algorithm performance under the optimistic and pessimistic assumptions when $\alpha = 0.01$

		$\alpha = 0.01$										
n	m	Optimistic			Pessimistic Algorithm 2			Pessimistic cutting-plane				Gap
		Avg	Max	z^*	Avg	Max	z_p^h	Avg	Max	# Cuts	z_p^*	
20	8	1.0	10.0	281.1	1.7	11.3	179.2	335.4	2371.1	299.4	179.2	0.36
120	48	0.4	1.9	207.6	11.6	37.6	162.8	52.8	171.2	108.1	162.8	0.22
220	88	1.5	7.3	172.7	46.7	261.6	148.4	31.0	112.7	35.0	148.4	0.14
320	128	2.1	6.2	159.8	172.0	1542.9	132.8	209.9	1841.8	67.9	132.8	0.17
420	168	6.6	38.3	159.6	105.8	430.6	130.6	98.2	352.4	33.9	130.6	0.18
520	208	6.6	24.0	145.8	282.3	2182.6	120.6	153.0	416.4	33.2	120.6	0.17
620	248	6.7	19.0	148.8	162.3	442.0	126.3	208.2	914.7	39.5	126.3	0.15
720	288	8.7	21.4	140.0	176.8	729.9	118.9	146.9	619.4	19.7	118.9	0.15
820	328	12.2	28.3	121.2	578.2	3646.9	104.0	314.1	968.1	32.5	104.0	0.14
920	368	24.8	60.3	122.3	207.9	427.7	105.3	363.8	1288.9	35.6	105.3	0.14

Tables 2.4 and 2.5 show that it takes considerably longer to solve the instances under the pessimistic formulation both in terms of the average and the maximum execution times. Our proposed cutting-plane algorithm solves all instances to optimality, with the largest execution time of 2132.7 seconds occurring in the $n = 20$ group with $\alpha = 0.01$. The difference in the optimal objective values is greater on the smaller instances than on the larger ones. Furthermore, as α decreases, there is more potential for finding alternative optimal solutions in the follower's problem. Therefore, the gap between the optimal optimistic and pessimistic objectives increases.

As α decreases the optimistic formulation becomes easier to solve because of the increased number of alternative optimal follower solutions. Algorithm 2 tends to outperform the cutting-plane algorithm for instances having larger gaps, because larger gaps tend to require more iterations of the cutting-plane algorithm. However, for instances in which the gap is small, the cutting-plane algorithm adds very few cuts and quickly finds provable optimal solutions.

2.6.4 Featured Study on Competitive Scheduling

We generated random instances with sizes ranging from $n = 10$ to $n = 30$ in increments of five (10 instances for each size). In every instance the leader's processing times are random integers uniformly distributed in the range $[10, 20]$. The follower processing times are set to 10, 50, or 100 with equal probability (1/3). Let $T = \sum_{i=1}^{n}(p_i^l + p_i^f)$ be the total time required to process all the jobs. For any job i, the leader's (follower's) due date is a random integer uniformly distributed between $[p_i^l, 0.5T]$ ($[p_i^f, 0.2T]$). Because the follower's objective is integer-valued, using $\delta = 1$ within Algorithm 2 guarantees optimality. We thus choose this pessimistic algorithm in the computational experiments on competitive scheduling problems.

Table 2.6 compares our algorithm's performance under the optimistic and pessimistic assumptions over the generated testbed. The first column shows the number of jobs (n), while all other columns are defined as before. We impose a time limit of four hours (14,400s) and calculate the average CPU time only among the instances solved within the time limit.

Table 2.6: Computational experiments on competitive scheduling

n	Optimistic			Pessimistic			Gap
	Avg	Max	z^*	Avg	Max	z_p^*	
10	0.9	3.4	6.8	1.3	4.0	5.3	0.22
15	1.8	3.8	10.2	6.0	14.7	8.0	0.22
20	4.8	11.9	13.7	16.9	74.0	11.0	0.20
25	10.8	20.8	17.2	64.1	122.7	14.4	0.16
30†	22.3	71.6	20.9	>495.5	>14,400	17.0	0.19

†: Nine of ten instances solved to optimality within the time limit for $n = 30$.

Table 2.6 shows that there is a considerable difference in the average number of on-time jobs under the optimistic and the pessimistic assumption. The pessimistic version of the problem is also significantly harder to solve, as evidenced by the comparison between average and maximum computation times between the optimistic and pessimistic cases. Furthermore, while our algorithm solves all instances under the optimistic assumption in less than roughly one minute, it fails to solve one $n = 30$ instance within the time limit under the pessimistic assumption. The difference in algorithmic performance is explained by the fact that RPEHPPs are considerably larger than REHPPs, along with the added difficulty required to solve the follower's subproblems under the pessimistic assumption.

2.6.5 Computing M-values in our Proposed Formulations

Table 2.7 presents general conditions on the M-values used in our models. Column 1 presents the vector of M-values being considered. Column 2 establishes conditions that a valid M-value must satisfy. Table 2.8 then shows the methods used to calculate the M-values.

Table 2.7: Conditions defining valid M-values

M-value	Validity conditions
$\mathbf{M}^1$	$M^1_j \geq -g^2_j(\mathbf{x}),\ \forall \mathbf{x} \in \Omega(\mathcal{X}),\ j = 1, \ldots, m_2$
$\mathbf{M}^2$	$M^2_{\hat{\mathbf{y}}} \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}}) - \phi^f(\mathbf{x}, \mathbf{y}),\ \forall (\mathbf{x}, \mathbf{y}) \in \Omega,\ \hat{\mathbf{y}} \in \mathcal{Y}$
$\mathbf{M}^3$	$M^3_{\hat{\mathbf{y}}} \geq \phi^f(\mathbf{x}, \hat{\mathbf{y}}) - \phi^f(\mathbf{x}, \mathbf{y}) + \delta,\ \forall (\mathbf{x}, \mathbf{y}) \in \Omega,\ \hat{\mathbf{y}} \in \mathcal{Y}$
$\mathbf{M}^4$	$M^4_{\hat{\mathbf{y}}} \geq \phi^l(\mathbf{x}, \mathbf{y}) - \phi^l(\mathbf{x}, \hat{\mathbf{y}}),\ \forall (\mathbf{x}, \mathbf{y}) \in \Omega,\ \hat{\mathbf{y}} \in \mathcal{Y}$
$\mathbf{M}^5$	$M^5_{\hat{\mathbf{y}}j} \geq g^1_j(\mathbf{x}) + h^1_j(\hat{\mathbf{y}}) - b^1_j,\ \forall \mathbf{x} \in \Omega(\mathcal{X}),\ \hat{\mathbf{y}} \in \mathcal{Y},\ j = 1, \ldots, m_1$
$\mathbf{M}^6$	$M^6_j \geq b^1_j - g^1_j(\mathbf{x}^i) - h^1_j(\mathbf{y}),\ \forall j = 1, \ldots, m_1,\ \mathbf{y} \in \mathcal{Y}(\mathbf{x}^i)$

Table 2.8: Calculating valid M-values

M-value	Value used for computations
$\mathbf{M}^1$	$\mathbf{M}^1 = \mathbf{0}$ because $g^2_j(\mathbf{x}) \geq 0$ in the BMILP testbed
$\mathbf{M}^2$	Set $l^f = \min\limits_{(\mathbf{x},\mathbf{y}) \in \Omega} \{\phi^f(\mathbf{x}, \mathbf{y})\}$ and $M^2_{\hat{\mathbf{y}}} = \phi^f(\mathbf{x}, \hat{\mathbf{y}}) - l^f,\ \forall \hat{\mathbf{y}} \in \mathcal{Y}$
$\mathbf{M}^3$	$M^3_{\hat{\mathbf{y}}} = M^2_{\hat{\mathbf{y}}} + \delta,\ \forall \hat{\mathbf{y}} \in \mathcal{Y}$
$\mathbf{M}^4$	$M^4_{\hat{\mathbf{y}}} = z^{\mathrm{HPP}} - \phi^l(\mathbf{x}, \hat{\mathbf{y}}),\ \forall \hat{\mathbf{y}} \in \mathcal{Y}$
$\mathbf{M}^5$	$M^5_{\hat{\mathbf{y}}j} = h^1_j(\hat{\mathbf{y}}),\ \forall \hat{\mathbf{y}} \in \mathcal{Y},\ j = 1, \ldots, m_1$ since $g^1_j(\mathbf{x}) \leq b^1_j,\ \forall \mathbf{x} \in \Omega(\mathcal{X}),\ j = 1, \ldots, m_1$ in the BMILP testbed
$\mathbf{M}^6$	$M^6_j = b^1_j - g^1_j(\mathbf{x}^i)$ because $h^1_j(\mathbf{y}) \geq 0,\ \forall \mathbf{y} \in \mathcal{Y}(\mathbf{x}^i)$ in the BMILP testbed

Chapter 3

A Backward Sampling Framework for Interdiction Problems with Fortification

3.1 Problem Statement

We consider defender-attacker-defender problems that are modeled as three-level, two-player Stackelberg games. In the first stage a defender (also known as the "owner" or "operator") can fortify a subset of assets, while in the second stage an attacker (often called the "interdictor") destroys a subset of the unprotected assets. The attacker's goal in the second stage is to maximize damage to the defender's objective, which is determined by solving an optimization problem in the third stage, using the surviving assets from the initial system.

Formally, let $\mathbf{w}$, $\mathbf{x}$, and $\mathbf{y}$ be the decision variables for the first-, second-, and third-stage problems, respectively. We assume that the third-stage problem can take any general form, while the first- and second-stage problems include only binary variables, i.e., $\mathbf{w} \in \{0,1\}^{n_{\mathbf{w}}}$ and $\mathbf{x} \in \{0,1\}^{n_{\mathbf{x}}}$, where $n_{\mathbf{w}}$ ($n_{\mathbf{x}}$) is the number of variables required to model asset fortification (attack). Let $\mathcal{W}$ be the set of feasible solutions to the first-stage problem. Let $\mathcal{X}(\mathbf{w})$ be the set of feasible second-stage solutions given a defense vector $\mathbf{w}$, and let $\mathcal{Y}(\mathbf{x})$ be the set of feasible third-stage solutions for a given attack vector $\mathbf{x}$. Also, define $\mathcal{X} = \bigcup_{\mathbf{w}\in\mathcal{W}} \mathcal{X}(\mathbf{w})$ and $\mathcal{Y} = \bigcup_{\mathbf{x}\in\mathcal{X}} \mathcal{Y}(\mathbf{x})$, i.e., $\mathcal{X}$ and $\mathcal{Y}$ are the set of all possible second- and third-stage feasible solutions, respectively. Finally, let $f(\mathbf{y})$ be the defender's objective function. We study problems of the form:

$$\mathcal{P}: \quad z^* = \min_{\mathbf{w}\in\mathcal{W}} \max_{\mathbf{x}\in\mathcal{X}(\mathbf{w})} \min_{\mathbf{y}\in\mathcal{Y}(\mathbf{x})} f(\mathbf{y}), \tag{3.1}$$

We refer to the first-, second-, and third-stage problems as *fortification*, *attack*, and *recourse* problems, respectively.

We present a novel backward sampling framework for solving three- (and two-) stage interdiction problems in which the recourse problem can take any form (e.g., it can be nonlinear, and can have integer variables), provided that all variables in the first two stages are restricted to be binary-valued. Hence, both fortification and interdiction of critical assets in the problem are "all or none" type decisions. An asset that is fortified is completely immune to attacks, and no assets can be only partially attacked. This framework is primarily designed to improve the solution of the interdiction problem, by solving relatively easy interdiction problem relaxations in which the defender is restricted to choose its recourse actions from a sample of the third-stage solution space. These problems provide upper bounds on the optimal interdiction solution; lower bounds can then be obtained by fixing an interdiction solution and optimizing the (original) recourse problem as a function of the fixed interdiction actions. This framework avoids linearizing a (potentially large) bilinear program, and also eliminates the need for applying combinatorial Benders' cuts at the interdiction stage (although we still require them to solve the fortification problem).

Using our framework, we construct an algorithm for the shortest path interdiction problem with fortification (SPIPF) that compares favorably to the current state-of-the-art algorithm, finding optimal solutions over random grid networks having up to 3,600 nodes and 17,000 arcs, and over real-road networks having up to 300,000 nodes and more than 1,000,000 arcs. We also consider the capacitated lot sizing interdiction problem with fortification (CLSIPF), in which the NP-hard third-stage problem is modeled as a MIP. We extend our framework to solve the CLSIPF, and demonstrate its ability to solve instances of this new problem class.

The remainder of this chapter is organized as follows. Section 3.2 presents the backward sampling framework and establishes the finite convergence of our approach to an optimal solution. Sections 3.3 and 3.4 describe how to specialize the framework for the SPIPF and CLSIPF, respectively. Section 3.5 presents our computational experiments.

3.2 The Backward Sampling Framework

The core idea behind the backward sampling framework is to iteratively sample the third-stage solution space so that instead of solving the original problem $\mathcal{P}$ directly, we solve *restricted problems* defined over smaller recourse solution spaces. We exploit this idea to more efficiently solve two-level max-min interdiction problems over $\mathbf{x}$ and $\mathbf{y}$, given a fixed defense vector $\mathbf{w}$. The solution of these restricted problems yields an upper bound on z^*, and also affords a mechanism for finding a lower bound on z^* as well. Finally, we embed this procedure within an outer optimization scheme that optimizes over $\mathbf{w}$. Section 3.2.1 describes our sampling procedure. Section 3.2.2 presents our proposed approach for solving the interdiction problems, and Section 3.2.3 discusses the outer optimization algorithm. Section 3.2.4 analyzes strategies for improving the effectiveness of the overall algorithm.

For convenience, we provide a summary of relevant definitions and notation used throughout this chapter in Table 3.1.

Table 3.1: Relevant definitions and notation

Symbol	Explanation
	Section 3.1
$\mathcal{W}$	Set of feasible solutions to the first-stage problem
$\mathcal{X}(\mathbf{w})$	Set of feasible second-stage solutions given a $\mathbf{w} \in \mathcal{W}$
$\mathcal{X}$	Set of all possible second-stage feasible solutions
$\mathcal{Y}(\mathbf{x})$	Set of feasible third-stage solutions for a given $\mathbf{x} \in \mathcal{X}$
$\mathcal{Y}$	Set of all possible third-stage feasible solutions
	Section 3.2.1
$\hat{\mathcal{Y}}$	A sample of third-stage solutions
$\hat{\mathcal{Y}}(\mathbf{x})$	$\hat{\mathcal{Y}} \cap \mathcal{Y}(\mathbf{x})$
	Section 3.2.2
$\mathcal{Q}(\mathbf{w})$	Two-level interdiction problem associated with a $\mathbf{w} \in \mathcal{W}$
$z^I(\mathbf{w})$	Optimal objective function value for $\mathcal{Q}(\mathbf{w})$
$\mathbf{x}^*$	An optimal solution to the attacker problem for a given $\mathbf{w} \in \mathcal{W}$
$\mathbf{y}^*$	An optimal solution to the recourse problem for a given $\mathbf{x} \in \mathcal{X}$
$\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$	Two-level interdiction problem associated with a $\mathbf{w} \in \mathcal{W}$ and a sample $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$
$z^I(\mathbf{w}, \hat{\mathcal{Y}})$	Optimal objective function value for $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$
$z^R(\hat{\mathbf{x}})$	Real damage of an attack $\hat{\mathbf{x}} \in \mathcal{X}$, obtained by solving $z^R(\hat{\mathbf{x}}) = \min\limits_{\mathbf{y} \in \mathcal{Y}(\hat{\mathbf{x}})} f(\mathbf{y})$
$\mathcal{Y}_z$	Set of feasible third-stage solutions whose objective value is less than or equal to z
	Section 3.2.3
$\mathcal{C}$	Set of covering constraints added to the fortification problem
$\mathcal{W}(\mathcal{C})$	Set of feasible first-stage solutions that satisfy all constraints in $\mathcal{C}$
$\bar{z}$	Global upper bound on z^*
	Section 3.2.4
$\hat{\psi}$	Tentative covering constraint
$\mathcal{C}^\psi$	Set of tentative covering constraints
$\mathcal{L}$	Waiting list that stores triples $(\hat{\mathbf{w}}, z^R(\hat{\mathbf{x}}), \hat{\psi})$
ϵ	Parameter that controls the addition of elements into $\mathcal{L}$

3.2.1 Sampling the Third-stage Solution Space

The sampling procedure selects a subset of third-stage solutions $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$, and throughout the algorithm augments $\hat{\mathcal{Y}}$ with new third-stage solutions from $\mathcal{Y}$. The sampling procedure would ideally be able to quickly identify several near-optimal solutions; however, we do not require this procedure to guarantee the generation of any new solutions in order for our framework to converge to an optimal solution. An appropriate strategy would tailor the sampling procedure for the problem at hand, as would be done for heuristic approaches. Some candidate methods may involve randomly restarted greedy heuristics; the use of optimal y-vectors corresponding to fixed x-values, along with neighboring solutions (obtained, e.g., by 2-opt swaps); or solutions generated via metaheuristics. We present two problem-specific sampling procedures in this study (see Sections 3.3 and 3.4), but emphasize that any sampling method can be employed in our overall (exact) optimization scheme.

For any attack vector $\mathbf{x}$ and third-stage solution sample $\hat{\mathcal{Y}}$, we denote by $\hat{\mathcal{Y}}(\mathbf{x}) \equiv \hat{\mathcal{Y}} \cap \mathcal{Y}(\mathbf{x})$ the subset of solutions that belong to $\hat{\mathcal{Y}}$ and are feasible given the attack vector $\mathbf{x}$. Anticipating the case for which there exists an attack $\mathbf{x} \in \mathcal{X}$ for which $\hat{\mathcal{Y}}(\mathbf{x}) = \emptyset$, we seed $\hat{\mathcal{Y}}$ with an artificial third-stage solution $\mathbf{y}^a$ that cannot be interdicted and has objective value $f(\mathbf{y}^a) = \infty$. This artificial solution ensures that $\hat{\mathcal{Y}}(\mathbf{x}) \neq \emptyset$ for any $\mathbf{x} \in \mathcal{X}$.

3.2.2 Solving Bilevel Interdiction Problems

Consider any feasible defense vector $\mathbf{w} \in \mathcal{W}$ and let

$$\mathcal{Q}(\mathbf{w}) : z^I(\mathbf{w}) = \max_{\mathbf{x} \in \mathcal{X}(\mathbf{w})} \min_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} f(\mathbf{y}) \tag{3.2}$$

be its associated two-level interdiction problem. Note that if there exists a defense $\mathbf{w} \in \mathcal{W}$ such that $\mathcal{X}(\mathbf{w}) = \emptyset$, then problem (3.2) is not defined. Hence, without loss of generality, we assume that $\mathcal{X}(\mathbf{w}) \neq \emptyset,\ \forall \mathbf{w} \in \mathcal{W}$.

Let $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$ be any third-stage solution sample and

$$\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}}) : z^I(\mathbf{w}, \hat{\mathcal{Y}}) = \max_{\mathbf{x} \in \mathcal{X}(\mathbf{w})} \min_{\mathbf{y} \in \hat{\mathcal{Y}}(\mathbf{x})} f(\mathbf{y}) \tag{3.3}$$

be the restricted problem in which recourse (third-stage) decisions are restricted to $\hat{\mathcal{Y}}$. The following result establishes that solving a restricted problem $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$ yields a valid upper bound on $z^I(\mathbf{w})$, which is in turn a valid upper bound on z^*.

Proposition 8. *Consider any* $\mathbf{w} \in \mathcal{W}$ *and third-stage solution sample* $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$. *Then we have* $z^I(\mathbf{w}, \hat{\mathcal{Y}}) \geq z^I(\mathbf{w}) \geq z^*$.

Proof Because $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$, we have that $\hat{\mathcal{Y}}(\mathbf{x}) \subseteq \mathcal{Y}(\mathbf{x})$, which implies $\min_{\mathbf{y} \in \hat{\mathcal{Y}}(\mathbf{x})} f(\mathbf{y}) \geq \min_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} f(\mathbf{y})$ for any attack $\mathbf{x}$. As a result, $z^I(\mathbf{w}, \hat{\mathcal{Y}}) \geq z^I(\mathbf{w})$. Also, $\mathcal{Q}(\mathbf{w})$ is a restriction of problem $\mathcal{P}$ in which $\mathbf{w}$ is fixed, and so $z^I(\mathbf{w}) \geq z^*$. This completes the proof. ■

We now establish conditions under which we can obtain an optimal solution to $\mathcal{Q}(\mathbf{w})$, for some $\mathbf{w} \in \mathcal{W}$, from a restricted problem $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$. First, let $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ be an optimal solution to the restricted problem $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$. We say that $z^I(\mathbf{w}, \hat{\mathcal{Y}})$ is the *perceived damage* of $\hat{\mathbf{x}}$ given $\hat{\mathcal{Y}}$, because the interdictor perceives that the recourse decision must come from $\hat{\mathcal{Y}}$. However, the defender may instead select from uninterdicted solutions in $\mathcal{Y}$, and so we define the *real damage* of attack $\hat{\mathbf{x}}$ over the original third-stage solution space $\mathcal{Y}$ as

$$z^R(\hat{\mathbf{x}}) = \min_{\mathbf{y} \in \mathcal{Y}(\hat{\mathbf{x}})} f(\mathbf{y}). \tag{3.4}$$

Observe that $z^R(\hat{\mathbf{x}}) \leq z^I(\mathbf{w}) \leq z^I(\mathbf{w}, \hat{\mathcal{Y}})$ for any $\hat{\mathbf{x}} \in \mathcal{X}(\mathbf{w})$. Proposition 9 states a condition in which an optimal solution to $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$ also optimizes $\mathcal{Q}(\mathbf{w})$.

Proposition 9. *Let* $\mathbf{w} \in \mathcal{W}$ *be a feasible defense,* $\hat{\mathcal{Y}}$ *be a third-stage solution sample, and* $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ *be an optimal solution to* $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$. *If* $z^I(\mathbf{w}, \hat{\mathcal{Y}}) = z^R(\hat{\mathbf{x}})$, *then* $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ *optimizes* $\mathcal{Q}(\mathbf{w})$.

Proof Suppose by contradiction that $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is not optimal to $\mathcal{Q}(\mathbf{w})$, and that there exists an attack $\mathbf{x}' \in \mathcal{X}(\mathbf{w})$ such that $z^R(\mathbf{x}') > z^R(\hat{\mathbf{x}})$. However, $\hat{\mathcal{Y}}(\mathbf{x}') \subseteq \mathcal{Y}(\mathbf{x}')$ implies that $\min_{\mathbf{y} \in \hat{\mathcal{Y}}(\mathbf{x}')} f(\mathbf{y}) \geq z^R(\mathbf{x}') > z^R(\hat{\mathbf{x}}) = z^I(\mathbf{w}, \hat{\mathcal{Y}})$, which contradicts the fact that $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is an optimal solution to $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$. ■

Our algorithm uses these results to solve $\mathcal{Q}(\mathbf{w})$, given $\mathbf{w} \in \mathcal{W}$, by iteratively solving restricted problems $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}}^i)$ defined over different third-stage samples $\hat{\mathcal{Y}}^i \subseteq \mathcal{Y}$. Algorithm 4 presents this approach, in which each iteration i yields an upper bound UB_i on $z^I(\mathbf{w})$ from solving $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}}^i)$, and a lower bound LB_i on $z^I(\mathbf{w})$ by obtaining $z^R(\hat{\mathbf{x}})$, for some $\hat{\mathbf{x}} \in \mathcal{X}(\mathbf{w})$. As we will demonstrate, the sequence of UB_i-values is nonincreasing, although the LB_i-values need not be monotone. The main while-loop (line 4) is executed until the optimality condition described in Proposition 9 is met. Line 6 solves the restricted problem $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}}^i)$ defined over the current sample $\hat{\mathcal{Y}}^i$. Line 7 calculates the real damage $z^R(\hat{\mathbf{x}})$ for attack $\hat{\mathbf{x}}$ and sets LB_i equal to this value (see Remark 7 for additional explanation). Line 8 creates $\hat{\mathcal{Y}}^{i+1}$ by including solutions in $\hat{\mathcal{Y}}^i$ along with $\hat{\mathbf{y}}^*$, i.e., an optimal third-stage response to attack $\hat{\mathbf{x}}$.

If the perceived damage obtained is less than the upper bound at the previous iteration, then a new upper bound on $z^I(\mathbf{w})$ has been obtained, and the algorithm executes lines 10–12. Line 10 removes from $\hat{\mathcal{Y}}^{i+1}$ all those solutions whose objective value is greater than UB_i, and lines 11–12 attempt to add new solutions to $\hat{\mathcal{Y}}^{i+1}$ from $\mathcal{Y}_{UB_i} \equiv \{\mathbf{y} \in \mathcal{Y} \mid f(\mathbf{y}) \leq UB_i\}$ by sampling the third-stage solution space $\mathcal{Y}$ and retaining only those samples having objective no more than UB_i. If the optimality condition in line 14 is satisfied, then line 15 returns an optimal solution.

Remark 6. *Using a large sample size increases the chances of obtaining tighter upper bounds in line 6. However, if $|\hat{\mathcal{Y}}|$ is too large, then $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$ will be large as well, and may potentially be too difficult to solve. On the other hand, if third-stage solutions in $\hat{\mathcal{Y}}$ are not diverse, then the attacker can easily interdict all $\mathbf{y} \in \hat{\mathcal{Y}}$ by attacking a few critical assets. This leads to poor upper bounds from solving $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$. It is thus desirable to use a sampling scheme that generates a diverse sample of moderate size, containing optimal or near-optimal uninterdicted third-stage solutions, which are likely to be optimal responses to attacks $\hat{\mathbf{x}}$ explored by the algorithm. Sections 3.3.2.1 and 3.4.2.1 present our sampling scheme tailored for the SPIPF and CLSIPF, respectively.*

Remark 7. *Intuitively, it may seem better to set LB_i to the maximum of LB_{i-1} and the real damage at iteration i, given by $\min\limits_{\mathbf{y} \in \mathcal{Y}(\hat{\mathbf{x}})} f(\mathbf{y})$. However, doing so creates the possibility that the optimal objective is found, but not a solution that achieves that objective. This could happen if the objective value (perceived damage) for $\hat{\mathbf{x}}$ obtained in line 6 is such that $z^I(\mathbf{w}, \hat{\mathcal{Y}}^i) > z^R(\hat{\mathbf{x}})$, even though $z^I(\mathbf{w}, \hat{\mathcal{Y}}^i) = \max\limits_{k=1,\ldots,i} \{\mathrm{LB}_k\}$.*

Algorithm 4 Solving bilevel interdiction problem $Q(\mathbf{w})$ via backward sampling

```
Input: Problem $\mathcal{P}$ and a feasible defense $\mathbf{w} \in \mathcal{W}$
Output: An optimal solution to $\mathcal{Q}(\mathbf{w})$
1: Set $UB_0 = \infty$ and $LB_0 = -\infty$                                  ▷ Initialization
2: Select $\hat{\mathcal{Y}}^1 \subseteq \mathcal{Y}$ as a sampling of the third-stage solution space, and compute $f(\mathbf{y})$ for each solution
   $\mathbf{y} \in \hat{\mathcal{Y}}^1$                                        ▷ See Remark 6
3: Set counter $i = 0$
4: while $LB_i < UB_i$ do                                                      ▷ Main while-loop
5:     Set $i = i + 1$
6:     Solve $UB_i = z^I(\mathbf{w}, \hat{\mathcal{Y}}^i) = \max_{\mathbf{x} \in \mathcal{X}(\mathbf{w})} \min_{\mathbf{y} \in \hat{\mathcal{Y}}^i(\mathbf{x})} f(\mathbf{y})$ and obtain an optimal solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$
7:     Solve $LB_i = z^R(\hat{\mathbf{x}}) = \min_{\mathbf{y} \in \mathcal{Y}(\hat{\mathbf{x}})} f(\mathbf{y})$ and obtain an optimal solution $\hat{\mathbf{y}}^*$   ▷ See Remark 7
8:     Set $\hat{\mathcal{Y}}^{i+1} = \hat{\mathcal{Y}}^i \cup \{\hat{\mathbf{y}}^*\}$
9:     if $UB_i < UB_{i-1}$ then
10:        Remove from $\hat{\mathcal{Y}}^{i+1}$ all solutions having objective value greater than $UB_i$
11:        Select $\hat{\mathcal{Y}}' \subseteq \mathcal{Y}$ as a sampling of the third-stage solution space
12:        Add to $\hat{\mathcal{Y}}^{i+1}$ all new solutions in $\hat{\mathcal{Y}}' \cap \mathcal{Y}_{UB_i}$
13:    end if
14:    if $LB_i = UB_i$ then
15:        Terminate with solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$
16:    end if
17: end while
```

Proposition 10 shows that the sequence of UB_i-values obtained is nonincreasing, and Proposition 11 states the finiteness and correctness of the proposed algorithm.

Proposition 10. *The upper bounds UB_i produced by Algorithm 4 are nonincreasing, and at iteration i, $\hat{\mathcal{Y}}^1_{UB_i} \subseteq \hat{\mathcal{Y}}^2_{UB_i} \subseteq \cdots \subseteq \hat{\mathcal{Y}}^{i+1}_{UB_i}$, where $\hat{\mathcal{Y}}^j_{UB_i} \equiv \{\mathbf{y} \in \hat{\mathcal{Y}}^j \mid f(\mathbf{y}) \leq UB_i\}$.*

Proof We establish the result by induction. As a base case, $UB_0 = \infty \geq UB_1$ is obvious. Also, if $UB_1 = UB_0$, then $\hat{\mathcal{Y}}^2 = \hat{\mathcal{Y}}^1 \cup \{\hat{\mathbf{y}}^*\}$ for some $\hat{\mathbf{y}}^*$, and if $UB_1 < UB_0$, then each $\mathbf{y} \in \hat{\mathcal{Y}}^1$ such that $f(\mathbf{y}) \leq UB_1$ also belongs to $\hat{\mathcal{Y}}^2$. Hence, $\hat{\mathcal{Y}}^1_{UB_1} \subseteq \hat{\mathcal{Y}}^2_{UB_1}$ in either case. Next, suppose that by induction, $UB_{i-1} \geq UB_i$ and $\hat{\mathcal{Y}}^i_{UB_k} \subseteq \hat{\mathcal{Y}}^{i+1}_{UB_k}$ $\forall i = 1, \ldots, k$, for some $k \geq 1$. We compute $UB_{k+1} = z^I(\mathbf{w}, \hat{\mathcal{Y}}^{k+1})$. Note that because $UB_k = z^I(\mathbf{w}, \hat{\mathcal{Y}}^k)$, then $z^I(\mathbf{w}, \hat{\mathcal{Y}}^k) = z^I(\mathbf{w}, \hat{\mathcal{Y}}^k_{UB_k})$ (because the attacker can ignore solutions $\mathbf{y} \in \hat{\mathcal{Y}}^k : f(\mathbf{y}) > UB_k$). Noting that $\hat{\mathcal{Y}}^k_{UB_k} \subseteq \hat{\mathcal{Y}}^{k+1}_{UB_k}$ by assumption, we have that $\hat{\mathcal{Y}}^k_{UB_k} \subseteq \hat{\mathcal{Y}}^{k+1}$ and $UB_k = z^I(\mathbf{w}, \hat{\mathcal{Y}}^k_{UB_k}) \geq z^I(\mathbf{w}, \hat{\mathcal{Y}}^{k+1}) = UB_{k+1}$.

