Decomposition Based Refinement for the Network Interdiction Problem

Krish Matta
School of Computer Science
Carnegie Mellon University
Pittsburgh, USA
kmatta@andrew.cmu.edu

Xiaoyuan Liu
Fujitsu Research of America
Sunnyvale, USA
xliu@fujitsu.com

Ilya Safro
Department of Computer and Information Sciences
University of Delaware
Newark, USA
isafro@udel.edu

Abstract—The shortest path network interdiction (SPNI) problem poses significant computational challenges due to its NP-hardness. Current solutions, primarily based on integer programming methods, are inefficient for large-scale instances. In this paper, we introduce a novel hybrid algorithm that can utilize Ising Processing Units (IPUs) alongside classical solvers. This approach decomposes the problem into manageable sub-problems, which are then offloaded to the slow but high-quality classical solvers or IPU. Results are subsequently recombined to form a global solution. Our method demonstrates comparable quality to existing whole problem solvers while reducing computational time for large-scale instances. Furthermore, our approach is amenable to parallelization, allowing for simultaneous processing of decomposed sub-problems.

Reproducibility: Our source code and experimental results are available at https://github.com/krishxmatta/network-interdiction.

Index Terms—decomposition, Ising processing hardware, network interdiction, refinement, integer programming

I. Introduction

Network interdiction refers to a class of challenging combinatorial optimization problems that involve strategically disrupting flows in a network to achieve specific objectives [30]. These problems can be characterized as a game between attackers and defenders of a network. The defender seeks to optimize a predefined objective across the network, such as maximizing flow between two nodes, while the attacker aims to impede the defender's objective by inflicting maximum disruption upon the network. This disruption may manifest in a variety of forms, such as targeted attacks on the network arcs aiming to destroy or impair them.

The topic of network interdiction has gained significant attention due to its ability to model complex situations in the defense domain. For example, Ghare et al. [9] demonstrated how network interdiction can be used in wartime to capacitate an enemy's supply network to maximally disrupt the flow of enemy troops. Network interdiction can additionally be used to identify weaknesses in critical infrastructure to make them more resilient to terrorist attacks and natural disasters. An example of such an application can be found in a study by Salmerón et al. [27], which details the use of network interdiction to model attacks on large-scale power grids. The study then utilizes this model to determine how best to minimize economic loss resulting from these attacks. Another study by Nandi et al. [23] details the usage of a network interdiction model on cyber attack graphs to pinpoint vulnerabilities in cyberinfrastructures, thereby helping prevent organizations from cyberattacks. Such problems are even extremely useful outside of military applications, e.g. in this line of work [2], [11], [19] the authors show how network interdiction can be used to minimize the generally defined infection spread and attacks.

Within this paper, we study the Shortest Path Network Interdiction (SPNI) problem. In this problem, the defender wishes to traverse the minimum length path between two specified nodes s and t in a directed network. The attacker uses their limited resources to destroy certain arcs, or increase their effective length, to increase the defender's shortest s-t path length. Each arc is either interdicted or not, and additionally, it is known beforehand how much an arc costs to interdict and how much interdiction of an arc causes its effective length to increase. When solving SPNI, we take the viewpoint of the attacker, and thus our goal is to maximize the shortest s-t path length.

Our contribution The SPNI is an $\mathcal{NP}$-Hard problem [16]. To the best of our knowledge, almost all current SPNI algorithms rely on integer programming methods. On large problem instances, these methods are time costly—for example, in one of the most important works regarding SPNI, the largest network solved by Israeli et al. had 240 nodes and 1,042 arcs [16], so these methods are either not scalable or produce large gaps or low quality results. To ameliorate this issue, we propose a novel algorithm which may leverage Ising processing units (IPUs) [5], specialized computational devices specifically tailored to solve the Ising model, as well as classical solvers that can exhibit high quality solutions for small instances in a reasonable computational time. Examples of IPUs include but are not limited to quantum annealers, gate-based quantum machines equipped with the Ising model solvers, and both digital and analog annealers [20], [22]. Since physical limitations of existing IPU hardware severely restrict the size of problems they can handle and connectivity between variables, we adopt a hybrid IPU-classical approach in which a classical computer decomposes SPNI into subproblems small enough for an IPU, offloads them for computation, then combines the IPU's results into a global solution. For proof of concept, we use exact solvers instead of a real IPU. Our

decomposition based results demonstrate an ability to achieve almost the same quality as the whole problem slow solvers that are prohibitive for large-scale instances and IPU hardware. Additionally, our approach is parallelization friendly, namely, sub-problems obtained as a result of SPNI decomposition can be tackled in parallel.

