# Reformulation and Sampling to Solve a Stochastic Network Interdiction Problem

UDOM JANJARASSUK, JEFF LINDEROTH

*Department of Industrial and Systems Engineering,*
*Lehigh University, 200 W. Packer Ave. Bethlehem, PA 18015*

udj2@lehigh.edu · jtl3@lehigh.edu

January 31, 2006

**Abstract**

The Network Interdiction Problem involves interrupting an adversary's ability to maximize flow through a capacitated network by destroying portions of the network. A budget constraint limits the amount of the network that can be destroyed. In this paper, we study a stochastic version of the network interdiction problem in which the successful destruction of an arc of the network is a Bernoulli random variable, and the objective is to minimize the maximum expected flow of the adversary. Using duality and linearization techniques, an equivalent deterministic mixed integer program is formulated. The structure of the reformulation allows for the application of decomposition techniques for its solution. Using a parallel algorithm designed to run on a distributed computing platform known as a computational grid, we give computational results showing the efficacy of a sampling-based approach to solving the problem.

## 1 Introduction

The network interdiction problem can be thought of as a game played on a network consisting of two players, a *leader* and a *follower*. The leader has a fixed budget that can be used to deteriorate or destroy portions of the network, an action called an *interdiction*. Examples of interdictions include removing arcs from the network and reducing the capacity of arcs in the network. After the leader has performed an interdiction, the follower solves an optimization problem on the modified network. The follower may try to find a shortest path, as in [8, 9], or the follower may maximize flow through the network, which is the case studied this work. The leader's objective is to thwart the follower; to maximize the follower's shortest path or minimize the follower's maximum flow. Network interdiction is an instance of a static Stackelberg game [29], and the problem has been studied for military application [34, 24], intercepting illicit materials [32, 26], and in designing robust telecommunication networks [11].

Stochastic Network Interdiction occurs when one or more of the components of the network interdiction problem are not known with certainty. Cormican, Morton, and Wood [4] provide a nice introduction to many variations of stochastic network interdiction, including problems in which the interdiction attempt may succeed or fail, the arc capacities or topology of the network are uncertain,

and in which multiple interdiction attempts may take place. In order to obtain solutions to the problem, Cormican, Morton, and Wood develop a sequential approximation technique. Also in their study, they state that

> "A large-scale deterministic equivalent binary integer program may be formed by (a) reformulating the problem as a simple minimization problem involving binary interdiction variables and binary second-stage variables... and (b) enumerating all possible realizations of the [uncertainty]... Unfortunately, solving such models would be computationally impractical for all but the smallest problems."

We intend to test the validity of these statements. Stochastic programs with integer recourse (second-stage) variables are nearly intractable, but we will show that there is no need to introduce binary variables in the second stage in such a reformulation. Second, recent theoretical and empirical evidence has suggested that a *sample average* or *sample-path* approach can be an extremely effective technique for solving two-stage stochastic problems with a discrete structure [28, 14, 17]. We will use a sample-average approach applied to our formulation, solve the resulting instances, and statistically test bounds on the optimal solution value using a distributed computational platform known as a *computational grid*.

Stochastic network interdiction has also been studied by Pan, Charlton, Morton [26]. They formulate the problem of identifying locations for installing smuggled nuclear material detectors as a two-stage stochastic mixed-integer program with recourse, and showed that the problem is NP-Hard. In Held, Hemmecke, Woodruff [12], the stochastic network interdiction problem is shown to be solved effectively by applying a decomposition-based method. Our approach to solving the problem is akin to that proposed by Cormican in her master's thesis [3], where she proposes using Bender's decomposition to solve a deterministic version of the problem. The duality-based approach we take in this work is also similar to that appearing in the work of Israeli and Wood [13], who solve the maximize-shortest-path version of the network interdiction problem.

The remainder of the paper is divided into three sections. In Section 2, we present a mixed integer linear program for the deterministic equivalent of the stochastic network interdiction problem. Section 3 contains a review of sample average approximation and describes a decomposition-based algorithm for solving sampled versions of the model. Computational results are presented in Section 4.

## 2 Formulation

The Stochastic Network Interdiction Problem (SNIP) is defined on a capacitated directed network $G = (V, A)$, with specified source node $r \in V$ and sink node $t \in V$. The capacity of arc $(i, j) \in A$ is $u_{ij}$. The realizations of uncertainty are captured in a countable set of scenarios $S$, in which each element $s \in S$ occurs with probability $p_s$. There is a finite budget $K$ available for interdiction, and the cost of interdicting on any arc $(i, j) \in A$ is $h_{ij}$. To write a mathematical program for SNIP, we define the binary decision variables

$$x_{ij} = \begin{cases} 1 & \text{if interdiction occurs on arc } (i,j) \in A, \\ 0 & \text{otherwise.} \end{cases}$$

With these definitions, SNIP can be written as the formulation $\mathcal{F}_1$:

$$
\begin{aligned}
(\mathcal{F}_1) \qquad & z_{\text{SNIP}} = \min \sum_{s \in S} p_s f_s(x) \\
\text{subject to} \quad & \sum_{(i,j) \in A} h_{ij} x_{ij} \leq K, \\
& x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A,
\end{aligned}
$$

where $f_s(x)$ is the maximum flow from node $r$ to node $t$ in scenario $s$ if interdictions occur on the arcs indicated by the $\{0,1\}$-vector $x$. The maximum flow value $f_s(x)$ is the solution of a different optimization problem, which can be written down with the help of the following definitions:

- $A' = A \cup \{r,t\}$,
- $\xi_{ijs} = \begin{cases} 1 & \text{if interdiction on arc } (i,j) \in A \text{ in scenario } s \in S \text{ would be successful,} \\ 0 & \text{otherwise,} \end{cases}$
- $y_{ij}$: flow on arc $(i,j)$.

We focus for the time-being on the case in which interdiction success is a Bernoulli random variable ($\xi$), postponing extensions and additional formulations to Section 2.1. The maximum flow from node $r$ to node $t$ in scenario $s$ with interdictions on arcs indicated by $x$ can be expressed as the solution to the optimization problem PMF$_s(x)$:

$$
\begin{aligned}
\text{PMF}_s(x) \qquad & f_s(x) = \max y_{tr} \\
\text{subject to} \quad & y_{ij} \leq u_{ij}(1 - \xi_{ijs} x_{ij}) \ \forall (i,j) \in A, && (1) \\
& y_{tr} + \sum_{j \in V \mid (j,r) \in A} y_{jr} - \sum_{j \in V \mid (r,j) \in A} y_{rj} = 0, && (2) \\
& -y_{tr} + \sum_{j \in V \mid (j,t) \in A} y_{jt} - \sum_{j \in V \mid (t,j) \in A} y_{tj} = 0, && (3) \\
& \sum_{j \in V \mid (j,i) \in A} y_{ji} - \sum_{j \in V \mid (i,j) \in A} y_{ij} = 0 \qquad \forall i \in V \setminus \{r \cup t\}, && (4) \\
& y_{ij} \geq 0 \qquad \forall (i,j) \in A'. && (5)
\end{aligned}
$$

Constraints (1) force the flow to be zero if arc $(i,j)$ was chosen for interdiction and if the interdiction was successful. Constraints (2)—(4) are simply the flow balance constraints of the max flow problem, which for subsequent ease of notation, we will write as $Ny = 0$, for an appropriate network matrix $N$.

The dual linear program of $\text{PMF}_s(x)$ is $\text{DMF}_s(x)$.

$$\begin{aligned}
\text{DMF}_s(x) \quad & \min \sum_{(i,j)\in A} u_{ij}(1-\xi_{ijs}x_{ij})\rho_{ij} \\
\text{subject to} \quad & \pi_r - \pi_t \geq 1, && (6)\\
& \rho_{ij} - \pi_i + \pi_j \geq 0 \quad \forall (i,j) \in A, && (7)\\
& \rho_{ij} \geq 0 \quad \forall (i,j) \in A. && (8)
\end{aligned}$$

Define the polyhedron

$$\mathcal{P}_s(x) \stackrel{\text{def}}{=} \{y \in \mathbb{R}^{|A'|} \mid y \text{ satisfies (1)—(5) }\},$$

and the polyhedron

$$\mathcal{D}_s(x) \stackrel{\text{def}}{=} \{(\pi,\rho) \in \mathbb{R}^{|V|} \times \mathbb{R}^{|A|} \mid (\pi,\rho) \text{ satisfies (6)—(8) }\}.$$

The polyhedron $\mathcal{P}_s(x)$ is never empty, since $0 \in \mathcal{P}_s(x)$ for any interdiction $x$ and scenario $s$. We assume that the objective value of the primal problem $\text{PMF}_s(x)$ is bounded $\forall x \in \mathbb{R}_+^{|A|}$, a sufficient condition for this being $u_{ij} < \infty \; \forall (i,j) \in A$. Since $\text{PMF}_s(x)$ is nonempty and bounded for all $x$ and $s$, the strong duality theorem of linear programming states that for a given interdiction $\hat{x}$ and scenario $s$, if there exists $y^* \in \mathcal{P}_s(\hat{x})$ and $(\pi^*,\rho^*) \in \mathcal{D}_s(\hat{x})$ such that

$$y^*_{tr} = \sum_{(i,j)\in A} u_{ij}(1-\xi_{ijs}\hat{x}_{ij})\rho^*_{ij},$$

then $y^*_{tr}$ is the value of the maximum flow from $r$ to $t$, i.e. $y^*_{tr} = f_s(\hat{x})$. With this insight, we can formulate SNIP by defining flow variables $y_{ijs}$ as the flow on arc $(i,j)$ in scenario $s$ and writing the conditions enforcing primal feasibility, dual feasibility, and equality of primal and dual objective functions for each scenario. We call the resulting formulation $\mathcal{F}_2$.