Moreover, since $\hat{\mathcal{Y}}^i_{UB_k} \subseteq \hat{\mathcal{Y}}^{i+1}_{UB_k}$ $\forall i = 1, \ldots, k$ by assumption, $UB_{k+1} \leq UB_k$ implies that $\hat{\mathcal{Y}}^i_{UB_{k+1}} \subseteq \hat{\mathcal{Y}}^{i+1}_{UB_{k+1}}$ $\forall i = 1, \ldots, k$. For $i = k + 1$, if $UB_{k+1} = UB_k$, then $\hat{\mathcal{Y}}^{k+2} = \hat{\mathcal{Y}}^{k+1} \cup \{\hat{\mathbf{y}}^*\}$, and otherwise any $\mathbf{y} \in \hat{\mathcal{Y}}^{k+1}$ satisfying $f(\mathbf{y}) \leq UB^{k+1}$ also belongs to $\hat{\mathcal{Y}}^{k+2}$. Hence, $\hat{\mathcal{Y}}^{k+1}_{UB_{k+1}} \subseteq \hat{\mathcal{Y}}^{k+2}_{UB_{k+1}}$. This completes the proof. ■

Proposition 11. *Algorithm 4 terminates finitely with an optimal solution.*

Proof The first time that an attack $\hat{\mathbf{x}}$ is a part of an optimal solution to $z^I(\mathbf{w}, \hat{\mathcal{Y}}^i)$ at line 6, the algorithm includes a corresponding optimal recourse response $\hat{\mathbf{y}}^*$ into $\hat{\mathcal{Y}}^{i+1}$. Suppose that $\hat{\mathbf{x}}$ is a part of an optimal solution to $z^I(\mathbf{w}, \hat{\mathcal{Y}}^k)$ for a second time at iteration $k > i$. Proposition 10 guarantees that $\hat{\mathbf{y}}^* \in \hat{\mathcal{Y}}^k$ for $k > i$. Therefore, upon encountering $\hat{\mathbf{x}}$ at iteration k, an optimal recourse response is $\hat{\mathbf{y}}^*$, thus ensuring that the optimality condition stated in Proposition 9 is met. Finite termination of the algorithm then follows from the finiteness of $\mathcal{X}$. ■

We now discuss similarities and differences between our sampling approach and Benders' decomposition [Benders, 1962]. Consider a two-level interdiction problem in which the recourse problem is a linear program whose objective function is parametrized by the attacker's decisions. Let $\mathbf{A}$ be the recourse constraint coefficient matrix, $\mathbf{b}$ be the right-hand-side vector, and $\mathbf{D}$ be a diagonal matrix with penalty values corresponding to attack decisions. We consider the following problem:

$$\max_{\mathbf{x}\in\mathcal{X}} \min \quad (\mathbf{c} + \mathbf{D}\mathbf{x})^\intercal \mathbf{y} \tag{3.5}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{y} = \mathbf{b} \tag{3.6}$$

$$\mathbf{y} \geq \mathbf{0}. \tag{3.7}$$

Since the recourse problem is a linear program, we reformulate (3.5)–(3.7) by considering its dual:

$$\max_{\mathbf{x}\in\mathcal{X}} \max \quad \mathbf{b}^\intercal \boldsymbol{\pi} \tag{3.8}$$

$$\text{s.t.} \quad \mathbf{A}^\intercal \boldsymbol{\pi} \leq \mathbf{c} + \mathbf{D}\mathbf{x}. \tag{3.9}$$

Note that solving (3.5)–(3.7) with our sampling algorithm is the same as solving (3.8)–(3.9) using Benders' decomposition since recourse solutions included in the sample are feasible solutions to the Benders' dual subproblem. However, if the attacker's decisions also impact the recourse constraints, then this equivalence is no longer true because recourse solutions in the sample need not be feasible solutions to the Benders' dual subproblem. Moreover, if the recourse problem takes a more general form (e.g., an integer program), then we cannot establish a direct mapping from our sampling approach to Benders' decomposition since a strong dual may not be available to transform the original max-min problem into a max-max type of problem.

Also, previous approaches for two-stage max-max (or min-min) problems employ variations of Benders decomposition. These approaches employ general duality [Carøe and Tind, 1998], inference duality [Hooker and Ottosson, 2003], or disjunctive decomposition algorithms [Sen and Sherali, 2006] to generate valid inequalities that approximate the recourse problem value function. Our sampling algorithm also approximates the recourse problem value function by iteratively adding new solutions into the sample. However, our approach differs in that it attacks min-max (or max-min) problems, and it does not rely on any notion of duality in order to generate the desired value function approximation.

3.2.3 Optimizing the Defense Decisions

We now propose an approach to solve the three-level problem $\mathcal{P}$. This approach is based on the identification of *critical attacks*, i.e., attacks that must be blocked in order to improve the defender's incumbent objective value. Formally, we define a critical attack as any attack $\hat{\mathbf{x}}$ such that its real damage $z^R(\hat{\mathbf{x}})$ is greater than or equal to a target upper bound $\bar{z}$. Our approach adds a *covering constraint* $\mathbf{w}^\intercal\hat{\mathbf{x}} \geq 1$ to the fortification problem for each critical attack $\hat{\mathbf{x}}$, which states that at least one of the assets attacked by $\hat{\mathbf{x}}$ must be fortified.

Proposition 12. *For problem $\mathcal{P}$ having optimal objective value z^*, consider any attack $\hat{\mathbf{x}} \in \mathcal{X}$. If $z^* < z^R(\hat{\mathbf{x}})$, then any optimal solution $(\mathbf{w}^*, \mathbf{x}^*, \mathbf{y}^*)$ satisfies $\mathbf{w}^{*\intercal}\hat{\mathbf{x}} \geq 1$.*

Proof By contradiction, suppose that $z^* < z^R(\hat{\mathbf{x}})$, and that there is an optimal solution $(\mathbf{w}^*, \mathbf{x}^*, \mathbf{y}^*)$ such that $\mathbf{w}^{*\intercal}\hat{\mathbf{x}} = 0$. Then $\hat{\mathbf{x}} \in \mathcal{X}(\mathbf{w}^*)$, and so $z^* = \max\limits_{\mathbf{x}\in\mathcal{X}(\mathbf{w}^*)} \min\limits_{\mathbf{y}\in\mathcal{Y}(\mathbf{x})} f(\mathbf{y}) \geq z^R(\hat{\mathbf{x}})$. This contradicts the assumption that $z^* < z^R(\hat{\mathbf{x}})$ and concludes the proof. ■

These covering constraints can be seen as a general case of the *combinatorial Benders' cut* [Codato and Fischetti, 2006] where the fortification problem acts as a master problem and $\mathcal{Q}(\hat{\mathbf{w}})$ as a subproblem. Similar so-called *supervalid inequalies* were introduced by Israeli and Wood [2002] for a two-level shortest path interdiction problem.

Our approach explores different defense vectors $\hat{\mathbf{w}} \in \mathcal{W}$ and solves the associated interdiction problems $\mathcal{Q}(\hat{\mathbf{w}})$ with a variation of Algorithm 4 that stops whenever it identifies a critical attack. When such an attack is identified, the algorithm adds a covering constraint to the fortification problem, forcing the defender to block each identified critical attack. When the fortification problem becomes infeasible, the algorithm terminates with the incumbent solution being optimal. This process must eventually terminate with an infeasible first-stage problem because $\mathcal{X}(\mathbf{w}) \neq \emptyset$, $\forall \mathbf{w} \in \mathcal{W}$, by assumption.

Algorithm 5 presents the proposed approach. Let $\mathcal{C}$ be the set of covering constraints added to the fortification problem and $\mathcal{W}(\mathcal{C}) \equiv \{\mathbf{w} \in \mathcal{W} \mid \mathbf{w} \text{ satisfies all constraints in } \mathcal{C}\}$. The algorithm starts with $\mathcal{C} = \emptyset$ and a global upper bound $\bar{z} = \infty$. The main while-loop (in line 4) is executed until the fortification problem becomes infeasible. The two main steps inside this while-loop are selecting a feasible defense $\hat{\mathbf{w}} \in \mathcal{W}(\mathcal{C})$ (in line 5), and solving its associated interdiction problem $\mathcal{Q}(\hat{\mathbf{w}})$ with a variation of Algorithm 4 (lines 6–23). The inner while-loop (in line 7) is executed until $LB_i = z^R(\hat{\mathbf{x}}) \geq \bar{z}$, for some $\hat{\mathbf{x}} \in \mathcal{X}(\hat{\mathbf{w}})$, indicating that $\hat{\mathbf{x}}$ is a critical attack. At this point, Algorithm 5 stops solving $\mathcal{Q}(\hat{\mathbf{w}})$ and adds a covering constraint to $\mathcal{C}$. Finally, lines 8–22 replicate Algorithm 4, except for updating the global upper bound $\bar{z}$ (in line 13), adding a covering constraint to $\mathcal{C}$ if a critical attack is identified (in lines 17–19), and updating the incumbent solution when an optimal solution to $\mathcal{Q}(\hat{\mathbf{w}})$ has an objective value equal to $\bar{z}$ (in lines 20–22).

Algorithm 5 terminates finitely because each critical attack $\hat{\mathbf{x}} \in \mathcal{X}$ triggers the generation of a covering constraint to $\mathcal{C}$, which excludes the fortification action $\hat{\mathbf{w}}$ from $\mathcal{W}(\mathcal{C})$. Finite termination of the algorithm then follows from the finiteness of $\mathcal{W}$ and from Proposition 11.

The correctness of Algorithm 5 results directly from Propositions 8 and 12. Note that the upper bound $\bar{z}$ is nonincreasing throughout the execution of the algorithm. Proposition 12 states that each of the covering constraints is necessary in order to achieve an objective value less than $\bar{z}$. As a result, once $\mathcal{W}(\mathcal{C})$ becomes empty we conclude that $z^* \geq \bar{z}$. Since $\bar{z}$ is an upper bound, we also have that $z^* \leq \bar{z}$, which guarantees that the algorithm terminates with the optimal value $\bar{z} = z^*$. For any $\hat{\mathbf{w}}$ that reduces $\bar{z}$, the algorithm solves $\mathcal{Q}(\hat{\mathbf{w}})$ to optimality, i.e., until $LB_i = UB_i = \bar{z}$, and updates the incumbent solution. As a result, the algorithm terminates with an optimal incumbent solution since its objective value is equal to $\bar{z} = z^*$.

3.2.4 Accelerating the Algorithm

We now describe a mechanism designed to reduce the number of restricted interdiction problems that are solved to optimality. The idea is to "pause" the exploration of any $\hat{\mathbf{w}} \in \mathcal{W}$ whenever the potential relative improvement to the current global upper bound is sufficiently small. At this point, we add a tentative covering constraint to the fortification problem, guessing that the

Algorithm 5 Backward sampling framework

```
Input: Problem $\mathcal{P}$
Output: An optimal solution to $\mathcal{P}$
1: Set the global upper bound $\bar{z} = \infty$ and covering constraints set $\mathcal{C} = \emptyset$    ▷ Initialization
2: Select $\hat{\mathcal{Y}}^1 \subseteq \mathcal{Y}$ as a sampling of the third-stage solution space, and compute $f(\mathbf{y})$ for each solution
   $\mathbf{y} \in \hat{\mathcal{Y}}^1$
3: Set counter $i = 0$
4: while $\mathcal{W}(\mathcal{C}) \neq \emptyset$ do    ▷ Main while-loop
5:     Select any $\hat{\mathbf{w}} \in \mathcal{W}(\mathcal{C})$
6:     Initialize $UB_i = \infty$ and $LB_i = -\infty$
7:     while $LB_i < \bar{z}$ do
8:         Set $i = i + 1$
9:         Solve $UB_i = z^I(\hat{\mathbf{w}}, \hat{\mathcal{Y}}^i) = \max_{\mathbf{x} \in \mathcal{X}(\hat{\mathbf{w}})} \min_{\mathbf{y} \in \hat{\mathcal{Y}}^i(\mathbf{x})} f(\mathbf{y})$ and obtain an optimal solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$
10:        Solve $LB_i = z^R(\hat{\mathbf{x}}) = \min_{\mathbf{y} \in \mathcal{Y}(\hat{\mathbf{x}})} f(\mathbf{y})$ and obtain an optimal solution $\hat{\mathbf{y}}^*$
11:        Set $\hat{\mathcal{Y}}^{i+1} = \hat{\mathcal{Y}}^i \cup \{\hat{\mathbf{y}}^*\}$
12:        if $UB_i < \bar{z}$ then
13:            Update global upper bound $\bar{z} \leftarrow UB_i$
14:            Remove from $\hat{\mathcal{Y}}^{i+1}$ all solutions having objective value greater than $UB_i$
15:            Select $\hat{\mathcal{Y}}' \subseteq \mathcal{Y}$ as a sampling of the third-stage solution space
16:            Add to $\hat{\mathcal{Y}}^{i+1}$ all new solutions in $\hat{\mathcal{Y}}' \cap \mathcal{Y}_{UB_i}$
17:        else if $LB_i \geq \bar{z}$ then    ▷ A critical attack has been identified
18:            Add the covering constraint $\mathbf{w}^\intercal \hat{\mathbf{x}} \geq 1$ to $\mathcal{C}$
19:        end if
20:        if $LB_i = UB_i = \bar{z}$ then
21:            Update the incumbent solution $(\bar{\mathbf{w}}, \bar{\mathbf{x}}, \bar{\mathbf{y}}) \leftarrow (\hat{\mathbf{w}}, \hat{\mathbf{x}}, \hat{\mathbf{y}})$
22:        end if
23:    end while
24: end while
25: Return $(\bar{\mathbf{w}}, \bar{\mathbf{x}}, \bar{\mathbf{y}})$
```

best known attack $\hat{\mathbf{x}}$ corresponding to $\hat{\mathbf{w}}$ is critical (which it will indeed be if the global upper bound is reduced by a relatively small amount). We store $\hat{\mathbf{w}}$ in a waiting list to be revisited later in the execution of the algorithm, at which time we either confirm that $\hat{\mathbf{x}}$ was critical and discard $\hat{\mathbf{w}}$ from the waiting list, or conclude that the attack may not be critical and continue exploring $\hat{\mathbf{w}}$.

Formally, let $\mathcal{C}$ be the set of covering constraints derived from (known) critical attacks and $\mathcal{C}^\psi$ be the set of tentative covering constraints. Let $\mathcal{L}$ be a waiting list that stores triples $(\hat{\mathbf{w}}, z^R(\hat{\mathbf{x}}), \hat{\psi})$, where $\hat{\mathbf{w}}$ is a defense vector that must be revisited, $z^R(\hat{\mathbf{x}})$ is the real damage for an attack $\hat{\mathbf{x}} \in \mathcal{X}(\hat{\mathbf{w}})$ that we guess is critical, and $\hat{\psi}$ is the corresponding covering constraint. Algorithm 6 formally states the accelerated backward sampling algorithm. If $\mathcal{W}(\mathcal{C} \cup \mathcal{C}^\psi) \neq \emptyset$, then line 6 selects a defense $\hat{\mathbf{w}} \in \mathcal{W}(\mathcal{C} \cup \mathcal{C}^\psi)$ and lines 7–22 explore problem $\mathcal{Q}(\hat{\mathbf{w}})$ as in Algorithm 5. When $\hat{\mathbf{x}}$ has not been shown to be critical, line 23 computes the ratio $(\bar{z} - LB_i)/\bar{z}$, assuming that $\bar{z} > 0$, to measure the percent reduction to $\bar{z}$ that could be achieved by continuing to solve $\mathcal{Q}(\hat{\mathbf{w}})$. If this ratio is not greater than some parameter $\epsilon > 0$, then lines 24–25 store $(\hat{\mathbf{w}}, z^R(\hat{\mathbf{x}}), \mathbf{w}^\intercal \hat{\mathbf{x}} \geq 1)$ in $\mathcal{L}$, add the corresponding tentative covering constraint to $\mathcal{C}^\psi$, and stop the exploration of the current $\hat{\mathbf{w}}$. When $\mathcal{W}(\mathcal{C} \cup \mathcal{C}^\psi) = \emptyset$, if $\mathcal{C}^\psi \neq \emptyset$, then lines 30–39 reconsider the items stored in the waiting list. The first for-loop (in lines 30–34) iterates over $\mathcal{L}$ and moves from $\mathcal{C}^\psi$ to $\mathcal{C}$ all the covering constraints corresponding to attacks with $z^R(\hat{\mathbf{x}}^k) > \bar{z}$, discarding the associated $\mathbf{w}^k$ from further exploration. Note that if $z^R(\hat{\mathbf{x}}^k) = \bar{z}$, then we cannot yet discard $\mathbf{w}^k$: even if $\bar{z} = z^*$, the algorithm might not have updated the incumbent $(\bar{\mathbf{w}}, \bar{\mathbf{x}}, \bar{\mathbf{y}})$. The second for-loop (in lines 35–39) iterates over the remaining items in $\mathcal{L}$ and resumes exploration for any $\mathbf{w}^k$ that is still in $\mathcal{W}(\mathcal{C})$, but with $\epsilon = 0$. Finally, line 40 discards the remaining constraints in $\mathcal{C}^\psi$, empties the waiting list, and returns to the main while-loop.

Algorithm 6 Backward sampling framework with waiting list

```
Input: Problem $\mathcal{P}$ and a threshold parameter $\epsilon > 0$
Output: An optimal solution to $\mathcal{P}$
1: Set $\bar{z} = \infty$                                        ▷ Initialization
2: Initialize covering constraints sets $\mathcal{C} = \emptyset$, $\mathcal{C}^{\psi} = \emptyset$, and waiting list $\mathcal{L} = \emptyset$
3: Select $\hat{\mathcal{Y}}^1 \subseteq \mathcal{Y}$ as a sampling of the third-stage solution space, and compute $f(\mathbf{y})$ for each solution $\mathbf{y} \in \hat{\mathcal{Y}}^1$
4: Set counter $i = 0$
5: while $\mathcal{W}(\mathcal{C} \cup \mathcal{C}^{\psi}) \neq \emptyset$ do                ▷ Main while-loop
6:     Select any $\hat{\mathbf{w}} \in \mathcal{W}(\mathcal{C} \cup \mathcal{C}^{\psi})$
7:     Initialize $UB_i = \infty$ and $LB_i = -\infty$
8:     while $LB_i < \bar{z}$ do
9:         Set $i = i + 1$
10:        Solve $UB_i = z^I(\hat{\mathbf{w}}, \hat{\mathcal{Y}}^i) = \max_{\mathbf{x} \in \mathcal{X}(\hat{\mathbf{w}})} \min_{\mathbf{y} \in \hat{\mathcal{Y}}^i(\mathbf{x})} f(\mathbf{y})$ and obtain an optimal solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$
11:        Solve $LB_i = z^R(\hat{\mathbf{x}}) = \min_{\mathbf{y} \in \mathcal{Y}(\hat{\mathbf{x}})} f(\mathbf{y})$ and obtain an optimal solution $\hat{\mathbf{y}}^*$
12:        Set $\hat{\mathcal{Y}}^{i+1} = \hat{\mathcal{Y}}^i \cup \{\hat{\mathbf{y}}^*\}$
13:        if $UB_i < \bar{z}$ then
14:            Update global upper bound $\bar{z} \leftarrow UB_i$
15:            Remove from $\hat{\mathcal{Y}}^{i+1}$ all solutions having objective value greater than $UB_i$
16:            Select $\hat{\mathcal{Y}}' \subseteq \mathcal{Y}$ as a sampling of the third-stage solution space
17:            Add to $\hat{\mathcal{Y}}^{i+1}$ all new solutions in $\hat{\mathcal{Y}}' \cap \mathcal{Y}_{UB_i}$
18:        else if $LB_i \geq \bar{z}$ then                ▷ A critical attack has been identified
19:            Add the covering constraint $\mathbf{w}^\intercal \hat{\mathbf{x}} \geq 1$ to $\mathcal{C}$
20:        end if
21:        if $LB_i = UB_i = \bar{z}$ then
22:            Update the incumbent solution $(\bar{\mathbf{w}}, \bar{\mathbf{x}}, \bar{\mathbf{y}}) \leftarrow (\hat{\mathbf{w}}, \hat{\mathbf{x}}, \hat{\mathbf{y}})$
23:        else if $(\bar{z} - LB_i)/\bar{z} \leq \epsilon$ and $LB_i < \bar{z}$ then
24:            Add $(\hat{\mathbf{w}}, z^R(\hat{\mathbf{x}}), \mathbf{w}^\intercal \hat{\mathbf{x}} \geq 1)$ to the waiting list $\mathcal{L}$
25:            Add the covering constraint $\mathbf{w}^\intercal \hat{\mathbf{x}} \geq 1$ to $\mathcal{C}^{\psi}$ and go to line 6
26:        end if
27:    end while
28: end while
29: if $\mathcal{C}^{\psi} \neq \emptyset$ then                ▷ Reconsider items stored in the waiting list
30:    for each list member $k \in \mathcal{L}$ represented by $(\mathbf{w}^k, z^R(\hat{\mathbf{x}}^k), \psi^k)$ do
31:        if $z^R(\hat{\mathbf{x}}^k) > \bar{z}$ then
32:            Add $\psi^k$ to $\mathcal{C}$, remove $\psi^k$ from $\mathcal{C}^{\psi}$, and remove $(\mathbf{w}^k, z^R(\hat{\mathbf{x}}^k), \psi^k)$ from $\mathcal{L}$
33:        end if
34:    end for
35:    for each list member $k \in \mathcal{L}$ represented by $(\mathbf{w}^k, z^R(\hat{\mathbf{x}}^k), \psi^k)$ do
36:        if $\mathbf{w}^k \in \mathcal{W}(\mathcal{C})$ then
37:            Resume solving $\mathcal{Q}(\mathbf{w}^k)$ with a threshold $\epsilon = 0$
38:        end if
39:    end for
40:    Reset $\mathcal{C}^{\psi} \leftarrow \emptyset$, $\mathcal{L} \leftarrow \emptyset$, and go to line 5
41: end if
42: Return $(\bar{\mathbf{w}}, \bar{\mathbf{x}}, \bar{\mathbf{y}})$
```

We further note that setting the value of the precision parameter ϵ to zero is equivalent to using no waiting list $\mathcal{L}$, which in turn converts Algorithm 6 to Algorithm 5.

3.3 Shortest Path Interdiction Problem with Fortification

A significant amount of research has been dedicated to the shortest path interdiction problem. However, fewer studies consider the SPIPF, in which the defender is able to fortify a subset of arcs before being attacked. Brown et al. [2006] include fortification decisions for the problem of protecting an electric power grid and Smith et al. [2007] consider fortification against three attacker strategies (including both heuristic and optimal strategies) in the context of survivable network design. Both approaches are based on a dualization of the recourse problem followed by a decomposition algorithm that generates Benders' cuts, and can be easily adapted for the SPIPF. Cappanera and Scaparra [2011] propose an implicit enumeration algorithm that is capable of finding optimal solutions to the SPIPF on networks having up to 225 nodes and 996 arcs.

3.3.1 Problem Statement

The SPIPF is formally defined on a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ is the set of arcs, s is the source node, and t is the destination node. For each arc $(i,j) \in \mathcal{A}$ there are two nonnegative attributes: the cost $c_{ij} \geq 0$ of traversing the arc, and the delay (or penalty) $d_{ij} \geq 0$ incurred when traversing an interdicted arc (so that crossing an interdicted arc costs $c_{ij} + d_{ij}$). Let $\mathbf{w}$ be the fortification decision variables defined over the arcs, where $\mathcal{W} \equiv \left\{\mathbf{w} : \mathbf{e}^\intercal \mathbf{w} \leq Q,\ \mathbf{w} \in \{0,1\}^{|\mathcal{A}|}\right\}$ enforces a cardinality constraint on the number of fortified arcs and ensures that the variables are binary. Similarly, let $\mathbf{x} \in \mathcal{X}(\mathbf{w})$ be the second-stage attack decision variables, where $\mathcal{X}(\mathbf{w}) \equiv \left\{\mathbf{x} : \mathbf{e}^\intercal \mathbf{x} \leq B,\ x_{ij} \leq 1 - w_{ij}\ \forall (i,j) \in \mathcal{A},\ \mathbf{x} \in \{0,1\}^{|\mathcal{A}|}\right\}$ ensures that a maximum of B unprotected arcs are attacked, and forces the $\mathbf{x}$-variables to be binary. Finally, let $\mathbf{y}$ be the third-stage arc-flow variables. The SPIPF can be formally stated as:

$$\min_{\mathbf{w} \in \mathcal{W}} \max_{\mathbf{x} \in \mathcal{X}(\mathbf{w})} \min \sum_{(i,j) \in \mathcal{A}} (c_{ij} + d_{ij} x_{ij}) y_{ij} \tag{3.10}$$

$$\text{s.t.} \sum_{\{j|(i,j)\in\mathcal{A}\}} y_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} y_{ji} = \begin{cases} 1, & \text{for } i = s \\ 0, & \text{for } i \in \mathcal{N}\backslash\{s,t\} \\ -1, & \text{for } i = t \end{cases} \tag{3.11}$$

$$y_{ij} \geq 0, \quad \forall (i,j) \in \mathcal{A}, \tag{3.12}$$

where in the objective function (3.10), the original cost of any arc is increased by d_{ij} when the arc is attacked (i.e., $x_{ij} = 1$). Constraints (3.11) define the shortest path flow conservation constraints, and (3.12) restrict the $\mathbf{y}$-variables to be nonnegative.

3.3.2 Solution Approach

The implementation of the backward sampling framework for the SPIPF requires a sampling scheme, an algorithm for solving two-level shortest path interdiction problems restricted over a sample of s-t paths, and a method to solve third-stage shortest path problems. The latter is simply accomplished via Dijkstra's algorithm [Dijkstra, 1959]. We discuss the first two components of our approach in the following subsections.

3.3.2.1 Sampling Scheme

We adapt the *pulse algorithm* [Lozano and Medaglia, 2013] for the constrained shortest path problem to sample s-t paths from $\mathcal{G}$. The pulse algorithm conducts a recursive implicit enumeration of the solution space, supported by pruning strategies that efficiently discard a vast number of suboptimal solutions. The algorithm conducts a depth-first search beginning at s. When a partial path is pruned or the search reaches node t, the algorithm backtracks and continues the search through unexplored regions of the solution space.

We implemented two pruning strategies: *bound* and *arc-usage* pruning. The bound pruning strategy [Lozano and Medaglia, 2013] discards any path whose cost exceeds the current upper bound $\bar{z}$. To do so, we first obtain the minimum cost needed to reach node t from any node i, denoted by $\underline{c}_{it}$. Then, we prune any partial path from node s to node i with cost c_{si}, such that $c_{si} + \underline{c}_{it} > \bar{z}$.

In the arc-usage pruning strategy, we define an upper limit $\bar{u}$ on the number of paths in $\hat{\mathcal{Y}}$ that can use any arc (i,j). Let u_{ij} be the number of paths in $\hat{\mathcal{Y}}$ that use arc (i,j). When the search reaches node t, we add an s-t path to $\hat{\mathcal{Y}}$ and increase u_{ij} by one, for each arc (i,j) traversed in the path. Once $u_{ij} = \bar{u}$, we eliminate arc (i, j), forcing the pulse algorithm to explore paths that do not traverse arc (i, j). This strategy yields a diverse sample of s-t paths, which is desirable in our backward sampling framework.

Finally, we stop the sampling procedure once a maximum sample size limit $|\hat{\mathcal{Y}}|_{\max}$ is reached or once a time limit is exceeded.

3.3.2.2 Solving the Restricted Problem

We formulate the restricted problem $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ as a MIP. Let $\mathcal{P}^k$ be the set of arcs corresponding to the k^{th} path in sample $\hat{\mathcal{Y}}$, and let $c(\mathcal{P}^k)$ denote its cost. We formulate $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ as follows:

$$\max \quad z \tag{3.13}$$

$$\text{s.t.} \quad z \leq c(\mathcal{P}^k) + \sum_{(i,j)\in\mathcal{P}^k} d_{ij}x_{ij}, \quad \forall \mathcal{P}^k \in \hat{\mathcal{Y}}, \tag{3.14}$$

$$\mathbf{x} \in \mathcal{X}(\hat{\mathbf{w}}). \tag{3.15}$$

The objective function (3.13) maximizes z, which is constrained by (3.14) to be no more than the least cost path in $\hat{\mathcal{Y}}$, after considering delays caused by arc interdiction. Finally, constraints (3.15) ensure that we only consider feasible attacks in $\mathcal{X}(\hat{\mathbf{w}})$.

Observe that if our algorithm generates an attack $\hat{\mathbf{x}} \in \mathcal{X}(\hat{\mathbf{w}})$ having a perceived damage greater than $\bar{z}$, then $\bar{z}$ cannot be improved in the current iteration. In this case, our algorithm does not utilize the precise perceived damage value (beyond establishing that it exceeds $\bar{z}$). It is thus not necessary to optimize model (3.13)–(3.15) if we have proven that its objective exceeds $\bar{z}$, and so we add the objective target constraint $z \leq \bar{z} + \delta$, for a small constant $\delta > 0$, to model (3.13)–(3.15). This ensures that any attack $\hat{\mathbf{x}} \in \mathcal{X}(\hat{\mathbf{w}})$ with perceived damage strictly greater than $\bar{z}$ is sufficient to allow the overall algorithm to continue, even though $\hat{\mathbf{x}}$ may not optimize $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$.

Furthermore, because the $\mathbf{x}$-variables are binary-valued and $d_{ij} \geq 0$, $\forall(i, j) \in \mathcal{A}$, the addition of the objective target constraint allows us to revise (3.14) as follows, where $(\bullet)^+ = \max\{0, \bullet\}$:

$$z \leq c(\mathcal{P}^k) + \sum_{(i,j)\in\mathcal{P}^k} \min\{d_{ij}, (\bar{z} + \delta - c(\mathcal{P}^k))^+\}x_{ij}, \quad \forall \mathcal{P}^k \in \hat{\mathcal{Y}}. \tag{3.16}$$

Constraints (3.16) are at least as tight as (3.14). (Note that (3.16) corresponding to some $\mathcal{P}^k$ may persist in our interdiction model for a few iterations after $\bar{z} + \delta \leq c(\mathcal{P}^k)$. We therefore require the coefficients of the **x**-variables to be nonnegative in order to ensure the validity of (3.16).)

3.4 Capacitated Lot Sizing Interdiction Problem with Fortification

The capacitated lot sizing problem (CLSP) is a well-known NP-hard problem [Bitran and Yanasse, 1982, Florian et al., 1980] in which a facility manufactures a single product to satisfy a known demand over a finite planning horizon subject to production capacity constraints. Among the many CLSP studies in the literature, we note the seminal MIP formulation of Karmarkar et al. [1987], later extended by Eppen and Martin [1987] with a variable redefinition technique, and the branch-and-cut framework by Belvaux and Wolsey [2000]. For a comprehensive CLSP literature review see surveys by Karimi et al. [2003] and Brahimi et al. [2006].

In the CLSIPF production capacity at any time period could be lost (e.g., due to machine failures). The system operator can ensure that capacity is protected against loss for some time periods (e.g., by performing preventive maintenance). In this context, an "attack" is not necessarily due to a malicious adversary, but represents some bounded worst-case scenario on capacity loss.

