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'.

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

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 $\hat{\lambda}$, $\widehat{\mathbf{y}} = (\widehat{y}_i)_{i \in I}$. Update $UB \leftarrow \hat{\lambda}$.
5: Define $\tilde{\mathbf{y}} = (\tilde{y}_i)_{i=1}^n$, where $\tilde{y}_i = \widehat{y}_i$ 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 $\mathbf{c}^T\mathbf{x} + \mathbf{g}^T\mathbf{w} + \sum_{i=1}^n d_i y_i x_i \leq \mathbf{c}^T\mathbf{x} + \mathbf{g}^T\mathbf{w} + \sum_{i=1}^n d_i x_i$. 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, $\gamma(\mathbf{x}) + \sum_{i=1}^n d_i x_i y_i = \gamma(\mathbf{x}) + \sum_{i \in I(\mathbf{x})} d_i y_i$ 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 $\mathbf{c}^T\mathbf{x} + \mathbf{g}^T\mathbf{w} \leq \mathbf{c}^T\mathbf{x} + \mathbf{g}^T\mathbf{w} + \sum_{i=1}^n d_i y_i x_i$. 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 $I(\tilde{\mathbf{x}}) \not\subseteq I$, 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, $(\mathbf{x}^1, \mathbf{w}^1), \ldots, (\mathbf{x}^H, \mathbf{w}^H)$, 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 9$^{\text{th}}$ 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).

$$\min \quad \sum_{i\in I} f_i y_i + \sum_{i\in I, j\in J} c_{ij} x_{ij} \tag{8}$$

$$\text{s.t.} \quad \sum_{i\in I} x_{ij} = 1, \qquad j \in J \tag{9}$$

$$\sum_{j\in J} q_j x_{ij} \leq K_i y_i, \qquad i \in I \tag{10}$$

$$\mathbf{x} \in \{0,1\}^{|I|\times|J|}, \qquad \mathbf{y} \in \{0,1\}^{|I|}. \tag{11}$$

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

$\|I\|$	Δ	$\|J\| = 500$				$\|J\| = 1000$			
		#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

$\|I\|$	Δ	$\|J\| = 500$				$\|J\| = 1000$			
		#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.

A Detailed results

In this online appendix we report detailed results of our algorithm, this is the lower bound, upper bound, and CPU time takes by the algorithm. Whenever the time limit of 1 day or the memory limit of 16GB are exceeded, we rather report it as TL or ML, respectively. For the former, lower, upper bounds and gaps are reported, while for the latter no log information is available.

A.1 Shortest path interdiction

In this section we report the detailed results of our method on the shortest pat instances considered in our study. The results are grouped by budget size and reportted in Tables 9–26. The results for small budgets ($\Delta \in \{1, 3, 5\}$) are reported in separate tables as those for large budgets ($\Delta \in \{10, 15, 20\}$).

Table 9: Detailed results for road network DC for small budgets

Δ	s	LB	UB	Gap	CPU
1	1	8,349	8,349	0.00	18.9
1	2	4,559	4,559	0.00	19.5
1	3	19,774	19,774	0.00	19.3
1	4	21,575	21,575	0.00	19.6
1	5	10,438	10,438	0.00	19.6
1	6	9,638	9,638	0.00	20.0
1	7	14,028	14,028	0.00	20.0
1	8	10,262	10,262	0.00	19.5
1	9	15,928	15,928	0.00	18.4
3	1	8,990	8,990	0.00	19.4
3	2	4,883	4,883	0.00	19.5
3	3	24,002	24,002	0.00	19.6
3	4	22,167	22,167	0.00	20.6
3	5	11,369	11,369	0.00	20.2
3	6	10,033	10,033	0.00	19.6
3	7	14,937	14,937	0.00	19.8
3	8	10,838	10,838	0.00	20.1
3	9	16,443	16,443	0.00	20.4
5	1	9,394	9,394	0.00	20.2
5	2	5,146	5,146	0.00	20.2
5	3	24,731	24,731	0.00	21.4
5	4	22,592	22,592	0.00	20.3
5	5	12,222	12,222	0.00	20.6
5	6	10,403	10,403	0.00	20.4
5	7	15,367	15,367	0.00	20.2
5	8	11,366	11,366	0.00	19.7
5	9	16,785	16,785	0.00	21.3

Table 10: Detailed results for road network DC for large budgets

Δ	s	LB	UB	Gap	CPU
10	1	10,209	10,209	0.00	20.1
10	2	5,620	5,620	0.00	20.3
10	3	25,683	25,683	0.00	22.9
10	4	23,369	23,369	0.00	21.2
10	5	13,157	13,157	0.00	21.0
10	6	11,220	11,220	0.00	20.0
10	7	16,276	16,276	0.00	20.2
10	8	12,319	12,319	0.00	20.5
10	9	17,447	17,447	0.00	23.3
15	1	10,869	10,869	0.00	20.6
15	2	5,897	5,897	0.00	21.8
15	3	26,198	26,198	0.00	28.0
15	4	24,003	24,003	0.00	22.3
15	5	13,728	13,728	0.00	20.8
15	6	11,893	11,893	0.00	20.3
15	7	17,066	17,066	0.00	20.4
15	8	13,016	13,016	0.00	20.8
15	9	18,028	18,028	0.00	29.5
20	1	11,342	11,342	0.00	21.9
20	2	6,082	6,082	0.00	25.1
20	3	26,607	26,607	0.00	26.7
20	4	24,485	24,485	0.00	23.0
20	5	14,183	14,183	0.00	21.5
20	6	12,464	12,464	0.00	21.3
20	7	17,770	17,770	0.00	20.3
20	8	13,605	13,605	0.00	21.4
20	9	18,506	18,506	0.00	41.0

Table 11: Detailed results for road network RI for small budgets

Δ	s	LB	UB	Gap	CPU
1	1	21,935	21,935	0.00	20.5
1	2	21,175	21,175	0.00	19.9
1	3	49,077	49,077	0.00	20.6
1	4	40,630	40,630	0.00	20.2
1	5	55,838	55,838	0.00	21.2
1	6	33,536	33,536	0.00	20.8
1	7	36,853	36,853	0.00	20.4
1	8	59,960	59,960	0.00	21.7
1	9	21,158	21,158	0.00	20.2
3	1	22,388	22,388	0.00	21.7
3	2	24,649	24,649	0.00	21.1
3	3	53,252	53,252	0.00	23.7
3	4	43,008	43,008	0.00	23.2
3	5	61,145	61,145	0.00	25.1
3	6	35,303	35,303	0.00	23.0
3	7	42,303	42,303	0.00	21.8
3	8	62,102	62,102	0.00	25.2
3	9	23,333	23,333	0.00	22.5
5	1	25,523	25,523	0.00	23.1
5	2	25,488	25,488	0.00	23.5
5	3	57,321	57,321	0.00	25.9
5	4	44,546	44,546	0.00	25.9
5	5	64,646	64,646	0.00	27.9
5	6	36,151	36,151	0.00	25.3
5	7	43,872	43,872	0.00	25.6
5	8	63,368	63,368	0.00	28.2
5	9	24,374	24,374	0.00	24.3

Table 12: Detailed results for road network RI for large budgets

Δ	s	LB	UB	Gap	CPU
10	1	27,407	27,407	0.00	26.3
10	2	26,148	26,148	0.00	26.3
10	3	60,325	60,325	0.00	40.8
10	4	48,401	48,401	0.00	31.9
10	5	68,532	68,532	0.00	44.4
10	6	38,358	38,358	0.00	31.8
10	7	46,712	46,712	0.00	27.0
10	8	66,102	66,102	0.00	94.6
10	9	26,023	26,023	0.00	30.9
15	1	31,037	31,037	0.00	30.5
15	2	26,462	26,462	0.00	27.3
15	3	61,627	61,627	0.00	252.7
15	4	49,833	49,833	0.00	40.3
15	5	70,346	70,346	0.00	276.0
15	6	40,041	40,041	0.00	42.7
15	7	48,191	48,191	0.00	50.5
15	8	67,785	67,785	0.00	232.2
15	9	27,214	27,214	0.00	42.5
20	1	32,113	32,113	0.00	45.3
20	2	26,694	26,694	0.00	33.0
20	3	63,360	63,360	0.00	226.1
20	4	51,115	51,115	0.00	181.2
20	5	71,561	71,561	0.00	630.7
20	6	41,214	41,214	0.00	53.6
20	7	49,159	49,159	0.00	34.3
20	8	69,491	69,491	0.00	638.1
20	9	28,190	28,190	0.00	61.7