II. RELATED WORK

In one of the earliest works studying SPNI, Malik et al. focused on a variant termed the k most vital arcs problem in which the interdiction of each arc requires exactly one unit of the attacker's budget and interdiction of an arc results in its complete removal [21]. Ball et al. show that the k most vital arcs problem is $\mathcal{NP}$-Hard. Since this is a variant of SPNI, it follows that SPNI is $\mathcal{NP}$-Hard as well [3]. Corley and Shaw [6] investigated the single-most-vital-arc problem, which is the k most vital arcs problem where $k = 1$, but this problem is a simple variant of SPNI that lies in $\mathcal{P}$. Rather than study the full removal of arcs, Fulkerson and Harding [8] and Golden [12] study related problems in which the length of an arc increases as more budget is allocated to its interdiction. In one of the most prominent works studying SNPI, Israeli and Wood [16] study the problem in its more general form, proposing two integer-programming based algorithms of different quality depending on whether arcs are fully removed or if arc length is increased as a result of interdiction.

More recent work on SPNI has since shifted away from integer-programming algorithms. Huang et al., for example, approximate SPNI using a reinforcement learning framework [15]. However, this approach generates quite significant gaps and the reinforcement learning is not particularly scalable. Rocco et al. [25] study an extension of SPNI where the attacker is not solely focused on maximizing the s-t path length, but also wishes to minimize interdiction cost, and propose an evolutionary algorithm to do so. This is a multiobjective problem that is different than ours. For more details, we refer the reader to the recent comprehensive survey on the network interdiction problem [30]. To our knowledge, however, no attempts have been made for parallelization friendly decomposition of SPNI for IPU hardware.

Several frameworks have been proposed to solve large instances of combinatorial optimization problems on small IPU hardware. The quadratic unconstrained binary optimization (QUBO) formulation is a popular formulation for combinatorial optimization problems that is equivalent to the Ising model, and thus is solvable on all IPU hardware [10]. QUBO formulations contain only quadratic binary variables with no constraints. A large number of work in the quantum computing space focuses on formulating problems into one large QUBO, then identifying sub-QUBOs that can be solved directly on an IPU. The qbsolv tool developed by D-Wave accomplishes exactly this, once allowing researchers to solve a 1254 binary variable problem using quantum hardware that can only solve problems with at most 64 binary variables [24]. An alternative approach is, instead of first formulating the problem as a QUBO then decomposing the QUBO itself, to decompose the

problem and then formulate these subproblems into QUBOs. For example, Shaydulin et al. [29] take this approach to solve the community detection problem on quantum hardware. This is the approach we opt to take in this paper, applied to SPNI.

III. PROBLEM FORMULATION

Let $G = (N, A)$ be a directed network with node set N and arc set A. The length of arc $k \in A$ is $c_k \geq 0$, and the added interdiction length is $d_k \geq 0$, meaning that if arc k is interdicted, then its effective length becomes $c_k + d_k$. If interdiction destroys arc k, then d_k can be set to a sufficiently large value. For any node $i \in N$, let $A^+(i) \subseteq A$ denote the set of all arcs directed out of node i, and similarly let $A^-(i) \subseteq A$ denote the set of all arcs directed into node i. While other forms of SNPI allow for multiple resource types and variant interdiction costs, for our algorithmic specializations we model a single resource type and the scenario in which interdiction of any arc in A costs only one unit of this resource. Thus, let r_0 denote the amount of budget we have available. Let $\mathbf{x} = \{x_k\}_{k=1}^{|A|}$ where for each arc $k \in A$, x_k is a binary variable denoting whether arc k is interdicted or not by the attacker, and let $\mathbf{y} = \{y_k\}_{k=1}^{|A|}$ where y_k is a binary variable denoting whether arc k is traversed by the defender. Finally, let $s, t \in N$ denote the source and sink node in G, respectively. The integer programming formulation of SNPI is

$$
\begin{aligned}
\max_{\mathbf{x} \in X} \min_{\mathbf{y}} \quad & \sum_{k \in A} (c_k + x_k d_k) y_k \\
\text{s.t.} \quad & \sum_{k \in A^+(s)} y_k - \sum_{k \in A^-(s)} y_k = 1 \\
& \sum_{k \in A^+(t)} y_k - \sum_{k \in A^-(t)} y_k = -1 \\
& \sum_{k \in A^+(i)} y_k - \sum_{k \in A^-(i)} y_k = 0, \forall i \in N \setminus \{s, t\},
\end{aligned}
$$

where $X = \{\mathbf{x} \in \{0, 1\}^{|A|} \mid \sum_{i=1}^{|A|} \mathbf{x}_i \leq r_0\}$.