3.4.1 Problem Statement

Formally, we define the CLSIPF as the problem of finding a subset of time periods to fortify, in order to minimize the total cost resulting from a worst-case attack that disables production capacity on some of the unprotected time periods. Let $\mathcal{T} = \{1, \ldots, |\mathcal{T}|\}$ be the set of time periods in the planning horizon. For each time period $t \in \mathcal{T}$, let d_t be the demand, C_t be the production capacity, and let c_t, f_t, h_t, and q_t be the production, setup, holding, and shortage cost, respectively. All parameters are assumed to be nonnegative.

Let $\mathbf{w} \in \mathcal{W}$ be the fortification decision variables and $\mathbf{x} \in \mathcal{X}(\mathbf{w})$ be the attack decision variables, where $\mathcal{W} \equiv \left\{\mathbf{w} : \mathbf{e}^{\intercal}\mathbf{w} \leq Q,\ \mathbf{w} \in \{0,1\}^{|\mathcal{T}|}\right\}$ establishes the defender's budget and ensures that the fortification variables are binary, and $\mathcal{X}(\mathbf{w}) \equiv \left\{\mathbf{x} : \mathbf{e}^{\intercal}\mathbf{x} \leq B,\ x_t \leq 1 - w_t\ \forall t \in \mathcal{T},\ \mathbf{x} \in \{0,1\}^{|\mathcal{T}|}\right\}$ ensures that a maximum of B unprotected time periods are attacked, and forces the attacker variables to be binary. Finally, let $\mathbf{y}$, $\mathbf{v}$, $\mathbf{I}$, and $\mathbf{s}$ be the third-stage decision variables modeling production, setup, inventory, and shortage, respectively. The CLSIPF can be formally stated as:

$$\begin{aligned}
\min_{\mathbf{w}\in\mathcal{W}} \max_{\mathbf{x}\in\mathcal{X}(\mathbf{w})} \min \quad & \sum_{t\in\mathcal{T}} c_t y_t + f_t v_t + h_t I_t + q_t s_t && && (3.17)\\
\text{s.t.} \quad & I_t = I_{t-1} + y_t + s_t - d_t, && \forall t\in\mathcal{T}, && (3.18)\\
& y_t \leq C_t v_t, && \forall t\in\mathcal{T}, && (3.19)\\
& v_t \leq 1 - x_t, && \forall t\in\mathcal{T}, && (3.20)\\
& y_t, I_t, s_t \geq 0, && \forall t\in\mathcal{T}, && (3.21)\\
& v_t \in \{0,1\}, && \forall t\in\mathcal{T}. && (3.22)
\end{aligned}$$

The objective function (3.17) minimizes the total cost after interdiction. Constraints (3.18) are inventory constraints, constraints (3.19) enforce production capacity limits, and constraints (3.20) forbid production on interdicted time periods. Constraints (3.21) and (3.22) place bounds and binary restrictions on the decision variables.

3.4.2 Solution Approach

In the following subsections we discuss the three components required for solving the CLSIPF: a sampling scheme, an approach for solving two-level CLSP interdiction problems restricted over a sample of third-stage solutions, and a method to solve third-stage CLSP problems.

3.4.2.1 Sampling Scheme

Let $\mathcal{S}$ denote a production plan (third-stage recourse solution) that specifies values for $\mathbf{y}$, $\mathbf{v}$, $\mathbf{I}$, and $\mathbf{s}$. To obtain a sample of production plans, we propose a simple random search that iteratively generates a random attack plan $\mathbf{x}^r$, and solves a MIP to compute the optimal recourse response given $\mathbf{x}^r$. In particular, $\mathbf{x}^r$ interdicts K time periods randomly selected among $\{0,\ldots,|\mathcal{T}|\}$. We then solve the following MIP given $\mathbf{x}^r$:

$$\min_{(\mathbf{y},\mathbf{v},\mathbf{I},\mathbf{s})\in\mathcal{Y}(\mathbf{x}^r)} \quad \sum_{t\in\mathcal{T}} c_t y_t + f_t v_t + h_t I_t + q_t s_t, \tag{3.23}$$

where $\mathcal{Y}(\mathbf{x}^r)$ is the third-stage feasible region defined by inserting $\mathbf{x}^r$ in constraints (3.18)–(3.22).

Let production plan $\mathcal{S}^* = \{\mathbf{y}^*, \mathbf{v}^*, \mathbf{I}^*, \mathbf{s}^*\}$ be an optimal solution to the MIP given an attack plan $\mathbf{x}^r$, and let $c(\mathcal{S}^*)$ denote its cost. If $c(\mathcal{S}^*) \leq \bar{z}$, then we add $\mathcal{S}^*$ to the sample, and otherwise we discard $\mathcal{S}^*$. We repeat this procedure for a prescribed number of iterations (regardless of how many production plans are added to the sample). Note that integer parameter K could be different from the attacker's budget B, and can take any value between $[0, |\mathcal{T}|]$. Large values of K result in a sample with more conservative production plans, which only produce during a few time periods, and are thus more difficult to interdict.

The repeated solution of MIPs in the sampling phase of this algorithm may ultimately be too computationally intensive to justify its use. We will demonstrate in our computational section that the solution of MIPs in this phase is justified. However, an alternative to this scheme would simply generate heuristic recourse solutions in response to randomly sampled attacks. The tradeoff thus involves the quality of sampled solutions (where higher quality samples tend to speed overall convergence) versus the time required to generate them.

3.4.2.2 Solving the Restricted Problem

As done in the SPIPF, we formulate the restricted problem $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ as a MIP. Let $\mathcal{S}^k = \{\mathbf{y}^k, \mathbf{v}^k, \mathbf{I}^k, \mathbf{s}^k\}$ denote production plan k in $\hat{\mathcal{Y}}$ and $\mathcal{T}(\mathcal{S}^k) \equiv \{t \in \mathcal{T} \mid y_t^k > 0\}$ be the set of time periods in which plan $\mathcal{S}^k$ produces a positive amount of items. We formulate $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ analogously to (3.13)–(3.15):

$$\max \quad z \tag{3.24}$$

$$\text{s.t.} \quad z \leq c(\mathcal{S}^k) + \sum_{t \in \mathcal{T}(\mathcal{S}^k)} M_t^k x_t, \quad \forall \mathcal{S}^k \in \hat{\mathcal{Y}}, \tag{3.25}$$

$$\mathbf{x} \in \mathcal{X}(\hat{\mathbf{w}}). \tag{3.26}$$

We use a suitably large cost M_t^k to penalize attacked production plans. To determine this cost, we decompose $\mathbf{y}^k$ into values $a_{tt}^k, \ldots, a_{t|\mathcal{T}|}^k$, $\forall t \in \mathcal{T}$, where a_{tj}^k denotes the amount produced at period t that satisfies demand at period j, for $j \geq t$. One possible way of adjusting a solution if an attack occurs at period t is to simply retain the previous solution, but with $y_t^k = 0$. As a result, there will be a savings of $f_t + c_t y_t^k$ due to eliminated fixed and variable costs, plus any holding costs that were incurred due to production in period t. However, without adjusting production at any other period, we would incur additional shortage costs of $q_j a^k_{tj}$ for each $j \geq t$. Accordingly, we define the cost penalty for any production plan $\mathcal{S}^k$ at time period t as

$$M^k_t = \left(\sum_{j \in \mathcal{T}: j \geq t} q_j a^k_{tj} \right) - f_t - c_t y^k_t - \left(\sum_{j \in \mathcal{T}: j > t} \sum_{l=t}^{j-1} h_l a^k_{tj} \right). \tag{3.27}$$

Proposition 13 shows that (3.25) remains valid when M^k_t is defined as in (3.27).

Proposition 13. *Consider any* $\mathbf{x} \in \mathcal{X}$ *and let* $\mathcal{S}^* \in \mathcal{Y}(\mathbf{x})$ *be its corresponding optimal recourse response. For any production plan* $\mathcal{S}^k$*, we have that* $c(\mathcal{S}^k) + \sum_{t \in \mathcal{T}(\mathcal{S}^k)} M^k_t x_t \geq c(\mathcal{S}^*)$*, where the* M*-values are defined in (3.27).*

Proof Let $\mathcal{S}'_k$ be a modification of $\mathcal{S}^k$ in which all the production from interdicted time periods is canceled, as described above. Because production is zero in solution $\mathcal{S}'_k$ at time periods interdicted by $\mathbf{x}$, then $\mathcal{S}'_k \in \mathcal{Y}(\mathbf{x})$, which implies that $c(\mathcal{S}'_k) \geq c(\mathcal{S}^*)$. Noting that $c(\mathcal{S}'_k) = c(\mathcal{S}^k) + \sum_{t \in \mathcal{T}(\mathcal{S}^k)} M^k_t x_t$, this completes the proof. ■

We use the objective target strategy introduced for the SPIPF in Section 3.2.2. Following the same logic in that section, we add the constraint $z \leq \bar{z} + \delta$ to model (3.24)–(3.26), which allows us to tighten (3.25) as follows:

$$z \leq c(\mathcal{S}^k) + \sum_{t \in \mathcal{T}(\mathcal{S}^k)} \min\{M^k_t, (\bar{z} + \delta - c(\mathcal{S}^k))^+\} x_t, \quad \forall \mathcal{S}^k \in \hat{\mathcal{Y}}. \tag{3.28}$$

Remark 8. *Recall that in our sampling strategy, we create recourse solutions that are optimal with respect to some attack vector* $\mathbf{x}$*. Hence, in those solutions, the* M^k_t*-parameters in (3.27) must not be negative, or else the recourse solution could be improved by simply eliminating production in period* t*. If exact optimization is not used to create recourse solutions in* $\hat{\mathcal{Y}}$*, then it is possible for some value of* M^k_t *to be negative. Constraint (3.28) remains valid in this case, but could be tightened by simply replacing the sampled solution with one in which* y_t *is modified to equal* 0 *whenever* $M^k_t < 0$*.*

3.4.2.3 Obtaining the Real Damage for an Attack

Calculating the real damage of an attack $\hat{\mathbf{x}}$ requires solving a CLSP in which the production capacity for time periods attacked by $\hat{\mathbf{x}}$ is set to zero. One simple approach solves the classical MIP model for the CLSP given attack plan $\hat{\mathbf{x}}$ stated in (3.23). Because the backward sampling framework does not require a specific solution method for the third-stage problem, we could employ any of the well-established methods for solving the CLSP, including the standard dynamic programming approach in which inventory at time t is used as state variable.

3.5 Computational Experiments

This section presents computational results on the SPIPF and the CLSIPF. We assess the performance of our algorithm on the SPIPF using randomly generated grid networks in Section 3.5.1 and on large-scale real road networks in Section 3.5.2. In Section 3.5.3 we analyze the effect of the defender's (attacker's) budget and the parameter ϵ on the performance of the algorithm for the SPIPF. In Section 3.5.4 we evaluate our algorithm on randomly generated CLSIPF instances.

We coded our algorithm in Java, using Eclipse SDK version 4.4.1, and executed the experiments on a machine having an Intel Core i7–3537U CPU (two cores) running at 2.00 GHz with 2 GB of RAM allocated to the Java Virtual Machine memory heap on Windows 8. We impose a time limit of four hours (14,400s) and solve all mathematical optimization problems using Gurobi 5.6. All instances and source code used in this section are available from the author at `http://people.clemson.edu/~jcsmith`.

3.5.1 Solving the SPIPF Over Directed Grid Networks

We generate directed grid networks with the same topology used by Israeli and Wood [2002] and Cappanera and Scaparra [2011]. These networks have a source node s, a sink node t, and $m \times n$ nodes arranged in a grid of m rows and n columns. There exists an arc from s to every node in the first column and an arc from every node in the last column to t. Also, arcs exist from each node in grid row r and column c to (existing) nodes in positions $(r+1, c)$, $(r-1, c)$, $(r, c+1)$, $(r+1, c+1)$, and $(r-1, c+1)$ provided that these are not vertical arcs in the first or last columns.

We build networks with sizes ranging from 10×10 to 60×60. For each network size we explore different (cost, delay) configurations in which arc costs (delays) are random integers uniformly distributed between $[1, c]$ ($[1, d]$), where c (d) denotes the maximum cost (delay). As done by Cappanera and Scaparra [2011], we explore the following (c, d) configurations: $(10, 5)$, $(10, 10)$, $(10, 20)$, $(100, 50)$, $(100, 100)$, and $(100, 200)$. For a fixed network size and (c, d) configuration, we generate ten instances with different random arc attributes for a total of $360 = 6 \times 6 \times 10$ different instances. We solve each instance six times with different Q values in $\{3, 4, 5, 7\}$ and B values in $\{3, 4, 5\}$, for a total of $2160 = 360 \times 6$ experiments. After tuning the algorithm parameters, we set the maximum sample size to 100, the sampling time limit $|\hat{\mathcal{Y}}|_{\max}$ to 1 second, the arc-usage upper limit to 20, threshold ϵ to 0.1, and δ to 1 (see (3.16)).

Tables 3.2 and 3.3 show the computational results for medium- and large-sized grid networks, respectively. The first five columns show grid size, number of nodes and arcs, and the defender's and attacker's budget (Q and B), respectively. For each of the six (c, d) configurations, the tables depict the average CPU time obtained over ten runs (Avg) and the largest CPU time obtained over those runs (Max).

Table 3.2: Computational time in CPU seconds for solving the SPIPF over medium-size grid networks

Instance	Nodes	Arcs	Q	B	Cost–delay configuration (c, d)											
					$(10, 5)$		$(10, 10)$		$(10, 20)$		$(100, 50)$		$(100, 100)$		$(100, 200)$	
					Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
10×10	102	416	3	3	0.1	0.3	0.1	0.2	0.1	0.2	0.1	0.2	0.1	0.3	0.1	0.2
			4	3	0.1	0.1	0.1	0.2	0.1	0.2	0.1	0.2	0.1	0.2	0.2	0.2
			3	4	0.1	0.2	0.2	0.5	0.2	0.5	0.3	0.5	0.3	0.7	0.3	0.7
			5	4	0.2	0.3	0.3	0.5	0.3	0.6	0.3	0.4	0.4	0.7	0.4	0.8
			4	5	0.3	0.4	0.5	1.1	0.8	2.2	0.6	1.5	1.0	2.9	1.1	2.8
			7	5	0.7	1.0	0.9	1.6	1.0	2.3	0.8	1.2	1.3	2.9	1.3	1.9
20×20	402	1826	3	3	0.3	0.9	0.5	1.9	0.6	3.3	0.7	2.0	0.7	1.3	0.8	2.0
			4	3	0.4	1.2	0.6	2.2	0.8	4.3	0.9	2.3	0.8	1.6	0.9	2.0
			3	4	0.6	1.5	1.1	3.2	1.1	5.1	2.2	5.1	2.4	5.9	2.4	8.0
			5	4	1.0	2.7	2.2	10.8	2.2	12.8	2.1	4.4	2.9	6.1	2.9	9.1
			4	5	1.8	5.6	4.2	18.2	4.1	16.8	6.5	16.1	8.3	22.4	9.7	33.4
			7	5	3.6	10.8	5.7	13.1	6.7	28.3	7.3	13.3	9.8	21.3	12.5	36.8
30×30	902	4236	3	3	0.8	1.1	1.2	3.6	1.9	7.3	1.8	7.5	2.0	3.9	2.3	6.0
			4	3	0.9	1.2	1.4	3.7	2.0	6.9	2.3	11.2	2.6	6.1	2.4	5.7
			3	4	1.6	2.6	3.8	14.9	6.0	25.6	4.4	15.0	5.5	13.3	6.3	15.5
			5	4	2.7	4.1	6.1	27.4	8.1	36.4	6.2	21.5	10.0	31.0	10.2	36.9
			4	5	5.2	11.8	15.9	77.2	23.2	94.8	15.9	60.2	24.2	61.1	27.1	73.2
			7	5	11.7	22.7	26.7	108.2	30.1	131.7	21.9	62.7	35.8	101.1	36.7	91.2

Table 3.2 shows that on average, our algorithm finds optimal solutions for the 10×10 and 20×20 networks in just a few seconds, and requires less than one minute to solve the 30×30 networks, which have more than 4000 arcs. The maximum execution times are close to the average times; even in the worst case (30×30 grids with $Q = 7$, $B = 5$, and $(c, d) = (10, 20)$), the algorithm terminates in just over two minutes.

Table 3.3: Computational time in CPU seconds for solving the SPIPF over large-size grid networks

Instance	Nodes	Arcs	Q	B	Cost–delay configuration (c, d)											
					$(10, 5)$		$(10, 10)$		$(10, 20)$		$(100, 50)$		$(100, 100)$		$(100, 200)$	
					Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
40×40	1602	7646	3	3	3.0	7.6	6.3	26.5	8.8	48.3	4.3	6.7	7.8	26.6	6.8	21.2
			4	3	3.5	8.1	7.5	35.2	9.6	51.4	5.2	8.2	9.3	37.6	7.9	26.7
			3	4	3.9	8.2	24.5	145.5	45.7	321.7	12.7	24.6	18.6	58.2	16.3	39.1
			5	4	6.8	12.9	34.2	196.6	54.0	376.6	16.0	31.4	28.3	127.2	24.4	91.8
			4	5	9.8	23.0	212.6	1786.3	194.0	1103.3	39.0	94.8	75.6	217.7	64.0	185.8
			7	5	23.4	61.7	211.4	1400.6	330.2	2153.0	56.9	101.2	181.4	1155.0	111.0	497.8
50×50	2502	12,056	3	3	6.7	16.7	10.2	18.5	15.6	43.0	19.3	45.6	38.3	141.7	47.6	130.1
			4	3	7.6	17.2	11.9	25.7	16.2	43.4	20.1	45.9	38.1	132.6	47.6	134.9
			3	4	10.5	50.1	30.8	126.3	35.7	190.2	28.0	61.8	114.6	717.4	80.7	166.0
			5	4	13.2	59.8	33.4	122.8	43.3	231.1	44.5	116.4	244.7	1979.0	117.2	444.0
			4	5	24.2	147.3	121.6	588.7	112.5	787.0	161.3	973.0	197.1	715.6	317.7	1049.9
			7	5	44.0	208.1	210.8	1355.7	142.1	847.1	344.8	1702.7	527.7	3614.9	375.1	1385.0
60×60	3602	17,466	3	3	13.3	23.2	24.3	53.9	38.8	64.7	29.5	46.6	60.6	230.6	82.3	236.8
			4	3	14.2	22.1	32.8	122.8	50.5	146.8	32.4	47.4	64.8	259.4	86.7	273.8
			3	4	17.0	32.2	64.5	272.4	79.8	291.9	34.6	52.5	107.2	339.8	149.7	512.6
			5	4	24.0	65.2	119.8	723.0	125.5	663.6	55.0	122.0	142.0	460.0	202.8	910.8
			4	5	35.6	67.5	223.5	1263.4	236.3	1070.4	155.9	578.8	1479.0	8103.9	1587.3	11,052.4
			7	5	81.2	273.9	346.0	1477.3	491.8	3382.5	317.1	1056.6	1575.9	8079.2	1733.4	11,177.9

Table 3.3 shows that, on average, the algorithm terminates in less than six minutes for the 40×40 networks, in less than nine minutes for the 50×50 networks, and in less than 29 minutes for the 60×60 networks, for any combination of c, d, Q, and B examined. The maximum execution times are larger relative to the average CPU times on these instances, and some of them require roughly four hours of computational time over the 60×60 networks. This behavior is expected, considering that these networks have more than 3000 nodes and 17,000 arcs. Table 3.3 suggests that the instances become more difficult as d grows larger relative to c and when the cost (delay) values increase (implying that $(c, d) = (100, 200)$ are typically the most challenging instances). Finally, an increase in the attacker's budget tends to have a greater impact on the computation time than an increase in the defender's budget. We further study this idea in Section 3.5.3.

We compare our approach (Sampling) to the current state-of-the-art algorithm by Cappanera and Scaparra [2011] over medium-sized instances, who graciously provided their code for the purposes of this comparison. They present two versions of their implicit enumeration algorithm, which are based on a shortest path formulation (SPI) and on a k-shortest-paths formulation (KSPI). The former performs better when the set of s-t paths whose cost is less than or equal to the objective value obtained at the root node of the enumeration tree is large, and the latter performs better when this set is small. For our test instances, SPI strengthened with the variable fixing rules and acceleration strategies proposed by Cappanera and Scaparra [2011] outperforms KSPI.

Table 3.4 shows the results for this comparison. Here, the "Avg" column depicts the average CPU time in seconds, computed only among the instances solved within the time limit. As before, "Max" refers to maximum CPU seconds out of the 60 instances solved for the row, and "# solved" gives the number of instances solved within the four-hour time limit.

Table 3.4: Comparing the backward sampling algorithm to the state-of-the-art algorithm for SPIPF

Instance	Nodes	Arcs	Q	B	Sampling			SPI		
					Avg	Max	# solved	Avg	Max	# solved
			3	3	0.1	0.3	60	1.9	4.6	60
10×10	102	416	5	4	0.3	0.8	60	30.9	162.8	60
			4	5	0.7	2.9	60	67.5	284.2	60
			3	3	0.6	3.3	60	26.7	305.1	60
20×20	402	1,826	5	4	2.2	12.8	60	1128.4	7723.4	60
			4	5	5.8	33.4	60	2495.2	>14,400	56
			3	3	1.7	7.5	60	766.9	12,728.8	60
30×30	902	4,236	5	4	7.2	36.9	60	4857.3	>14,400	45
			4	5	18.6	94.8	60	4256.8	>14,400	26

Table 3.4 shows that our algorithm compares favorably to SPI, consistently reducing computational

time by more than two orders of magnitude both in terms of average and maximum execution times, for any combination of (Q, B) examined. Moreover, our sampling algorithm solves all instances within the time limit while SPI solves 56 instances when $(Q, B) = (4, 5)$ on the 20×20 networks, 45 instances when $(Q, B) = (5, 4)$ on the 30×30 networks, and 26 instances when $(Q, B) = (4, 5)$.

3.5.2 Solving the SPIPF Over Real Road Networks

We use the road networks from Washington (DC), Rhode Island (RI), and New Jersey (NJ) presented by Raith and Ehrgott [2009]. These networks range from 9559 nodes and 39,377 arcs to 330,386 nodes and 1,202,458 arcs. For each road network, Raith and Ehrgott [2009] define nine randomly selected s-t pairs. We define c_{ij} as the arc distance and set $d_{ij} = 10{,}000$, $\forall (i, j) \in \mathcal{A}$. For each network and s-t pair we explore six budget configurations for a total of $162 = 3 \times 9 \times 6$ experiments. We use the same algorithm parameters as in the directed grid networks. Table 3.5 shows the results for these experiments.

Table 3.5: Computational time in CPU seconds for solving the SPIPF over road networks

Instance	Nodes	Arcs	Q	B	Avg	Max	# solved
DC	9559	39,377	3	3	45.8	124.9	9
			4	3	50.6	127.1	9
			3	4	92.2	402.9	9
			5	4	103.0	374.8	9
			4	5	492.8	2829.5	9
			7	5	450.1	1906.4	9
RI	53,658	192,084	3	3	284.5	756.0	9
			4	3	295.6	817.3	9
			3	4	800.7	4925.1	9
			5	4	946.2	5974.9	9
			4	5	560.1	>14,400	8
			7	5	754.0	>14,400	8
NJ	330,386	1,202,458	3	3	6743.8	10,551.9	9
			4	3	6345.8	>14,400	8
			3	4	6526.8	>14,400	8
			5	4	6964.3	>14,400	8
			4	5	7354.6	>14,400	8
			7	5	8452.6	>14,400	8

Table 3.5 shows that the algorithm solves all DC instances to optimality within the time limit. The average CPU time for these instances is less than nine minutes and the worst execution time is well under one hour. On the RI network, the algorithm solves all instances with $B \leq 4$ and solves all but one instance each when $(Q, B) = (4, 5)$ and $(7, 5)$. Average CPU times are less than 15 minutes among the instances solved to optimality within the time limit, for any choice of (Q, B). On the NJ network, the algorithm solves all instances with $Q = B = 3$ and solves all but one instance in each set corresponding to the other (Q, B) combinations. Average times are roughly two hours

among the instances solved to optimality.

3.5.3 Sensitivity Analysis for SPIPF

We conduct additional experiments to measure the effect of increasing the defender's (attacker's) budget on the execution time. For this purpose, we use a subset of ten 30×30 grid networks with $(c, d) = (100, 200)$ and solve instances that result from fixing an intermediate value of $Q = 4$ ($B = 4$) and increasing B (Q). The results of this experiment are shown in Tables 3.6 and 3.7.

Table 3.6: Measuring the effect of increasing B on CPU time over a subset of 30×30 grid networks

Q	B	Avg	Max	# solved
4	2	1.1	2.9	10
4	3	2.6	6.0	10
4	4	7.2	20.4	10
4	5	27.3	73.4	10
4	6	62.7	169.6	10
4	7	241.4	704.8	10
4	8	1208.5	4878.1	10
4	9	4901.8	>14,400	9
4	10	4750.8	>14,400	2

Table 3.7: Measuring the effect of increasing Q on CPU time over a subset of 30×30 grid networks

Q	B	Avg	Max	# solved
2	4	5.2	13.1	10
4	4	6.8	18.5	10
6	4	10.1	29.0	10
8	4	15.8	56.4	10
10	4	16.4	48.2	10
12	4	17.7	44.4	10
14	4	24.3	73.7	10
16	4	26.0	53.5	10
18	4	30.8	57.2	10
20	4	36.8	55.3	10

Table 3.6 shows that increasing B for a fixed value of Q has a dramatic impact on the computational time. Increasing B from 5 to 7 produces an increase of roughly one order of magnitude in the average execution time, and while the algorithm is able to solve all ten instances having $B = 8$, it is only able to solve two instances having $B = 10$. On the contrary, Table 3.7 shows that increasing Q for a fixed value of B has a less pronounced impact on the computational time. Even for $Q = 20$, the algorithm finds optimal solutions to all instances in less than one minute. This behavior may be explained by noting that increasing B directly affects the difficulty of the restricted MIP problems, which are solved in every iteration, while increasing Q affects only the fortification problem.

We also conduct an experiment that measures the effect of the parameter ϵ on the performance of the algorithm. For this purpose, we select a set of difficult instances, i.e., those requiring roughly one to three hours of computational time when $\epsilon = 0$ (listed in the first column of Table 3.8).

For each ϵ-value considered we report the number of defense plans evaluated (# of $\hat{\mathbf{w}}$ evaluated), the number of defense plans added to the waiting list $\mathcal{L}$ (# of $\hat{\mathbf{w}}$ interrupted), the number of restricted interdiction problems solved (# of $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ solved), the total time spent solving fortification problems (time$_F$), the total time spent solving restricted interdiction problems (time$_I$), and the total execution time.

Table 3.8 shows the results for this experiment, where the "Metric" column shows the performance metrics evaluated, and the last five columns show the results for ϵ-values ranging from 0 (i.e., not using the waiting list) to 0.2. First, observe that the time spent solving fortification problems is negligible compared to the time spent solving restricted interdiction problems, which is the most time-consuming task in the algorithm. It is thus vital to use ϵ to limit the number of restricted interdiction problems that the algorithm must solve.

Table 3.8: Profiling the algorithm on a subset of hard instances

Instance	(c, d)	Metric	$\epsilon = 0$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.15$	$\epsilon = 0.2$
		# of $\hat{\mathbf{w}}$ evaluated	59	91	95	93	112
		# of $\hat{\mathbf{w}}$ interrupted	0	77	78	73	87
		# of $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ solved	1745	1505	1383	1376	1532
40×40-0	$(10, 20)$	Time$_F$ (s)	2.1	3.2	3.2	3.0	3.7
		Time$_I$ (s)	4256.8	3493.3	2083.2	2773.7	2790.1
		Total time (s)	4320.9	3557.5	2146.5	2837.7	2849.3
		# of $\hat{\mathbf{w}}$ evaluated	68	107	122	76	120
		# of $\hat{\mathbf{w}}$ interrupted	0	87	97	57	99
		# of $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ solved	2218	2257	1910	2083	2045
60×60-5	$(10, 20)$	Time$_F$ (s)	6.7	9.3	9.6	6.1	9.2
		Time$_I$ (s)	3474.7	3769.8	2942.6	2923.0	3144.9
		Total time (s)	3531.0	3828.1	2999.5	2979.0	3204.9
		# of $\hat{\mathbf{w}}$ evaluated	81	120	142	148	160
		# of $\hat{\mathbf{w}}$ interrupted	0	101	123	127	134
		# of $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ solved	1270	1190	1226	1201	1239
60×60-1	$(100, 100)$	Time$_F$ (s)	8.7	11.7	12.5	13.3	15.3
		Time$_I$ (s)	6977.2	5130.5	8076.5	7513.5	6693.4
		Total time (s)	7074.0	5229.4	8162.9	7595.5	6773.6
		# of $\hat{\mathbf{w}}$ evaluated	55	122	128	135	96
		# of $\hat{\mathbf{w}}$ interrupted	0	93	99	111	82
		# of $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ solved	3043	2176	2110	2296	2394
60×60-6	$(100, 200)$	Time$_F$ (s)	5.2	10.5	11.4	12.0	7.7
		Time$_I$ (s)	10,902.3	3846.2	4303.6	5007.5	4973.1
		Total time (s)	10,989.7	3931.1	4388.2	5109.1	5068.6

Table 3.8 shows that there is not a single ϵ-value that achieves the best performance over all the instances. However, small positive values for ϵ (i.e., $\epsilon = 0.05$ and $\epsilon = 0.1$) produce significant computational improvements over other values of ϵ on average. As expected, the number of defense plans evaluated (and interrupted) increases for larger values of ϵ. However, the number of restricted interdiction problems solved over this subset of difficult instances is always smaller when using the waiting list ($\epsilon > 0$) than when we set $\epsilon = 0$.

3.5.4 Solving the CLSIPF

We generate random instances for the CLSIPF having $|\mathcal{T}| \in \{10, 20, 30, 40\}$. For each choice of $|\mathcal{T}|$ we generate ten instances in which d_t, C_t, c_t, f_t, and q_t are random integers uniformly distributed between $[10, 210]$, $[150, 200]$, $[5, 10]$, $[44, 64]$, and $[2c_t, 3c_t]$, respectively, and h_t is randomly selected in the interval $[0.3, 0.5]$. These intervals were defined based on the parameter structure of a classical instance introduced by Peterson and Silver [1979]. For each instance we consider all possible choices of $Q \in \{3, 5\}$ and $B \in \{2, 3, 4\}$, for a total of $240 = 4 \times 10 \times 6$ experiments. After tuning the algorithm parameters, we set the integer parameter K used to control the sampling scheme to $2B$, the number of iterations for the sampling procedure to 50, threshold ϵ to 0.1, and δ to 1 (see (3.28)). Because each iteration of our sampling scheme in Section 3.4.2.1 generates at most one sample, the initial sample size will be between 0 and 50.

We compare our approach, in which we directly solve the third-stage problem (MIP) to an alternative solution method in which the third-stage CLSP problem is transformed into a shortest path problem (SP) using a standard dynamic programming approach. Table 3.9 shows the results for these experiments. Here, the "Algorithm" column indicates the approach used. As before, the "Avg" column shows the average CPU time in seconds, computed only among the instances solved within the time limit, "Max" refers to maximum CPU time over ten runs, and "# sol" gives the number of instances solved within the four-hour time limit.