Table 13: Detailed results for road network NJ for small budgets

Δ	s	LB	UB	Gap	CPU
1	1	77,813	77,813	0.00	29.8
1	2	83,666	83,666	0.00	31.7
1	3	50,274	50,274	0.00	30.7
1	4	79,280	79,280	0.00	30.7
1	5	65,396	65,396	0.00	29.1
1	6	52,189	52,189	0.00	28.1
1	7	22,578	22,578	0.00	24.4
1	8	39,951	39,951	0.00	27.0
1	9	65,442	65,442	0.00	27.4
3	1	82,919	82,919	0.00	35.5
3	2	93,666	93,666	0.00	60.8
3	3	51,863	51,863	0.00	46.5
3	4	84,356	84,356	0.00	54.4
3	5	69,821	69,821	0.00	33.6
3	6	53,616	53,616	0.00	37.2
3	7	28,329	28,329	0.00	31.7
3	8	44,056	44,056	0.00	33.6
3	9	69,416	69,416	0.00	45.2
5	1	85,480	85,480	0.00	57.2
5	2	99,401	99,401	0.00	64.6
5	3	52,449	52,449	0.00	65.9
5	4	86,135	86,135	0.00	73.6
5	5	72,316	72,316	0.00	47.5
5	6	57,448	57,448	0.00	46.6
5	7	31,059	31,059	0.00	41.7
5	8	45,260	45,260	0.00	40.1
5	9	71,113	71,113	0.00	53.0

Table 14: Detailed results for road network NJ for large budgets

Δ	s	LB	UB	Gap	CPU
10	1	91,831	91,831	0.00	77.4
10	2	105,113	105,113	0.00	202.0
10	3	56,063	56,063	0.00	196.8
10	4	89,778	89,778	0.00	373.3
10	5	78,050	78,050	0.00	125.2
10	6	61,334	61,334	0.00	123.7
10	7	33,130	33,130	0.00	61.5
10	8	47,604	47,604	0.00	91.4
10	9	76,754	76,754	0.00	112.3
15	1	95,172	95,172	0.00	372.2
15	2	109,198	109,198	0.00	841.1
15	3	59,262	59,262	0.00	485.5
15	4	92,792	92,792	0.00	986.0
15	5	82,891	82,891	0.00	234.7
15	6	63,391	63,391	0.00	210.4
15	7	33,858	33,858	0.00	88.3
15	8	49,598	49,598	0.00	179.8
15	9	79,911	79,911	0.00	339.1
20	1	96,910	96,910	0.00	421.3
20	2	111,090	111,090	0.00	4,493.3
20	3	61,546	61,546	0.00	785.7
20	4	95,110	95,110	0.00	3,786.5
20	5	85,345	85,345	0.00	454.7
20	6	64,862	64,862	0.00	425.7
20	7	34,570	34,570	0.00	130.8
20	8	50,697	50,697	0.00	164.9
20	9	81,970	81,970	0.00	1,059.4

Table 15: Detailed results for road network NY for small budgets

Δ	s	LB	UB	Gap	CPU
1	1	175,225	175,225	0.00	65.5
1	2	80,937	80,937	0.00	44.1
1	3	84,957	84,957	0.00	53.8
1	4	97,566	97,566	0.00	53.7
1	5	113,724	113,724	0.00	46.2
1	6	79,085	79,085	0.00	58.1
1	7	74,401	74,401	0.00	42.2
1	8	114,827	114,827	0.00	51.8
1	9	131,558	131,558	0.00	65.2
3	1	183,534	183,534	0.00	103.1
3	2	87,183	87,183	0.00	85.9
3	3	90,262	90,262	0.00	68.3
3	4	105,954	105,954	0.00	90.1
3	5	123,724	123,724	0.00	57.6
3	6	81,198	81,198	0.00	91.9
3	7	84,401	84,401	0.00	84.5
3	8	124,827	124,827	0.00	91.5
3	9	138,156	138,156	0.00	106.6
5	1	187,873	187,873	0.00	139.6
5	2	92,753	92,753	0.00	123.4
5	3	93,055	93,055	0.00	93.4
5	4	108,713	108,713	0.00	100.2
5	5	130,287	130,287	0.00	82.4
5	6	82,515	82,515	0.00	128.0
5	7	91,439	91,439	0.00	101.9
5	8	128,258	128,258	0.00	130.7
5	9	143,676	143,676	0.00	127.5

Table 16: Detailed results for road network NY for large budgets

Δ	s	LB	UB	Gap	CPU
10	1	196,082	196,082	0.00	1,053.6
10	2	99,829	99,829	0.00	379.7
10	3	98,901	98,901	0.00	203.2
10	4	113,631	113,631	0.00	189.9
10	5	143,724	143,724	0.00	110.9
10	6	87,457	87,457	0.00	554.9
10	7	97,226	97,226	0.00	153.3
10	8	133,675	133,675	0.00	639.9
10	9	151,458	151,458	0.00	747.2
15	1	202,008	202,008	0.00	2,581.2
15	2	106,431	106,431	0.00	655.9
15	3	102,614	102,614	0.00	453.0
15	4	117,866	117,866	0.00	385.8
15	5	148,721	148,721	0.00	818.9
15	6	91,198	91,198	0.00	1,243.1
15	7	99,745	99,745	0.00	438.7
15	8	140,560	140,560	0.00	794.8
15	9	154,341	154,341	0.00	6,917.2
20	1	204,989	206,996	0.98	TL
20	2	109,453	109,453	0.00	1,272.6
20	3	104,677	104,677	0.00	713.3
20	4	121,505	121,505	0.00	604.5
20	5	152,981	152,981	0.00	2,514.6
20	6	93,747	93,759	0.10	TL
20	7	101,126	101,126	0.00	3,729.9
20	8	145,622	145,622	0.00	729.2
20	9	156,736	158,446	1.90	TL

Table 17: Detailed results for road network CA for small budgets

Δ	s	LB	UB	Gap	CPU
1	1	136,846	136,846	0.00	128.8
1	2	71,710	71,710	0.00	93.0
1	3	141,670	141,670	0.00	103.4
1	4	102,215	102,215	0.00	102.0
1	5	72,666	72,666	0.00	92.1
1	6	106,241	106,241	0.00	118.1
1	7	127,194	127,194	0.00	111.7
1	8	43,223	43,223	0.00	106.0
1	9	110,300	110,300	0.00	104.6
3	1	137,628	137,628	0.00	240.5
3	2	76,297	76,297	0.00	151.6
3	3	151,484	151,484	0.00	188.7
3	4	114,799	114,799	0.00	92.0
3	5	82,666	82,666	0.00	178.3
3	6	116,241	116,241	0.00	152.6
3	7	139,424	139,424	0.00	211.2
3	8	43,996	43,996	0.00	189.3
3	9	113,095	113,095	0.00	209.9
5	1	140,537	140,537	0.00	370.9
5	2	79,505	79,505	0.00	290.5
5	3	153,733	153,733	0.00	247.1
5	4	124,799	124,799	0.00	194.4
5	5	86,896	86,896	0.00	260.7
5	6	120,158	120,158	0.00	192.0
5	7	144,158	144,158	0.00	273.3
5	8	44,688	44,688	0.00	228.4
5	9	118,176	118,176	0.00	325.8

Table 18: Detailed results for road network CA for large budgets

Δ	s	LB	UB	Gap	CPU
10	1	147,628	147,628	0.00	1,023.5
10	2	84,821	84,821	0.00	455.0
10	3	161,484	161,484	0.00	1,157.8
10	4	135,862	135,862	0.00	226.6
10	5	94,665	94,665	0.00	358.3
10	6	126,316	126,316	0.00	541.8
10	7	153,852	153,852	0.00	731.1
10	8	45,841	45,841	0.00	321.3
10	9	123,669	123,669	0.00	864.8
15	1	152,818	152,818	0.00	2,590.3
15	2	87,581	87,581	0.00	830.4
15	3	165,612	165,612	0.00	1,363.8
15	4	138,792	138,792	0.00	488.8
15	5	99,159	99,159	0.00	588.9
15	6	130,846	130,846	0.00	2,014.8
15	7	159,407	159,407	0.00	1,646.0
15	8	47,153	47,153	0.00	491.0
15	9	129,270	129,270	0.00	1,695.7
20	1	155,602	155,657	0.40	TL
20	2	89,006	89,006	0.00	1,857.9
20	3	169,687	169,687	0.00	5,271.3
20	4	141,414	141,414	0.00	801.2
20	5	102,132	102,132	0.00	1,290.5
20	6	134,401	134,401	0.00	11,850.0
20	7	162,027	162,027	0.00	22,522.0
20	8	48,378	48,378	0.00	452.4
20	9	132,098	132,099	0.00	TL

