Les Cahiers du GERAD ISSN: 0711–2440

A Progressive approximation approach for the exact solution of sparse large-scale binary interdiction games

C. Contardo,
J. A. Sefair

G–2019–49

July 2019

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2019
– Bibliothèque et Archives Canada, 2019

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2019
– Library and Archives Canada, 2019

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

A Progressive approximation approach for the exact solution of sparse large-scale binary interdiction games

Claudio Contardo [a,b]
Jorge A. Sefair [c]

[a] *GERAD, Montréal (Québec), Canada, H3T 2A7*

[b] *École des Sciences de la Gestion, Université du Québec à Montréal, Montréal (Québec) Canada, H2X 3X2*

[c] *School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281*

claudio.contardo@gerad.ca
jorge.sefair@asu.edu

July 2019
Les Cahiers du GERAD
G–2019–49

Abstract: We present a progressive approximation algorithm for the exact solution of several classes of interdiction games in which two non-cooperative players (namely an *attacker* and a *follower*) interact sequentially. The follower must solve an optimization problem that has been previously perturbed by means of a series of attacking actions led by the attacker. These attacking actions aim at augmenting the cost of the decision variables of the follower's optimization problem. The objective, from the attacker's viewpoint, is that of choosing an attacking strategy that reduces as much as possible the quality of the optimal solution attainable by the follower. The progressive approximation mechanism consists in the iterative solution of a relaxed interdiction problem —in which the follower actions are restricted to a subset of the whole solution space—, and of a pricing subproblem invoked with the objective of proving the optimality of the attacking strategy. This scheme is especially useful when the optimal solutions to the follower's subproblem intersect with the decision space of the attacker only in a small number of decision variables. Under this assumption, the progressive approximation method can scale and solve interdiction games otherwise untractable for classical methods. We illustrate the efficiency of this framework on shortest path, 0-1 knapsack and facility location interdiction games.

Acknowledgments: We thank Leonardo Lozano for kindly giving us access to his source code, which allowed us to assess the efficiency of our framework on shortest path interdiction games.

1 Introduction

This article studies a sequential two-stage Stackelberg game [92] in which two players, *follower* and *attacker*, interact as follows. The follower aims at solving an optimization problem $P : \min\{\mathbf{c}^T\mathbf{x}+\mathbf{g}^T\mathbf{w} : \mathbf{A}\mathbf{x} + \mathbf{G}\mathbf{w} \leq \mathbf{b}, \mathbf{x} \in \{0,1\}^n, \mathbf{w} \in \mathcal{W}\}$, where $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{g} \in \mathbb{R}^p$, $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{G} \in \mathbb{R}^{m\times p}$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathcal{W} \subseteq \mathbb{R}^p$ are the problem parameters. Having complete knowledge of the follower problem, the *attacker* seeks to deteriorate the objective function value of P by increasing the cost of a subset of the follower's binary decision variables. To represent these decisions, we use binary decision vector $\mathbf{y} = (y_i)_{i=1}^n$ and parameter $(d_i)_{i=1}^n \in \mathbb{R}_+$ such that y_i equals 1 if the attacker decides to increase the cost of variable x_i from c_i to $c_i + d_i$ in P, and equals 0 otherwise. As a result, the follower problem subject to an attack $\mathbf{y} \in \{0,1\}^n$ can be written as $P(\mathbf{y}) : z(\mathbf{y}) = \min\{\sum_{i=1}^n (c_i + d_i y_i)x_i + \mathbf{g}^T\mathbf{w} : \mathbf{A}\mathbf{x} + \mathbf{G}\mathbf{w} \leq \mathbf{b}, \mathbf{x} \in \{0,1\}^n, \mathbf{w} \in \mathcal{W}\}$, where $z(\mathbf{y})$ is the optimal value attained given the attack vector $\mathbf{y}$. For realistic purposes, we assume that the attacker is constrained by a single resource of which $\Delta > 0$ units are available. Moreover, attacking the follower's decision x_i requires $\beta_i \geq 0$ units of resource such that $\sum_{i=1}^n \beta_i \gg \Delta$, implying that the attacker can only attack a small fraction of all follower's decisions. Under these conditions, the attacker's goal is to find a strategy that causes the most detrimental impact on the follower's optimization problem, which is achieved by solving the problem $Q : \max\{z(\mathbf{y}) : \boldsymbol{\beta}^T\mathbf{y} \leq \Delta, \mathbf{y} \in \{0,1\}^n\}$.

Problem Q belongs to the category of interdiction and bilevel games, which arise in a vast number of applications including logistics and transportation [53, 17, 18, 45], military operations [54, 47, 70], conservation planning [2, 83], critical infrastructure analysis [19, 77], and design of reliable service networks [95], among others. Regardless of the application, realistic interdiction problems are typically hard to solve and large-scale in nature, motivating the need for scalable optimization algorithms. Because interdiction models inherently reveal the worst-case value for the follower's problem, the development of effective exact algorithms able to prove the optimality of candidate attacking strategies is critical. Admitting an attacker's suboptimal solution may provide biased and incomplete information regarding the potentially devastating impact of the attacker's actions on the follower's problem.

In this paper, we propose a solution approach that builds upon ideas used in previous relaxation-based methods for other classes of problems such as the minimax diameter clustering problem [7],the vertex p-center problem [24, 31] and the discrete p-dispersion problem [30]. Moreover, we integrate recent findings in structural decomposition schemes for sparse integer programs with a subjacent network structure [64]. As a result, our algorithm can handle very large-scale problems under the assumption that the follower's optimal solution under any feasible attacking strategy is sparse. This is true in some combinatorial problems, where the number of nonzero decision variables in an optimal solution is a small with respect to the total number of decision variables. We provide computational evidence of the effectiveness of our algorithm by solving the interdiction version of several combinatorial problems that can be modeled as problem Q, including shortest path, 0-1 knapsack, and facility location.

The remainder of this article is structured as follows. In Section 2, we put our contribution in context by providing a detailed literature review on interdiction games, bilevel optimization, and their solution methods. In Section 3, we present our proposed solution approach. In Section 4, we discuss some acceleration strategies to our scheme, including super-valid inequalities, a metaheuristic to aproximate the solution at intermediate stages of the solution procedure, and a logarithmic-time search method for the exact solution of the subproblems. In Section 5, we present computational evidence of the effectiveness of our method for several classes of interdiction games. Section 6 concludes the article and provides insights for future research.

2 Literature review

The use of interdiction problems in different fields has dramatically grown in the last two decades [38, 85, 87]. Von Stackelberg [92] established the foundations of these types of multi-level optimization problems, after which many variants have been proposed. In their simplest version, two players with complete information of the system interact sequentially: an attacker seeks to deteriorate the quality of the underlying system and a follower seeks to optimize the operation of the resulting (perturbed) system. As a result of this interaction, interdiction games are usually much harder to solve than their single-level counterparts. For example, network flow problems are solvable in polynomial time but their associated two-level interdiction problems are often NP-hard [12, 26, 51, 94]. Moreover, when the associated single-level problems are already NP-hard, their interdiction counterparts are at least as hard to solve [22, 28]. Because of such complexity, the development of scalable algorithms for multi-level interdiction games has been the subject of substantial efforts. While some of the latest advances concern general bilevel programs, tailored algorithms have also been introduced to solve specific variants of interdiction games.

Network interdiction models are the most studied variant of interdiction games. These models can be used to optimize the design and operation of flow-based real systems subject to disruptions, having applications in telecommunication networks, power systems, and supply chains, among others. Since the early works of Wollmer [93], McMasters and Mustin [65], and Ghare et al. [46] on minimizing the maximum capacity of a network, many approaches have been proposed for the solution of two-stage network interdiction problems. A common element throughout these works is the use of linear programming duality to combine both follower and attacker problems into a single-level formulation [43, 48]. Depending on where the attacker's variables are in the follower problem (e.g., right-hand side), this *dualization* approach may also require a linearization or a penalty reformulation [94, 56, 69] so that the resulting problem can be solved using conventional mixed-integer programming techniques. Some examples of this *dualize-reformulate* scheme include shortest-path interdiction games with asymmetric network information [11], discrete and continuous multicommodity flows interdiction problems [56], and maximum flow interdiction problems to disrupt illegal drug networks [63], among others. To solve large-scale shortest-path interdiction problems, Israeli and Wood [51] use Benders cuts and valid and supervalid inequalities to strengthen the resulting (dualized) single-level formulation. Other approaches to solve variants of network interdiction problems include Lagrangian relaxation and cut enumeration [74], sequential approximation algorithms [32, 88], valid inequalities [69, 8], policy design [16], branch-and-price [49], and dynamic programming [81, 82], among others. Tailored solution techniques have been developed for the interdiction of other network problems such as maximum flow [88, 8, 3, 72], minimum spanning tree [42, 57, 12, 98], assignment [82], and matching [97]. Bertsimas et al. [14], Altner et al. [8], and Chestnut and Zenklusen [26] provide approximation bounds for some network interdiction variants.

The study of interdiction games has also been extended to other applications of optimization. In competitive marketing, firms need to decide which products to launch to maximize their profit while considering the competitors' reaction. Using an interdiction scheme, Smith et al. [86] propose a model to maximize revenue under the worst-case competitor's predatory actions. The problem is solved using a dualize-reformulate approach enhanced with problem-specific cutting planes. In the same area, DeNegre [36] and Caprara et al. [22] study a knapsack interdiction problem in which two firms allocate resources to maximize profits across multiple market segments. One firm has more market power than the other (e.g., attacker) so it decides which market segments to capture, leaving the remaining ones for the second firm to choose from (e.g., follower). To solve this problem, Caprara et al. [22] present a tailored iterative algorithm that is based on the ideas of DeNegre [36], but includes a pre-processing stage to compute good initial bounds and a cutting-plane approach that iteratively adds strong cuts to the problem. For a 0-1 knapsack interdiction game in which items selected by one agent are not available for the other, Croce and Scatamacchia [33] introduce a tailored algorithm based on a new class of benders-like inequalities that provide lower bounds on the objective function given a set of possible actions of the agent selecting items first. In the area of infrastructure resiliency, the problem of identifying critical assets is often modeled as a location interdiction problem in which facilities and their services are subject to unexpected events represented by the attacker actions (e.g., capacity reduction, site unavailability). The goal of these problems is to use a rational and optimal attacker to unveil infrastructure vulnerabilities that, if occurring, cause the worst-case deterioration in the system's performance. Examples of interdiction games in infrastructure applications include variants of the p-median, maximal covering, and capacitated facility location problems [28, 58, 13]. Other applications of interdiction games include the delay of a nuclear weapons project [20], disruptions in vehicle routing [75], selection of conservation areas under worst-case environmental disturbances [83, 2, 71], and design of resilient power systems [77], among others.