Table 3.9: Computational time in CPU seconds for solving the CLSIPF over randomly generated problem instances

Algorithm	Q	B	$\|\mathcal{T}\| = 10$			$\|\mathcal{T}\| = 20$			$\|\mathcal{T}\| = 30$			$\|\mathcal{T}\| = 40$		
			Avg	Max	# sol	Avg	Max	# sol	Avg	Max	# sol	Avg	Max	# sol
	3	2	97.7	376.7	10	1180.0	2507.6	10	1827.2	3929.1	10	4464.4	6587.7	10
	5	2	111.5	392.1	10	1186.8	2467.1	10	1907.4	4077.3	10	4635.5	6560.6	10
SP	3	3	85.9	202.9	10	1978.5	3361.6	10	9610.3	14051.6	9	-	-	0
	5	3	131.5	459.8	10	1997.7	3440.6	10	10070.3	14237.3	9	-	-	0
	3	4	110.0	197.8	10	5170.7	8529.8	10	-	-	0	-	-	0
	5	4	176.5	499.4	10	5284.2	8238.6	10	-	-	0	-	-	0
	3	2	1.8	3.8	10	5.7	8.9	10	17.7	33.8	10	46.7	69.0	10
	5	2	1.4	2.6	10	6.3	11.3	10	19.3	30.7	10	53.4	70.0	10
MIP	3	3	2.8	5.9	10	26.4	72.8	10	153.3	603.3	10	702.4	1667.7	10
	5	3	2.2	4.4	10	29.7	76.7	10	169.3	561.1	10	994.6	2032.9	10
	3	4	2.9	4.8	10	115.3	265.6	10	1711.1	>14,400	9	4933.4	>14,400	5
	5	4	2.2	3.4	10	109.6	177.3	10	2622.2	12,182.0	10	10,086.0	>14,400	4

Table 3.9 shows that SP solves all instances having $|\mathcal{T}| \leq 20$, 38 out of 60 instances having $|\mathcal{T}| = 30$, and 20 out of 60 instances having $|\mathcal{T}| = 40$, within the time limit. However, the sampling method that directly uses the MIP recourse problem solves all but one instance having $|\mathcal{T}| \leq 30$, and 49 out of 60 instances having $|\mathcal{T}| = 40$, within the time limit. Solving instances having $|\mathcal{T}| \leq 20$ requires on average less than two minutes, and even the worst execution times are less than five minutes. For instances having $|\mathcal{T}| = 30$, MIP requires on average less than one hour of CPU time; however, one instance cannot be solved to optimality within four hours when $(Q, B) = (3, 4)$. For instances having $|\mathcal{T}| = 40$, MIP performs well when $B \leq 3$, solving all instances in less than 20 minutes. However, when $B = 4$ it fails to solve 11 instances (five when $Q = 3$ and six when $Q = 5$) within the time limit. These results show that MIP outperforms SP over all instance sizes and (Q, B) configurations, reducing computational time by about two orders of magnitude. Also, as observed in the SPIPF, an increase in the attacker's budget has a dramatic impact on the computational time. For example, when $|\mathcal{T}| = 30$, increasing B by one results in about a tenfold increase in the average CPU time. On the contrary, increasing Q tends to have a minor effect on the computational time.

Chapter 4

Solving the Traveling Salesman Problem with Interdiction and Fortification

4.1 Problem Statement

We formally define the TSPIF on a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A} \subset \mathcal{N} \times \mathcal{N}$ is the set of arcs. For each arc $(i, j) \in \mathcal{A}$, let $c_{ij} \geq 0$ be the cost of traversing an uninterdicted arc and $d_{ij} \geq 0$ be the additional cost (delay) incurred when traversing an interdicted arc. Thus, the total cost of traversing an interdicted arc is $c_{ij} + d_{ij}$. Let $\mathbf{w} \in \mathcal{W}$ be the fortification decision variables, where $\mathcal{W} \equiv \left\{\mathbf{w} \mid \mathbf{T}\mathbf{w} \leq \mathbf{b},\ \mathbf{w} \in \{0,1\}^{|\mathcal{A}|}\right\}$ ensures that the variables are binary and enforces a set of linear constraints that limits the extent to which the defender can fortify arcs. Let $\mathbf{x} \in \mathcal{X}(\mathbf{w})$ be the attack decision variables, where $\mathcal{X}(\mathbf{w}) \equiv \left\{\mathbf{x} \mid \mathbf{T}'\mathbf{x} \leq \mathbf{b}',\ x_{ij} \leq 1 - w_{ij},\ \forall (i,j) \in \mathcal{A},\ \mathbf{x} \in \{0,1\}^{|\mathcal{A}|}\right\}$ forces the $\mathbf{x}$-variables to be binary, ensures that only unfortified arcs are interdicted, and imposes a set of linear constraints that models the ability of the attacker to interdict arcs. Finally, let $\mathbf{y}$ be a vector of binary arc-selection variables such that $y_{ij} = 1$ if arc (i, j) is used in the optimal tour identified for the recourse problem, and $y_{ij} = 0$ otherwise, for all $(i, j) \in \mathcal{A}$. We restrict $\mathbf{y} \in \mathcal{Y}$, where $\mathcal{Y}$ includes the set of binary vectors $\mathbf{y}$

that correspond to TSP solutions in $\mathcal{G}$. The TSPIF can be formally stated as:

$$z^* = \min_{\mathbf{w} \in \mathcal{W}} \max_{\mathbf{x} \in \mathcal{X}(\mathbf{w})} \min_{\mathbf{y} \in \mathcal{Y}} \sum_{(i,j) \in \mathcal{A}} (c_{ij} + d_{ij} x_{ij}) y_{ij}, \tag{4.1}$$

where in the objective function (4.1), the original cost of any arc is increased by d_{ij} when the arc is attacked (i.e., $x_{ij} = 1$).

The TSP with interdiction and fortification may occur in a defense scenario in which troops need to monitor a set of locations and return to a base. If the troops wish to perform these tasks as quickly as possible, they solve a TSP. An adversary might attempt to impair the troops' movement by degrading (interdicting) roadways or bridges. The adversary's actions could be anticipated by the troops, who secure pathways ahead of time by stationing personnel or other resources to deter interdictions. Hence, the troops act first to fortify arcs, after which the adversary interdicts unfortified arcs, and the troops respond by solving a TSP on the resulting network. The TSPIF may also arise in a civilian application in which a set of areas has been affected by a disaster such as an earthquake. Nodes now correspond to areas that need emergency supplies, and the defender deploys a relief vehicle to provide supplies to the nodes. Fortification is performed ahead of time in a planning stage to either improve bridges or roads in anticipation of a disaster, or to pre-position assets (e.g., ferries or pontoon bridges) that would facilitate travel in the event that the disaster damages existing infrastructure. In the worst case, damage will be inflicted on a set of arcs that had not been fortified in the planning stage.

Since the TSP is a nonconvex combinatorial problem, it is not practical to combine the second- and third-stage problems, because no polynomial-size strong dual formulation for the TSP is known to exist. Furthermore, enumerating all (exponentially-many) feasible recourse solutions is computationally prohibitive. Therefore, a new approach is required to solve problems like the TSPIF. In Chapter 3 we propose a backward sampling framework (BSF) for interdiction problems with fortification in which the recourse problem can take any form. We solve defender-attacker-defender games played over shortest path (SPIF) and capacitated lot sizing problems (CLSIF). The BSF significantly outperforms prior approaches for solving the SPIF, and yields an effective mechanism for solving interdiction and fortification problems. However, the design of practically effective algorithms for interdiction and fortification problems defined over a strongly $\mathcal{NP}$-hard problem like the TSP is still an open research question, despite the abundance of research separately developed for the TSP and for network interdiction.

In this chapter we explore the solution of the TSPIF via the BSF. Our contribution analyzes two problem restrictions, where both restrictions serve as a heuristic for the TSPIF and model the situation in which the defender lacks the computational resources to compute an optimal response to an attack. We also demonstrate that these restrictions are instrumental in reducing computational time for solving the TSPIF within an exact two-phase approach. We then use these developments to tailor a BSF-based approach for this problem, and demonstrate the efficacy of our approach using TSP instances from the literature.

The remainder of this chapter is organized as follows. Section 4.2 presents the BSF for the TSPIF. Section 4.3 discusses our proposed sampling approaches. Section 4.4 describes alternative restrictions for the recourse problem. Given these algorithm components, Section 4.5 presents our proposed two-phase approach for the TSPIF. Section 4.6 presents our computational experiments.

4.2 Solving the TSPIF

We now adapt the backward sampling framework presented in Chapter 3 for the TSPIF. Algorithm 7 presents the proposed approach. Let $\mathcal{C}$ be the set of covering constraints added to the (outer) fortification problem and $\mathcal{W}(\mathcal{C}) = \{\mathbf{w} \in \mathcal{W} \mid \mathbf{w} \text{ satisfies all constraints in } \mathcal{C}\}$. The algorithm starts with an empty set of covering constraints and a global upper bound $\bar{z} = \infty$. Line 2 selects an initial sample of tours over $\mathcal{G}$. As we will discuss in Section 4.3, the size and diversity of the initial sample plays an important role in the overall efficiency of the algorithm. The outer while-loop (line 4) is executed until the fortification problem becomes infeasible. Line 5 selects a feasible defense $\hat{\mathbf{w}}$ and lines 6–21 solve the corresponding problem $\mathcal{Q}(\hat{\mathbf{w}})$ with our proposed sampling approach. The inner while-loop (line 7) is executed until the global upper bound cannot be further reduced by the current choice of $\hat{\mathbf{w}}$. Line 9 obtains an upper bound on $z^I(\hat{\mathbf{w}})$ by solving the restricted problem $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}}^i)$ and obtaining an attack vector $\hat{\mathbf{x}} \in \mathcal{X}(\hat{\mathbf{w}})$. Line 10 solves a TSP given the fixed attack $\hat{\mathbf{x}}$. The optimal tour, $\hat{\mathbf{y}}^*$, identified in this step yields a lower bound. Line 11 defines the sample at the next iteration as the solutions in the previous sample along with $\hat{\mathbf{y}}^*$. Line 12 checks if UB_i reduces the current global upper bound; if so, then line 13 updates the global upper bound, and line 14 removes from the sample all tours whose cost is greater than $\bar{z}$. Line 15 determines if attack $\hat{\mathbf{x}}$ is critical by checking if $LB_i \geq \bar{z}$, and if so, line 16 adds a covering constraint to the fortification problem. Finally, if the optimality condition is satisfied (line 18), then line 19 updates the incumbent solution.

Algorithm 7 Backward sampling framework for the TSPIF

```
1: Set the global upper bound $\bar{z} = \infty$ and covering constraints set $\mathcal{C} = \emptyset$        ▷ Initialization
2: Select $\hat{\mathcal{Y}}^1 \subseteq \mathcal{Y}$ as a sampling of tours from $\mathcal{G}$, and compute their objective values
3: Set counter $i = 0$
4: while $\mathcal{W}(\mathcal{C}) \neq \emptyset$ do        ▷ Main while-loop
5:     Select any $\hat{\mathbf{w}} \in \mathcal{W}(\mathcal{C})$
6:     Initialize $LB_i = -\infty$
7:     while $LB_i < \bar{z}$ do
8:         Set $i = i + 1$
9:         Solve $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}}^i)$, set $UB_i = z^I(\hat{\mathbf{w}}, \hat{\mathcal{Y}}^i)$, and record an optimal solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$
10:        Solve $LB_i = z^R(\hat{\mathbf{x}}) = \min_{\mathbf{y} \in \mathcal{Y}} \sum_{(i,j) \in \mathcal{A}} (c_{ij} + d_{ij}\hat{x}_{ij}) y_{ij}$ and obtain an optimal tour $\hat{\mathbf{y}}^*$
11:        Set $\hat{\mathcal{Y}}^{i+1} = \hat{\mathcal{Y}}^i \cup \{\hat{\mathbf{y}}^*\}$
12:        if $UB_i < \bar{z}$ then
13:            Update global upper bound $\bar{z} \leftarrow UB_i$
14:            Remove from $\hat{\mathcal{Y}}^{i+1}$ all tours having cost greater than $\bar{z}$
15:        else if $LB_i \geq \bar{z}$ then        ▷ A critical attack has been identified
16:            Add the covering constraint $\mathbf{w}^\intercal \hat{\mathbf{x}} \geq 1$ to $\mathcal{C}$
17:        end if
18:        if $LB_i = UB_i = \bar{z}$ then
19:            Update the incumbent solution $(\bar{\mathbf{w}}, \bar{\mathbf{x}}, \bar{\mathbf{y}}) \leftarrow (\hat{\mathbf{w}}, \hat{\mathbf{x}}, \hat{\mathbf{y}})$
20:        end if
21:    end while
22: end while
23: Return $(\bar{\mathbf{w}}, \bar{\mathbf{x}}, \bar{\mathbf{y}})$
```

4.3 Sampling TSP Tours

The only condition required on the initial sample to ensure that our algorithm terminates with an optimal solution is that $\hat{\mathcal{Y}}^1 \subseteq \mathcal{Y}$. However, $\hat{\mathcal{Y}}^1$ has an important effect on the performance of the BSF since both the tightness of the upper bounds obtained by solving restricted problems $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}}^i)$ and the number of constraints in formulation (3.13)–(3.15) depend on the choice of $\hat{\mathcal{Y}}^1$.

We now describe desirable features for a choice of $\hat{\mathcal{Y}}^1$. Tours in $\hat{\mathcal{Y}}^1$ should be diverse in the sense that they do not contain too many of the same arcs, or else the attacker could interdict many tours in the sample by interdicting a few arcs common to those tours. Tours in $\hat{\mathcal{Y}}^1$ should also be optimal or near-optimal solutions to the TSP when $\hat{\mathbf{x}} = \mathbf{0}$. Finally, if $|\hat{\mathcal{Y}}^1|$ is too large, then formulation (3.13)–(3.15) will be large as well, and may potentially be too difficult to solve.

Attempting to achieve a balance between the desirable features listed, we propose a genetic algorithm (GA) based on the NSGA-II framework [Deb et al., 2002] in which each solution has two objectives, both of which are to be minimized. The first objective is the tour length with respect to the uninterdicted graph. The second objective is a measure of the individual solution's similarity to some reference set of TSP tours. We include in our reference set all tours whose length is not more than ϵ percent greater than the best tour-length seen so far. We compute our second objective as the number of times each solution arc appears in the reference set, divided by the total number of arcs in the set population. A solution that has no arcs in common with any tour in the reference set is in some manner "maximally different" and is desirable, having a second objective of 0.

Each solution is represented using the bidimensional array from [Buriol et al., 2004]. The population size is set at 400 and the algorithm stops if the total number of evaluations reaches 2,000,000 or the total number of iterations reaches 1000. The initial population is formed by seeding with the solution to the original problem found using LKH. One solution is taken directly from the LKH result, and the rest are formed from randomly selected pairwise-interchanges of nodes from this original tour.

In order to create the offspring population, 90% of the offspring population is filled using the strategic arc crossover [Buriol et al., 2004] with a requirement that any such generated solutions have tour lengths within 20% of the best seen tour length. The rest of the population is filled using random pairwise interchange of the initial LKH-generated tour. Each solution then undergoes mutation with 0.1% probability, where mutations are performed via a pair-wise interchange of nodes on the tour. Each solution's tour length is computed. The reference set is updated, including only those solutions within $\epsilon = 5\%$ of the best seen tour length but allowing duplicated tours to remain. Then the second (similarity) objective can be computed for each solution. Once both objectives are computed for each solution, the next population is formed using the front framework from NSGA-II.

Alternatively, one simple option is to seed $\hat{\mathcal{Y}}^1$ with one TSP tour. In this case we solve the TSP when $\hat{\mathbf{x}} = \mathbf{0}$, and use only that tour in our initial sample.

4.4 Alternative Restrictions for the Recourse Problem

We now present two restrictions for the recourse problem that model the case in which the defender must compute a quick response to an attack, rather than expending the computational resources required to compute an optimal response. These restrictions are also instrumental in devising a more computationally effective exact TSPIF algorithm.

The first restriction is inspired by very large-scale neighborhood search algorithms Ahuja et al. [2002]. We start with a base tour $\mathbf{y}^*$ obtained by solving the TSP to optimality given costs $c_{ij},\ \forall (i,j) \in \mathcal{A}$. For the symmetric case, the defender recourse responses are restricted to belong to the set of tours that can be obtained by performing a series of so-called disjoint 2-opt swaps on $\mathbf{y}^*$. (See Remark 9 for an extension of this idea for the asymmetric case.) To understand the concept of disjoint 2-opt swaps, we first order the nodes in tour $\mathbf{y}^*$ as $v_{(1)}, v_{(2)}, \dots, v_{(|\mathcal{N}|)}$. Let $v_{(|\mathcal{N}|+1)} \equiv v_{(1)}$. A 2-opt swap is performed by identifying two tour indices i and j, where $i \geq 1$, $j \leq |\mathcal{N}| + 1$, and $i + 3 \leq j$. The tour formed by a 2-arc swap replaces arcs $(v_{(i)}, v_{(i+1)})$ and $(v_{(j-1)}, v_{(j)})$ with arcs $(v_{(i)}, v_{(j-1)})$ and $(v_{(i+1)}, v_{(j)})$ in the original tour. Arcs $(v_{(k)}, v_{(k+1)}),\ k = i+1, \dots, j-2$, would now be traversed in the opposite direction after the symmetric 2-opt arc swap. (See Figure 4.1 for an illustration.) A set of 2-opt swaps are disjoint if the 2-opt swaps are performed over indices $(i_1, j_1), (i_2, j_2), \dots, (i_k, j_k)$ such that $j_h \leq i_{h+1},\ \forall h = 1, \dots, k-1$.

We model the disjoint 2-opt swap restriction on $\mathbf{y}^*$ by transforming the recourse problem into a shortest path problem defined over a new graph $\mathcal{G}' = (\mathcal{N}', \mathcal{A}')$. The set of nodes $\mathcal{N}' = \{1, \dots, |\mathcal{N}| + 1\}$ represents each ordered node in tour $\mathbf{y}^*$, where $|\mathcal{N}| + 1$ is a duplicate of the first node. The set of arcs $\mathcal{A}' = \mathcal{A}'_1 \cup \mathcal{A}'_2$ comprises two kinds of arcs. Arcs in $\mathcal{A}'_1 = \{(i, i+1) \mid i \in \mathcal{N}',\ i \leq |\mathcal{N}|\}$ correspond to arcs in the original tour $\mathbf{y}^*$. Accordingly, we define their cost as $c'_{ij} = c_{v_{(i)}v_{(j)}}$ and delay for a given attack $\mathbf{x}$ as $d'_{ij} = d_{v_{(i)}v_{(j)}} x_{v_{(i)}v_{(j)}}$, for all $(i,j) \in \mathcal{A}'_1$. Arcs in $\mathcal{A}'_2 = \{(i,j) \mid \forall i = 1, \dots, |\mathcal{N}| - 2,\ j = i+3, \dots, |\mathcal{N}| + 1\}$ represent a 2-opt swap as illustrated in Figure 4.1. For arc $(i,j) \in \mathcal{A}'_2$ the cost and delay for a given attack $\mathbf{x}$ are defined as:

$$c'_{ij} = c_{v_{(i)}v_{(j-1)}} + c_{v_{(i+1)}v_{(j)}} + \sum_{k=i+2}^{j-1} c_{v_{(k)}v_{(k-1)}} \qquad \forall (i,j) \in \mathcal{A}'_2 \tag{4.2}$$

$$d'_{ij} = d_{v_{(i)}v_{(j-1)}} x_{v_{(i)}v_{(j-1)}} + d_{v_{(i+1)}v_{(j)}} x_{v_{(i+1)}v_{(j)}} + \sum_{k=i+2}^{j-1} d_{v_{(k)}v_{(k-1)}} x_{v_{(k)}v_{(k-1)}} \qquad \forall (i,j) \in \mathcal{A}'_2, \tag{4.3}$$

respectively. Note that the third term in equations (4.2) and (4.3) accounts for the arcs traversed from $v_{(i+1)}$ to $v_{(j-1)}$ in the original tour, though in the reverse direction.

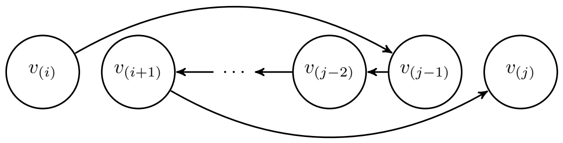


Figure 4.1: Graphical representation of a 2-arc swap for the symmetric case

Every path from 1 to $|\mathcal{N}|+1$ in $\mathcal{G}'$ corresponds to a TSP tour in the original graph $\mathcal{G}$. These paths encode the original tour $\mathbf{y}^*$ along with all solutions that can be obtained via disjoint 2-opt swaps from that tour.

Remark 9. *For the asymmetric case, arcs in $\mathcal{A}_1'$ are given as before. Arcs $(i,j) \in \mathcal{A}_2'$ represent a 3-arc swap that replaces arcs $(v_{(i)}, v_{(i+1)})$, $(v_{(q)}, v_{(q+1)})$, and $(v_{(j-1)}, v_{(j)})$ with arcs $(v_{(i)}, v_{(q+1)})$, $(v_{(j-1)}, v_{(i+1)})$, and $(v_{(q)}, v_{(j)})$ in the original tour, where*

$$q \in \operatorname*{argmin}_{i+1 \le \bar{q} \le j-2} \{c_{v_{(i)}v_{(\bar{q}+1)}} + c_{v_{(j-1)}v_{(i+1)}} + c_{v_{(\bar{q})}v_{(j)}} - c_{v_{(i)}v_{(i+1)}} - c_{v_{(q)},v_{(q+1)}} - c_{v_{(j-1)}v_{(j)}}\}. \tag{4.4}$$

Note that q is chosen in (4.4) so that the perturbed route corresponding to arc $(i,j) \in \mathcal{A}_2'$ is as close to optimal as possible with respect to the uninterdicted graph. An alternative implementation might create $j-i-3$ parallel arcs that connect i and j, one corresponding to each possible choice of q in the set $\{i+1, \dots, j-2\}$. This transformation expands the recourse solution space, but at the expense of creating a much larger graph $\mathcal{G}'$. $\square$

Cost and delay attributes are defined analogous to the symmetric case:

$$c'_{ij} = c_{v_{(i)}v_{(q+1)}} + c_{v_{(j-1)}v_{(i+1)}} + c_{v_{(q)}v_{(j)}} + \sum_{k=q+1}^{j-2} c_{v_{(k)}v_{(k+1)}} + \sum_{k=i+1}^{q-1} c_{v_{(k)}v_{(k+1)}} \qquad \forall (i,j) \in \mathcal{A}_2'$$

$$\begin{aligned} d'_{ij} = {} & d_{v_{(i)}v_{(q+1)}} x_{v_{(i)}v_{(q+1)}} + d_{v_{(j-1)}v_{(i+1)}} x_{v_{(j-1)}v_{(i+1)}} + d_{v_{(q)}v_{(j)}} x_{v_{(q)}v_{(j)}} \\ & + \sum_{k=q+1}^{j-2} d_{v_{(k)}v_{(k+1)}} x_{v_{(k)}v_{(k+1)}} + \sum_{k=i+1}^{q-1} d_{v_{(k)}v_{(k+1)}} x_{v_{(k)}v_{(k+1)}} \qquad \forall (i,j) \in \mathcal{A}_2'. \end{aligned}$$

Figure 4.2 illustrates the 3-arc swap represented by arc $(i,j) \in \mathcal{A}_2'$ for the asymmetric case.

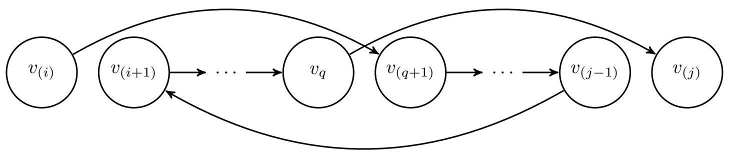


Figure 4.2: Graphical representation of a 3-arc swap for the asymmetric case

We also consider a second restriction that constrains the defender to solve the recourse problem using the Lin-Kernighan heuristic (LKH) [Lin and Kernighan, 1973], which is one of the best performing TSP heuristics. Note that the first restriction is modeled as a network flow problem, which can then be solved using existing network interdiction and fortification algorithms [Cappanera and Scaparra, 2011, Bayrak and Bailey, 2008, Fulkerson and Harding, 1977, Golden, 1978, Held et al., 2005, Israeli and Wood, 2002]. On the other hand, the second restriction will most likely yield a stronger upper bound given the success of the LKH in obtaining near-optimal TSP solutions, but the restriction cannot practically be modeled as a linear program.

4.5 Two-Phase Approach

We devise a two-phase approach that first solves a restriction of the TSPIF to identify a set of covering constraints, an initial sample of tours, and an upper bound on z^*. This first phase is based on the solution of a heuristic TSP restriction, and provides a warm start to exactly solve the problem using Algorithm 7 in a second phase.

Algorithm 8 describes our proposed two-phase approach. Line 1 solves a restriction of the TSPIF using a variation of Algorithm 7 in which one of the proposed restrictions in Section 4.4 is used to compute recourse solutions. We record an optimal solution, $(\mathbf{w}^0, \mathbf{x}^0, \mathbf{y}^0)$, and the set of all critical attacks explored, $\hat{\mathcal{X}}$, solving this restricted problem. Note that even though attacks in $\hat{\mathcal{X}}$ are critical for the restricted problem, they are not necessarily critical for the original (exact) problem. Line 2 solves to optimality the interdiction problem corresponding to $\mathbf{w}^0$ (using our inner sampling-based algorithm) and line 3 updates the upper bound and incumbent solution accordingly. Lines 5–11 explore all attacks $\hat{\mathbf{x}} \in \hat{\mathcal{X}}$ to generate the initial sample and possibly identify covering constraints. For every $\hat{\mathbf{x}} \in \hat{\mathcal{X}}$, line 6 solves a TSP to find an optimal recourse tour $\hat{\mathbf{y}}^*$ and calculates $z^R(\hat{\mathbf{x}})$. Line 7 adds $\hat{\mathbf{y}}^*$ into the initial sample. Line 8 determines if attack $\hat{\mathbf{x}}$ is critical by checking if $z^R(\hat{\mathbf{x}}) \geq \bar{z}$. If so, then line 9 adds a covering constraint to the fortification problem. Finally, line

12 continues solving the TSPIF using Algorithm 7, starting with an initial sample $\hat{\mathcal{Y}}^1$, a covering constraint set $\mathcal{C}$, and an upper bound $\bar{z}$.

Algorithm 8 Two-phase algorithm for the TSPIF

```
 1: Obtain an optimal solution $(\mathbf{w}^0, \mathbf{x}^0, \mathbf{y}^0)$ to a restriction of the TSPIF and let $\hat{\mathcal{X}}$ be the set of all
    critical attacks identified in the solution process                          ▷ Begin phase one
 2: Obtain an optimal solution $(\mathbf{x}^*, \mathbf{y}^*)$ to $\mathcal{Q}(\mathbf{w}^0)$
 3: Set $\bar{z} = z^I(\mathbf{w}^0)$ and update the incumbent solution $(\bar{\mathbf{w}}, \bar{\mathbf{x}}, \bar{\mathbf{y}}) \leftarrow (\mathbf{w}^0, \mathbf{x}^*, \mathbf{y}^*)$
 4: Initialize sample $\hat{\mathcal{Y}}^1 = \emptyset$ and covering constraints set $\mathcal{C} = \emptyset$            ▷ Begin phase two
 5: for $\hat{\mathbf{x}} \in \hat{\mathcal{X}}$ do
 6:     Solve $z^R(\hat{\mathbf{x}}) = \min_{\mathbf{y} \in \mathcal{Y}} \sum_{(i,j) \in \mathcal{A}} (c_{ij} + d_{ij}\hat{x}_{ij}) y_{ij}$ and obtain an optimal tour $\hat{\mathbf{y}}^*$
 7:     Add $\hat{\mathbf{y}}^*$ into $\hat{\mathcal{Y}}^1$
 8:     if $z^R(\hat{\mathbf{x}}) \geq \bar{z}$ then                            ▷ A critical attack has been identified
 9:         Add the covering constraint $\mathbf{w}^\intercal \hat{\mathbf{x}} \geq 1$ to $\mathcal{C}$
10:     end if
11: end for
12: Solve the TSPIF using Algorithm 7 warm-started with $\hat{\mathcal{Y}}^1$, $\mathcal{C}$, and $\bar{z}$
```

4.6 Computational Results

We coded our algorithm in Java, using Eclipse SDK version 4.4.2, and executed all computational experiments on a machine having an Intel Core i7–3537U CPU (two cores) running at 2.00 GHz with 8 GB of RAM on Windows 8. We solve the TSP instances using CONCORDE [Applegate et al., 1998], all other optimization problems using Gurobi 5.6.0, and use the LKH implementation provided by [Helsgaun, 2000].

Our set of test instances consists of 100 instances derived from 10 networks (5 symmetric and 5 asymmetric) from TSPLIB [Reinelt, 1991]. In every instance the cost coefficient for arc $(i, j) \in \mathcal{A}$ corresponds to the distance between nodes i and j in the original network. The delay coefficient for arc $(i, j) \in \mathcal{A}$ is initially taken to be a random integer uniformly distributed between $[1, c_{ij}]$. We generate 10 instances with random arc delay coefficients for each of the original networks.

We define the defender's feasible region as $\mathcal{W} \equiv \{\mathbf{w} \mid \mathbf{e}^\intercal\mathbf{w} \leq Q,\ \mathbf{w} \in \{0,1\}^{|\mathcal{A}|}\}$, which enforces a cardinality constraint on the number of fortified arcs and ensures that the variables are binary. We also define $\mathcal{X}(\mathbf{w}) \equiv \{\mathbf{x} \mid \mathbf{e}^\intercal\mathbf{x} \leq B,\ x_{ij} \leq 1 - w_{ij},\ \forall (i,j) \in \mathcal{A},\ \mathbf{x} \in \{0,1\}^{|\mathcal{A}|}\}$, which ensures that a maximum of B unfortified arcs are interdicted, and forces the **x**-variables to be binary.

In Section 4.6.1 we examine the performance of four versions of our proposed algorithm. In

Section 4.6.2 we assess our proposed restrictions both as heuristic approaches for the TSPIF and within the exact two-phase approach given in Algorithm 8. In Section 4.6.3 we study the effect of varying the defender's budget, the attacker's budget, and the delay coefficients on the performance of the algorithm.

4.6.1 Solving the TSPIF

We compared four versions of the proposed algorithm. The first one (one-tour sampling) initiates the sample as a single tour that optimizes the TSP when no arcs have been interdicted. The second one (GA sampling) implements the proposed GA sampling scheme. The third one (two-phase 2-opt) implements the two-phase algorithm in Section 4.5 with the 2-opt restriction, and the fourth one (two-phase LKH) implements the two-phase algorithm with the LKH restriction. We solve each instance three times with different budget configurations (Q, B) in $\{(3,3),(5,4),(4,5)\}$.