Table 19: Detailed results for road network FL for small budgets

Δ	s	LB	UB	Gap	CPU
1	1	106,366	106,366	0.00	70.3
1	2	41,519	41,519	0.00	55.6
1	3	100,298	100,298	0.00	78.2
1	4	42,619	42,619	0.00	53.9
1	5	89,912	89,912	0.00	63.4
1	6	123,956	123,956	0.00	82.5
1	7	107,570	107,570	0.00	77.8
1	8	126,256	126,256	0.00	82.9
1	9	26,265	26,265	0.00	52.0
3	1	112,532	112,532	0.00	185.4
3	2	45,249	45,249	0.00	58.0
3	3	106,836	106,836	0.00	113.0
3	4	43,938	43,938	0.00	83.2
3	5	94,429	94,429	0.00	137.6
3	6	129,784	129,784	0.00	177.2
3	7	113,473	113,473	0.00	171.5
3	8	131,371	131,371	0.00	168.9
3	9	27,763	27,763	0.00	100.5
5	1	119,703	119,703	0.00	214.6
5	2	45,928	45,928	0.00	114.4
5	3	112,016	112,016	0.00	198.2
5	4	45,366	45,366	0.00	90.8
5	5	99,327	99,327	0.00	169.3
5	6	132,598	132,598	0.00	225.1
5	7	119,235	119,235	0.00	149.1
5	8	138,038	138,038	0.00	195.6
5	9	28,621	28,621	0.00	135.7

Table 20: Detailed results for road network FL for large budgets

Δ	s	LB	UB	Gap	CPU
10	1	126,006	126,006	0.00	771.5
10	2	48,971	48,971	0.00	208.1
10	3	117,904	117,904	0.00	550.1
10	4	47,875	47,875	0.00	131.5
10	5	104,672	104,672	0.00	527.5
10	6	138,444	138,444	0.00	1,079.4
10	7	125,125	125,125	0.00	447.0
10	8	147,658	147,658	0.00	403.3
10	9	30,934	30,934	0.00	196.5
15	1	130,224	130,224	0.00	2,439.2
15	2	51,667	51,667	0.00	284.7
15	3	123,265	123,265	0.00	1,670.7
15	4	49,014	49,014	0.00	124.7
15	5	107,805	107,805	0.00	2,260.8
15	6	142,062	142,062	0.00	4,531.6
15	7	129,098	129,098	0.00	1,235.0
15	8	152,851	152,851	0.00	1,014.7
15	9	31,950	31,950	0.00	323.7
20	1	134,457	134,457	0.00	42,592.0
20	2	52,566	52,566	0.00	337.4
20	3	126,521	126,521	0.00	4,681.5
20	4	50,126	50,126	0.00	135.2
20	5	110,581	110,581	0.00	12,665.4
20	6	145,560	145,599	0.30	TL
20	7	131,168	131,168	0.00	7,066.6
20	8	156,409	156,409	0.00	9,912.6
20	9	32,650	32,650	0.00	279.5

Table 21: Detailed results for road network TX for small budgets

Δ	s	LB	UB	Gap	CPU
1	1	152,895	152,895	0.00	134.0
1	2	62,247	62,247	0.00	78.0
1	3	171,331	171,331	0.00	136.9
1	4	154,008	154,008	0.00	93.0
1	5	185,548	185,548	0.00	132.2
1	6	105,812	105,812	0.00	121.8
1	7	143,700	143,700	0.00	139.8
1	8	67,570	67,570	0.00	83.2
1	9	182,579	182,579	0.00	116.9
3	1	169,929	169,929	0.00	123.7
3	2	68,271	68,271	0.00	198.1
3	3	183,220	183,220	0.00	174.5
3	4	165,306	165,306	0.00	112.3
3	5	205,548	205,548	0.00	135.6
3	6	115,812	115,812	0.00	207.6
3	7	148,847	148,847	0.00	198.4
3	8	75,765	75,765	0.00	128.9
3	9	186,332	186,332	0.00	305.4
5	1	177,628	177,628	0.00	266.5
5	2	69,612	69,612	0.00	366.5
5	3	193,220	193,220	0.00	277.0
5	4	175,306	175,306	0.00	182.1
5	5	219,536	219,536	0.00	250.8
5	6	123,809	123,809	0.00	249.7
5	7	153,700	153,700	0.00	274.9
5	8	78,974	78,974	0.00	237.7
5	9	192,579	192,579	0.00	312.9

Table 22: Detailed results for road network TX for large budgets

Δ	s	LB	UB	Gap	CPU
10	1	191,369	191,369	0.00	328.6
10	2	76,920	76,920	0.00	546.3
10	3	203,280	203,280	0.00	720.7
10	4	187,317	187,317	0.00	410.4
10	5	232,314	232,314	0.00	349.3
10	6	136,651	136,651	0.00	304.2
10	7	163,888	163,888	0.00	995.3
10	8	86,107	86,107	0.00	363.8
10	9	200,801	200,801	0.00	599.0
15	1	198,556	198,556	0.00	867.0
15	2	79,200	79,200	0.00	941.2
15	3	209,228	209,228	0.00	6,196.7
15	4	192,154	192,154	0.00	1,185.5
15	5	239,665	239,665	0.00	966.0
15	6	143,364	143,364	0.00	389.2
15	7	172,270	172,270	0.00	771.9
15	8	88,561	88,561	0.00	645.8
15	9	205,166	205,179	0.10	TL
20	1	201,474	201,474	0.00	12,185.6
20	2	81,959	81,959	0.00	1,153.5
20	3	214,308	214,331	0.10	TL
20	4	199,114	199,114	0.00	3,835.1
20	5	246,356	246,356	0.00	4,698.6
20	6	147,412	147,412	0.00	738.0
20	7	177,083	177,083	0.00	1,710.0
20	8	90,771	90,771	0.00	496.9
20	9	212,632	212,632	0.00	2,118.4

Table 23: Detailed results for road network USE for small budgets

Δ	s	LB	UB	Gap	CPU
1	1	174,195	174,195	0.00	168.0
1	2	303,889	303,889	0.00	146.1
1	3	247,938	247,938	0.00	278.3
1	4	166,443	166,443	0.00	141.1
1	5	282,113	282,113	0.00	230.9
1	6	198,639	198,639	0.00	167.6
1	7	260,480	260,480	0.00	127.4
1	8	358,531	358,531	0.00	147.3
1	9	320,375	320,375	0.00	141.6
3	1	183,930	183,930	0.00	236.4
3	2	317,614	317,614	0.00	309.8
3	3	255,613	255,613	0.00	234.8
3	4	173,069	173,069	0.00	245.9
3	5	302,113	302,113	0.00	291.6
3	6	204,614	204,614	0.00	307.0
3	7	274,093	274,093	0.00	352.6
3	8	369,053	369,053	0.00	325.5
3	9	340,375	340,375	0.00	354.0
5	1	190,440	190,440	0.00	481.7
5	2	329,364	329,364	0.00	407.4
5	3	265,956	265,956	0.00	654.3
5	4	181,665	181,665	0.00	363.9
5	5	311,958	311,958	0.00	439.7
5	6	209,719	209,719	0.00	568.9
5	7	284,093	284,093	0.00	425.8
5	8	379,053	379,053	0.00	902.6
5	9	355,599	355,599	0.00	434.7

Table 24: Detailed results for road network USE for large budgets

Δ	s	LB	UB	Gap	CPU
10	1	202,798	202,798	0.00	1,331.3
10	2	349,253	349,253	0.00	1,096.1
10	3	283,452	283,452	0.00	1,354.2
10	4	196,203	196,203	0.00	952.8
10	5	331,713	331,713	0.00	694.3
10	6	220,384	220,384	0.00	1,983.4
10	7	303,014	303,014	0.00	929.5
10	8	396,066	396,066	0.00	1,242.6
10	9	379,405	379,405	0.00	780.6
15	1	208,913	209,095	0.90	TL
15	2	362,338	362,338	0.00	2,657.6
15	3	298,389	298,389	0.00	3,007.5
15	4	209,605	209,605	0.00	1,473.2
15	5	341,713	341,713	0.00	1,340.0
15	6	226,662	226,685	0.10	TL
15	7	311,262	311,262	0.00	7,596.1
15	8	406,645	406,645	0.00	43,186.2
15	9	395,609	395,609	0.00	3,904.0
20	1	210,486	217,514	3.34	TL
20	2	ML	ML	ML	ML
20	3	307,382	310,262	0.94	TL
20	4	217,986	217,986	0.00	5,152.5
20	5	351,065	351,065	0.00	20,974.7
20	6	234,231	234,241	0.00	TL
20	7	315,460	320,168	1.49	TL
20	8	414,769	417,956	0.77	TL
20	9	393,692	413,764	5.10	TL

