Looking for the Maximum Independent Set: A New Perspective on the Stable Path Problem

Yichao Cheng[‡], Ning Luo[‡], Jingxuan Zhang[‡*], Timos Antonopoulos[‡], Ruzica Piskac[‡], Qiao Xiang[†‡],
[†]Xiamen University, [‡]Yale University, [*]Tongji University,

***Abstract*—The stable path problem (SPP) is a unified model for analyzing the convergence of distributed routing protocols (*e.g.*, BGP), and a foundation for many network verification tools. Although substantial progress has been made on finding solutions (*i.e.*, stable path assignments) for particular subclasses of SPP instances and analyzing the relation between properties of SPP instances and the convergence of corresponding routing policies, the non-trivial challenge of finding stable path assignments to generic SPP instances still remains. Tackling this challenge is important because it can enable multiple important, novel routing use cases. To fill this gap, in this paper we introduce a novel data structure called solvability digraph, which encodes key properties about stable path assignments in a compact graph representation. Thus SPP is equivalently transformed to the problem of finding in the solvability digraph a maximum independent set (MIS) of size equal to the number of autonomous systems (ASes) in the given SPP instance. We leverage this key finding to develop a heuristic polynomial algorithm GREEDYMIS that solves strictly more SPP instances than state-of-the-art heuristics. We apply GREEDYMIS to designing two important, novel use cases: (1) a centralized interdomain routing system that uses GREEDYMIS to compute paths for ASes and (2) a secure multi-party computation (SMPC) protocol that allows ASes to use GREEDYMIS collaboratively to compute paths without exposing their routing preferences. We demonstrate the benefits and efficiency of these use cases via evaluation using real-world datasets.**

I. INTRODUCTION

The stable path problem (SPP) [1], [2] is a unified model for distributed routing protocols. It is used in various routing protocol convergence studies (*e.g.*, [1]–[7]), but it has also been adopted by many emerging network configuration tools as their foundation [8]–[11]. Specifically, in SPP, the behavior of path vector routing protocols, such as the Border Gateway Protocol (BGP) [12], is abstracted into a *simple path vector protocol* (SPVP), where each router receives paths from neighbors, selects the best path to use, and extends and sends the selected path to all its neighbors in the form of a path vector. Using this abstraction, an SPP instance is composed of a network graph G, where each node has a set of permitted paths and a ranking function that ranks these paths, and executes SPVP. The SPP problem is defined as a decision problem that checks whether such an instance admits a *stable path assignment*, in which each router selects the best path available to it and does not change the selected path. Originally proposed to only model path vector protocols, SPP has recently been used to model generic distributed routing protocols [8]–[10] (*e.g.*, OSPF).

The seminal paper [1] proves the NP-completeness of SPP. This means that a brute-force solution that enumerates all path assignments is not feasible as it might be de facto exponential. Existing studies have made substantial progress on using SPP to analyze the convergence of BGP policies [3]–[7], [13]–[16]. Some studies develop polynomial time algorithms to find solutions (*i.e.*, *stable path assignments*) for different subclasses of SPP instances that have unique solutions and converge [1], [2], [4]. There are also studies that aim to identify the relation between properties of SPP instances and the convergence of the corresponding BGP policies. For example, dispute-wheel-free [1], [2], [5], Gao-Rexford conditions [3], and acyclic path digraph [6] each implies BGP convergence.

Despite all this progress, a non-trivial basic challenge still remains: *how to find stable path assignments to generic SPP instances other than by enumeration?* This challenge is overlooked by existing studies mainly because SPP solvability (even unique solvability) does not guarantee the convergence of BGP. However, in this paper, we argue that tackling this challenge is of great importance for several reasons.

First, being able to find solutions to generic SPP instances enables autonomous systems (ASes) to collaboratively compute interdomain paths without worrying about convergence issues of their interdomain policies, which improves the flexibility of end-to-end interdomain route control [17]. In practice this means that a trusted controller (*e.g.*, [17]–[21]) or a secure multi-party computation (SMPC) system (*e.g.*, [22], [23]) could be deployed to compute the stable path assignment and send to ASes, instead of letting ASes exchange their private routing information via BGP. Second, better understanding of SPP solvability can shed light on understanding the gap between solvable SPP and safe SPP (*i.e.*, whose corresponding BGP policies always converge). Third, with many network verification tools using SPP as the foundation to model network configurations [8]–[11], finding all stable solutions for a given SPP instance has the potential of significantly improving the performance of network verification tools.

This paper makes the following **main contributions**:

Key finding: a solvability digraph data structure for reasoning about SPP solvability (Section III). We design a novel data structure solvability digraph to encode key properties about the stable path assignment in a compact graph representation. We show that the SPP is equivalent to the problem of finding a maximum independent set (MIS) of size $|V(G)|$ (the number of ASes in the SPP instance) in solvability digraph. This equivalence gives us a new perspective on characterizing and understanding SPP solvability, because MIS problem is a well-known NP-hard problem for which many

The corresponding author is Qiao Xiang.

exact and heuristic algorithms have been developed [24], [25]. To the best of our knowledge, this is the first general result on solving generic SPP instances.

A polynomial time heuristic GREEDYMIS **(Section IV)**. Leveraging the equivalence between SPP and MIS, we developed the GREEDY++ algorithm for solving generic SPP instances. We prove that GREEDY++ solves strictly more SPP instances than state-of-the-art SPP algorithms [1], [4].

Two novel interdomain use cases (Section V). As a proof of concept, we apply the GREEDYMIS algorithm to three novel use cases. First, we design a centralized interdomain routing system, where a logically centralized server collects ASes interdomain policies and uses GREEDYMIS to find a stable path assignment for ASes. This platform is suitable for collaborative interdomain networks where ASes have high-level trust to each other and are willing to work together to manage interdomain routing (*e.g.*, an example of such interdomain networks is the Large Hadron Collider [26]).

Second, we developed an SMPC protocol that allows ASes to collaboratively use GREEDY++ to find a stable path assignment without exposing their routing preferences. This protocol is suitable for interdomain networks where ASes want to achieve more flexible interdomain routing while preserving policy privacy.

Evaluation to demonstrate benefits and efficiency (Section VI). We performed extensive experiments on three use cases listed above, using real-world datasets. Our results shows that the solvability digraph and the GREEDYMIS algorithm work well in practice, with reasonable overhead.

II. BACKGROUND AND RELATED WORK

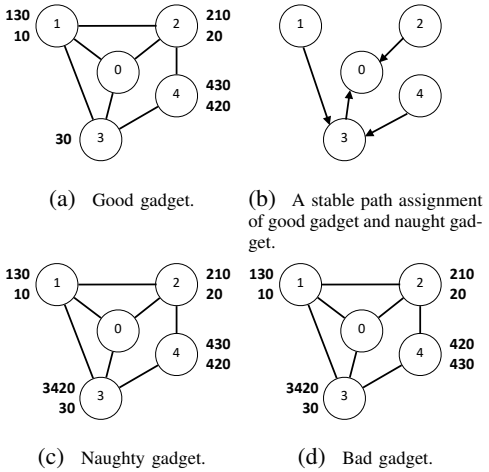
A. Stable Path Problem in a Nutshell

We provide a brief overview of the stable path problem. For simplicity of presentation, we focus on the original SPP model of BGP in [1], and refer readers to [8]–[10] for details on how SPP models other routing protocols (*i.e.*, distance vector and link state protocols).

