J Glob Optim (2007) 38:181–199
DOI 10.1007/s10898-006-9067-3

ORIGINAL PAPER

Survivable network design under optimal and heuristic interdiction scenarios

J. Cole Smith · Churlzu Lim · Fransisca Sudargho

Received: 3 January 2006 / Accepted: 1 July 2006 / Published online: 9 August 2006

Abstract We examine the problem of building or fortifying a network to defend against enemy attacks in various scenarios. In particular, we examine the case in which an enemy can destroy any portion of any arc that a designer constructs on the network, subject to some interdiction budget. This problem takes the form of a three-level, two-player game, in which the designer acts first to construct a network and transmit an initial set of flows through the network. The enemy acts next to destroy a set of constructed arcs in the designer's network, and the designer acts last to transmit a final set of flows in the network. Most studies of this nature assume that the enemy will act optimally; however, in real-world scenarios one cannot necessarily assume rationality on the part of the enemy. Hence, we prescribe optimal network design algorithms for three different profiles of enemy action: an enemy destroying arcs based on capacities, based on initial flows, or acting optimally to minimize our maximum profits obtained from transmitting flows.

Keywords Network design · Integer programming · Network interdiction · Game theory

1 Introduction

The continuous increase of telecommunications and transportation needs has made the design of cost-efficient networks that meet requirements concerning flexibility and survivability a major challenge. Many traditional network design algorithms do not take into account the survivability aspect of a network, which can perhaps leave the network susceptible to failures of small subsets of its arcs. It is important to

J. C. Smith (✉) · C. Lim
Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL, USA
e-mail: cole@ise.ufl.edu

F. Sudargho
Department of Systems and Industrial Engineering, The University of Arizona, Tucson, AZ, USA

design networks that are robust with respect to accidental failures like transportation breakdowns, road closures, and telephone line breaks, or to failures made maliciously by enemy entities. Vital applications that inspire this research include networks in voice and data communication, military services, and mass transit.

In general, the Survivable Network Design (SND) problem seeks a minimum-cost robust network configuration that provides a number of alternative paths between nodes of the network. Many SND studies have focused on telecommunication applications (see [1, 8, 9, 20, 21, 23–25], e.g.). Given point-to-point traffic demands, the general SND problem assigns capacities to arcs (perhaps from among a finite set of alternatives) in order to minimize construction/expansion costs, while satisfying certain survivability constraints under any single node or arc failure. A critical survivability requirement is the minimum degree of flow feasibility after a network component failure. That is, the network is survivable if specified percentages of demands can be satisfied when any single node or arc fails. Some alternative survivability conditions impose upper bounds on path lengths [1] and/or require the existence of multiple paths between each point-to-point demand [9].

We remark that the aforementioned studies only consider the random failure of network components. Furthermore, most of these research efforts analyze single component failures (some exceptions can be found in [21], who consider the simultaneous failure of pairs of arcs, and [14], who consider the case of multiple arc failures in a Benders decomposition scheme). However, when the network is maliciously attacked, this random failure assumption might not be reasonable since the enemy will attempt to make the maximum impact on the network (see [22], e.g.). Also, for the common scenario in which the enemy can simultaneously attack multiple arcs, enumerating failure scenarios consisting of "any k arcs" for some integer k becomes computationally intractable. With this motivation, we are particularly interested in designing a survivable network under an enemy's choice of multiple attacks. To this end, we need to understand and model how a rational enemy will act for a given network topology, which is referred to *network interdiction* in the literature.

The network interdiction problem has received much attention in the literature due to its applications in military and homeland security operations. Wollmer [28] proposes an algorithm that destroys (i.e., *interdicts*) a prescribed number of arcs in a network in order to minimize the follower's maximal flow. Wood [29] provides an integer programming formulation for a discrete interdiction problem, and extends the model to allow for continuous interdiction, multiple sources and sinks, undirected networks, multiple interdiction resources, and multiple commodities. This work was continued by Cormican et al. [11], in which the leader minimizes the expected maximum flow, given uncertainties regarding the success of interdiction and arc capacities. A different interdiction problem is examined by Fulkerson and Harding [13], who examine the problem of maximizing the shortest source-sink path in the presence of arc-extension costs, which serve as interdiction costs. A recent study by Israeli and Wood [15] develops two decomposition algorithms using super-valid inequalities and set covering master problems. Finally, Lim and Smith [17] examine the multicommodity flow network interdiction problem under assumptions of discrete and continuous enemy action.

The problem studied here can be modeled as a three-stage problem. In the first stage, the network designer/operator (referred to as the "designer" in this paper) constructs a network in which each arc has a fixed construction cost, a maximum capacity, and a per-unit flow profit (as assumed in [11, 17, 27]). The designer is also restricted

by a budget on the arc construction costs. Before the enemy acts, the designer places an initial set of multicommodity flows on the network, which yields some measure of initial profits (measured by the revenue generated by successfully shipping units of commodity from origins to destinations minus the flow costs).

In the second stage, the enemy inflicts damage to the network by reducing the capacity of certain arcs. We consider three enemy algorithms in this paper: greedily destroy the largest-capacity arcs, greedily destroy the arcs having the largest initial flows, or optimally minimize the maximum possible post-interdiction flow profit. The first two cases might represent the case in which the enemy intends to maximally disrupt the network, but is acting according to a heuristic strategy due to real-time considerations or due to limited information. For all of these cases, it is important to note that the enemy can destroy portions of arcs by removing some of their capacity, rather than being restricted to integer interdiction actions. This assumption is made in contrast to other studies in which interdiction is binary; i.e., each arc must be either completely interdicted or left entirely alone.

Finally, in the third stage, the designer maximizes post-interdiction profit by solving a multicommodity flow problem on the remaining network. Hence, our problem is different from those in previous SND studies in that our objective function includes a weighted combination of flow profits before and after enemy interdiction minus arc construction costs.

Note that when arcs are heuristically disrupted, the enemy has a different objective function from the designer's. A mathematical programming version of this type of Stackelberg game is a (linear) *bilevel programming* (also known as *two-level programming* or *hierarchical optimization*) problem [2, 6, 12, 16] in which the enemy's optimization problem appears in the set of the designer's constraints. Similar to bilinear programming problems, it is known that an optimal solution to the linear bilevel programming problem can be found among pairs of the extreme points to the designer's and enemy's polytopes [3]. Two notable solution methods for solving this problem are implicit enumeration of extreme points [5, 7] and reformulation using KKT (Karush–Kuhn–Tucker) optimality conditions [4, 12]. For a more comprehensive review of general bilinear programming (we refer the reader to [3, 10, 27]).

The remainder of this paper is organized as follows. In Sect. 2, we provide a formal description and notation of the problem with examples for each interdiction scenario. Then, in Sect. 3, we present solution methods that find an optimal network design under three scenarios. We report computational results that illustrate the efficacy of the proposed methods in Sect. 4. Finally, we conclude the paper in Sect. 5.

2 Problem statement and notation

In this section, we describe the mathematical notation for our problem, along with a precise description of the three interdiction scenarios considered in this paper. Consider a directed graph $G(N, A)$ with node set N and arc set A. Associated with each arc $i \in A$ are a nonnegative construction cost c_i, a nonnegative interdiction cost b_i, and an arc capacity q_i. The designer is limited by a budget of C for constructing arcs in the network, while the enemy is constrained by an interdiction budget of B.