Two-stage interdiction models are useful to identify critical system components and to assess the impact of adversarial actions. However, they provide little information regarding which components should be optimally protected if the follower had the opportunity to shield some of them. This additional feature is commonly known as *fortification* and has been subject of scientific interest for a variety of problems including infrastructure location [6, 27, 45, 55, 54], travel salesperson [62], shortest-path [76], and inter-modal transportation planning [78], among others. Regardless of the application, the common goal of these problems is to reduce the system's vulnerability to attacks by protecting a subset of system components that become harder to disrupt or inaccessible to the attacker. Interdiction games with fortification are typically modeled as three-level optimization programs, extending the modeling of the simpler—two-level—interdiction games. Other multi-stage interdiction games include several rounds of interactions between attacker and follower, where each agent adapts its strategy based on the previous decisions of the other [81, 82]. Existing solution techniques for solving multi-stage interdiction problems include implicit enumeration algorithms [45, 79], hybrid approaches combining heuristic and exact methods [100, 80], dynamic programming [81, 82], and heuristic approaches [54, 5], among others. Smith and Song [84] present a comprehensive review of existing models and mathematical programming solution techniques for interdiction problems.

Bilevel programs are a generalization of interdiction games and have been studied extensively in the literature. Although they are a class of Stackelberg games, in bilevel programs the attacker and follower may not be hostile, i.e., one agent is not necessarily trying to deteriorate the optimal objective function value of the other. Instead, agents may optimize their own functions and indirectly influence the operation of other agents given the order in which they play. For this reason, in bilevel problems the attacker is typically called *leader*, indicating that it plays first in the game. As an example, Labbé et al. [53] study a toll-setting problem on a transportation network in which an operator (i.e., leader) locates tolls in some roads to maximize total revenue. Road users (i.e., follower) optimize their travel along the network based on a generalized cost function that combines travel time and toll cost. Variants of this problem are also studied in Brotcorne et al. [17, 18]. Other applications of bilevel programs include detection of smuggler's activities Goldberg [47], competitive facility location under disruptions [4], bioengineering [21], traffic planning [66], and image processing [52, 99], among others. From a methodological perspective, bilevel programs have been solved using hybrid methods combining heuristic and exact approaches [99], branch-and-bound and branch-and-cut algorithms [68, 37, 89, 41, 40], extreme point enumeration [91], binary-search based approaches [90, 10], problem reformulations using optimality conditions of the follower's problem to produce a single-stage equivalent problem [34, 35], and problem reformulations based on projections of related problems [23], among others. Some approximation algorithms are available for bilevel knapsack problems [25], Colson et al. [29], Bard [9], and Migdalas et al. [67] present comprehensive surveys on bilevel (and multilevel) models, algorithms, and applications.

Because of their applicability, the literature on models and algorithms for the solution of bilevel and interdiction problems is increasing rapidly. However, general and scalable algorithms are still elusive and in most cases algorithms exploit particular structures of the underlying optimization problems. Of special interest for our work is the framework proposed by Lozano and Smith [60]. In this paper, authors introduced a general-purpose backward sampling algorithm that limits the follower actions and solves the restricted interdiction problems by means of a cut generation scheme. With respect to advances in the solution of general bilevel programs, Fischetti et al. [41] introduce intersection cuts for general bilevel programs. These valid inequalities along with a new class of valid inequalities are embedded within an exact solver developed by the same authors [40], which outperforms the state-of-the-art methods for general bilevel programs by a large margin. To the best of our knowledge, this algorithm is now the state-of-the-art solver for bilevel programming. Recently, Yue et al. [96] proposed a progressive relaxation scheme for the solution of general bilevel programs. Their computational experience indicates that, when the solutions to the single-level problem have sparse support, the decomposition scheme converges quickly and the solution of the resulting master problems becomes easy. Lozano and Smith [61] introduced a value-function based exact solver for a class of bilevel mixed-integer problem in which the upper-level variables are assumed integer. They exploit a single-level reformulation of the problem that is solved by means of a cut generation scheme in an iterative fashion.

Progressive approximation methods are not totally novel in the scientific literature although only scarcely used to solve a handful of problems. They rely on the existence of a single-level mathematical formulation satisfying two conditions: 1) Only a few constraints are binding in the optimal solutions; and 2) The relaxed model can further be tightened by removing a large number of variables and still yield an equivalent relaxed model. Implementations of progressive approximation methods can be found for clustering problems [7], facility location [24, 31, 30], network design [15] and bilevel optimization [96].

3 Progressive approximation approach

In this section we introduce some required notation and describe our progressive approximation algorithm. We define $\mathcal{X} = \{\mathbf{x} \in \{0,1\}^n : \exists \mathbf{w} \in \mathcal{W} \text{ such that } \mathbf{A}\mathbf{x} + \mathbf{G}\mathbf{w} \leq \mathbf{b}\}$, which is a finite set but potentially prohibitively large to be enumerated. For a given $\mathbf{x} \in \mathcal{X}$ we define the index set $I(\mathbf{x}) = \{i \in \{1 \dots n\} : x_i = 1\}$ and the function $\gamma(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \min\{\mathbf{g}^T\mathbf{w} : \mathbf{G}\mathbf{w} \leq \mathbf{b} - \mathbf{A}\mathbf{x}, \mathbf{w} \in \mathcal{W}\}$. For a given $\mathbf{x} \in \mathcal{X}$, this function returns the objective function value of the follower's problem at $(\mathbf{x}, \mathbf{w}^*)$ in the absence of the attacker, where $\mathbf{w}^* = \arg\min_{\mathbf{w}}\{\mathbf{g}^T\mathbf{w} : \mathbf{G}\mathbf{w} \leq \mathbf{b} - \mathbf{A}\mathbf{x}, \mathbf{w} \in \mathcal{W}\}$. Using these definitions, we can fully characterize the follower actions by only using $\mathbf{x} \in \mathcal{X}$ as $\mathbf{w}^*$ can be found implicitly through $\gamma(\mathbf{x})$ (although $\mathbf{w}^*$ may not be unique). Using this notation and the results from [60], Problem Q can be reformulated as the following single-level mixed-integer program, which we call Problem Q'.

$$
\begin{aligned}
[Q'] \quad & \max \quad \lambda && (1)\\
& \text{s.t.} \quad \lambda \leq \gamma(\mathbf{x}) + \sum_{i \in I(\mathbf{x})} d_i y_i, \quad \mathbf{x} \in \mathcal{X} && (2)\\
& \qquad \boldsymbol{\beta}^T\mathbf{y} \leq \Delta && (3)\\
& \qquad \mathbf{y} \in \{0,1\}^n && (4)
\end{aligned}
$$

The proposed solution method *progressively* approximates formulation (1)–(4) by introducing decision variables and constraints. The goal of such strategy is to maintain a reduced number of variables and constraints at every iteration so that problem (1)–(4) remains tractable, while preserving its structural properties. For a given set of indices $I \subseteq \{1 \dots n\}$, we let $\mathcal{X}(I) = \{\mathbf{x} \in \mathcal{X} : x_i = 0,\ i \notin I\}$ be the set of feasible follower actions (x-variables) whose support is included in the variables indexed by I. In addition, we define $Q'(I)$ as the *restricted* version of problem Q' that results from both allowing positive values only for those y-variables whose indices are in I (i.e., $y_i = 0,\ \forall i \notin I$) and imposing Constraints (2) only for $\mathbf{x} \in \mathcal{X}(I)$.

Algorithm 1 describes our progressive approximation mechanism to solve problem Q. In Line 1, problem $P(\mathbf{y})$ is solved using attack $\mathbf{y} = \mathbf{0}$, producing an initial optimal follower solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$ and objective function value $\tilde{z} = z(\mathbf{0})$. This solution also provides a lower bound on the objective value as there are no attacker decisions in the follower's objective function. Using $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$, Line 2 initializes the set of indices I and the lower bound on the optimal objective value, LB. Line 2 also initializes an upper bound, UB, by solving problem $P(\mathbf{y})$ with attack $\mathbf{y} = \mathbf{1}$, meaning that all follower's x-variables are attacked. The while-loop in Lines 3–11 progressively approximates problem Q' by adding elements to the index set I, which ultimately adds y-variables and constraints to $Q'(I)$. Line 4 solves the restricted problem $Q'(I)$ in order to update the upper bound value. Using the solution from Line 4, Line 5 constructs a feasible attack to problem Q', denoted by $\tilde{\mathbf{y}}$, that is used to solve problem $P(\tilde{\mathbf{y}})$

in Line 6. This step produces a feasible solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$ to problem Q' with objective value $\tilde{z}$. Line 7 verifies whether the lower bound and incumbent solution need to be updated, which is done in Line 8. Line 10 updates the set of indices and the γ-function value corresponding to the $\tilde{x}$-variables obtained in Line 6, which helps creating a new constraint related to the follower response $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$ to attack $\tilde{\mathbf{y}}$. Line 12 returns an optimal solution to problem Q for both the attacker and follower, as well as the corresponding objective function value.

Algorithm 1 Progressive approximation

```
1: Calculate $z(\mathbf{0})$ to obtain an optimal solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$ and optimal value $\tilde{z}$. Set $\gamma(\tilde{\mathbf{x}}) = \tilde{z}$.
2: Initialize $I \leftarrow I(\tilde{\mathbf{x}}), LB \leftarrow \tilde{z}$, and $UB \leftarrow z(\mathbf{1})$.
3: while $UB \geq LB$ do
4:     Solve $Q'(I)$ to obtain an optimal solution $\hat{\lambda}$, $\hat{\mathbf{y}} = (\hat{y}_i)_{i \in I}$. Update $UB \leftarrow \hat{\lambda}$.
5:     Define $\tilde{\mathbf{y}} = (\tilde{y}_i)_{i=1}^n$, where $\tilde{y}_i = \hat{y}_i$ for $i \in I$ and $\tilde{y}_i = 0$ for $i \notin I$.
6:     Solve $P(\tilde{\mathbf{y}})$ to obtain an optimal solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$ and optimal value $\tilde{z}$.
7:     if $\tilde{z} > LB$ then
8:         Update $\mathbf{y}^* \leftarrow \tilde{\mathbf{y}}$, $(\mathbf{x}^*, \mathbf{w}^*) \leftarrow (\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$, $LB \leftarrow \tilde{z}$, and $z^* \leftarrow \tilde{z}$.
9:     end if
10:    Update $I \leftarrow I \cup I(\tilde{\mathbf{x}})$ and calculate $\gamma(\tilde{\mathbf{x}}) = \mathbf{c}^T\tilde{\mathbf{x}} + \mathbf{g}^T\tilde{\mathbf{w}}$.
11: end while
12: return $\mathbf{y}^*$, $\mathbf{x}^*$, $\mathbf{w}^*$, $z^*$.
```

The following propositions establish the correctness and finite termination of Algorithm 1, as well as some intermediate results. In particular, Proposition 1 proves that function $\gamma(\mathbf{x})$ can be computed in Lines 1 and 10 using the solution to problem $P(\mathbf{y})$ in Lines 1 and 6, respectively, without additional computational effort.