Note that the above formulation is biobjective, and thus can not be directly converted into a QUBO model. As Israeli et al. [16] show, we may fix $\mathbf{x}$, take the dual of the inner minimization, make some modifications, then release $\mathbf{x}$, and obtain a single objective formulation

$$
\begin{aligned}
\max_{\mathbf{x}, \pi} \quad & \pi_t \\
\text{s.t.} \quad & \pi_j - \pi_t - d_k x_k \leq c_k, \forall k = (i, j) \in A \\
& \pi_s = 0 \\
& \mathbf{x} \in X.
\end{aligned} \tag{1}
$$

We may interpret π_i as the post-interdiction shortest-path length from s to i. As such, we may impose bounds on $\pi_i \in [0, |N| \max_{k' \in A} \{c_{k'} + d_{k'}\}]$. Since each variable is constrained and we have a single objective, we may now convert each constraint to a penalty function and decompose bounded variables into several binary variables, for example using the mapping proposed by Karimi et al. [17]. Letting P be a sufficiently large positive penalty value, our QUBO formulation is

$$\max_{\mathbf{x},\pi,\mathbf{m},n} \quad \pi_t - P \sum_{k=(i,j)\in A} (c_k - (m_k + \pi_j - \pi_i - d_k x_k))^2$$

$$-P(\pi_s)^2 - P(r_0 - (n + \sum_{k\in A} x_k))^2$$

$$\text{s.t.} \quad \pi_i \in [0, |N| \max_{k'\in A}\{c_{k'} + d_{k'}\}], \forall i \in N$$

$$m_k \in [0, |N| \max_{k'\in A}\{c_{k'} + d_{k'}\} + d_k], \forall k \in A$$

$$n \in [0, r_0]$$

$$\mathbf{x} \in \{0,1\}^{|A|}.$$

$$(2)$$

Note that we have included bounded non-binary variables for simplicity. Additionally, we have included variables $\mathbf{m}$ and n to act as slack variables for inequality constraints. In the actual QUBO formulation, all bounded variables and slack variables should be mapped to several binary variables. The given QUBO formulation is directly solvable on an IPU.

We now describe how to formulate subproblems of SPNI for the purposes of our algorithm. We define a connected partition of a network $G = (N, A)$ as a subset of nodes N such that its induced subgraph is weakly connected. Say we have a connected partition of G, i.e., a subset of nodes $N_p \subseteq N$, a sink node $t' \in N_p$, and let A_p represent all arcs associated with this subset of nodes, i.e. $A_p = \{(u,v) \in A | v \in N_p\}$. Note that we may partition $A_p = A_{p,1} \cup A_{p,2}$ where $A_{p,1} = \{(u,v) \in A | u \in N_p \wedge v \in N_p\}$ and $A_{p,2} = \{(u,v) \in A | u \notin N_p \wedge v \in N_p\}$. We define a subproblem of SPNI over this partition N_p and sink node t' by restricting our free variables to only those associated with N_p—rather than define π_i for all $i \in N$, we will only define π_i for all $i \in N_p$, and similarly only define x_k for all $k \in A_p$. Additionally, we change the objective formula to maximize $\pi_{t'}$ rather than π_t.

For those variables not associated with N_p, we may fix their values to constants. Assume that we have a current solution $A' \subseteq A$—by "solution," we refer to a subset of arcs chosen for interdiction. We use Γ_i to denote a constant representing the shortest-path length from node s to i after all arcs in A' have been interdicted. Since π_i represents the postinterdiction shortest-path length from nodes s to i, we can fix π_i to the value Γ_i for all nodes i not in N_p. Additionally, we limit our budget from r_0 to $r_0 - |A' \setminus A_p|$, ensuring we use only the budget we have allocated within N_p. The subproblem formulation over N_p with sink node t' is thus

$$\max_{\mathbf{x},\pi,\mathbf{m},n} \quad \pi_{t'} - P \sum_{k=(i,j)\in A_{p,1}} (c_k - (m_k + \pi_j - \pi_i - d_k x_k))^2$$

$$-P \sum_{k=(i,j)\in A_{p,2}} (c_k - (m_k + \pi_j - \Gamma_i - d_k x_k))^2$$

$$-P(\pi_s)^2$$

$$-P((r_0 - |A' \setminus A_p|) - (n + \sum_{k\in A_p} x_k))^2$$

$$\text{s.t.} \quad \pi_i \in [0, |N| \max_{k'\in A}\{c_{k'} + d_{k'}\}], \forall i \in N_p$$

$$m_k \in [0, |N| \max_{k'\in A}\{c_{k'} + d_{k'}\} + d_k], \forall k \in A_p$$

$$n \in [0, r_0].$$

$$(3)$$

Note that in the above formulation, we have restricted our free variables to those associated with N_p, thereby decreasing computational load and potentially allowing the problem to be solved on an IPU depending on the size of N_p.

IV. ALGORITHM

In this section we present an algorithm for approximating SPNI by leveraging small IPU hardware. Our algorithm first proceeds by generating an initial greedy solution, then refining this solution over several iterations.

First we define terminology and helper functions. A problem instance of SPNI is a tuple $\mathcal{P} = (G, \mathbf{c}, \mathbf{d}, s, t, r_0)$ where $G = (N, A)$ is a directed network, $\mathbf{c}$ is a vector representing arc lengths of A, $\mathbf{d}$ is a vector representing added interdiction arc lengths of A, $s \in N$ is the source node, t is the sink node, and r_0 is the interdiction budget.