Let $G = (V, E)$ be an undirected graph. $V = \{0, \ldots, n\}$ corresponds to the set of $n + 1$ ASes, and AS 0 is the destination AS. An edge $(i, j) \in E$ represents the BGP peering between AS i and j. A path p in G is a sequence of nodes $(v_k, v_{k-1}, \ldots, v_0)$ such that $(v_i, v_{i-1}) \in E$ for each $1 \leq i \leq k$. A simple path is a path without loops, or in other words a path $(v_k, v_{k-1}, \ldots, v_0)$ such that for any $1 \leq i < j \leq k$, $v_i \neq v_j$. If $k = 0$, then (v_0) is the trivial path of zero length from v_0 to itself. For each $0 \leq i \leq k-1$, path $(v_i, v_{i-1}, \ldots, v_0)$ is a *suffix* of p, and we use **suffixes**(p) to denote the set of all suffixes of p. For a path $p = (v_k, v_{k-1}, \ldots, v_0)$, node v_{k-1} is called the *next hop* of p, and we say that path p is one hop away from path $(v_{k-1}, v_{k-2}, \ldots, v_0)$. The *concatenation* of two non-empty paths, $p_2 = (v_k, v_{k-1}, \ldots, v_j)$ and $p_1 = (v_j, v_{j-1}, \ldots, v_0)$, is the path $p_3 = (v_k, v_{k-1}, \ldots, v_j, v_{j-1}, \ldots, v_0)$ and we denote that by $p_3 = p_1 p_2$. For a given graph G, we let $V(G)$ denote its vertex set and let $E(G)$ denote its edge set.

A simple path is a path without loops. In SPP, each $v \in V$ aims to find a simple path in G to the destination AS. To this end, each v executes a simple path vector protocol (SPVP), which is an abstract version of BGP. In particular, each v is assigned

1) a set of *permitted paths* $\mathcal{P}^v$, such that $\mathcal{P}^0 = \{(0)\}$ and for $v \neq 0$, $\mathcal{P}^v$ is composed of simple paths starting from v and ending at 0 as well as the empty path ϵ, and
2) a ranking function $\lambda^v : \mathcal{P}^v \to \mathcal{N}$ that ranks paths in $\mathcal{P}^v$ in a descending order of preference, such that for $p_1, p_2 \in \mathcal{P}^v$, $\lambda^v(p_1) > \lambda^v(p_2)$ means p_1 is preferred over p_2 by v.



(a) Good gadget. (b) A stable path assignment of good gadget and naught gadget. (c) Naughty gadget. (d) Bad gadget.

Fig. 1: Examples of stable path problems taken from [1].

Consider an example given in Fig. 1a: to reach the destination node (node "0"), node 2 can use two paths: either to go to 0 directly, or via node 1. The path (210) is higher ranked than the path (20).

Given two paths $p_1, p_2 \in \mathcal{P}^v$, p_1 and p_2 can only be equally ranked if they have the same next hop, which is that $p_1 = (v, v')p'_1$ and $p_2 = (v, v')p'_2$ for some $v' \in V$.

Permitted paths and ranking functions capture the import/-export and selection policies of ASes, respectively. Given a path $(v_k, v_{k-1}, \ldots, v_0)$ in G, it is a permitted path if and only if (1) v_0 announces (v_0) to v_1, (2) v_k's import policy does not filter out $(v_{k-1}, \ldots, v_0)$, and (3) for each $i, 0 < i < k$, the import policy of v_i does not filter out $(v_{i-1}, \ldots, v_0)$ and v_i's export policy announces $(v_i, \ldots, v_0)$ to v_{i+1}.

Each v locally maintains an incoming routing information base $rib\text{-}in_v$ to store the latest paths sent by its neighbors. When v receives a path p_w from a neighbor w, v extends it to $p'_w = (v, w)p_w$ and checks whether $p'_w \in \mathcal{P}^v$. If not, p'_w is updated to ϵ. v uses p'_w to replace the last path sent by w in $rib\text{-}in_v$ and selects best path p^*_v as $\arg\max_{p \in rib\text{-}in_v} \lambda^v(p)$. If $p^*_v = p'_w$, v sends it to all its neighbors.

A path assignment π is a function that maps each $v \in V$ to a path $\pi(v) \in \mathcal{P}^v$, where $\pi(0)$ is always the trivial path (0). Given a path assignment π, it is a *stable path assignment* if $\pi(0) = (0)$ and for each $1 \leq i \leq n$, $\pi(i) = best(i, \{(i, j)\pi(j) : (i, j) \in E\} \cap \mathcal{P}^i)$, where $best(i, \varnothing) = \epsilon$, and $\text{best}(i, S) = \arg\max_{p \in S} \lambda^i(p)$ for $S \neq \varnothing$.

An instance of the stable path problem is then the triple $\mathcal{S} = (G, \mathcal{P}, \Lambda)$, where $\mathcal{P} = \cup_{i \in V(G)} \mathcal{P}^i$ and $\Lambda = \{\lambda^i\}_{i \in V(G)}$. The decision version of the stable path problem is defined as:

Definition II.1 (Stable Path Problem)**.** Given an SPP instance $\mathcal{S} = (G, \mathcal{P}, \Lambda)$, does $\mathcal{S}$ have a stable path assignment?

For completeness of presentation, Fig. 1 shows some typical SPP instances from [1] (*i.e.*, good gadget in Fig. 1a, naughty gadget in Fig. 1c and bad gadget in Fig. 1d). Both good gadget and naughty gadget have the same stable path assignment $\{(0), (130), (20), (30), (430)\}$, which implicitly defines a tree rooted at AS 0 (Fig. 1b), while the bad gadget does not have a solution and its SPVP protocol always diverges.

B. Related Work

The original purpose of SPP was to analyze behaviors of BGP, a path vector protocol deployed in interdomain routing [3], enterprise networks [27], and data centers [28]. In particular, SPP has been shown to be a powerful model for understanding the convergence of different BGP policies. The seminal paper [1] proves that SPP is NP-complete via a reduction from the 3-SAT problem. It shows that the existence of a solution to an SPP instance, even a unique one, does not imply the convergence of the corresponding BGP policies. One such example is the naughty gadget in Fig. 1c, where there is a unique stable path assignment but the network may oscillate for arbitrarily long before converging. [1] characterized a subclass of SPP instances using the now-classic *dispute-wheel-free* condition, such that if an SPP instance has no *dispute wheel*, then it has a unique solution and the network always converges to that solution (*e.g.*, Fig. 1a). For this subclass of SPP instances, [1] also presented a heuristic algorithm called GREEDY, which finds the unique solution in polynomial time.

Since then, many studies have made substantial progress on designing polynomial-time algorithms to find solutions to different subclasses of SPP instances (*e.g.*, [3], [4]), as well as identifying the relation between different subclasses of SPP instances and the convergence of their corresponding BGP policies (*e.g.*, [5]–[7], [13]–[16]). For example, [3] proposes the Gao-Rexford condition that has become the foundation of Internet stability, and [4] extends the *dispute-wheel-free* condition to a GREEDY$^+$-solvable condition that identifies a larger subclass of SPP instances which are uniquely solvable and converge. The proposed GREEDY$^+$ algorithm solves strictly more SPP instances than the GREEDY algorithm in [1]. [5]–[7], [13], [29] did not develop algorithms for SPP, but instead, they introduced graph data structures (*i.e.*, dispute digraph, path digraph, multipath digraph, and p-graph) to derive new sufficient conditions (*e.g.*, acyclic graphs) for SPP instances to have unique solutions and converge. We refer readers to [2] for a comprehensive survey on SPP and BGP convergence.

Recently, people have started to explore SPP from a different angle. For instance, many network configuration verification tools [8]–[11], [30] choose SPP as the foundation for modeling not only BGP, but also generic distributed routing protocols [8]–[10]. By encoding SPP instances as logical formulas (*e.g.*, SMT formulas), they enumerate over every possible message sequence in executions of routing protocols, so as to compute all stable path assignments. They then verify whether these assignments satisfy certain network properties (*e.g.*, reachability). However, these tools do not scale well due to the costly nature of exhaustive search.

Despite these studies, there has been little progress made on finding stable path assignments to generic SPP instances. Tackling this challenge can enable important use cases such as flexible interdomain route control, policy convergence verification, and faster network verification. As such, in the next few sections, we present our main result that tackles this challenge and show how to use it to design new routing use cases.

III. MAIN RESULT: SOLVING GENERIC SPP THROUGH SOLVABILITY DIGRAPH

In this section, we present our key findings on the solvability of generic SPP instances. We first briefly characterize the main properties of stable path assignments. Then we introduce the novel *solvability digraph* data structure which encapsulates these properties for a given SPP instance, and demonstrate how to use the solvability digraph data structure to reason about SPP solvability.