Table 4.1 shows the results for these experiments. The first five rows present results for symmetric instances and the last five rows for asymmetric instances. The first two columns present the name of the network (which includes the number of nodes) and the number of arcs, respectively. Columns 3 and 4 show the defender and attacker budget. The remaining columns present the average CPU time in seconds obtained over 10 instances derived from the same network (Avg), the largest CPU time obtained over those runs (Max), and the number of instances solved within a four-hour time limit (# solved) for the four versions of the algorithm. We calculate the average CPU time only among the instances solved within the time limit and report an overall CPU time average for the symmetric and asymmetric instances. For each row, the best average and maximum CPU times are highlighted in bold.

Table 4.1: Comparing four different versions of our proposed algorithm for solving the TSPIF

Instance	Arcs	Q	B	One-tour sampling			GA sampling			Two-phase 2-opt			Two-phase LKH		
				Avg	Max	# solved	Avg	Max	# solved	Avg	Max	# solved	Avg	Max	# solved
bayg29	406	3	3	10	22	10	15	30	10	12	24	10	**7**	**16**	10
		5	4	48	103	10	61	123	10	47	105	10	**40**	**87**	10
		4	5	99	302	10	119	218	10	101	213	10	**82**	**167**	10
hk48	1128	3	3	10	24	10	18	37	10	13	26	10	**7**	**17**	10
		5	4	63	213	10	94	330	10	72	224	10	**50**	**156**	10
		4	5	151	**666**	10	212	878	10	197	821	10	**140**	721	10
brazil58	1653	3	3	15	30	10	24	48	10	19	60	10	**12**	**23**	10
		5	4	63	149	10	107	261	10	72	232	10	**55**	**138**	10
		4	5	137	**368**	10	215	738	10	159	527	10	**118**	458	10
eli76	2850	3	3	32	53	10	42	72	10	43	74	10	**22**	**51**	10
		5	4	174	**295**	10	231	460	10	174	**295**	10	**119**	298	10
		4	5	379	**797**	10	565	1294	10	445	892	10	**378**	1172	10
gr96	4560	3	3	222	523	10	238	628	10	235	554	10	**124**	**389**	10
		5	4	1913	9088	10	2061	9534	10	1938	8250	10	**1411**	**7762**	10
		4	5	3351	>14,400	9	3938	>14,400	9	3391	>14,400	9	**2249**	>14,400	9
Overall average				444			529			461			**321**		
Asymmetric instances															
br17	272	3	3	**2**	3	10	3	3	10	3	3	10	**2**	**2**	10
		5	4	6	7	10	**5**	**6**	10	8	10	10	**5**	**6**	10
		4	5	7	9	10	**6**	**8**	10	10	11	10	**6**	**8**	10
p43	1806	3	3	222	287	10	197	248	10	194	242	10	**111**	**153**	10
		5	4	409	620	10	421	681	10	464	589	10	**246**	**361**	10
		4	5	565	690	10	556	735	10	606	804	10	**328**	**493**	10
ry48p	2256	3	3	1007	1193	10	976	1102	10	1267	1657	10	**452**	**995**	10
		5	4	3237	4462	10	3269	4579	10	3666	4810	10	**1280**	**1833**	10
		4	5	6481	8576	10	6648	9353	10	7386	10,321	10	**2758**	**5471**	10
ft53	2756	3	3	53	79	10	54	92	10	66	116	10	**19**	**27**	10
		5	4	194	338	10	206	371	10	202	364	10	**104**	**215**	10
		4	5	405	940	10	420	1010	10	484	833	10	**199**	**499**	10
ftv64	4160	3	3	230	356	10	210	329	10	270	408	10	**95**	**184**	10
		5	4	737	1278	10	706	1088	10	838	1435	10	**370**	**647**	10
		4	5	859	1630	10	921	1828	10	984	1674	10	**429**	**659**	10
Overall average				961			973			1097			**427**		

Table 4.1 shows that two-phase LKH outperforms the other algorithms both in terms of the average and maximum CPU times. For the symmetric instances, two-phase LKH is on average about 40% faster than one-tour sampling and two-phase 2-opt, and about 65% faster than GA sampling. However, there is one instance from network gr96 that none of the algorithms solve within the time limit. For the harder asymmetric instances, two-phase LKH is on average more than two times faster than the other algorithms. The maximum CPU times follow a similar behavior.

Comparing the one-phase algorithms, one-tour sampling outperforms GA sampling on the symmetric instances, while on the asymmetric instances their performance is roughly the same. These results suggest that for the TSPIF, it is preferable to use a simple sampling scheme having a small sample size rather than an elaborate one that leads to larger sample sizes.

4.6.2 Assessing the Effectiveness of the Proposed Restrictions

Table 4.2 compares algorithms two-phase 2-opt and two-phase LKH when $Q = 4$ and $B = 5$, which was the most difficult budget configuration in Table 4.1. We omitted the one gr96 instance not solved to optimality. Column "z^*" presents the average optimal objective value obtained over the instances derived from the same network. The remaining columns show the average upper bound obtained at the end of phase one ($\bar{z}$), the average objective function value gap, calculated as $(\bar{z} - z^*)/z^* \times 100$ (% Gap), the average CPU time in seconds for phase one (I), the average CPU time for phase two (II), the average total CPU time (Total), and the number of covering constraints added at the end of phases one and two (Cuts).

Table 4.2: Comparing the performance of the proposed restrictions within the two-phase approach

		Two-phase 2-opt						Two-phase LKH					
Instance	z^*	$\bar{z}$	% Gap	Time (s)		Cuts		$\bar{z}$	% Gap	Time (s)		Cuts	
				I	II	I	II			I	II	I	II
bayg29	1702	1722	1.20	17	84	9	21	1702	0	80	2	23	0
hk48	12,057	12,140	0.68	60	137	5	25	12,057	0	136	4	26	0
brazil58	26,349	26,477	0.49	49	110	6	22	26,349	0	112	6	29	0
eli76	560	563	0.70	44	401	1	36	560	0	365	13	39	0
gr96	56,951	57,183	0.41	749	2643	4	26	56,951	0	2157	92	28	0
	Overall average		0.69	184	675	5	26		0	570	24	29	0
					Asymmetric instances								
br17	40	41	1.75	2	8	1	15	40	0	5	2	16	0
p43	5625	5630	0.08	107	499	1	19	5626	0.01	72	257	19	2
ry48p	14,884	14,991	0.72	1188	6198	1	23	14,885	0.01	610	2148	21	2
ft53	7185	7243	0.81	78	407	3	28	7185	0	167	32	27	0
ftv64	1921	1929	0.41	140	843	2	34	1922	0.05	47	381	37	1
	Overall average		0.75	303	1591	2	24		0.01	180	564	24	1

Table 4.2 shows that for symmetric instances, two-phase 2-opt quickly obtains near-optimal heuristic solutions with an objective function gap less than 1% on average. However, the total time to find an optimal solution by two-phase 2-opt is larger than the time required by two-phase LKH, which finds an optimal solution for every instance at the end of phase one. The 2-opt restriction identifies only a small number of covering constraints compared to the LKH restriction.

We conclude that for symmetric instances the 2-opt restriction is the best choice for a stand-alone heuristic, and the LKH restriction is better when embedded in our two-phase exact algorithm. For the asymmetric instances, the LKH restriction outperforms the 2-opt restriction both as a stand-alone heuristic and within our exact algorithm, finding heuristic solutions with a smaller average gap in less computational time. Both algorithms require considerably more time for phase two on the asymmetric instances, due to the increased difficulty in solving asymmetric TSPs.

4.6.3 Sensitivity Analysis

We conduct additional sensitivity analysis experiments related to the defender's budget, Q, the attacker's budget, B, and the range of the arc delay coefficient. For this purpose, we select a subset of 10 symmetric instances based on network eli76 and 10 asymmetric instances based on network ftv64, and begin by solving each instance with intermediate values of $Q = 4$ and $B = 4$. We then vary B and Q to determine the impact that these parameters have on computational time and objective function value. For this experiment we use the two-phase LKH algorithm since it is the best performer among the proposed algorithms. Table 4.3 presents the results of this experiment. The first two columns show the value of Q and B, respectively. Columns 3–6 present results for symmetric instances and columns 7–10 for asymmetric instances. As before, the "Avg" column shows the average CPU time in seconds, computed only among the instances solved within the time limit, "Max" presents the maximum CPU time over ten runs, "# solved" shows the number of instances solved within the four-hour time limit, and z^* refers to the average optimal objective value obtained over the instances derived from the same network.

Table 4.3: Measuring the effect of Q and B on execution time and objective

Q	B	eli76 (symmetric)				ftv64 (asymmetric)			
		Avg	Max	# solved	z^*	Avg	Max	# solved	z^*
0	4	20	57	10	560	21	43	10	1933
2	4	63	134	10	558	95	238	10	1915
4	4	133	382	10	556	242	467	10	1907
6	4	276	699	10	554	454	723	10	1899
8	4	352	486	10	553	676	1087	10	1892
10	4	679	1127	10	552	1004	1501	10	1887
4	2	14	18	10	547	67	130	10	1875
4	4	132	381	10	556	241	464	10	1907
4	6	2193	5874	10	563	594	832	10	1935
4	8	5823	>14,400	6	570†	1641	3354	10	1959

†: Average optimal objective value computed only among the instances solved within the time limit.

Table 4.3 shows that increasing the attacker's budget has a dramatic effect on the execution time of the algorithm. For the symmetric instances, the computational times increase from 14 seconds to over 2000 seconds as B grows from 2 to 6, and only six out of ten instances are solved to optimality when $B = 8$. For the asymmetric instances the computational time increases by roughly a factor of three when increasing B by two units. On the other hand, increasing the defender's budget has a less pronounced effect on the computational time.

Regarding the optimal objective value, increasing the defender's budget by ten units decreases z^* by about 1.5% for the symmetric instances and 2.4% for the asymmetric instances. Increasing the attacker's budget by six units results in an objective value increase of roughly 4% for both the symmetric and the asymmetric instances. Note that in the experiments depicted in Table 4.3, the average value for z^* is a convex decreasing function of Q, and a concave increasing function of B. However, this behavior is not reflected by each individual instance, and so no general claim of convexity or concavity follows.

We also conduct experiments to measure the effect of increasing the arc delay coefficient range on the execution times. For this purpose, we generate new instances based on symmetric network bayg29 and asymmetric network p43. We generate 20 instances (10 symmetric and 10 asymmetric) having random arc delay coefficients uniformly distributed between $[1, 2c_{ij}]$, 20 instances having delay coefficients between $[1, 3c_{ij}]$, and 20 instances having delay coefficients between $[1, M]$, where $M = 10 \max_{(i,j)\in\mathcal{A}}\{c_{ij}\}$. The latter delay configuration models the case in which an arc may become unavailable when interdicted. Table 4.4 presents results over this new set of instances. As before, we use intermediate values of $Q = 4$ and $B = 4$.

Table 4.4: Measuring the effect of varying the delay coefficient range on execution time

Delay configuration	bayg29 (symmetric)				p43 (asymmetric)			
	Avg	Max	# solved	z^*	Avg	Max	# solved	z^*
$[1, c_{ij}]$	25	46	10	1686	185	363	10	5624
$[1, 2c_{ij}]$	119	173	10	1712	214	357	10	5624
$[1, 3c_{ij}]$	344	875	10	1722	225	337	10	5626
$[1, M]$	>14,400	>14,400	0	-	749	1505	10	5633

Table 4.4 shows that for the symmetric instances, the computational time increases by roughly a factor of 15 as the delay range increases from $[1, c_{ij}]$ to $[1, 3c_{ij}]$, and none of the instances for which the delay coefficients are between $[1, M]$ terminate within the four-hour time limit. On the contrary, for the asymmetric instances the computational time exhibits a moderate increase with respect to the delay coefficient range and the instances having delay coefficients between $[1, M]$ are solved on average in about 12 minutes. The increased difficulty of solving symmetric instances as the range of the d-parameters grows may be due to the amount of similar tours that exist in the sample, which we expect to be considerably greater for symmetric instances than for asymmetric instances. The attacker can therefore interdict all tours in the sample, significantly increasing their costs due to the comparatively large delay coefficients. This results in poor upper bounds from solving the attacker's problem, which ultimately leads to increased computational times.

To mitigate the extent to which similar tours are included in the sample, we modify the objective function in phase one of our algorithm to include a penalty function based on sample diversity. Let u_{ij} be the number of tours in the sample that use arc (i, j). For a given attack $\hat{\mathbf{x}}$, we set the third-stage objective function in phase one as

$$\min_{\mathbf{y} \in \mathcal{Y}} \sum_{(i,j) \in \mathcal{A}} (c_{ij} + d_{ij}\hat{x}_{ij})y_{ij} + Pu_{ij}y_{ij}, \tag{4.5}$$

where P is an arbitrary constant penalty. Table 4.5 presents results of this experiment. After fine-tuning the two-phase LKH algorithm, we set P equal to 1/2.

Table 4.5: Imposing a penalty on the number of times an arc is used in the sample

Delay configuration	bayg29 (symmetric)				p43 (asymmetric)			
	Avg	Max	# solved	z^*	Avg	Max	# solved	z^*
$[1, c_{ij}]$	39	65	10	1686	346	607	10	5624
$[1, 2c_{ij}]$	122	184	10	1712	455	645	10	5624
$[1, 3c_{ij}]$	250	520	10	1722	462	553	10	5626
$[1, M]$	5180	6826	10	1759	1562	1886	10	5633

Table 4.5 shows that including the penalty in the objective function greatly reduces the computational time for solving the symmetric instances having delay coefficients between $[1, M]$, which are now solved in less than 2 hours. The variation in the execution times is almost negligible when the delay coefficients lie between $[1, c_{ij}]$ and $[1, 2c_{ij}]$, and there is roughly a 1.4 speedup when the delay coefficients lie between $[1, 3c_{ij}]$. On the other hand, including the penalty in the objective function has a negative effect on the execution time for the asymmetric instances, which in the worst case increases by roughly a factor of 2. This behavior illustrates the importance of achieving a balance between tour quality and diversity in the sample.

Chapter 5

A Binary Decision Diagram Based Algorithm for Solving a Class of Integer Two-Stage Stochastic Programs

5.1 Problem Statement

Two-stage stochastic programming is an approach for solving optimization problems under uncertainty in which a decision maker sequentially selects two sets of variables. In the first stage, the decision maker selects a vector of decisions before the realization of the uncertain parameters. In the second stage (or recourse problem), the decision maker determines the remaining variable values in response to the first-stage variables and to the realization of the uncertain parameters. The goal is to minimize the first-stage cost plus the expected second-stage cost. We examine two-stage stochastic programs of the following form:

$$z^* = \min \quad \mathbf{c}^\mathsf{T}\mathbf{x} + \mathbb{E}_{\omega\in\Omega}\left[\mathcal{Q}(\mathbf{x},\omega)\right] \tag{5.1a}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{x} \geq \mathbf{b} \tag{5.1b}$$

$$\mathbf{x} \in \{0,1\}^{n_1}, \tag{5.1c}$$

with $\mathcal{Q}(\mathbf{x}, \omega)$ defined as the optimal objective function value to the following problem:

$$\begin{aligned} \text{SP}(\mathbf{x}, \omega): \quad \mathcal{Q}(\mathbf{x}, \omega) = \min \quad & f(\omega)^\intercal \mathbf{y} && (5.2a)\\ \text{s.t.} \quad & W(\omega)\mathbf{y} \geq h(\omega) - T(\omega)\mathbf{x} && (5.2b)\\ & \mathbf{y} \in \{0,1\}^{n_2}, && (5.2c) \end{aligned}$$

where $\mathbf{x}$ is an n_1-dimensional vector of first-stage variables and $\mathbf{y}$ is an n_2-dimensional vector of recourse variables. Here, ω is a random vector from a probability space $(\Omega, \mathcal{F}, \mathcal{P})$, $\mathbf{A} \in \mathbb{R}^{m_1 \times n_1}$, $f(\omega) \in \mathbb{R}^{n_2}$, $W(\omega) \in \{0,1\}^{m_2 \times n_2}$, and all other data has conforming dimensions. In addition to assuming that $W(\omega)$ is binary, we assume that $h(\omega) \in \{0,1\}^{m_2}$ and $T(\omega) \in \{0,1\}^{m_2 \times n_1}$ is a matrix such that $\sum_{j=1}^{n_1} T(\omega)_{ij} \leq 1$ for all $i = 1, \ldots, m_2$, $\omega \in \Omega$, and $\sum_{i=1}^{m_2} T(\omega)_{ij} \leq 1$ for all $j = 1, \ldots, n_1$, $\omega \in \Omega$. The latter assumption ensures that at most one first-stage variable interacts with any given second stage constraint, and vice versa.

We adopt the following assumptions:

1. For all $\mathbf{x} \in \mathcal{X} = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} \geq \mathbf{b},\ \mathbf{x} \in \{0,1\}^{n_1}\}$ and $\omega \in \Omega$, there exists a feasible solution to $\text{SP}(\mathbf{x}, \omega)$.

2. ω follows a distribution with finite support $\Omega = \{\omega^1, \ldots, \omega^K\}$ with $\mathcal{P}(\omega = \omega^k) = p^k$.

Our first assumption ensures *relatively complete recourse*, which along with the binariness of $\mathbf{x}$ and $\mathbf{y}$, requires that the recourse problem has a finite optimum. This assumption is not limiting, because binary artificial variables can be added to the formulation to ensure the existence of a feasible solution for any $\mathbf{x}$ and ω. Our second assumption states that we will employ discrete probability distributions, as common in the literature. These assumptions lead to the following deterministic equivalent monolithic formulation, in which expectation is represented by a weighted sum over all possible realizations of ω, known as *scenarios*.

$$\begin{aligned} \min \quad & \mathbf{c}^\intercal \mathbf{x} + \sum_{k=1}^{K} p^k f(\omega^k)^\intercal \mathbf{y}^k && (5.3a)\\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} && (5.3b) \end{aligned}$$

$$W(\omega^k)\mathbf{y}^k \geq h(\omega^k) - T(\omega^k)\mathbf{x} \qquad \forall k = 1, \ldots, K \tag{5.3c}$$

$$\mathbf{x} \in \{0,1\}^{n_1} \tag{5.3d}$$

$$\mathbf{y}^k \in \{0,1\}^{n_2} \qquad \forall k = 1, \ldots, K. \tag{5.3e}$$

Note that problem (5.3) is a large block-angular integer programming problem having complicating variables $\mathbf{x}$.

Problems of the form (5.3) can be seen as a type of stochastic set covering formulation. Constraints (5.3c) represent the stochastic set-covering conditions that must be satisfied in each scenario. By setting certain x-variables equal to 1 in the first stage, however, the constraints corresponding to those x-variables do not need to be satisfied within the second-stage problem. Noting that constraints (5.3b) and (5.3d) constrain the first-stage variables, the problem is to determine which of the set-covering constraints to satisfy before uncertainty is realized (using $\mathbf{x}$), and in each scenario, how to satisfy the remaining constraints by using the second-stage variables $\mathbf{y}$. Later in this chapter, we will consider a stochastic vertex covering problem. In this problem, we can remove some edges in the first stage with costs given by the $\mathbf{c}$-vector, and then solve a weighted vertex covering problem (with scenario-dependent weights) in the second stage over edges that still need to be covered.

The contributions we make in this chapter are as follows. One, we provide a modeling mechanism for this class of problems that reformulates the second-stage integer program as a shortest-path problem using binary decision diagrams (BDDs), albeit at the expense of an exponential number of nodes. Two, we show how to limit the size of these reformulations based on variable-ordering strategies within the BDD, relating the size of the resulting shortest-path problems to the branchwidth of an associated hypergraph. Three, we investigate methods for strengthening Benders' inequalities stemming from our reformulation, and show their effectiveness on stochastic vertex cover instances.

The remainder of this chapter is organized as follows. Section 5.2 presents our proposed second-stage problem reformulation. Section 5.3 describes our Benders' decomposition algorithm along with strengthening strategies for the optimality cuts. Section 5.4 presents our computational experiments on stochastic vertex cover problems.

5.2 Reformulating the Recourse Problem

The algorithm we describe in this chapter is based on a transformation of the recourse problem into a linear program parameterized by first-stage decision variables. We then develop a Benders' decomposition approach for the equivalent transformed problem. Section 5.2.1 presents the proposed recourse problem reformulation. Section 5.2.2 describes how to limit the size of the reformulation using a branch decomposition heuristic.

5.2.1 Formulating the Recourse Problem via Binary Decision Diagrams

5.2.1.1 Dynamic Programming Formulation

We consider a dynamic programming (DP) formulation for problem $\mathrm{SP}(\hat{\mathbf{x}}, \omega)$, which consists of a *state space*, a set of *transition functions*, and a set of *cost functions*. A dynamic program sequentially determines values for decision variables in a series of stages. The outcome of these decisions is modeled by states that, at a given stage i, store information about the state of the system after assigning values for variables $1, \ldots, i-1$. The state space is the set of all admissible states, which for our problems includes an initial (or root) state, a terminal state, and an infeasible state. Transition functions determine how the system transitions between states. Cost functions establish the cost incurred for any given transition.

We define states as m_2-dimensional binary vectors in which each component corresponds to one of the structural constraints in (5.2b). A state component $j \in \{1, \ldots, m_2\}$ equals 1 if the corresponding constraint has not been satisfied using the variables chosen so far, and equals 0 otherwise. For a given scenario $\omega^k \in \Omega$, let $\mathcal{S}(\omega^k)$ be the state space, where $r(\omega^k) = h(\omega^k)$ is the root state, $t = \mathbf{0}$ is the terminal state, and $\{\}$ is the infeasible state. We denote by $\mathbf{s}^i$ the state of the system at stage i, i.e., before assigning a value to variable i. For modeling purposes, we consider both x- and y-variables in our DP formulation, even though x-variables are fixed parameters in the second-stage problem. (Including $\mathbf{x}$ as fixed decision variables will ultimately allow us to gain sensitivity information about $\mathrm{SP}(\mathbf{x}, \omega^k)$ with respect to $\mathbf{x}$.) Let $\boldsymbol{\gamma}$ be an ordering of x- and y-variables. The system transitions from a given state $\mathbf{s}^i \in S(\omega^k)$ to a new state according to the following transition functions:

$$\mathbf{s}^{i+1} = \phi_i(\mathbf{s}^i, \gamma_i, \omega^k), \quad \forall i = 1, \ldots, n_1 + n_2, \tag{5.4}$$

where γ_i denotes the i^{th} variable in ordering $\boldsymbol{\gamma}$, the initial state $\mathbf{s}^1 = r(\omega^k)$ is given, and $\phi_i(\mathbf{s}^i, \gamma_i, \omega^k)$ is defined below. Let $\mathbf{t}(\omega^k)_q$ and $\mathbf{w}(\omega^k)_q$ be the q^{th} column of matrices $T(\omega^k)$ and $W(\omega^k)$, respectively. Also, let $p_x(q)$ be the position of x_q in the ordering $\boldsymbol{\gamma}$, $\forall q = 1, \dots, n_1$, define $p_y(q)$ as the position of y_q in $\boldsymbol{\gamma}$, $\forall q = 1, \dots, n_2$, and let s^i_j be the j^{th} component of state vector $\mathbf{s}^i$. Let

$$U^i_j = \sum_{q \in \{1,\dots,n_1\}: p_x(q) > i} T(\omega^k)_{jq} + \sum_{q \in \{1,\dots,n_2\}: p_y(q) > i} W(\omega^k)_{jq} \tag{5.5}$$

denote the sum of coefficients corresponding to undecided variables, other than the variable corresponding to γ_i, in constraint j at stage i. Defining $(\bullet)^+ = \max\{0, \bullet\}$, the state transition functions are defined as follows.

$$\phi_i(\mathbf{s}^i, \gamma_i, \omega^k) = \begin{cases} \{\} \quad \text{if } \mathbf{s}^i = \{\} \\ \{\} \quad \text{if there exists a constraint } j \in \{1, \dots, m_2\} \text{ such that:} \\ \qquad \text{if } \gamma_i = x_q, \text{ for } q \in \{1,\dots,n_1\}, \text{ then } s^i_j - T(\omega^k)_{jq} x_q - U^i_j = 1 \\ \qquad \text{if } \gamma_i = y_q, \text{ for } q \in \{1,\dots,n_2\}, \text{ then } s^i_j - W(\omega^k)_{jq} y_q - U^i_j = 1 \\ (\mathbf{s}^i - \mathbf{t}(\omega^k)_q x_q)^+ \quad \text{if } \gamma_i = x_q, \text{ for } q \in \{1,\dots,n_1\} \\ (\mathbf{s}^i - \mathbf{w}(\omega^k)_q y_q)^+ \quad \text{if } \gamma_i = y_q, \text{ for } q \in \{1,\dots,n_2\} \end{cases} \tag{5.6}$$

The first case of (5.6) corresponds to the event in which $\mathbf{s}^i$ is already infeasible, while the second case occurs when there is no longer any way of finding a feasible solution for some constraint j, even if all remaining variables in $\boldsymbol{\gamma}$ are set to 1. The third case updates $\mathbf{s}^i$ after setting an x-variable, and the fourth case updates $\mathbf{s}^i$ after setting a y-variable.

Each transition associated with a variable y_q incurs a cost $f(\omega^k)_q y_q$, where $f(\omega^k)_q$ is the q^{th} component of the second-stage cost vector $f(\omega^k)$. Transitions associated with x-variables do not incur any cost since x-variables are decided in the first stage.

We illustrate our DP formulation using the following second-stage problem:

$$\text{SP}(\mathbf{x}, \omega^k): \quad \mathcal{Q}(\mathbf{x}, \omega^k) = \min \quad y_1 + y_2 + y_3 \tag{5.7a}$$

$$\text{s.t.} \quad y_1 + y_2 \geq 1 - x_1 \tag{5.7b}$$

$$y_1 + y_3 \geq 1 - x_2 \tag{5.7c}$$

$$y_2 + y_3 \geq 1 - x_3 \tag{5.7d}$$

$$\mathbf{y} \in \{0,1\}^3, \tag{5.7e}$$

where first-stage decisions belong to set $\mathcal{X} = \{0,1\}^3$. Let $\boldsymbol{\gamma} = [y_1, y_2, x_1, y_3, x_2, x_3]$. This ordering leverages the fact that y_1, y_2, and x_1 are the only variables having nonzero coefficients in constraint (5.7b), y_3 and x_2 have nonzero coefficients in constraint (5.7c), and x_3 has a nonzero coefficient in (5.7d). Figure 5.1 shows the proposed state transition graph. Dashed arcs represent assigning a value of 0 to the current variable and solid arcs represent assigning a value of 1 to that variable.

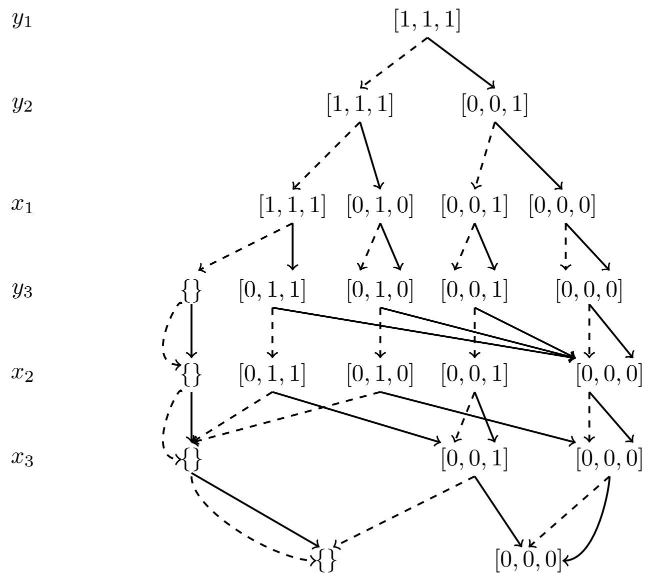


Figure 5.1: State transition graph for problem (5.7)

Although our proposed DP formulation is valid for problem SP($\hat{\mathbf{x}}, \omega^k$), it could in the worst case contain one state for each possible combination of values for variables $\mathbf{x}$ and $\mathbf{y}$, i.e., an order of $2^{n_1+n_2}$ states. However, we can attempt to exploit the structure of the problem to limit the size of the state space. (Section 5.2.2 further explores this idea by using a branch decomposition of an associated hypergraph to derive a variable ordering that ensures an upper bound on the size of the state space.)

For a given first-stage decision $\hat{\mathbf{x}} \in \mathcal{X}$, the transition graph can be naturally simplified by fixing the arcs corresponding to the choice of x-variables, albeit at the expense of no longer parameterizing the state transition graph as a function of $\mathbf{x}$. Figure 5.2 shows the reduced state transition graph for first-stage decisions $[\hat{x}_1, \hat{x}_2, \hat{x}_3] = [0, 0, 0]$ and $[\hat{x}_1, \hat{x}_2, \hat{x}_3] = [1, 0, 0]$. Each path

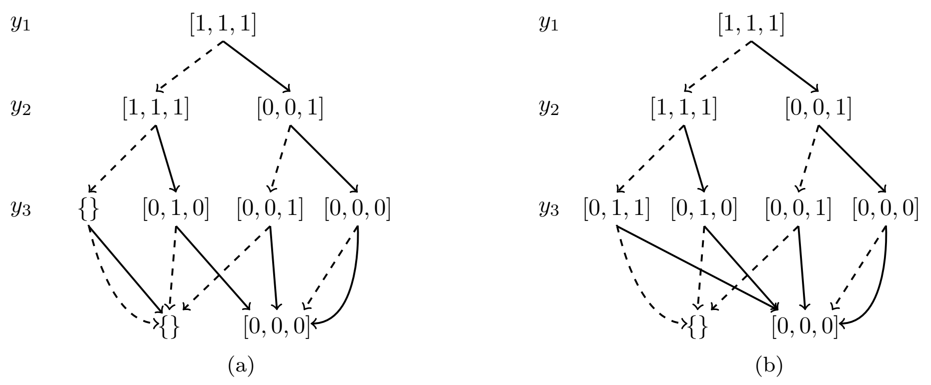


Figure 5.2: (a) Simplified state transition graph for $[\hat{x}_1, \hat{x}_2, \hat{x}_3] = [0, 0, 0]$ (b) Simplified state transition graph for $[\hat{x}_1, \hat{x}_2, \hat{x}_3] = [1, 0, 0]$

from the root node to the terminal node in Figure 5.2 represents a feasible solution to problem (5.7). Note that for $[\hat{x}_1, \hat{x}_2, \hat{x}_3] = [0, 0, 0]$ the transition from state $[1, 1, 1]$ in the second stage, when assigning $y_2 = 0$, leads to the infeasible state. For $[\hat{x}_1, \hat{x}_2, \hat{x}_3] = [1, 0, 0]$ the same transition results in state $[0, 1, 1]$. As a result, setting $x_1 = 1$ allows an additional second-stage feasible solution $[y_1, y_2, y_3] = [0, 0, 1]$.

5.2.1.2 Reduced BDD Representation Using Capacitated Arcs

Our next goal is to create a simplified version of the state transition graph, in which the availabilities of arcs in the transition graph are still given as a function of the x-variables. This representation is smaller (as in Figure 5.2), but allows us to parameterize our optimal DP solutions as a function of $\mathbf{x}$, which becomes critical in using decomposition techniques for scenario-based stochastic programs.

