INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Backward Sampling Framework for Interdiction Problems with Fortification

Leonardo Lozano, J. Cole Smith

INFORMS Journal on Computing
Vol. 29, No. 1, Winter 2017, pp. 123–139
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

https://doi.org/10.1287/ijoc.2016.0721
© 2016 INFORMS

A Backward Sampling Framework for Interdiction Problems with Fortification

Leonardo Lozano, J. Cole Smith
Department of Industrial Engineering, Clemson University, Clemson, South Carolina 29634
{llozano@g.clemson.edu, jcsmith@clemson.edu}

This paper 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.

Keywords: interdiction; fortification; shortest path problem; capacitated lot sizing problem
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received December 2014; revised January 2016; accepted May 2016. Published online December 21, 2016.

1. Introduction

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 $\mathscr{W}$ be the set of feasible solutions to the first-stage problem. Let $\mathscr{X}(\mathbf{w})$ be the set of feasible second-stage solutions given a defense vector $\mathbf{w}$, and let $\mathscr{Y}(\mathbf{x})$ be the set of feasible third-stage solutions for a given attack vector $\mathbf{x}$. Also, define $\mathscr{X} = \bigcup_{\mathbf{w}\in\mathscr{W}} \mathscr{X}(\mathbf{w})$ and $\mathscr{Y} = \bigcup_{\mathbf{x}\in\mathscr{X}} \mathscr{Y}(\mathbf{x})$, i.e., $\mathscr{X}$ and $\mathscr{Y}$ 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:

$$\mathscr{P}: \quad z^* = \min_{\mathbf{w}\in\mathscr{W}} \max_{\mathbf{x}\in\mathscr{X}(\mathbf{w})} \min_{\mathbf{y}\in\mathscr{Y}(\mathbf{x})} f(\mathbf{y}). \tag{1}$$

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

These 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 paper are previous studies on defender-attacker-defender problems that fit within our problem framework. 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 (MIP) problem 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 this field 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. (2013) 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. (2013) 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 Tang et al. (2016) 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. Moore and Bard (1990) also consider bilevel problems, but assume that the interdictor seeks to optimize its own objective (instead of minimizing the defender's objective), and the interdictor's objective function depends on the defender's decisions. See also the work of Bard and Moore (1992), Dempe (2002), and Vicente et al. (1996) for related work on solving discrete bilevel problems.

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 paper is organized as follows. Section 2 presents the backward sampling framework and establishes the finite convergence of our approach to an optimal solution. Sections 3 and 4 describe how to specialize the framework for the SPIPF and CLSIPF, respectively. Section 5 presents our computational experiments, and Section 6 concludes the paper with a summary of our work and a discussion of future extensions.

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 2.1 describes our sampling procedure. Section 2.2 presents our proposed approach for solving the interdiction problems, and Section 2.3 discusses the outer optimization algorithm. Section 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 the paper in Table 1.

Table 1 Relevant Definitions and Notation

Symbol	Explanation
	Section 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 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 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 $z^R(\hat{\mathbf{x}}) = \min_{\mathbf{y} \in \mathcal{Y}(\hat{\mathbf{x}})} f(\mathbf{y})$
$\mathcal{Y}_z$	Set of feasible third-stage solutions whose objective value is less than or equal to z
	Section 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 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 $\mathcal{L}$

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, such as 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 and 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 $\hat{\mathcal{Y}}(\mathbf{x}) = \varnothing$, 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 $\hat{\mathcal{Y}}(\mathbf{x}) \neq \varnothing$ for any $\mathbf{x} \in \mathcal{X}$.

2.2. Solving Bilevel Interdiction Problems

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

$$\mathcal{Q}(\mathbf{w}): \quad z^I(\mathbf{w}) = \max_{\mathbf{x} \in \mathcal{X}(\mathbf{w})} \min_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} f(\mathbf{y}) \tag{2}$$

be its associated two-level interdiction problem. Note that if there exists a defense $\mathbf{w} \in \mathcal{W}$ such that $\mathcal{X}(\mathbf{w}) = \varnothing$, then problem (2) is not defined. Hence, without loss of generality, we assume that $\mathcal{X}(\mathbf{w}) \neq \varnothing$, $\forall\, \mathbf{w} \in \mathcal{W}$.

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

$$\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}}): \quad z^I(\mathbf{w}, \hat{\mathcal{Y}}) = \max_{\mathbf{x} \in \mathcal{X}(\mathbf{w})} \min_{\mathbf{y} \in \hat{\mathcal{Y}}(\mathbf{x})} f(\mathbf{y}) \tag{3}$$

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 1. *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

$$z^R(\hat{\mathbf{x}}) = \min_{\mathbf{y} \in \mathcal{Y}(\hat{\mathbf{x}})} f(\mathbf{y}). \tag{4}$$

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 2 states a condition in which an optimal solution to $\mathcal{Q}(\mathbf{w}, \hat{\mathcal{Y}})$ also optimizes $\mathcal{Q}(\mathbf{w})$.

Proposition 2. *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 1 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 2 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 2 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 1. 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 3.2.1 and 4.2.1 present our sampling scheme tailored for the SPIPF and CLSIPF, respectively.