Proposition 1 *Let $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$ be an optimal solution to $P(\tilde{\mathbf{y}})$ for any $\tilde{\mathbf{y}}$ satisfying (3) and (4). Then, $\gamma(\tilde{\mathbf{x}}) = \mathbf{c}^T\tilde{\mathbf{x}} + \mathbf{g}^T\tilde{\mathbf{w}}$.*

Proof. Suppose for a contradiction that a solution $\mathbf{w}' \in \mathcal{W}$ exists such that $\mathbf{c}^T\tilde{\mathbf{x}}+\mathbf{g}^T\mathbf{w}' < \mathbf{c}^T\tilde{\mathbf{x}}+\mathbf{g}^T\tilde{\mathbf{w}} = \gamma(\tilde{\mathbf{x}})$ and $\mathbf{G}\mathbf{w}' \leq \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$. Because $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$ is optimal to $P(\tilde{\mathbf{y}})$, then $\mathbf{c}^T\tilde{\mathbf{x}} + \mathbf{g}^T\tilde{\mathbf{w}} + \sum_{i=1}^n d_i\tilde{y}_i\tilde{x}_i \leq \mathbf{c}^T\tilde{\mathbf{x}} + \mathbf{g}^T\mathbf{w} + \sum_{i=1}^n d_i\tilde{y}_i\tilde{x}_i$, for any $\mathbf{w} \in \mathcal{W}$ satisfying $\mathbf{G}\mathbf{w} \leq \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$, including $\mathbf{w}'$. However, this implies that $\mathbf{c}^T\tilde{\mathbf{x}} + \mathbf{g}^T\mathbf{w}' \geq \mathbf{c}^T\tilde{\mathbf{x}} + \mathbf{g}^T\tilde{\mathbf{w}}$, a contradiction. Because $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$ satisfies $\mathbf{G}\tilde{\mathbf{w}} \leq \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$, we conclude that $\gamma(\tilde{\mathbf{x}}) = \mathbf{c}^T\tilde{\mathbf{x}} + \mathbf{g}^T\tilde{\mathbf{w}}$. □

Proposition 2 *At every iteration of the algorithm, UB provides an upper bound on the optimal value of Problem Q'.*

Proof. The result holds for the initial upper bound in Line 2 because for any $(\mathbf{x}, \mathbf{w})$ that is feasible for P it is true that $\mathbf{c}^T\mathbf{x} + \mathbf{g}^T\mathbf{w} + \sum_{i=1}^n d_iy_ix_i \leq \mathbf{c}^T\mathbf{x} + \mathbf{g}^T\mathbf{w} + \sum_{i=1}^n d_ix_i$. Now, because $\mathcal{X}(I) \subseteq \mathcal{X}$, it follows that relaxing problem Q' by using only the Constraints (2) indexed by the elements in $\mathcal{X}(I)$ provides a valid upper bound for Q'. Now, note that Problem $Q'(I)$ may have less y-variables than problem Q'. However, by construction of the set $\mathcal{X}(I)$, we have that $I(\mathbf{x}) \subseteq I$, for any $\mathbf{x} \in \mathcal{X}(I)$. Therefore, $\gamma(\mathbf{x})+\sum_{i=1}^n d_ix_iy_i = \gamma(\mathbf{x})+\sum_{i\in I(\mathbf{x})} d_iy_i$ because $x_i = 0$, for all $i \notin I(\mathbf{x})$, and $x_i = 1$, for all $i \in I(\mathbf{x})$. Because $\boldsymbol{\beta} \geq 0$, the support of any optimal attacker solution to $Q'(I)$ when all y-variables are included in Constraint (2) is included in I, allowing us to use only the subset of y-variables indexed in I. □

Proposition 3 *At every iteration of the algorithm, LB provides a lower bound on the optimal value of problem Q'.*

Proof. The result holds for the initial lower bound in Line 2 because for any $(\mathbf{x}, \mathbf{w})$ that is feasible for P it is true that $\mathbf{c}^T\mathbf{x} + \mathbf{g}^T\mathbf{w} \leq \mathbf{c}^T\mathbf{x} + \mathbf{g}^T\mathbf{w} + \sum_{i=1}^n d_iy_ix_i$. For Line 8, the result follows because any feasible solution $\hat{\mathbf{y}}$ to problem $Q'(I)$ obtained in Line 4 is extended to a feasible solution $\tilde{\mathbf{y}}$ to Problem Q' in Line 5. This is also true for the incumbent solution updated in Line 8, which records the maximum observed lower bound. □

Proposition 4 *Algorithm 1 performs at most n iterations of the loop and returns an optimal solution to Problem Q'.*

Proof. Consider an optimal solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$ of problem $P(\tilde{\mathbf{y}})$ obtained in Line 6 at any iteration of the algorithm. We consider two cases. In the first case, we assume that for this solution $I(\tilde{\mathbf{x}}) \subseteq I$, meaning that set I is not updated in Line 10. This means that $\tilde{\mathbf{x}}$ was already in $\mathcal{X}(I)$. From Constraints (2), we have that $UB = \widehat{\lambda} \leq \gamma(\tilde{\mathbf{x}}) + \sum_{i \in I(\tilde{\mathbf{x}})} d_i \tilde{y}_i$. Further, from Propositions 2 and 3, we have that $LB \leq UB$ and because $(\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$ is an optimal solution to problem $P(\tilde{\mathbf{y}})$, it follows that $z(\tilde{\mathbf{y}}) = \gamma(\tilde{\mathbf{x}}) + \sum_{i \in I(\tilde{\mathbf{x}})} d_i \tilde{y}_i \leq LB$. From these observations it follows that $LB \leq UB \leq \gamma(\tilde{\mathbf{x}}) + \sum_{i \in I(\tilde{\mathbf{x}})} d_i \tilde{y}_i = z(\tilde{\mathbf{y}}) \leq LB$. Therefore, the stopping criterion in Line 3 is met and the algorithm returns an optimal solution to Q'. In the second case, we assume that $I(\tilde{\mathbf{x}}) \nsubseteq I$, meaning that set I needs to be updated in Line (10) of the algorithm. If the stopping criterion is not met, this leads to a new iteration in which Lines 4–10 are executed again. The process continues until $I(\tilde{\mathbf{x}}) \subseteq I$, where the results from the first case apply, or until the stopping condition is met, meaning that there is an alternative optimal solution to Q' (which will be discovered in the next iteration because $\widehat{\lambda} = LB = UB$ in Line 4). In any case, the maximum number of times that set I is updated is n, which is when indices are added one at a time. As a result, Algorithm 1 is correct and finishes in no more than n iterations. □

Remark 1 *The scalability of the proposed progressive approximation scheme is based on the assumption that the optimal follower actions $\mathbf{x} \in \mathcal{X}$ to each subproblem $P(\mathbf{y})$ have sparse support. As such, sets $I(\mathbf{x})$ are small and therefore the resulting problems $Q'(I)$ are tractable. If the algorithm along its resolution finds an optimal follower strategy $\mathbf{x} \in \mathcal{X}$ to a problem $P(\mathbf{y})$ of dense support, the restricted problems $Q'(I)$ may become just as difficult as the interdiction problem Q.*

4 Acceleration strategies

In this section we describe additional algorithmic features that can be used to accelerate the progressive approximation method described in Section 3. These enhancements include a metaheuristic procedure to quickly find solutions to the restricted problem $Q'(I)$, a binary search to dynamically reduce the search space based on the objective function value found at any iteration, a cutting planes method to solve restricted interdiction subproblems and super-valid inequalities to strengthen the mathematical formulation of $Q'(I)$.

4.1 Metaheuristic

We use a *Multi-Start Iterated Local Search* (MSILS) [59] heuristic to find near-optimal solutions to problem $Q'(I)$ that can improve the current lower bound at any iteration of Algorithm 1. The MSILS consists of three phases: *multi-start pool*, *local search*, and *shaking*. The local search and shaking phases constitute the *iterative local search* (ILS) part of MSILS. If no improving solution is found at the end of MSILS, then $Q'(I)$ is solved exactly using an MIP.

The MSILS starts by constructing a set of solutions to be used as starting points in the local search procedure, which we refer to as the multi-start pool. To do so, we randomly generate T_1 feasible attacks over the current index set I. Because an attack that does not deplete the attacker's budget is suboptimal (because d-parameters are nonnegative), we only generate attacking strategies that are maximal, meaning that they cannot be augmented by setting more y-variables to one without exceeding the attacker's budget Δ. We estimate a score for each of those T_1 attacks $\mathbf{y}$ by computing $s(\mathbf{y}) = \sum_{i \in I} d_i y_i$. Only T_2 ($< T_1$) attacks with the largest scores are kept for further consideration. For each of these attacks, we solve Problem $P(\mathbf{y})$ and compute $z(\mathbf{y})$. We select T_3 ($< T_2$) of these attacks with the largest z-values to be part of the multi-start pool of our MSILS.

For each of the T_3 starting solutions, our procedure performs an ILS that uses a simple `SWAP` operator. We use this operator in both the local search and shaking phases. This operator takes two indices, $i, j \in I$, such that $y_i + y_j = 1$ and swaps them. That is, if $y_i = 1$ and $y_j = 0$, then `SWAP` creates a solution $\mathbf{y}'$ where $y'_j = 1$ and $y'_i = 0$, with all other components equal to those in $\mathbf{y}$. The resulting attack $\mathbf{y}'$ after applying the SWAP operator is kept only if it is feasible (i.e., it satisfies the attacker's budget constraint) and it is discarded, otherwise. For the local search phase, it is also required that $\mathbf{y}'$ improves the attacker's objective function value with respect to attack $\mathbf{y}$. This second condition is expensive to evaluate within the local search procedure as it involves the solution of $P(\mathbf{y}')$. Instead, we replace this evaluation by performing the following surrogate estimation. We select H of the previously visited follower solutions, $(\mathbf{x}^1, \mathbf{w}^1), \dots, (\mathbf{x}^H, \mathbf{w}^H)$, for some $H \geq 1$, with the smallest nominal cost (i.e., objective function of Problem P). This cost is independent of the attacker's strategy so a sorted list with these solutions can be maintained every time a follower solution is found. For each of those H solutions, we compute the change in the follower's objective function attained by performing the swap move. The minimum of these changes, which can be positive, negative, or zero, is the surrogate value of the swap move. If positive, we compute $z(\mathbf{y}')$ by solving Problem $P(\mathbf{y}')$, and the swap is *accepted* only if $z(\mathbf{y}') > z(\mathbf{y})$. In this case, incumbent solution and objective values are updated given that $\mathbf{y}'$ improves the current solution. The local search continues using the incumbent solution as starting point. We use a *first improvement* policy in our ILS that guarantees that the incumbent is updated only if the objective function value improves.