Define $RS(j)$ and $FS(j)$, $\forall j \in N$, to be the reverse star and forward star of node j, respectively, i.e., $RS(j) = \{i \in A : \text{arc } i \text{ enters node } j\}$ and $FS(j) = \{i \in A : \text{arc } i$

leaves node j}. Furthermore, define K to be the set of commodities, and let d_k^j denote the demand of commodity $k \in K$ at node $j \in N$. If $d_k^j > 0$, then j is a supply node of commodity k, while $d_k^j < 0$ implies that j is a destination node for commodity k. Without loss of generality, we assume that $\sum_{j \in N} d_k^j = 0$, $\forall k \in K$. We are given a per-unit flow profit f_k^i for transmitting a unit of commodity k over arc i for each $i \in A$ and $k \in K$. This value includes the (nonpositive) per-unit flow cost on arc i, plus (positive) revenue for successfully shipping a unit of commodity k if it enters a destination node of k. This reward is subtracted from the flow profit if arc i exits a destination node for k.

It is possible that there may not exist a feasible multicommodity flow in the network (especially after interdiction), or that it might not be cost-effective to route all of the requested demands for some commodity. Thus, we create dummy arcs in the network to allow some origin nodes to send less than their supplies, and destination nodes to receive less than their demands. These dummy arcs have zero construction and flow profits (unless some disposal or shortage costs are appropriate), and large enough interdiction costs and capacities to ensure that they cannot be destroyed by the enemy.

The objective function maximizes some convex combination of the profit obtained from transmitting flows across the network before and after interdiction, where profit is measured by revenues gained from successful shipment of goods minus arc construction costs. Suppose that $100\rho\%$ of flows occur before enemy interdiction for some $\rho \in [0, 1]$. Then our total profit is ρ times our pre-interdiction flow profits plus $(1 - \rho)$ times our post-interdiction flow profits, minus the arc construction costs.

In this paper, we consider three interdiction scenarios. In case 1, the enemy destroys arcs having the largest capacity until his budget is exhausted or until no arcs remain. In case 2, the enemy destroys arcs on which the largest pre-interdiction flows exist. Finally, in case 3, the enemy acts to (optimally) minimize the designer's maximum post-interdiction flow profit.

To illustrate these interdiction schemes, suppose that the network designer can build arcs on the network depicted in Fig. 1(a). There exist two commodities in the problem whose origin-destination pairs are (1, 3) and (2, 4). Each origin (destination) node can supply (receive) at most ten units. (Recall that we permit shortages in supplies and demands by drawing dummy arcs from node 1 to node 3, and from node 2 to node 4. These arcs are omitted in Fig. 1(a) for ease of readability.) The arcs are labeled by their capacity, unit flow profit of commodity 1, and unit flow profit of commodity 2, i.e., (q_i, f_i^1, f_i^2) $\forall i = 1, \ldots, 5$. For this instance, suppose that the designer achieves a revenue of 5 for each unit of commodity 1 delivered, and a revenue of 10 for each unit of commodity 2 delivered, while the unit flow cost is 1 for any commodity on any arc. Assume that each arc can be constructed by the designer and destroyed by the enemy with a common cost of 10, i.e., $c_i = b_i = 10$ $\forall i = 1, \ldots, 5$. Let the designer's budget be $C = 50$ and the enemy's budget be $B = 20$. Finally, let the profit weight be $\rho = 0.5$. If no interdiction action is taken, the optimal design and flows would be given by those in Fig. 1(b), yielding a profit of 110 $(= 130 - 20)$ in this scenario.

Figure 2 depicts the optimal solution in case 1, where Fig. 2(a) depicts the optimal network design and pre-interdiction flows, and Fig. 2(b) depicts the optimal post-interdiction flows. From this network design, the enemy would destroy arcs (1,2) and (2,4), since they have the largest capacities. Note that the designer needs to alter the initial flows of commodity 2 using arcs (2,3) and (3,4) after interdiction. The overall profit is now reduced to 75 $(= (0.5)(130) + (0.5)(120) - 50)$.

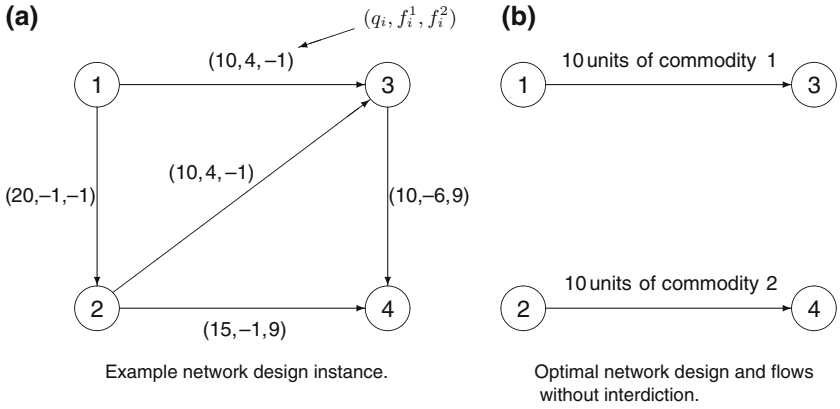

Fig. 1 Network topology and optimal solution without interdiction

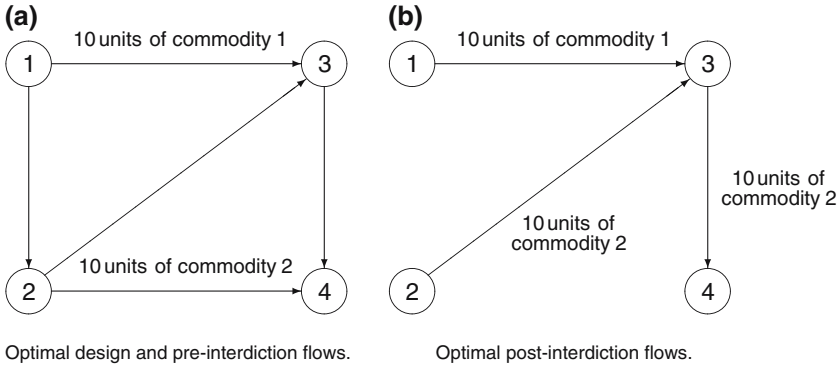

Fig. 2 Network design and flows under case 1 interdiction

In case 2, the enemy will destroy arcs having the largest initial flows. Figure 3 shows the optimal network design and flow patterns. Note that the initial flow on arc (1, 3) is slightly smaller than those on arcs (2, 3) and (3, 4) by some $\varepsilon > 0$ units so that the enemy can be induced to destroy the latter two arcs as desired by the network designer. The network designer's profit is $85 - 2\varepsilon$ $(= (0.5)(120 - 4\varepsilon) + (0.5)(130) - 40)$.