Main properties of stable path assignments. Given any SPP instance $\mathcal{S} = (G = (V, E), \mathcal{P}, \Lambda)$, if π is a stable path assignment for this instance, then the set of paths assigned by π, $\pi(V) = \{\pi(i) : i \in V\}$, has the following three properties:

Property III.1. For each AS i, $\pi(V)$ contains one and only one path from $\mathcal{P}^i$.

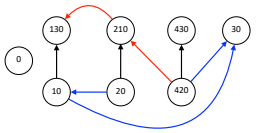
Property III.2. For any path $p \in \pi(V)$, **suffixes**$(p) \subseteq \pi(V)$.

Property III.3. For any $v \in V$, if w is a neighbor of v, then $\lambda^v(\pi(v)) \geq \lambda^v((v, w)\pi(w))$.

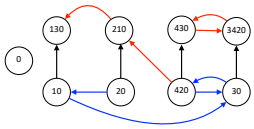
These three properties all directly follow from the definition of stable path assignments, as shown in [1]. Note that Property III.2 implies that a stable path assignment implicitly defines a tree rooted at the destination AS (*e.g.*, Figure 1b), while Property III.3 states that this tree is locally optimal.

The solvability digraph. We propose the novel *solvability digraph* data structure, which takes motivation from existing graph data structures such as the dispute digraph [5], the path digraph [6], and the multipath digraph [7]). Different from their focus on analyzing the convergence of SPP, we aim to analyze the solvability of SPP using solvability digraph.

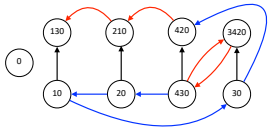
The idea behind the solvability digraph is to encapsulate the three main properties of stable path assignments in a compact graph representation, so that SPP can be equivalently transformed to a classic graph theoretical problem. Note that for simplicity of presentation, in the remaining of this paper, we only consider SPP instances where for each AS i, $\lambda^i(p_1) \neq \lambda^i(p_2)$ for any distinct pair of paths p_1 and p_2 in $\mathcal{P}^i$ (*i.e.*, each ranking function λ^i totally orders $\mathcal{P}^i$). To handle paths p_1 and p_2 that are equally preferred by some AS i, and the fact that there may not be any edges between the two respective nodes in the solvability digraph we present below we can extend the definition of the solvability digraph with a separate type of preference edge connecting two such paths



(a) The solvability digraph for good gadget.



(b) The solvability digraph for naughty gadget.



(c) The solvability digraph for bad gadget.

Fig. 2: The solvability digraphs for the SPP instances in Fig. 1

in the solvability digraph, and carefully adapt the algorithm accordingly.

Definition III.4 (Solvability Digraph)**.** Given a stable path problem instance $\mathcal{S} = (G, \mathcal{P}, \Lambda)$, the *solvability digraph* of $\mathcal{S}$ is a directed graph G^s, where the vertex set $V(G^s)$ is $\mathcal{P}$, and for any two distinct paths $p_1, p_2 \in V(G^s)$, $(p_1, p_2) \in E(G^s)$ if one of the following conditions is satisfied:

1) p_1 and p_2 belong to the same $\mathcal{P}^i$ and $\lambda^i(p_1) < \lambda^i(p_2)$;
2) there exists a path p_3 such that p_2 and p_3 belong to the same $\mathcal{P}^i$, $\lambda^i(p_2) \neq \lambda^i(p_3)$, and p_3 is a suffix of p_1;
3) there exists a path p_3 such that p_1 and p_3 belong to the same $\mathcal{P}^i$, $\lambda^i(p_3) > \lambda^i(p_1)$, and p_3 is one hop away from p_2 such that $p_3 = (v_i, v_j)p_2$ for some j.

Note that in the above definition of solvability digraph, the three conditions for the existence of an edge are distinct, and they each correspond to a main property of stable path assignments. In Fig. 2, we illustrate the respective solvability digraph for each SPP instance from Fig. 1. The black, red and blue edges represent edges in solvability digraph satisfying condition (1), (2), and (3), respectively.

A necessary and sufficient condition for generic SPP solvability. Having presented the solvability digraph data structure and shown how it encapsulates the three main properties of stable path assignments in a compact graph representation, we are now ready to present our main result. Specifically, we give the following necessary and sufficient condition for generic SPP solvability.

Theorem III.5. *Given any SPP instance $\mathcal{S} = (G, \mathcal{P}, \Lambda)$ and the corresponding solvability digraph G^s, a path assignment π is a solution to $\mathcal{S}$ if and only if $\pi(V(G)) = \{\pi(v) : v \in V(G)\}$ is an independent set of size $|V(G)|$ in G^s.*

Proof. We first prove that if π is a stable path assignment for $\mathcal{S}$, then $\pi(V(G))$ is an independent set in G^s. To this end, we show that for any pair of distinct ASes, i and j, $(\pi(i), \pi(j)) \notin E(G^s)$. Firstly, $(\pi(i), \pi(j))$ does not satisfy condition (1) in Definition III.4 because $\pi(i)$ and $\pi(j)$ are permitted paths from different ASes. Secondly, suppose $(\pi(i), \pi(j))$ is an edge satisfying condition (2) in Definition III.4. Then, there exists a path $p_j \in \mathcal{P}^j$ that is a suffix of $\pi(i)$. From Property III.2, p_j is also in $\pi(V(G))$. By Property III.1, both p_j and $\pi(j)$ being in $\pi(V(G))$ is a contradiction. As such, $(\pi(i), \pi(j))$ does not satisfy condition (2). Thirdly, suppose $(\pi(i), \pi(j))$ is an edge satisfying condition (3) in Definition III.4. Then, there exists a path $p_i \in \mathcal{P}^i$ such that $p_i = (i, j)\pi(j)$ and $\lambda^i(p_i) > \lambda^i(\pi(i))$, which is a direct contradiction with Property III.3.

Next, we prove that if $V^I \subseteq V(G^s)$ is an independent set of size $|V(G)|$ in G^s, then there exists some stable path assignment π, such that $\pi(V(G)) = V^I$. For convenience, let $V(G) = \{0, 1, \ldots, n\}$ denote the set of ASes, where 0 is the destination AS. Then we also have $n+1 = |V(G)|$. No pair of distinct paths in V^I are from the same AS to the destination, for otherwise, the fact that no two paths at the same AS are equally preferred would imply the existence of an edge by condition (1) in Definition III.4. Hence, each node in V^I corresponds to a path from a different AS to the destination. Let p_i denote the path in V^I from AS i to the destination, for each $0 \leq i \leq n$, so that $V^I = \{p_i : 0 \leq i \leq n\}$. Let π be the path assignment defined by $\pi(i) = p_i$ for each $0 \leq i \leq n$. We first have that $\pi(0) = (0)$, since $\mathcal{P}^0 = \{(0)\}$. Then, to prove that π is a stable path assignment, it's sufficient to show that for each $1 \leq i \leq n$, **(a)** $\pi(i) = (i, j)\pi(j)$, for some neighbor j of i in G, and **(b)** for any neighbor j of i, $\lambda^i(\pi(i)) \geq \lambda^i((i, j)\pi(j))$.

To prove **(a)**, assume that for some AS i, it is the case that $\pi(i) = (i, j)p'_j$, where $p'_j \neq \pi(j)$. This means that $\lambda^j(p'_j) \neq \lambda^j(\pi(j))$, thus implying an edge $(\pi(i), \pi(j)) \in E(G^s)$ by condition (2) in Definition III.4, which is impossible since $\pi(i)$ and $\pi(j)$ are both in independent set V^I of G^s. To prove **(b)**, assume, by way of contradiction, that there exists a path $p'_i = (i, j)\pi(j)$ where j is a neighbor of i, such that $p'_i \neq \pi(i)$ and $\lambda^i(p'_i) > \lambda^i(\pi(i))$. This implies an edge $(\pi(i), \pi(j)) \in E(G^s)$ by condition (3) in Definition III.4, which is a contradiction. Thus, we can conclude that π is indeed a stable path assignment. ■

As an example of Theorem III.5, we have that the stable path assignment to both the good gadget and the naughty gadget, which is $\{(0), (130), (20), (30), (430)\}$ as shown in Fig. 1b, is an independent set in the corresponding solvability digraphs (Fig. 2a and Fig. 2b, respectively). The novel solvability digraph data structure and Theorem III.5 give us a new perspective on characterizing and understanding SPP solvability, by transforming it into a maximum independent set (MIS) problem, a well-known NP-hard problem for which many exact and heuristic algorithms have been developed [24], [25]. This graph theoretic approach opens up many opportunities for the development of new tools and algorithms that tackle various SPP related problems (*e.g.*, flexible route control, convergence checking, and network verification). As proof of concepts, in the next two sections, we leverage this new finding to develop a polynomial time heuristic algorithm for solving generic SPP, and apply it to designing two novel interdomain use cases.