Remark 2. 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 $\min_{\mathbf{y} \in \mathcal{Y}(\hat{\mathbf{x}})} f(\mathbf{y})$. 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 $z^I(\mathbf{w}, \hat{\mathcal{Y}}^i) = \max_{k=1,\ldots,i}\{LB_k\}$.

Proposition 3 shows that the sequence of UB_i-values obtained is nonincreasing, and Proposition 4 states the finiteness and correctness of the proposed algorithm.

Proposition 3. *The upper bounds UB_i produced by Algorithm 1 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 $UB_k = z^I(\mathbf{w}, \hat{\mathcal{Y}}^k_{UB_k}) \geq z^I(\mathbf{w}, \hat{\mathcal{Y}}^{k+1}) = UB_{k+1}$.

Algorithm 1 (Solving bilevel interdiction problem $Q(\mathbf{w})$ via backward sampling)
Input: Problem $\mathscr{P}$ and a feasible defense $\mathbf{w} \in \mathscr{W}$
Output: An optimal solution to $\mathscr{Q}(\mathbf{w})$

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

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

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

Proof. The first time that an attack $\hat{\mathbf{x}}$ is a part of an optimal solution to $z^I(\mathbf{w}, \hat{\mathscr{Y}}^i)$ at line 6, the algorithm includes a corresponding optimal recourse response $\hat{\mathbf{y}}^*$ into $\hat{\mathscr{Y}}^{i+1}$. Suppose that $\hat{\mathbf{x}}$ is a part of an optimal solution to $z^I(\mathbf{w}, \hat{\mathscr{Y}}^k)$ for a second time at iteration $k > i$. Proposition 3 guarantees that $\hat{\mathbf{y}}^* \in \hat{\mathscr{Y}}^k$ 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 2 is met. Finite termination of the algorithm then follows from the finiteness of $\mathscr{X}$. □

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:

$$\max_{\mathbf{x}\in\mathscr{X}} \min \quad (\mathbf{c} + \mathbf{D}\mathbf{x})^\top \mathbf{y} \tag{5}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{y} = \mathbf{b} \tag{6}$$

$$\mathbf{y} \ge \mathbf{0}. \tag{7}$$

Since the recourse problem is a linear program, we reformulate (5)–(7) by considering its dual:

$$\max_{\mathbf{x}\in\mathscr{X}} \max \quad \mathbf{b}^\top \boldsymbol{\pi} \tag{8}$$

$$\text{s.t.} \quad \mathbf{A}^\top \boldsymbol{\pi} \le \mathbf{c} + \mathbf{D}\mathbf{x}. \tag{9}$$

Note that solving (5)–(7) with our sampling algorithm is the same as solving (8)–(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.

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* $\mathbf{w}^\intercal\hat{\mathbf{x}} \ge 1$ 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 5. *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 $\mathbf{w}^{*\intercal}\hat{\mathbf{x}} \ge 1$.*

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 $z^* = \max_{\mathbf{x}\in\mathcal{X}(\mathbf{w}^*)} \min_{\mathbf{y}\in\mathcal{Y}(\mathbf{x})} f(\mathbf{y}) \ge z^R(\hat{\mathbf{x}})$. 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 1 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 $\mathcal{X}(\mathbf{w}) \neq \varnothing$, $\forall\, \mathbf{w} \in \mathcal{W}$, by assumption.

Algorithm 2 presents the proposed approach. Let $\mathcal{C}$ be the set of covering constraints added to the fortification problem and $\mathcal{W}(\mathcal{C}) \equiv \{\mathbf{w} \in \mathcal{W} \mid \mathbf{w}$ satisfies all constraints in $\mathcal{C}\}$. The algorithm starts with $\mathcal{C} = \varnothing$ and a global upper bound $\bar{z} = \infty$. The main while-loop (line 4) is executed until the fortification problem becomes infeasible. The two main steps inside this while-loop are selecting a feasible defense $\hat{\mathbf{w}} \in \mathcal{W}(\mathcal{C})$ (line 5), and solving its associated interdiction problem $\mathcal{Q}(\hat{\mathbf{w}})$ with a variation of Algorithm 1 (lines 6–23). The inner while-loop (line 7) is executed until $LB_i = z^R(\hat{\mathbf{x}}) \ge \bar{z}$, for some $\hat{\mathbf{x}} \in \mathcal{X}(\hat{\mathbf{w}})$, indicating that $\hat{\mathbf{x}}$ is a critical attack. At this point, Algorithm 2 stops solving $\mathcal{Q}(\hat{\mathbf{w}})$ and adds a covering constraint to $\mathcal{C}$. Finally, lines 8–22 replicate Algorithm 1, except for updating the global upper bound $\bar{z}$ (line 13), adding a covering constraint to $\mathcal{C}$ if a critical attack is identified (lines 17–19), and updating the incumbent solution when an optimal solution to $\mathcal{Q}(\hat{\mathbf{w}})$ has an objective value equal to $\bar{z}$ (lines 20–22).

Algorithm 2 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 4.

