

A Network Diversion Vulnerability Problem

Ariel Cintron-Arias^{*} Norman Curet[†] Lisa Denogean[‡] Robert Ellis[§]
Corey Gonzalez[¶] Shobha Oruganti^{||} Patrick Quillen^{**}

July 28, 2000

1 Introduction

This paper is concerned with analyzing the susceptibility of a communications network to a specific type of attack. Attacks may consist of physically disabling communication links or flooding links with information in order to prevent legitimate communications. Large-scale denial of service attacks have been prevalent in recent years, affecting many commercial and e-commerce sites [11].

We focus on network diversion attacks that disable network links and force information to flow across one or more specified “diversionary links”. A diversionary link may be identified as a link that is especially vulnerable in the network, possibly susceptible to tapping or other malice from outside sources. Our goal is to compute the minimum amount of resources needed to launch a successful diversion attack in order to determine the network’s vulnerability. Our paper’s contribution is to develop a methodology for analyzing this vulnerability. The end result is an automated procedure that can be employed as a high-level strategic network analysis tool [3].

We model a given network as a directed graph containing a set of nodes and arcs. Each arc contains a cost parameter indicating the amount of resources required to disable or “cut” that arc from the network. Given a specified source node representing the transmission point and a sink node representing the reception point of communication through the network, we would like to determine a minimum cost set of arcs to cut that forces all flow from source to sink across at least one arc in a specified diversion set. In this paper we consider one diversionary arc and formulate this problem as an integer linear programming (IP) model.

Unfortunately, general purpose integer programming techniques are ineffective in providing a solution to the network diversion problem [3]. Difficulties arise when we consider large networks consisting of thousands of nodes and links due to the combinatorial explosion of feasible solutions in the search space. The main contribution of this paper is to offer a decomposition algorithm that is capable of exploiting the special structure of the underlying integer programming model. While, in general, integer programming problems are NP-hard, we found that good solutions can be found efficiently through our

^{*}Cornell University, email: ac103@cornell.edu

[†]National Security Agency, email: curet@math.umbc.edu

[‡]Cornell University, email: lrd8@cornell.edu

[§]University of California, San Diego, email: rellis@math.ucsd.edu

[¶]University of Maryland, email: corey@wam.umd.edu

^{||}Mississippi State University, email: so1@math.msstate.edu

^{**}University of Kentucky, email: quillen@ms.uky.edu

Lagrangian Relaxation decomposition algorithm. Indeed, in some instances, the solutions we found were provably optimal.

2 Diversion Model

2.1 Model

Consider a connected, directed graph $G = (\mathcal{N}, \mathcal{A})$, where \mathcal{N} is the set of nodes and \mathcal{A} is the set of arcs. Each arc is represented by the notation $e = (t(e), h(e))$, where $t(e)$ is the tail node and $h(e)$ is the head node. Denote the source node as s and the sink node as t and assume that there exists a directed path from s to t . We consider diversionary arc d , where $v = t(d)$ is the tail node of the diversionary arc and $w = h(d)$ is the head node. For any subset of the nodes X with $s \in X$, define an s - t cut as the set of arcs $[X, \bar{X}] = \{e \in \mathcal{A} : t(e) \in X, h(e) \in \bar{X}\}$, where $s \in X, t \in \bar{X}, X \cup \bar{X} = \mathcal{N}$, and $X \cap \bar{X} = \emptyset$. For each node $i \in \mathcal{N}$ we define $\delta^+(i) = \{e \in \mathcal{A} : t(e) = i\}$ and $\delta^-(i) = \{e \in \mathcal{A} : h(e) = i\}$. The problem is stated algebraically as follows:

$$\text{Minimize } \sum_{e \in \mathcal{A}} c_e z_e \quad \text{subject to}$$

$$y_{t(e)} - y_{h(e)} + z_e \geq 0, \quad \forall e \in \mathcal{A} \quad (2.1)$$

$$\sum_{e \in \delta^+(i)} x_e - \sum_{e \in \delta^-(i)} x_e = p_i = \begin{cases} 1, & \text{if } i=s \text{ or } w; \\ -1, & \text{if } i=v \text{ or } t; \\ 0, & \text{otherwise,} \end{cases} \quad \forall i \in \mathcal{N} \quad (2.2)$$

$$x_e + z_e \leq 1, \quad \forall e \in \mathcal{A} \quad (2.3)$$

$$y_s = 0, \quad y_t = 1, \quad (2.4)$$

$$x_e, y_i, z_e \in \{0, 1\}, \quad \forall e \in \mathcal{A} \text{ and } i \in \mathcal{N},$$

where we define $c_e \geq 0$ as the cost of cutting arc e . We interpret y , z , and x as follows. If $y_i = 0$ for node i , then $i \in X$; otherwise, $i \in \bar{X}$. If $z_e = 1$ for arc e , then $e \in [X, \bar{X}]$ and thus arc e is in the cut set; otherwise arc e is not cut. If $x_e = 1$ for arc e , then e is used along a flow-preserving path from either s to v or from w to t ; otherwise, e is not used in such a flow-preserving path.