$$(\mathcal{F}_2) \qquad z_{\text{SNIP}} = \min \sum_{s \in S} p_s y_{trs}$$

$$\text{subject to} \quad y_{trs} - \sum_{(i,j) \in A} u_{ij}(1 - \xi_{ijs} x_{ij}) \rho_{ijs} = 0 \qquad \forall s \in S, \tag{9}$$

$$\sum_{(i,j) \in A} h_{ij} x_{ij} \leq K, \tag{10}$$

$$y_{ijs} - u_{ij}(1 - \xi_{ijs} x_{ij}) \leq 0 \qquad \forall (i,j) \in A, \forall s \in S, \tag{11}$$

$$N y_s = 0 \qquad \forall s \in S, \tag{12}$$

$$\pi_{rs} - \pi_{ts} \geq 1 \qquad \forall s \in S, \tag{13}$$

$$\rho_{ijs} - \pi_{is} + \pi_{js} \geq 0 \qquad \forall (i,j) \in A, \forall s \in S, \tag{14}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A,$$

$$y_{ijs} \geq 0 \qquad \forall (i,j) \in A', \forall s \in S,$$

$$\rho_{ijs} \geq 0 \qquad \forall (i,j) \in A, \forall s \in S.$$

The equations (9) constrain the primal and dual objective values of the max flow problem to agree for each scenario. Inequality (10) is the budget constraint. The constraints (11) and (12) are the primal feasibility constraints on the flow, and constraints (13) and (14) are the dual feasibility constraints on the flow.

Equations (9) contain nonlinear terms of the form $x_{ij}\rho_{ijs}$. Since $x_{ij}$ may take only the value 0 or 1 in a feasible solution, the terms can be linearized by introducing a variable $z_{ijs}$ and enforcing the relationship $z_{ijs} = x_{ij}\rho_{ijs}$ through the equivalence

$$x_{ij} \in \{0,1\}, z_{ijs} = x_{ij}\rho_{ijs} \Leftrightarrow z_{ijs} \leq M x_{ij}, z_{ijs} \leq \rho_{ijs}, z_{ijs} \geq \rho_{ijs} + M(x_{ij} - 1), \tag{15}$$

where $M$ is an upper bound on $\rho_{ijs}$ in an optimal solution to SNIP. A derivation of this linearization is given in [33]. Typically, "big M's" are to be avoided in formulations, however, since the variables $\rho_{ijs}$ are simply the dual variables on arcs in a maximum flow problem, a well-known upper bound on these variables is $\rho_{ijs} \leq 1 \; \forall (i,j) \in A, \forall s \in S$.

Introducing this linearization to the formulation $\mathcal{F}_2$ gives a mixed integer linear programming formulation of SNIP, which we label $\mathcal{F}_3$.

$$(\mathcal{F}_3) \qquad z_{\text{SNIP}} = \min \sum_{s \in S} p_s y_{trs}$$

$$\begin{aligned}
\text{subject to} \quad y_{trs} - \sum_{(i,j)\in A} u_{ij}\rho_{ijs} + \sum_{(i,j)\in A} u_{ij}\xi_{ijs}z_{ijs} &= 0 && \forall s \in S, \\
z_{ijs} - x_{ij} &\le 0 && \forall (i,j) \in A, \forall s \in S, \\
z_{ijs} - \rho_{ijs} &\le 0 && \forall (i,j) \in A, \forall s \in S, \\
\rho_{ijs} - z_{ijs} + x_{ij} &\le 1 && \forall (i,j) \in A, \forall s \in S, \\
\sum_{(i,j)\in A} h_{ij}x_{ij} &\le K, && \\
Ny_s &= 0 && \forall s \in S, \\
y_{ijs} - u_{ij}(1 - \xi_{ijs}x_{ij}) &\le 0 && \forall (i,j) \in A, \forall s \in S, \\
\pi_{rs} - \pi_{ts} &\ge 1 && \forall s \in S, \\
\rho_{ijs} - \pi_{is} + \pi_{js} &\ge 0 && \forall (i,j) \in A, \forall s \in S, \\
x_{ij} &\in \{0,1\} && \forall (i,j) \in A, \\
y_{ijs} &\ge 0 && \forall (i,j) \in A', \forall s \in S, \\
\rho_{ijs} &\ge 0 && \forall (i,j) \in A, \forall s \in S, \\
z_{ijs} &\ge 0 && \forall (i,j) \in A, \forall s \in S.
\end{aligned}$$

Formulation $\mathcal{F}_3$ is a mixed integer linear program whose solution gives an optimal solution to the stochastic network interdiction problem. A desirable feature of this formulation is that if $x$ is fixed, then the formulation decouples into $|S|$ independent linear programs. The decomposable structure of the formulation is exploited by the solution algorithm given in Section 3.2.

## 2.1 Extensions

Formulation $\mathcal{F}_3$ lends itself to modeling a wide variety of network interdiction problems. For example, we briefly show how to extend formulation $\mathcal{F}_3$ to model the classes of stochastic network interdiction that are discussed by Cormican, Morton, and Wood [4]. The formulation $\mathcal{F}_3$ is designed for the case when interdiction successes are binary random variables, what Cormican, Morton, and Wood called SNIP(IB). Likewise, by using a random variable $u_{ijs}$ to represent the capacity of arc $(i,j)$ in scenario $s$, $\mathcal{F}_3$ can be extended to handle the cases when both interdictions and arc capacities are random. These are the problem classes SNIP(CB), SNIP(CD), and SNIP(ICB) in [4].

When the probabilities of interdiction success are low, it may be useful to consider allowing multiple interdictions per arc. Cormican, Morton, and Wood call this problem SNIP(IM). If the number of attempts on an arc is limited to two and the success or failure of the first attempt and the second attempt are independent, we can modify $\mathcal{F}_3$ by adding binary decision variables for the second interdiction attempt and parameters (random variables) for the second interdiction success. With these definitions,

the extension is straightforward. However, the resulting formulation contains many more nonlinear terms that must be linearized with the equivalence (15), leading to a formulation with significant more binary variables for SNIP(IM).

# 3 Solution Approach

This section contains a description of our approach to solving the large-scale MIP formulation ($\mathcal{F}_3$) of the stochastic network interdiction problem. The approach is based on a combination of sampling, decomposition, and parallel computing.

## 3.1 Sample Average Approximation

If there are $R$ arcs on which the leader can interdict, and the probability that an interdiction is successful on one arc is independent of its success on other arcs, then there are $2^R = |S|$ realizations of the uncertainty. This exponential growth in the number of scenarios precludes using the formulation $\mathcal{F}_3$ for all but the smallest instances. However, there has been a wealth of recent theoretical and empirical research showing that a *sample average* approach can very accurately solve stochastic programs, even in the case that the number of scenarios is prohibitively large [28, 14, 17, 31, 7].

In the sample-average approach to solving SNIP, instead of minimizing the true expected maximum flow

$$F(x) \stackrel{\text{def}}{=} \sum_{s \in S} p_s f_s(x),$$

we instead minimize a *sample average approximation* $F_T(x)$ to this function,

$$F_T(x) \stackrel{\text{def}}{=} \sum_{s \in T} |T|^{-1} f_s(x).$$

Here, the set $T \subset S$ is drawn from the same probability distribution that defines the original set of scenarios $S$ so that $F_T(x)$ is an unbiased estimator for $F(x)$ for all $x$, i.e. $\mathbb{E}F_T(x) = F(x)\ \forall x$. In the sample average approach to solving SNIP, if

$$X = \{x \in \{0,1\}^{|A|} \mid \sum_{(i,j) \in A} h_{ij} x_{ij} \leq K\}$$

is the set of feasible interdictions, then instead of solving the problem

$$z_{\text{SNIP}} \stackrel{\text{def}}{=} \min_{x \in X} F(x),$$

the approximating problem

$$v_T \stackrel{\text{def}}{=} \min_{x \in X} F_T(x) \tag{16}$$

is solved. The value $v_T$ will approach $z_{\text{SNIP}}$ as the number of scenarios considered in the sample average problem ($|T|$) approaches $|S|$ [5]. However, the solution to $v_T$ is biased in the sense that $\mathbb{E}v_T \leq z_{\text{SNIP}}$ [25, 22].