The correctness of Algorithm 2 results directly from Propositions 1 and 5. Note that the upper bound $\bar{z}$ is nonincreasing throughout the execution of the algorithm. Proposition 5 states that each of the covering constraints is necessary to achieve an objective value less than $\bar{z}$. As a result, once $\mathcal{W}(\mathcal{C})$ becomes empty we conclude that $z^* \ge \bar{z}$. Since $\bar{z}$ is an upper bound, we also have that $z^* \le \bar{z}$, 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 $\bar{z} = z^*$.

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 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

Algorithm 2 (Backward sampling framework)
Input: Problem $\mathscr{P}$
Output: An optimal solution to $\mathscr{P}$

```
 1: Set the global upper bound $\bar{z} = \infty$ and covering constraints set $\mathscr{C} = \varnothing$    ▷ Initialization
 2: Select $\hat{\mathscr{Y}}^1 \subseteq \mathscr{Y}$ as a sampling of the third-stage solution space, and compute $f(\mathbf{y})$ for each solution $\mathbf{y} \in \hat{\mathscr{Y}}^1$
 3: Set counter $i = 0$
 4: while $\mathscr{W}(\mathscr{C}) \neq \varnothing$ do    ▷ Main while-loop
 5:     Select any $\hat{\mathbf{w}} \in \mathscr{W}(\mathscr{C})$
 6:     Initialize $UB_i = \infty$ and $LB_i = -\infty$
 7:     while $LB_i < \bar{z}$ do
 8:         Set $i = i + 1$
 9:         Solve $UB_i = z^I(\hat{\mathbf{w}}, \hat{\mathscr{Y}}^i) = \max_{\mathbf{x} \in \mathscr{X}(\hat{\mathbf{w}})} \min_{\mathbf{y} \in \hat{\mathscr{Y}}^i(\mathbf{x})} f(\mathbf{y})$ and obtain an optimal solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$
10:         Solve $LB_i = z^R(\hat{\mathbf{x}}) = \min_{\mathbf{y} \in \mathscr{Y}(\hat{\mathbf{x}})} f(\mathbf{y})$ and obtain an optimal solution $\hat{\mathbf{y}}^*$
11:         Set $\hat{\mathscr{Y}}^{i+1} = \hat{\mathscr{Y}}^i \cup \{\hat{\mathbf{y}}^*\}$
12:         if $UB_i < \bar{z}$ then
13:             Update global upper bound $\bar{z} \leftarrow UB_i$
14:             Remove from $\hat{\mathscr{Y}}^{i+1}$ all solutions having objective value greater than $UB_i$
15:             Select $\hat{\mathscr{Y}}' \subseteq \mathscr{Y}$ as a sampling of the third-stage solution space
16:             Add to $\hat{\mathscr{Y}}^{i+1}$ all new solutions in $\hat{\mathscr{Y}}' \cap \mathscr{Y}_{UB_i}$
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 $\mathscr{C}$
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 $(\bar{\mathbf{w}}, \bar{\mathbf{x}}, \bar{\mathbf{y}})$
```

is critical, and $\hat{\psi}$ is the corresponding covering constraint. Algorithm 3 formally states the accelerated backward sampling algorithm. If $\mathscr{W}(\mathscr{C} \cup \mathscr{C}^\psi) \neq \varnothing$, then line 6 selects a defense $\hat{\mathbf{w}} \in \mathscr{W}(\mathscr{C} \cup \mathscr{C}^\psi)$ and lines 7–22 explore problem $\mathscr{Q}(\hat{\mathbf{w}})$ as in Algorithm 2. 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 $\mathscr{Q}(\hat{\mathbf{w}})$. 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 $\mathscr{L}$, add the corresponding tentative covering constraint to $\mathscr{C}^\psi$, and stop the exploration of the current $\hat{\mathbf{w}}$. When $\mathscr{W}(\mathscr{C} \cup \mathscr{C}^\psi) = \varnothing$, if $\mathscr{C}^\psi \neq \varnothing$, then lines 30–39 reconsider the items stored in the waiting list. The first for-loop (lines 30–34) iterates over $\mathscr{L}$ and moves from $\mathscr{C}^\psi$ to $\mathscr{C}$ 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 (lines 35–39) iterates over the remaining items in $\mathscr{L}$ and resumes exploration for any $\mathbf{w}^k$ that is still in $\mathscr{W}(\mathscr{C})$, but with $\epsilon = 0$. Finally, line 40 discards the remaining constraints in $\mathscr{C}^\psi$, empties the waiting list, and returns to the main while-loop.

We further note that setting the value of the precision parameter ϵ to zero is equivalent to using no waiting list $\mathscr{L}$, which in turn converts Algorithm 3 to Algorithm 2.

3. Shortest Path Interdiction Problem with Fortification

A significant amount of research has been dedicated to the shortest path interdiction problem. However, few 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.