Table 25: Detailed results for road network USW for small budgets

Δ	s	LB	UB	Gap	CPU
1	1	391,616	391,616	0.00	351.7
1	2	297,824	297,824	0.00	232.7
1	3	349,856	349,856	0.00	309.2
1	4	132,629	132,629	0.00	192.4
1	5	291,721	291,721	0.00	387.1
1	6	151,992	151,992	0.00	246.9
1	7	291,890	291,890	0.00	393.0
1	8	332,881	332,881	0.00	239.4
1	9	497,970	497,970	0.00	492.9
3	1	401,616	401,616	0.00	972.1
3	2	317,824	317,824	0.00	556.0
3	3	361,869	361,869	0.00	429.0
3	4	142,629	142,629	0.00	427.2
3	5	305,088	305,088	0.00	681.8
3	6	161,992	161,992	0.00	622.3
3	7	308,250	308,250	0.00	574.6
3	8	352,881	352,881	0.00	359.8
3	9	517,970	517,970	0.00	675.4
5	1	411,616	411,616	0.00	899.3
5	2	337,824	337,824	0.00	653.5
5	3	371,869	371,869	0.00	1,185.1
5	4	147,786	147,786	0.00	769.2
5	5	319,391	319,391	0.00	1,006.4
5	6	170,498	170,498	0.00	928.4
5	7	318,250	318,250	0.00	683.7
5	8	362,881	362,881	0.00	1,060.3
5	9	529,710	529,710	0.00	1,051.9

Table 26: Detailed results for road network USW for large budgets

Δ	s	LB	UB	Gap	CPU
10	1	429,797	429,797	0.00	2,079.2
10	2	380,010	380,010	0.00	1,062.1
10	3	395,213	395,213	0.00	2,003.4
10	4	167,734	167,734	0.00	1,443.4
10	5	344,186	344,186	0.00	1,622.7
10	6	182,314	182,314	0.00	2,301.5
10	7	341,431	341,431	0.00	1,512.1
10	8	383,612	383,612	0.00	1,307.8
10	9	555,972	555,972	0.00	6,312.6
15	1	445,719	445,719	0.00	2,929.9
15	2	404,890	404,890	0.00	1,319.2
15	3	412,928	412,928	0.00	3,776.6
15	4	178,184	178,184	0.00	2,787.8
15	5	361,645	361,645	0.00	3,067.6
15	6	192,712	192,712	0.00	3,548.5
15	7	354,651	354,651	0.00	1,914.1
15	8	395,638	395,638	0.00	7,793.4
15	9	ML	ML	ML	ML
20	1	457,266	457,661	0.90	TL
20	2	422,084	422,084	0.00	4,013.4
20	3	427,443	427,631	0.40	TL
20	4	186,594	186,594	0.00	28,692.2
20	5	373,758	373,758	0.00	7,956.0
20	6	198,901	200,216	0.66	TL
20	7	370,476	370,476	0.00	3,098.0
20	8	410,254	410,254	0.00	5,728.4
20	9	571,123	590,648	3.42	TL

A.2 0-1 Knapsack interdiction

In this section, we report detailed results of our method on the 0-1 knapsack instances considered in this study. The results are given separately for every value of $N, \kappa, \alpha^f, \alpha^l$. The results for the 9 problems generated for every choice of these parameters are given in Tables 27–62.

Table 27: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.125$, $\alpha^l = 5$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	60,896	60,896	0.00	28.2
2	61,037	61,037	0.00	25.3
3	59,636	59,636	0.00	27.4
4	57,959	57,959	0.00	26.0
5	63,405	63,405	0.00	26.6
6	64,231	64,231	0.00	35.6
7	60,746	60,746	0.00	25.6
8	65,931	65,931	0.00	27.6
9	62,295	62,295	0.00	23.3

Table 28: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.125$, $\alpha^l = 5$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	103,150	103,150	0.00	50.6
2	107,638	107,638	0.00	49.5
3	105,588	105,588	0.00	51.2
4	106,232	106,232	0.00	35.9
5	107,606	107,606	0.00	42.1
6	101,920	101,920	0.00	32.7
7	111,386	111,386	0.00	42.8
8	101,571	101,571	0.00	42.8
9	101,765	101,765	0.00	58.9

Table 29: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.125$, $\alpha^l = 10$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	62,201	62,201	0.00	32.9
2	59,757	59,757	0.00	29.9
3	61,239	61,239	0.00	43.2
4	60,787	60,787	0.00	33.1
5	62,630	62,630	0.00	50.9
6	62,048	62,048	0.00	36.7
7	58,835	58,835	0.00	38.1
8	62,011	62,011	0.00	31.2
9	63,219	63,219	0.00	29.6

Table 30: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.125$, $\alpha^l = 10$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	105,844	105,844	0.00	163.8
2	102,603	102,603	0.00	58.4
3	101,905	101,905	0.00	58.6
4	100,884	100,884	0.00	90.3
5	104,213	104,213	0.00	67.1
6	106,431	106,431	0.00	161.1
7	107,572	107,572	0.00	83.0
8	104,299	104,299	0.00	141.2
9	101,258	101,258	0.00	124.9

Table 31: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.125$, $\alpha^l = 20$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	61,405	61,405	0.00	67.5
2	58,416	58,416	0.00	47.3
3	59,080	59,080	0.00	132.1
4	62,893	62,893	0.00	89.0
5	63,927	63,927	0.00	39.9
6	59,992	59,992	0.00	73.8
7	60,234	60,234	0.00	43.6
8	62,360	62,360	0.00	44.7
9	57,273	57,273	0.00	62.9

Table 32: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.125$, $\alpha^l = 20$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	100,062	100,062	0.00	239.3
2	108,966	108,966	0.00	360.2
3	105,324	105,324	0.00	161.4
4	102,745	102,745	0.00	307.4
5	108,363	108,363	0.00	171.2
6	102,292	102,292	0.00	249.0
7	104,403	104,403	0.00	265.0
8	100,114	100,114	0.00	158.0
9	103,654	103,654	0.00	462.1

Table 33: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.25$, $\alpha^l = 5$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	56,857	56,857	0.00	30.6
2	60,949	60,949	0.00	25.0
3	58,435	58,435	0.00	28.4
4	55,561	55,561	0.00	27.5
5	63,921	63,921	0.00	26.5
6	59,146	59,146	0.00	35.8
7	61,213	61,213	0.00	28.4
8	61,131	61,131	0.00	32.3
9	63,511	63,511	0.00	29.2

Table 34: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.25$, $\alpha^l = 5$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	102,905	102,905	0.00	76.6
2	105,931	105,931	0.00	55.8
3	104,949	104,949	0.00	39.4
4	104,810	104,810	0.00	42.4
5	100,545	100,545	0.00	54.5
6	101,264	101,264	0.00	73.5
7	100,055	100,055	0.00	40.1
8	106,178	106,178	0.00	70.0
9	99,505	99,505	0.00	91.7

Table 35: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.25$, $\alpha^l = 10$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	59,135	59,135	0.00	35.5
2	65,893	65,893	0.00	40.4
3	63,561	63,561	0.00	31.9
4	57,975	57,975	0.00	38.9
5	64,456	64,456	0.00	35.2
6	62,467	62,467	0.00	53.5
7	64,238	64,238	0.00	34.4
8	60,033	60,033	0.00	48.9
9	64,081	64,081	0.00	61.5

Table 36: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.25$, $\alpha^l = 10$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	103,259	103,259	0.00	199.5
2	100,406	100,406	0.00	133.5
3	100,335	100,335	0.00	504.8
4	98,992	98,992	0.00	300.2
5	100,254	100,254	0.00	107.5
6	101,075	101,075	0.00	127.4
7	96,748	96,748	0.00	76.9
8	96,467	96,467	0.00	214.2
9	99,234	99,234	0.00	83.7

Table 37: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.25$, $\alpha^l = 20$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	55,344	55,344	0.00	247.8
2	58,843	58,843	0.00	53.6
3	59,925	59,925	0.00	152.3
4	58,058	58,058	0.00	110.4
5	57,914	57,914	0.00	657.8
6	61,660	61,660	0.00	123.6
7	60,089	60,089	0.00	82.5
8	59,154	59,154	0.00	204.1
9	60,693	60,693	0.00	143.8

Table 38: Results for 0-1 knapsack instance with N = 10000, $\kappa = 0.25$, $\alpha^l = 20$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	99,287	99,287	0.00	1,421.8
2	100,384	100,384	0.00	740.3
3	101,551	101,551	0.00	728.6
4	106,355	106,355	0.00	1,189.2
5	100,166	100,169	0.00	TL
6	100,796	100,798	0.00	TL
7	98,441	98,441	0.00	999.8
8	97,151	97,151	0.00	190.6
9	92,534	92,534	0.00	315.4