The value $v_T$ is a random variable, as it depends on the random sample $T$. However, a confidence interval on $v_T$, a lower bound of $z_{\text{SNIP}}$, can be built in the following manner. First, independent samples

$T_1, T_2, \ldots, T_M$, each with the same cardinality ($T_j = N$) are drawn. Next, the associated sample average approximating problems (16) are solved for each sample. The random variables $v_{T_1}, v_{T_2}, \ldots v_{T_M}$ are all independent and identically distributed, so by applying the Central Limit Theorem, a confidence interval on $v_T$ for a sample of size $|T| = N$ can be constructed. With the definitions

$$\mathcal{L} = \frac{1}{M}\sum_{j=1}^{M} v_{T_j}, \text{ and}$$

$$s_{\mathcal{L}} = \left(\frac{1}{M-1}\sum_{j=1}^{M}(v_{T_j} - \mathcal{L})^2\right)^{1/2},$$

a $1-\alpha$ confidence interval for the value of $v_T$ is

$$\left[\mathcal{L} - \frac{t_{\alpha/2,M-1}s_{\mathcal{L}}}{\sqrt{M}}, \mathcal{L} + \frac{t_{\alpha/2,M-1}s_{\mathcal{L}}}{\sqrt{M}}\right], \tag{17}$$

where $t_{\alpha,D}$ is the value of student-t distribution with $D$ degrees of freedom evaluated at $\alpha$. Mak, Morton, and Wood were the first to suggest such a construction [22].

For each solution to the sample average approximation (16) with a sample of size $|T_j| = N$,

$$x_j^* \in \arg\min_{x \in X} F_{T_j}(x),$$

the value $F(x_j^*)$ is a random variable that quantifies the *actual* expected maximum flow if the interdictions are specified from the solution to an SAA instance (16) of size $N$. This quantity $A_N$ is undoubtedly larger than the minimum expected max flow, i.e. $A_N \geq z_{\text{SNIP}}$. Further, $F(x_j^*)$ is an unbiased estimator of $A_N$, and the random values $F(x_1^*), F(x_2^*), \ldots, F(x_M)$ are independent and identically distributed, so we can use these values to obtain a confidence interval on the value $A_N$, an upper bound on $z_{\text{SNIP}}$, in the following manner. First, take a sample $U_j \subset S$ and compute

$$w_{U_j}(x_j^*) \stackrel{\text{def}}{=} \sum_{s \in U_j} |U_j|^{-1} f_s(x_j^*).$$

Again, the sample $U_j \subset S$ must be drawn from the same probability distribution that defines $S$ so that $w_{U_j}(x_j^*)$ is an unbiased estimator of $F(x_j^*)$ and $A_N$. With the definitions

$$\mathcal{U} = \frac{1}{M}\sum_{j=1}^{M} w_{U_j}(x_j^*), \text{ and}$$

$$s_{\mathcal{U}} = \left(\frac{1}{M-1}\sum_{j=1}^{M}(w_{U_j}(x_j^*) - \mathcal{U})^2\right)^{1/2},$$

a $1-\alpha$ confidence interval for the value of $A_N$ is

$$\left[\mathcal{U} - \frac{t_{\alpha/2,M-1}s_{\mathcal{U}}}{\sqrt{M}}, \mathcal{U} + \frac{t_{\alpha/2,M-1}s_{\mathcal{U}}}{\sqrt{M}}\right]. \tag{18}$$

Freimer, Thomas, and Linderoth give a similar construction for a statistical upper bound on a sampled approximation to a stochastic program in [7].

In order for the confidence interval formulae (17) and (18) to be valid, the elements within the samples $T_j$ and $U_j$ need *not* be independent, only distributed like $S$. Thus, variance reduction techniques such as Latin Hypercube sampling or Antithetic variates can be used to construct the samples. In Section 4.5 we give an example of the variance and bias reduction that may be achieved by using a more sophisticated sampling procedure.

## 3.2 Decomposition Algorithm

### 3.2.1 Justification

Initial attempts at solving sampled versions of $\mathcal{F}_3$ relied on solving the instance directly with a commercial MIP solver. Details of the test instances are given subsequently in Section 4. For the three smallest of the test instances, ten different sampled approximations were built for different sample sizes $N$ and solved with CPLEX v9.1. Table 1 shows the average solution times, the average number of nodes of the branch and bound tree, and the percentage of instances whose initial linear programming relaxation yielded an integer solution.

Table 1: CPLEX Solution Times

| Instance | Sample Size | Average CPU Time (sec.) | Average Number of Nodes | % Integer Solution at Root Node |
|---|---|---|---|---|
| SNIP4x4 | 50 | 0.58 | 1.0 | 100 |
| | 100 | 2.22 | 1.0 | 100 |
| | 200 | 25.40 | 1.0 | 100 |
| | 500 | 87.68 | 1.0 | 100 |
| SNIP4x9 | 50 | 8.86 | 1.2 | 90 |
| | 100 | 37.52 | 1.0 | 100 |
| | 200 | 196.45 | 2.0 | 60 |
| | 500 | 1289.22 | 2.2 | 40 |
| SNIP7x5 | 50 | 10.58 | 1.2 | 90 |
| | 100 | 46.90 | 1.3 | 90 |
| | 200 | 171.97 | 1.3 | 90 |
| | 500 | 1387.63 | 1.0 | 100 |

The results of this initial experiment are encouraging, since for most instances, the solution to the initial linear programming relaxation of the formulation $\mathcal{F}_3$ of SNIP was integer-valued. However, the initial results make it clear that for large-scale instances, the time required to solve the linear programming relaxation using a sequential simplex method would be prohibitive. For example, CPLEX required more than 20 minutes simply to solve LP relaxation of the instance SNIP7x5 with $N = 500$. For these reasons, in the next section, we describe a method for solving sampled approximations to SNIP based on decomposing the problem by scenario and solving the independent portions on a parallel computing architecture.

### 3.3 L-Shaped Method

The formulation $\mathcal{F}_3$ (or a sampled approximation to $\mathcal{F}_3$) can be viewed as a two-stage stochastic integer program in which the integer interdiction variables appear only in the first stage. More specifically, if the interdictions $\hat{x}$ are fixed, then the formulation $\mathcal{F}_3$ decomposes into $|S|$ *independent* linear programs, one for each scenario:

$$
\begin{aligned}
(LP_s(\hat{x})) \quad f_s(\hat{x}) = \min & \sum_{(i,j)\in A} u_{ij}\xi_{ijs}z_{ij} - \sum_{(i,j)\in A} u_{ij}\rho_{ij} \\
\text{subject to} \quad z_{ij} &\le \hat{x}_{ij} && \forall (i,j) \in A, \\
z_{ij} - \rho_{ij} &\le 0 && \forall (i,j) \in A, \\
\rho_{ij} - z_{ij} &\le 1 - \hat{x}_{ij} && \forall (i,j) \in A, \\
Ny &= 0, \\
y_{ij} &\le u_{ij}(1 - \xi_{ijs}\hat{x}_{ij}) && \forall (i,j) \in A, \\
\pi_r - \pi_t &\ge 1, \\
\rho_{ij} - \pi_i + \pi_j &\ge 0 && \forall (i,j) \in A, \\
y, \rho, z &\ge 0.
\end{aligned}
$$

Changing $x$ changes the right-hand side of the linear program $LP_s(x)$. It follows from linear programming duality theory and elementary convex analysis that $F(x) = \sum_{s\in S} p_s f_s(x)$ is a piecewise-linear convex function of $x$. The L-Shaped method of Van Slyke and Wets [30] is a method for solving stochastic programs that exploits the decomposable structure of the original formulation and the convexity of $F(x)$. However, the L-Shaped method is designed to solve problems in which the first-stage decision variables $x$ are real-valued, i.e. not constrained to be integers.

We first discuss the variant of the L-Shaped method that we employ, then we describe how this method is modified to solve stochastic programs that have integer variables in the first stage. For purposes of this discussion, it is important to note that the stochastic program $\mathcal{F}_3$ has *relatively complete recourse*; that is, for any value of the first-stage variables $x \in X$, there exists a feasible solution to the second stage problem $LP_s(x)$ for every scenario $s$. For ease of notation, we describe the algorithm in terms of minimizing the true expected maximum flow $F(x)$, whereas in reality, we are often minimizing some sample average approximation to $F(x)$.

The version of the L-Shaped method we employ is a multi-cut version that proceeds by finding subgradients of partial sums of the terms that make up $F(x)$. Suppose that the scenarios $S$ are partitioned into $C$ clusters denoted by $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_C$. Let $F_{[c]}$ represent the partial sum of $F(x)$ corresponding to the cluster $\mathcal{N}_c$; that is,

$$F_{[c]}(x) = \sum_{i\in\mathcal{N}_c} p_i f_i(x). \tag{19}$$

The algorithm maintains a model function $m^k_{[c]}$ which is a piecewise linear lower bound on $F_{[c]}$ for each cluster $\mathcal{N}_c$. We define this function at iteration $k$ of the algorithm by

$$m^k_{[c]}(x) = \inf\{\theta_c \,|\, \theta_c e \ge O^k_{[c]}x + o^k_{[c]}\}, \tag{20}$$

where $e = (1, 1, \ldots, 1)^T$ and $O^k_{[c]}$ is a matrix whose rows are subgradients of $F_{[c]}$ at previous iterates of the algorithm. The constraints in (20) are called *optimality cuts*. Optimality cuts are obtained as a byproduct of solving the linear programs $LP_s(x)$. Solving the $|S|$ linear programs $LP_s(x)$ is necessary simply to evaluate the function $F(x)$, so obtaining the optimality cuts adds very little computational burden to the algorithm.