We define PARTITION to be a function that takes in a graph $G = (N, A)$ and an integer n as inputs, then returns $P \subseteq \mathcal{P}(V)$, a connected partitioning of G where $\forall N_p \in P$, $|N_p| \approx n$. These partitions are how we will choose to define subproblems. The purpose of the size restriction is to ensure that subproblems are sufficiently small for IPU hardware. Additionally, PARTITION is necessarily non-deterministic ensuring different partitions on each call to prevent the solver from being stuck in a local attraction basin.

We define CALC-LENGTH, a function that takes in a problem instance $\mathcal{P} = (G = (N, A), \mathbf{c}, \mathbf{d}, s, t, r_0)$, and a set $A' \subseteq A$ where $|A'| \leq r_0$. It returns the shortest s-t path length in G after all arcs in A' have been interdicted.

We also define CALC-PATH that takes in a problem instance $\mathcal{P} = (G = (N, A), \mathbf{c}, \mathbf{d}, s, t, r_0)$, and a set $A' \subseteq A$ where $|A'| \leq r_0$. It returns the set $N' \subseteq N$ consisting of all nodes in the shortest s-t path in G after all arcs in A' have been interdicted.

Finally, function SOLVE takes in a problem instance $\mathcal{P} = (G = (N, A), \mathbf{c}, \mathbf{d}, s, t, r_0)$, a set $N_p \subseteq N$ representing a continuous partition of G, a node $t' \in N_p$ for the sink node of the subproblem, and a set of arcs $A' \subseteq A$ where $|A'| \leq r_0$ representing the current working solution. It formulates a subproblem based off these parameters according to (3), solves the subproblem, and returns a new arc set $A'' \subseteq A$ representing which arcs to interdict to maximize $\pi_{t'}$. Here is where computational work may be offloaded to an IPU. Note that due to the subproblem restriction, SOLVE can only modify the interdiction status of arcs associated with N_p, and thus all arcs not associated with N_p but in A' will remain in A''.

Initial solution generation The pseudocode is given in Algorithm 1. It receives in input a problem instance $\mathcal{P} = (G = (N, A), \mathbf{c}, \mathbf{d}, s, t, r_0)$ alongside an integer n representing the approximate nodes allowed per partition. To generate an initial solution to SPNI, the algorithm takes a greedy approach. As each arc costs exactly one unit of the budget to interdict, the algorithm iterates over each unit of the budget attempting to decide which arc to allocate this unit to. To make this decision, the algorithm first partitions the network using PARTITION

and additionally calculates the shortest s-t path with CALC-PATH. Note that it is sufficient to only consider edges along the s-t path for interdiction, since edges outside of this path will not affect the s-t path length after interdiction. For each node in the s-t path, the algorithm determines which partition this node belongs to, then solves a subpartition using this node as the sink node by calling SOLVE. It then takes the output solution, and calls CALC-LENGTH to determine this solution's impact on the s-t path length. If this path length is greater than the current maximum path length, the algorithm notes it as a potential candidate solution and updates the maximum. After repeating this process for each node in the s-t path, the algorithm chooses a candidate solution randomly, then continues this process until the budget has ran out. Note that in each iteration, the algorithm incrementally adds one arc to the current solution.

Solution refinement The pseudocode is given in Algorithm 2. It receives in input a problem instance $\mathcal{P} = (G = (N, A), \mathbf{c}, \mathbf{d}, s, t, r_0)$, an integer n representing the approximate number of nodes per partition, an integer λ for the number of iterations to run the algorithm, and a solution $A' \subseteq A$ to improve. Note that the given solution A' is expected to be the output of Algorithm 1. This algorithm attempts to improve the initial solution. To do so, it iterates λ times, and in each iteration, partitions the graph using PARTITION and calculates nodes in the current s-t path given the current interdiction solution using CALC-PATH. For each of these nodes, alongside all nodes found in the shortest path for the previous iteration, it determines what partition the node lies in and solves a subproblem where this node is the target node using SOLVE. CALC-LENGTH is then called to determine whether the outputted solution's s-t path length is greater than the current s-t path length, and if so, marks this solution as a potential candidate solution. After iterating over each node, if a better s-t path length is found, the current solution is chosen to be a randomly selected candidate solution. If a better s-t path length is not found, the algorithm attempts to explore alternate solutions by iterating over each arc in the current solution, temporarily deleting that arc from the solution, and observing whether this deletion can result in a better solution through a partitioning process similar to the above. If a better solution is found, the deletion of this arc becomes permanent and the better solution is adopted. Note that to prevent excessive computation, the algorithm keeps track of arcs which, upon being temporarily deleted, have not previously increased the s-t path length in a set named good-arcs. In future iterations, the algorithm does not consider temporarily deleting arcs found in good-arcs.