Table 39: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.125$, $\alpha^l = 5$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	140,060	140,060	0.00	53.0
2	140,749	140,749	0.00	59.4
3	147,956	147,956	0.00	58.3
4	144,451	144,451	0.00	54.8
5	131,698	131,698	0.00	58.5
6	135,021	135,021	0.00	56.2
7	144,009	144,009	0.00	57.2
8	144,590	144,590	0.00	46.8
9	139,361	139,361	0.00	51.7

Table 40: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.125$, $\alpha^l = 5$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	214,823	214,823	0.00	76.5
2	208,238	208,238	0.00	64.5
3	217,726	217,726	0.00	154.7
4	222,164	222,164	0.00	76.5
5	214,099	214,099	0.00	92.5
6	199,284	199,284	0.00	68.8
7	215,593	215,593	0.00	80.5
8	215,440	215,440	0.00	137.8
9	213,736	213,736	0.00	79.4

Table 41: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.125$, $\alpha^l = 10$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	139,619	139,619	0.00	78.1
2	134,010	134,010	0.00	68.7
3	142,692	142,692	0.00	72.5
4	141,291	141,291	0.00	72.7
5	149,119	149,119	0.00	68.9
6	148,163	148,163	0.00	108.1
7	147,292	147,292	0.00	67.2
8	141,382	141,382	0.00	61.7
9	140,828	140,828	0.00	75.6

Table 42: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.125$, $\alpha^l = 10$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	213,225	213,225	0.00	160.0
2	204,291	204,291	0.00	341.8
3	215,264	215,264	0.00	367.1
4	220,989	220,989	0.00	484.4
5	218,539	218,539	0.00	115.4
6	220,269	220,269	0.00	144.3
7	216,189	216,189	0.00	114.3
8	220,249	220,249	0.00	185.4
9	211,341	211,341	0.00	130.3

Table 43: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.125$, $\alpha^l = 20$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	141,408	141,408	0.00	169.0
2	154,594	154,594	0.00	262.2
3	139,519	139,519	0.00	133.6
4	141,317	141,317	0.00	212.9
5	145,990	145,990	0.00	158.7
6	139,568	139,568	0.00	348.9
7	140,685	140,685	0.00	167.5
8	148,522	148,522	0.00	116.6
9	138,618	138,618	0.00	144.4

Table 44: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.125$, $\alpha^l = 20$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	206,416	206,416	0.00	251.9
2	218,242	218,242	0.00	612.3
3	203,238	203,238	0.00	233.6
4	210,597	210,597	0.00	1,232.4
5	205,447	205,447	0.00	354.5
6	207,856	207,856	0.00	865.6
7	211,134	211,134	0.00	230.8
8	216,618	216,618	0.00	536.9
9	205,618	205,618	0.00	1,543.3

Table 45: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.25$, $\alpha^l = 5$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	132,747	132,747	0.00	44.7
2	135,946	135,946	0.00	49.4
3	140,686	140,686	0.00	49.3
4	139,239	139,239	0.00	83.3
5	141,921	141,921	0.00	49.9
6	138,655	138,655	0.00	64.6
7	137,429	137,429	0.00	47.5
8	143,028	143,028	0.00	59.1
9	137,592	137,592	0.00	43.3

Table 46: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.25$, $\alpha^l = 5$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	215,844	215,844	0.00	199.0
2	218,653	218,653	0.00	92.9
3	200,480	200,480	0.00	118.7
4	213,602	213,602	0.00	166.3
5	210,844	210,844	0.00	82.8
6	218,694	218,694	0.00	114.2
7	216,168	216,168	0.00	81.5
8	224,094	224,094	0.00	108.8
9	215,088	215,088	0.00	128.8

Table 47: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.25$, $\alpha^l = 10$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	132,071	132,071	0.00	80.3
2	131,662	131,662	0.00	96.1
3	147,317	147,317	0.00	97.3
4	141,610	141,610	0.00	88.2
5	139,800	139,800	0.00	146.5
6	150,124	150,124	0.00	100.9
7	137,928	137,928	0.00	93.1
8	146,551	146,551	0.00	77.3
9	145,486	145,486	0.00	89.4

Table 48: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.25$, $\alpha^l = 10$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	211,177	211,177	0.00	217.3
2	203,481	203,481	0.00	487.0
3	215,158	215,158	0.00	247.3
4	222,308	222,308	0.00	149.9
5	200,625	200,625	0.00	551.8
6	213,710	213,710	0.00	266.8
7	207,824	207,824	0.00	242.6
8	212,851	212,851	0.00	631.7
9	214,580	214,580	0.00	223.4

Table 49: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.25$, $\alpha^l = 20$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	141,348	141,348	0.00	205.4
2	136,483	136,483	0.00	275.3
3	135,975	135,975	0.00	695.1
4	143,899	143,899	0.00	178.5
5	137,323	137,323	0.00	319.3
6	136,102	136,102	0.00	223.1
7	145,754	145,754	0.00	184.3
8	133,013	133,013	0.00	380.5
9	151,758	151,758	0.00	196.4

Table 50: Results for 0-1 knapsack instance with N = 100000, $\kappa = 0.25$, $\alpha^l = 20$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	212,819	212,819	0.00	5,824.0
2	213,382	213,382	0.00	1,210.0
3	220,149	220,149	0.00	2,330.8
4	217,938	217,938	0.00	1,537.2
5	213,822	213,822	0.00	2,322.1
6	205,095	205,095	0.00	1,692.9
7	216,475	216,475	0.00	1,079.7
8	204,269	204,269	0.00	680.8
9	211,911	211,911	0.00	877.7

Table 51: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.125$, $\alpha^l = 5$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	449,131	449,131	0.00	218.6
2	461,727	461,727	0.00	232.9
3	460,979	460,979	0.00	196.7
4	480,508	480,508	0.00	267.3
5	476,949	476,949	0.00	222.1
6	471,821	471,821	0.00	293.0
7	462,813	462,813	0.00	264.6
8	444,178	444,178	0.00	226.6
9	471,440	471,440	0.00	245.5

Table 52: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.125$, $\alpha^l = 5$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	607,598	607,598	0.00	314.9
2	627,920	627,920	0.00	304.8
3	620,674	620,674	0.00	432.1
4	645,212	645,212	0.00	441.6
5	619,161	619,161	0.00	281.5
6	637,675	637,675	0.00	480.9
7	629,186	629,186	0.00	436.1
8	635,042	635,042	0.00	261.4
9	632,482	632,482	0.00	351.4

Table 53: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.125$, $\alpha^l = 10$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	465,407	465,407	0.00	427.5
2	484,464	484,464	0.00	433.9
3	481,724	481,724	0.00	509.1
4	462,348	462,348	0.00	287.2
5	480,426	480,426	0.00	1,202.2
6	469,724	469,724	0.00	442.4
7	479,074	479,074	0.00	433.4
8	465,615	465,615	0.00	295.4
9	467,571	467,571	0.00	304.0

Table 54: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.125$, $\alpha^l = 10$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	642,016	642,016	0.00	913.4
2	651,265	651,265	0.00	569.5
3	633,880	633,880	0.00	460.7
4	623,772	623,772	0.00	376.4
5	656,737	656,737	0.00	661.8
6	637,635	637,635	0.00	939.8
7	614,466	614,466	0.00	703.6
8	618,463	618,463	0.00	540.8
9	614,508	614,508	0.00	677.2

Table 55: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.125$, $\alpha^l = 20$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	449,575	449,575	0.00	700.4
2	483,146	483,146	0.00	945.2
3	459,556	459,556	0.00	740.8
4	467,401	467,401	0.00	640.4
5	451,916	451,916	0.00	1,063.7
6	481,795	481,795	0.00	633.0
7	461,210	461,210	0.00	417.8
8	486,720	486,720	0.00	1,350.4
9	457,971	457,971	0.00	1,325.4

Table 56: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.125$, $\alpha^l = 20$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	627,525	627,525	0.00	1,876.4
2	607,355	607,355	0.00	2,948.2
3	626,309	626,309	0.00	2,017.8
4	663,047	663,047	0.00	1,584.2
5	617,285	617,285	0.00	1,520.9
6	601,189	601,189	0.00	1,721.7
7	621,825	621,825	0.00	1,356.0
8	599,561	599,561	0.00	2,848.3
9	619,260	619,260	0.00	1,710.8