Algorithm 3 (Backward sampling framework with waiting list)
Input: Problem $\mathscr{P}$ and a threshold parameter $\epsilon > 0$
Output: An optimal solution to $\mathscr{P}$

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

3.1. Problem Statement

The SPIPF is formally defined on a directed graph $\mathscr{G} = (\mathscr{N}, \mathscr{A})$, where $\mathscr{N}$ is the set of nodes and $\mathscr{A} \subseteq \mathscr{N} \times \mathscr{N}$ is the set of arcs, s is the source node, and t is the destination node. For each arc $(i, j) \in \mathscr{A}$ 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 $\mathscr{W} \equiv \{\mathbf{w}\colon \mathbf{e}^\intercal \mathbf{w} \leq Q, \mathbf{w} \in \{0,1\}^{|\mathscr{A}|}\}$ enforces a cardinality constraint on the number of fortified arcs and ensures that the variables are binary. Similarly, let $\mathbf{x} \in \mathscr{X}(\mathbf{w})$ be the second-stage attack decision variables, where $\mathscr{X}(\mathbf{w}) \equiv \{\mathbf{x}\colon \mathbf{e}^\intercal \mathbf{x} \leq B, x_{ij} \leq 1 - w_{ij}\ \forall (i,j) \in \mathscr{A}, \mathbf{x} \in \{0,1\}^{|\mathscr{A}|}\}$ ensures that a maximum of B unprotected arcs are attacked, and forces the **x**-variables to be binary. Finally, let **y** be the third-stage arc-flow variables. The SPIPF can be formally stated as

$$\min_{\mathbf{w}\in\mathscr{W}} \max_{\mathbf{x}\in\mathscr{X}(\mathbf{w})} \min \sum_{(i,j)\in\mathscr{A}} (c_{ij}+d_{ij}x_{ij})y_{ij} \tag{10}$$

$$\text{s.t.} \sum_{\{j\,|\,(i,j)\in\mathscr{A}\}} y_{ij} - \sum_{\{j\,|\,(j,i)\in\mathscr{A}\}} y_{ji} = \begin{cases} 1, & \text{for } i=s \\ 0, & \text{for } i\in\mathscr{N}\backslash\{s,t\} \\ -1, & \text{for } i=t \end{cases} \tag{11}$$

$$y_{ij}\geq 0, \quad \forall\,(i,j)\in\mathscr{A}, \tag{12}$$

where in the objective function (10), the original cost of any arc is increased by d_{ij} when the arc is attacked (i.e., $x_{ij}=1$). Constraints (11) define the shortest path flow conservation constraints, and (12) restrict the **y**-variables to be nonnegative.

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 sections.

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 $\mathscr{G}$. 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 $\hat{\mathscr{Y}}$ that can use any arc (i,j). Let u_{ij} be the number of paths in $\hat{\mathscr{Y}}$ that use arc (i,j). When the search reaches node *t*, we add an *s*-*t* path to $\hat{\mathscr{Y}}$ 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 $|\hat{\mathscr{Y}}|_{\max}$ is reached or once a time limit is exceeded.

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

$$\max \; z \tag{13}$$

$$\text{s.t.} \; z\leq c(\mathscr{P}^k)+\sum_{(i,j)\in\mathscr{P}^k} d_{ij}x_{ij}, \quad \forall\,\mathscr{P}^k\in\hat{\mathscr{Y}}, \tag{14}$$

$$\mathbf{x}\in\mathscr{X}(\hat{\mathbf{w}}). \tag{15}$$

The objective function (13) maximizes *z*, which is constrained by (14) to be no more than the least cost path in $\hat{\mathscr{Y}}$, after considering delays caused by arc interdiction. Finally, constraints (15) ensure that we only consider feasible attacks in $\mathscr{X}(\hat{\mathbf{w}})$.

Observe that if our algorithm generates an attack $\hat{\mathbf{x}}\in\mathscr{X}(\hat{\mathbf{w}})$ 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 (13)–(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 (13)–(15). This ensures that any attack $\hat{\mathbf{x}}\in\mathscr{X}(\hat{\mathbf{w}})$ 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 $\mathscr{Q}(\hat{\mathbf{w}},\hat{\mathscr{Y}})$.

Furthermore, because the **x**-variables are binary-valued and $d_{ij}\geq 0$, $\forall\,(i,j)\in\mathscr{A}$, the addition of the objective target constraint allows us to revise (14) as follows, where $(\bullet)^+=\max\{0,\bullet\}$:

$$z\leq c(\mathscr{P}^k)+\sum_{(i,j)\in\mathscr{P}^k}\min\{d_{ij},(\bar{z}+\delta-c(\mathscr{P}^k))^+\}x_{ij}, \quad \forall\,\mathscr{P}^k\in\hat{\mathscr{Y}}. \tag{16}$$

Constraints (16) are at least as tight as (14). (Note that (16) corresponding to some $\mathscr{P}^k$ may persist in our interdiction model for a few iterations after $\bar{z}+\delta\leq c(\mathscr{P}^k)$. We therefore require the coefficients of the **x**-variables to be nonnegative to ensure the validity of (16).)

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.