Finally, Fig. 4 illustrates the optimal network design for the third case, where the enemy optimally destroys arcs to minimize the designer's flow profit after interdiction. While the network design is same as the one in case 2, the enemy optimally disrupts arcs (2, 3) and (2, 4) by observing that commodity 2 yields a greater profit to the network designer than commodity 1 does. Then, the designer can send post-interdiction flows only on arc (1, 3) as in Fig. 4(b). The profit is 45 $(= (0.5)(130) + (0.5)(40) - 40)$. (The enemy also has an alternative optimal solution in which arcs (2, 4) and (3, 4) are destroyed.) Despite the fact that arc (3,4) does not carry any pre- or post-interdiction flow, the designer must build this arc to prevent the enemy from simply destroying arcs (1,3) and (2,4), which, without the presence of (3,4), would leave the designer with no post-interdiction flows. (The designer also has an alternative optimal solution in which only arcs (1,3) and (2,4) are constructed, and in which no post-interdiction flows exist.)

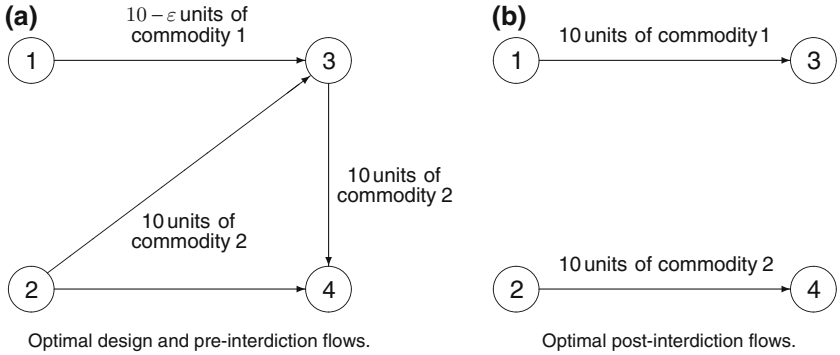

Fig. 3 Network design and flows under case 2 interdiction

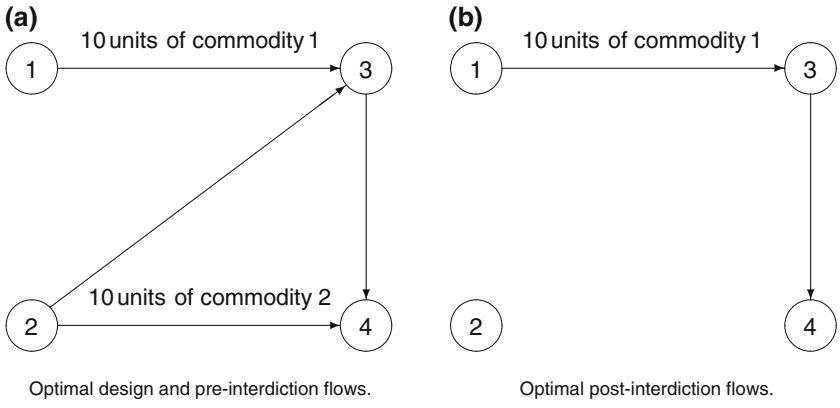

Fig. 4 Network design and flows under case 3 interdiction

Before proceeding to the next section, we remark that all three problems described in this section are difficult to solve. In particular, even a special case in which the follower's budget is set as zero reduces to a capacitated multicommodity network design problem, which is NP-hard in the strong sense [18, 19].

3 Network design algorithms

In this section, we provide mathematical programming algorithms for solving the network design problem under each interdiction scenario. We define the following set of decision variables for these models. Let x_i, $\forall i \in A$, be a binary decision variable that equals to 1 if arc i is constructed and 0 otherwise. For the flow decision variables, we let u_i^k and v_i^k, $\forall i \in A$, $\forall k \in K$, represent the flow of commodity k on arc i before and after enemy interdiction, respectively. Also, let $w_i \in [0,1]$, $\forall i \in A$, represent the remaining percentage of arc is capacity after enemy interdiction. Although w_i is determined by the enemy, we view it as a decision variable induced by the designer's choice of x-variables.

In the next three sections, we propose solution methods for each case. As mentioned earlier, the enemy has a different objective function than the designer's for the first two cases. Hence, these problems can be solved via conventional bilevel programming techniques. However, exploiting the knapsack constraint in the enemy's interdiction problem, we provide equivalent mixed integer formulations. These formulations can then in turn be solved by standard commercial software. For the third case, we design a cutting-plane algorithm that finds an optimal solution in a finite number of steps. (A sketch of these ideas appears in [26].)

3.1 Case 1: capacity-based interdiction

In this section, suppose that the enemy repeatedly destroys arcs having the largest capacity until the budget B is exhausted. For our initial discussion, we assume that all arc capacities are unique (we discuss the implication of this assumption further at the end of this subsection). Hence, we can order the arc indices $i = 1, \ldots, |A|$ so that $q_i < q_{i+1}$ $\forall i = 1, \ldots, |A| - 1$, and so the enemy will prefer to destroy arc i before destroying $i + 1$. Note that in general, this overall ordering can be based on any input data criteria. We examine two different approaches for solving this problem.

3.1.1 Cutting plane algorithm

A partial formulation for this problem is given as follows.

$$\text{Maximize} \quad \rho \sum_{k\in K}\sum_{i\in A} f_i^k u_i^k + (1-\rho)\sum_{k\in K}\sum_{i\in A} f_i^k v_i^k - \sum_{i\in A} c_i x_i, \tag{1a}$$

$$\text{subject to} \quad \sum_{i\in A} c_i x_i \le C, \tag{1b}$$

$$\sum_{i\in FS(j)} u_i^k - \sum_{i\in RS(j)} u_i^k = d_j^k \quad \forall k \in K \quad \forall j \in N, \tag{1c}$$

$$\sum_{i\in FS(j)} v_i^k - \sum_{i\in RS(j)} v_i^k = d_j^k \quad \forall k \in K \quad \forall j \in N, \tag{1d}$$

$$\sum_{k\in K} u_i^k \le q_i x_i \quad \forall i \in A, \tag{1e}$$

$$\sum_{k\in K} v_i^k \le q_i w_i \quad \forall i \in A, \tag{1f}$$

$$w_i \le x_i \quad \forall i \in A, \tag{1g}$$

$$w_i \ge 0 \quad \forall i \in A, \tag{1h}$$

$$u_i^k,\ v_i^k \ge 0 \quad \forall i \in A \quad \forall k \in K, \tag{1i}$$

$$x_i \in \{0,1\} \quad \forall i \in A. \tag{1j}$$

The objective (1a) minimizes the pre-interdiction flow costs weighted by ρ, plus the post-interdiction flow costs weighted by $(1-\rho)$, minus the arc construction costs. The arc construction budget constraint is given by (1b), while the flow conservation constraints before and after enemy interdiction are given by (1c) and (1d), respectively. Constraints (1e) and (1f) represent arc capacity restrictions on the flows. Constraints (1g) state that $w_i = 0$ if arc i was not constructed, and (1h) and (1j) state logical

restrictions on the variables. However, these conditions are only necessary for the w-solution to this problem to reflect the true decision of the enemy, and are obviously not sufficient.