Table 57: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.25$, $\alpha^l = 5$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	461,314	461,314	0.00	335.6
2	467,614	467,614	0.00	230.6
3	458,924	458,924	0.00	231.3
4	469,911	469,911	0.00	244.3
5	481,566	481,566	0.00	241.8
6	466,905	466,905	0.00	351.6
7	485,951	485,951	0.00	310.8
8	464,944	464,944	0.00	239.2
9	489,914	489,914	0.00	168.1

Table 58: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.25$, $\alpha^l = 5$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	611,258	611,258	0.00	714.2
2	620,324	620,324	0.00	816.1
3	617,994	617,994	0.00	499.1
4	621,540	621,540	0.00	1,345.9
5	635,260	635,260	0.00	350.8
6	605,841	605,841	0.00	445.3
7	634,493	634,493	0.00	382.0
8	622,171	622,171	0.00	367.2
9	630,532	630,532	0.00	389.7

Table 59: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.25$, $\alpha^l = 10$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	463,187	463,187	0.00	1,272.5
2	458,985	458,985	0.00	1,412.8
3	457,749	457,749	0.00	624.2
4	481,440	481,440	0.00	452.3
5	460,927	460,927	0.00	327.5
6	457,200	457,200	0.00	714.0
7	489,749	489,749	0.00	388.9
8	480,903	480,903	0.00	357.2
9	477,564	477,564	0.00	440.1

Table 60: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.25$, $\alpha^l = 10$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	614,986	614,986	0.00	923.5
2	611,189	611,189	0.00	1,112.2
3	625,453	625,453	0.00	764.3
4	613,827	613,827	0.00	1,461.9
5	613,341	613,341	0.00	759.4
6	605,321	605,321	0.00	868.2
7	605,458	605,458	0.00	1,376.8
8	638,763	638,763	0.00	6,883.8
9	627,978	627,978	0.00	1,005.2

Table 61: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.25$, $\alpha^l = 20$, $\alpha^f = 20$

s	LB	UB	Gap	CPU
1	470,074	470,074	0.00	3,396.2
2	455,770	455,770	0.00	3,053.3
3	473,507	473,507	0.00	1,175.2
4	477,873	477,873	0.00	1,175.4
5	459,963	459,963	0.00	2,496.7
6	454,724	454,724	0.00	3,302.3
7	457,526	457,526	0.00	689.8
8	470,765	470,765	0.00	921.3
9	462,172	462,172	0.00	3,861.6

Table 62: Results for 0-1 knapsack instance with N = 1000000, $\kappa = 0.25$, $\alpha^l = 20$, $\alpha^f = 40$

s	LB	UB	Gap	CPU
1	624,832	624,832	0.00	2,767.5
2	648,759	648,759	0.00	1,834.6
3	608,793	608,793	0.00	3,685.7
4	647,504	647,504	0.00	3,592.4
5	609,511	609,511	0.00	3,749.2
6	608,582	608,582	0.00	6,897.3
7	601,179	601,179	0.00	3,357.9
8	620,676	620,676	0.00	6,196.9
9	598,703	598,703	0.00	11,133.3

A.3 Facility location

In this section we report detailed results of our method for solving the facility location instances considered in our study. In Tables 63–102 we report the results for every choice of N, F and Δ, and for erach of the 9 instances generated for that choice of parameters.

Table 63: Results for UFLP with interdiction for $N = 500, F = 50, \Delta = 1$

s	LB	UB	Gap	CPU
1	132,734	132,734	0.00	77.6
2	136,770	136,770	0.00	124.4
3	133,829	133,829	0.00	51.1
4	134,854	134,854	0.00	59.3
5	132,189	132,189	0.00	154.1
6	132,055	132,055	0.00	61.5
7	131,091	131,091	0.00	56.9
8	135,312	135,312	0.00	87.4
9	132,338	132,338	0.00	82.4

Table 64: Results for UFLP with interdiction for $N = 500, F = 50, \Delta = 2$

s	LB	UB	Gap	CPU
1	133,509	133,509	0.00	128.0
2	137,437	137,437	0.00	226.6
3	134,321	134,321	0.00	231.9
4	135,196	135,196	0.00	130.1
5	133,233	133,233	0.00	138.8
6	133,128	133,128	0.00	210.2
7	132,940	132,940	0.00	120.2
8	136,182	136,182	0.00	122.9
9	132,870	132,870	0.00	210.0

Table 65: Results for UFLP with interdiction for $N = 500, F = 50, \Delta = 3$

s	LB	UB	Gap	CPU
1	134,095	134,095	0.00	129.7
2	138,158	138,158	0.00	347.5
3	134,847	134,847	0.00	712.3
4	135,914	135,914	0.00	404.3
5	134,242	134,242	0.00	319.6
6	134,074	134,074	0.00	237.2
7	133,974	133,974	0.00	144.3
8	136,942	136,942	0.00	261.1
9	133,691	133,691	0.00	372.9

Table 66: Results for UFLP with interdiction for $N = 500, F = 50, \Delta = 4$

s	LB	UB	Gap	CPU
1	134,781	134,781	0.00	270.4
2	138,466	138,466	0.00	677.1
3	135,339	135,339	0.00	821.6
4	136,469	136,469	0.00	555.9
5	134,680	134,680	0.00	506.5
6	135,116	135,116	0.00	620.7
7	134,489	134,489	0.00	438.6
8	137,590	137,590	0.00	224.2
9	134,140	134,140	0.00	249.4

Table 67: Results for UFLP with interdiction for $N = 500, F = 50, \Delta = 5$

s	LB	UB	Gap	CPU
1	135,136	135,136	0.00	320.9
2	138,885	138,885	0.00	638.2
3	135,511	135,511	0.00	940.2
4	137,181	137,181	0.00	381.4
5	134,956	134,956	0.00	633.8
6	135,276	135,276	0.00	568.3
7	135,134	135,134	0.00	291.2
8	138,104	138,104	0.00	443.9
9	135,015	135,015	0.00	295.1

Table 68: Results for UFLP with interdiction for $N = 500, F = 100, \Delta = 1$

s	LB	UB	Gap	CPU
1	130,239	130,239	0.00	298.8
2	130,274	130,274	0.00	336.6
3	131,439	131,439	0.00	329.5
4	131,114	131,114	0.00	186.4
5	131,410	131,410	0.00	238.1
6	131,814	131,814	0.00	415.9
7	131,411	131,411	0.00	195.5
8	131,574	131,574	0.00	277.7
9	129,858	129,858	0.00	166.0

Table 69: Results for UFLP with interdiction for $N = 500, F = 100, \Delta = 2$

s	LB	UB	Gap	CPU
1	130,638	130,638	0.00	1,159.2
2	130,649	130,649	0.00	1,668.8
3	131,755	131,755	0.00	1,184.5
4	132,097	132,097	0.00	349.7
5	132,144	132,144	0.00	1,136.9
6	132,462	132,462	0.00	1,245.5
7	131,523	131,523	0.00	790.0
8	132,444	132,444	0.00	584.2
9	130,638	130,638	0.00	548.8

Table 70: Results for UFLP with interdiction for $N = 500, F = 100, \Delta = 3$

s	LB	UB	Gap	CPU
1	131,335	131,335	0.00	1,779.8
2	131,177	131,177	0.00	2,345.6
3	132,133	132,133	0.00	2,021.6
4	132,655	132,655	0.00	706.5
5	132,424	132,424	0.00	1,624.3
6	133,166	133,166	0.00	2,026.6
7	131,986	131,986	0.00	3,463.9
8	132,589	132,589	0.00	1,380.4
9	131,361	131,361	0.00	668.3

Table 71: Results for UFLP with interdiction for $N = 500, F = 100, \Delta = 4$

s	LB	UB	Gap	CPU
1	131,772	131,772	0.00	2,465.8
2	131,402	131,402	0.00	4,595.1
3	132,795	132,795	0.00	3,054.1
4	133,375	133,375	0.00	1,045.7
5	132,971	132,971	0.00	2,216.6
6	133,833	133,833	0.00	2,421.3
7	132,696	132,696	0.00	2,379.6
8	133,767	133,767	0.00	1,505.8
9	131,893	131,893	0.00	1,305.0

Table 72: Results for UFLP with interdiction for $N = 500, F = 100, \Delta = 5$

s	LB	UB	Gap	CPU
1	132,058	132,058	0.00	2,444.5
2	131,759	131,759	0.00	5,690.8
3	133,345	133,345	0.00	2,528.3
4	133,868	133,868	0.00	1,218.2
5	133,252	133,252	0.00	2,707.1
6	134,174	134,174	0.00	2,617.1
7	132,776	132,776	0.00	2,463.7
8	133,933	133,933	0.00	980.4
9	132,161	132,161	0.00	2,595.3