The MSILS performs a shaking procedure when the local search finds no solution that improves the current incumbent. This shaking procedure consists of a set of random swaps that are performed until a feasible attack is found, regardless of its objective function value. When this happens, the local search is executed again starting from the *shook* solution. Upon calibration of the method, we determine that two random shakes during the ILS are sufficient to provide a meaningful diversification, and that only marginal gains (if any) can be attained beyond this number. After performing three local searches and two shakes for a given attack (i.e., local search $\rightarrow$ shake $\rightarrow$ local search $\rightarrow$ shake $\rightarrow$ local search), the ILS is repeated using a new attack from the multi-start pool. The MSILS procedure terminates when all attacks in the multi-start pool are used (i.e., T_3 iterations).

We point out that any solution $(\mathbf{y}', \lambda')$ obtained by MSILS is feasible to $Q'(I)$, and then its objective function value is a lower bound on the value of $Q'(I)$. If the best solution found by the MSILS suggests a potential improvement in LB, i.e., $\lambda' > LB$, then $\hat{\mathbf{y}} = \mathbf{y}'$ in Line 4 and Algorithm 1 continues as described, except that UB is not updated using λ'. This is because λ' is not necessarily an upper bound on the value of Q' as $\mathbf{y}'$ may not be optimal for the attacker. If the best solution found by MSILS is such that $\lambda' \leq LB$, Line 4 is executed as described in Algorithm 1 and $Q'(I)$ is solved exactly using an MIP. Based on our empirical experience, the MSILS procedure is especially useful when the size of the restricted interdiction problems $Q'(I)$ starts to grow, which makes the MIP executed in Line 4 computationally too demanding to solve.

4.2 Binary search

We further accelerate the solution of Problem $Q'(I)$ in Line 4 of Algorithm 1 by using a *binary search* procedure. This procedure is executed whenever the metaheuristic finds no improving solution, meaning that $Q'(I)$ has to be solved exactly. To describe our binary search procedure, let l and u be a lower and an upper bound on the optimal solution to problem $Q'(I)$, respectively, and let $m = \lceil (l+u)/2 \rceil$. The values of l and u can be initialized using $z(\mathbf{0})$ and $z(\mathbf{1})$, respectively. Define $Q'(I, m, u)$ as the problem resulting from adding the constraint $m \leq \lambda \leq u$ into problem $Q'(I)$. If $Q'(I, m, u)$ is feasible, then we know that a solution exists with optimal objective value of at least m, so we set $l = m$, a tighter lower bound. We determine whether a feasible solution exists by attempting to solve $Q'(I, m, u)$ using an MIP solver (e.g., Gurobi) and stopping the solution procedure once the first (integer) feasible solution is found. If $Q'(I, m, u)$ is infeasible, then we know that no feasible solution exists with objective value in the interval $[m, u]$, so we set a tighter upper bound given by $u = m - \epsilon$, where $\epsilon > 0$ represents a predetermined tolerance. If the value of $z(\mathbf{y})$ is integer for any attack satisfying Constraints (3) and (4), then $\epsilon = 1$. This is the case of the interdiction version of problems such as shortest path, facility location, and knapsack, among others, where it is also typically assumed that parameters are integer valued.

We iterate this procedure until $l \geq u$. Upon termination, l is is the optimal value of $Q'(I)$ and the last feasible solution found is an optimal solution. The bounds m and u encountered when solving problem $Q'(I, m, u)$ can be used to further strengthen the solution to Problem $Q'(I)$. The lower bound m can be used to derive supervalid inequalities, as described in Section 4.4. The upper bound u, on the other hand, can be used to tighten Constraints (2). Using the results from [60], we define $\widetilde{d}_i = \max\{0, \min\{d_i, u - \gamma(\mathbf{x})\}\}$, for each $i \in I$ and $\mathbf{x} \in \mathcal{X}(I)$. As a result, the inequality

$$\lambda \leq \gamma(\mathbf{x}) + \sum_{i \in I(\mathbf{x})} \widetilde{d}_i y_i. \tag{5}$$

is valid for Problem Q' because no attack will improve the optimal value of the attacker's problem beyond u (a valid upper bound). We emphasize that u is always an upper bound on Q' (up to a tolerance of ϵ) along the binary search because it is only updated when $Q'(I, m, u)$ is infeasible, thus we know there is no feasible solution with objective value in the interval $[m, u]$. Inequalities (5) are tighter than their counterpart in Constraints (2) (i.e., for the same $\mathbf{x} \in \mathcal{X}(I)$) because $\widetilde{d}_i \leq d_i$ for any $i \in I(\mathbf{x})$. Moreover, we point out that as the algorithm progresses and the value of u declines, Inequalities (5) become tighter.

4.3 Cutting-planes algorithm

In this section, we describe a cutting-planes algorithm to find a feasible solution to Problem $Q'(I, m, u)$ or to determine its infeasibility at any iteration of the binary search. The underlying motivation is twofold. First, for a given I, not all of the constraints

$$\lambda \leq \gamma(\mathbf{x}) + \sum_{i \in I(\mathbf{x})} d_i y_i, \qquad \mathbf{x} \in \mathcal{X}(I) \tag{6}$$

are needed to determine the feasibility of a candidate solution $(\mathbf{y}, \lambda)$ to $Q'(I, m, u)$, but only that with the minimum value of $\gamma(\mathbf{x})$ among $\mathbf{x} \in \mathcal{X}(I)$. Second, set $\mathcal{X}(I)$ can be potentially very large to enumerate even for a small index set I. For these reasons, we maintain a subset of Constraints (6) and add new constraints as needed in a cutting-plane fashion until we find a feasible attack to $Q'(I, m, u)$. To formally describe this algorithm, we define $\bar{\mathcal{X}}(I) \subseteq \mathcal{X}(I)$ as a subset of follower actions for a given subset I. We also define Problem $P(\mathbf{y}, I)$ as the version of Problem $P(\mathbf{y})$ that only uses problem elements (i.e., variables or constraints) that are indexed in I. Note that $P(\mathbf{y}, I)$ is always feasible by construction of $\mathcal{X}$ and because $\bar{\mathcal{X}}(I) \subseteq \mathcal{X}(I) \subseteq \mathcal{X}$. Moreover, we define $\bar{Q}'(I, m, u)$ as the version of Problem $Q'(I, m, u)$ with Constraints (6) only for $\mathbf{x} \in \bar{\mathcal{X}}(I)$.

Algorithm 2 describes our strategy. Line 1 finds a follower solution to initialize set $\bar{\mathcal{X}}(I)$ by solving Problem $P(\mathbf{1}, I)$. This line also computes $\gamma(\mathbf{x})$, which is necessary to solve $\bar{Q}'(I, m, u)$. Line 2 initializes set $\bar{\mathcal{X}}(I)$ and the objective values to Problems $\bar{Q}'(I, m, u)$ and $Q'(I, m, u)$ as $-\infty$, respectively. Line 3 seeks a feasible solution to Problem $\bar{Q}'(I, m, u)$. If no solution is found, then the algorithm goes to Line 9 and returns $\bar{\lambda} = -\infty$, indicating that $Q'(I, m, u)$ is infeasible. Otherwise, the feasible attack is stored in $\bar{\mathbf{y}}$ and its objective function value in $\bar{\lambda}$. The while-loop in Lines 4–9 verifies that the current attack is feasible to Problem $Q'(I, m, u)$, otherwise adding follower solutions to $\bar{\mathcal{X}}(I)$. Line 5 solves Problem $P(\bar{\mathbf{y}}, I)$ to find the follower's response to attack $\bar{\mathbf{y}}$ and its corresponding γ-value. This response is added to set $\bar{\mathcal{X}}(I)$ in Line 6, which also calculates the objective function value of the combined attacker-follower decisions, which we denote by λ_I. Line 7 seeks a feasible solution to Problem $\bar{Q}'(I, m, u)$ using the updated set $\bar{\mathcal{X}}(I)$. If none is found, then the algorithm returns $\bar{\lambda} = -\infty$. Otherwise, the information of such feasible solution is stored and the loop goes back to Line 4. Note that $\bar{\lambda} > \lambda_I$ indicates that attack $\bar{\mathbf{y}}$ is infeasible to $Q'(I, m, u)$, because there is a follower solution $\bar{\mathbf{x}} \in \mathcal{X}(I)$ such that $\bar{\lambda} > \lambda_I = \gamma(\bar{\mathbf{x}}) + \sum_{i \in I(\bar{\mathbf{x}})} d_i \bar{y}_i$, which violates Constraints (6). Upon termination, Algorithm 2 returns a feasible solution to Problem $Q'(I, m, u)$ because any feasible attack to Problem $\bar{Q}'(I, m, u)$ satisfies the budget constraint and also because the termination condition of the while-loop indicates that $\bar{\lambda} \leq \lambda_I = \gamma(\bar{\mathbf{x}}) + \sum_{i \in I(\bar{\mathbf{x}})} d_i \bar{y}_i \leq \gamma(\mathbf{x}) + \sum_{i \in I(\bar{\mathbf{x}})} d_i \bar{y}_i$, $\forall \mathbf{x} \in \mathcal{X}(I)$, where the last inequality holds given the results in Proposition 1 with $\bar{\mathbf{x}}$ being an optimal solution to $P(\bar{\mathbf{y}}, I)$.

Algorithm 2 finishes in a finite number of iterations because the number of feasible solutions to Problem $Q'(I,m,u)$ is finite, if any exist.

In practice, we implement Algorithm 2 as a conventional Branch-and-Cut algorithm, in which feasible solutions in Lines 3 and 7 correspond to local solutions at any active node in the exploration tree. Line 5 acts as a separation problem that creates an optimality cut that is imposed in Line 6 by adding a new element to set $\bar{\mathcal{X}}(I)$. In this case, the algorithm stops once the solution at an active node is found to be feasible after solving the separation problem. Further, the initialization of $\bar{\mathcal{X}}$ in Lines 1 and 2 can include attacks from previous iterations of Algorithm 1 and other problems within the binary search. Also, the problem of finding feasible attacks to $\bar{Q}'(I,m,u)$ in Lines 3 and 7 is further strengthened by using the valid inequalities described in Section 4.4.