Accordingly, we transform problem SP($\hat{\mathbf{x}}, \omega^k$) into a shortest-path problem using a BDD based on the proposed DP formulation. Some arcs in this shortest path problem are available only when certain x-variables equal one, and so we refer to those arcs as *capacitated.* For each scenario $\omega^k \in \Omega$ we propose the generation of a single BDD, which is a layered directed acyclic graph $\mathcal{G}'_k = (\mathcal{N}^k, \mathcal{A}^k)$ whose nodes and arcs correspond to states and transitions in the state transition graph, respectively. The set of nodes $\mathcal{N}^k$ is partitioned into layers $\mathcal{L}^k_1, \ldots, \mathcal{L}^k_{n_1+n_2}$ corresponding to variables in ordering $\boldsymbol{\gamma}$, plus an additional terminal layer $\mathcal{L}^k_{n_1+n_2+1}$ that contains only the terminal node. We refer to arcs emanating from nodes in $\mathcal{L}_i^k$ as belonging to layer i. We explain the process of generating this graph in two phases.

The first phase creates a large shortest-path problem having capacitated arcs. We first remove nodes corresponding to the infeasible state. Next, for each arc $a \in \mathcal{A}^k$ we define a set of indices $\mathcal{U}_a^k \subseteq \{1, \dots, n_1\}$ such that the flow on arc a cannot be positive unless $\hat{x}_q = 1$, for all $q \in \mathcal{U}_a^k$. Hence, for a given first-stage solution $\hat{\mathbf{x}}$, $\hat{x}_q$ imposes an upper bound on flow on arc a, $\forall q \in \mathcal{U}_a^k$. The arc upper bounds imposed by $\mathcal{U}_a^k$ ensure that each path from $r(\omega^k)$ to t represents a feasible solution to SP$(\hat{\mathbf{x}}, \omega^k)$. In the first phase of our BDD construction, we set $\mathcal{U}_a^k = \{q\}$ if arc a corresponds to setting $x_q = 1$, and $\mathcal{U}_a^k = \emptyset$ otherwise (either because a corresponds to setting $x_q = 0$, or because a corresponds to a y-variable). We define arc-cost vector $\mathbf{g}^k$ as follows:

$$g_a^k = \begin{cases} f(\omega^k)_i & \text{if arc } a \text{ corresponds to setting } y_i = 1 \\ 0 \text{ otherwise.} \end{cases} \tag{5.8}$$

Figure 5.3 shows our proposed BDD for the example second-stage problem (5.7). Nonempty sets $\mathcal{U}_a^k$ are displayed alongside the arcs. In the second phase of our process, we reduce the size of our

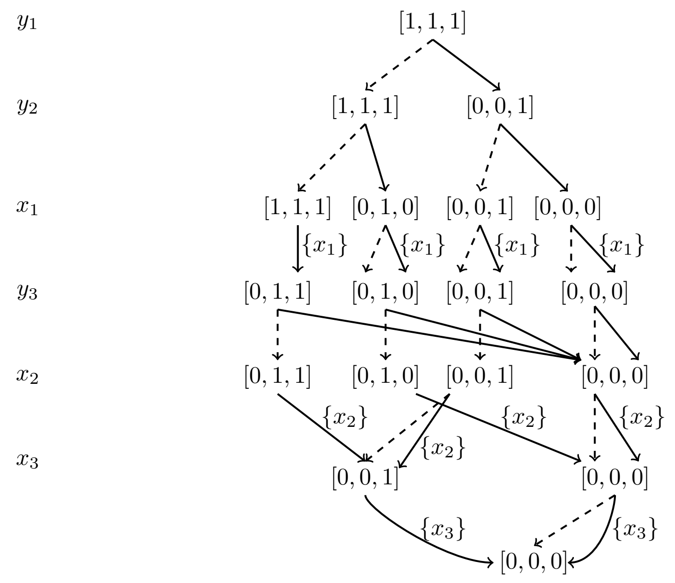


Figure 5.3: Proposed BDD for problem (5.7) after phase one

proposed BDD by adding *long arcs* [Bryant, 1986] that skip one or more layers corresponding to x-variables. Consider layers i and j such that $j \geq i + 2$, layer i corresponds to a y-variable, and all layers in $[i+1, \ldots, j-1]$ correspond to x-variables $x_{\psi(i+1)}, \ldots, x_{\psi(j-1)}$, respectively. For each node $u \in \mathcal{L}_i^k$, we create a long arc to node $v \in \mathcal{L}_j^k$ if there exists a path from u to v in the BDD from the first phase. For each such long arc a, we include $\psi(q)$ in $\mathcal{U}_a^k$ if and only if the path from u to v corresponding to a uses an arc for which $x_{\psi(q)} = 1$. Observe that parallel long arcs can now exist between node pairs in this network. The number of layers is now reduced to $n_2 + 1$.

To simplify this network, we eliminate unnecessary long arcs. Definition 1 aids us toward this goal by introducing the concept of dominance for long arcs.

Definition 1. *Consider long arcs a' and a'' having the same origin and destination nodes. If a' and a'' correspond to setting a y-variable to the same value, and $\mathcal{U}_{a'}^k \subset \mathcal{U}_{a''}^k$, then a' dominates a''.*

To finish the second phase of our graph simplification, we reduce the size of our BDD by removing dominated long arcs that do not encode additional second-stage solutions. Figure 5.4 presents all long arcs emanating from state $[0,1,1]$ in the fourth layer of our BDD for problem (5.7). Long arcs

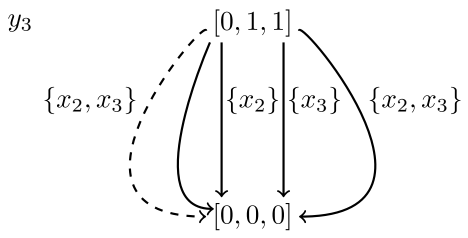


Figure 5.4: Long arcs emanating from state $[0,1,1]$ in our proposed BDD

from left to right correspond to assignments $[y_3, x_2, x_3] \in \{[0,1,1], [1,0,0], [1,1,0], [1,0,1], [1,1,1]\}$, respectively. Note that the last three arcs are dominated by the second arc and can be removed from the BDD. Figure 5.5 presents our reduced BDD for problem (5.7) after removing all dominated long arcs. The long arc from state $[1,1,1]$ to state $[0,1,1]$ corresponds to assigning $y_1 = 0$. This assignment is feasible only if $x_1 = 1$; thus, its upper bound set includes x_1. There are two long arcs from state $[0,1,1]$ to state $[0,0,0]$. The dashed arc corresponds to assigning $y_3 = 0$, which is a feasible assignment only if $x_2 = 1$ and $x_3 = 1$; thus, its upper bound set includes x_2 and x_3. The solid arc corresponds to assigning $y_3 = 1$, which is a feasible assignment no matter the choice of x_2 and x_3; thus, its upper bound set is empty. Note that for a given first-stage decision vector $\hat{\mathbf{x}} \in \mathcal{X} = \{0,1\}^3$, upper bounds imposed by sets $\mathcal{U}_a^k$ admit only paths that represent feasible solutions to $\mathrm{SP}(\hat{\mathbf{x}}, \omega^k)$. For example, if $[\hat{x}_1, \hat{x}_2, \hat{x}_3] = [0,0,0]$, then feasible paths from the root node

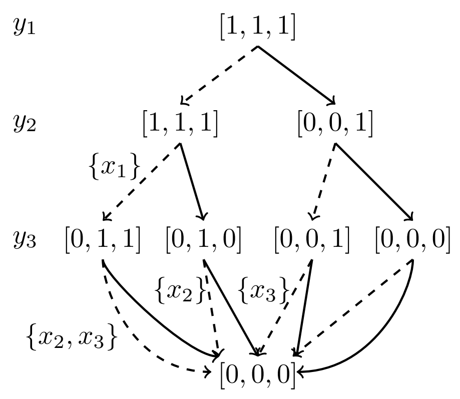


Figure 5.5: Reduced BDD for problem (5.7) after phase two

to the terminal node encode second-stage solutions $[y_1, y_2, y_3] \in \{[1,1,1],[1,1,0],[1,0,1],[0,1,1]\}$. However, for any vector $\hat{\mathbf{x}}$ such that $\hat{x}_1 = 1$ there is an additional feasible path corresponding to second-stage solution $[y_1, y_2, y_3] = [0,0,1]$.

Remark 10. *The size of network $\mathcal{G}'_k$ can be further reduced by removing arcs whose associated upper bound index set $\mathcal{U}^k_a$ corresponds to an infeasible first-stage solution $\mathbf{x} \notin \mathcal{X}$.* ■

The final BDD we construct exhibits the following two structural properties. First, there is a feasible path from any node i to the terminal node t. This is true because every node different from t has at least one uncapacitated outgoing arc that corresponds to setting a y-variable to one. Second, a first-stage variable x_q is included in sets $\mathcal{U}^k_{a_1}, \mathcal{U}^k_{a_2}, \ldots, \mathcal{U}^k_{a_m}$ only if long arcs $a_1, a_2, \ldots, a_m$ belong to the same layer. As a result, $\mathcal{U}^k_a \cap \mathcal{U}^k_{a'} = \emptyset$ for any two long arcs $a, a' \in \mathcal{A}^k$ belonging to different layers.

5.2.1.3 Recourse Problem Reformulation

For a given scenario $\omega^k \in \Omega$ and first stage decision $\hat{\mathbf{x}} \in \mathcal{X}$, we reformulate $\text{SP}(\hat{\mathbf{x}}, \omega^k)$ as the following shortest-path problem defined over a BDD given by $\mathcal{G}'_k = (\mathcal{N}^k, \mathcal{A}^k)$, where $\mathbf{v}$ is a vector of arc-flow variables.

$$\text{LSP}(\hat{\mathbf{x}}, \omega^k): \quad \min \sum_{(i,j)\in\mathcal{A}^k} g^k_{ij} v_{ij} \tag{5.9a}$$

$$\text{s.t.} \sum_{\{j|(i,j)\in\mathcal{A}^k\}} v_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}^k\}} v_{ji} = \begin{cases} 1, & \text{for } i = r(\omega^k) \\ 0, & \text{for } i \in \mathcal{N}^k\backslash\{r(\omega^k), t\} \\ -1, & \text{for } i = t \end{cases} \tag{5.9b}$$

$$v_{ij} \leq \hat{x}_q \qquad \forall (i,j) \in \mathcal{A}^k,\ q \in \mathcal{U}_{ij}^k \tag{5.9c}$$

$$v_{ij} \geq 0 \qquad \forall (i,j) \in \mathcal{A}^k. \tag{5.9d}$$

The objective function (5.9a) minimizes the total cost of traversing the arcs, which corresponds to the cost of assigning values to second-stage variables $\mathbf{y}$. Constraints (5.9b) ensure flow conservation. Constraints (5.9c) impose the arc upper bounds defined by $\mathcal{U}_{ij}^k$, and (5.9d) require the v-variables to be nonnegative.

Lemma 7. *For a given scenario $\omega^k \in \Omega$ and first stage decision $\hat{\mathbf{x}} \in \mathcal{X}$, LSP$(\hat{\mathbf{x}}, \omega^k)$ is equivalent to SP$(\hat{\mathbf{x}}, \omega^k)$.*

Proof Every feasible solution to SP$(\hat{\mathbf{x}}, \omega^k)$ corresponds to exactly one feasible path from $r(\omega^k)$ to t in graph $\mathcal{G}'_k$ having the same objective, and vice versa. ■

5.2.2 Variable Ordering Based on a Branch Decomposition Heuristic

The variable ordering $\boldsymbol{\gamma}$ influences the number of nodes and arcs in the corresponding BDD. We explore this relationship in this subsection by first determining where to place the x-variables in $\boldsymbol{\gamma}$ given a fixed ordering of the y-variables. Then, we determine how to order the y-variables in $\boldsymbol{\gamma}$ (with positions of the x-variables now implied) to limit the size of the BDD.

5.2.2.1 Placement of x-Variables Within γ

Our first goal is to place the x-variables in $\boldsymbol{\gamma}$ relative to the y-variables in a way so that the resulting BDD has at most two nondominated long arcs emanating from any node. Let $\boldsymbol{\gamma}_y$ be an ordering of only the y-variables and define $p_y(q)$ as the position of y_q in $\boldsymbol{\gamma}_y$, $\forall q = 1, \dots, n_2$. Consider a first-stage variable x_q having a nonzero coefficient in constraint $j \in \{1, \dots, m_2\}$. Because of the special structure of matrix $T(\omega^k)$, there is at most one constraint j for which x_q has a nonzero coefficient. We place x_q in between y-variables in positions i and $i+1$ of $\boldsymbol{\gamma}_y$ for the position i

satisfying

$$i = \min\{\hat{\imath} \mid W(\omega^k)_{jh} = 0,\ \forall h \in \{1, \dots, n_2\} : p_y(h) > \hat{\imath}\}. \tag{5.10}$$

That is, the y-variable in position i is the last y-variable in $\boldsymbol{\gamma}_y$ that has a nonzero coefficient in constraint j. Variable x_q is used to repair feasibility for those states in which constraint j is not satisfied after assigning a value to the y-variable in position i. As a result, a node in the BDD has at most two long nondominated arcs, which correspond to assigning a value of 0 and a value of 1 to the y-variable associated with the node layer defined by (5.10).

5.2.2.2 Impact of y-ordering on BDD Size

Now, we consider the problem of ordering the y-variables in $\boldsymbol{\gamma}$. Unfortunately, the problem of finding an ordering of the y-variables that minimizes the size of the resulting BDD is generally $\mathcal{NP}$-hard [Bryant, 1986]. We investigate a key feature that determines the *breadth* of our BDD, which we define as the maximum number of nodes that exist at any layer of the BDD.

Clearly, the breadth of a BDD is never more than 2^{m_2}, and the number of nodes in layer j of the BDD cannot exceed 2^{j-1}, for $j = 1, \dots, n_2$. To obtain an alternative bound, define μ_j as the set of all state elements that take a value of zero at some node in layer j of the BDD and a value of one at another node in layer j. Layer j thus contains no more than $2^{|\mu_j|}$ nodes. For instance, in Figure 5.5 we have that $|\mu_1| = 0$, $|\mu_2| = 2$ (due to the first and second state elements), and $|\mu_3| = 2$ (due to the second and third state elements). The number of nodes at layer j of the Figure 5.5 BDD is given by $2^{\min\{j-1,|\mu_j|\}}$, for $j = 1, 2, 3$. The breadth of a BDD is thus bounded by the largest value of $2^{\min\{j-1,|\mu_j|\}}$ over all j. However, this bound is potentially weak, and so we examine a strategy for variable orderings that yield tighter bounds on the breadth of the BDD.

In order to aid us in determining an ordering $\boldsymbol{\gamma}_y$ that yields small BDDs, we turn to the notion of branch decompositions over hypergraphs. For each recourse problem, we generate an associated hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, omitting the scenario index for notational ease, where $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ is a set of hyperedges. Let $\mathcal{V} = \{1, \dots, n_2\}$, where each vertex corresponds to a y-variable. Define $\mathcal{E} = \{\mathcal{E}_1, \dots, \mathcal{E}_{m_2}\}$, where each hyperedge corresponds to a second-stage constraint and $\mathcal{E}_j = \{i \in \{1, \dots, n_2\} \mid W(\omega^k)_{ji} > 0\}$ for all $j = 1, \dots, m_2$. Vertices contained in subset $\mathcal{E}_j$ correspond to all y-variables that have a nonzero coefficient in constraint j and are said to be *incident* to hyperedge $\mathcal{E}_j$. We illustrate our proposed hypergraph using the following second-stage problem:

$$\begin{aligned}
\text{SP}(\mathbf{x},\omega^k): \quad \mathcal{Q}(\mathbf{x},\omega^k) = \min \quad & y_1 + y_2 + y_3 + y_4 + y_5 && \text{(5.11a)}\\
\text{s.t.} \quad & y_1 + y_2 + y_3 \geq 1 - x_1 && \text{(5.11b)}\\
& y_3 + y_4 + y_5 \geq 1 - x_2 && \text{(5.11c)}\\
& y_2 + y_4 \geq 1 - x_3 && \text{(5.11d)}\\
& y_1 + y_3 \geq 1 - x_4 && \text{(5.11e)}\\
& y_3 + y_5 \geq 1 - x_5 && \text{(5.11f)}\\
& y_4 + y_5 \geq 1 - x_6 && \text{(5.11g)}\\
& \mathbf{y} \in \{0,1\}^5. && \text{(5.11h)}
\end{aligned}$$

The hypergraph associated with problem (5.11) is given by set of vertices $\mathcal{V} = \{1,2,3,4,5\}$ and set of hyperedges $\mathcal{E} = \{\{1,2,3\},\{3,4,5\},\{2,4\},\{1,3\},\{3,5\},\{4,5\}\}$. Observe that every second-stage variable appears in at least two constraints in this example. If, for instance, the last constraint were revised to $y_4 + y_5 + y_6 \geq 1 - x_6$, then y_6 is said to be an *isolated variable*. More formally, y_i is an isolated variable in scenario ω^k if $\sum_{j=1}^{m_2} W(\omega^k)_{ji} = 1$. For now, we assume that no isolated variables exist, but discuss how we can handle isolated variables in Remark 11.

Branch decompositions were first introduced by Robertson and Seymour [1991]. Let $\mathcal{T}$ be a tree with $|\mathcal{E}|$ leaves and edge set $\mathcal{E}_{\mathcal{T}}$. Additionally, every non-leaf node in $\mathcal{T}$ has degree three. Let v be a bijection from the hyperedges of $\mathcal{G}$ to the leaves of $\mathcal{T}$. The pair $(\mathcal{T}, v)$ is called a branch decomposition of $\mathcal{G}$. Removing edge e from $\mathcal{T}$ partitions the hyperedges of $\mathcal{G}$ into two subsets $\mathcal{A}_e$ and $\mathcal{B}_e$. Let $\mathcal{V}(\mathcal{A}_e)$ and $\mathcal{V}(\mathcal{B}_e)$ denote the set of vertices that are incident to hyperedges in $\mathcal{A}_e$ and $\mathcal{B}_e$, respectively. The *load* of edge e, denoted by $\Gamma(e)$, is given by $\mathcal{V}(\mathcal{A}_e) \cap \mathcal{V}(\mathcal{B}_e)$. The *width* of a branch decomposition $(\mathcal{T}, v)$ is the maximum load cardinality among all edges in $\mathcal{T}$. The *branchwidth* of $\mathcal{G}$ is the minimum width among all branch decompositions of $\mathcal{G}$. A branch decomposition whose width is equal to the branchwidth is called an *optimal branch decomposition*. Figure 5.6 presents a branch decomposition tree with the corresponding edge loads for the hypergraph associated with problem (5.11).

The problem of finding an optimal branch decomposition of a hypergraph is $\mathcal{NP}$-hard

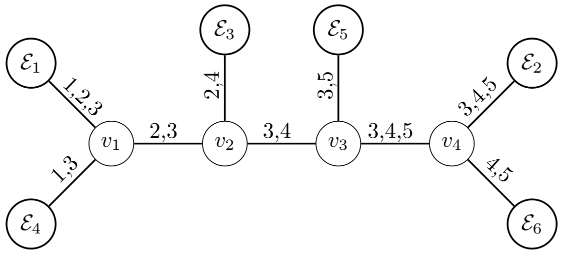


Figure 5.6: Branch decomposition tree for problem (5.11)

[Seymour and Thomas, 1994]. Thus, we use the heuristic tree-building procedure of Hicks [2002]. However, we modify the procedure to generate a branch decomposition of $\mathcal{G}$ in which the resulting tree has a special *linear* structure. We define a linear tree as a tree where every non-leaf node is adjacent to at least one leaf node (e.g., as Figure 5.6). Note that a linear branch decomposition tree has two *endpoints*, which are non-leaf nodes that are adjacent to exactly two leaf nodes.

Our next step is to show how y-variables can be ordered in $\boldsymbol{\gamma}$, given a linear branch decomposition tree. Let $\boldsymbol{\gamma}_y$ be our ordering of the y-variables.

We start by labeling the non-leaf nodes of the BDD as $v_1, \ldots, v_{|\mathcal{E}|-2}$, where v_1 and $v_{|\mathcal{E}|-2}$ are endpoints. We say that edge e_i connects v_i to v_{i+1}. The leaf nodes in $\mathcal{T}$ correspond to hyperedges in $\mathcal{E}$, and hence with a slight abuse of notation we say that hyperedges connect to the non-leaf nodes of $\mathcal{T}$. Index the hyperedges so that $\mathcal{E}_{\ell_1}$ and $\mathcal{E}_{\ell_2}$ are connected to v_1, $\mathcal{E}_{\ell_{i+1}}$ is connected to v_i for $i = 2, \ldots, |\mathcal{E}| - 3$, and both $\mathcal{E}_{\ell_{|\mathcal{E}|-1}}$ and $\mathcal{E}_{\ell_{|\mathcal{E}|}}$ are connected to $v_{|\mathcal{E}|-2}$. We order the variables in $\boldsymbol{\gamma}_y$ according to the following groups, where multiple variables in a single group can be ordered arbitrarily.

Group 0 consists of all variables in $(\mathcal{E}_{\ell_1} \cup \mathcal{E}_{\ell_2}) \setminus \Gamma(e_1)$. Then, Group i consists of all variables in $\Gamma(e_i)$ that have not appeared in Groups $0, \ldots, i-1$, for $i = 1, \ldots, |\mathcal{E}| - 3$. Finally, Group $|\mathcal{E}| - 2$ consists of variables in $\left(\mathcal{E}_{\ell_{|\mathcal{E}|-1}} \cup \mathcal{E}_{\ell_{|\mathcal{E}|}}\right) \setminus \Gamma(e_{|\mathcal{E}|-3})$. Applying this procedure on the tree from Figure 5.6 implies that variable y_1 is ordered first, followed by y_2 and y_3 in either order, then y_4 and y_5 in that order.

Let $w = \max_{i \in \{1,\ldots,|\mathcal{E}|-3\}} \{\Gamma(e_i)\}$ be the maximum load among non-leaf edges of the branch decomposition we obtain using the modified approach of Hicks [2002]. The following proposition bounds the breadth of our BDD as an exponential function of w.

Proposition 14. *Consider a BDD generated according to the foregoing variable ordering scheme.*

The breadth of the BDD is no more than $2^{\max\{w,1\}}$.

Proof We first introduce the following concepts. Consider a constraint $j \in \{1, \ldots, m_2\}$ and recall that $p_y(q)$ denotes the position of a variable y_q in $\boldsymbol{\gamma}_y$. We say that constraint j is *irrelevant* at layer i if $W(\omega^k)_{jq} = 0,\ \forall q \in \{1, \ldots, n_2\} : p_y(q) \geq i$. That is, a constraint that is irrelevant at layer i must have been satisfied by variables that correspond to layers $1, \ldots, i-1$. Therefore, the state element corresponding to an irrelevant constraint j takes a value of zero at every BDD node in every layer $j \geq i$. Also, we say that a variable y_q is *forgotten* if it is incident only to hyperplanes that correspond to irrelevant constraints.

We prove this result by induction. Let $\mathcal{L}_{g_1^i}, \ldots, \mathcal{L}_{g_{r_i}^i}$ denote the layers corresponding to variables in Group i, and let $|\mathcal{L}_{g_h^i}|$ denote the number of nodes in layer $\mathcal{L}_{g_h^i}$ of the BDD. We begin by showing that if $\Gamma(e_1) \subset \Gamma(e_2)$, then $\mathcal{L}_{g_1^2} \leq 2^{\Gamma(e_1)}$, and if $\Gamma(e_1) \not\subset \Gamma(e_2)$, then $\mathcal{L}_{g_1^2} \leq 2^{|\Gamma(e_1) \cap \Gamma(e_2)|+1}$. Each variable in Group 0, if any, appears in *both* constraints ℓ_1 and ℓ_2 but do not appear in any other constraints. As a result, there are at most two nodes in any layer $\mathcal{L}_{g_1^0}, \ldots, \mathcal{L}_{g_{r_0}^0}$. Note that after assigning a value to the last variable in Group 0, we generate layer $\mathcal{L}_{g_1^1}$, which also contains at most two nodes if Group 0 is nonempty. Let δ be a binary parameter that equals one if Group 0 is nonempty, and zero otherwise.

After assigning a value to the first variable in Group 1 we have that $|\mathcal{L}_{g_2^1}| \leq 2^{1+\delta}$, given by the possible combinations of the one (if $\delta = 0$) or two (if $\delta = 1$) nodes in $\mathcal{L}_{g_1^1}$ and the two values that the first variable in Group 1 can take. By the same argument we have that $|\mathcal{L}_{g_3^1}| \leq 2^{2+\delta}$, ..., $|\mathcal{L}_{g_{r_1}^1}| \leq 2^{|\Gamma(e_1)|+\delta-1}$. After assigning a value to the last variable in Group 1, we observe that no layer indexed g_1^2 or larger corresponds to a y-variable q that is incident to ℓ_1 or ℓ_2, and so constraints ℓ_1 and ℓ_2 become irrelevant. The state elements corresponding to ℓ_1 and ℓ_2 take a value of zero at every node in layer $\mathcal{L}_{g_1^2}$. At this point we forget variables in Group 0.

We then generate layer $\mathcal{L}_{g_1^2}$, for which we consider two cases.

Case 1. $\Gamma(e_1) \subset \Gamma(e_2)$. In this case, $|\mathcal{L}_{g_1^2}| \leq 2^{|\Gamma(e_1)|}$, corresponding to the combination of values that variables in $\Gamma(e_1)$ can take, and ignoring combinations of forgotten variables in Group 0.

Case 2. $\Gamma(e_1) \not\subset \Gamma(e_2)$. Thus, $\Gamma(e_1) \setminus \Gamma(e_2) \neq \emptyset$. Variables in $\Gamma(e_1) \setminus \Gamma(e_2)$ appear only in constraint ℓ_3 and in either ℓ_1 or ℓ_2 (or both). Also, the only non-irrelevant constraint incident to variables in $\Gamma(e_1) \setminus \Gamma(e_2)$ is ℓ_3. As a result, $|\mathcal{L}_{g_1^2}|$ is no greater than the number of combinations between the state element corresponding to ℓ_3 and the values that variables in $\Gamma(e_1) \cap \Gamma(e_2)$ can take. Thus,

$|\mathcal{L}_{g_1^2}| \leq 2^{|\Gamma(e_1)\cap\Gamma(e_2)|+1}$, which is less than or equal to $2^{|\Gamma(e_1)|}$ since $|\Gamma(e_1) \cap \Gamma(e_2)| < |\Gamma(e_1)|$. Note that at this point, variables in $\Gamma(e_1) \setminus \Gamma(e_2)$ are not forgotten, but are completely represented by the state element corresponding to ℓ_3.

We now show that if the above results for Cases 1 and 2 are true for Group $i \in \{1, \ldots, |\mathcal{E}|-4\}$, then they are also true for Group $i+1$. Assume for Group i that if $\Gamma(e_i) \subset \Gamma(e_{i+1})$, then $|\mathcal{L}_{g_1^{i+1}}| \leq 2^{|\Gamma(e_i)|}$, and if $\Gamma(e_i) \not\subset \Gamma(e_{i+1})$, then $|\mathcal{L}_{g_1^{i+1}}| \leq 2^{|\Gamma(e_i)\cap\Gamma(e_{i+1})|+1}$. Now, consider Group $i+1$. If $\Gamma(e_i) \subset \Gamma(e_{i+1})$, then $|\Gamma(e_i)| < |\Gamma(e_{i+1})|$ and after assigning values to all but the last variable in Group $i+1$ we have that $|\mathcal{L}_{g_2^{i+1}}| \leq 2^{|\Gamma(e_i)|+1}$, $\ldots$, $|\mathcal{L}_{g_{r_{i+1}}^{i+1}}| \leq 2^{|\Gamma(e_{i+1})|-1}$. Otherwise, we get that $|\mathcal{L}_{g_2^{i+1}}| \leq 2^{|\Gamma(e_i)\cap\Gamma(e_{i+1})|+2}$, $\ldots$, $|\mathcal{L}_{g_{r_{i+1}}^{i+1}}| \leq 2^{|\Gamma(e_{i+1})|}$. After assigning a value to the last variable in Group $i+1$, constraint ℓ_{i+2} becomes irrelevant, and all variables in Groups $0, \ldots, i$ that do not appear in Group $i+1$ are forgotten. We generate layer $\mathcal{L}_{g_1^{i+2}}$, for which we consider two cases.

Case 1. $\Gamma(e_{i+1}) \subset \Gamma(e_{i+2})$. Because all variables in Groups $0, \ldots, i$ have been forgotten, we have that $|\mathcal{L}_{g_1^{i+2}}| \leq 2^{|\Gamma(e_{i+1})|}$.

Case 2. $\Gamma(e_{i+1}) \not\subset \Gamma(e_{i+2})$. Variables in $\Gamma(e_{i+1}) \setminus \Gamma(e_{i+2})$ appear in some constraints ℓ_h, $h \leq i+3$. Since constraints $\ell_1, \ldots, \ell_{i+2}$ are irrelevant at layer $\mathcal{L}_{g_1^{i+2}}$, we have that $|\mathcal{L}_{g_1^{i+2}}|$ is no greater than the number of combinations between the state element corresponding to ℓ_{i+3} and the values that variables in $\Gamma(e_{i+1}) \cap \Gamma(e_{i+2})$ can take. As a result, $|\mathcal{L}_{g_1^{i+2}}| \leq 2^{|\Gamma(e_{i+1})\cap\Gamma(e_{i+2})|+1}$, which is less than or equal to $2^{|\Gamma(e_{i+1})|}$.

We conclude that layers corresponding to variables in Group i have at most $2^{|\Gamma(e_i)|}$ nodes for $i = 1, \ldots, |\mathcal{E}|-3$. Moreover, we can strengthen the bound of $|\mathcal{L}_{g_1^{|\mathcal{E}|-2}}| \leq 2^{|\Gamma(e_{|\mathcal{E}|-3})|}$ if $|\Gamma(e_{|\mathcal{E}|-3})| > 2$. Since at this point $\ell_{|\mathcal{E}|-1}$ and $\ell_{|\mathcal{E}|}$ are the only non-irrelevant constraints, we get that $|\mathcal{L}_{g_1^{|\mathcal{E}|-2}}| \leq 2^{\min\{|\Gamma(e_{|\mathcal{E}|-3})|,2\}}$. Now, because variables in Group $|\mathcal{E}|-2$ appear in both constraints $\ell_{|\mathcal{E}|-1}$ and $\ell_{|\mathcal{E}|}$ but do not appear in any other constraints, we get that there are at most $\max\left\{2^{\min\{|\Gamma(e_{|\mathcal{E}|-3})|,2\}}, 2\right\}$ nodes in any layer corresponding to variables in Group $|\mathcal{E}|-2$. This completes the proof. ■

Remark 11. *We now explain how to incorporate isolated variables into our proposed BDD. Assume without loss of generality that for each constraint $j \in \{1, \ldots, m_2\}$ there is at most one isolated variable q for which $W(\omega^k)_{jq} = 1$. (Otherwise, we would eliminate all but the least-cost isolated variable in constraint j.) Let $\mathcal{S} = \{q \in \{1, \ldots, n_2\} \mid \sum_{j=1}^{m_2} W(\omega^k)_{jq} = 1\}$ be the set of all isolated variables. Because $\sum_{q\in\mathcal{S}} W(\omega^k)_{jq} \leq 1$ for all $j = 1, \ldots, m_2$, isolated variables can be treated similarly to the x-variables: We represent isolated variables with long arcs that skip one or more layers corresponding* *to isolated variables in the BDD. For each such long arc a we set $\mathcal{U}_a^k = \emptyset$ and cost g_a corresponding to the sum of costs of the isolated variables that are set to 1 by traversing the arc. We place isolated variables in $\boldsymbol{\gamma}$ in a similar way as we do for x-variables. Consider an isolated variable y_q having a nonzero coefficient in constraint $j \in \{1, \ldots, m_2\}$. We place y_q in between non-isolated y-variables in positions i and $i+1$ for the position i satisfying*

$$i = \min\{\hat{\imath} \mid W(\omega^k)_{jh} = 0,\ \forall h \in \{1, \ldots, n_2\} \setminus \mathcal{S} : p_y(h) > \hat{\imath}\}, \tag{5.12}$$

where the y-variable in position i is the last non-isolated y-variable that has a nonzero coefficient in constraint j. ■

5.3 A Benders' Decomposition Algorithm

We now present our proposed Benders' decomposition algorithm based on problem reformulation (5.9). Section 5.3.1 describes the main components of the approach and presents our algorithm. Section 5.3.2 presents strategies to strengthen the optimality cuts used in our Benders' decomposition algorithm.