Table 73: Results for UFLP with interdiction for $N = 1000, F = 50, \Delta = 1$

s	LB	UB	Gap	CPU
1	213,806	213,806	0.00	168.0
2	212,033	212,033	0.00	135.5
3	213,992	213,992	0.00	181.4
4	215,872	215,872	0.00	287.4
5	215,127	215,127	0.00	954.7
6	212,313	212,313	0.00	1,110.3
7	214,200	214,200	0.00	1,255.3
8	216,698	216,698	0.00	544.2
9	211,052	211,052	0.00	543.8

Table 74: Results for UFLP with interdiction for $N = 1000, F = 50, \Delta = 2$

s	LB	UB	Gap	CPU
1	215,144	215,144	0.00	402.9
2	213,455	213,455	0.00	529.5
3	215,067	215,067	0.00	643.4
4	216,750	216,750	0.00	933.2
5	215,541	215,541	0.00	1,754.2
6	213,386	213,386	0.00	2,602.8
7	215,083	215,083	0.00	2,056.5
8	217,047	217,047	0.00	2,828.6
9	212,204	212,204	0.00	916.8

Table 75: Results for UFLP with interdiction for $N = 1000, F = 50, \Delta = 3$

s	LB	UB	Gap	CPU
1	216,217	216,217	0.00	1,239.5
2	214,738	214,738	0.00	604.8
3	215,827	215,827	0.00	1,795.0
4	217,240	217,240	0.00	3,661.1
5	215,740	215,740	0.00	4,221.4
6	214,025	214,025	0.00	3,661.2
7	215,547	215,547	0.00	2,513.3
8	218,270	218,270	0.00	3,015.7
9	213,304	213,304	0.00	1,834.3

Table 76: Results for UFLP with interdiction for $N = 1000, F = 50, \Delta = 4$

s	LB	UB	Gap	CPU
1	217,238	217,238	0.00	1,197.0
2	215,972	215,972	0.00	854.2
3	216,902	216,902	0.00	1,844.1
4	217,900	217,900	0.00	2,998.1
5	216,422	216,422	0.00	6,399.8
6	215,039	215,039	0.00	4,298.4
7	216,476	216,476	0.00	6,318.7
8	218,588	218,588	0.00	3,071.0
9	214,018	214,018	0.00	4,769.2

Table 77: Results for UFLP with interdiction for $N = 1000, F = 50, \Delta = 5$

s	LB	UB	Gap	CPU
1	218,073	218,073	0.00	1,619.9
2	216,657	216,657	0.00	467.3
3	217,513	217,513	0.00	1,836.6
4	218,570	218,570	0.00	4,780.0
5	216,910	216,910	0.00	6,505.1
6	215,678	215,678	0.00	4,146.8
7	217,070	217,070	0.00	7,705.5
8	219,757	219,757	0.00	3,633.1
9	214,792	214,792	0.00	2,939.7

Table 78: Results for UFLP with interdiction for $N = 1000, F = 100, \Delta = 1$

s	LB	UB	Gap	CPU
1	206,686	206,686	0.00	1,662.3
2	209,270	209,270	0.00	919.8
3	212,371	212,371	0.00	1,139.2
4	208,173	208,173	0.00	859.7
5	206,549	206,549	0.00	1,396.2
6	207,488	207,488	0.00	1,819.9
7	212,392	212,392	0.00	1,481.9
8	207,685	207,685	0.00	277.2
9	205,030	205,030	0.00	1,682.7

Table 79: Results for UFLP with interdiction for $N = 1000, F = 100, \Delta = 2$

s	LB	UB	Gap	CPU
1	207,203	207,203	0.00	4,465.1
2	209,746	209,746	0.00	2,155.2
3	213,117	213,117	0.00	2,488.9
4	209,088	209,088	0.00	2,084.5
5	207,847	207,847	0.00	3,632.4
6	208,213	208,213	0.00	3,831.7
7	212,830	212,830	0.00	6,725.0
8	208,931	208,931	0.00	743.4
9	205,759	205,759	0.00	8,291.3

Table 80: Results for UFLP with interdiction for $N = 1000, F = 100, \Delta = 3$

s	LB	UB	Gap	CPU
1	207,790	207,790	0.00	12,012.2
2	210,835	210,835	0.00	6,649.2
3	213,754	213,754	0.00	6,304.6
4	209,964	209,964	0.00	4,146.8
5	208,412	208,412	0.00	6,239.2
6	209,062	209,062	0.00	8,954.5
7	213,632	213,632	0.00	8,579.0
8	209,432	209,432	0.00	1,269.7
9	206,682	206,682	0.00	3,930.9

Table 81: Results for UFLP with interdiction for $N = 1000, F = 100, \Delta = 4$

s	LB	UB	Gap	CPU
1	208,307	208,307	0.00	14,586.5
2	211,331	211,331	0.00	7,650.6
3	214,108	214,108	0.00	11,270.5
4	210,772	210,772	0.00	7,639.8
5	209,020	209,020	0.00	6,989.3
6	209,412	209,412	0.00	5,776.0
7	214,099	214,099	0.00	10,245.4
8	209,808	209,808	0.00	5,850.6
9	207,043	207,043	0.00	7,611.4

Table 82: Results for UFLP with interdiction for $N = 1000, F = 100, \Delta = 5$

s	LB	UB	Gap	CPU
1	208,702	208,702	0.00	29,031.9
2	211,853	211,853	0.00	12,696.4
3	214,732	214,732	0.00	11,022.0
4	211,596	211,596	0.00	4,294.4
5	209,817	209,817	0.00	5,310.0
6	210,196	210,196	0.00	15,458.9
7	214,458	214,458	0.00	12,446.7
8	210,193	210,193	0.00	9,669.6
9	207,993	207,993	0.00	10,843.3

Table 83: Results for SSCFLP with interdiction for $N = 500, F = 50, \Delta = 1$

s	LB	UB	Gap	CPU
1	132,905	132,905	0.00	149.4
2	138,037	138,037	0.00	762.0
3	134,727	134,727	0.00	391.6
4	135,364	135,364	0.00	366.5
5	132,522	132,522	0.00	286.4
6	132,229	132,229	0.00	191.1
7	133,378	133,378	0.00	1,604.9
8	136,892	136,892	0.00	1,053.5
9	133,642	133,642	0.00	353.1

Table 84: Results for SSCFLP with interdiction for $N = 500, F = 50, \Delta = 2$

s	LB	UB	Gap	CPU
1	134,266	134,266	0.00	234.4
2	139,075	139,075	0.00	1,372.1
3	135,111	135,111	0.00	407.7
4	135,488	135,488	0.00	1,104.1
5	133,531	133,531	0.00	747.3
6	133,998	133,998	0.00	260.9
7	134,448	134,448	0.00	5,776.7
8	137,574	137,574	0.00	4,959.4
9	134,713	134,713	0.00	552.8

Table 85: Results for SSCFLP with interdiction for $N = 500, F = 50, \Delta = 3$

s	LB	UB	Gap	CPU
1	135,307	135,307	0.00	353.6
2	139,435	139,435	0.00	1,701.4
3	135,745	135,745	0.00	2,349.8
4	136,818	136,818	0.00	1,099.5
5	134,518	134,518	0.00	882.2
6	135,132	135,132	0.00	1,479.7
7	134,933	134,933	0.00	11,226.2
8	138,157	138,157	0.00	5,039.6
9	135,388	135,388	0.00	1,206.4

Table 86: Results for SSCFLP with interdiction for $N = 500, F = 50, \Delta = 4$

s	LB	UB	Gap	CPU
1	135,758	135,758	0.00	1,184.8
2	139,725	139,725	0.00	7,997.1
3	136,129	136,129	0.00	4,142.6
4	136,961	136,961	0.00	3,720.3
5	135,527	135,527	0.00	1,108.5
6	135,535	135,535	0.00	2,823.8
7	135,763	135,763	0.00	9,338.1
8	138,756	138,756	0.00	6,705.6
9	135,781	135,781	0.00	3,578.1

Table 87: Results for SSCFLP with interdiction for $N = 500, F = 50, \Delta = 5$

s	LB	UB	Gap	CPU
1	136,685	136,685	0.00	2,350.7
2	140,044	140,044	0.00	10,379.5
3	136,721	136,721	0.00	3,935.7
4	137,805	137,805	0.00	4,172.8
5	135,576	135,576	0.00	3,471.2
6	136,504	136,504	0.00	3,414.6
7	136,318	136,318	0.00	13,370.8
8	139,209	139,209	0.00	8,933.1
9	136,059	136,059	0.00	11,629.3