Iterates of the multi-cut L-shaped method are obtained by minimizing an aggregate model function

$$m_k(x) \stackrel{\text{def}}{=} \sum_{c=1}^{C} m^k_{[c]}(x)$$

over the linear relaxation of the feasible region:

$$LP(X) \stackrel{\text{def}}{=} \{[0,1]^{|A|} \mid \sum_{(i,j)\in A} h_{ij}x_{ij} \leq K\}.$$

The iterate-finding master problem takes the form

$$\min_{x \in LP(X)} m_k(x). \tag{21}$$

By substituting from (20), the problem (21) is transformed to a *Master Linear Program* ($MLP_k$):

$$\begin{aligned}
(MLP_k) \qquad & \min \sum_{c=1}^{C} \theta_c \\
\text{subject to } & \sum_{(i,j)\in A} h_{ij}x_{ij} \leq K, \\
& \theta_c e \geq O^k_{[c]}x + o^k_{[c]} \qquad \forall c \in \{1, 2, \ldots, C\}, && (22) \\
& \theta_c \geq L \qquad\qquad\quad \forall c \in \{1, 2, \ldots, C\}, && (23) \\
& x_{ij} \geq 0 \qquad\qquad\quad \forall (i,j) \in A.
\end{aligned}$$

An artificial lower bound $L$ on the value of the $\theta_c$ is required by inequalities (23) to ensure that the minimum in (20) is achieved.

Algorithm 1 is a description of the L-Shaped method. The algorithm solves the linear programming relaxation of the formulation $\mathcal{F}_3$, with two small enhancements. First, the iterates of the algorithm that are integer-valued are recorded. Second, the optimal dual multipliers from the final master linear program are stored.

Choosing $x_0 \in X$ in step (1) is trivial due to the simple form of the constraints defining $X$. The evaluation of $F(x^k)$ in step (3) is performed in parallel. That is, different scenarios are assigned to different computational processors, and the processors simultaneously solve $LP_s(x^k)$ for the scenarios assigned to them. More discussion of the specific parallel platform is given in Section 4.2. The value $u_{IP}$, the corresponding integer solution $x_{IP}$, the final optimal dual multipliers $\lambda^*$, and the master linear program $MLP_{k^*}$ (for the last iteration $k^*$) are all retained and used later in the solution process. The basic L-Shaped method of Algorithm 1 is further enhanced with a trust-region mechanism for regularizing the steps taken by iterations of the algorithm and with the ability to consider multiple iterates in an asynchronous fashion, which is indispensable for our parallel implementation. The algorithm

**Algorithm 1** The L-Shaped Method

Choose $x_0 \in X$. Set $k = 0, u_{IP} = \infty, u_{LP} = \infty$
**repeat**
  Evaluate $F(x^k)$. Collect and create elements $O^k_{[c]}, o^k_{[c]}$ for updating model functions $m^k_{[c]}$.
  $u_{LP} = \min\{u_{LP}, F(x^k)\}$
  **if** $x^k \in X$ and $F(x^k) < u_{IP}$ **then**
    $u_{IP} = F(x^k)$
    $x_{IP} = x^k$
  **end if**
  Solve $MLP_k$, obtaining solution $x^{k+1} \in LP(X)$ with optimal value $\ell = m_k(x^{k+1})$.
  $k \leftarrow k + 1$
**until** $u_{LP} - \ell \leq \epsilon_1(1 + |u_{LP}|)$
Let $\lambda^*$ be the (final) optimal dual multipliers for optimality cuts (22).

is instrumented to run on a computational grid using the Master-Worker (MW) runtime library [10]. A complete description of the algorithm is out of the scope of this paper. For the full details of our implementation, please refer to the paper of Linderoth and Wright [19].

Algorithm 1 solves the linear programming relaxation of the MILP formulation ($\mathcal{F}_3$) of SNIP. If it turns out that the solution to the LP relaxation $x^*_{LP} \in X$, then the problem is solved. Otherwise, a mechanism to enforce integrality is required. A well-known method for enforcing integrality is branch-and-bound. Applied here, we could simply branch on a fractional component in $x^*_{LP}$ and repeat Algorithm 1 with the additional necessary bound constraints on the LP relaxation added to the master linear program $MLP_k$. Laporte and Louveaux [16] suggest a different approach in which the recourse function is evaluated (and the model function $m(x)$ is improved) only at points $\hat{x}$ in which a branch-and-bound procedure has $\hat{x} \in \{0, 1\}^{|A|}$.

The approach in this work is a combination of these two approaches. Initial computational experiments detailed in Section 3.2.1 indicated that the solution of the linear programming relaxation of $\mathcal{F}_3$ was often nearly integer-valued. This fact might also imply that the model function $m_{k^*}(x)$ that was created during the solution of the linear programming relaxation by Algorithm 1 would be a good starting approximation to the function $F(x)$ evaluated at feasible (integer-valued) points. Therefore, the solution approach suggested here solves the linear programming relaxation of $\mathcal{F}_3$ to optimality, retains (tight) optimality cuts (22) from the final master linear program $MLP_{k^*}$, then resorts to the method of Laporte and Louveaux, using a known upper bound to speed the branch-and-bound process at each iteration.

The algorithm solves a series of *integer-restricted* versions of the master linear program $MLP_k$. That is, at iteration $k$ of the algorithm, the method will solve the integer program $MIP_k$:

$$
\begin{aligned}
(MIP_k) \qquad & \min \sum_{c=1}^{C} \theta_c \\
\text{subject to } & \sum_{(i,j)\in A} h_{ij} x_{ij} \leq K, \\
& \theta_c e \geq O^k_{[c]} x + o^k_{[c]} \qquad \forall c \in \{1, 2, \ldots, C\}, \\
& \sum_{c=1}^{C} \theta_j \leq u_{IP}, \qquad (24) \\
& x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A.
\end{aligned}
$$

The upper bound on the solution value that the master problem seeks is imposed by constraint (24). Algorithm 2 is a description of the method employed to find integer-valued solutions to SNIP. An advantage of using optimality cuts from $MLP_{k^*}$ for the solution of $MIP_k$ was that we did not need to stabilize the iterates of Algorithm 2 with a trust region, as suggested by Santoso *et al.* [27].

**Algorithm 2** Adaptation of L-Shaped Method for Integer Variables

**Require:** $MLP_k$, $u_{IP}$, $x_{IP}$, $\lambda^*$ from solution to linear programming relaxation (Algorithm 1).

1: $k = 0$
2: $MIP_0 = MLP_k$
3: Remove all optimality cuts $i$ from $MIP_0$ such that $\lambda^*_i > 0$
4: **repeat**
5: Solve $MIP_k$, obtaining solution $x^{k+1} \in X$ with value $\ell = m_k(x^{k+1})$. If no solution exists, exit. $x_{IP}$ is the optimal solution.
6: Evaluate $F(x^{k+1})$. Collect and create elements $O^{k+1}_{[c]}, o^{k+1}_{[c]}$ for updating model functions $m^{k+1}_{[c]}$
7: **if** $F(x^{k+1}) < u_{IP}$ **then**
8: $u_{IP} = F(x^{k+1})$
9: $x_{IP} = x^{k+1}$
10: **end if**
11: $k \leftarrow k + 1$
12: **until** $u_{IP} - \ell \leq \epsilon_2(1 + |u_{IP}|)$

# 4 Computational Experiments

This section contains a description of the empirical behavior of the algorithm developed in Section 3.2 on the formulation $\mathcal{F}_3$ for the stochastic network interdiction problem derived in Section 2.