5.3.1 Formulating a Benders' Decomposition Approach

For a given scenario $\omega^k \in \Omega$ and first stage decision $\hat{\mathbf{x}} \in \mathcal{X}$, consider the dual problem associated with LSP$(\hat{\mathbf{x}}, \omega^k)$. Let $\boldsymbol{\pi}$ denote the dual variables associated with flow-balance constraints (5.9b) and let $-\boldsymbol{\alpha}$ be the dual variables associated with upper bound constraints (5.9c).

$$\text{DSP}(\hat{\mathbf{x}}, \omega^k): \quad \max \quad \pi_{r(\omega^k)} - \sum_{(i,j)\in\mathcal{A}^k} \sum_{q\in\mathcal{U}_{ij}^k} \hat{x}_q \alpha_{ijq} \tag{5.13a}$$

$$\text{s.t.} \quad \pi_i - \pi_j - \sum_{q\in\mathcal{U}_{ij}^k} \alpha_{ijq} \le g_{ij}^k \qquad \forall (i,j) \in \mathcal{A}^k \tag{5.13b}$$

$$\pi_i \ge 0 \qquad \forall i \in \mathcal{N}^k \tag{5.13c}$$

$$\alpha_{ijq} \ge 0 \qquad \forall (i,j) \in \mathcal{A}^k,\ q \in \mathcal{U}_{ij}^k. \tag{5.13d}$$

We set $\pi_t = 0$ because the flow balance constraint corresponding to node t is linearly dependent on the remaining flow balance constraints.

Let $\{(\boldsymbol{\pi}^{hk}, \boldsymbol{\alpha}^{hk}),\ h = 1, \dots, H^k\}$ be the set of extreme points of the polyhedron defined by (5.13b)–(5.13d) corresponding to scenario k. We formulate our master problem as follows:

$$\begin{aligned}
\text{MP}: \quad \min \quad & \mathbf{c}^\intercal \mathbf{x} + \sum_{k=1}^{K} p^k z_k && (5.14\text{a}) \\
\text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} && (5.14\text{b}) \\
& z_k \geq \pi^{hk}_{r(\omega^k)} - \sum_{(i,j)\in\mathcal{A}^k} \sum_{q\in\mathcal{U}^k_{ij}} x_q \alpha^{hk}_{ijq} \quad \forall k = 1, \dots, K,\ h = 1, \dots, H^k && (5.14\text{c}) \\
& z_k \text{ unrestricted} \quad \forall k = 1, \dots, K && (5.14\text{d}) \\
& \mathbf{x} \in \{0, 1\}^{n_1}. && (5.14\text{e})
\end{aligned}$$

Problem (5.14) is a standard Benders' master problem, which is equivalent to the original problem (5.1), due to the feasibility and boundedness of (5.9), and the fact that z_k takes on the optimal objective function value of (5.9) in any optimal solution to (5.14). We now describe our proposed cutting-plane algorithm. Let $\mathcal{C}$ be a subset of optimality cuts (5.14c). For a given $\mathcal{C}$ we define a *relaxed master problem*, RMP($\mathcal{C}$), which is defined exactly as MP but includes only optimality cuts contained in $\mathcal{C}$. We then generate new optimality cuts iteratively as described below. Let L_k denote a finite lower bound on z_k, which we can obtain as $L_k = \sum_{i=1}^{n_2} \min\{0, f(\omega^k)_i\}$, noting that $\mathbf{y^k} \in \{0, 1\}^{n_2}$.

Step 0: Generate a BDD $\mathcal{G}'_k = (\mathcal{N}^k, \mathcal{A}^k)$ for every scenario $\omega^k \in \Omega$. Set $UB_0 = \infty$ and $i = 0$. Initialize incumbent solution $\bar{\mathbf{x}} = \bar{\mathbf{y}} = \emptyset$ and the set of optimality cuts $\mathcal{C}$ as the bounding constraints $z_k \geq L_k$.

Step 1: Set $i = i + 1$. Solve RMP($\mathcal{C}$). If the problem is infeasible, then terminate and conclude that the original problem is infeasible. Otherwise, let LB_i be the optimal objective function value obtained for this relaxed master problem, and record the optimal solution $(\hat{\mathbf{x}}, \hat{\mathbf{z}})$ found.

Step 2: Compute an optimal solution, $\hat{\mathbf{y}}^k$, to LSP($\hat{\mathbf{x}}, \omega^k$) for every scenario $\omega^k \in \Omega$. An upper bound on the objective function value is given by $UB = \mathbf{c}^\intercal \hat{\mathbf{x}} + \sum_{k=1}^{K} p^k f(\omega^k)^\intercal \hat{\mathbf{y}}^k$. If $UB < UB_{i-1}$, then set $UB_i = UB$ and update incumbent $(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = (\hat{\mathbf{x}}, \hat{\mathbf{y}})$. Otherwise, set $UB_i = UB_{i-1}$.

Step 3: If $LB_i = UB_i$, then terminate with an optimal solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$. Otherwise, compute an optimal dual solution, $(\hat{\boldsymbol{\pi}}^k, \hat{\boldsymbol{\alpha}}^k)$, to DSP($\hat{\mathbf{x}}, \omega^k$) for every scenario $\omega^k \in \Omega$. If $\hat{z}^k < \hat{\pi}^k_{r(\omega^k)} - \sum_{(i,j)\in\mathcal{A}^k} \sum_{q\in\mathcal{U}^k_{ij}} \hat{x}_q \hat{\alpha}^k_{ijq}$, then add the optimality cut derived from dual solution $(\hat{\boldsymbol{\pi}}^k, \hat{\boldsymbol{\alpha}}^k)$ to $\mathcal{C}$.

Return to Step 1.

The exactness and finiteness of our cutting plane algorithm follows from the fact that there is a finite number of extreme points for each dual subproblem.

Remark 12. *For a given scenario $\omega^k \in \Omega$ and first stage decision $\hat{\mathbf{x}} \in \mathcal{X}$, LSP$(\hat{\mathbf{x}}, \omega^k)$ is a shortest-path problem defined over a layered directed acyclic graph. As a result, we can obtain optimal primal and dual solutions to LSP$(\hat{\mathbf{x}}, \omega^k)$ in $\mathcal{O}(|\mathcal{A}^k|)$ steps by means of a label-correcting algorithm as done by Ahuja et al. [1993]. Because there are generally multiple optimal dual solutions to LSP$(\hat{\mathbf{x}}, \omega^k)$, and these dual solutions impact the form of the Benders' inequality that we ultimately generate, we briefly describe our process for recovering optimal solutions to* (5.13).

We first compute values for the π-variables by setting π_i equal to the shortest path length from node i to node t in the shortest-path network. This step is possible because there exists a path from every node to node t by construction of our shortest-path network. Next, we select α-variables to satisfy $\sum_{q \in \mathcal{U}_{ij}^k} \alpha_{ijq} \geq \pi_i - \pi_j - g_{ij}^k$ for all $(i,j) \in \mathcal{A}^k$. If $\pi_i - \pi_j - g_{ij}^k \leq 0$, then we set $\alpha_{ijq} = 0$ for all $q \in \mathcal{U}_{ij}^k$. Otherwise, we set $\sum_{q \in \mathcal{U}_{ij}^k : \hat{x}_q = 0} \alpha_{ijq} = \pi_i - \pi_j - g_{ij}^k$ and $\alpha_{ijq} = 0$ for all $q \in \mathcal{U}_{ij}^k : \hat{x}_q = 1$. Note that if $\pi_i - \pi_j - g_{ij}^k > 0$, then at least one first-stage variable $\hat{x}_q = 0$ for $q \in \mathcal{U}_{ij}^k$, or else there would be a feasible path that traverses arc (i,j) and has length shorter than π_i.

We exploit the fact that there may exist alternative optimal values for the α-variables in our first strategy presented in Section 5.3.2. ■

5.3.2 Strengthening the Cuts

We examine two strategies to strengthen the optimality cuts used in our Benders' decomposition algorithm. Henceforth, we will develop our strengthening strategies assuming a given scenario $\omega^k \in \Omega$, first-stage decision $\hat{\mathbf{x}} \in \mathcal{X}$, and optimal solution to DSP$(\hat{\mathbf{x}}, \omega^k)$ denoted by $(\boldsymbol{\pi}, \boldsymbol{\alpha})$, with the scenario index omitted.

Our first strategy exploits the structural property of our BDD by which $\mathcal{U}_a^k \cap \mathcal{U}_{a'}^k = \emptyset$ for any two long arcs $a, a' \in \mathcal{A}^k$ belonging to different layers. Note that a path from $r(\omega^k)$ to t traverses exactly one arc from each layer. As a result, for each $q \in \{1, \ldots, n_1\}$, a path from $r(\omega^k)$ to t traverses at most one arc (i,j) for which $q \in \mathcal{U}_{ij}^k$. Proposition 15 exploits this special structure to strengthen the optimality cuts.

Proposition 15. *For all $q = 1, \ldots, n_1$ define $\gamma_q = \max_{(i,j) \in \mathcal{A}^k} \{\alpha_{ijq} \mid q \in \mathcal{U}_{ij}^k\}$. The following is a* *valid inequality that is at least as strong as optimality cut* (5.14c).

$$z_k \geq \pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \gamma_q x_q. \tag{5.15}$$

Proof For a given $\mathbf{x} \in \mathcal{X}$, let arc-flow vector $\mathbf{v} \in \{0,1\}^{|\mathcal{A}^k|}$ be a feasible solution to LSP($\mathbf{x}, \omega^k$). Multiplying both sides of constraints (5.13b) in DSP($\hat{\mathbf{x}}, \omega^k$) by v_{ij}, $(i,j) \in \mathcal{A}^k$, yields:

$$v_{ij}\left(\pi_i - \pi_j - \sum_{q \in \mathcal{U}_{ij}^k} \alpha_{ijq}\right) \leq g_{ij}^k v_{ij} \quad \forall (i,j) \in \mathcal{A}^k. \tag{5.16}$$

Summing over all equations (5.16) yields the following,

$$\pi_{r(\omega^k)} - \sum_{(i,j) \in \mathcal{A}^k} \sum_{q \in \mathcal{U}_{ij}^k} v_{ij}\alpha_{ijq} \leq \sum_{(i,j) \in \mathcal{A}^k} g_{ij}^k v_{ij}, \tag{5.17}$$

where $\sum_{(i,j) \in \mathcal{A}^k} v_{ij}(\pi_i - \pi_j) = \pi_{r(\omega^k)}$ because $\mathbf{v}$ represents a path from $r(\omega^k)$ to t, and because $\pi_t = 0$ by assumption. Constraints (5.9c) imply that $v_{ij} = x_q v_{ij}$ for all $(i,j) \in \mathcal{A}^k$, $q \in \mathcal{U}_{ij}^k$. Replacing v_{ij} by $x_q v_{ij}$ in (5.17) yields

$$\pi_{r(\omega^k)} - \sum_{(i,j) \in \mathcal{A}^k} \sum_{q \in \mathcal{U}_{ij}^k} x_q v_{ij}\alpha_{ijq} \leq \sum_{(i,j) \in \mathcal{A}^k} g_{ij}^k v_{ij}, \tag{5.18}$$

which is equivalent to

$$\pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \sum_{(i,j) \in \mathcal{A}^k : q \in \mathcal{U}_{ij}^k} x_q v_{ij}\alpha_{ijq} \leq \sum_{(i,j) \in \mathcal{A}^k} g_{ij}^k v_{ij}. \tag{5.19}$$

For any $q \in \{1, \ldots, n_1\}$, the path described by $\mathbf{v}$ traverses at most one arc (i,j) for which $q \in \mathcal{U}_{ij}^k$. Thus, $\sum_{(i,j) \in \mathcal{A}^k : q \in \mathcal{U}_{ij}^k} v_{ij} \leq 1$ for all $q = 1, \ldots, n_1$. As a result,

$$\sum_{q=1}^{n_1} \sum_{(i,j) \in \mathcal{A}^k : q \in \mathcal{U}_{ij}^k} x_q v_{ij}\alpha_{ijq} \leq \sum_{q=1}^{n_1} \gamma_q x_q, \tag{5.20}$$

and we get that

$$\pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \gamma_q x_q \leq \sum_{(i,j)\in\mathcal{A}^k} g_{ij}^k v_{ij}. \tag{5.21}$$

Note that (5.21) holds for any $\mathbf{v}$ feasible to LSP$(\mathbf{x}, \omega^k)$. Let $\mathbf{v}^*$ be an optimal solution to LSP$(\mathbf{x}, \omega^k)$. Lemma 7 implies that $\mathcal{Q}(\mathbf{x}, \omega^k) = \sum_{(i,j)\in\mathcal{A}^k} g_{ij}^k v_{ij}^*$, and so

$$\pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \gamma_q x_q \leq \mathcal{Q}(\mathbf{x}, \omega^k). \tag{5.22}$$

Because (5.22) is valid for any $\mathbf{x} \in \mathcal{X}$, we conclude that (5.15) is also valid. To prove that (5.15) is at least as strong as (5.14c), note that

$$\pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \gamma_q x_q \geq \pi_{r(\omega^k)} - \sum_{(i,j)\in\mathcal{A}^k} \sum_{q\in\mathcal{U}_{ij}^k} x_q \alpha_{ijq}.$$

This completes the proof. ■

As noted in Remark 12, after computing π-variables by setting π_i equal to the shortest path length from node i to node t in the shortest-path network, there may exist alternative feasible values for the α-variables. We select these values by first considering arcs $(i,j) \in \mathcal{A}^k$ for which $|\mathcal{U}_{ij}^k| = 1$ and then considering arcs $(i,j) \in \mathcal{A}^k$ for which $|\mathcal{U}_{ij}^k| \geq 2$. Recall that we set $\alpha_{ijq} = 0$, $\forall q \in \mathcal{U}_{ij}^k$ for arcs such that $\pi_i - \pi_j - g_{ij}^k \leq 0$. We also set $\alpha_{ijq} = 0$ for all $(i,j) \in \mathcal{A}^k$ and $q \in \mathcal{U}_{ij}^k : \hat{x}_q = 1$. Let $\boldsymbol{\alpha}'$ be an initial value assignment for the α-variables computed as

$$\alpha'_{ijq} = \begin{cases} \pi_i - \pi_j - g_{ij}^k & \text{if } \pi_i - \pi_j - g_{ij}^k > 0 \text{ and } |\mathcal{U}_{ij}^k| = 1 \\ 0 \text{ otherwise.} \end{cases} \qquad \forall (i,j) \in \mathcal{A}^k,\ q \in \mathcal{U}_{ij}^k. \tag{5.23}$$

Note that if $|\mathcal{U}_{ij}^k| = 1$ and $\pi_i - \pi_j - g_{ij}^k > 0$, then $\hat{x}_q = 0$ for the only element $q \in \mathcal{U}_{ij}^k$. For all $q = 1, \ldots, n_1$ we define $\gamma'_q = \max_{(i,j)\in\mathcal{A}^k}\{\alpha'_{ijq} \mid q \in \mathcal{U}_{ij}^k\}$. Our strategy is to keep the γ-vector as small as possible in order to make (5.15) as strong as possible. Note that $\boldsymbol{\gamma}'$ establishes a lower bound on γ. Therefore, setting values for α_{ijq} in the range $[0, \gamma'_q]$ will not increase γ_q and hence will not weaken (5.15).

Accordingly, consider arcs $(i,j) \in \mathcal{A}^k$ for which $|\mathcal{U}_{ij}^k| \geq 2$ and $\pi_i - \pi_j - g_{ij}^k > 0$. If

$\sum_{q\in\mathcal{U}^k_{ij}:\hat{x}_q=0}\gamma'_q \geq \pi_i - \pi_j - g^k_{ij}$, then we set $\alpha_{ijq} = \gamma'_q$, $\forall q \in \mathcal{U}^k_{ij} : \hat{x}_q = 0$. Otherwise, we define $\delta_{ij} = \pi_i - \pi_j - g^k_{ij} - \sum_{q\in\mathcal{U}^k_{ij}:\hat{x}_q=0}\gamma'_q$, and select any $\hat{q} \in \mathcal{U}^k_{ij} : \hat{x}_{\hat{q}} = 0$. We set $\alpha_{ij\hat{q}} = \gamma'_{\hat{q}} + \delta_{ij}$, and then $\alpha_{ijq} = \gamma'_q$, $\forall q \in \mathcal{U}^k_{ij} \setminus \{\hat{q}\} : \hat{x}_q = 0$.

Observe that by selecting the α-variables as described above, we get that $\gamma_{\bar{q}} = \gamma'_{\bar{q}}$ for all $\bar{q} \in \{1, \ldots, n_1\}$ such that $\sum_{q\in\mathcal{U}^k_{ij}:\hat{x}_q=0}\gamma'_q \geq \pi_i - \pi_j - g^k_{ij}$ for all $(i,j) \in \mathcal{A}^k : \bar{q} \in \mathcal{U}^k_{ij}$.

Our second strategy uses lower bounds on the second-stage objective function. We identify the minimum achievable value of $\mathcal{Q}(\mathbf{x}, \omega^k)$ when a given capacitated arc is traversed. To compute the desired bounds, we solve the following optimization problem for every $(\hat{\imath}, \hat{\jmath}) \in \mathcal{A}^k$ such that $\mathcal{U}^k_{\hat{\imath}\hat{\jmath}} \neq \emptyset$:

$$l_{\hat{\imath}\hat{\jmath}} = \min \sum_{(i,j)\in\mathcal{A}^k} g^k_{ij} v_{ij} \tag{5.24a}$$

$$\text{s.t.} \sum_{\{j|(i,j)\in\mathcal{A}^k\}} v_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}^k\}} v_{ji} = \begin{cases} 1, & \text{for } i = r(\omega^k) \\ 0, & \text{for } i \in \mathcal{N}^k \backslash \{r(\omega^k), t\} \\ -1, & \text{for } i = t \end{cases} \tag{5.24b}$$

$$v_{ij} \leq x_q \qquad \forall (i,j) \in \mathcal{A}^k,\ q \in \mathcal{U}^k_{ij} \tag{5.24c}$$

$$v_{\hat{\imath}\hat{\jmath}} = 1 \tag{5.24d}$$

$$\mathbf{Ax} \geq \mathbf{b} \tag{5.24e}$$

$$v_{ij} \geq 0 \qquad \forall (i,j) \in \mathcal{A}^k \tag{5.24f}$$

$$\mathbf{x} \in \{0,1\}^{n_1}. \tag{5.24g}$$

Constraint (5.24d) ensures that arc $(\hat{\imath}, \hat{\jmath})$ is traversed and in conjunction with constraint (5.24c) implies that $x_q = 1$ for all $q \in \mathcal{U}^k_{\hat{\imath}\hat{\jmath}}$. Constraints (5.24e) enforce the first-stage constraints.

The values computed from (5.24) enable us to strengthen our cutting planes due to the following proposition.

Proposition 16. *Define* $\underline{\alpha}_{ijq} = \min\{\alpha_{ijq}, (\pi_{r(\omega^k)} - l_{ij})^+\}$ *for all* $(i,j) \in \mathcal{A}^k$ *such that* $\mathcal{U}^k_{ij} \neq \emptyset$. *For all* $q = 1, \ldots, n_1$ *define* $\underline{\gamma}_q = \max_{(i,j)\in\mathcal{A}^k}\{\underline{\alpha}_{ijq} \mid q \in \mathcal{U}^k_{ij}\}$. *The following is a valid inequality that is at least as strong as* (5.15).

$$z_k \geq \pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \underline{\gamma}_q x_q. \tag{5.25}$$

Proof For a given $\mathbf{x} \in \mathcal{X}$, let $\mathbf{v}^*$ be an optimal solution to LSP$(\mathbf{x}, \omega^k)$. If there exists an arc $(i,j) \in \mathcal{A}^k$ and a first-stage variable $x_{\hat{q}}$ for which $v^*_{ij} = 1$, $\hat{q} \in \mathcal{U}^k_{ij}$, and $\underline{\alpha}_{ij\hat{q}} = \pi_{r(\omega^k)} - l_{ij}$, then $\pi_{r(\omega^k)} - \underline{\alpha}_{ij\hat{q}} = l_{ij}$ and $l_{ij} \leq \mathcal{Q}(\mathbf{x}, \omega^k)$, because $(\mathbf{v}^*, \mathbf{x})$ is a feasible solution to problem (5.24) defined for arc (i,j). Since $\underline{\gamma}_{\hat{q}} \geq \underline{\alpha}_{ij\hat{q}}$, we have that

$$\pi_{r(\omega^k)} - \underline{\gamma}_{\hat{q}} \leq \pi_{r(\omega^k)} - \underline{\alpha}_{ij\hat{q}}. \tag{5.26}$$

Now, because $\underline{\gamma}_q \geq 0$ for all $q = 1, \ldots, n_1$ and $v^*_{ij} = 1$ implies that $x_{\hat{q}} = 1$, we get that

$$\pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \underline{\gamma}_q x_q \leq \pi_{r(\omega^k)} - \underline{\gamma}_{\hat{q}}. \tag{5.27}$$

We conclude that

$$\pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \underline{\gamma}_q x_q \leq \mathcal{Q}(\mathbf{x}, \omega^k), \tag{5.28}$$

and thus (5.25) is valid.

Otherwise, we have that $\underline{\alpha}_{ijq} = \alpha_{ijq}$ for all $(i,j) \in \mathcal{A}^k : v^*_{ij} = 1,\ q \in \mathcal{U}^k_{ij}$. Since $\sum_{(i,j)\in\mathcal{A}^k : q \in \mathcal{U}^k_{ij}} v^*_{ij} \leq 1$ for all $q = 1, \ldots, n_1$, we have that

$$\pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \underline{\gamma}_q x_q \leq \pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \sum_{(i,j)\in\mathcal{A}^k : q \in \mathcal{U}^k_{ij}} x_q v^*_{ij} \underline{\gamma}_q. \tag{5.29}$$

Now, for any $q \in \{1, \ldots, n_1\}$, it holds that $\underline{\gamma}_q \geq \underline{\alpha}_{ijq}$ for all $(i,j) \in \mathcal{A}^k : q \in \mathcal{U}^k_{ij}$. This implies that

$$\pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \sum_{(i,j)\in\mathcal{A}^k : q \in \mathcal{U}^k_{ij}} x_q v^*_{ij} \underline{\gamma}_q \leq \pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \sum_{(i,j)\in\mathcal{A}^k : q \in \mathcal{U}^k_{ij}} x_q v^*_{ij} \underline{\alpha}_{ijq}. \tag{5.30}$$

Replacing $\underline{\alpha}_{ijq}$ by α_{ijq} in the right-hand side of (5.30) yields

$$\pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \underline{\gamma}_q x_q \leq \pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \sum_{(i,j)\in\mathcal{A}^k : q \in \mathcal{U}^k_{ij}} x_q v^*_{ij} \alpha_{ijq}. \tag{5.31}$$

Because of (5.19) we have that

$$\pi_{r(\omega^k)} - \sum_{q=1}^{n_1} \sum_{(i,j)\in\mathcal{A}^k : q\in\mathcal{U}_{ij}^k} x_q v_{ij}^* \alpha_{ijq} \leq \mathcal{Q}(\mathbf{x}, \omega^k), \tag{5.32}$$

and we conclude that (5.25) is valid.

Since $\underline{\gamma}_q \leq \gamma_q$, for all $q = 1, \ldots, n_1$, we conclude that (5.25) is at least as strong as (5.15). This completes the proof. ■

Problems (5.24) are generally $\mathcal{NP}$-hard due to (5.24e). However, they must be solved only once at a preprocessing stage and would typically not constitute a bottleneck in the algorithm. Alternatively, we could obtain $l_{\hat{i}\hat{j}}$ by solving a relaxation of (5.24), and use those bounds in lieu of $l_{\hat{i}\hat{j}}$ in (5.25).

5.4 Computational Experiments

This section presents computational results on a stochastic vertex covering problem (SVCP). We formulate the SVCP in Section 5.4.1 and describe our set of test instances in Section 5.4.2. In Section 5.4.3 we compare our algorithm's performance with the deterministic equivalent monolithic formulation presented in Section 5.1.

We coded our algorithm in Java using Eclipse SDK version 4.6.1. We executed all computational experiments on a machine having an Intel Core i7–3537U CPU (two cores) running at 2.00 GHz with 8 GB of RAM on Windows 8. All optimization problems were solved using Gurobi 5.6.0 with an imposed time limit of one hour (3600s).

5.4.1 Stochastic Vertex Cover Problem Formulation

The SVCP is a two-stage problem on an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. The decision maker can remove up to B edges from $\mathcal{E}$ in the first stage before solving a minimum weighted vertex cover problem in the second stage. The cost of including a vertex in the cover is uncertain. Define $f(\omega)_i$ as the cost of including vertex $i \in \mathcal{V}$ in the cover under scenario ω. Let x_{ij} be a binary variable that takes a value of 1 if and only if edge $(i, j) \in \mathcal{E}$ is removed from the graph in the first stage and let y_i be a binary variable that takes a value of 1 if and only if vertex i is included in the cover in the second stage. We formulate the SCVP as:

$$
\begin{aligned}
z^* = \min \quad & \mathbb{E}_{\omega \in \Omega}\left[\mathcal{Q}(\mathbf{x}, \omega)\right] && (5.33a)\\
\text{s.t.} \quad & \sum_{(i,j)\in\mathcal{E}} x_{ij} \leq B && (5.33b)\\
& \mathbf{x} \in \{0,1\}^{|\mathcal{E}|}, && (5.33c)
\end{aligned}
$$

with $\mathcal{Q}(\mathbf{x}, \omega)$ defined as the optimal objective function value to the following vertex cover problem:

$$
\begin{aligned}
\mathcal{Q}(\mathbf{x}, \omega) = \min \quad & \sum_{i\in\mathcal{V}} f(\omega)_i y_i && (5.34a)\\
\text{s.t.} \quad & y_i + y_j \geq 1 - x_{ij} \quad \forall (i,j) \in \mathcal{E} && (5.34b)\\
& \mathbf{y} \in \{0,1\}^{|\mathcal{V}|}. && (5.34c)
\end{aligned}
$$

The objective function (5.33a) minimizes the total expected second-stage cost. Constraint (5.33b) imposes a cardinality constraint on the number of arcs that can be removed from the graph in the first stage. Constraints (5.33c) ensure that the x-variables are binary. The second-stage objective function (5.34a) minimizes the total cost of including vertices in the cover. Constraints (5.34b) ensure that each edge is incident to at least one vertex in the cover, except for those edges removed from the graph in the first stage. Constraints (5.34c) ensure that the y-variables are binary.

Note that the second-stage constraints remain the same across all the scenarios. As a result, we only generate one BDD and consider scenario-dependent arc costs.

5.4.2 Stochastic Vertex Cover Problem Instances

Our set of test instances consists of 225 instances derived from 15 networks. These networks belong to three different classes.

- The first class of networks are *compiler* instances obtained from control-flow graphs for C compilations, taken from [Hicks, 2002].
- The second class of networks are *grid* networks taken from [Lozano and Smith, 2016].
- The third class of networks are planar *TSP* graphs obtained as Delaunay triangulations of traveling salesman instances from the TSPLIB [Reinelt, 1991], taken from [Hicks, 2002].

For each original network we generate five instances having random vertex cost coefficients following a $N(10, 1)$ distribution for each scenario, five instances having cost coefficients following a $N(10, 2)$ distribution, and five instances having cost coefficients following a $N(10, 3)$ distribution, where $N(\mu, \sigma)$ denotes a normal distribution with mean μ and standard deviation σ. We solve each instance nine times with different B values in $\{2, 3, 5\}$ and number of scenarios, K, in $\{100, 1000, 5000\}$. No instance contains an isolated variable.

Table 5.1 presents the width for the branch decomposition trees obtained using the modified approach of Hicks [2002] on the instances in our dataset. The first two columns show the network class and name. Columns 3–4 present the number of nodes and arcs, respectively. Column 5 shows the width of the branch decomposition tree. Note that instances derived from the same network have equal width, because the width is independent from the vertex cost coefficients.

Table 5.1: Dataset description and average branch decomposition tree width

Class	**Network**	**Nodes**	**Arcs**	**Width**
	nprio	17	22	3
	bcndb	25	34	4
Compiler	rffti1	67	89	4
	bcnd	55	77	5
	fmin	78	105	6
	5×5	27	86	6
	8×8	66	254	9
Grid	8×16	130	542	9
	10×10	102	416	11
	10×12	122	508	11
	eil51	51	140	9
	rat99	99	279	10
TSP	pr76	76	218	11
	eil76	76	215	12
	rd100	100	285	14

Compiler networks have the smallest width values, followed by grid networks, and TSP networks. Table 5.2 presents the number of nodes and arcs of the final BDD obtained after phase two of our graph generation process. As noted in Remark 10, we remove from the BDD long arcs a for which $|\mathcal{U}_a^k| > B$, because those arcs correspond to an infeasible first-stage solution.

Table 5.2: Number of nodes and arcs in the proposed BDD

Network	$B = 2$		$B = 3$		$B = 5$	
	Nodes	**Arcs**	**Nodes**	**Arcs**	**Nodes**	**Arcs**
nprio	96	188	96	190	96	190
bcndb	184	361	184	366	184	366
rffti1	570	1133	570	1138	570	1138
bcnd	508	1005	508	1014	508	1014
fmin	1062	2101	1062	2120	1062	2122
5×5	775	1231	857	1415	1030	1848
8×8	11,021	16,449	12,975	20,456	18,369	31,190
8×16	22,517	33,153	26,903	41,880	39,297	65,966
10×10	53,148	77,512	64,204	99,897	99,798	165,483
10×12	63,850	92,756	77,506	120,223	121,780	201,303
eil51	8417	13,109	10,382	17,375	12,094	22,354
rat99	19,407	30,073	23,937	39,885	27,720	51,150
pr76	21,513	33,198	27,229	45,017	32,472	59,456
eil76	49,223	74,827	66,391	107,498	85,439	153,239
rd100	140,411	211,494	201,581	321,475	289,471	506,530

Table 5.2 shows that the size of the BDDs is an exponential increasing function of the width as proved in Proposition 14. The impact of removing infeasible long arcs, along with the resulting disconnected nodes, is greater for the grid and TSP networks, for which there is a considerable increase in the number of nodes and arcs for larger values of B.