4.1. Problem Statement

Formally, we define the CLSIPF as the problem of finding a subset of time periods to fortify 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 $\mathcal{W} \equiv \{\mathbf{w}\colon \mathbf{e}^\intercal \mathbf{w} \le Q, \mathbf{w} \in \{0,1\}^{|\mathcal{T}|}\}$ establishes the defender's budget and ensures that the fortification variables are binary, and $\mathcal{X}(\mathbf{w}) \equiv \{\mathbf{x}\colon \mathbf{e}^\intercal \mathbf{x} \le B, x_t \le 1 - w_t \ \forall t \in \mathcal{T}, \mathbf{x} \in \{0,1\}^{|\mathcal{T}|}\}$ 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

$$\min_{\mathbf{w}\in\mathcal{W}} \max_{\mathbf{x}\in\mathcal{X}(\mathbf{w})} \min \sum_{t\in\mathcal{T}} c_t y_t + f_t v_t + h_t I_t + q_t s_t \tag{17}$$

$$\text{s.t.} \quad I_t = I_{t-1} + y_t + s_t - d_t, \quad \forall t \in \mathcal{T}, \tag{18}$$

$$y_t \le C_t v_t, \quad \forall t \in \mathcal{T}, \tag{19}$$

$$v_t \le 1 - x_t, \quad \forall t \in \mathcal{T}, \tag{20}$$

$$y_t, I_t, s_t \ge 0, \quad \forall t \in \mathcal{T}, \tag{21}$$

$$v_t \in \{0,1\}, \quad \forall t \in \mathcal{T}. \tag{22}$$

The objective function (17) minimizes the total cost after interdiction. Constraints (18) are inventory constraints, constraints (19) enforce production capacity limits, and constraints (20) forbid production on interdicted time periods. Constraints (21) and (22) place bounds and binary restrictions on the decision variables.

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.

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 $\mathbf{x}^r$:

$$\min_{(\mathbf{y},\mathbf{v},\mathbf{I},\mathbf{s})\in\mathcal{Y}(\mathbf{x}^r)} \sum_{t\in\mathcal{T}} [c_t y_t + f_t v_t + h_t I_t + q_t s_t], \tag{23}$$

where $\mathcal{Y}(\mathbf{x}^r)$ is the third-stage feasible region defined by inserting $\mathbf{x}^r$ in constraints (18)–(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 $c(\mathcal{S}^*) \le \bar{z}$, 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.

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 (13)–(15):

$$\max \quad z \tag{24}$$

$$\text{s.t.} \quad z \le c(\mathcal{S}^k) + \sum_{t\in\mathcal{T}(\mathcal{S}^k)} M_t^k x_t, \quad \forall \mathcal{S}^k \in \hat{\mathcal{Y}}, \tag{25}$$

$$\mathbf{x} \in \mathcal{X}(\hat{\mathbf{w}}). \tag{26}$$

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 $j \ge t$. 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 $q_j a_{tj}^k$ for each $j \ge t$. Accordingly, we define the cost penalty for any production plan $\mathcal{S}^k$ at time period t as

$$M_t^k = \left(\sum_{j\in\mathcal{T}:\, j\ge t} q_j a_{tj}^k\right) - f_t - c_t y_t^k - \left(\sum_{j\in\mathcal{T}:\, j>t}\ \sum_{l=t}^{j-1} h_l a_{tj}^k\right). \tag{27}$$

Proposition 6 shows that (25) remains valid when M_t^k is defined as in (27).

Proposition 6. *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* $c(\mathcal{S}^k) + \sum_{t\in\mathcal{T}(\mathcal{S}^k)} M_t^k x_t \ge c(\mathcal{S}^*)$, *where the M-values are defined in* (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 previously described. 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 $c(\mathcal{S}'_k) \ge c(\mathcal{S}^*)$. Noting that $c(\mathcal{S}'_k) = c(\mathcal{S}^k) + \sum_{t\in\mathcal{T}(\mathcal{S}^k)} M_t^k x_t$, 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 $z \le \bar{z} + \delta$ to model (24)–(26), which allows us to tighten (25) as follows:

$$z \le c(\mathcal{S}^k) + \sum_{t\in\mathcal{T}(\mathcal{S}^k)} \min\{M_t^k, (\bar{z} + \delta - c(\mathcal{S}^k))^+\} x_t, \quad \forall \mathcal{S}^k \in \hat{\mathcal{Y}}. \tag{28}$$

Remark 3. 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_t^k-parameters in (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_t^k to be negative. Constraint (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 $M_t^k < 0$.

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 (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.

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 5.1 and on large-scale real road networks in Section 5.2. In Section 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 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 RAM allocated to the Java Virtual Machine memory heap on Windows 8. We impose a time limit of four hours (14,400 s) and solve all mathematical optimization problems using Gurobi 5.6. All instances and source code used in this section are available in the online supplement (available as supplemental material at https://doi.org/10.1287/ijoc.2016.0721).

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 $2{,}160 = 360 \times 6$ experiments. After tuning the algorithm parameters, we set the maximum sample size to 100, the sampling time limit $|\hat{\mathcal{Y}}|_{\max}$ to one second, the arc-usage upper limit to 20, threshold ϵ to 0.1, and δ to 1 (see (16)).