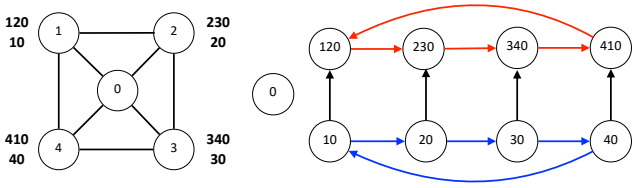


Fig. 3: DISPUTE-WHEEL-4 (left) and the solvability digraph of DISPUTE-WHEEL-4 (right)

.

```
1  GREEDYMIS(G^s, |V(G)|):
2    S_0 = {}; G^s_0 = G^s; i = 1
3    while (|S_{i-1}| < |V(G)| and |V(G^s_{i-1})| > 0):
4      B_i = NodesWithZeroOutDegree(G^s_{i-1})
5      if (|B_i| = 0):
6        B_i = NodeWithNoPrefEdges(G^s_{i-1})
7      if (|B_i| = 0):
8        B_i = NodeWithLowestOutDegree(G^s_{i-1})
9      G^s_i = G^s_{i-1}
10     for v ∈ B_i:
11       N = Neighbors(G^s_{i-1}, v)
12       G^s_i = Remove(G^s_i, N ∪ {v})
13     S_i = S_{i-1} ∪ B_i
14     i = i + 1
15   return S_{i-1}
```

Fig. 4: The GREEDYMIS algorithm

IV. GREEDYMIS: A HEURISTIC ALGORITHM FOR SPP

Leveraging the main result in Section III, we develop a heuristic polynomial time algorithm GREEDYMIS to solve SPP by looking for MIS on the solvability digraph. We present the details of GREEDYMIS and prove that it solves strictly more SPP instances than a state-of-the-art polynomial time algorithm.

A. GREEDYMIS: *Details*

Given $\mathcal{S} = (G, \mathcal{P}, \Lambda)$, GREEDYMIS takes the corresponding solvability digraph G^s, and $|V(G)|$ as input, and returns an independent set of G^s. The edges in the solvability digraph are labeled with their respective types, (type (1), (2), or (3)), corresponding to which one of the three conditions in Definition III.4 they satisfy.

Basic idea. The basic idea of GREEDYMIS consists in iteratively adding nodes from the solvability digraph G^s into an independent set S_i until either $|S_i| = |V(G)|$ or there are no more nodes left in G^s that can be added to S_i (Fig. 4). Specifically, in each iteration i, GREEDYMIS constructs B_i from one of the three subroutines, with graph $H \subseteq G^s$ as input: (1) NodesWithZeroOutDegree(H) (Line 4) returns the set of nodes with no outgoing edges in H; (2) NodeWithNoPrefEdges(H) (Line 6) returns a singleton set $\{p\} \subseteq V(H)$ such that $p \in \mathcal{P}^i$ for some vertex i and $\mathcal{P}^i \cap V(H) = \{p\}$; (3) NodeWithLowestOutDegree(H) (Line 8) non-deterministiacly returns a singleton set whose only node is such that no other node in H has a smaller out-degree. Then, GREEDYMIS obtains G^s_i as a subgraph of G^s_{i-1}, induced by removing B_i as well as the neighbors of nodes in B_i from the vertex set (Line 11, 12).

Properties. The following three properties show that GREEDYMIS always terminates in polynomial time and outputs an independent set of the input solvability digraph G^s.

Property IV.1. GREEDYMIS exits after at most n iterations.

This is because at each iteration i, $|B_i| > 0$ and thus $|S_{i+1}| > |S_i|$, while $|S_i|$ can be at most $|V(G)|$.

Property IV.2. At any iteration i before GREEDYMIS exits, for any pair of distinct nodes p and p' in G^s_{i-1}, if (p, p') is not an edge in G^s_{i-1}, then it is also not an edge in G^s.

This is true because when edges and nodes are removed from the solvability digraph (Line 12), an edge will only be removed if at least one of its two end nodes is removed.

Property IV.3. At each iteration i before GREEDYMIS exits, S_i is an independent set of the input solvability digraph G^s.

Proof. By Property IV.2, it is sufficient to show that (1) each B_i is an independent set of G^s_{i-1}, and that (2) for any pair of iterations i and j with $i > j$, there is no edge between B_i and B_j in G^s_{j-1}. Point (1) follows from the fact that B_i constructed from either Line 6 or Line 8 has only one node, while any edge in B_i constructed from Line 4 would cause a contradiction by implying that some node in the set has an outgoing edge. Point (2) follows directly from Lines 11 and 12, for they imply that there is no edge between B_j and $V(G^s_j)$ in G^s_{j-1}, and we know that given $i > j$, $B_i \subseteq V(G^s_{i-1}) \subseteq V(G^s_j)$. ■

Example. We demonstrate a successful execution of GREEDYMIS using the SPP instance DISPUTE-WHEEL-4 in Fig. 3. Notice that DISPUTE-WHEEL-4 has at least two solutions, namely $\{(120), (340), (40), (20)\}$ and $\{(10), (230), (30), (410)\}$. This can be verified noting that both these sets form independent sets in the solvability digraph corresponding to DISPUTE-WHEEL-4 (Fig. 3),

To see how GREEDYMIS will non-deterministically find one of the two solutions, notice that on the network graph in Fig. 3, initially the algorithm finds no nodes to include in B_1, at both Line 4 and Line 6, thus choosing one of the nodes in $\{(120), (230), (340), (410)\}$ at Line 8. Without loss of generality, let the node chosen be the path (120). Then, the paths $\{(120), (10), (230), (410)\}$ are removed from the solvability digraph at Line 12 in iteration 1, and, because they have no outgoing edges in G^s_1, the paths (340) and (40) are selected at Line 4 in iteration 2. As a result, path (30) is thus removed from the graph. In the final iteration, by having no outgoing edges, the path (20) is selected so that $\{(120), (340), (40), (20)\}$ is the independent set output by GREEDYMIS. Since the tie-breaking at Line 8 is arbitrary and non-deterministic, the algorithm could have also found the solution $\{(10), (230), (30), (410)\}$.

B. GREEDYMIS: *Analysis*

We conduct rigorous analysis to prove that GREEDYMIS solves strictly more SPP instances than a state-of-the-art SPP polynomial algorithm [4] called GREEDY$^+$. We omit the proofs of lemmas due to space limit.