Algorithm 2 Cutting-plane algorithm for finding a feasible solution to $Q'(I,m,u)$

```
1: Solve $P(\mathbf{1},I)$ to obtain a solution $(\bar{\mathbf{x}},\bar{\mathbf{w}})$ and compute $\gamma(\bar{\mathbf{x}}) = \mathbf{c}^T\bar{\mathbf{x}} + \mathbf{g}^T\bar{\mathbf{w}}$.
2: Initialize $\bar{\mathcal{X}}(I) = \{\bar{\mathbf{x}}\}$, $\bar{\lambda} = -\infty$, and $\lambda_I = -\infty$.
3: Obtain a feasible solution $\bar{\mathbf{y}}$ to $\bar{Q}'(I,m,u)$ and denote its objective value by $\bar{\lambda}$.
4: while $\bar{\lambda} > \lambda_I$ do
5:     Solve $P(\bar{\mathbf{y}},I)$ to obtain an optimal solution $(\bar{\mathbf{x}},\bar{\mathbf{w}})$ and compute $\gamma(\bar{\mathbf{x}}) = \mathbf{c}^T\bar{\mathbf{x}} + \mathbf{g}^T\bar{\mathbf{w}}$.
6:     Add $\bar{\mathbf{x}}$ to $\bar{\mathcal{X}}(I)$ and update $\lambda_I \leftarrow \gamma(\bar{\mathbf{x}}) + \sum_{i\in I(\bar{\mathbf{x}})} d_i\bar{y}_i$.
7:     Obtain a feasible solution $\bar{\mathbf{y}}$ to $\bar{Q}'(I,m,u)$ and denote its objective value by $\bar{\lambda}$. If none found, set $\bar{\lambda} = -\infty$.
8: end while
9: return $\bar{\mathbf{y}}$, $\bar{\lambda}$.
```

4.4 Supervalid inequalities

Supervalid inequalities, as defined by Israeli and Wood [51], are inequalities that that may cut integer feasible or even optimal solutions but guarantee that at least one optimal solution is saved in an incumbent. We use super-valid inequalities to tighten the problems solved in Lines 3 and 7 in Algorithm 2, which need to be solved multiple times for each Problem $Q'(I,m,u)$ at every iteration of the binary search procedure. The goal of the supervalid inequalities is to speed up the search for a feasible solution to Problem $Q'(I,m,u)$ or to quickly determine its infeasibility while taking advantage of the existing lower bound m. In this context, we use the following alternative definition of super-valid inequalities [83].

Definition 1 *Given an objective function value m, an inequality $\mathbf{a}^T\mathbf{y} \geq b$, with $\mathbf{a}$, $\mathbf{y} \in \mathbb{R}^n$, for some integer $n > 0$, and $b \in \mathbb{R}$, is super-valid if $\mathbf{a}^T\hat{\mathbf{y}} \geq b$ for all feasible $\hat{\mathbf{y}}$ having an objective function value greater than m.*

Using Definition 1, we construct supervalid inequalities for problem $Q'(I,m,u)$ corresponding to follower solutions $\mathbf{x} \in \mathcal{X}$ such that $\gamma(\mathbf{x}) < m$. If this is the case, we know that at least one attack can be executed so that $m \leq \gamma(\mathbf{x}) + \sum_{i\in I(\mathbf{x})} d_i y_i$. Whenever an inequality (5) is added, we also add a constraint of the form

$$\sum_{i\in I(\mathbf{x})} y_i \geq k(\mathbf{x},m), \tag{7}$$

where $k(\mathbf{x},m)$ is constructed according to the following steps.

- **Step 1:** Sort indices in $I(\mathbf{x})$ in nonincreasing order with respect to d-parameters, in such a way that $d_{I_1} \geq d_{I_2} \geq \cdots \geq d_{I_t}$ with $t = |I(\mathbf{x})|$.
- **Step 2:** Define $k(\mathbf{x},m) \leftarrow \arg\min_s\{s : \sum_{1\leq k\leq s} d_{I_k} \geq m - \gamma(\mathbf{x})\}$.

Inequality (7) is supervalid for problem $Q'(I,m,u)$ because, if not satisfied, an action of value strictly lower than m could be attained by the follower, meaning that m is not a valid lower bound. This means that in addition to reducing the time to find a feasible solution, supervalid inequalities in (7) also help reduce the time to declare the infeasibility of $Q'(I,m,u)$. Similar to Inequalities (5), as the algorithm progresses and the value of m increases, Inequalities (7) become tighter.

5 Computational results

We use our progressive approximation method to solve the interdiction version of three problems: shortest path, 0-1 knapsack, and facility location. In all these problems, we assume that the budget-constrained attacker can only modify the cost of the follower's binary decision variables in the objective function, as stated in the definition of Problem Q. Although we only focus on these three problems, our solution strategy is general for any problem that can be written as Problem Q.

We report the performance of our method for a set of randomly generated instances as well as for instances available in the literature. Although data sets exist for the non-interdiction variants of the 0-1 knapsack and facility location problems, as well as for 0-1 knapsack with interdiction, we created new sparse data sets to illustrate the scalability of our method under the conditions noted in Remark 1. This is because in most of the existing data sets, the follower's optimal solution is dense (i.e., the number of nonzero binary variables in an optimal solution is large with respect to the total number of variables) and our method is designed to exploit the optimal solution's sparsity. For the shortest-path interdiction problem, we present a comparison of our method with the method of Lozano and Smith [60], which to our knowledge is the state-of-the-art for such problem. Naturally, sparsity is not the only factor determining the solution time. Other factors such as the relative magnitudes between c- and d-parameters also play a role. All the data sets used in this article, either new or existent, are available at http://claudio.contardo.org.

Our implementation uses Julia 1.1 with the JuMP interface v18.5 and IBM CPLEX 12.8 as multipurpose optimization solver, which runs on an Intel Xeon E5-2637 v2 @ 3.50 GHz with 128 GB of RAM. Although this machine is capable of executing code in parallel, for reproducibility purposes we only use one core per run. In all our algorithms we parameterized the MSILS procedure with $T_1 = 500$ (i.e., number of random attacks generated), $T_2 = 50$ (i.e., the number of attacks kept to solve $P(\mathbf{y})$), $T_3 = 5$ (i.e., the number of attacks used in the multi-start procedure), and $H = 100$ (i.e., the number of attacks to for the surrogate estimation of an attack's performance).

5.1 Shortest path interdiction

We generate shortest-path interdiction games using nine very-large-scale road networks available in the literature. These networks appeared in the 9th DIMACS Implementation Challenge [1] and contain between 35,697 and 20,394,245 arcs. Some of these instances are undirected, so we follow the directions provided by Raith and Ehrgott [73] to make them directed while ensuring their connectivity. Table 1 provides the number of nodes and arcs after the necessary modifications for each data set. Data sets RI, NJ, NY, FL, CA, TX, and DC contain the road networks of the states of Rhode Island, New Jersey, New York, Florida, California, and Texas, as well as the District of Columbia, all in the United States. Similar to Lozano and Smith [60], we produce nine instances for each network by randomly generating the same number of OD-pairs.

Table 1: Road networks

Data set	# Nodes	# Arcs
DC	9,559	35,697
RI	56,658	171,413
NJ	330,386	1,062,339
NY	716,215	2,214,801
FL	1,048,506	3,284,715
CA	1,613,325	4,901,941
TX	2,073,870	6,437,637
USE	3,598,623	11,727,919
USW	6,262,104	20,394,245

We compare our method against the backward sampling method of Lozano and Smith [60], which we recompiled and executed on the exact same machine. To make the comparison, we disable the fortification stage in the backward sampling method, so the resulting method resembles Problem Q.

Moreover, β-parameters in Problem Q are all equal to one, as required by Lozano and Smith's method. In other words, the cost of each attacker's action is unitary, which reduces Constraint 3 to a cardinality constraint. We use budgets (i.e., Δ) varying between 1 and 20 units.

Table 2 presents the average CPU time (in seconds over the nine generated instances) taken by each method to solve each set of instances for the same budget. We allow each method to run for 24 hours and record the time to obtain an optimal solution, if achieved. The backward sampling method rapidly runs out of memory for any budget and any instance from the data sets NY, FL, CA, TX, USE and USW. For this reason, we restrict the comparison to the three smaller data sets, DC, RI, and NJ, where both algorithms successfully solve all instances to optimality within the time limit. We observe in Table 2 that the backward sampling method is faster on average for all network sizes and budgets considered. However, the proposed progressive approximation is still competitive, considering that the backward sampling method cannot scale beyond the network sizes in Table 2.

Table 2: Average CPU time to solve the shortest-path interdiction problem using the proposed progressive approximation method and backward sampling framework [60]

Set	Progressive approximation						Backward sampling					
	Δ						Δ					
	1	3	5	10	15	20	1	3	5	10	15	20
DC	19.4	19.9	20.5	21.1	22.7	24.7	2.9	2.9	3.3	7.0	8.4	15.1
RI	20.6	23.0	25.5	39.3	110.5	211.6	4.7	9.1	14.0	26.1	65.9	113.4
NJ	28.8	42.1	54.5	151.5	415.2	1,302.5	22.7	35.3	51.9	106.0	188.9	647.9

We now report the performance of our method when solving the very-large-scale interdiction problems on road network data sets NY, CA, FL, TX, USE and USW. Table 3 shows the number of problems solved to optimality (column *#Opt*), the average CPU time to solve those instances (column *CPU*, in seconds), and the average number of edges required to achieve optimality (column $|E|$) for different budgets. We only report the average CPU times and the average number of edges for those problems solved to optimality within the time limit. Moreover, we include the number of edges necessary to unvelil the optimal solution to illustrate the effectiveness of our solution strategy in keeping the size of the problem small.

Table 3 shows that the scalability of our method is less sensitive to the size of the underlying network than to the attacker's budget. This is because a larger budget admits more elements in the support of the x-decision variables, which implies that I-sets are larger and problems in Line 4 of Algorithm 1 are harder to solve. Although a larger network implies more choices available for the agents, the nature of the problem guarantees that the follower still uses a small portion of the network, which is what our method exploits (recall that any shortest-path uses no more than $n-1$ arcs, where n is the number of nodes in the underlying network.) Further, Table 3 shows that the progressive approximation is able to solve all problems with $\Delta \leq 10$ and most of those with a larger budget within the time limit. This is because our approximation only uses very small portion of the total number of edges ($\sim$0.06% in the worst-case for NY and $\Delta = 20$) to identify an optimal solution to the problem over the full network. As a result, the scalability of our method is superior to other methods available in the literature for these types of instances.