V. COMPUTATIONAL RESULTS

To evaluate the performance of our algorithm and similar to other network interdiction literature, we have generated grid networks with randomized edge weights and randomized interdiction weights. To each grid we added the source node s connected to all nodes in the first column, and the sink node t connected to all nodes in the last column, with all of these arcs

---

Algorithm 1: Initial solution generation

Require: A problem instance
 $\quad\quad \mathcal{P} = (G = (N, A), \mathbf{c}, \mathbf{d}, s, t, r_0)$
Require: Approximate number of nodes per partition
 $\quad\quad n$
1   $A' \leftarrow \{\}$
2   prev-nodes $\leftarrow$ CALC-PATH($\mathcal{P}, A'$)
3   $r' \longleftarrow 1$
4   **while** $r' \leq r_0$ **do**
5     $P \leftarrow$ PARTITION(G, n)
6   curr-nodes $\leftarrow$ CALC-PATH($\mathcal{P}, A'$)
7   next-sols $\leftarrow \{A'\}$
8   best-length $\leftarrow$ CALC-LENGTH($\mathcal{P}, A'$)
9   **foreach** *node* $\in$ *curr-nodes* $\cup$ *prev-nodes* **do**
10      $N_p \leftarrow \emptyset$
11   **foreach** $N'_p \in P$ **do**
12   **if** *node* $\in N'_p$ **then**
13        $N_p \leftarrow N'_p$
14   **end**
15   **end**
16   curr-sol $\leftarrow$ SOLVE($\mathcal{P}, N_p$, node, A')
17   curr-length $\leftarrow$ CALC-LENGTH($\mathcal{P}$, curr-sol)
18   **if** *curr-length > best-length* **then**
19   next-sols $\leftarrow \emptyset$
20   **end**
21   **if** *curr-length == best-length* **then**
22   next-sols $\leftarrow$ next-sols $\cup\{$curr-sol$\}$
23   **end**
24   **end**
25     $A' \xleftarrow{R}$ next-sols   ▷ Pick a solution randomly
26     $r' \leftarrow r' + 1$
27   **end**
28   **return** A'

---

having length 0 and interdiction length 0 to make the problem more challenging. All horizontal arcs are oriented forward and all vertical arcs are oriented down. All arcs not connected to s or t have a randomly assigned length and interdiction length, each in the range $[1, 10]$. Figure 1 depicts an example of grid 3x3.

Models were built using Pyomo, and were solved using the open-source solver CBC [7]. No significant difference was observed in experiments with Gurobi solver [13]. Graph representation and utilities such as Dijkstra's was done using NetworkX [14]. Graph partitioning was done using METIS [18]. All problems were ran on the University of Delaware's Caviness cluster which uses a Linux system and has exclusively Intel E5-2695 v4 18 core processors in its pool.

Below we plot the results of our computational experiments. The "Size" axis represents the number of nodes in the experiment's network ($|N|$). Note that by the structure of the grid network, the number of arcs can be calculated by $|A| = 2(|N| - 2)$. The "Quality" axis displays a metric to evaluate the quality of our solutions (defined in Eq. (4)).

Algorithm 2: Solution refinement

Require: A problem instance
$\mathcal{P} = (G = (N, A), \mathbf{c}, \mathbf{d}, s, t, r_0)$
Require: Approximate number of nodes per partition
n
Require: Number of iterations λ
Require: Current solution A'

```
1  prev-nodes ← CALC-PATH(P, A')
2  next-sols ← {A'}
3  best-length ← CALC-LENGTH(P, A')
4  good-arcs ← ∅
5  i ← 0
6  while i < λ do
7  │   P ← PARTITION(G, n)
8  │   curr-nodes ← CALC-PATH(P, A')
9  │   foreach node ∈ curr-nodes ∪ prev-nodes do
10 │   │   N_p ← ∅
11 │   │   foreach N'_p ∈ P do
12 │   │   │   if node ∈ N'_p then
13 │   │   │   │   N_p ← N'_p
14 │   │   │   end
15 │   │   end
16 │   │   curr-sol ← SOLVE(P, N_p, node, A')
17 │   │   curr-length ← CALC-LENGTH(P, curr-sol)
18 │   │   if curr-length > best-length then
19 │   │   │   next-sols ← ∅
20 │   │   end
21 │   │   if curr-length == best-length then
22 │   │   │   next-sols ← next-sols ∪{curr-sol}
23 │   │   end
24 │   end
25 │   prev-length ← CALC-LENGTH(P, A')
26 │   if best-length > prev-length then
27 │   │   A' ←ᴿ next-sols        ▷ Pick a solution randomly
28 │   else
29 │   │   foreach arc ∈ A'∖ good-arcs do
30 │   │   │   A'' ← A' ∖ {arc}
31 │   │   │   foreach node ∈ curr-nodes ∪ prev-nodes
32 │   │   │   │   partition = ∅
33 │   │   │   │   foreach p ∈ P do
34 │   │   │   │   │   if node ∈ p then
35 │   │   │   │   │   │   partition ← p
36 │   │   │   │   │   end
37 │   │   │   │   end
38 │   │   │   │   curr-sol ← SOLVE(P, N_p, node, A'')
39 │   │   │   │   curr-length ←
                       CALC-LENGTH(P, curr-sol)
40 │   │   │   │   if curr-length > prev-length then
41 │   │   │   │   │   A' ← curr-sol
42 │   │   │   │   else
43 │   │   │   │   │   good-arcs ← good-arcs ∪{ arc }
44 │   │   │   │   end
45 │   │   │   end
46 │   │   end
47 │   end
48 return A'
```
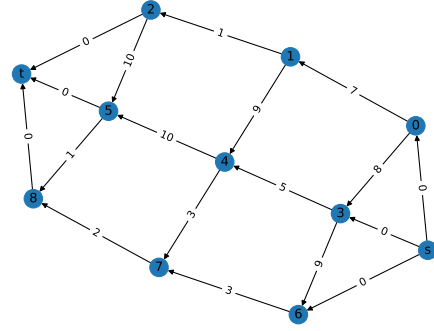