A review on GREEDY$^+$. We first briefly review the GREEDY$^+$ algorithm. Given a path $p \in \mathcal{P}$ and a set $S \subseteq \mathcal{P}$, we say

that p is *consistent with* the set S if $p = \epsilon$, or $p = (0)$, or $p = (v, w)p'$, where p' is some path in S, $(v, w) \in V(G)$, and p' is consistent with S. For each $v \in V(G)$, the *useful set* of paths, denoted by $\bar{\mathcal{P}}^v$, is initialized with the paths in $\mathcal{P}^v$ that are consistent with $\mathcal{P}$, where we recall that $\mathcal{P}^v$ is the set of permitted paths from v to the destination. Given input SPP instance $\mathcal{S} = (G, \mathcal{P}, \Lambda)$ and the useful set $\bar{\mathcal{P}} = \cup_{v \in V(G)} \bar{\mathcal{P}}^v$, GREEDY$^+$ proceeds as follows, for each iteration $i \geq 1$. In the beginning, $V_0 = \{0\}$ and $\bar{\mathcal{P}}_0 = \bar{\mathcal{P}}$.

i) Prune paths from $\bar{\mathcal{P}}_{i-1}$: for any $v \in V(G) \setminus V_{i-1}$, if v has a neighbor $w \in V_{i-1}$, then we know $|\bar{\mathcal{P}}^w_{i-1}| = 1$ and let let p_w denote that path. Remove from $\bar{\mathcal{P}}_{i-1}$ all paths in $\bar{\mathcal{P}}^v_{i-1}$ that have a lower ranking than path $(v, w)p_w$.
ii) Prune paths from $\bar{\mathcal{P}}_{i-1}$: for any $v \in V(G) \setminus V_{i-1}$, remove from $\bar{\mathcal{P}}_{i-1}$ any $p \in \bar{\mathcal{P}}^v_{i-1}$ that is not *consistent* with $\bar{\mathcal{P}}_{i-1}$.
iii) Constructs the set of candidate vertices, C_i, by selecting *all* $v \in V(G) \setminus V_{i-1}$, such that the highest ranked path in $\bar{\mathcal{P}}^v_{i-1}$ either is ϵ or has a next hop in V_{i-1}. Additionally, for each $v \in C_i$, prune all but the highest ranked path in $\bar{\mathcal{P}}^v_{i-1}$ from $\bar{\mathcal{P}}_{i-1}$. Let $\bar{\mathcal{P}}_i = \bar{\mathcal{P}}_{i-1}$ and $V_i = V_{i-1} \cup C_i$. If $|C_i| = 0$, then exit.

The basic idea of GREEDY$^+$ is to start from the destination AS 0, and then iteratively "grow" a stable path assignment, such that after each iteration i, the useful set of paths left for any $v \in V_i$, $\bar{\mathcal{P}}^v_i$, is a singleton set containing the path assigned to v, while for each $v \in V(G) \setminus V_i$, a path is not assigned yet. In the remaining of this paper, we use P_i to denote the set of paths assigned to the ASes in V_i by GREEDY$^+$ after iteration i, *i.e.*, $P_i = \cup_{v \in V_i} \bar{\mathcal{P}}^v_i$ and $|P_i| = |V_i|$.

GREEDY$^+$ was proposed to analyze BGP convergence using the SPP model. It guarantees that given $\mathcal{S} = (G, \mathcal{P}, \Lambda)$, if it finds a stable path solution, this solution must be unique and the corresponding BGP policie must converge. As such, the SPP instances it can solve are limited. For example, GREEDY$^+$ fails to find a solution to DISPUTE-WHEEL-4 in Fig. 3. This is because no paths are pruned at Step i) or ii), and at each AS other than 0, the highest ranked path does not have a next hop in $V_0 = \{0\}$. So with $C_1 = \varnothing$, GREEDY$^+$ exits at Step iii) of iteration 1.

At the end of this review, we note that since our presentation only considers SPP instances where no two paths are equally ranked, Step iii) of GREEDY$^+$ is well defined as there is only one path that has the highest ranking, in each $\bar{\mathcal{P}}^v_{i-1}$. We also note that Step iii) defined above differs from the description in [4], in that it returns the set of *all* nodes that satisfy the condition, rather than just an arbitrary one. But this is equivalent to the original algorithm, by Property 4.2 in [4].

Connecting GREEDY$^+$ to the solvability digraph and GREEDYMIS. Before presenting the key analysis result, we first remark on the key connections between GREEDY$^+$ and GREEDYMIS. Note that the notations used in the following are consistent with those in the description of the algorithms.

1) Step i) of GREEDY$^+$ prunes paths that can no longer be locally optimal, which corresponds exactly to condition (3) in Definition III.4.
2) Step ii) of GREEDY$^+$ prunes paths when they are no longer consistent with the useful set of paths in the current iteration. This is reflected exactly by condition (2) in Definition III.4.
3) At Step iii) of GREEDY$^+$, once an AS has been assigned a path, all other paths from that AS are pruned, the reasoning of which is reflected exactly by condition (1) in Definition III.4.

These connections directly imply the correspondence between the removal of nodes (other than those included in B_i) from the solvability digraph at Line 12 of GREEDYMIS and the pruning of paths from the useful set at Step i), ii), and iii) of GREEDY$^+$. More formally, consider the execution of GREEDY$^+(\mathcal{S}, \bar{\mathcal{P}})$ and GREEDYMIS(G^s), where $\mathcal{S} = (G, \mathcal{P}, \Lambda)$ is a given SPP instance, and $\bar{\mathcal{P}}$ and G^s are the corresponding useful set and solvability digraph, respectively. We have the following lemma.

Lemma IV.4. *After j iterations of* GREEDYMIS *and i iterations of* GREEDY$^+$*, respectively, if $S_j \subseteq P_i$, then for any path $p \in (V(G^s) \setminus (S_j \cup V(G^s_j))$ we have $p \notin \bar{\mathcal{P}}_i$.*

In other words, the Lemma above states that if $S_j \subseteq P_i$, then if a path is removed at Line 12 by GREEDYMIS as a neighbor of some node in $B_t, 1 \leq t \leq j$, it is also pruned by GREEDY$^+$ in one of the first i iterations.

Key analysis result of GREEDYMIS. We show that GREEDYMIS solves strictly more SPP instances than GREEDY$^+$. The strictness follows from the SPP instance DISPUTE-WHEEL-4, which is solvable by GREEDYMIS but not GREEDY$^+$. We now prove that any instance solvable by GREEDY$^+$ is also solvable by GREEDYMIS. Let $\mathcal{S} = (G, \mathcal{P}, \Lambda)$ be an instance of the stable path problem solvable by GREEDY$^+$ with solution π, and let G^s be the corresponding solvability digraph. Let k_1 and k_2 denote the number of iterations it took for GREEDYMIS and GREEDY to complete, respectively, such that GREEDYMIS and GREEDY terminate at iteration $k_1 + 1$ and $k_2 + 1$, respectively. We first give the following lemma (whose proof is omitted due to page limit).

Lemma IV.5. *At each iteration i where $1 \leq i \leq \min(k_1, k_2)$, (1) $P_i \subseteq S_i \subseteq P_{k_2}$, and (2) in* GREEDYMIS*, there exists at least one node in G^s_{i-1} that has no outgoing edges in G^s_{i-1}.*

Notice that Lemma IV.5 implies that at each iteration $i, 1 \leq i \leq \min(k_1, k_2)$, GREEDYMIS constructs B_i by selecting *all and only* nodes in G^s_{i-1} that have no outgoing edges in G^s_{i-1} (Line 4).

Theorem IV.6. *Any instance of the stable path problem solvable by* GREEDY$^+$ *is also solvable by* GREEDYMIS.

Proof. From Lemma IV.5, we have that $k_1 \leq k_2$, thus we can replace $\min(k_1, k_2)$ with k_1 everywhere. At iteration $i = k_1 + 1$ where GREEDYMIS exits, if $|S_{i-1}| = |V(G)|$, then $S_{i-1} \subseteq P_{k_2}$ (by Lemma IV.5) and $|P_{k_2}| = |V(G)|$ together imply that $S_{i-1} = P_{k_2}$, as desired. If this is not the case, then it must be that at iteration $i = k_1 + 1$, $|S_{i-1}| < |V(G)|$ and $|V(G^s_{i-1})| = 0$. This means that there exists some $p \in P_{k_2} \subseteq \bar{\mathcal{P}}_{k_2}$, such that

$p \notin S_{i-1}$. By Lemma IV.5, we have $S_{i-1} \subseteq P_{k_2}$. Given that $|V(G^s_{i-1})| = 0$, we have $V(G^s)\backslash(S_{i-1} \cup V(G^s_{i-1})) = V(G^s)\backslash S_{i-1}$. Thus, $p \notin S_{i-1}$ is equivalent to $p \in V(G^s)\backslash S_{i-1}$, which is equivalent to $V(G^s) \setminus (S_{i-1} \cup V(G^s_{i-1}))$. By Lemma IV.4, $S_{i-1} \subseteq P_{k_2}$ together with $p \in V(G^s) \setminus S_{i-1}$ implies that $p \notin \bar{\mathcal{P}}_{k_2}$, a contradiction. ■

In Theorem IV.6, the solution found by both algorithms will be the same, since by Property 4.4 in [4], any instance solvable by GREEDY$^+$ has a unique solution. We leave it for future work to study how well the heuristics developed for solving the MIS problem on a generic graph can perform on the solvability digraph, compared with GREEDYMIS.