The problem stated above is to find a minimum cost s - t cut subject to cut, flow and linking constraints. Constraints (2.1) are the cut constraints which ensure that for any arc e , if $h(e) \in \bar{X}$ then either $t(e) \in \bar{X}$ or arc e is cut. Minimizing the nonnegative objective function over (2.1), together with (2.4), ensures that all cut edges will have $t(e) \in X$ and $h(e) \in \bar{X}$. Flow constraints (2.2) provide flow-preserving paths from node s to node v and from node w to node t . Linking constraints (2.3) ensure that arcs can be part of the cut or the flow-preserving path, but not both. Constraint (2.4) forces $s \in X$ and $t \in \bar{X}$. In the resulting solution, the diversionary edge $(v, w) \in [X, \bar{X}]$ is restored, and we have at least one s - t path that uses exactly those arcs $\{e : x_e = 1\} \cup \{d\}$.

Now, we convert the problem to matrix form.

$$\text{Minimize } \mathbf{c}^T \mathbf{z} \quad \text{subject to} \quad (2.5)$$

$$A^T \mathbf{y} + \mathbf{z} \geq \mathbf{0}, \quad (2.6)$$

$$A\mathbf{x} = \mathbf{p}, \quad (2.7)$$

$$\mathbf{x} + \mathbf{z} \leq \mathbf{1}, \quad (2.8)$$

$$y_s = 0, \quad y_t = 1, \quad (2.9)$$

$$\mathbf{x}, \mathbf{z} \in \{0, 1\}^{|\mathcal{A}|}, \quad \mathbf{y} \in \{0, 1\}^{|\mathcal{N}|}, \quad (2.10)$$

where we define the following: \mathbf{z} is a binary arc-labeling vector (length $|\mathcal{A}|$); \mathbf{c} is the nonnegative vector of costs to cut the arcs (length $|\mathcal{A}|$); \mathbf{y} is a binary node-labeling vector (length $|\mathcal{N}|$); A is the node-arc incidence matrix of the graph (size $|\mathcal{N}| \times |\mathcal{A}|$); \mathbf{x} is a binary arc-labeling vector (length $|\mathcal{A}|$); \mathbf{p} is a vector indicating node flow requirements (length $|\mathcal{N}|$); and $\mathbf{1}$ is the all one's vector (length $|\mathcal{A}|$).

2.2 Complications

Difficulties arise when considering the constraint matrix

$$B = \begin{bmatrix} A^T & I & 0 \\ 0 & 0 & A \\ 0 & I & I \end{bmatrix}.$$

A matrix is *totally unimodular* provided the determinant of each square submatrix is 1, -1 , or 0. The matrix B is not totally unimodular. This implies that optimal extreme point solutions to a linear programming relaxation of this problem are no longer guaranteed to be integer [8]. However, submatrices of B still retain a totally unimodular structure.

We define $C = \begin{bmatrix} A^T & I & 0 \\ 0 & 0 & A \end{bmatrix}$, and similarly, define $D = \begin{bmatrix} A^T & I & 0 \\ 0 & I & I \end{bmatrix}$, where C

represents constraints (2.6) and (2.7) and D represents (2.6) and (2.8). It is known [1] that the node-arc incidence matrix A is totally unimodular.

Proposition: C and D are totally unimodular.

Proof:

The proof is immediate by noting that any square submatrix of C or D is in block diagonal form with each block unimodular. \square

Further difficulties arise when we consider the size of the problems that we wish to solve. Very large problems admit exceptionally large solution spaces, which in turn limit the effectiveness of traditional integer programming techniques, such as cutting plane methods and branch-and-bound strategies.

In the following section, we develop an algorithm to find approximate solutions to IP (2.5)-(2.10). However, the computations are a major consideration since this problem is NP-hard, which we now discuss.