Table 3: Average CPU time to solve the shortest-path interdiction problem on very-large-road networks

Set	$\Delta = 1$			$\Delta = 3$			$\Delta = 5$			$\Delta = 10$			$\Delta = 15$			$\Delta = 20$		
	#Opt	CPU	$\lvert E\rvert$	#Opt	CPU	$\lvert E\rvert$	#Opt	CPU	$\lvert E\rvert$	#Opt	CPU	$\lvert E\rvert$	#Opt	CPU	$\lvert E\rvert$	#Opt	CPU	$\lvert E\rvert$
NY	9	53.4	283	9	86.6	502	9	114.1	668	9	448.1	997	9	1,587.6	1,297	6	1,594.0	1,135
CA	9	106.6	285	9	179.3	466	9	264.8	694	9	631.1	1,053	9	1,301.1	1,341	7	6,292.2	1,400
FL	9	68.5	256	9	132.8	459	9	165.9	594	9	479.4	979	9	1,542.8	1,345	8	9,708.8	1,515
TX	9	115.1	239	9	176.1	366	9	268.7	479	9	513.1	789	8	1,495.4	950	8	3,367.0	1,160
USE	9	172.0	151	9	295.3	308	9	519.9	433	9	1,151.6	703	7	9,023.5	948	2	13,063.6	1,035
USW	9	316.1	173	9	588.7	291	9	915.3	405	9	2,182.8	588	8	3,392.1	813	5	9,897.6	855

5.2 0-1 Knapsack interdiction

We study a new variant of the 0-1 knapsack interdiction problem in which the follower solves a traditional knapsack problem aiming to maximize the value of the chosen objects, while the attacker aims to decrease the value of a subset of objects subject to a budget constraint. That is, objects are still available for the follower but with less value. We test the performance of our progressive approximation on a set of randomly generated instances for this new problem. Although several data sets exist for knapsack problems in the scientific literature, none of them is useful to illustrate both the benefits and the limits of our method. The instance generators provided by Caprara et al. [22] and DeNegre [36] produce problems where a substantial number of items can be chosen by the follower in an optimal solution, meaning that the follower's solution is not sparse. As a result, the progressive approximation scheme becomes of no practical use as problems in Line 4 of Algorithm 1 quickly become as difficult to solve as the original problem (see Remark 1). Because this problem has a min-max nature, we modify Problems Q and P accordingly, without any further algorithmic change.

We use the following procedure to construct instances containing $10,000$, $100,000$, and $1,000,000$ objects with small knapsack capacities. The nominal value c_i of an object is randomly picked using an uniform distribution in the interval $[1; 1,000]$. For each object, the decrease in value induced by the attacker, d_i, is randomly selected from the interval $[1, d_{max}]$ with $d_{max} \in \{125, 250\}$. Because of the large number of objects available in the knapsack and the relatively small knapsack capacity (i.e., budget), we avoid using uniform distributions to generate the objects' weights. The reason is that using uniform distributions will tend to produce many items with a very large value-to-weight ratio, and thus make the follower's problem trivially solvable. Instead, we generate the objects' weights using a two-step procedure. First, for every item i we generate a random number ξ_i using a normal distribution with parameters $\mu = 100$ and $\sigma = 30$. Then, the weight a_i of object i is computed as $a_i \leftarrow \max\{1, \lceil \xi_i \rceil\}$. The same weight generator is used to calculate the attacker's β-parameters in Constraint (3). To generate the knapsack capacities for the attacker and follower problems, we use values $b \in \{200, 400\}$ and $\Delta \in \{500, 1000, 2000\}$, respectively. This instance generation procedure produces a diverse range of object weights, admitting objects with large value-to-weight ratios—i.e., with high values and low weights—which results on knapsacks being able to fit several hundred objects for some of the larger instances. However, these cases are less frequent compared to that when uniform distributions are used. Similar to the shortest-path interdiction problem, we use a time limit of 24 hours and generate nine instances for each possible combination of b and Δ parameters.

Tables 4–6 summarize the results for the 0-1 knapsack interdiction problem, where N is the number of available objects. For each combination of parameters, we report the number of problems solved to optimality and the minimum, average, and maximum CPU times (in seconds) taken by our algorithm to solve each set of instances. We report the average CPU time only for those instances solved to optimality within the time limit. Our method can solve most of the generated instances (only two instances timed out), illustrating our algorithm's limit.

We emphasize that due to memory issues, we use a different algorithm to handle the 0-1 knapsack problems in Line 6 of Algorithm 1 and Line 5 of Algorithm 2. That is, problems $P(\mathbf{y})$ and $P(\mathbf{y}, I)$, respectively. Because problem $Q'(I)$ is restricted to a typically small subset of objects indexed in I, we use dynamic programming for the solution of the associated 0-1 knapsack problem $P(\mathbf{y}, I)$. Using a similar approach for problems $P(\mathbf{y})$ would require a prohibitively large amount of RAM. Instead, we formulate $P(\mathbf{y})$ as pure-integer linear programs and use a conventional integer programming solver to find their solution.

As expected, the knapsack capacities have a significant impact on our method's scalability. Small values for the knapsack capacities (both the follower's and the attacker's problems) result in easier problems. Further, the number of items does not seem to have a significant impact on the CPU times. Problems with 1M nodes are not 100x more difficult than those with 10,000 nodes. This is due to the solution strategy that keeps the problem size moderate even when the initial problem is very large. Moreover, the impact of d_{max} becomes more relevant for the difficult problems. For small knapsack capacities with a small number of objects, the value of d_{max} seems to have less impact compared to those problems with large capacities and large number of objects.

Using the same instance generation procedure, we explore solving problems containing 10M objects, with larger knapsacks capacities, or using $d_{max} = 500$. Although we solve some instances, these problem size cannot be solved in general by our algorithm.

Table 4: Progressive approximation for 0-1 knapsack interdiction with $N = 10,000$

Δ	b	$d_{max} = 125$				$d_{max} = 250$			
		#opt	min	avg	max	#opt	min	avg	max
500	2000	9	23.3	27.3	35.6	9	25.0	29.3	35.8
	4000	9	32.7	45.2	58.9	9	39.4	60.4	91.7
1000	2000	9	29.6	36.2	50.9	9	31.9	42.2	61.5
	4000	9	58.4	105.4	163.8	9	76.9	194.2	504.8
2000	2000	9	39.9	66.8	132.1	9	53.6	197.3	657.8
	4000	9	158.0	263.7	462.1	7	190.6	798.0	TL

Table 5: Progressive approximation for 0-1 knapsack interdiction with $N = 100,000$

Δ	b	$d_{max} = 125$				$d_{max} = 250$			
		#opt	min	avg	max	#opt	min	avg	max
500	2000	9	46.8	55.1	59.4	9	43.3	54.6	83.3
	4000	9	64.5	92.4	154.7	9	81.5	121.4	199.0
1000	2000	9	61.7	74.8	108.1	9	77.3	96.6	146.5
	4000	9	114.3	227.0	484.4	9	149.9	335.3	631.7
2000	2000	9	116.6	190.4	348.9	9	178.5	295.3	695.1
	4000	9	230.8	651.3	1,543.3	9	680.8	1,950.6	5,824.0

Table 6: Progressive approximation for 0-1 knapsack interdiction with $N = 1,000,000$

Δ	b	$d_{max} = 125$				$d_{max} = 250$			
		#opt	min	avg	max	#opt	min	avg	max
500	2000	9	196.7	240.8	293.0	9	168.1	261.5	351.6
	4000	9	261.4	367.2	480.9	9	350.8	590.0	1,345.9
1000	2000	9	287.2	481.7	1,202.2	9	327.5	665.5	1,412.8
	4000	9	376.4	649.2	939.8	9	759.4	1,683.9	6,883.8
2000	2000	9	417.8	868.6	1,350.4	9	689.8	2,230.2	3,861.6
	4000	9	1,356.0	1,953.8	2,948.2	9	1,834.6	4,801.6	11,133.3

5.3 Facility location interdiction

We now assess the effectiveness of our method when solving the interdiction version of some facility location problems (FLPs). Although our modeling and algorithmic approach can solve many variants of FLPs, we focus on the uncapacitated FLP (UFLP) and the single-source capacitated FLP (SSCFLP). Extensive literature exists on the classical versions of FLPs, but very few works focus on their interdiction counterpart. To our knowledge, there is no study that focuses on the interdiction version of UFLP and SSCFLP in which location costs can be increased by an attacker who is subject to a general budget constraint.

In UFLP, the input data consists of two sets, I and J, representing the set of potential facilities and customers, respectively. A fixed cost f_i is associated with selecting location $i \in I$, which is typically related to the fixed operational cost of using a facility. Parameters c_{ij} describe the cost per unit of satisfying the demand of customer j from facility i. The goal is to open a subset of facilities and to assign customers to the open facilities at minimum total cost in such a way that every customer is assigned to exactly one facility. In addition to the elements in UFLP, in SSCFLP customers have known demands denoted by q_j, $j \in J$. Moreover, there is a finite capacity for each facility, denoted by K_i, that limits the amount of units available for customers. This capacity constraint makes the SSCFLP harder to solve because of the *bin-packing* problem lies underneath its feasibility. In addition, integrality constraints on the asignment variables become necessary, which are otherwise redundant for the uncapacitated variant. As a result, the SSCFLP is substantially harder to solve than the UFLP.

We employ binary variables y_i to indicate whether facility $i \in I$ is chosen. Further, for every pair facility-customer, we use binary variable x_{ij} to indicate whether customer $j \in J$ is assigned to facility $i \in I$. Using these elements, we formulate UFLP and SSCFLP as mixed-integer problems [see, for instance 50] and solve them using a commercial solver. Certainly, significant speed-ups can be achieved if customized solution algorithms are used. For state-of-the-art algorithms for these two problems we refer the reader to Fischetti et al. [39], Gadegaard et al. [44]. The resulting integer program for the SSCFLP is shown in (8)–(11).

$$\begin{aligned}
\min \quad & \sum_{i \in I} f_i y_i + \sum_{i \in I, j \in J} c_{ij} x_{ij} & & (8)\\
\text{s.t.} \quad & \sum_{i \in I} x_{ij} = 1, & j \in J \qquad & (9)\\
& \sum_{j \in J} q_j x_{ij} \leq K_i y_i, & i \in I \qquad & (10)\\
& \mathbf{x} \in \{0,1\}^{|I| \times |J|}, & \mathbf{y} \in \{0,1\}^{|I|}. \qquad & (11)
\end{aligned}$$

In the absence of capacities, the (allocation) x-variables can be relaxed to take values in $[0, 1]$ and the model still yields an integer solution. Indeed, once the (location) y-variables are known, each customer can be assigned to the closest open facility. It is well known in the literature that Inequalities (10) are weak linking constraints. However, they are necessary in SSCFLP, whose formulation can be strengthened with the stronger constraints

$$x_{ij} \leq y_i, \qquad \forall i \in I, j \in J. \tag{12}$$

Constraints (12) can be added as-needed to (8)–(11) in a cutting plane fashion or all at once to strengthen the linear relaxation. The formulation for UFLP consist of minimizing (8) subject to (9), (12), $\mathbf{x} \in [0,1]^{|I| \times |J|}$, and $\mathbf{y} \in \{0,1\}^{|I|}$.