V. USE CASES

In this section, we apply the GREEDYMIS algorithm to design two novel interdomain routing use cases.

A. Use Case 1: A Centralized Interdomain Routing System

This platform leverages the architecture of recent interdomain routing systems [17]–[21] to introduce a centralized route control server, which uses SPP as the foundation for computing interdomain routes. Specifically, the server collects the interdomain network topology as well as the policies of ASes (*i.e.*, both route export and selection policy), translates them into an SPP instance using existing tools [4], [8], and transforms the SPP instance to a solvability digraph. Then the server applies the GREEDYMIS algorithm to compute the MIS in the solvability digraph. If an MIS of size $|V(G)|$ is found, where G is the network graph of the SPP instance, the server sends the corresponding stable path assignment back to the ASes. Otherwise, the server notifies the ASes and ask them to adjust their policies.

A main advantage of this platform is that it greatly expands the set of routing policies ASes can use, and thus improves the flexibility of interdomain routing. The reasons for this are that centralized route computation does not need to worry about convergence in distributed routing protocols, and that the set of BGP policies that correspond to solvable SPP instances is a superset of the set of BGP policies that correspond to convergable SPP instances. As such, this platform is suitable for collaborative interdomain networks (*e.g.*, the Large Hadron Collider [26]), where ASes are willing to expose their policies in exchange for more flexible route control.

B. Use Case 2: Compute Interdomain Routing via SMPC

For interdomain networks where ASes want to achieve more flexible interdomain routing while preserving policy privacy, we develop a secure multi-party computation (SMPC) protocol, called P-GREEDYMIS, which allows ASes to collaboratively use GREEDYMIS to find a stable path assignment without exposing their routing preferences.

Setting. Given an SPP instance $\mathcal{S} = (G, \mathcal{P}, \Lambda)$ with $n+1$ ASes (*i.e.*, $|V(G)| = n+1$) and the corresponding solvability digraph G^s, we use $N = |V(G^s)|$ to denote the number of nodes in the solvability digraph, and index the nodes from 1 to N. In this use case, each AS exposes its export/import policies so that each AS k can compute its permitted paths $\mathcal{P}^k$ while keeping its routing selection policies private. This setting is reasonable in that even if ASes choose to keep their export/import policies private, thee export policies can be accurately inferred [31]–[33], even in the case where they go beyond the Gao-Rexford condition to include sibling/peer+ AS relationship [34], and behaviors of import policies can be observed at BGP Looking glass and RIPE database. In contrast, the routing selection policies are generally considered very hard to infer [35]. In addition, we assume a semi-honest security model, where each AS faithfully executes the protocol, but tries to infer private information of other ASes.

Preliminaries. The SMPC protocol is based on a linear secret sharing scheme [36], where inputs are shared by $n+1$ ASes at the beginning, such that no single AS has the complete knowledge of the solvability digraph. We refer the reader to [36] for details about this secret sharing scheme. Later computation is performed over the secret shares while all intermediate results are also shared by all ASes. We use $[x]$ to denote that value x is secret shared. We also use $[f(x, y)]$ to denote that the ASes perform some computation f over secret shared inputs x and y, and then secret share the result. We have that addition is free and can be locally computed whereas multiplication involves some communication between the ASes. In particular, given $[x]$ and $[y]$, each AS can compute their share of $[x+y]$ using only their share of $[x]$ and $[y]$ and therefore the computation can happen locally. On the other hand, obtaining the secret shared value for $[x * y]$, given $[x]$ and $[y]$, involves the communication of certain values between all the ASes. These values are independent of $[x]$ and $[y]$ and can happen at the initialization phase.

An adjacency matrix representation of solvability digraph. We denote using M the adjacency matrix of the solvability digraph G^s, where the i-th row and column correspond to the i-th node of the solvability digraph. In particular, given an edge $(p_1, p_2) \in G^s$, if (p_1, p_2) satisfy conditions (1) or (3) in Definition III.4, its existence ($M_{p_1p_2} = 1$) is private information known only to AS k, where $p_1 \in \mathcal{P}^k$. Otherwise, it satisfies condition (2) and $M_{p_1p_2} = 1$ can be derived and known by all ASes.

Initialization. With the matrix representation, each entry of M is secret shared by all $n+1$ parties using the linear secret sharing scheme in [36]. This can be easily implemented, because the edges of the solvability digraph by Definition III.4 either are public or can be computed locally by a single party. For the former type of edges the $n+1$ parties secret share 1 for their corresponding entries in M at initialization. The rest of the entries in M can be computed by each AS locally. Indexing each row and column of M by the node in G^s it corresponds to, each AS k computes the p_k-th row of M for each $p_k \in \mathcal{P}^k$ and then shares each entry of the row with the other ASes. During the initialization, ASes also share two binary vectors Z and S of length N. Each entry of Z indicates whether a node has been removed from the solvability digraph, and each entry of S indicates whether a node has been added into the independent set constructed so far. Finally, a number of values related to the number of secret shared multiplications as described earlier, are shared among all ASes.

```
1  P-GreedyMIS([M], n+1, N, {P^k}_{0≤k≤n}):
2    [S] = [[0] * N]
3    [Z] = [[1] * N]
4    for (i in [0, n]):
5      [ind] = [lst] = [d] = [u] = [b] = [[0] * N]
6      for (k in [1, N]):
7        [d_k] = [M_{k,1} + ··· + M_{k,N}]
8        [ind_k] = [d_k == 0]
9      [lst] = [MiniFind(d, lst)]
10     [b_add] = [1]
11     for (k in [1, n]):
12       [b_k] = [( Σ_{p∈P^k} Z_p == 1) * b_add]
13       [b_add] = [b_add * (1 − b_k)]
14       for (j in P^k):
15         [u_j] = [Z_j * b_k]
16         [tmp] = [ind_j]
17         [tmp] = [tmp + (1 − ind_j) * u_j]
18         [tmp] = [tmp + (1 − ind_j) * (1 − u_j) * lst_j]
19         [S_j] = [(1 − S_j) * tmp + S_j]
20     for (k in [1, N]):
21       for (j in [1, N]):
22         [Z_k] = [(1 − S_k) * Z_k]
23         [Z_k] = [(1 − S_j * M_{k,j}) * Z_k]
24         [Z_k] = [(1 − S_j * M_{j,k}) * Z_k]
25     for (k in [1, N]):
26       for (j in [1, N]):
27         [M_{k,j}] = [Z_k * Z_j * M_{k,j}]
28   Reveal [Σ_{i=1}^{N} S_i == n + 1]
```

Fig. 5: P-GREEDYMIS: an MPC protocol for finding independent set based on GREEDYMIS

Protocol. After the initialization, the SMPC protocol takes as input the secret shared matrix M, the number of ASes $n+1$, the number of nodes N of the solvability digraph, and the permitted paths $\{\mathcal{P}^k\}_{0\leq k\leq n}$. During the protocol, the ASes update $[M]$, $[Z]$, and $[S]$ collaboratively at each iteration. The control flow does not reveal private information except the number of iterations, which is bounded by the number of nodes N. Hence we can replace the while loop with a constant number of N rounds to avoid any leakage from the control flow. Moreover, the memory accesses in P-GREEDYMIS are all data-independent, and no private address is accessed, for example, by accessing the i-th element of some vector L when i is a value secret shared by all the ASes. As such, the protocol in Fig. 5 can avoid the use of oblivious RAM, as well as any considerable overhead that would be introduced by them [37].