Fig. 1. Example of a generated 3x3 grid

We run both our refinement algorithm and CBC attempting to solve Equation (1) across the entire network—we shall hereby refer to the latter as the "full problem solver." Note that the full problem solver is significantly slower than the refinement algorithm, and thus, we timeout the full problem solver once it has exceeded the time that the refinement algorithm takes, then proceed to take the current best solution. *In every problem instance below, the full problem solver always timed out.* For any given experiment, if r is the shortest s-t path length after interdicting the arcs from the solution of our refinement algorithm and f the shortest s-t path length after interdicting the arcs from the full problem solver's solution, the ratio can be calculated by

$$(r - f)/\max(r, f). \tag{4}$$

Thus, the higher the quality, the better—positive quality indicates that the refinement solver performed better than the full problem solver.

We run three sets of experiments. In each experiment, we set the number of refinement iterations λ to 50. (However, the convergence is often observed earlier.) In the first set of experiments, represented by Figure 2, we set the interdiction budget r_0 to 0.25% of the number of arcs in the network and limit our subproblem size to $n = 20$ nodes. The average quality for this set of experiments is -0.008, implying that a subproblem size of only 20 nodes is sufficient for a good estimation of SPNI. In the second set of experiments, represented by Figure 3, we set the interdiction budget r_0 to 0.5% of the number of arcs in the network and limit our subproblem size to $n = 20$ nodes. The average quality for this set of experiments is -0.013, indicating relative consistency in estimation regardless of budget. In the final set of experiments, we set the interdiction budget r_0 to 0.25% of the number of arcs in the network and limit our subproblem size to $n = 40$ nodes. The average quality for this set of experiments is -0.012, implying that *an increase in subproblem size is not necessary for estimation and limited capacity IPU machines can already be integrated to solve this problem using decomposition-based refinement.*

Out of all 39 experiments ran, the difference between the s-t path length produced by the refinement solver and the path length produced by the full problem solver was ≥ -10 for 35

of the experiments. In 10 of these experiments, the refinement solver performed better than the full problem solver. *These results are significant as they indicate that in the cases when full problem solver was better the refinement missed the value of a single arc.*
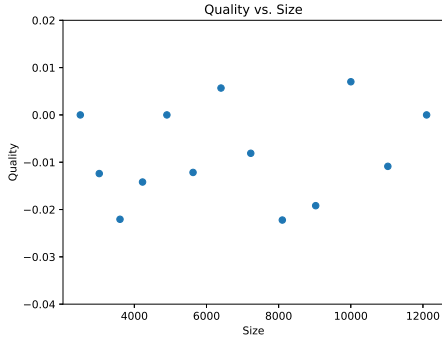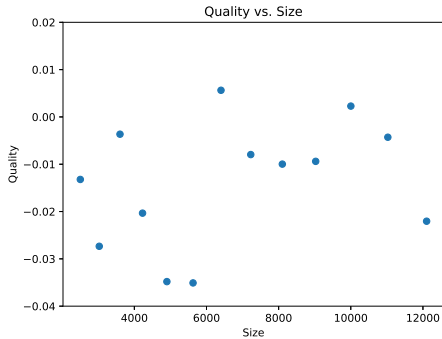

Fig. 2. Computational results for different size networks with parameters: $\lambda = 50$, $n = 20$, $r_0 = 0.0025|A|$. Each point corresponds to a graph. The horizontal axis corresponds to the number of nodes. The vertical axis is the quality defined in Eq. (4).

Fig. 3. Computational results for different size networks with parameters: $\lambda = 50$, $n = 20$, $r_0 = 0.05|A|$. Each point corresponds to a graph. The horizontal axis corresponds to the number of nodes. The vertical axis is the quality defined in Eq. (4).