5.4.3 Comparison with the Monolithic Formulation

We compare our Benders' decomposition algorithm with the deterministic equivalent monolithic formulation. Tables 5.3–5.5 present the results of this experiment for instances whose cost coefficients follow an $N(10, 1)$, $N(10, 2)$, and $N(10, 3)$ distribution, respectively. The first column shows the original network name. The second column presents the number of scenarios. The remaining columns show the average CPU time in seconds (Avg), and the number of instances solved within the one-hour time limit (# sol), for both algorithms. We record a time of 3600 seconds if the instance is not solved within the time limit.

Table 5.3: Comparing the Benders' decomposition algorithm to the monolithic formulation on compiler instances having an $N(10, 1)$ cost-coefficient distribution

		$B = 2$				$B = 3$				$B = 5$			
Network	**Scenarios**	**Benders**		**Monolithic**		**Benders**		**Monolithic**		**Benders**		**Monolithic**	
		Avg	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**
nprio	100	<1	5	<1	5	<1	5	3	5	<1	5	3	5
	1000	<1	5	5	5	<1	5	43	5	1	5	36	5
	5000	<1	5	153	5	1	5	1038	5	3	5	810	5
bcndb	100	<1	5	2	5	<1	5	<1	5	<1	5	3	5
	1000	<1	5	82	5	<1	5	10	5	1	5	67	5
	5000	1	5	1622	5	1	5	294	5	3	5	1208	5
rffti1	100	<1	5	1	5	<1	5	<1	5	2	5	1	5
	1000	<1	5	19	5	1	5	21	5	6	5	28	5
	5000	1	5	484	5	5	5	586	5	21	5	804	5
bcnd	100	<1	5	<1	5	<1	5	<1	5	4	5	10	5
	1000	<1	5	21	5	1	5	21	5	16	5	359	5
	5000	3	5	594	5	5	5	563	5	84	5	3600	0
fmin	100	<1	5	1	5	1	5	1	5	4	5	1	5
	1000	3	5	51	5	7	5	59	5	23	5	79	5
	5000	14	5	1228	5	40	5	1541	5	113	5	2137	5

Table 5.4: Comparing the Benders' decomposition algorithm to the monolithic formulation on compiler instances having an $N(10, 2)$ cost-coefficient distribution

		$B = 2$				$B = 3$				$B = 5$			
Network	**Scenarios**	**Benders**		**Monolithic**		**Benders**		**Monolithic**		**Benders**		**Monolithic**	
		Avg	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**
nprio	100	<1	5	<1	5	<1	5	2	5	1	5	5	5
	1000	<1	5	5	5	<1	5	51	5	1	5	38	5
	5000	<1	5	167	5	1	5	910	5	4	5	907	5
bcndb	100	<1	5	2	5	<1	5	<1	5	<1	5	3	5
	1000	<1	5	97	5	<1	5	10	5	1	5	106	5
	5000	1	5	1543	5	1	5	324	5	6	5	1260	5
rffti1	100	<1	5	1	5	<1	5	<1	5	4	5	1	5
	1000	<1	5	20	5	1	5	22	5	18	5	29	5
	5000	1	5	514	5	3	5	622	5	65	5	871	5
bcnd	100	<1	5	<1	5	<1	5	<1	5	5	5	12	5
	1000	1	5	24	5	1	5	23	5	21	5	395	5
	5000	3	5	659	5	4	5	602	5	100	5	3555	1
fmin	100	<1	5	1	5	1	5	1	5	27	5	1	5
	1000	3	5	53	5	9	5	64	5	81	5	86	5
	5000	15	5	1333	5	53	5	1694	5	314	5	2334	5

Table 5.5: Comparing the Benders' decomposition algorithm to the monolithic formulation on compiler instances having an $N(10, 3)$ cost-coefficient distribution

		$B = 2$				$B = 3$				$B = 5$			
Network	**Scenarios**	**Benders**		**Monolithic**		**Benders**		**Monolithic**		**Benders**		**Monolithic**	
		Avg	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**
nprio	100	<1	5	<1	5	<1	5	2	5	1	5	5	5
	1000	<1	5	5	5	<1	5	50	5	1	5	38	5
	5000	<1	5	172	5	1	5	1212	5	4	5	1143	5
bcndb	100	<1	5	2	5	<1	5	<1	5	1	5	15	5
	1000	<1	5	105	5	<1	5	11	5	2	5	117	5
	5000	1	5	1968	5	2	5	335	5	12	5	1350	5
rffti1	100	<1	5	1	5	<1	5	<1	5	15	5	1	5
	1000	<1	5	20	5	1	5	24	5	34	5	31	5
	5000	1	5	518	5	4	5	646	5	139	5	901	5
bcnd	100	<1	5	<1	5	<1	5	<1	5	13	5	10	5
	1000	1	5	23	5	2	5	24	5	35	5	428	5
	5000	3	5	599	5	10	5	616	5	149	5	3600	0
fmin	100	<1	5	1	5	2	5	1	5	475	5	1	5
	1000	3	5	51	5	16	5	65	5	895	5	79	5
	5000	17	5	1293	5	95	5	1636	5	1949	5	2225	5

Tables 5.3–5.5 show that our algorithm executes substantially faster than the monolithic formulation over the compiler instances, consistently reducing computational time for the probability distributions examined. Both algorithms solve the instances having 100 scenarios in a few seconds on average. Over the instances having 1000 scenarios, the Benders algorithm exhibits speedups of up to roughly 20 over the monolithic formulation (Table 5.3, bcnd network, and $B = 5$). Over the instances having 5000 scenarios, the Benders algorithm dramatically outperforms the monolithic formulation. Moreover, our algorithm solves all instances within the time limit, while the monolithic formulation fails to solve 14 bcnd instances.

Increasing the value of B increases the computational time for both algorithms. For our proposed Benders algorithm, the computational time increases from 17 seconds to almost 2000 seconds as B grows from 2 to 5 (Table 5.3, fmin network, and 5000 scenarios). Increasing the standard deviation increases the computational time for our Benders algorithm and slightly decreases the computational time for the monolithic formulation. Also note that our algorithm performance is directly related to the width of the branch decomposition tree. Instances having a larger width take considerably more time to solve, especially when $B = 5$.

Table 5.6 presents a summary of the results for the grid and TSP instances. When computing the average CPU time, we again record a computational time of 3600 seconds for instances that reach the time limit.

Table 5.6: Comparing the Benders' decomposition algorithm to the monolithic formulation on grid and TSP networks

Network	Scenarios	$B = 2$				$B = 3$				$B = 5$			
		Benders		**Monolithic**		**Benders**		**Monolithic**		**Benders**		**Monolithic**	
class		**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**	**Avg**	**# sol**
	100	50	75	2931	16	1173	60	2881	15	3042	15	2949	15
Grid	1000	1660	45	3600	0	2222	30	3600	0	3101	15	3600	0
	5000	2881	15	3600	0	2881	15	3600	0	3507	15	3600	0
	100	25	75	3190	23	652	70	3589	3	3600	0	3600	0
TSP	1000	797	60	3600	0	2405	35	3600	0	3600	0	3600	0
	5000	2921	15	3600	0	3488	5	3600	0	3600	0	3600	0

Table 5.6 shows that overall, our algorithm outperforms the monolithic formulation over the grid and TSP instances. When $B = 2$ our algorithm solves all instances having 100 scenarios in less than a minute, more than half of the instances having 1000 scenarios in less than 30 minutes, and one-fifth of the instances having 5000 scenarios in under 50 minutes. On the other hand, the monolithic formulation only solves 39 instances having 100 scenarios and none of the instances having 1000 and 5000 scenarios. When $B = 3$ our algorithm solves 130 instances having 100 scenarios in less than 20 minutes, 75 instances having 1000 scenarios, and 20 instances having 5000 scenarios, while the monolithic formulation only solves 18 instances having 100 scenarios and none of the others. When $B = 5$ our algorithm solves 45 instances derived from grid networks, while the monolithic formulation solves 15. Both algorithms fail to solve any of the instances derived from TSP networks when $B = 5$, which is explained by the fact that these are the instances having the largest widths from the dataset.

Chapter 6

Conclusions

In Chapter 2 we present an exact approach for solving bilevel mixed-integer programs under the three assumptions listed in Section 2.1. Obtaining upper bounds for the BMIP is one of the major challenges for devising solution approaches. Our proposed algorithm is built upon a relaxation that imposes constraints generated from a sample of feasible follower responses. These relaxed problems yield upper bounds, whose strength depends on the sample selected. Thus, our algorithm iteratively updates the sample with the goal of obtaining tighter upper bounds, while discovering feasible solutions that yield lower bounds. From a computational perspective, our algorithm outperforms the current state-of-the-art algorithm for BMILPs, achieving speedups of up to 17 times. We also present a featured study on competitive scheduling that illustrates the flexibility of our approach on a nonlinear model that employs the pessimistic assumption.

One line of future research may seek to broaden the scope of problems that can be solved within this framework. The strongest assumption we make is that the leader variables must all be integer-valued. If some of the leader variables are continuous, and if they interact in constraints or the objective with the follower variables, then our approach will not terminate finitely. It may be possible to execute a continuous-variables branch-and-bound scheme for this purpose, but with the caveat that new relaxations and restrictions will be needed to obtain valid bounds over these regions. Another line of future research may entail a thorough study of an important application area, such as the natural gas cash-out problem [Dempe et al., 2005, 2011, Kalashnikov et al., 2010]. Although our research here contributes a promising new way to solve bilevel discrete optimization problems, the implementation details for specific applications require a substantial amount of attention.

Furthermore, as discussed in Section 2.1, there exist many algorithms that have been proposed for the types of problems presented here. Many of these algorithms have varying assumptions on variable integrality and function nonlinearities allowed, but there exist several problems (e.g., linear (pure) integer programs with bounded variables) on which these algorithms all produce optimal solutions. A thorough computational study that compares these algorithms on various problem classes would be of substantial value to the community. We leave these questions for further study.

In Chapter 3 we propose a novel framework for solving interdiction and fortification problems having binary variables in the first two stages, which allows the third-stage problem to take any form. Previous methods for solving these problems convert the second-stage (interdiction) problem to a bilinear programming problem using the strong dual of the third-stage problem. However, when a (polynomial-size) strong dual formulation cannot be found, this reformulation approach is not appropriate. Even when dualization of the third-stage problem is practical, the resulting bilinear interdiction program is usually converted to a large linear mixed-integer program that often exhibits a weak linear programming relaxation and requires a substantial amount of time to solve.

Our approach obviates both of these difficulties by iteratively sampling feasible solutions to the third-stage problem, and finitely converges to an optimal solution. Computationally, we demonstrate that the approach significantly outperforms prior approaches to solving shortest-path interdiction and fortification problems, and is also capable of solving the CLSIPF (in which the third-stage problem is NP-hard) within reasonable computational times.

Future research will examine how this framework can be adapted in the context of more difficult recourse problems. The shortest-path and lot-sizing problems demonstrate how a direct application of the framework can be used to effectively solve very difficult problems, but a focused study might yield new insights on how specific problem structures can be exploited within this framework. Also, while three-stage problems in the literature almost exclusively contain only binary variables in the first two stages, an interesting challenge would be to investigate how this approach can accommodate fractional rather than binary attack and/or fortification actions.

In Chapter 4 we solve for the first time the traveling salesman problem with interdiction and fortification, which is particularly challenging since its recourse problem is strongly $\mathcal{NP}$-hard. Previous approaches for interdiction problems from the literature usually rely on strong duality thus requiring the recourse problem to be linear or convex. We circumvent this limitation by specializing the sampling approach proposed in Chapter 3 and proposing a two-phase exact algorithm that is based on alternative restrictions for the TSP. Our approach is able to optimally solve moderate-sized instances with up to 96 nodes for the symmetric case and 64 nodes for the asymmetric case in reasonable computational times. The proposed TSP restrictions can be also used as stand-alone heuristics that quickly find near-optimal solutions.

In Chapter 5 we focused on a class of two-stage stochastic integer programming problems having set-covering constraints in the second stage. In order to allow the use of Benders' decomposition for these problems, we converted the second-stage integer programming problems into shortest-path problems parameterized by the first-stage variables. This conversion employed binary decision diagrams to create the shortest-path problem, and leveraged results from branch decompositions to limit the size of the shortest-path network. We demonstrated the efficacy of this approach on stochastic vertex cover problems taken from instances appearing in the literature.

While many future research directions stem from the approach introduced here, the most important one might be to expand this idea to handle broader classes of stochastic integer programming problems. A naive extension of our approach could indeed handle problems in which the assumptions on $W(\omega^k)$, $h(\omega^k)$, and $T(\omega^k)$ are relaxed, but at the expense of second-stage shortest-path problems that are too large to solve within reasonable computational limits. Research is thus needed to possibly generate partial BDD networks for this case, which can be iteratively refined to yield an optimal solution. Also, the formulations here do not allow for the presence of continuous variables. Future research may examine methods to implicitly optimize over continuous variables, given fixed integer variables selected within this process, using the proposed framework.

Bibliography

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

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Upper Saddle River, NJ, 1993.

R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1):75–102, 2002.

S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, c-27(6):509–516, 1978.

A. Alonso-Ayuso, L. F. Escudero, and M. T. Ortuno. BFC, a branch-and-fix coordination algorithmic framework for solving some types of stochastic pure and mixed 0–1 programs. *European Journal of Operational Research*, 151(3):503–519, 2003.

D. Applegate, R. Bixby, V. Chvátal, and W. Cook. CONCORDE TSP solver, available at `www.math.uwaterloo.ca/tsp/concorde.html`.

D. Applegate, R. Bixby, W. Cook, and V. Chvátal. *On the solution of traveling salesman problems.* Rheinische Friedrich-Wilhelms-Universität, Bonn, Germany, 1998.

E. Balas and R. Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69, 1972.

J. F. Bard. An efficient point algorithm for a linear two-stage optimization problem. *Operations Research*, 31(4):670–684, 1983.

J. F. Bard and J. T. Moore. An algorithm for the discrete bilevel programming problem. *Naval Research Logistics*, 39(3):419–435, 1992.

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

B. Becker, M. Behle, F. Eisenbrand, and R. Wimmer. BDDs in a branch and cut framework. In S. E. Nikoletseas, editor, *Lecture Notes in Computer Science: Experimental and Efficient Algorithms*, volume 3503, pages 452–463. Springer Berlin Heidelberg, 2005.

M. Behle and F. Eisenbrand. 0/1 vertex and facet enumeration with BDDs. In D. Applegate and G. S. Brodal, editors, *Proceedings of the Meeting on Algorithm Engineering & Experiments*, pages 158–165, New Orleans, LA, 2007. Society for Industrial and Applied Mathematics.

G. Belvaux and L. A. Wolsey. bc–prod: A specialized branch-and-cut system for lot-sizing problems. *Management Science*, 46(5):724–738, 2000.

J. F. Benders. Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016.

G. R. Bitran and H. H. Yanasse. Computational complexity of the capacitated lot size problem. *Management Science*, 28(10):1174–1186, 1982.

G. M. Bollas, P. I. Barton, and A. Mitsos. Bilevel optimization formulation for parameter estimation in vapor–liquid (–liquid) phase equilibrium problems. *Chemical Engineering Science*, 64(8):1768–1783, 2009.

N. Brahimi, S. Dauzere-Peres, N. M. Najid, and A. Nordli. Single item lot sizing problems. *European Journal of Operational Research*, 168(1):1–16, 2006.

L. Brotcorne, M. Labbé, P. Marcotte, and G. Savard. A bilevel model for toll optimization on a multicommodity transportation network. *Transportation Science*, 35(4):345–358, 2001.

G. Brown, M. Carlyle, D. Diehl, J. Kline, and K. Wood. A two-sided optimization for theater ballistic missile defense. *Operations Research*, 53(5):745–763, 2005a.

G. G. Brown, W. M. Carlyle, J. Salmerón, and R. K. Wood. Analyzing the vulnerability of critical infrastructure to attack and planning defenses. In H. J. Greenberg and J. C. Smith, editors, *Tutorials in Operations Research: Emerging Theory, Methods, and Applications*, pages 102–123. INFORMS, Hanover, MD, 2005b.

G. G. Brown, W. M. Carlyle, J. Salmerón, and R. K. Wood. Defending critical infrastructure. *Interfaces*, 36(6):530–544, 2006.

G. G. Brown, W. M. Carlyle, R. Harney, E. Skroch, and R. K. Wood. Interdicting a nuclear-weapons project. *Operations Research*, 57(4):866–877, 2009.

R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 100(8):677–691, 1986.

A. P. Burgard, P. Pharkya, and C. D. Maranas. Optknock: a bilevel programming framework for identifying gene knockout strategies for microbial strain optimization. *Biotechnology and Bioengineering*, 84(6):647–657, 2003.

L. Buriol, P. M. França, and P. Moscato. A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics*, 10(5):483–506, 2004.

W. Candler and R. Townsley. A linear two-level programming problem. *Computers and Operations Research*, 9(1):59–76, 1982.

P. Cappanera and M. P. Scaparra. Optimal allocation of protective resources in shortest-path networks. *Transportation Science*, 45(1):64–80, 2011.

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

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

R. L. Church and M. P. Scaparra. The r-interdiction median problem with fortification. *Geographical Analysis*, 39(2):129–146, 2007.

R. L. Church, M. P. Scaparra, and R. S. Middleton. Identifying critical infrastructure: The median and covering facility interdiction problems. *Annals of the Association of American Geographers*, 94(3):491–502, 2004.

A. A. Cire and W.-J. van Hoeve. Multivalued decision diagrams for sequencing problems. *Operations Research*, 61(6):1411–1428, 2013.

G. Codato and M. Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

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

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

E. Delage and D. A. Iancu. Robust multistage decision making. In D. M. Aleman and A. C. Thiele, editors, *Tutorials in Operations Research*, pages 20–46. INFORMS, Catonsville, MD, 2015.

S. Dempe and S. Franke. On the solution of convex bilevel optimization problems. *Computational Optimization and Applications*, 2015. Article in press. doi: 10.1007/s10589-015-9795-8.

S. Dempe and M. Pilecka. Necessary optimality conditions for optimistic bilevel programming problems using set-valued programming. *Journal of Global Optimization*, 61(4):769–788, 2015.

S. Dempe and A. B. Zemkoho. Bilevel road pricing: theoretical analysis and optimality conditions. *Annals of Operations Research*, 196(1):223–240, 2012.

S. Dempe and A. B. Zemkoho. The bilevel programming problem: reformulations, constraint qualifications and optimality conditions. *Mathematical Programming*, 138(1):1–27, 2013.

S. Dempe and A. B. Zemkoho. KKT reformulation and necessary conditions for optimality in nonsmooth bilevel optimization. *SIAM Journal on Optimization*, 24(4):1639–1669, 2014.

S. Dempe, V. V. Kalashnikov, and R. Z. Rios-Mercado. Discrete bilevel programming: application to a natural gas cash-out problem. *European Journal of Operational Research*, 166(2):469–488, 2005.

S. Dempe, J. Dutta, and B. S. Mordukhovich. New necessary optimality conditions in optimistic bilevel programming. *Optimization*, 56(5–6):577–604, 2007.

S. Dempe, V. Kalashnikov, G. A. Pérez-Valdés, and N. I. Kalashnykova. Natural gas bilevel cash-out problem: convergence of a penalty function method. *European Journal of Operational Research*, 215(3):532–538, 2011.

S. Dempe, B. S. Mordukhovich, and A. B. Zemkoho. Necessary optimality conditions in pessimistic bilevel programming. *Optimization*, 63(4):505–533, 2014.

S. Dempe, V. Kalashnikov, G. A. Pérez-Valdés, and N. Kalashnykova. *Bilevel Programming Problems*. Springer, Heidelberg, 2015.

S. T. DeNegre and T. K. Ralphs. A branch-and-cut algorithm for integer bilevel linear programs. In J. W. Chinneck, B. Kristjansson, and M. J. Saltzman, editors, *Operations Research and Cyber-Infrastructure*, pages 65–78. Springer, New York, 2009.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1): 269–271, 1959.

L. F. Domínguez and E. N. Pistikopoulos. Multiparametric programming based algorithms for pure integer and mixed-integer bilevel programming problems. *Computers and Chemical Engineering*, 34(12):2097–2106, 2010.

G. D. Eppen and R. K. Martin. Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35(6):832–848, 1987.

L. F. Escudero, M. A. Garín, M. Merino, and G. Pérez. An exact algorithm for solving large-scale two-stage stochastic mixed-integer problems: Some theoretical and experimental aspects. *European Journal of Operational Research*, 204(1):105–116, 2010.

N. P. Faisca, V. Dua, B. Rustem, P. M. Saraiva, and E. Pistikopoulos. Parametric global optimisation for bilevel programming. *Journal of Global Optimization*, 38(4):609–623, 2007.

D. Fanghänel and S. Dempe. Bilevel programming with discrete lower level problems. *Optimization*, 58(8):1029–1047, 2009.

M. M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.

M. Florian, J. K. Lenstra, and A. H. G. Rinnooy Kan. Deterministic production planning: Algorithms and complexity. *Management Science*, 26(7):669–679, 1980.

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

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

B. Golden. A problem in network interdiction. *Naval Research Logistics Quarterly*, 25(4):711–713, 1978.

B. L. Gorissen and D. den Hertog. Robust counterparts of inequalities containing sums of maxima of linear functions. *European Journal of Operational Research*, 277(1):30–43, 2013.

W. K. K. Haneveld and M. H. van der Vlerk. Stochastic integer programming: General models and algorithms. *Annals of Operations Research*, 85(0):39–57, 1999.

W. K. K. Haneveld, L. Stougie, and M. H. van der Vlerk. Simple integer recourse models: convexity and convex approximations. *Mathematical Programming*, 108(2):435–473, 2006.

P. Hansen, B. Jaumard, and G. Savard. New branch-and-bound rules for linear bilevel programming. *SIAM Journal of Scientific and Statistical Computing*, 13(5):1194–1217, 1992.

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

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.

K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

M. Hemmati and J. C. Smith. A mixed-integer bilevel programming approach for a competitive prioritized set covering problem. *Discrete Optimization*, to appear, 2016.

I. V. Hicks. Branchwidth heuristics. *Congressus Numerantium*, 159:31–50, 2002.

J. N. Hooker. Decision diagrams and dynamic programming. In C. Gomes and M. Sellmann, editors, *Lecture Notes in Computer Science: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874, pages 94–110. Springer Berlin Heidelberg, 2013.

J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96 (1):33–60, 2003.

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

V. V. Kalashnikov, G. A. Pérez, and N. I. Kalashnykova. A linearization approach to solve the natural gas cash-out bilevel problem. *Annals of Operations Research*, 181(1):423–442, 2010.

B. Karimi, S. M. T. Fatemi Ghomi, and J. M. Wilson. The capacitated lot sizing problem: a review of models and algorithms. *Omega*, 31(5):365–378, 2003.

U. S. Karmarkar, S. Kekre, and S. Kekre. The dynamic lot-sizing problem with startup and reservation costs. *Operations Research*, 35(3):389–398, 1987.

B. Kell and W.-J. van Hoeve. An MDD approach to multidimensional bin packing. In C. Gomes and M. Sellmann, editors, *Lecture Notes in Computer Science: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874, pages 128–143. Springer Berlin Heidelberg, 2013.

M. Köppe, M. Queyranne, and C. T. Ryan. Parametric integer programming algorithm for bilevel mixed integer programs. *Journal of Optimization Theory and Applications*, 146(1):137–150, 2010.

M. Labbé, P. Marcotte, and G. Savard. A bilevel model of taxation and its application to optimal highway pricing. *Management Science*, 44(12):1608–1622, 1998.

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

G. Laporte, F. V. Louveaux, and L. van Hamme. An integer L-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research*, 50(3):415–423, 2002.

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, New York, 1985.

C.-Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, 1959.

C. Lim and J. C. Smith. Algorithms for discrete and continuous multicommodity flow network interdiction problems. *IIE Transactions*, 39(1):15–26, 2007.

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

L. Lozano and A. L. Medaglia. On an exact method for the constrained shortest path problem. *Computers and Operations Research*, 40(1):378–384, 2013.

L. Lozano and J. C. Smith. A backward sampling framework for interdiction problems with fortification. *INFORMS Journal on Computing*, 29(1):123–139, 2016.

G. Lulli and S. Sen. A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems. *Management Science*, 50(6):786–796, 2004.

A. Migdalas. Bilevel programming in traffic planning: models, methods and challenge. *Journal of Global Optimization*, 7(4):381–405, 1995.

A. Mitsos. Global solution of nonlinear mixed-integer bilevel programs. *Journal of Global Optimization*, 47(4):557–582, 2010.

A. Mitsos, P. Lemonidis, and P. Barton. Global solution of Bilevel Programs with a nonconvex inner program. *Journal of Global Optimization*, 42(4):475–513, 2008. ISSN 0925-5001.

A. Mitsos, G. M. Bollas, and P. I. Barton. Bilevel optimization formulation for parameter estimation in liquid–liquid phase equilibrium problems. *Chemical Engineering Science*, 64(3):548–559, 2009a.

A. Mitsos, G. M. Bollas, and P. I. Barton. Model and parameter identification in phase equilibria. *Computer Aided Chemical Engineering*, 26:597–601, 2009b.

J. Moore. A n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968.

J. T. Moore and J. F. Bard. The mixed integer linear bilevel programming problem. *Operations Research*, 38(5):911–921, 1990.

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

L. Ntaimo and S. Sen. The million-variable "march" for stochastic combinatorial optimization. *Journal of Global Optimization*, 32(3):385–400, 2005.

F. Pan, W. Charlton, and D. P. Morton. Interdicting smuggled nuclear material. In D. L. Woodruff, editor, *Network Interdiction and Stochastic Integer Programming*, pages 1–20. Kluwer Academic Publishers, Boston, MA, 2003.

R. Peterson and E. A. Silver. *Decision systems for inventory management and production planning.* Wiley, New York, 1979.

V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

M. Prince, J. Geunes, and J. Smith. Procurement allocation planning with multiple suppliers under competition. *International Journal of Production Research*, 51(23–24):6900–6922, 2013a.

M. Prince, J. C. Smith, and J. Geunes. A three-stage procurement optimization problem under uncertainty. *Naval Research Logistics*, 60(1):395–412, 2013b.

A. Raith and M. Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers and Operations Research*, 36(4):1299–1331, 2009.

G. Reinelt. TSPLIB–A traveling salesman problem library. *INFORMS Journal on Computing*, 3(4): 376–384, 1991.

N. Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

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.

G. K. Saharidis and M. G. Ierapetritou. Resolution method for mixed integer bi-level linear problems based on decomposition technique. *Journal of Global Optimization*, 44(1):29–51, 2009.

J. Salmerón, K. Wood, and R. Baldick. Analysis of electric grid security under terrorist threat. *IEEE Transactions on Power Systems*, 19(2):905–912, 2004.

J. Salmerón, K. Wood, and R. Baldick. Worst-case interdiction analysis of large-scale electric power grids. *IEEE Transactions on Power Systems*, 24(1):96–104, 2009.

M. P. Scaparra and R. L. Church. A bilevel mixed-integer program for critical infrastructure protection planning. *Computers and Operations Research*, 35(6):1905–1923, 2008a.

M. P. Scaparra and R. L. Church. An exact solution approach for the interdiction median problem with fortification. *European Journal of Operational Research*, 189(1):76–92, 2008b.

R. Schultz. Stochastic programming with integer variables. *Mathematical Programming*, 97(1): 285–309, 2003.

R. Schultz, L. Stougie, and M. H. van der Vlerk. Solving stochastic programs with integer recourse by enumeration: A framework using Gröbner basis. *Mathematical Programming*, 83(1):229–252, 1998.

S. Sen. Algorithms for stochastic mixed-integer programming models. In K. Aardal, G. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, pages 515–558. Elsevier, Amsterdam, The Netherlands, 2005.

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

S. Sen and H. D. Sherali. Decomposition with branch-and-cut approaches for two-stage stochastic mixed-integer programming. *Mathematical Programming*, 106(2):203–223, 2006.

P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

H. D. Sherali and W. P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

H. D. Sherali and B. M. P. Fraticelli. A modification of Benders' decomposition algorithm for discrete subproblems: An approach for stochastic programs with integer recourse. *Journal of Global Optimization*, 22(1):319–342, 2002.

H. D. Sherali and J. C. Smith. Two-stage stochastic risk threshold and hierarchical multiple risk problems: Models and algorithms. *Mathematical Programming*, 120(2):403–427, 2009.

C. Shi, J. Lu, and G. Zhang. An extended Kuhn-Tucker approach for linear bilevel programming. *Applied Mathematics and Computation*, 162(1):51–63, 2005.

C. Shi, J. Lu, G. Zhang, and H. Zhou. An extended branch and bound algorithm for linear bilevel programming. *Applied Mathematics and Computation*, 180(2):529–537, 2006.

J. C. Smith. Basic interdiction models. In J. Cochran, editor, *Wiley Encyclopedia of Operations Research and Management Science*, pages 323–330. Wiley, Hoboken, NJ, 2010.

J. C. Smith and C. Lim. Algorithms for network interdiction and fortification games. In A. Migdalas, P. M. Pardalos, L. Pitsoulis, and A. Chinchuluun, editors, *Pareto Optimality, Game Theory and Equilibria*, Nonconvex Optimization and its Applications Series, pages 609–644. Springer, New York, 2008.

J. C. Smith, C. Lim, and F. Sudargho. Survivable network design under optimal and heuristic interdiction scenarios. *Journal of Global Optimization*, 38(2):181–199, 2007.

A. Tsoukalas, B. Rustem, and E. N. Pistikopoulos. A global optimization algorithm for generalized semi-infinite, continuous minimax with coupled constraints and bi-level problems. *Journal of Global Optimization*, 44(2):235–250, 2009.

H. Tuy, A. Migdalas, and P. Värbrand. A global optimization approach for the linear two-level program. *Journal of Global Optimization*, 3(1):1–23, 1993.

M. H. van der Vlerk. Convex approximations for complete integer recourse models. *Mathematical Programming*, 99(2):297–310, 2004.

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

W. Wiesemann, A. Tsoukalas, P.-M. Kleniati, and B. Rustem. Pessimistic bilevel optimization. *SIAM Journal on Optimization*, 23(1):353–380, 2013.

R. D. Wollmer. Removing arcs from a network. *Operations Research*, 12(6):934–940, 1964.

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

J. Xu and P. Wei. A bi-level model for location-allocation problem of construction & demolition waste management under fuzzy random environment. *International Journal of Civil Engineering*, 10(1):1–12, 2012.

J. Xu, Y. Tu, and Z. Zeng. Bilevel optimization of regional water resources allocation problem under fuzzy random environment. *Journal of Water Resources Planning and Management*, 139 (3):246–264, 2012.

P. Xu and L. Wang. An exact algorithm for the bilevel mixed integer linear programming problem under three simplifying assumptions. *Computers and Operations Research*, 41(1):309–318, 2014.

J. Ye and D. Zhu. Optimality conditions for bilevel programming problems. *Optimization*, 33(1): 9–27, 1995.

J. J. Ye. Constraint qualifications and KKT conditions for bilevel programming problems. *Mathematics of Operations Research*, 31(4):811–824, 2006.

J. Yen, J.-P. P. Richard, and J. C. Smith. A class of algorithms for mixed-integer bilevel min-max optimization. Journal of Global Optimization (to appear), 2014.