NP is the class of problems for which a solution can be verified in polynomial time. For many problems in the class NP, there are no known polynomial time algorithm solutions. In fact, it is suspected that these problems are intractable; that is, polynomial time algorithmic solutions do not exist. A problem is said to be NP-hard if it can be used to solve any other NP problem via polynomial time reduction. A problem is NP-complete if it is both NP-hard and in NP. The optimization variant of an NP-complete recognition problem is in NP-hard, see [10]. In order to show that our network diversion IP problem is NP-hard, we use a result of Fortune, Hopcroft, and Wyllie [5]. Therein they show that finding a simple path between two designated nodes that includes a specified arc is NP-complete. It is straightforward to provide a polynomial time reduction of the network diversion problem to this simple path identification problem.

3 Approach

3.1 Lagrangian Relaxation

We use Lagrangian relaxation to relax the linear constraints (2.8). In order to do this, we bring the constraints up into the objective function (2.5) and multiply these constraints by an associated Lagrange multiplier vector, $\boldsymbol{\mu}$. The new relaxed Lagrangian subproblem becomes

$$L(\boldsymbol{\mu}) = \min_{\mathbf{x}, \mathbf{y}, \mathbf{z}} [\mathbf{c}^T \mathbf{z} + \boldsymbol{\mu}^T (\mathbf{x} + \mathbf{z} - \mathbf{1})] \quad \text{subject to}$$

$$\begin{aligned} A^T \mathbf{y} + \mathbf{z} &\geq \mathbf{0}, \\ A \mathbf{x} &= \mathbf{p}, \\ y_s &= 0, & y_t &= 1, \\ \mathbf{x}, \mathbf{z} &\in \{0, 1\}^{|\mathcal{A}|}, & \mathbf{y} &\in \{0, 1\}^{|\mathcal{N}|}, \end{aligned}$$

where $\mathbf{1}$ is the column vector of ones. We call $L(\boldsymbol{\mu})$ the Lagrangian function.

However, solving this relaxed subproblem may not give us feasible solutions to the original problem. But, we do know that for any nonnegative Lagrange multiplier $\boldsymbol{\mu}$, $L(\boldsymbol{\mu})$ is a lower bound for the cost of the optimal solution of the original problem [1]. Therefore, our algorithm will use the subgradient method to find the greatest lower bound over all $\boldsymbol{\mu}$, or:

$$L_* := \max_{\boldsymbol{\mu} \geq 0} L(\boldsymbol{\mu}).$$

Denote the solution to the Lagrangian subproblem by $\mathbf{x}(\boldsymbol{\mu})$, $\mathbf{y}(\boldsymbol{\mu})$ and $\mathbf{z}(\boldsymbol{\mu})$. In the process of finding the greatest lower bound, we would also like to test to see if these solutions are, in fact, the optimal solutions for which we are looking. From [1] if we can find a solution to the relaxed problem, which is feasible in the original unrelaxed problem and which satisfies the following complementary slackness condition:

$$\boldsymbol{\mu}^T (\mathbf{x}(\boldsymbol{\mu}) + \mathbf{z}(\boldsymbol{\mu}) - \mathbf{1}) = 0$$

then that solution is optimal for the original problem.

We can separate the Lagrangian subproblem into two separate optimization problems:

$$\min_{\mathbf{y}, \mathbf{z}} [(\mathbf{c} + \boldsymbol{\mu})^T \mathbf{z}] \quad \text{subject to}$$

$$\begin{aligned} A^T \mathbf{y} + \mathbf{z} &\geq \mathbf{0}, \\ y_s &= 0, & y_t &= 1, \\ \mathbf{z} \in \{0, 1\}^{|\mathcal{A}|}, & & \mathbf{y} \in \{0, 1\}^{|\mathcal{N}|} \end{aligned}$$

and

$$\min_{\mathbf{x}} [\boldsymbol{\mu}^T \mathbf{x}] \quad \text{subject to}$$

$$\begin{aligned} A\mathbf{x} &= \mathbf{p}, \\ \mathbf{x} &\in \{0, 1\}^{|\mathcal{A}|}. \end{aligned}$$

The first optimization problem is a minimum cost s - t cut problem where $(\mathbf{c} + \boldsymbol{\mu})$ is the cost on the associated arcs. The second optimization problem can be viewed as a shortest path problem from s to v and w to t . Each of these problems can be solved using efficient network flow techniques. We use Dijkstra's method [1] for the shortest path problem, and a preflow-push maximum flow solver [6] to solve the minimum cost s - t cut problem.