There are an exponential number of valid inequalities that can be used to enforce the w-solution to (1) to match the enemy's actual decisions. We examine a set of valid inequalities that are useful in at least partially enforcing this relationship. First, note that the largest-capacity arcs will be attacked by the enemy if constructed, and hence some arc capacity will never be available after interdiction. First determine the largest index ℓ such that all constructed arcs $1,\ldots,\ell-1$ will be completely destroyed by the enemy. Define

$$B'_i = \sum_{g=1}^{i-1} b_g \quad \forall i \in A, \tag{2}$$

where B'_1 is taken to be 0. Hence, index ℓ is the arc in A for which $B'_\ell \leq B$ and $B'_\ell + b_\ell > B$, and we have that $w_i = 0\ \forall i < \ell$. Similarly, note that the maximum proportion of arc ℓ that can remain after interdiction is

$$\theta = 1 - \frac{B - B'_\ell}{b_\ell} \tag{3}$$

and hence we can state that $w_\ell \leq \theta x_\ell$; however, we will tighten this inequality in the following discussion.

Next, for each $i \in A,\ i \geq \ell$, we derive valid inequalities corresponding to each arc $g \in A,\ g < i$ of the form

$$w_i \leq \gamma_{ig} x_i + (\delta_i - \gamma_{ig}) x_g \quad \forall i, g \in A,\ g < i,\ \ell \leq i, \tag{4}$$

where:

$$\gamma_{ig} = \max\left\{0, 1 - \max\left\{0, \frac{B - B'_i + b_g}{b_i}\right\}\right\} \quad \forall i, g \in A,\ g < i,\ \ell \leq i \tag{5}$$

and where $\delta_\ell = \theta$ and $\delta_i = 1$ for $i \in A, i > \ell$. For constraints (4), γ_{ig} represents the maximum remaining proportion of arc i if $x_i = 1$ and $x_g = 0$ (i.e., the enemy has not spent any of its resources on arc g). This maximum value is possible when all other arcs $1,\ldots,i-1$ except for g have been built. The maximum remaining proportion of arc i if both x_i and x_g are constructed is given by δ_i, so we add $\delta_i - \gamma_{ig}$ back to the right-hand side if $x_g = 1$. Note that if $\gamma_{ig} = 1$, this inequality simply duplicates the $w_i \leq x_i$ constraint stated in (1g).

Finally, we can also develop valid inequalities under the assumption that arcs $1,\ldots,i-1$ have not yet been constructed, and then state the maximum possible increase in the capacity for arc i as lower-indexed arcs are constructed. Observe that if arc i is constructed, its capacity must be at least $\lambda_i = \max\{0, (b_i - B)/b_i\}$, since if $b_i > B$, there must exist at least $(b_i - B)/b_i$ percent of arc is capacity remaining. Also, if arc $g \in A,\ 1 \leq g < i$ is constructed, the enemy can spend up to b_g units of its budget on arc g, which implies that up to $\min\{b_g/b_i,\ (1-\lambda_i)\}$ percent of arc i might be recovered. These inequalities are given as follows.

$$w_i \leq \lambda_i x_i + \sum_{g=1}^{i-1} \min\{b_g/b_i,\ (1-\lambda_i)\} x_g \quad \forall i \in A, i \geq \ell. \tag{6}$$

We can then solve (1) in conjunction with these valid inequalities, and obtain optimal variable values $\hat{x}$ and $\hat{w}$. Let $\overline{w}_i$ be the true action of the enemy for $i \in A$ given $\hat{x}$ (i.e., the $\overline{w}$-variables describe the actual remaining arc capacities after interdiction, given $\hat{x}$). If $\overline{w}_i = \hat{w}_i$ for each $i \in A$, then we have obtained the optimal solution. Else, we add cutting planes to force the w-variables to match the enemy's true decision. For each $i \in A$ such that $\overline{w}_i < \hat{w}_i$, we add the following constraint:

$$w_i \leq \overline{w}_i x_i + \sum_{g=1}^{i-1} \min\{b_g/b_i,\ 1 - \overline{w}_i\}(1 - \hat{x}_g)x_g. \tag{7}$$

Equation (7) imposes an upper bound of $\overline{w}_i$ on w_i, unless some other higher-preference arcs that are not currently being built will be built in the next iteration, thus forcing the enemy to expend resources on other arcs. This valid inequality cuts off the current solution since the right-hand side is currently equal to $\overline{w}_i$. Each such constraint is then passed back to (1), and the model is resolved until $\overline{w} = \hat{w}$.

3.1.2 Static optimization model

A model that can be solved without need for cutting planes makes use of the fact that there will exist only one w-variable value that can be fractional, since the enemy is essentially solving a linear knapsack problem. Define binary decision variables z_i, $\forall i \in A$, equal to one if and only if $x_i = 1$, all constructed arcs with an index smaller than i are completely destroyed, and all constructed arcs with an index greater than i are not affected by the enemy. Arc i itself may be completely or partially interdicted, or unaffected by the enemy. We assume that the enemy exhausts the entire interdiction budget. (This assumption forces us to build at least enough capacity so that the enemy can destroy B units; we handle this assumption by adding a dummy arc between two dummy nodes disconnected from the rest of the network. This arc will have a zero arc construction cost, zero flow profit, any arbitrary capacity, and an interdiction cost of B.) These restrictions, along with (1g), are enforced as follows:

$$\sum_{i \in A} z_i = 1, \tag{8a}$$

$$w_i \leq \sum_{g=1}^{i} z_g \quad \forall i \in A, \tag{8b}$$

$$w_i \geq x_i - \sum_{g=i}^{|A|} z_g \quad \forall i \in A, \tag{8c}$$

$$\sum_{i \in A} b_i(x_i - w_i) = B, \tag{8d}$$

$$z_i \in \{0, 1\} \quad \forall i \in A. \tag{8e}$$

The static optimization model thus incorporates (8) into (1). Constraint (8a) requires that exactly one variable serves as the dividing point, such that all arcs having a higher capacity than q_i are destroyed (enforced by (8b)), and all arcs having a smaller capacity than q_i are not interdicted at all (enforced by (8c)). Constraint (8d) ensures that the enemy will use the entire interdiction budget, while (8e) states logical constraints

on the z-variables. We compare the effectiveness of the dynamic and static models in Sect. 4.

Recall that we have assumed that all capacities are different. Before proceeding to the next case, we discuss how to handle the problem in the presence of arcs having equal capacities. A possible problem that arises when a tie exists is shown in Fig. 5, where there is a single commodity with ten units of supply at node 1 and ten units of demand at node 3. The three numbers in parentheses represent q_i, $c_i(= b_i)$, and f_i^1, respectively. For this example, we set $\rho = 0.5$, $C = 3$, and $B = 2$.

The optimal solution to this problem is to construct all three arcs, and expect that the enemy will cooperatively destroy (1,2) and (2,3). However, in reality, the enemy would not necessarily break ties in favor of the network designer. One way to resolve this difficulty is to slightly perturb arc capacities as desired, if possible. That is, if $q_i = q_j$ and the network designer wishes the enemy to destroy arc i before arc j, then it might be possible to put $q_i \leftarrow q_i - \varepsilon$ (or even $q_j \leftarrow q_j + \varepsilon$) for a small value $\varepsilon > 0$. In the above example, increasing capacities of (1,2) and (2,3) by ε will induce the enemy to destroy (1,2) and (2,3).