To update $[M]$, $[Z]$, and $[S]$, all the parties need to construct two secret shared binary vectors $[\mathsf{ind}]$ and $[\mathsf{lst}]$ of size N. The vector $[\mathsf{ind}]$ is used to represent which nodes have out-going neighbors. Using a vector $[d]$ to represent each node's out-going degree, $[\mathsf{ind}]$ can be computed by checking if the entries are 0. The vector $[\mathsf{lst}]$ is such that exactly one of its entries is 1, and the rest are 0. We use $[\mathsf{lst}]$ to represent one of the nodes with the smallest out-going degree. In the case where there are more than one nodes with the smallest out-going degree, we pick the one with the greatest index in the adjacency matrix. Moreover, $[\mathsf{lst}]$ can be computed through tailored sorting networks [38], using $N-1$ comparisons and $4N$ multiplications. After $n+1$ iterations, the $n+1$ parties reveal if $\sum_{i=1}^{N} S_i$ equals $n+1$.

Analysis. Theorem V.1 presents the correctness, privacy, and complexity of our $(n+1)$-party MPC protocol P-GREEDYMIS. We provide the intuition while omitting the details. The privacy claim follows from the security of the secret sharing scheme and the data-independent control flow. We assume that used in our protocol is a t-out-of $(n+1)$ secure secret sharing scheme for the field $\mathbb{F}_q$, for q a large enough constant prime [36]. We also measure the complexity of P-GREEDYMIS by counting the number of multiplication gates, noting that additions can be "free" if the underlying secret sharing scheme is linear.

Theorem V.1. *The following holds true for the $(n+1)$-party MPC protocol* P-GREEDYMIS*:*

- *Privacy: Assuming $N \leq q$ where q is a constant prime and there exists a t-out-of $(n+1)$ secure secret sharing scheme for the field $\mathbb{F}_q$,* P-GREEDYMIS *is t-out-of $(n+1)$ secure and does not reveal any information other than whether* GREEDYMIS *can find an independent set in G^s of size $n+1$ under the semi-honest setting.*
- *Complexity: Assuming there is an $(n+1)$-party protocol `miniFind` that takes an array of length N as input and returns a secret shared binary array $[\mathsf{lst}]$ as described above using $N-1$ comparisons and $4N$ multiplications, then* P-GREEDYMIS *involves $n(N-1)$ comparisons, $n(N+n)+1$ equality tests, and $n(2n+9N+7N^2)$ multiplications.*
- *Correctness:* P-GREEDYMIS *returns true if and only if* GREEDYMIS *finds an independent set of size $n+1$.*

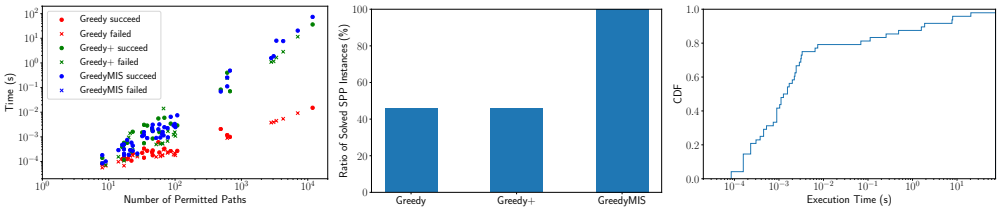
We reason about the correctness of P-GREEDYMIS by checking that it implements each step of GREEDYMIS using a sequence of arithmetic computations. In particular, lines 14-19 update S_j by $ind_j+(1-ind_j)*u_j+(1-ind_j)*(1-u_j)*lst_j$. Noting that S_j indicates whether a node is in the independent set, we have the code snippet implements the heuristic exactly: it first tries to add to the independent set all nodes with no outgoing edges; if such nodes do not exist, it attempts to add to the independent set a node from some $\mathcal{P}^k$ that contains only one node; finally if neither type of nodes exist, it adds a node of lowest out-degree to the independent set. Lines 20-24 assign 0 to Z_j if the node j is a neighbor of some node in the independent set S_j. Lines 25-27 assign 0 to M_{ij} if either node i or j is removed. Conceptually, lines 20- 27 remove from the graph the nodes of the independent set and their neighbors.

VI. PERFORMANCE EVALUATION

We study the benefits and efficiency of the GREEDYMIS algorithm in both use cases using real-world datasets. We introduce the experiment methodology and then the results.

A. Experiment Methodology

Generating SPP instances from real-world datasets. We use the AS-level Internet topology derived from the CAIDA dataset [39] to generate SPP instances of different sizes. In particular, we first extract multiple AS-level topology graphs from subgraphs of the whole Internet AS-level topology inferred from the CAIDA dataset collected on July 1st, 2020. We



(a) Detailed solvability and execution time of GREEDYMIS.

(b) Percentage of SPP instances different algorithms solved.

(c) CDF of the time GREEDYMIS uses to find a stable path solution.

Fig. 6: Solvability and efficiency of GREEDYMIS in centralized interdomain routing computation.

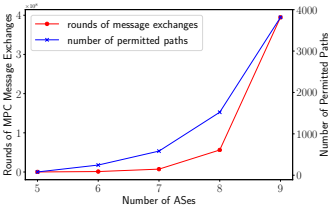


Fig. 7: Rounds of communications performed by P-GREEDYMIS protocol.

use the public registry information of each AS to identify its country. We then extract the maximum connected components of the whole Internet topology containing ASes from the same country. The rationale is that ASes from the same country are usually from the same community or administrated/coordinated by collaborative ISPs, hence they are more willing to collaborate to improve the flexibility of interdomain routing (*e.g.*, sharing policies to a trusted controller).

For each extracted AS-level topology, we identify the customer-provider business relationship between peering ASes using the inferred AS relationship dataset [31], [39]. Based on these business relationships, we configure the import/export policy and the route selection policy of each AS to follow a combination of Gao-Rexford conditions [3]. To add randomness, we also randomly add export policies that do not announce routes containing certain ASes (blackhole policy) or certain AS segments (forbidden segment policy) [17], [35]. We construct the ranking function of each AS as a total order based on the local preference, the AS path length, and the next hop. We simulate the BGP route announcement process without applying route selection policies to generate all permitted paths.We generate SPP instances by AS-level topologies from 48 different countries. The input topology graphs are scaled from 11 ASes to 50 ASes. And the number of permitted paths varies from 8 to 12k.

Comparing with state-of-the-art heuristics. We compare the performance of GREEDYMIS with two state-of-the-art heuristic algorithms for solving SPP, namely GREEDY [1] and GREEDY$^+$ [4]. Comparing GREEDYMIS with exact algorithms (*e.g.*, SAT solving) is an important next step and will be explored in future work.

B. Results

We present the results of GREEDYMIS in both use cases.

Centralized interdomain routing computation. In this use case, we use GREEDYMIS, GREEDY$^+$, and GREEDY each as the algorithm on the controller to find the stable path assignment. We first demonstrate that GREEDYMIS solves strictly more SPP instances than the other two algorithms. Fig. 6a gives the complete result on the solvability and efficiency of all three algorithms, where each point represents a solution status of an SPP instance by a solver. A circle mark means the algorithm found a stable solution successfully and a cross mark represents a failure. Fig. 6b gives the summarized result on solvability. In our experiments, GREEDYMIS finds a solution to all SPP instances, while for over 50% of the SPP instances, both GREEDY and GREEDY$^+$ cannot find a solution.

Next, we show that GREEDYMIS is efficient in finding SPP instances. We find that the execution time of GREEDYMIS over the number of permitted paths is approximately linear (Fig. 6a). Fig. 6c provides the CDF of computation time of GREEDYMIS. Even for the largest SPP instance in our experiment that has 12k permitted paths, the GREEDYMIS can find a solution in 80 seconds. The execution time of GREEDYMIS is close to that of GREEDY$^+$ in most cases, and even faster in some cases.