Tables 2 and 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 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 4,000 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 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 3,000 nodes and 17,000 arcs. Table 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 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

Table 2 Computational Time in CPU Seconds for Solving the SPIPF Over Medium-Sized Grid Networks

					Cost-delay configuration (c, d)											
					(10, 5)		(10, 10)		(10, 20)		(100, 50)		(100, 100)		(100, 200)	
Instance	Nodes	Arcs	Q	B	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	1,826	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	4,236	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 Computational Time in CPU Seconds for Solving the SPIPF Over Large-Sized Grid Networks

					Cost-delay configuration (c, d)											
					(10, 5)		(10, 10)		(10, 20)		(100, 50)		(100, 100)		(100, 200)	
Instance	Nodes	Arcs	Q	B	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
40×40	1,602	7,646	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	1,786.3	194.0	1,103.3	39.0	94.8	75.6	217.7	64.0	185.8
			7	5	23.4	61.7	211.4	1,400.6	330.2	2,153.0	56.9	101.2	181.4	1,155.0	111.0	497.8
50×50	2,502	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	1,979.0	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	1,049.9
			7	5	44.0	208.1	210.8	1,355.7	142.1	847.1	344.8	1,702.7	527.7	3,614.9	375.1	1,385.0
60×60	3,602	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	1,263.4	236.3	1,070.4	155.9	578.8	1,479.0	8,103.9	1,587.3	11,052.4
			7	5	81.2	273.9	346.0	1,477.3	491.8	3,382.5	317.1	1,056.6	1,575.9	8,079.2	1,733.4	11,177.9

acceleration strategies proposed by Cappanera and Scaparra (2011) outperforms KSPI.

Table 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 "No. solved" gives the number of instances solved within the four-hour time limit.

Table 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 $(Q, B) = (4, 5)$.

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 9,559 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 5 shows the results for these experiments.

Table 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

Table 4 Comparing the Backward Sampling Algorithm to the State-of-the-Art Algorithm for SPIPF

					Sampling			SPI		
Instance	Nodes	Arcs	Q	B	Avg	Max	No. solved	Avg	Max	No. solved
10×10	102	416	3	3	0.1	0.3	60	1.9	4.6	60
			5	4	0.3	0.8	60	30.9	162.8	60
			4	5	0.7	2.9	60	67.5	284.2	60
20×20	402	1,826	3	3	0.6	3.3	60	26.7	305.1	60
			5	4	2.2	12.8	60	1,128.4	7,723.4	60
			4	5	5.8	33.4	60	2,495.2	>14,400	56
30×30	902	4,236	3	3	1.7	7.5	60	766.9	12,728.8	60
			5	4	7.2	36.9	60	4,857.3	>14,400	45
			4	5	18.6	94.8	60	4,256.8	>14,400	26

Table 5 Computational Time in CPU Seconds for Solving the SPIPF Over Road Networks

Instance	Nodes	Arcs	Q	B	Avg	Max	No. solved
DC	9,559	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	2,829.5	9
			7	5	450.1	1,906.4	9
RI	53,658	192,084	3	3	284.5	756.0	9
			4	3	295.6	817.3	9
			3	4	800.7	4,925.1	9
			5	4	946.2	5,974.9	9
			4	5	560.1	>14,400	8
			7	5	754.0	>14,400	8
NJ	330,386	1,202,458	3	3	6,743.8	10,551.9	9
			4	3	6,345.8	>14,400	8
			3	4	6,526.8	>14,400	8
			5	4	6,964.3	>14,400	8
			4	5	7,354.6	>14,400	8
			7	5	8,452.6	>14,400	8

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.

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 10 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 6 and 7.

Table 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 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 8). For each ϵ-value considered we report the number of defense plans evaluated (No. of $\hat{\mathbf{w}}$ evaluated), the number of defense plans added to the waiting list $\mathscr{L}$ (No. of $\hat{\mathbf{w}}$ interrupted), the number of restricted interdiction problems solved (No. of $\mathscr{Q}(\hat{\mathbf{w}}, \hat{\mathscr{Y}})$ 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 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 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,

Table 6 Measuring the Effect of Increasing B on CPU Time Over a Subset of 30×30 Grid Networks

Q	B	Avg	Max	No. 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	1,208.5	4,878.1	10
4	9	4,901.8	>14,400	9
4	10	4,750.8	>14,400	2

Table 7 Measuring the Effect of Increasing Q on CPU Time Over a Subset of 30×30 Grid Networks

Q	B	Avg	Max	No. 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 8 **Profiling the Algorithm on a Subset of Hard Instances**

Instance	(c, d)	Metric	$\epsilon = 0$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.15$	$\epsilon = 0.2$
		No. of $\hat{\mathbf{w}}$ evaluated	59	91	95	93	112
		No. of $\hat{\mathbf{w}}$ interrupted	0	77	78	73	87
		No. of $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ solved	1,745	1,505	1,383	1,376	1,532
40×40-0	(10, 20)	Time$_F$ (s)	2.1	3.2	3.2	3.0	3.7
		Time$_I$ (s)	4,256.8	3,493.3	2,083.2	2,773.7	2,790.1
		Total time (s)	4,320.9	3,557.5	2,146.5	2,837.7	2,849.3
		No. of $\hat{\mathbf{w}}$ evaluated	68	107	122	76	120
		No. of $\hat{\mathbf{w}}$ interrupted	0	87	97	57	99
		No. of $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ solved	2,218	2,257	1,910	2,083	2,045
60×60-5	(10, 20)	Time$_F$ (s)	6.7	9.3	9.6	6.1	9.2
		Time$_I$ (s)	3,474.7	3,769.8	2,942.6	2,923.0	3,144.9
		Total time (s)	3,531.0	3,828.1	2,999.5	2,979.0	3,204.9
		No. of $\hat{\mathbf{w}}$ evaluated	81	120	142	148	160
		No. of $\hat{\mathbf{w}}$ interrupted	0	101	123	127	134
		No. of $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ solved	1,270	1,190	1,226	1,201	1,239
60×60-1	(100, 100)	Time$_F$ (s)	8.7	11.7	12.5	13.3	15.3
		Time$_I$ (s)	6,977.2	5,130.5	8,076.5	7,513.5	6,693.4
		Total time (s)	7,074.0	5,229.4	8,162.9	7,595.5	6,773.6
		No. of $\hat{\mathbf{w}}$ evaluated	55	122	128	135	96
		No. of $\hat{\mathbf{w}}$ interrupted	0	93	99	111	82
		No. of $\mathcal{Q}(\hat{\mathbf{w}}, \hat{\mathcal{Y}})$ solved	3,043	2,176	2,110	2,296	2,394
60×60-6	(100, 200)	Time$_F$ (s)	5.2	10.5	11.4	12.0	7.7
		Time$_I$ (s)	10,902.3	3,846.2	4,303.6	5,007.5	4,973.1
		Total time (s)	10,989.7	3,931.1	4,388.2	5,109.1	5,068.6