The key to our algorithm then is in the subgradient method for maximizing $L(\boldsymbol{\mu})$. As in [1], our subgradient for any nonnegative Lagrange multiplier $\boldsymbol{\mu}$ is

$$(\mathbf{x}(\boldsymbol{\mu}) + \mathbf{z}(\boldsymbol{\mu}) - \mathbf{1}).$$

If, after finding the shortest path and the minimum s - t cut for the given $\boldsymbol{\mu}$, we find that our minimum cut includes arcs on our shortest path, we increase the cost of using or cutting those arcs in the path. If, on the other hand, we find that neither of our two network flow algorithms use a given arc, then we lower its cost, and make it more enticing to be used in the next iteration. However, if only one of the network flow algorithms uses the arc, we keep the cost the same, since all of the constraints are satisfied on that arc. Once our subgradient is known, we can then choose an appropriate step size, θ , and adjust the Lagrange multipliers accordingly.

Our basic algorithm is thus:

1. Find the shortest $s - v$ and $w - t$ cardinality paths. If these two paths are not node disjoint, EXIT as the topology of the network may not afford us a solution by this method.
2. Begin loop:
3. On the first pass, set the costs of those arcs along the s - v and w - t paths to be infinite, and find a minimum cost s - t cutset, \mathbf{z} , which includes our diversionary arc. On all other passes, solve the minimum cost s - t cutset with cost vector given by $(\mathbf{c} + \boldsymbol{\mu})$.
4. Find the shortest paths from s to v and w to t with cost (distance) on each arc given by the associated Lagrange multiplier. Set $x_e(\boldsymbol{\mu}) = 1$ for each edge e along these two shortest paths.
5. Compute the subgradient $(\mathbf{x}(\boldsymbol{\mu}) + \mathbf{z}(\boldsymbol{\mu}) - \mathbf{1})$.
6. If $\mathbf{x}(\boldsymbol{\mu}) + \mathbf{z}(\boldsymbol{\mu}) \leq \mathbf{1}$, then if the complementary slackness condition $\boldsymbol{\mu}^T(\mathbf{x}(\boldsymbol{\mu}) + \mathbf{z}(\boldsymbol{\mu}) - \mathbf{1}) = 0$ EXIT with the optimal solution. Otherwise store this solution as an approximate optimal solution.
7. Compute θ , our step size. Calculate our new value of $\boldsymbol{\mu}$, $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \theta \cdot (\mathbf{x}(\boldsymbol{\mu}) + \mathbf{z}(\boldsymbol{\mu}) - \mathbf{1})$.

8. End loop

We use the following heuristic to determine stepsize.

$$\theta_{new} = \frac{\lambda(\text{UB} - L(\boldsymbol{\mu}_{new}))}{\|(\mathbf{x}(\boldsymbol{\mu}) + \mathbf{z}(\boldsymbol{\mu}) - \mathbf{1})\|_1}$$

UB is an upper bound for the total cost which we obtain either by determining a feasible (though not necessarily optimal) cut set or by heuristic techniques. The parameter λ is a value strictly between 0 and 2 and is used to 'reward' a good step, and 'punish' a bad. For every iteration of our loop, we multiply λ by a scalar to indicate this reward or punishment. This approach allows us to stop the loop if λ becomes particularly small, which in turn, leads θ to be small. A small λ is the result of a series of iterations with bad steps, corresponding to slow convergence.

3.2 Heuristic Approach

As an alternative to the Lagrangian Relaxation technique, which provides a lower bound to the minimum cut size and may not directly provide a feasible solution, we can bound the cut size from above heuristically by feasible solutions. Generally, we start with a feasible solution, and look for ways to improve that solution in an iterative fashion. Improvements are done locally in order to maintain computational feasibility.

When a heuristic approximation and Lagrangian Relaxation are combined, we obtain both an upper and lower bound, plus feasible solutions from the heuristic, at the very least. In practice, we may find that the two bounds together give a tight bound on the optimal objective function value, and thus one of the approximations will return an optimal solution.

We now outline one such heuristic approximation algorithm, which is tailored for networks with strong spatial ordering.

1. **Setup.** Find the shortest s - v path and the shortest w - t path separately. Reserve these arcs as flow constraint arcs (for which \mathbf{x} is 1) and compute the minimum cost s - t cut on the remaining arcs.
2. **Refinement.** Loop through all arcs in both paths computed above. For each arc e do the following. Recompute the s - v and w - t paths, except forcing a detour around e . Recompute the minimum cost s - t cut.
3. **Decide search space.** Remember better solutions produced in the loop, and use one or more of these as a new starting solution for refinement in step 2.