VI. DISCUSSION

A. Solving on Ising Processing Hardware

The current results shown in this paper rely on using an exact solver to solve subproblems. We have previously attempted to utilize classical *heuristic* QUBO solvers, such as D-Wave's *qbsolv* and D-Wave Ocean, to solve these sub-problems, but such solvers provided poor quality solutions even on much smaller problem sizes that those we experiment with. Additionally, due to lack of accessibility to massive experiments with the IPU hardware, utilizing exact solvers is necessary to demonstrate the efficiency of our algorithm. We hope that as IPUs continue to develop in capacity, their superior solution quality and speed when compared to classical heuristic solvers will be able to make full use of the algorithm
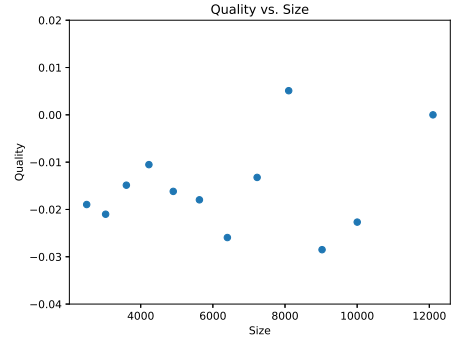

Fig. 4. Computational results for different size networks with parameters: $\lambda = 50$, $n = 40$, $r_0 = 0.0025|A|$. Each point corresponds to a graph. The horizontal axis corresponds to the number of nodes. The vertical axis is the quality defined in Eq. (4).

presented in this paper, generating solutions that are able to beat classical exact solvers consistently.

B. Algorithmic Challenges and Obstacles

Refinement algorithms for solving SPNI may face several challenges, the most notable being the problem of proper budget allocation amongst partitions. If an algorithm decides to improperly allocate budget to a subproblem, for example deciding to interdict an arc where this unit of budget would ideally be spent much further away in the graph, it proves to be very difficult to shift this budget significantly far away from its current position. Within our refinement approach, the movement of interdicted arcs is primarily dictated by random partitioning—for example, in one iteration, the subproblem solver may determine that a specific arc should be interdicted under the context of the current partititioning configuration, i.e. within the context of the current subproblem the arc lies in. But when we repartition in the next iteration, the subproblem solver reevaluates the placement of interdicted arcs—the cost unit used to interdict the previous arc may be relocated elsewhere based on the new subproblem formulation. Since the cost unit used for an interdicted arc may only shift to arcs relatively near this arc, it is unlikely for the cost unit to travel far away in the graph. To resolve this issue, a multilevel approach widely known in combinatorial scientific computing and applied graph algorithms [4] may be taken. Multilevel algorithms create a hierarchy of coarse problem representations, find a solution to the coarsest (small-est and most compressed) problem, then gradually derive a solution to the original finest problem by projecting the solution back from the coarse to fine levels and refining it using our algorithm, thereby potentially aiding in proper budget allocation. This approach has seen great success in solving various (hyper)graph problems [1], [26], [28] including hybrid quantum-classical multi- and single-level refinement for partitioning and community detection [29], [31], indicating that its application to SPNI may improve solution quality.

C. Performance

One of our main goals was to develop a trivial IPU parallelization friendly algorithm. Our current implementation of the proposed refinement does not take advantage of the fact that subproblems produced by a given partitioning can be solved independently of one another, since they only depend on the current working solution. Consequently, subproblems can be solved in parallel rather than sequentially, offering a potentially large optimization to the current performance of the algorithm.

D. Generalizability

The algorithm introduced in this paper does not handle SPNI problem instances in which the binary interdiction of a given arc may cost more than one unit of the interdictor's budget. The algorithm can currently handle instances where an arc between two nodes can take on multiple values of interdiction (e.g., if interdicted once add 1 to the length, if interdicted twice add 2 to the length) through the introduction of multi-edges with appropriate lengths, but instances in which a binary decision needs to be made (i.e., interdicted or not) and differing levels of interdiction cost are not supported.

Another scenario which the algorithm is unable to handle is in the case of different resource types. For example of such a scenario, let r_0 and r_1 represent the budget for two different types of resources, then arc k_0 may cost 1 of r_0 and 2 of r_1, but arc k_1 may cost 0 of r_0 and 3 of r_1. This restriction is not much of a concern, however, considering that in one of the most prominent works regarding SPNI, the authors of [16] also do not accommodate this case for their algorithmic cases.

VII. CONCLUSION

Solving SPNI holds crucial importance in the defense, engineering and other domains, addressing challenges in securing critical infrastructure from terrorist attacks, natural disasters and disrupting enemy supply networks among other applications. Current solutions to SPNI, however, are often too slow to be scalable to large networks, and thus prove to be impractical for real world purposes. In this paper, we have introduced a novel decomposition approach addressing SPNI harnessing the rapid solving abilities of IPU hardware to yield an approximate solution. We have further shown that solutions generated by our decomposition algorithm are extremely close in quality to state-of-the-art integer programming methods. We anticipate that as IPU hardware advancements continue to improve speed in solving optimization problems, our algorithm will enable an even quicker approximation of SPNI thereby facilitating a more efficient approach for real world scenarios of SPNI. More work is under way to produce higher quality approximations in even shorter amounts of time and extend the algorithm to more general forms of SPNI.

Reproducibility: Our results and code are available at https://github.com/krishxmatta/network-interdiction.

REFERENCES

[1] A. Abou-Rjeili and G. Karypis. Multilevel algorithms for partitioning power-law graphs. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 10 pp.–, 2006.