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 $\epsilon = 0$.

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 (28)). Because each iteration of our sampling scheme in Section 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 (SP) problem using a standard dynamic programming approach. Table 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 "No. sol" gives the number of instances solved within the four-hour time limit.

Table 9 shows that SP solves all instances having $|\mathcal{T}| \leq 20$, 38 of 60 instances having $|\mathcal{T}| = 30$, and 20 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 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

Table 9 **Computational Time in CPU Seconds for Solving the CLSIPF Over Randomly Generated Problem Instances**

			$\lvert\mathcal{T}\rvert = 10$			$\lvert\mathcal{T}\rvert = 20$			$\lvert\mathcal{T}\rvert = 30$			$\lvert\mathcal{T}\rvert = 40$		
Algorithm	Q	B	Avg	Max	No. sol	Avg	Max	No. sol	Avg	Max	No. sol	Avg	Max	No. sol
SP	3	2	97.7	376.7	10	1,180.0	2,507.6	10	1,827.2	3,929.1	10	4,464.4	6,587.7	10
	5	2	111.5	392.1	10	1,186.8	2,467.1	10	1,907.4	4,077.3	10	4,635.5	6,560.6	10
	3	3	85.9	202.9	10	1,978.5	3,361.6	10	9,610.3	14,051.6	9	—	—	0
	5	3	131.5	459.8	10	1,997.7	3,440.6	10	10,070.3	14,237.3	9	—	—	0
	3	4	110.0	197.8	10	5,170.7	8,529.8	10	—	—	0	—	—	0
	5	4	176.5	499.4	10	5,284.2	8,238.6	10	—	—	0	—	—	0
MIP	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
	3	3	2.8	5.9	10	26.4	72.8	10	153.3	603.3	10	702.4	1,667.7	10
	5	3	2.2	4.4	10	29.7	76.7	10	169.3	561.1	10	994.6	2,032.9	10
	3	4	2.9	4.8	10	115.3	265.6	10	1,711.1	>14,400	9	4,933.4	>14,400	5
	5	4	2.2	3.4	10	109.6	177.3	10	2,622.2	12,182.0	10	10,086.0	>14,400	4

in the attacker's budget has a dramatic impact on the computational time. For example, when $\lvert\mathcal{T}\rvert = 30$, 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.

6. Conclusions

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 (e.g., on interdiction and fortification for the traveling salesman problem) 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. Finally, we note that the more general class of bilevel optimization problems (in which the leader and follower do not play a zero-sum game) is also notoriously difficult to optimize, particularly when the follower's problem is nonconvex. Future research may investigate how the foregoing framework could be applied to handle these types of problems.

Supplemental Material

Supplemental material to this paper is available at https://doi.org/10.1287/ijoc.2016.0721.

Acknowledgments

The authors thank Dr. Paola Scaparra at University of Kent (UK) and Dr. Paola Cappanera at University of Florence (Italy) for their generosity in sharing their code for solving the SPIPF. The authors thank the referees and editorial staff for their insightful comments, which led to an improved version of this paper. Dr. Smith gratefully acknowledges the support of the Defense Threat Reduction Agency [Grant HDTRA-10-01-0050], the Air Force Office of Scientific Research [Grant FA9550-12-1-0353], and the Office of Naval Research [Grant N000141310036].

References

Bard JF, Moore JT (1992) An algorithm for the discrete bilevel programming problem. *Naval Res. Logist.* 39(3):419–435.

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

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

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

Bitran GR, Yanasse HH (1982) Computational complexity of the capacitated lot size problem. *Management Sci.* 28(10):1174–1186.

Brahimi N, Dauzere-Peres S, Najid NM, Nordli A (2006) Single item lot sizing problems. *Eur. J. Oper. Res.* 168(1):1–16.

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

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

Brown GG, Carlyle WM, Salmerón J, Wood RK (2005b) Analyzing the vulnerability of critical infrastructure to attack and planning defenses. Greenberg HJ, Smith JC, eds. *Tutorials in Operations Research: Emerging Theory, Methods, and Applications* (INFORMS, Hanover, MD), 102–123.