Table 88: Results for SSCFLP with interdiction for $N = 500, F = 100, \Delta = 1$

s	LB	UB	Gap	CPU
1	131,377	131,377	0.00	3,227.2
2	130,990	130,990	0.00	2,126.4
3	132,053	132,053	0.00	6,054.3
4	132,980	132,980	0.00	3,523.7
5	133,239	133,239	0.00	5,169.4
6	132,639	132,639	0.00	2,044.9
7	132,510	132,510	0.00	4,171.5
8	133,147	133,147	0.00	2,318.1
9	131,547	131,547	0.00	12,517.8

Table 89: Results for SSCFLP with interdiction for $N = 500, F = 100, \Delta = 2$

s	LB	UB	Gap	CPU
1	131,768	131,768	0.00	6,005.2
2	131,177	131,177	0.00	6,377.3
3	132,294	132,294	0.00	9,583.8
4	133,669	133,669	0.00	3,831.0
5	134,075	134,075	0.00	19,265.9
6	132,986	132,986	0.00	6,888.1
7	133,120	133,120	0.00	23,290.1
8	133,567	133,567	0.00	5,633.9
9	132,217	132,217	0.00	7,009.4

Table 90: Results for SSCFLP with interdiction for $N = 500, F = 100, \Delta = 3$

s	LB	UB	Gap	CPU
1	132,263	132,263	0.00	13,664.9
2	131,482	131,482	0.00	20,131.4
3	132,632	132,632	0.00	23,823.0
4	134,378	134,378	0.00	17,023.0
5	ML	ML	ML	ML
6	133,894	133,894	0.00	9,812.2
7	133,414	133,414	0.00	51,189.4
8	ML	ML	ML	ML
9	132,609	132,609	0.00	25,946.0

Table 91: Results for SSCFLP with interdiction for $N = 500, F = 100, \Delta = 4$

s	LB	UB	Gap	CPU
1	133,161	133,161	0.00	17,430.3
2	131,982	131,982	0.00	29,799.0
3	133,473	133,473	0.00	15,227.1
4	ML	ML	ML	ML
5	134,532	134,532	0.00	35,982.2
6	134,068	134,068	0.00	17,829.1
7	133,832	133,832	0.00	55,934.4
8	134,663	134,663	0.00	12,961.6
9	ML	ML	ML	ML

Table 92: Results for SSCFLP with interdiction for $N = 500, F = 100, \Delta = 5$

s	LB	UB	Gap	CPU
1	133,204	133,204	0.00	28,659.1
2	132,260	132,260	0.00	50,118.0
3	133,659	133,659	0.00	36,374.7
4	135,085	135,085	0.00	35,543.6
5	ML	ML	ML	ML
6	134,714	134,714	0.00	18,536.5
7	134,154	134,154	0.00	76,777.5
8	134,766	134,766	0.00	20,402.4
9	133,381	133,381	0.00	79,470.4

Table 93: Results for SSCFLP with interdiction for $N = 1000, F = 50, \Delta = 1$

s	LB	UB	Gap	CPU
1	213,806	213,806	0.00	238.2
2	212,033	212,033	0.00	182.9
3	213,992	213,992	0.00	314.3
4	215,872	215,872	0.00	729.8
5	215,128	215,128	0.00	2,899.0
6	212,313	212,313	0.00	1,745.0
7	214,200	214,200	0.00	2,476.2
8	216,698	216,698	0.00	1,062.9
9	211,052	211,052	0.00	946.6

Table 94: Results for SSCFLP with interdiction for $N = 1000, F = 50, \Delta = 2$

s	LB	UB	Gap	CPU
1	215,144	215,144	0.00	626.2
2	213,455	213,455	0.00	745.8
3	215,067	215,067	0.00	1,003.8
4	216,750	216,750	0.00	2,512.8
5	215,541	215,541	0.00	4,760.2
6	213,386	213,386	0.00	4,545.5
7	215,083	215,083	0.00	3,207.5
8	217,047	217,047	0.00	4,576.5
9	212,204	212,204	0.00	1,105.2

Table 95: Results for SSCFLP with interdiction for $N = 1000, F = 50, \Delta = 3$

s	LB	UB	Gap	CPU
1	216,217	216,217	0.00	1,349.0
2	214,738	214,738	0.00	830.7
3	215,827	215,827	0.00	2,594.4
4	217,240	217,240	0.00	7,284.3
5	215,740	215,740	0.00	11,888.8
6	214,025	214,025	0.00	8,997.8
7	215,547	215,547	0.00	8,413.6
8	218,270	218,270	0.00	2,770.6
9	213,304	213,304	0.00	1,883.9

Table 96: Results for SSCFLP with interdiction for $N = 1000, F = 50, \Delta = 4$

s	LB	UB	Gap	CPU
1	217,238	217,238	0.00	2,038.9
2	215,972	215,972	0.00	1,340.8
3	216,902	216,902	0.00	2,190.6
4	217,900	217,900	0.00	14,182.8
5	216,423	216,423	0.00	13,142.7
6	215,039	215,039	0.00	7,190.9
7	216,476	216,476	0.00	15,810.2
8	218,588	218,588	0.00	7,335.0
9	214,018	214,018	0.00	4,560.7

Table 97: Results for SSCFLP with interdiction for $N = 1000, F = 50, \Delta = 5$

s	LB	UB	Gap	CPU
1	218,073	218,073	0.00	1,492.1
2	216,668	216,668	0.00	1,789.1
3	217,513	217,513	0.00	3,615.2
4	218,570	218,570	0.00	14,747.2
5	216,910	216,910	0.00	16,394.7
6	215,678	215,678	0.00	7,654.2
7	217,070	217,070	0.00	23,622.3
8	219,757	219,757	0.00	7,125.3
9	214,792	214,792	0.00	10,195.3

Table 98: Results for SSCFLP with interdiction for $N = 1000, F = 100, \Delta = 1$

s	LB	UB	Gap	CPU
1	206,686	206,686	0.00	7,190.9
2	209,270	209,270	0.00	4,212.2
3	212,371	212,371	0.00	2,769.5
4	208,173	208,173	0.00	2,671.7
5	206,549	206,549	0.00	5,225.4
6	207,488	207,488	0.00	8,329.3
7	212,392	212,392	0.00	6,788.2
8	207,685	207,685	0.00	2,110.0
9	205,030	205,030	0.00	5,045.2

Table 99: Results for SSCFLP with interdiction for $N = 1000, F = 100, \Delta = 2$

s	LB	UB	Gap	CPU
1	207,203	207,203	0.00	21,715.4
2	209,746	209,746	0.00	9,857.1
3	213,117	213,117	0.00	7,067.9
4	209,088	209,088	0.00	5,538.9
5	207,847	207,847	0.00	17,967.5
6	208,213	208,213	0.00	20,613.6
7	212,830	212,830	0.00	32,304.7
8	208,931	208,931	0.00	6,210.4
9	205,759	205,759	0.00	18,690.5

Table 100: Results for SSCFLP with interdiction for $N = 1000, F = 100, \Delta = 3$

s	LB	UB	Gap	CPU
1	207,790	207,790	0.00	39,913.2
2	210,835	210,835	0.00	25,668.4
3	213,754	213,754	0.00	21,233.7
4	209,964	209,964	0.00	12,753.8
5	208,412	208,412	0.00	19,516.6
6	209,062	209,062	0.00	25,472.0
7	213,632	213,632	0.00	51,788.5
8	209,432	209,432	0.00	7,022.2
9	206,682	206,682	0.00	11,647.4

Table 101: Results for SSCFLP with interdiction for $N = 1000, F = 100, \Delta = 4$

s	LB	UB	Gap	CPU
1	208,307	208,307	0.00	51,088.8
2	211,331	211,331	0.00	31,255.3
3	214,108	214,108	0.00	32,394.4
4	210,772	210,772	0.00	17,439.5
5	209,020	209,020	0.00	23,443.3
6	209,412	209,412	0.00	39,497.7
7	214,099	214,099	0.00	60,765.2
8	209,808	209,808	0.00	24,489.9
9	207,043	207,043	0.00	29,450.2

Table 102: Results for SSCFLP with interdiction for $N = 1000, F = 100, \Delta = 5$

s	LB	UB	Gap	CPU
1	208,702	208,702	0.00	75,711.9
2	211,853	211,853	0.00	53,405.7
3	214,732	214,732	0.00	47,055.9
4	211,596	211,596	0.00	18,722.2
5	209,817	209,817	0.00	20,408.5
6	210,196	210,196	0.00	80,734.3
7	214,458	214,459	0.00	TL
8	210,193	210,193	0.00	53,943.7
9	207,993	207,993	0.00	27,690.4

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.