Of course, this capacity adjustment strategy might not be possible. In that case, the modeling of the enemy's behavior needs to be better specified. If there exist secondary "tie-breaking" criteria on which arcs the enemy will prefer to destroy, then we can simply use our prior algorithm without modification. Else, if the enemy is assumed to break ties randomly, then we could solve a stochastic program in which the random outcomes correspond to random tie-breaking priorities. The first-stage decisions would correspond to the x-variables, the random outcomes would be a set of tie-breaking orderings among arcs having the same capacity, and the recourse decisions would simultaneously yield the enemy's decision and the designer's post-interdiction flows. Finally, the enemy might be assumed to break ties by acting optimally among ties. While addressing these scenarios formally is beyond the scope of this paper, the cutting-plane strategy in Sect. 3.3 establishes one possible framework for solving such problem extensions.

3.2 Case 2: flow-based interdiction

In this case, the enemy interdicts the arcs having the largest pre-interdiction flows. Since the enemy decision does not depend on a simple set of binary decision variables, the static optimization model in the previous section must be modified for this case.

Once again, we determine an index i^* such that arc i^* may be partially destroyed, implying that every arc with a greater flow than the flow on arc i^* must be completely destroyed, while all arcs with a smaller flow than the flow on arc i^* cannot be interdicted. Unlike the previous case, we do not define a decision variable to determine

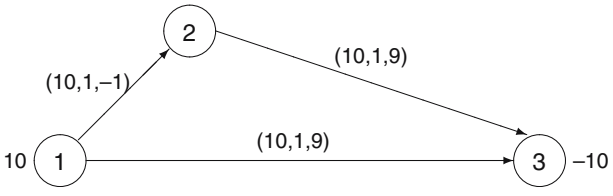

Fig. 5 Illustration of ties in the enemy interdiction problem

the identity of i^*, but instead we must solve one integer program for each possible value that i^* can take. Note that arcs with a larger flow than the flow on arc i^* will be completely destroyed while those with smaller flows will survive. Define $\varepsilon > 0$ as an arbitrarily small constant (whose precise value we will discuss subsequently). Then, we add the following constraints to model (1):

$$\sum_{k\in K} u_j^k - \sum_{k\in K} u_{i^*}^k \le M_{ji^*}(x_j - w_j) - \varepsilon x_j \quad \forall j \in A,\ j \ne i^*, \tag{9a}$$

$$\sum_{k\in K} u_j^k - \sum_{k\in K} u_{i^*}^k \ge -M_{i^*j}(1 - (x_j - w_j)) + \varepsilon x_j \quad \forall j \in A,\ j \ne i^*, \tag{9b}$$

$$\sum_{i\in A} b_i(x_i - w_i) = B, \tag{9c}$$

$$0 \le w_{i^*} \le 1 \quad \forall i^* \in A, \tag{9d}$$

$$w_i \in \{0,1\} \quad \forall i \in A - \{i^*\}, \tag{9e}$$

$$x_{i^*} = 1, \tag{9f}$$

where $M_{ij} = q_i + \varepsilon$. The addition of the constraints in (9) to model (1) captures the flow-based greedy enemy interdiction model, assuming that the enemy completely destroys all arcs having more flow than the flow on i^*, and does not interdict arcs having less flow than arc i^*. By the disjunction of (9a) and (9b), the flow of arc j must satisfy either $\sum_{k\in K} u_j^k \le \sum_{k\in K} u_{i^*}^k - \varepsilon x_j$ or $\sum_{k\in K} u_j^k \ge \sum_{k\in K} u_{i^*}^k + \varepsilon x_j$. This guarantees the uniqueness of the enemy's solution (the implications of which are discussed in more detail below). Note that if $x_j = 0$, then $w_j = 0$ as well. On the right-hand side of (9a) would then be zero, and since $\sum_{k\in K} u_j^k$ must be zero, this inequality is valid. On the right-hand side of (9b) is $-M_{i^*j}$, so this inequality remains valid as well even if $\sum_{k\in K} u_{i^*}^k = q_{i^*}$. Now assume that $x_j = 1$. If there exists more initial flow on arc j than on i^* (i.e., $\sum_{k\in K} u_j^k \ge \sum_{k\in K} u_{i^*}^k + \varepsilon x_j$), then since $x_j = 1$, we have by (9a) that arc j must be interdicted (by setting $w_j = 0$). Similarly, (9b) forces $w_j = x_j$ if there is less flow on j than i^* (i.e., $\sum_{k\in K} u_j^k \le \sum_{k\in K} u_{i^*}^k - \varepsilon x_j$).

The value of ε is an important factor in this algorithm. Once again, consider the instance depicted in Fig. 5 without ε. The optimal solution to this problem is to construct all three arcs, send slightly more than five units on arcs (1,2) and (2,3), and slightly less than five units on (1,3), inducing the enemy to destroy (1,2) and (2,3). Our post-interdiction flows would then send all ten units on (1,3). There exists no optimal solution to this problem; however, the optimal objective function approaches the situation in which exactly five units are sent on all arcs before interdiction, and the enemy decides to interdict (1,2) and (2,3). Hence, the value of ε is used to approximate this convergence and to assure the existence of a solution.

3.3 Case 3: optimal interdiction

Finally, we consider the third case in which the enemy optimally disrupts arcs so as to minimize the designer's profit. (The enemy has complete information of the network design, including arc capacities, flow profits, and demands.) Given a network topology x, therefore, the enemy solves the following continuous multicommodity flow network interdiction problem.

$$\underset{w\in W}{\text{Minimize}} \quad \text{maximize} \quad \sum_{k\in K}\sum_{i\in A} f_i^k v_i^k, \tag{10a}$$

$$\text{subject to} \quad \sum_{k\in K} v_i^k \le q_i x_i w_i, \quad \forall i \in A, \tag{10b}$$

$$\sum_{i\in FS(j)} v_i^k - \sum_{i\in RS(j)} v_i^k = d_j^k \quad \forall k \in K \ \forall j \in N, \tag{10c}$$

$$v_i^k \ge 0 \quad \forall i \in A, \ \forall k \in K, \tag{10d}$$

where $W = \{w \in R^{|A|} : \sum_{i\in A} b_i(1 - w_i) = B,\ 0 \le w_i \le 1,\ \forall i \in A\}$. Note that the inner maximization problem has an optimal solution given any enemy's choice of w (perhaps with the use of dummy arcs that ensure the existence of a feasible multicommodity flow). Taking the linear dual of the inner maximization problem, where β_i is associated with (10b) $\forall i \in A$ and α_j^k is associated with (10c) $\forall j \in A$ and $\forall k \in K$, the interdiction problem can be reformulated as the following disjointly constrained bilinear program (BLP).

$$\text{Minimize} \quad \sum_{k\in K}\sum_{j\in N} d_j^k \alpha_j^k + \sum_{i\in A}(q_i x_i) w_i \beta_i, \tag{11a}$$

$$\text{subject to} \quad \alpha_{f(i)}^k - \alpha_{t(i)}^k + \beta_i \ge f_i^k, \quad \forall i \in A, \ \forall k \in K, \tag{11b}$$

$$\alpha_j^k \text{ unrestricted}, \ \forall j \in N, \ \forall k \in K, \tag{11c}$$

$$\beta_i \ge 0, \quad \forall i \in A, \tag{11d}$$

$$w \in W, \tag{11e}$$

where arc i exits node $f(i)$ and enters node $t(i)$. Note that the enemy's decision variables (w) and the designer's dual variables associated with arc capacity constraints (β) constitute bilinear terms in the objective function. Also, the x-variables appear only in the objective function.