Brown GG, Carlyle WM, Harney RC, Skroch EM, Wood RK (2009) Interdicting a nuclear-weapons project. *Oper. Res.* 57(4): 866–877.

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

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

Church RL, Scaparra MP (2007) The *r*-interdiction median problem with fortification. *Geographical Anal.* 39(2):129–146.

Church RL, Scaparra MP, Middleton RS (2004) Identifying critical infrastructure: The median and covering facility interdiction problems. *Ann. Assoc. Amer. Geographers* 94(3):491–502.

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

Cormican KJ, Morton DP, Wood RK (1998) Stochastic network interdiction. *Oper. Res.* 46(2):184–197.

Dempe S (2002) *Foundations of Bilevel Programming* (Kluwer Academic Publishers, Boston).

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

Eppen GD, Martin RK (1987) Solving multi-item capacitated lot-sizing problems using variable redefinition. *Oper. Res.* 35(6): 832–848.

Florian M, Lenstra JK, Rinnooy Kan AHG (1980) Deterministic production planning: Algorithms and complexity. *Management Sci.* 26(7):669–679.

Fulkerson DR, Harding GC (1977) Maximizing minimum source-sink path subject to a budget constraint. *Math. Programming* 13(1):116–118.

Golden B (1978) A problem in network interdiction. *Naval Res. Logist. Quart.* 25(4):711–713.

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

Held H, Hemmecke R, Woodruff DL (2005) A decomposition algorithm applied to planning the interdiction of stochastic networks. *Naval Res. Logist.* 52(4):321–328.

Hooker JN, Ottosson G (2003) Logic-based Benders decomposition. *Math. Programming* 96(1):33–60.

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

Karimi B, Fatemi Ghomi SMT, Wilson JM (2003) The capacitated lot sizing problem: A review of models and algorithms. *Omega* 31(5):365–378.

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

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

Lozano L, Medaglia AL (2013) On an exact method for the constrained shortest path problem. *Comput. Oper. Res.* 40(1): 378–384.

Moore JT, Bard JF (1990) The mixed integer linear bilevel programming problem. *Oper. Res.* 38(5):911–921.

Morton DP, Pan F, Saeger KJ (2007) Models for nuclear smuggling interdiction. *IIE Trans.* 39(1):3–14.

Pan F, Charlton W, Morton DP (2003) Interdicting smuggled nuclear material. Woodruff DL, ed. *Network Interdiction and Stochastic Integer Programming* (Kluwer Academic Publishers, Boston), 1–20.

Peterson R, Silver EA (1979) *Decision Systems for Inventory Management and Production Planning* (Wiley, New York).

Prince M, Smith JC, Geunes J (2013) A three-stage procurement optimization problem under uncertainty. *Naval Res. Logist.* 60(1):395–412.

Raith A, Ehrgott M (2009) A comparison of solution strategies for biobjective shortest path problems. *Comput. Oper. Res.* 36(4): 1299–1331.

Royset JO, Wood RK (2007) Solving the bi-objective maximum-flow network-interdiction problem. *INFORMS J. Comput.* 19(2): 175–184.

Salmerón J, Wood K, Baldick R (2004) Analysis of electric grid security under terrorist threat. *IEEE Trans. Power Systems* 19(2): 905–912.

Salmerón J, Wood K, Baldick R (2009) Worst-case interdiction analysis of large-scale electric power grids. *IEEE Trans. Power Systems* 24(1):96–104.

Scaparra MP, Church RL (2008a) A bilevel mixed-integer program for critical infrastructure protection planning. *Comput. Oper. Res.* 35(6):1905–1923.

Scaparra MP, Church RL (2008b) An exact solution approach for the interdiction median problem with fortification. *Eur. J. Oper. Res.* 189(1):76–92.

Sen S, Sherali HD (2006) Decomposition with branch-and-cut approaches for two-stage stochastic mixed-integer programming. *Math. Programming* 106(2):203–223.

Smith JC (2010) Basic interdiction models. Cochran J, ed. *Wiley Encyclopedia of Operations Research and Management Science* (Wiley, Hoboken, NJ), 323–330.

Smith JC, Lim C (2008) Algorithms for network interdiction and fortification games. Migdalas A, Pardalos PM, Pitsoulis L, Chinchuluun A, eds. *Pareto Optimality, Game Theory and Equilibria*, Nonconvex Optimization and Its Applications Series (Springer, New York), 609–644.

Smith JC, Lim C, Sudargho F (2007) Survivable network design under optimal and heuristic interdiction scenarios. *J. Global Optim.* 38(2):181–199.

Tang Y, Richard J-PP, Smith JC (2016) A class of algorithms for mixed-integer bilevel min–max optimization. *J. Global Optim.* 66(2):225–262.

Vicente L, Savard G, Judice J (1996) Discrete linear bilevel programming problem. *J. Optim. Theory Appl.* 89(3):597–614.

Washburn R, Wood K (1995) Two-person zero-sum games for network interdiction. *Oper. Res.* 43(2):243–251.

Wollmer R (1964) Removing arcs from a network. *Oper. Res.* 12(6):934–940.

Wood RK (1993) Deterministic network interdiction. *Math. Comput. Model.* 17(2):1–18.