To guarantee the sparsity in the optimal solution, we generate a new set of instances for the two FLP variants of our interest. We generate random demands using an integer uniform distribution in the interval $[1, 100]$ for 500 and 1000 customers. We assume that only location decisions (i.e., y-variables) can be interdicted, which is the usual assumption in the literature [4, 5, 6]. We use 50 and 100 candidate facilities, whose fixed and interdiction costs (i.e., d-parameters in Problem P) are integer values uniformly selected from the intervals $[8,000; 10,000]$ and $[1,000; 3,000]$, respectively. Facility and customer locations are randomly distributed in a square of dimensions 1000x1000. Using such locations, the cost of assigning customer j to facility i, c_{ij}, is computed as the euclidean distance in the plane multiplied by a random number uniformly distributed in the interval $[0.9; 1.1]$. Therefore, our instances may not satisfy the triangle inequality. We use parameter κ to control the number of facilities to locate in an optimal solution. Defining D as the sum of all customer demands, the average demand a facility is expected to cover is $\widehat{D} = D/\kappa$ units. Therefore, capacities are randomly generated using an integer uniform distribution in the interval $[\lceil 0.9\widehat{D} \rceil; \lceil 1.1\widehat{D} \rceil]$. We use $\kappa = 5$ in all experiments and vary the attacker budget using integer values in the interval $\Delta = [1, \ldots, 5]$. We assume that the interdiction costs (i.e., β-parameters) are unitary. Following this instance generation approach, we guarantee that all quantities (demands, costs, capacities) are integer. For each parameter setting, we generate nine problems and set a solution time limit of 24 hours.

Tables 7 and 8 show the performance of our method when solving UFLP and SSCFLP, respectively. In these tables, we report the minimum, average and maximum CPU times among all the instances solved to optimality for the same parameter configuration. We observe that the interdiction versions of UFLP and SSCFLP maintain the same hierarchy of complexity than their non-interdiction counterparts, where UFLP is easier to solve than SSCFLP. This behavior is mainly due to the complexity of the core problems, given that problem $P(\mathbf{y})$ in our algorithm is still a FLP. Although not reported in the tables, we observe that the number of iterations required to achieve optimality is similar regardless of the instance and parameter configuration. This shows that our algorithm has the potential to scale as long as the subproblems at each iteration are solved within reasonable time. Further, we observe that changes in the attacker's budget impact the CPU time in a similar way than in the shortest path and 0-1 knapsack interdiction problems, where a larger budget results in a more difficult problem to solve. The effectiveness of our progressive approximation method on these instances is also due to both parameter κ and the fixed costs, which keep the number of optimal locations low for any interdiction pattern. This feature results in restricted interdiction problems $Q'(I)$ with an equally low number of binary variables, which helps at keeping all subproblems tractable.

Table 7: CPU time (in seconds) for UFLP with interdiction using progressive approximation

$\lvert I\rvert$	Δ	$\lvert J\rvert = 500$				$\lvert J\rvert = 1000$			
		#opt	min	avg	max	#opt	min	avg	max
50	1	9	51.1	83.9	154.1	9	135.5	575.6	1,255.3
50	2	9	120.2	168.7	231.9	9	402.9	1,407.5	2,828.6
50	3	9	129.7	325.4	712.3	9	604.8	2,505.1	4,221.4
50	4	9	224.2	484.9	821.6	9	854.2	3,527.8	6,399.8
50	5	9	291.2	501.4	940.2	9	467.3	3,737.1	7,705.5
100	1	9	166.0	271.6	415.9	9	277.2	1,248.8	1,819.9
100	2	9	349.7	963.1	1,668.8	9	743.4	3,824.2	8,291.3
100	3	9	668.3	1,779.7	3,463.9	9	1,269.7	6,454.0	12,012.2
100	4	9	1,045.7	2,332.1	4,595.1	9	5,776.0	8,624.5	14,586.5
100	5	9	980.4	2,582.8	5,690.8	9	4,294.4	12,308.1	29,031.9

Table 8: CPU time (in seconds) for SSCFLP with interdiction using progressive approximation

$\lvert I\rvert$	Δ	$\lvert J\rvert = 500$				$\lvert J\rvert = 1000$			
		#opt	min	avg	max	#opt	min	avg	max
50	1	9	149.4	573.2	1,604.9	9	182.9	1,177.2	2,899.0
50	2	9	234.4	1,712.8	5,776.7	9	626.2	2,564.8	4,760.2
50	3	9	353.6	2,815.4	11,226.2	9	830.7	5,112.6	11,888.8
50	4	9	1,108.5	4,511.0	9,338.1	9	1,340.8	7,532.5	15,810.2
50	5	9	2,350.7	6,850.9	13,370.8	9	1,492.1	9,626.2	23,622.3
100	1	9	2,044.9	4,572.6	12,517.8	9	2,110.0	4,926.9	8,329.3
100	2	9	3,831.0	9,765.0	23,290.1	9	5,538.9	15,551.8	32,304.7
100	3	7	9,812.2	23,084.3	51,189.4	9	7,022.2	23,890.6	51,788.5
100	4	7	12,961.6	26,452.0	55,934.4	9	17,439.5	34,424.9	60,765.2
100	5	8	18,536.5	43,235.3	79,470.4	8	18,722.2	47,209.1	80,734.3

6 Concluding remarks

In this article we introduce a progressive approximation method for several classes of interdiction games in which two players —namely an attacker and a follower— interact sequentially. The attacker chooses a subset of decision variables from the follower's optimization problem and increases their cost. The follower aims at solving a combinatorial optimization problem to minimize its a cost function that is affected by the attacker's actions. From the attacker's point of view, the objective is to maximize the minimum objective function value achieved by the follower. The proposed progressive approximation framework constructs smaller interdiction games in an iterative fashion, which are proven to yield the optimal solution of the complete game. Under certain conditions, these games are orders of magnitude smaller than the original interdiction game, making problems that could not be handled using conventional techniques tractable. Through an exhaustive computational campaign, we demonstrate that the proposed progressive approximation framework can be useful to solve problems that are orders of magnitude larger than those solved with traditional methods. The new framework is not only effective, but also general as it can handle a vast family of interdiction games.

We identify several potential avenues for future research. First, to study the benefits of embedding the progressive approximation method within a three level Stackelberg game in which follower's decisions also include the protection (or *fortification*) of some assets before the attacker's actions occur. Second, to assess the effectiveness of the progressive approximation scheme for the solution of other classes of interdiction games or bilevel programs. Third, to reduce the sensitivity of the proposed scheme to the sparsity of the follower actions, which as of now is required for the method to be useful in practice.

Appendix

See the online appendix for Tables 9–102.

References

[1] 9th DIMACS implementation challenge - shortest paths. `http://users.diag.uniroma1.it/challenge9/download.shtml`. Accessed: 2019-07-03.

[2] M. A. Acevedo, J. A. Sefair, J. C. Smith, B. Reichert, and R. J. Fletcher Jr. Conservation under uncertainty: Optimal network protection strategies for worst-case disturbance events. Journal of Applied Ecology, 52(6):1588–1597, 2015.

[3] I. Akgün, B. Ç. Tansel, and R. K. Wood. The multi-terminal maximum-flow network-interdiction problem. European Journal of Operational Research, 211(2):241–251, 2011. doi: 10.1016/j.ejor.2010.12.011.

[4] D. Aksen and N. Aras. A bilevel fixed charge location model for facilities under imminent attack. Computers & Operations Research, 39(7):1364–1381, 2012.

[5] D. Aksen and N. Aras. A matheuristic for leader-follower games involving facility location-protection-interdiction decisions. In Metaheuristics for Bi-level Optimization, pages 115–151. Springer, 2013.

[6] D. Aksen, N. Piyade, and N. Aras. The budget constrained r-interdiction median problem with capacity expansion. Central European Journal of Operations Research, 18(3):269–291, 2010.

[7] D. Aloise and C. Contardo. A sampling-based exact algorithm for the solution of the minimax diameter clustering problem. Journal of Global Optimization, 71(3):613–630, 2018.

[8] D. S. Altner, Ö. Ergun, and N. A. Uhan. The maximum flow network interdiction problem: Valid inequalities, integrality gaps, and approximability. Oper. Res. Lett., 38(1):33–38, 2010. doi: 10.1016/j.orl.2009.09.013.

[9] J. F. Bard. Practical bilevel optimization: algorithms and applications, volume 30. Springer Science & Business Media, 2013.

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

[11] H. Bayrak and M. D. Bailey. Shortest path network interdiction with asymmetric information. Networks, 52(3):133–140, 2008. doi: 10.1002/net.20236.

[12] C. Bazgan, S. Toubaline, and D. Vanderpooten. Critical edges/nodes for the minimum spanning tree problem: complexity and approximation. Journal of Combinatorial Optimization, 26(1):178–189, 2013.

[13] V. Beresnev and A. Melnikov. Exact method for the capacitated competitive facility location problem. Computers & Operations Research, 95:73–82, 2018.

[14] D. Bertsimas, E. Nasrabadi, and J. B. Orlin. On the power of randomization in network interdiction. Oper. Res. Lett., 44(1):114–120, 2016. doi: 10.1016/j.orl.2015.11.005.

[15] N. Boland, M. Hewitt, L. Marshall, and M. W. P. Savelsbergh. The continuous-time service network design problem. Operations Research, 65(5):1303–1321, 2017. doi: 10.1287/opre.2017.1624.

[16] J. S. Borrero, O. A. Prokopyev, and D. Sauré. Sequential shortest path interdiction with incomplete information. Decision Analysis, 13(1):68–98, 2016. doi: 10.1287/deca.2015.0325.

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

[18] L. Brotcorne, M. Labbé, P. Marcotte, and G. Savard. Joint design and pricing on a network. Operations research, 56(5):1104–1115, 2008.

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

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

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

[22] A. Caprara, M. Carvalho, A. Lodi, and G. J. Woeginger. Bilevel knapsack with interdiction constraints. INFORMS Journal on Computing, 28(2):319–333, 2016.

[23] C. Casorrán, B. Fortz, M. Labbé, and F. Ordóñez. A study of general and security stackelberg game formulations. European Journal of Operational Research, 278(3):855–868, 2019.

[24] D. Chen and R. Chen. New relaxation-based algorithms for the optimal solution of the continuous and discrete p-center problems. Computers & Operations Research, 36(5):1646–1655, 2009.

[25] L. Chen and G. Zhang. Approximation algorithms for a bi-level knapsack problem. Theoretical Computer Science, 497:1–12, 2013.

[26] S. R. Chestnut and R. Zenklusen. Hardness and approximation for network flow interdiction. Networks, 69(4):378–387, 2017. doi: 10.1002/net.21739.

[27] R. L. Church and M. P. Scaparra. Protecting critical assets: the r-interdiction median problem with fortification. Geographical Analysis, 39(2):129–146, 2007.

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

[29] B. Colson, P. Marcotte, and G. Savard. Bilevel programming: A survey. 4or, 3(2):87–107, 2005.

[30] C. Contardo. Decremental clustering for the solution of p-dispersion problems to proven optimality. Technical Report G–2019–22, GERAD, April 2019. URL https://www.gerad.ca/fr/papers/G-2019-22/view.

[31] C. Contardo, M. Iori, and R. Kramer. A scalable exact algorithm for the vertex p-center problem. Computers & Operations Research, 103:211–220, 2019.