One important property of this BLP formulation is that a global optimum can be found among pairs of extreme points from respective feasible regions. In particular, the enemy has a single knapsack constraint besides bounds on variables, and hence, each extreme point has only one basic variable (whose value lies in the interval [0,1]) while all other nonbasic variables are either set at their lower bound of 0 or upper bound of 1. Exploiting this fact, Lim and Smith [17] proposed a partitioning algorithm that solves linearized mixed integer subproblems obtained by designating one variable as basic. Letting w_h be basic and substituting $w_h = \frac{\sum_{i\in A\setminus\{h\}} b_i(1-w_i)+b_h-B}{b_h}$ in (11), we have the following BLP in which w_h is eliminated and w_i, $\forall i \in A \setminus \{h\}$, are now binary.

$$\text{Minimize} \quad \sum_{k\in K}\sum_{j\in N} d_j^k \alpha_j^k + \sum_{i\in A\setminus\{h\}} (q_i x_i) w_i \beta_i + (q_h x_h)\beta_h \frac{\sum_{i\in A\setminus\{h\}} b_i(1 - w_i) + b_h - B}{b_h}, \tag{12a}$$

$$\text{subject to} \quad (11b), (11c), \text{ and } (11d) \tag{12b}$$

$$0 \le \frac{\sum_{i\in A\setminus\{h\}} b_i(1 - w_i) + b_h - B}{b_h} \le 1, \tag{12c}$$

$$w_i \text{ binary}, \ \forall i \in A \setminus \{h\}. \tag{12d}$$

Let us substitute $w_i\beta_i$ and $w_i\beta_h$ by s_i and t_i, respectively. Accordingly, we add the following standard linearization constraints for $i \in A \setminus \{h\}$:

$$\beta_i + \overline{\beta_i} w_i - s_i \leq \overline{\beta_i}, \tag{13a}$$

$$t_i - \beta_i \leq 0, \tag{13b}$$

$$t_i - \overline{\beta_i} w_i \leq 0, \tag{13c}$$

$$s_i,\ t_i \geq 0, \tag{13d}$$

where $\overline{\beta_i}$ is an upper bound on β_i. Note that we have removed upper bounds for s_i and lower bounds for t_i since they are not necessary for this particular problem. (See [17] for a discussion on possible upper bounding schemes for β_i, and for a more detailed discussion on the sufficiency of the linearization constraints to ensure $s_i = w_i\beta_i$ and $t_i = w_i\beta_h\ \forall i$.) Now, the resulting linearized mixed integer subproblem can be solved via widely used discrete optimization techniques such as branch-and-bound/cut methods. Finally, an exact solution can be identified after solving $|A|$ subproblems, one corresponding to each possible value of $h \in A$.

Note that there exists a finite number of pairs of extreme points for disjoint polyhedral sets in (11b)–(11e). Let Π denote the set of such pairs. Furthermore, let $\phi_\pi(x)$ denote the objective function value of (11a) at $\pi \in \Pi$ given x (i.e., $\phi_\pi(x)$ is the optimal reward from post-interdiction flows given x, assuming optimal arc interdiction on the part of the enemy). Then, our network design problem can be formulated as follows.

$$\text{Maximize}\ \ \rho \sum_{k\in K}\sum_{i\in A} f_i^k u_i^k + (1-\rho)\min\{\phi_\pi(x) : \pi \in \Pi\} - \sum_{i\in A} c_i x_i, \tag{14a}$$

$$\text{subject to}\ \ (1b), (1c), (1e), (1j), \quad \text{and} \quad u_i^k \geq 0 \quad \forall i \in A\ \ \forall k \in K. \tag{14b}$$

Observing the linearity of $\phi_\pi(x)$ with respect to x given π, we have that $\min\{\phi_\pi(x) : \pi \in \Pi\}$ is concave with respect to x. Therefore, we can prescribe a cutting-plane algorithm (or outer-linearization method), which we call **BCPA**, that generates Benders cuts in an iterative fashion. At iteration j of BCPA, we have the following master program.

$$\text{Maximize}\ \ \rho \sum_{k\in K}\sum_{i\in A} f_i^k u_i^k + (1-\rho) z - \sum_{i\in A} c_i x_i, \tag{15a}$$

$$\text{subject to}\ \ (1b), (1c), (1e), (1j), \quad \text{and} \quad u_i^k \geq 0 \quad \forall i \in A\ \ \forall k \in K, \tag{15b}$$

$$z \leq \phi_\pi(x) \quad \forall \pi \in \Pi_j, \tag{15c}$$

$$z \quad \text{unrestricted}, \tag{15d}$$

where $\Pi_j \subseteq \Pi$ is the set π-vectors obtained in prior iterations by solving the interdiction problem. A detailed algorithm can be described as follows.

Algorithm BCPA

Step 0 Set $\Pi^1 = \emptyset$ and $j = 1$.
Step 1 Solve the problem (14) to obtain a solution x^j and z^j.
Step 2 Given x^j, solve the problem (11) to obtain a solution π and its objective value $\phi_\pi(x^j)$.
Step 3 If $z^j \leq \phi_\pi(x^j)$, then x^j is optimal and stop. Else, put $\Pi^{j+1} = \Pi^j \cup \{\pi\}$, increment $j \leftarrow j+1$, and return to Step 1.

4 Computational study

In this section, we report a computational study that investigates the efficacy of the proposed algorithms. Recall that case 1 is relatively easier to solve due to the simple structure of the enemy interdiction problems, whereas case 3 requires a considerable amount of computational effort because we need to solve $|A|$ mixed-integer subproblems at each iteration. Hence, we experiment with different sets of test instances having different sizes as summarized in Table 1. Five instances from each set are generated in the following manner.

Given a commodity $k \in K$, the number of origin nodes is randomly selected over the integers $1, 2, \ldots, \lceil 0.25|N| \rceil$, and so is the number of destination nodes. Using these numbers, nodes are randomly designated as origin or destination nodes (but not both) of commodity k. At each origin node, the supply of commodity k is randomly chosen from integer values in [1, 25]. The total supply of the commodity is then randomly assigned to the corresponding destination nodes so that the total demand is equal to the total supply for commodity k. After generating these values for all commodities, a set of arcs, A, is built by randomly generating paths for all combinations of origin-destination (o-d) pairs for each commodity. If there exists a node at which no arc is incident, we randomly generated an o-d path that contains this node, and added all arcs in this path to A (if they are not already present). It is thus possible to generate some networks with the same numbers of nodes and commodities that have considerably different numbers of arcs in the network. Since the number of integer variables depends on the number of arcs in A, a large deviation in $|A|$ among a common set of test instances is not desirable for making meaningful computational comparisons. In order to maintain consistency, we enforced the existence of a minimum number of arcs for each instance belonging to the same test set. The minimum number of arcs for each instance in a test set is displayed in the column MINARC in Table 1. If the number of arcs in A is less than MINARC, we repeatedly generated random o-d paths and added those arcs not already in the arc set to A, until the total number of arcs was at least MINARC. Ranges of the actual numbers of arcs generated in these test instances are reported in the fifth column of Table 1.