## 4.1 Test Problems

The test set consists of randomly generated problem instances that have the same structure as the instances of Cormican, Morton, and Wood [4]. The instances are on a network grid with a specified

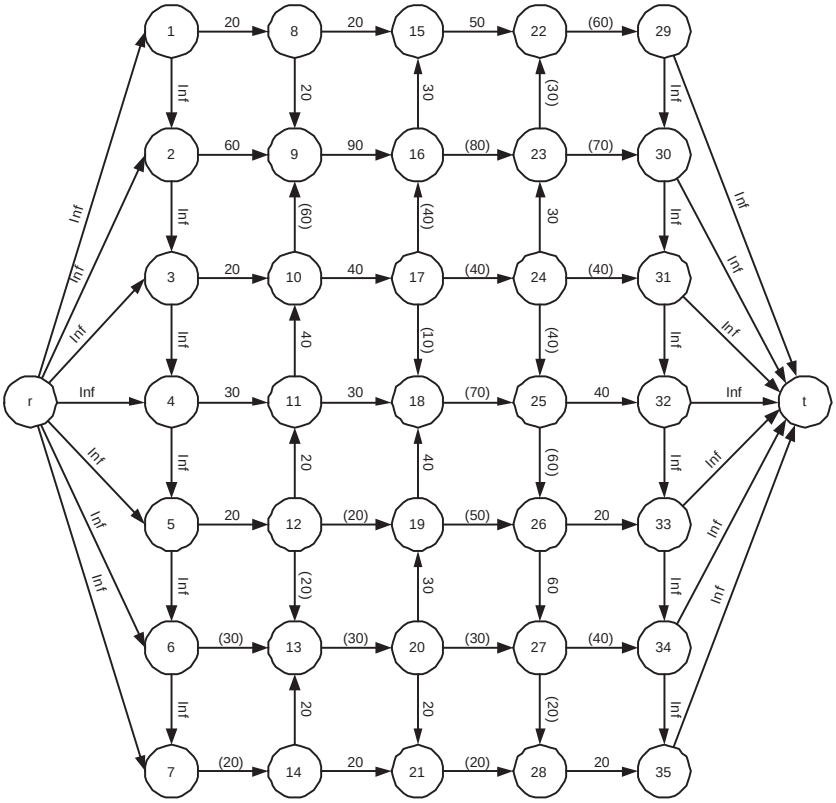


Figure 1: Instance SNIP7x5

number of rows and columns. Our naming convention is SNIP$i$x$j$, where $i$ is the number of rows in the grid and $j$ is the number of columns in the grid. The source node is connected to all nodes in the first column, and the sink node is connected to all nodes in the last column. All horizontal arcs are oriented forwards, whereas the direction of the vertical arcs can be either up or down, and this direction is chosen randomly with probability 1/2. The arcs connected to the source and sink nodes and the arcs in the first and last columns are un-interdictable and have unlimited capacities. A random subset of roughly 35% of the arcs are chosen to be interdictable. The capacitated arcs have capacities ranging uniformly between 10 and 100 at multiples of 10. The interdiction cost of each arc is $h_{ij} = 1$, and the budget $K$ will be varied in the instances. The probability of a successful interdiction was set to 75% on each interdictable arc, and these random variables are all pairwise-independent. Further characteristics of the test instances are summarized in Table 2. The instances SNIP7x5 and SNIP4x9 are taken directly from [4], and the instance SNIP7x5 is depicted in Figure 1. In Figure 1, the arcs upon which interdiction may occur are indicated by parentheses around the capacity value. The small instance SNIP4x4 was used only for debugging purposes and for the initial computational tests summarized in Table 1. It will not be included in the additional computational tests.

The models were built using the XPRESS-SP module in Dash Optimization's Mosel environment. A recently added feature of XPRESS-SP allows the user to write out instances in SMPS format. These SMPS files were used as input to the decomposition-based solver described in Section 3.2. The instances are available from the web site `http://coral.ie.lehigh.edu/sp-instances`. To read the stochastic

Table 2: Characteristics of Test Instances

| Instance | Budget | Number of Nodes | Number of Arcs | Number of Interdictable Arcs | Number of Scenarios |
|---|---|---|---|---|---|
| SNIP4x4 | 4 | 18 | 32 | 9 | 512 |
| SNIP7x5 | 6 | 37 | 72 | 22 | $4.2 \times 10^6$ |
| SNIP4x9 | 6 | 38 | 67 | 24 | $1.7 \times 10^7$ |
| SNIP10x10 | 10 | 102 | 200 | 65 | $3.7 \times 10^{19}$ |
| SNIP20x20 | 20 | 402 | 800 | 253 | $1.4 \times 10^{76}$ |

program specified in the SMPS file and to sample and manipulate the instance, the SUTIL stochastic programming utility library was used. SUTIL is available under an open-source license from the web site `http://coral.ie.lehigh.edu/sutil`. To solve the master linear programs $MLP_k$ and master integer programs $MIP_k$ required by Algorithms 1 and 2, the commercial optimization package CPLEX (v9.1) was employed. To solve the linear programs $LP_s(\hat{x})$ required to evaluate the objective function, the open-source linear programming solver `Clp` from COIN-OR (`www.coin-or.org`) was used. The experiments were done with a *full multi-cut* version of the algorithm, in which there is one model function $m_{[\cdot]}(x)$ for every scenario. The initial linear programming relaxation was solved to an accuracy of $\epsilon_1 = 10^{-6}$ by Algorithm 1, and the integer programs were solved to an accuracy of $\epsilon_2 = 10^{-2}$ by Algorithm 2.

## 4.2 Computational Platform

As mentioned in Section 3.3, the evaluation of the objective function $F(x)$ can be done efficiently using parallel processors. The parallel computational platform employed here is known as a *computational grid* [6]. Computational grids can be quite different than traditional parallel computing platforms. One fundamental difference is that the processors making up a computational grid are often heterogeneous and non-dedicated. That is, a processor participating in a grid computation may be reclaimed at any time by the owner of the processor or by a central scheduling mechanism. Our computational grid was built using the Condor software toolkit [21], which is available form the Condor web site `http://www.cs.wisc.edu/condor`. The default behavior of Condor is to vacate running jobs on a processor if the machine is being used interactively or if a job with a higher priority requests the processor. Thus, the computing environment is quite dynamic, and algorithms running in this environment must be able to cope with faults. Algorithms 1 and 2 were implemented to run on this dynamic computational platform using the run-time library MW [10]. MW is a C++ class library that can help users write master-worker type parallel algorithms to run on computational grids. With MW, the user (re)-implements abstract classes to specify the operations of the master and the worker and to define the computational tasks. MW then interacts with the lower-level grid scheduling software, such as Condor, to find available resources on which to run the computational tasks, and MW ensures that all tasks are run, even if processors leave the computation. Source code for MW is available from the web site `http://www.cs.wisc.edu/condor/mw`. For a longer discussion on computational grids and their use for stochastic programming, the reader is directed to the paper of Linderoth and Wright [20].

The majority of our experiments were run on the idle cycles harvested from roughly 40 processors

in the Center for Optimization Research @ Lehigh (COR@L). The processors in the COR@L Condor pool are all Intel-based processors running the Linux Operating System and ranging in clock speed from 400MHz to 2.4GHz. Since processors on this computational grid are heterogeneous and dynamic, statistics relating to computational time are difficult to interpret. For instance, the time required to solve the same instance can vary by an order of magnitude based on the availability of processors in the Condor pool. For our results, we will present the average wall clock time required to solve the instance.

### 4.3 Effectiveness of Method

The first experiment was aimed at determining the effectiveness of our solution approach to SNIP. Can an accurate solution to the true instance be obtained via sampled approximations of the new formulation $\mathcal{F}_3$? How many samples are required to obtain an accurate solution? Are the computing times with the parallel decomposition approach reasonable?

To answer these questions, we estimated the optimal value of each large instance in Table 2 using sampled approximations of various sizes ranging from $|T| = N = 50$ to $|T| = N = 5000$. For each sample size $N$, $M = 10$ replications were solved to obtain a 95% confidence interval on a lower bound on the optimal solution value via Equation (17). To compute a confidence interval on an upper bound on the optimal solution value, $F(x_j^*)$ was estimated with a sample size of $|U_j| = 100,000$ for each solution $x_j^*$ to the sampled instance, and equation (18) was employed. Monte Carlo (uniform random) sampling was used to generate the samples in all instances.

Table 3 shows how the lower and upper bounds on optimal solution value vary as a function of the sample size for each instance. The average wall clock time time required to achieve the solution $x_j^*$ to the sample average approximation is also listed for each instance and sample size, as are the average number of iterations required to solve the initial linear programming relaxation by Algorithm 1 and the average number of integer programs necessary to solve the instance via Algorithm 2. Estimating $F(x_j^*)$, by solving 100,000 linear programs, took roughly 10 minutes for the instance SNIP7x5, 15 minutes for SNIP4x9, 22 minutes for SNIP10x10, and 62 minutes for SNIP20x20 on our computational platform.

The solution time for most instances is quite moderate and shows a relatively low dependence on the number of scenarios $N$. The low dependence on $N$ is primarily accounted for by the fact that the number of LP iterations and IP solves remains nearly constant regardless of $N$. Instances with larger values of $N$ require more computational effort, but they can also be parallelized to a higher degree, as there are more scenario linear programs to solve at each iteration. Cormican, Morton, and Wood report solving an instance of the size of SNIP10x10 to 1% accuracy in 23,970 seconds on an RS/6000 Model 590 [4]. It is not our intention to directly compare our computational results with theirs. Indeed, any comparison is inherently skewed, since the sequential approximation algorithm in [4] produces *guaranteed* bounds on the solution, while the bounds here are statistical and require the repeated solution of sampled instances of the problem. Nevertheless, a fair conclusion to draw from this experiment is that a sample average approximation with a moderate number of scenarios ($< 5000$) is likely to produce a solution that is close to the true optimal $z_{\text{SNIP}}$, and for reasonably-sized instances, the time required to achieve this solution is relatively small. For larger instances (like SNIP 20x20), the CPU time becomes much larger. We will discuss mechanisms for improving the time to achieve an accurate solution in Sections 4.5 and 4.6.