[32] K. J. Cormican, D. P. Morton, and R. K. Wood. Stochastic network interdiction. Operations Research, 46(2):184–197, 1998. doi: 10.1287/opre.46.2.184.

[33] F. D. Croce and R. Scatamacchia. A new exact approach for the bilevel knapsack with interdiction constraints. Technical report, ArXiv preprint, 2018. URL http://arxiv.org/abs/1811.02822.

[34] S. Dempe and S. Franke. On the solution of convex bilevel optimization problems. Computational Optimization and Applications, 63(3):685–703, 2016.

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

[36] S. DeNegre. Interdiction and discrete bilevel linear programming. phdthesis, Lehigh University, Bethlehem, PA, 2011.

[37] S. T. DeNegre and T. K. Ralphs. A branch-and-cut algorithm for integer bilevel linear programs. In Operations research and cyber-infrastructure, pages 65–78. Springer, 2009.

[38] N. B. Dimitrov and D. P. Morton. Interdiction models and applications. In Handbook of Operations Research for Homeland Security, pages 73–103. Springer, 2013.

[39] M. Fischetti, I. Ljubić, and M. Sinnl. Redesigning benders decomposition for large-scale facility location. Management Science, 63(7):2146–2162, 2016.

[40] M. Fischetti, I. Ljubić, M. Monaci, and M. Sinnl. A new general-purpose algorithm for mixed-integer bilevel linear programs. Operations Research, 65(6):1615–1637, 2017.

[41] M. Fischetti, I. Ljubić, M. Monaci, and M. Sinnl. On the use of intersection cuts for bilevel optimization. Mathematical Programming, 172(1–2):77–103, 2018.

[42] G. N. Frederickson and R. Solis-Oba. Increasing the weight of minimum spanning trees. Journal of Algorithms, 33(2):244–266, 1999.

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

[44] S. L. Gadegaard, A. Klose, and L. R. Nielsen. An improved cut-and-solve algorithm for the single-source capacitated facility location problem. EURO Journal on Computational Optimization, 6(1):1–27, 2018.

[45] N. Ghaffarinasab and R. Atayi. An implicit enumeration algorithm for the hub interdiction median problem with fortification. European Journal of Operational Research, 267(1):23–39, 2018.

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

[47] N. Goldberg. Non-zero-sum nonlinear network path interdiction with an application to inspection in terror networks. Naval Research Logistics (NRL), 64(2):139–153, 2017.

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

[49] D. Granata, G. Steeger, and S. Rebennack. Network interdiction via a critical disruption path: Branch-and-price algorithms. Computers & OR, 40(11):2689–2702, 2013. doi: 10.1016/j.cor.2013.04.016.

[50] G. Guastaroba and M. Speranza. A heuristic for bilp problems: The single source capacitated facility location problem. European Journal of Operational Research, 238(2):438–450, 2014. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2014.04.007. URL http://www.sciencedirect.com/science/article/pii/S0377221714003166.

[51] E. Israeli and R. K. Wood. Shortest-path network interdiction. Networks, 40(2):97–111, 2002. doi: 10.1002/net.10039.

[52] K. Kunisch and T. Pock. A bilevel optimization approach for parameter learning in variational models. SIAM Journal on Imaging Sciences, 6(2):938–983, 2013.

[53] M. Labbé, P. Marcotte, and G. Savard. A bilevel model of taxation and its application to optimal highway pricing. Management science, 44(12-part-1):1608–1622, 1998.

[54] F. Liberatore, M. P. Scaparra, and M. S. Daskin. Analysis of facility protection strategies against an uncertain number of attacks: The stochastic r-interdiction median problem with fortification. Computers & Operations Research, 38(1):357–366, 2011.

[55] F. Liberatore, M. P. Scaparra, and M. S. Daskin. Hedging against disruptions with ripple effects in location analysis. Omega, 40(1):21–30, 2012.

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

[57] K.-C. Lin and M.-S. Chern. The most vital edges in the minimum spanning tree problem. Information Processing Letters, 45(1):25–31, 1993.

[58] C. Losada, M. P. Scaparra, R. L. Church, and M. S. Daskin. The stochastic interdiction median problem with disruption intensity levels. Annals of Operations Research, 201(1):345–365, 2012.

[59] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In F. Glover and G. A. Kochenberger, editors, Handbook of Metaheuristics, pages 320–353. Springer US, Boston, MA, 2003.

[60] L. Lozano and J. C. Smith. A backward sampling framework for interdiction problems with fortification. INFORMS Journal on Computing, 29(1):123–139, 2017. doi: 10.1287/ijoc.2016.0721.

[61] L. Lozano and J. C. Smith. A value-function-based exact approach for the bilevel mixed-integer programming problem. Operations Research, 65(3):768–786, 2017. doi: 10.1287/opre.2017.1589.

[62] L. Lozano, J. C. Smith, and M. E. Kurz. Solving the traveling salesman problem with interdiction and fortification. Oper. Res. Lett., 45(3):210–216, 2017. doi: 10.1016/j.orl.2017.02.007.

[63] A. Malaviya, C. Rainwater, and T. Sharkey. Multi-period network interdiction problems with applications to city-level drug enforcement. IIE Transactions, 44(5):368–380, 2012.

[64] N. Matin-Moghaddam and J. Sefair. Route assignment and scheduling with trajectory coordination. Technical report, Arizona State University, September 2018. URL http://www.public.asu.edu/%7Ejsefair/main%20v3.0.pdf.

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

[66] A. Migdalas. Bilevel programming in traffic planning: Models, methods and challenge. Journal of global optimization, 7(4):381–405, 1995.

[67] A. Migdalas, P. M. Pardalos, and P. Värbrand. Multilevel optimization: algorithms and applications, volume 20. Springer Science & Business Media, 2013.

[68] J. T. Moore and J. F. Bard. The mixed integer linear bilevel programming problem. Operations research, 38(5):911–921, 1990.

[69] D. P. Morton. Stochastic network interdiction. Wiley Encyclopedia of Operations Research and Management Science, 2010.

[70] V. Norkin. Optimization models of anti-terrorist protection. Cybernetics and Systems Analysis, 54(6): 918–929, 2018.

[71] J. R. O'Hanley, R. L. Church, and J. K. Gilless. Locating and protecting critical reserve sites to minimize expected and worst-case losses. Biological Conservation, 134(1):130–141, 2007.

[72] M. A. Rad and H. T. Kakhki. Two extended formulations for cardinality maximum flow network interdiction problem. Networks, 69(4):367–377, 2017. doi: 10.1002/net.21732.

[73] A. Raith and M. Ehrgott. A comparison of solution strategies for biobjective shortest path problems. Computers & OR, 36(4):1299–1331, 2009. doi: 10.1016/j.cor.2008.02.002. URL https://doi.org/10.1016/j.cor.2008.02.002.

[74] J. O. Royset and R. K. Wood. Solving the bi-objective maximum-flow network-interdiction problem. INFORMS Journal on Computing, 19(2):175–184, 2007. doi: 10.1287/ijoc.1060.0191.

[75] M. E. H. Sadati, D. Aksen, and N. Aras. The r-interdiction selective multi-depot vehicle routing problem. International Transactions in Operational Research.

[76] S. Sadeghi, A. Seifi, and E. Azizi. Trilevel shortest path network interdiction with partial fortification. Computers & Industrial Engineering, 106:400–411, 2017. doi: 10.1016/j.cie.2017.02.006.

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

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

[79] M. P. Scaparra and R. L. Church. A bilevel mixed-integer program for critical infrastructure protection planning. Computers & Operations Research, 35(6):1905–1923, 2008.

[80] M. P. Scaparra and R. L. Church. An exact solution approach for the interdiction median problem with fortification. European Journal of Operational Research, 189(1):76–92, 2008.

[81] J. A. Sefair and J. C. Smith. Dynamic shortest-path interdiction. Networks, 68(4):315–330, 2016. doi: 10.1002/net.21712.

[82] J. A. Sefair and J. C. Smith. Exact algorithms and bounds for the dynamic assignment interdiction problem. Naval Research Logistics, 64(5):373–387, 2017.

[83] J. A. Sefair, J. C. Smith, M. A. Acevedo, and R. J. Fletcher Jr. A defender-attacker model and algorithm for maximizing weighted expected hitting time with application to conservation planning. IISE Transactions, 49(12):1112–1128, 2017.

[84] J. Smith and Y. Song. A survey of network interdiction models and algorithms. European Journal of Operational Research, 2019.

[85] J. C. Smith. Basic interdiction models. Wiley Encyclopedia of Operations Research and Management Science, 2010.

[86] J. C. Smith, C. Lim, and A. Alptekinoğlu. New product introduction against a predator: A bilevel mixed-integer programming approach. Naval Research Logistics (NRL), 56(8):714–729, 2009.

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

[88] K. M. Sullivan and J. C. Smith. Exact algorithms for solving a euclidean maximum flow network interdiction problem. Networks, 64(2):109–124, 2014. doi: 10.1002/net.21561.

[89] S. Tahernejad, T. K. Ralphs, and S. T. DeNegre. A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation. Technical Report 16T-015-R3, Industrial and Systems Engineering, Lehigh University, 2016.

[90] A. Tsoukalas, B. Rustem, and E. N. Pistikopoulos. A global optimization algorithm for generalized semi-infinite, continuous minimax with coupled constraints and bi-level problems. Journal of Global Optimization, 44(2):235–250, 2009.

[91] H. Tuy, A. Migdalas, and P. Värbrand. A global optimization approach for the linear two-level program. Journal of Global Optimization, 3(1):1–23, 1993.

[92] H. Von Stackelberg. The theory of the market economy. Oxford University Press, 1952.

[93] R. Wollmer. Removing arcs from a network. Operations Research, 12(6):934–940, 1964.

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

[95] A. B. Yaghlane, M. N. Azaiez, and M. Mrad. System survivability in the context of interdiction networks. Reliability Engineering & System Safety, 2019.

[96] D. Yue, J. Gao, B. Zeng, and F. You. A projection-based reformulation and decomposition algorithm for global optimization of a class of mixed integer bilevel linear programs. Journal of Global Optimization, 73(1):27–57, Jan 2019. ISSN 1573-2916. doi: 10.1007/s10898-018-0679-1. URL https://doi.org/10.1007/s10898-018-0679-1.

[97] R. Zenklusen. Matching interdiction. Discrete Applied Mathematics, 158(15):1676–1690, 2010.

[98] R. Zenklusen. Connectivity interdiction. Operations Research Letters, 42(6):450–454, 2014.

[99] Y. Zhang, L. V. Snyder, T. K. Ralphs, and Z. Xue. The competitive facility location problem under disruption risks. Transportation Research Part E: Logistics and Transportation Review, 93:453–473, 2016.

[100] K. Zheng and L. A. Albert. An exact algorithm for solving the bilevel facility interdiction and fortification problem. Operations Research Letters, 46(6):573–578, 2018.