For each arc $i \in A$, we randomly generated c_i, b_i, q_i, and $f_i^k, \forall k \in K$ from integer values in the ranges [10, 20], [10, 20], [20, 60], and [1, 5], respectively. C was set as $\sum_{i \in A} c_i$ multiplied by a random factor between 0.8 and 1, while B was set as $\sum_{i \in A} b_i$

Table 1 Summary of test problems from general network topologies

Set	Number of nodes	Number of commodities	MINARC	Number of arcs	Tested cases		
					Case 1	Case 2	Case 3
S1	6	3	13	14–16	o	o	o
S2	6	5	18	18–19	o	o	o
S3	6	7	21	21–25	o	o	o
S4	9	3	30	31–40	o	o	
S5	9	5	45	46–47	o	o	
S6	9	7	60	60–62	o	o	
S7	12	3	70	70–74	o		
S8	12	5	80	80–84	o		
S9	12	7	100	101–108	o		

multiplied by a random factor in [0.1, 0.4]. For each o-d pair of commodity $k \in K$, we computed the maximum flow cost to deliver one unit of commodity k over all o-d paths generated as described above. Then, the reward for shipping one unit of commodity $k \in K$ to the destination node was set as this path flow cost multiplied by a random factor between 1.5 and 2. After rewards for all destination nodes of commodity k are computed, the f_i^k-values are accordingly adjusted if arc i is incident to a destination node of commodity k. All runs were conducted using CPLEX 8.1 C++ Concert Technology on a Sun Fire 280R server with 900 MHz UltraSPARC-III CPU. A time limit is set as 7,200 s for solving each problem.

When reporting results below, the cutting plane algorithm and the single mixed integer program for solving case 1 problems are denoted as CASE1a and CASE1b, respectively. Similarly, CASE2 denotes the partitioning method described in Sect. 3.2. We implemented three variants of the cutting plane algorithm, BCPA, for case 3. Recall that BCPA solves each interdiction problem by solving $|A|$ subproblems in order to generate a new cut. We refer to this standard implementation as CASE3a. However, instead of solving all subproblems, we could stop after identifying a single cutting plane, which would reduce the number of integer programming subproblems that must be solved at each iteration, at the risk of adding a weak cutting plane when a stronger one would be derived from a subsequent subproblem. (Naturally, all subproblems must be solved at the last iteration to prove optimality.) We refer to this implementation as CASE3b. At the other extreme, we can solve all subproblems as done in CASE3a, but instead of adding the cutting plane from the subproblem that yields the optimal interdiction, we can add a cutting plane to the master problem from each subproblem that cuts off the current master problem solution. We call this implementation CASE3c.

We report the average performances of the case 1 implementations in Table 2. The columns labeled "CPU Time" provide the average CPU seconds to solve the five instances in that set, if all five instances could be solved within the 7,200 s time limit. Else, this column displays in parentheses the number of instances that could be solved in this set within the time limit. The columns marked by a are averaged only over those instances that CASE1a solved to optimality, while those marked by b are averaged over those instances that CASE1a did not solve to optimality.

Table 2 Average performances for solving case 1 instances

	CASE1a				CASE1b	
Set	CPU time	CPU time[a]	OPT GAP %[b]	Number of cuts[a]	CPU time	CPU time[a]
S1	0.15	0.15	0	3.2	0.08	0.08
S2	1.30	1.30	0	13.6	0.23	0.23
S3	2.33	2.33	0	17	0.30	0.30
S4	(3)	24.99	2.19	105.7	0.82	0.83
S5	(4)	103.02	5.61	111.5	6.00	2.51
S6	(3)	1103.33	0.36	251.7	16.01	11.96
S7	(1)	428.57	0.91	51	18.93	20.54
S8	(0)	–	2.57	–	350.22	–
S9	(0)	–	0.88	–	1696.43	–

[a] Average taken over instances that CASE1a solved to optimality

[b] Average taken over instances that CASE1a did not solve to optimality

The column labeled "OPT GAP %" reports the average value of 100*(final upper bound—optimal objective value)/(optimal objective value), and "Number of Cuts" reports the average number of cutting planes generated in the course of the CASE1a algorithm. (In CASE1a, a valid upper bound is obtained from the solution of any master program, and a valid lower bound can be obtained by ascertaining the enemy's true interdiction action, given the first-stage network design, and solving the post-interdiction multicommodity flow problem to identify a complete feasible solution.) These results clearly demonstrate that CASE1b is a more efficient approach to solving these problems than CASE1a regardless of the instance size, and that the relative efficiency of CASE1b over CASE1a seems to increase as the problem size increases. It appears that the cutting planes employed by the CASE1a implementation are too weak to force a rapid convergence, as evidenced by the number of cutting planes typically required to solve a single instance.

The CASE2 algorithm terminated with optimal solutions for each instances in sets S1–S4, and for four out of five instances in S5. However, it was unable to solve any instance in S6 within the time limit. Average solution times for those instances solved within the time limit are 4.6, 17.8, 61.3, 200.8, and 3076.7 s for instance sets S1–S5. Note that these instances are much harder to solve using the CASE2 algorithm than using the CASE1b algorithm: Over instance sets S1–S3, CASE2 consumed an average of 57.5, 77.4, and 76.6 times the computational effort required by CASE1b.

We report our computational comparison of the case 3 implementations in Table 3. All three procedures were able to yield optimal solutions for test instances in sets S1 and S2, while only two S3 instances were solved within the time limit, as displayed by the "Solved" column. The column "BND GAP%" is the same as "OPT GAP%" except that it replaces the optimal objective value by the best lower bound obtained, since we did not obtain the optimal solution for these instances. The remaining column labels are the same as those used in Table 2, except that the last column displays both the average number of cuts generated and the average number of iterations required to generate these cuts in parentheses, since multiple cuts are generated at each iteration of the CASE3c implementation. (For CASE3a and CASE3b, the number of cuts equals the number of iterations.) Note that since we do not solve the enemy's problem to optimality in CASE3b (because we stop when any valid inequality is identified), we do not obtain a lower bound on the optimal objective value.

CASE3b consistently outperformed other two methods by consuming 27.0, 39.1, and 29.9% of CPU times required by the second best method for solving instance sets S1, S2, and S3, respectively. Interestingly, CASE3c performs the fewest iterations

Table 3 Average performances for solving case 3

Set	Solved	CASE3a			CASE3b		CASE3c		
		CPU time[a]	BND GAP %[b]	Number of cuts[a]	CPU time[a]	Number of cuts[a]	CPU time[a]	BND GAP %[b]	Number of cuts[a]
S1	5	70.2	–	40.2	15.8	50.6	58.5	–	333.6 (23.4)
S2	5	1449.5	–	170.6	426.4	207.8	1089.6	–	1474.6 (81.8)
S3	2	4880.0	18.5	236	1457	357.0	5498.8	15.5	1263.7 (146)

[a] Average taken over instances solved to optimality

[b] Average taken over instances not solved to optimality

on average, followed by CASE3a. This was anticipated because CASE3c is capable of generating multiple cuts in addition to the one generated by CASE3a that may become binding in future iterations, and because CASE3b does not necessarily use the best cut from the enemy's problem, unlike CASE3a. However, each iteration of the CASE3a and CASE3c algorithms requires the solution of $|A|$ enemy subproblems. Since CASE3b only requires the solution of this number of subproblems in the worst case (occurring at the last iteration and at any iteration in which only the last subproblem encountered yields a cutting plane), this implementation successfully trades-off the solution of more master problems in exchange for solving fewer subproblem.