| Instance | $N$ | LB | UB | Avg. Sol Time (sec.) | Avg # LP Iterations | Avg # IP Solves |
|---|---|---|---|---|---|---|
| SNIP7x5 | 50 | $76.42 \pm 4.24$ | $80.87 \pm 0.91$ | 227.39 | 12.10 | 1.00 |
| SNIP7x5 | 100 | $77.31 \pm 3.14$ | $80.87 \pm 0.79$ | 136.83 | 12.80 | 1.40 |
| SNIP7x5 | 200 | $78.51 \pm 3.12$ | $80.85 \pm 0.75$ | 118.96 | 12.30 | 0.00 |
| SNIP7x5 | 500 | $78.62 \pm 1.37$ | $80.50 \pm 0.52$ | 118.92 | 12.50 | 0.00 |
| SNIP7x5 | 1000 | $80.10 \pm 0.70$ | $79.99 \pm 0.21$ | 149.78 | 13.20 | 0.00 |
| SNIP7x5 | 2000 | $79.50 \pm 1.09$ | $80.42 \pm 0.53$ | 218.65 | 12.10 | 0.70 |
| SNIP7x5 | 5000 | $80.34 \pm 0.45$ | $80.14 \pm 0.31$ | 630.75 | 10.10 | 0.00 |
| SNIP4x9 | 50 | $9.96 \pm 1.02$ | $11.16 \pm 0.26$ | 328.22 | 9.20 | 2.00 |
| SNIP4x9 | 100 | $9.47 \pm 0.81$ | $10.95 \pm 0.12$ | 273.13 | 9.90 | 2.00 |
| SNIP4x9 | 200 | $10.92 \pm 0.55$ | $11.02 \pm 0.17$ | 129.03 | 10.40 | 4.90 |
| SNIP4x9 | 500 | $10.45 \pm 0.38$ | $10.94 \pm 0.05$ | 580.33 | 9.60 | 6.70 |
| SNIP4x9 | 1000 | $10.70 \pm 0.24$ | $10.91 \pm 0.03$ | 567.63 | 10.00 | 11.50 |
| SNIP4x9 | 2000 | $10.55 \pm 0.18$ | $10.90 \pm 0.05$ | 787.59 | 10.10 | 11.60 |
| SNIP4x9 | 5000 | $10.89 \pm 0.15$ | $10.82 \pm 0.16$ | 501.64 | 10.10 | 5.30 |
| SNIP10x10 | 50 | $88.28 \pm 1.86$ | $88.75 \pm 0.34$ | 133.44 | 26.00 | 3.20 |
| SNIP10x10 | 100 | $88.09 \pm 2.92$ | $88.52 \pm 0.38$ | 140.34 | 20.90 | 2.60 |
| SNIP10x10 | 200 | $87.08 \pm 1.83$ | $88.55 \pm 0.33$ | 141.34 | 17.70 | 1.60 |
| SNIP10x10 | 500 | $88.39 \pm 0.85$ | $88.20 \pm 0.21$ | 210.54 | 17.40 | 2.20 |
| SNIP10x10 | 1000 | $87.90 \pm 0.40$ | $88.15 \pm 0.12$ | 362.42 | 17.80 | 1.80 |
| SNIP10x10 | 2000 | $88.03 \pm 0.41$ | $88.09 \pm 0.21$ | 960.40 | 16.40 | 0.60 |
| SNIP20x20 | 50 | $173.82 \pm 3.88$ | $192.03 \pm 7.41$ | 1098.74 | 86.50 | 2.30 |
| SNIP20x20 | 100 | $180.36 \pm 4.09$ | $188.51 \pm 3.25$ | 2640.75 | 71.60 | 3.00 |
| SNIP20x20 | 200 | $179.39 \pm 2.50$ | $186.31 \pm 5.98$ | 3415.14 | 68.38 | 3.12 |
| SNIP20x20 | 500 | $182.38 \pm 1.72$ | $184.67 \pm 3.28$ | 21253.54 | 39.00 | 3.60 |

Table 3: Statistical Bounds on Optimal Solution Value and Required Computational Effort

## 4.4 Effect of Budget Size

Cormican, Morton, and Wood noted that the approximation method they employed started to take significant CPU time as the budget $K$ was increased for instances of the size of SNIP10x10 [4]. The second experiment was aimed at determining the effect of the budget on the difficulty of the SNIP instance. To test this, the instance SNIP10x10 with budgets of size $K = 10, 15,$ and $20$ was solved. We solved each instance approximately using the sample-average decomposition-based approach on the formulation $\mathcal{F}_3$ with $M = 10$ trials. Table 4 shows the convergence of lower bound and upper bounds on the optimal solution value, the average CPU time required to solve the sampled formulation, and the average number of LP iterations and IP solves necessary to solve the instance for different budgets and sample sizes.

| $K$ | $N$ | LB | UB | Avg. Sol Time (sec.) | Avg # LP Iterations | Avg # IP Solves |
|---|---|---|---|---|---|---|
| 10 | 50 | $88.28 \pm 1.86$ | $88.75 \pm 0.34$ | 133.44 | 26.00 | 3.20 |
| 10 | 100 | $88.09 \pm 2.92$ | $88.52 \pm 0.38$ | 140.34 | 20.90 | 2.60 |
| 10 | 200 | $87.08 \pm 1.83$ | $88.55 \pm 0.33$ | 141.34 | 17.70 | 1.60 |
| 10 | 500 | $88.39 \pm 0.85$ | $88.20 \pm 0.21$ | 210.54 | 17.40 | 2.20 |
| 10 | 1000 | $87.90 \pm 0.40$ | $88.15 \pm 0.12$ | 362.42 | 17.80 | 1.80 |
| 10 | 2000 | $88.03 \pm 0.41$ | $88.09 \pm 0.21$ | 960.40 | 16.40 | 0.60 |
| 15 | 50 | $76.46 \pm 2.39$ | $78.97 \pm 1.10$ | 120.63 | 33.00 | 3.60 |
| 15 | 100 | $75.98 \pm 1.66$ | $78.26 \pm 0.20$ | 192.32 | 26.20 | 4.00 |
| 15 | 200 | $76.88 \pm 1.03$ | $78.27 \pm 0.22$ | 219.90 | 22.70 | 3.90 |
| 15 | 500 | $77.09 \pm 0.90$ | $78.29 \pm 0.16$ | 658.39 | 21.70 | 4.30 |
| 15 | 1000 | $77.66 \pm 0.30$ | $78.21 \pm 0.27$ | 3023.90 | 22.90 | 5.00 |
| 15 | 2000 | $77.75 \pm 0.42$ | $78.26 \pm 0.14$ | 23587.25 | 20.40 | 6.90 |
| 20 | 50 | $71.34 \pm 1.74$ | $71.06 \pm 0.49$ | 84.50 | 33.90 | 2.40 |
| 20 | 100 | $70.51 \pm 2.67$ | $70.60 \pm 0.30$ | 102.87 | 30.30 | 3.20 |
| 20 | 200 | $70.04 \pm 1.12$ | $70.42 \pm 0.29$ | 129.07 | 25.20 | 3.10 |
| 20 | 500 | $69.98 \pm 0.75$ | $70.20 \pm 0.19$ | 222.20 | 23.50 | 3.40 |
| 20 | 1000 | $69.75 \pm 0.45$ | $70.16 \pm 0.23$ | 445.44 | 25.10 | 3.40 |
| 20 | 2000 | $70.26 \pm 0.26$ | $70.18 \pm 0.11$ | 2044.28 | 22.30 | 3.60 |

Table 4: Effect of Budget on Convergence of Solution Bounds for SNIP10X10 Instance

Instances with a budget size of $K = 15$ are significantly more difficult to solve than the $K = 10$ or $K = 20$ instances. The difficulty arises on two fronts. First, the bias in the sampled approximation is worse. For a sample of size $N = 2000$, there is still a gap between the statistical lower and upper bounds on the optimal objective function value $z_{\text{SNIP}}$ for $K = 15$, but for $K = 10$ and $K = 20$, this gap is negligible. Second, the average CPU time required to solve the $K = 15$ instances is *significantly* larger than for the $K = 10$ or $K = 20$ instances. The increase in CPU time is nearly entirely due to the increased number of IPs that need to be solved for the $K = 15$ instances. Often, especially at the later iterations of the algorithm, an integer program can take hours to solve. This is highly undesirable, and we will discuss heuristic mechanisms addressing this liability in Section 4.6. An interesting (and promising) point to be made about this experiment is that even though the $K = 15$ instances with

samples of size $N = 1000, 2000$ took a long time to solve, instances with much smaller sample sizes were producing high-quality solutions, as evidenced by the fact that the statistical upper bound remains nearly constant as $N$ increases.

## 4.5 Variance Reduction

The third experiment was designed to demonstrate the effect of selecting the elements within the sample $T$ using Latin Hypercube Sampling (LHS), a well-known variance reduction technique [23]. To generate a Latin Hypercube sampled approximation to SNIP, for each interdictable arc, a distribution consisting of $pN$ "success" events and $(1-p)N$ "failure" events is built. The scenarios are created by sampling from these distributions *without replacement*. Thus, in a Latin Hypercube Sample with $N = 100$, if the probability of a successful interdiction is $p = 75\%$, exactly 75 of the scenarios will have a successful interdiction for each arc. It is insightful to contrast this sample with one drawn from Monte Carlo (uniform random) sampling, which can be viewed as sampling *with replacement*. In a Monte Carlo sample, there are not necessarily 75 successful interdictions on each arc; rather, the number of successful interdictions is a random variable with expected value $pN = 75$. Elements within a sample are not independent in LHS; it is precisely this negative covariance between the elements that can lead to variance reduction. Sampling "more intelligently" via variance reduction techniques has shown considerable promise in obtaining more accurate solutions to stochastic programs [18, 7, 15]. The SUTIL stochastic programming utility we employ can automatically generate Latin Hypercube samples of the true instance specified in SMPS format.