[2] Mine Altunay, Sven Leyffer, Jeffrey T Linderoth, and Zhen Xie. Optimal response to attacks on the open science grid. *Computer Networks*, 55(1):61–73, 2011.

[3] Michael O. Ball, Bruce L. Golden, and Rakesh V. Vohra. Finding the most vital arcs in a network. *Operations Research Letters*, 8(2):73–76, 1989.

[4] Cédric Chevalier and Ilya Safro. Comparison of coarsening schemes for multilevel graph partitioning. pages 191–205, 01 2009.

[5] Carleton Coffrin, Harsha Nagarajan, and Russell Bent. Evaluating ising processing units with integer programming. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 163–181. Springer International Publishing, 2019.

[6] H.W Corley and David Y Sha. Most vital links and nodes in weighted networks. *Operations Research Letters*, 1(4):157–160, 1982.

[7] John Forrest and Robin Lougee-Heimer. Cbc user guide. In *Emerging theory, methods, and applications*, pages 257–277. INFORMS, 2005.

[8] D. R. Fulkerson and Gary C. Harding. Maximizing the minimum source-sink path subject to a budget constraint. *Math. Program.*, 13(1):116–118, dec 1977.

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

[10] Fred W. Glover and Gary A. Kochenberger. A tutorial on formulating QUBO models. *CoRR*, abs/1811.11538, 2018.

[11] Noam Goldberg, Sven Leyffer, and Ilya Safro. Optimal response to epidemics and cyber attacks in networks. *Networks*, 66(2):145–158, 2015.

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

[13] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[14] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[15] Dian Huang, Zhaofang Mao, Kan Fang, and Lin Chen. Solving the shortest path interdiction problem via reinforcement learning. *International Journal of Production Research*, 61(1):31–48, 2023.

[16] Eitan Israeli and R. Kevin Wood. Shortest-path network interdiction. *Networks*, 40(2):97–111, 2002.

[17] Sahar Karimi and Pooya Ronagh. Practical integer-to-binary mapping for quantum annealers. *Quantum Information Processing*, 18(4), feb 2019.

[18] George Karypis and Vipin Kumar. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, September 1998.

[19] Sven Leyffer and Ilya Safro. Fast response to infection spread and cyber attacks on large-scale networks. *Journal of Complex Networks*, 1(2):183–199, 2013.

[20] Xiaoyuan Liu, Hayato Ushijima-Mwesigwa, Avradip Mandal, Sarvagya Upadhyay, Ilya Safro, and Arnab Roy. Leveraging special-purpose hardware for local search heuristics. *Computational Optimization and Applications*, 82(1):1–29, 2022.

[21] K. Malik, A.K. Mittal, and S.K. Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, 1989.

[22] Naeimeh Mohseni, Peter L McMahon, and Tim Byrnes. Ising machines as hardware solvers of combinatorial optimization problems. *Nature Reviews Physics*, 4(6):363–379, 2022.

[23] Apurba K. Nandi, Hugh R. Medal, and Satish Vadlamani. Interdicting attack graphs to protect organizations from cyber attacks: A bi-level defender–attacker model. *Computers Operations Research*, 75:118–131, 2016.

[24] Florian Neukart, Gabriele Compostella, Christian Seidel, David von Dollen, Sheir Yarkoni, and Bob Parney. Traffic flow optimization using a quantum annealer. *Frontiers in ICT*, 4, 2017.

[25] Claudio M. Rocco S. and José Emmanuel Ramirez-Marquez. A bi-objective approach for shortest-path network interdiction. *Computers & Industrial Engineering*, 59(2):232–240, 2010.

[26] Ilya Safro and Boris Temkin. Multiscale approach for the network compression-friendly ordering. *Journal of Discrete Algorithms*, 9(2):190–202, 2011.

[27] Javier Salmeron, Kevin Wood, and Ross Baldick. Worst-case interdiction analysis of large-scale electric power grids. *IEEE Transactions on Power Systems*, 24(1):96–104, 2009.

[28] Ruslan Shaydulin, Jie Chen, and Ilya Safro. Relaxation-based coarsening for multilevel hypergraph partitioning. *Multiscale Modeling & Simulation*, 17(1):482–506, 2019.

[29] Ruslan Shaydulin, Hayato Ushijima-Mwesigwa, Ilya Safro, Susan Mniszewski, and Yuri Alexeev. Network community detection on small quantum computers. *Advanced Quantum Technologies*, 2(9):1900029, 2019.

[30] J Cole Smith and Yongjia Song. A survey of network interdiction models and algorithms. *European Journal of Operational Research*, 283(3):797–811, 2020.

[31] Hayato Ushijima-Mwesigwa, Ruslan Shaydulin, Christian FA Negre, Susan M Mniszewski, Yuri Alexeev, and Ilya Safro. Multilevel combinatorial optimization across quantum architectures. *ACM Transactions on Quantum Computing*, 2(1):1–29, 2021.