A large amount of latitude exists in the implementation of this algorithm. One obvious choice is how many solutions, generated by the loop step, to keep around. We might also compute smallest cardinality paths (i.e., smallest number of arcs involved), instead of shortest paths. We might compute smallest cardinality paths whose smallest arc cost is as large as possible, which would tend to relinquish arcs of smaller cost to be used in minimizing the objective function.

4 Results

We employ Lagrangian Relaxation, and our approach is applied to networks that consist of up to 10,002 nodes and 39,800 arcs. In each case we report the cost of the best solution found using Lagrangian Relaxation, where “ * ” indicates the solution was provably optimal.

Network	# Nodes	# Arcs	Cost
Net1 [7]	24	34	10,752*
Net5 [9]	34	158	9*
Net6 [9]	42	188	28*
Grid20	402	1560	83
Grid50	2502	9900	351*
Grid100	10,002	39,800	769

Table: Lagrangian Relaxation Solutions

The above networks are compiled from various sources. Net1 is the graph representation of a military C2C network, while Net5 and Net6 are representations of a public telephone switch network in the Washington, DC area. The last three grids represent $N \times N$ grid networks taken from [4].

These results demonstrate the validity of our original model formulation and solution approach. Not only are we duplicating the results that Curet [3] found from using a linear programming relaxation, we are finding solutions with smaller cost in the case of Grid20 and Grid50.

5 Conclusion

We have formulated and implemented an algorithm that solves network diversion problems for which strong duality with the relaxed problem hold. For those problems which fail to exhibit strong duality, the Lagrangian relaxation algorithm may not find a feasible diversion solution, but yields a lower bound to the cost of a diversion attack. In this case, we provide a heuristic approach that gives an upper bound. We could then use these bounds to limit the search space of a traditional branch and bound algorithm to zero-in on an optimal solution.

The case for multiple diversionary arcs separates into multiple cases of single diversionary arcs. We only need to investigate the minimum constrained cut set for each diversionary arc and choose the one with minimum cost.

Left to future investigation is the Lagrangian relaxation problem formed by relaxing the flow constraints (2.7) instead of the linking constraints (2.8). The linear program given by this Lagrangian problem would also admit integer solutions, as the cut constraints (2.6) and the linking constraints (2.8) together retain total unimodularity, as shown previously.

Another approach to this problem could be the identification of valid inequalities which limit the solution space. A goal of such an approach would be to develop the inequalities *a priori*, based solely on graph properties [2].

Finally, we leave a more efficient implementation of this particular algorithm to future work. Such an implementation could increase performance and allow for the solution of larger problems in real time.

Acknowledgements

The authors wish to express their gratitude to the National Security Agency, the Institute for Mathematics and its Applications, the University of Minnesota and Dr. Norman Curet for their support of this project.

References

- [1] Rav Ahuja, James Orlin and Tom Magnanti, *Network Flows*, Prentice Hall, New Jersey, 1993.

- [2] Lorenzo Brunetta, Michele Conforti and Matteo Fischetti, A Polyhedral Approach to an Integer Multicommodity Flow Problem, preprint, June 28, 1999.
- [3] N. Curet, The Network Diversion Model, to appear in *Military Operations Research*, 2001.
- [4] N. Curet, M. Gaston and J. DeVinney, An Efficient Network Flow Code for Finding All Minimum Cost s-t Cutsets, to appear in *Computers and Operations Research*, 2001.
- [5] S. Fortune, J. Hopcroft and J. Wyllie, The Directed Subgraph Homeomorphism Problem, *Theoretical Computer Science*, 1980; 111-121.
- [6] A. Goldberg and B. Cherkassky, On Implementing Push-Relabel Method for the Maximum Flow Problem, *Algorithmica*, 1994; 390-410.
- [7] Leinhart, J. "A Network Disruption Modeling Tool", AFIT M.S. Thesis, 1998.
- [8] George Nemhauser and Laurence Wolsey, *Integer and Combinatorial Optimization*, Wiley, 1988.
- [9] Olson, A. "Classifying PSTN Switching Stations", Naval Postgraduate School M.S. Thesis, 1998.
- [10] Papadimitiou, C., and Steiglitz, K, *Combinatorial Optimization*, Prentice-Hall, 1982.
- [11] URL: <http://www.cert.org>, Network Attacks.