Tables 5 and 6 detail the computational results obtained by repeating the experiments from Sections 4.3 and 4.4, only using Latin Hypercube sampling in all cases. The results show that the bias is reduced by creating the sample average approximation with LHS. Namely, for a fixed instance and fixed value of $N$, the statistical lower bound obtained in the Latin Hypercube case is nearly always larger than the lower bound when computed using Monte Carlo sampled approximations. The reader will also note that in nearly all cases, the variance (and hence confidence interval) of the estimates is smaller when using LHS as opposed to Monte Carlo sampling. This experiment provides even more computational evidence that advanced sampling techniques should be employed when creating sample average problems.

## 4.6 Heuristic Algorithms

For large and difficult instances, such as SNIP20x20 or the SNIP10x10 instance with a budget of $K = 15$, the computational times of the exact algorithm for solving the sample average approximation to $\mathcal{F}_3$ are prohibitively long. In this section, we explore two simple heuristic ideas for obtaining solutions to SNIP. Both ideas are based on the observation that often nearly all of the components of the solution to the linear programming relaxation $x^*_{LP} \in \arg\min_{x \in LP(X)} F(x)$ are integer-valued.

Since the instances in these experiments are more difficult, the experiments were run in a larger Condor pool. For these tests, a Condor pool consisting of three different Beowulf clusters at Lehigh University was used. In total, 260 processors were available for computation, but due to the nature of the Condor scheduling software, only a fraction were available to use at any one time.

| Instance | $N$ | LB | UB | Avg. Sol Time (sec.) | Avg # LP Iterations | Avg # IP Solves |
|---|---|---|---|---|---|---|
| SNIP7x5 | 50 | $79.82 \pm 1.08$ | $80.23 \pm 0.27$ | 204.20 | 13.50 | 1.10 |
| SNIP7x5 | 100 | $79.96 \pm 0.88$ | $80.31 \pm 0.43$ | 135.92 | 13.20 | 0.40 |
| SNIP7x5 | 200 | $80.32 \pm 0.32$ | $80.12 \pm 0.02$ | 324.21 | 13.40 | 0.30 |
| SNIP7x5 | 500 | $80.03 \pm 0.21$ | $80.11 \pm 0.01$ | 335.58 | 12.20 | 0.00 |
| SNIP7x5 | 1000 | $80.15 \pm 0.17$ | $80.12 \pm 0.02$ | 314.92 | 12.80 | 0.00 |
| SNIP7x5 | 2000 | $80.25 \pm 0.12$ | $80.11 \pm 0.02$ | 352.18 | 12.40 | 0.00 |
| SNIP4x9 | 50 | $10.00 \pm 0.67$ | $10.98 \pm 0.10$ | 190.33 | 9.30 | 0.00 |
| SNIP4x9 | 100 | $10.06 \pm 0.53$ | $10.94 \pm 0.02$ | 259.21 | 8.70 | 1.90 |
| SNIP4x9 | 200 | $10.19 \pm 0.36$ | $10.91 \pm 0.02$ | 198.41 | 9.30 | 2.50 |
| SNIP4x9 | 500 | $10.49 \pm 0.23$ | $10.91 \pm 0.03$ | 144.75 | 10.10 | 6.40 |
| SNIP4x9 | 1000 | $10.84 \pm 0.16$ | $10.91 \pm 0.02$ | 204.31 | 9.40 | 11.00 |
| SNIP4x9 | 2000 | $10.78 \pm 0.13$ | $10.92 \pm 0.02$ | 326.47 | 10.40 | 9.70 |
| SNIP10x10 | 50 | $88.62 \pm 0.60$ | $88.50 \pm 0.57$ | 201.92 | 18.80 | 1.40 |
| SNIP10x10 | 100 | $88.21 \pm 0.70$ | $88.41 \pm 0.32$ | 147.95 | 17.20 | 1.40 |
| SNIP10x10 | 200 | $88.16 \pm 0.31$ | $88.12 \pm 0.02$ | 436.03 | 16.40 | 1.40 |
| SNIP10x10 | 500 | $88.24 \pm 0.22$ | $88.13 \pm 0.01$ | 466.75 | 15.70 | 0.00 |
| SNIP10x10 | 1000 | $88.16 \pm 0.08$ | $88.13 \pm 0.02$ | 582.20 | 15.80 | 0.00 |
| SNIP10x10 | 2000 | $88.12 \pm 0.09$ | $88.13 \pm 0.02$ | 959.02 | 15.30 | 0.00 |

Table 5: Convergence of Solution Bounds with Latin Hypercube Sampling

#### 4.6.1 Rounding

Since the constraint set $X$ is simple, an LP solution $x^*_{LP}$ can be easily rounded to obtain a feasible solution $x_R \in X$. Further, $\|x_R - x^*_{LP}\|$ is typically small, so one might hope that $F(x_R) - F(x^*_{LP})$, which is an (expected) upper bound on $F(x_R) - z_{\text{SNIP}}$, the suboptimality of the solution $x_R$, is also small.

To test this hypothesis, the linear programming relaxation of an instance of SNIP20x20 with $N = 500$ scenarios was solved, resulting in an optimal solution $x^*_{LP}$ with objective function value $v^* = 181.63$. In the solution $x^*_{LP}$, 13 components had value of one, 14 were fractional, and the remaining components had value zero. As the budget for SNIP20x20 was $K = 20$, we simply took as the rounded integer solution $x_R$ the thirteen arcs whose value was one, and the seven fractional arcs whose value in $x^*_{LP}$ was largest. Using a new sample, the function $F(x_R)$ was estimated to be $195.14 \pm 0.03$. From the computational results presented in Table 3, it seems likely that the optimal solution value of SNIP20x20 is less than 184, so $x_R$ is not a good solution. From this, we conclude that by simply rounding (naively) the solution of the LP relaxation to $\mathcal{F}_3$, one cannot expect to get a good solution to SNIP.

#### 4.6.2 Variable Fixing

The second heuristic idea for reducing solution time is to restrict the master integer program $MIP_k$ by fixing the variables that take integral values in the solution to the linear programming relaxation. That

| $K$ | $N$ | LB | UB | Avg. Sol Time (sec.) | Avg # LP Iterations | Avg # IP Solves |
|---|---|---|---|---|---|---|
| 10 | 50 | $88.62 \pm 0.60$ | $88.50 \pm 0.57$ | 201.92 | 18.80 | 1.40 |
| 10 | 100 | $88.21 \pm 0.70$ | $88.41 \pm 0.32$ | 147.95 | 17.20 | 1.40 |
| 10 | 200 | $88.16 \pm 0.31$ | $88.12 \pm 0.02$ | 436.03 | 16.40 | 1.40 |
| 10 | 500 | $88.24 \pm 0.22$ | $88.13 \pm 0.01$ | 466.75 | 15.70 | 0.00 |
| 10 | 1000 | $88.16 \pm 0.08$ | $88.13 \pm 0.02$ | 582.20 | 15.80 | 0.00 |
| 10 | 2000 | $88.12 \pm 0.09$ | $88.13 \pm 0.02$ | 959.02 | 15.30 | 0.00 |
| 15 | 50 | $77.04 \pm 0.50$ | $78.31 \pm 0.25$ | 164.55 | 24.10 | 9.50 |
| 15 | 100 | $78.09 \pm 0.48$ | $78.20 \pm 0.08$ | 237.22 | 21.80 | 20.60 |
| 15 | 200 | $77.41 \pm 0.43$ | $78.18 \pm 0.06$ | 3998.76 | 20.10 | 21.10 |
| 15 | 500 | $77.95 \pm 0.31$ | $78.05 \pm 0.06$ | 24122.11 | 17.80 | 29.10 |
| 15 | 1000 | $77.98 \pm 0.17$ | $78.05 \pm 0.08$ | 5070.12 | 17.70 | 12.60 |
| 15 | 2000 | $78.07 \pm 0.10$ | $78.07 \pm 0.08$ | 15358.83 | 17.10 | 13.00 |
| 20 | 50 | $71.06 \pm 0.87$ | $71.11 \pm 0.38$ | 165.65 | 24.50 | 1.40 |
| 20 | 100 | $70.38 \pm 0.69$ | $70.68 \pm 0.48$ | 234.01 | 22.30 | 4.00 |
| 20 | 200 | $69.93 \pm 0.71$ | $70.39 \pm 0.19$ | 228.50 | 20.10 | 3.50 |
| 20 | 500 | $70.30 \pm 0.32$ | $70.38 \pm 0.21$ | 492.42 | 20.10 | 4.10 |
| 20 | 1000 | $70.05 \pm 0.34$ | $70.25 \pm 0.19$ | 630.94 | 19.50 | 2.90 |
| 20 | 2000 | $70.42 \pm 0.24$ | $70.36 \pm 0.20$ | 1237.80 | 19.90 | 4.70 |

Table 6: Convergence of Solution Bounds with Latin Hypercube Sampling for SNIP10X10

is, before beginning Algorithm 2, the constraints

$$x_{ij} = 0 \; \forall (i,j) \mid x^*_{ij} = 0, \qquad x_{ij} = 1 \; \forall (i,j) \mid x^*_{ij} = 1, \tag{25}$$

where $x^*$ is the solution of linear programming relaxation from Algorithm 1, are added to $MIP_0$.