Additionally, despite the fact that CASE3c requires fewer iterations than CASE3a, and that both algorithms solve $|A|$ subproblems at each iteration, CASE3c requires more computational time than CASE3a on the two S3 instances solved to optimality. This is due to the additional computational effort required to solve the larger master problems encountered by CASE3c, due to the fact that multiple cuts are added to the master problem at each CASE3c iteration. However, the average relative gap of CASE3c is slightly smaller than that of CASE3a on the three instances not solved to optimality. In order to include CASE3b in this comparison, we recalculated the relative gap using the best lower bound between those obtained from CASE3a and CASE3c. The resulting average relative gaps of CASE3a, CASE3b, and CASE3c are 18.4, 12.0, and 14.7%, respectively. Based on these observations, we conclude that CASE3b is the best method of our three procedures.

Finally, for the sake of interest, we report the optimal objective values for test sets S1 and S2 when each case is solved in Table 4. Differences between optimal objective values for cases 1 and 2 are slight, and indicate that neither heuristic interdiction strategy seems to dominate the other. However, when the enemy acts optimally, the designer's profit decreases significantly. In particular, the optimal objective value of case 3 is less than 50% of those obtained from cases 1 and 2 on average. Therefore, the enemy's ability to optimally solve the interdiction problem can indeed be a crucial factor in effectively reducing the network designer's profit.

5 Concluding remarks

The three problems described above are all strongly NP-hard, but can be optimally solved with the use of integer programming and decomposition algorithms. The development of these models is only part of the challenge in the field of network interdiction.

Table 4 Optimal objective values of instances in S1 and S2 for cases 1–3

Set	Instance	Case 1	Case 2	Case 3
S1	1	1547.2	1519.3	668.3
	2	636.6	728.2	428
	3	524.5	488.8	220
	4	1344.5	1307.8	582.5
	5	916.5	890.8	462.2
S2	1	615.7	640	200
	2	770	823.4	343.5
	3	704.4	737.4	225.5
	4	1315	1336.6	739.8
	5	1544.9	1585.4	1327.3

For instance, we can then devise a design algorithms for various other modes of heuristic interdiction, perhaps corresponding to those observed within context-specific applications. Also, recall that the network design problem presented in this paper is strongly NP-hard, regardless of the interdiction strategy being employed by the follower. Hence, another possible avenue to explore for future research involves the analysis of certain network topologies for which these design problems can be solved in polynomial time.

Acknowledgements The authors gratefully acknowledge the support of the Office of Naval Research under Grant Number N00014-03-1-0510 and the Air Force Office of Scientific Research under Grant Number F49620-03-1-0377. They are also grateful for the comments and feedback offered by two anonymous referees.

References

1. Alevras, D., Grötschel, M., Jonas, P., Paul, U., Wessaly, R.: Survivable mobile phone network architectures: Models and solution methods. IEEE Commun. Magazine **36**, 88–93 (1998)
2. Bard, J.: An algorithm for solving the general bilevel programming problem. Math. Oper. Res. **8**, 260–272 (1983)
3. Bard, J.: Practical Bilevel Optimization: Algorithms and Applications. Kluwer Academic Publishers, Norwell, MA, (1998)
4. Bard, J., Falk, J.: An explicit solution to the multilevel programming problem. Comput. Oper. Res. **9**, 77–100 (1982)
5. Bialas, W., Karwan, M.: Two-level linear programming. Manag. Sci. **30**, 1004–1021 (1984)
6. Bracken, J., McGill, J.: Mathematical programs with optimization problems in the constraints. Oper. Res. **21**, 37–44 (1973)
7. Candler, W., Townsley, R.: A linear two-level programming problem. Comput. Oper. Res. **9**, 59–76 (1982)
8. Clarke, L., Anandalingam, G.: A bootstrap heuristic for designing minimum cost survivable networks. Comput. Oper. Res. **22**, 921–934 (1995)
9. Colson, P. M. B., Savard, G.: A cutting plane algorithm for multicommodity survivable network design problems. INFORMS J. Comput. **10**, 1–11, (1998)
10. Colson, P. M. B., Savard, G.: Bilevel programming: a survey. 4OR **3**, 87–107 (2005)
11. Cormican, K. J., Morton, D. P., Wood, R. K.: Stochastic network interdiction. Oper. Res. **46**(2), 184–196 (1998)
12. Fortuny-Amat, J., McCarl, B.: A representation and economic interpretation of a two-level programming problem. J. Oper. Res. Soc. **32**, 783–792 (1981)
13. Fulkerson, D. R., Harding, G. C.: Maximizing minimum source-sink path subject to a budget constraint. Math. Program. **13**(1), 116–118 (1977)
14. Garg, M., Smith, J. C.: Models and algorithms for the design of survivable multicommodity flow networks with general failure scenarios. OMEGA, to appear (2006)
15. Israeli, E., Wood, R. K.: Shortest-path network interdiction. Networks **40**(2), 97–111 (2002)
16. Kydland, F.: Hierarchical decomposition in linear economic models. Manag. Sci. **21**, 1029–1039 (1975)
17. Lim, C., Smith, J. C.: Algorithms for discrete and continuous multicommodity flow network interdiction problems. IIE Trans., to appear (2006)
18. Magnanti, L. T., Wong, R. T.: Network design and transportation planning: Models and algorithms. Transportation Sci. **18**(1), 1–55 (1984)
19. Minoux, M.: Network synthesis and optimum network design problems: Models, solution methods and applications. Networks **19**(3), 313–360 (1989)
20. Myung, Y. S., Kim, H. J., Tcha, D. W.: Design of communication networks with survivability constraints. Manag. Sci. **45**, 238–252 (1999)
21. Ouveysi, I., Wirth, A.: On design of a survivable network architecture for dynamic routing: Optimal solution strategy and efficient heuristic. Eur. J. Oper. Res. **117**, 30–44 (1999)
22. Paul, G., Tanizawa, T., Havlin, S., Stanley, H. E.: Optimization of robustness of complex networks. Eur. Phys. J. B **38**, 187–191 (2004)

23. Rajan, D., Atamtürk, A.: Survivable network design: routing of flows and slacks. In: Anandalingam, G., Raghavan, S. (ed) Telecommunication Network Design and Management, pp. 65–81 Kluwer Academic Publishers, Dordrecht, Boston, London, (2002)
24. Ríos, M., Marianov, V., Gutierrez, M.: Survivable capacitated network design problem: New formulation and lagrangian relaxation. J. Oper. Res. Soc. **51**, 574–582 (2000)
25. Shaio, J.: Constraint generation for network reliability problems. Ann. Oper. Res. **106**, 155–180 (2001)
26. Smith, J. C., Sudargho, F., Lim, C.: Survivable network design under various interdiction scenarios, pp. 225–230. San José, Spain (2005)
27. Vicente, L., Calamai, P.: Bilevel and multilevel programming—a bibliography review. J. Glob. Optim. **5**, 291–306 (1994)
28. Wollmer, R.: Removing arcs from a network. Oper. Res. **12**(6), 934–940 (1964)
29. Wood, R. K.: Deterministic network interdiction. Math. Comput. Model. **17**(2), 1–18 (1993)