Privacy-preserving interdomain routing computation. We next evaluate P-GREEDYMIS, the SMPC protocol for ASes to collaboratively solve SPP using GREEDYMIS, without exposing their routing preferences. We construct 5 SPP instances by selecting different numbers of tier-1/tier-2 ASes from the CAIDA dataset and one destination AS. In all 5 instances in the experiment, P-GREEDYMIS can find a solution. We plot the communication overhead of P-GREEDYMIS in terms of the rounds of message exchanges between ASes and the number of permitted paths for different instances in Fig. 7. The results show that when there are 3800 permitted paths in an SPP, the number of messages needed for P-GREEDYMIS is about 4×10^8, which consumes little bandwidth. The growth of communication overhead of P-GREEDYMIS conforms with the increase of the number of permitted paths in SPP instances. This shows the feasibility of P-GREEDYMIS to compute stable path assignments for interdomain routing. We leave more comprehensive evaluations, as well as performance optimizations, as future work.

VII. CONCLUSION

In this paper we studied the important problem of finding a solution for generic SPP instances. We introduced a novel data structure, called a solvability digraph, which encapsulates the possible conflicts between paths in a solution. Our main insight was to transform the SPP to the problem of finding an independent set of size $n+1$ in the solvability digraph, where $n+1$ is the number of ASes. We develop an efficient heuristic algorithm that solves strictly more SPP instances than existing state-of-the-art heuristics. Finally, we applied it to novel interdomain routing use cases, and demonstrated its benefits and efficiency via extensive experiments. We believe that our new insights to this problem will shed light towards better understanding and potential new applications of SPP.

ACKNOWLEDGMENTS

The authors dedicate this paper to Professor Stanley C. Eisenstat, a pillar of the Department of Computer Science at Yale University, who recently passed away. The authors thank the anonymous reviewers for their valuable comments. This research is supported in part by NSFC grant #61702373, NSF awards CCF-1553168 and CNS-1565208, the Office of Naval Research under Grant N00014-17-1-2787 and Facebook Research Award.

REFERENCES

[1] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Transactions On Networking*, vol. 10, no. 2, pp. 232–243, 2002.

[2] L. Cittadini, G. D. Battista, and M. Rimondini, "On the stability of interdomain routing," *ACM Comput. Surv.*, vol. 44, no. 4, Sep. 2012. [Online]. Available: https://doi.org/10.1145/2333112.2333121

[3] L. Gao and J. Rexford, "Stable internet routing without global coordination," *IEEE/ACM Transactions on networking*, vol. 9, no. 6, pp. 681–692, 2001.

[4] L. Cittadini, M. Rimondini, S. Vissicchio, M. Corea, and G. Di Battista, "From theory to practice: Efficiently checking bgp configurations for guaranteed convergence," *IEEE Transactions on Network and Service Management*, vol. 8, no. 4, pp. 387–400, 2011.

[5] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "Policy disputes in path-vector protocols," in *Proceedings. Seventh International Conference on Network Protocols*. IEEE, 1999, pp. 21–30.

[6] J. L. Sobrinho, "Network routing with path vector protocols: Theory and applications," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 49–60.

[7] D. Perouli, T. G. Griffin, O. Maennel, S. Fahmy, C. Pelsser, A. Gurney, and I. Phillips, "Detecting unsafe bgp policies in a flexible world," in *2012 20th IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2012, pp. 1–10.

[8] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 155–168.

[9] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "Control plane compression," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 476–489.

[10] S. Prabhu, K. Y. Chou, A. Kheradmand, B. Godfrey, and M. Caesar, "Plankton: Scalable network configuration verification through model checking," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 953–967.

[11] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Tiramisu: Fast multilayer network verification," in *17th USENIX Symposium on Networked Systems Design and Implementation*, 2020, pp. 201–219.

[12] Y. Rekhter, S. Hares, and D. T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, Jan. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4271.txt

[13] H. Wang, H. Xie, Y. R. Yang, L. E. Li, Y. Liu, and A. Silberschatz, "Stable egress route selection for interdomain traffic engineering: model and analysis," in *13TH IEEE International Conference on Network Protocols (ICNP'05)*, 2005.

[14] A. Fabrikant and C. H. Papadimitriou, "The complexity of game dynamics: Bgp oscillations, sink equilibria, and beyond." in *SODA*, vol. 8. Citeseer, 2008, pp. 844–853.

[15] M. Chiesa, L. Cittadini, G. Di Battista, and S. Vissicchio, "Local transit policies and the complexity of bgp stability testing," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 2957–2965.

[16] S. Vissicchio, L. Cittadini, and G. Di Battista, "On ibgp routing policies," *IEEE/ACM Transactions on Networking*, vol. 23, no. 1, pp. 227–240, 2015.

[17] Q. Xiang, J. Zhang, K. Gao, Y.-s. Lim, F. Le, G. Li, and Y. R. Yang, "Toward optimal software-defined interdomain routing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1529–1538.

[18] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, 2005, pp. 15–28.

[19] K. Lakshminarayanan, I. Stoica, S. Shenker, and J. Rexford, *Routing as a Service*. Computer Science Division, University of California Berkeley, 2004.

[20] S. Pouryousef, L. Gao, and A. Venkataramani, "Towards logically centralized interdomain routing," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 739–757.

[21] Q. Xiang, J. J. Zhang, X. T. Wang, Y. J. Liu, C. Guok, F. Le, J. MacAuley, H. Newman, and Y. R. Yang, "Toward fine-grained, privacy-preserving, efficient multi-domain network resource discovery," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1924–1940, 2019.

[22] Q. Chen, S. Shi, X. Li, C. Qian, and S. Zhong, "Sdn-based privacy preserving cross domain routing," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[23] G. Asharov, D. Demmler, M. Schapira, T. Schneider, G. Segev, S. Shenker, and M. Zohner, "Privacy-preserving interdomain routing at internet scale," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 3, pp. 147–167, 2017.

[24] M. Xiao and H. Nagamochi, "Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs," *Theoretical Computer Science*, vol. 469, pp. 92 – 104, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397512008729

[25] M. Xiao and H. Nagamochi, "Exact algorithms for maximum independent set," *Information and Computation*, vol. 255, pp. 126 – 146, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0890540117300950

[26] "The Large Hadron Collider (LHC) Experiment," https://home.cern/topics/large-hadron-collider.

[27] R. White, D. McPherson, and S. Sangli, *Practical Bgp*. Addison Wesley Longman Publishing Co., Inc., 2004.

[28] P. Lapukhov and A. Premji, "J. mitchell, ed.," use of bgp for routing in large-scale data centers," RFC 7938, DOI 10.17487/RFC7938, August 2016,¡ https://www. rfc-editor. org , Tech. Rep.

[29] H. Wang, H. Xie, Y. R. Yang, L. E. Li, Y. Liu, and A. Silberschatz, "On the stability of rational, heterogeneous interdomain route selection," in *13TH IEEE International Conference on Network Protocols (ICNP'05)*. IEEE, 2005, pp. 11–pp.

[30] X. Shao and L. Gao, "Verifying policy-based routing at internet scale," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2293–2302.

[31] L. Gao, "On inferring autonomous system relationships in the internet," *IEEE/ACM Transactions on networking*, vol. 9, no. 6, pp. 733–745, 2001.

[32] M. Luckie, B. Huffaker, A. Dhamdhere, V. Giotsas, and K. Claffy, "As relationships, customer cones, and validation," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013, pp. 243–256.

[33] Y. Jin, C. Scott, A. Dhamdhere, V. Giotsas, A. Krishnamurthy, and S. Shenker, "Stable and practical {AS} relationship inference with problink," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 581–598.

[34] J. L. Sobrinho, D. Fialho, and P. Mateus, "Stabilizing bgp through distributed elimination of recurrent routing loops," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, 2017, pp. 1–10.

[35] P. Gill, M. Schapira, and S. Goldberg, "A survey of interdomain routing policies," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 1, pp. 28–34, 2013.

[36] R. Cramer, I. Damgård, and U. Maurer, "General secure multi-party computation from any linear secret sharing scheme," Cryptology ePrint Archive, Report 2000/037, 2000, https://eprint.iacr.org/2000/037.

[37] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram: an extremely simple oblivious ram protocol," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 299–310.

[38] M. T. Goodrich, "Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in o(n log n) time," *CoRR*, vol. abs/1403.2777, 2014. [Online]. Available: http://arxiv.org/abs/1403.2777

[39] CAIDA, "As relationships and internet traffic dataset," http://www.caida.org/data/as-relationships/, 2016.