The idea was first tested on an instance of SNIP10x10 with budget size $K = 15$, one of the most computationally challenging instances we encountered. Solving the linear programming relaxation of this instance took 68 iterations, and of the 200 variables in the problem, all but eight variables were fixed according to the criteria (25). Only one integer program, with 2 nodes in the branch-and-bound tree was necessary for Algorithm 2 to produce a solution once the variables were fixed. In total, finding this solution took 1851 seconds on an average of 98 processors in the Condor pool. The objective value of the solution was estimated using a new sample and found to be $78.03 \pm 0.03$. Comparing this result with the results in Tables 4 and 6 reveals that the solution obtained in this manner is of high quality.

As a final test, an instance of SNIP20x20 with $N = 1000$ scenarios was solved. This is the largest instance that we attempted to solve, and the sampled version of the formulation $\mathcal{F}_3$ for this problem has over 4.6 million rows and over 2.8 million columns. To solve this instance, we warm-started Algorithm 1 with an initial iterate $x_0$ taken as the solution to the $N = 500$ instance from Section 4.6.1. Solving the LP relaxation to this instance took 14 iterations, and 4 IP iterations were necessary for Algorithm 2 to terminate with a solution. The wall clock time required to find this solution was 3866 seconds on an average of 134 processors. The objective value of the best solution found by the heuristic was estimated with a new sample to be $182.60 \pm 0.04$. Examining our (preliminary) attempts and the statistical bounds

obtained in Table 4, we see that this solution is likely of high quality. Fixing variables suggested by the LP relaxation to $\mathcal{F}_3$ appears to be an effective mechanism for reducing solution times, while sacrificing little in the way of solution quality.

# 5 Conclusions and Future Directions

We have given a MILP formulation of a stochastic network interdiction problem aimed at minimizing an adversary's expected maximum flow. Since the integer variables in the formulation appear only in the first stage, a combination of decomposition, sampling, parallel computing, and heuristics allowed us to solve very accurately instances of larger sizes than have been previously reported in the literature.

Continuing work is aimed at increasing the size of instances that can be solved with this method. In particular, we will investigate more sophisticated mechanisms for switching between solving the master linear program $MLP_k$ and the master integer program $MIP_k$. Also, since the software is already instrumented to run in parallel on a computational grid, it is natural to consider parallelizing the solution of the integer master problem $MIP_k$. Grid-enabled solvers for mixed integer programming have already been developed in [1, 2]. We also plan to investigate the impact of adding cover inequalities from relaxed optimality cuts, of regularizing the integer master-problem, and of generating non-dominated optimality cuts. These ideas have been applied to a stochastic program arising from a supply chain planning application by Santoso *et al.* [27]. A long-term goal is to enhance the decomposition algorithm to become an effective approach for general two-stage stochastic integer programs in which the integrality restrictions are limited to the first-stage variables.

# 6 Acknowledgments

The authors would like to thank the development team of Condor, the developers and maintainers of the COIN-OR suite of open-source operations research software tools, and the developers at Dash Optimization for providing software without which this research could not have been accomplished. In particular, Nitin Verma from Dash Optimization was most helpful in answering our questions about SMPS file generation from within the XPRESS-SP environment. The authors thank Shabbir Ahmed for bring the reference [27] to their attention.

This work was supported in part by the US National Science Foundation (NSF) under grant CNS-0330607, and by the US Department of Energy under grant (DE-FG02-05ER25694). Computational resources at Lehigh are provided in part by equipment purchased by the NSF through the IGERT Grant DGE-9972780. Author Janjarassuk is supported by a grant for the Thai government.

# References

[1] Q. Chen and M. C. Ferris. FATCOP: A fault tolerant Condor-PVM mixed integer program solver. *SIAM Journal on Optimization*, 11:1019–1036, 2001.

[2] Q. Chen, M. C. Ferris, and J. T. Linderoth. FATCOP 2.0: Advanced features in an opportunistic mixed integer programming solver. *Annals of Operations Research*, 103:17–32, 2001.

[3] K. J. Cormican. Computational methods for deterministic and stochasic network interdiction problems. Master's thesis, Operations Research Department, Naval Postgraduate School, Monterey, CA, 1995.

[4] K. J. Cormican, D. P. Morton, and R. K. Wood. Stochastic network interdiction. *Operations Research*, 46:184–197, 1998.

[5] J. Dupačová and R. J.-B. Wets. Asymptotic behavior of statistical estimators and of optimal solutions of stochastic optimization problems. *The Annals of Statistics*, 16:1517–1549, 1988.

[6] I. Foster and C. Kesselman. Computational grids. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999. Chapter 2.

[7] M. Freimer, D. Thomas, and J. Linderoth. Reducing bias in stochastic linear programming with sampling methods. Technical Report 05T-002, Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, Pennsylvania, 2005.

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

[9] B. Golden. A problem in network interdiction. *Naval Research Logistics Quarterly*, 25:711–713, 1978.

[10] J.-P. Goux, S. Kulkarni, J. T. Linderoth, and M. Yoder. Master-Worker : An enabling framework for master-worker applications on the computational grid. *Cluster Computing*, 4:63–70, 2001.

[11] M. Grötschel, C. Monma, and M. Stoer. Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research*, 40: 309–330, 1992.

[12] H. Held, R. Hemmece, and D. Woodruff. A decomposition approach applied to planning the interdiction of stochastic networks. *Naval Research Logistics*, 52:321–328, 2005. `http://www.uni-duisburg.de/FB11/disma/heldnet4.pdf`.

[13] E. Israeli and R. K. Wood. Shortest-path network interdiction. *Networks*, 40:97–111, 2002.

[14] A. Kleywegt, A. Shapiro, and T. Homem-de-Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 2:479–502, 2001.

[15] M. Koivu. Variance reduction in sample approximations of stochastic programs. *Mathematical Programming*, pages 463–485, 2005.

[16] Gilbert Laporte and François V. Louveaux. The integer $L$-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 13:133–142, 1993.

[17] J. T. Linderoth, A. Shapiro, and S. J. Wright. The empirical behavior of sampling methods for stochastic programming. Technical Report Optimization Technical Report 02-01, Computer Sciences Department, University of Wisconsin-Madison, 2002.

[18] J. T. Linderoth, A. Shapiro, and S. J. Wright. The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research*, 142:219–245, 2006.

[19] J. T. Linderoth and S. J. Wright. Implementing a decomposition algorithm for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24:207–250, 2003. Special Issue on Stochastic Programming.

[20] J. T. Linderoth and S. J. Wright. Computational grids for stochastic programming. In S. Wallace and W. Ziemba, editors, *Applications of Stochastic Programming*, SIAM Mathematical Series on Optimization, pages 61–77. SIAM, 2005.

[21] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP*, 11, 1997.

[22] W. K. Mak, D. P. Morton, and R. K. Wood. Monte carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24:47–56, 1999.

[23] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21: 239–245, 1979.

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

[25] V. Norkin, G. Pflug, and A. Ruszczyński. A branch and bound method for stochastic global optimization. *Mathematical Programming*, 83:425–450, 1998.

[26] F. Pan, W. S. Charlton, and David P. Morton. A stochastic program for interdicting smuggled nuclear material. In D.L. Woodruff, editor, *Network Interdiction and Stochastic Integer Programming*, pages 1–20. Kluwer Academic Publishers, 2003.

[27] T. Santoso, S. Ahmed, M. Goetschalckx, and A. Shapiro. A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research*, 167: 96–115, 2005.

[28] A. Shapiro and T. Homem-de-Mello. On the rate of convergence of optimal solutions of Monte Carlo approximations of stochastic programs. *SIAM Journal on Optimization*, 11:70–86, 2000.

[29] M. Simaan and J. B. Cruz. On the Stackelberg strategy in nonzero-sum games. *Journal of Optimization Theory and Applications*, 11:533–555, 1973.

[30] R. Van Slyke and R.J-B. Wets. L-shaped linear programs with applications to control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17:638–663, 1969.

[31] B. Verweij, S. Ahmed, A. Kleywegt, G. Nemhauser, and A. Shapiro. The sample average approximation method applied to stochastic routing problems: A computational study. *Computational Optimization and Applications*, 24:289–333, 2003.

[32] A. Washburn and K. Wood. Two-person zero-sum games for network interdiction. *Operations Research*, 42:243–251, 1994.

[33] H. P. Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, 1999.

[34] R. D. Wollmer. Removing arcs from a network. *Journal of the Operational Research Society of America*, 12:934–940, 1964.