A Special Case of Forbidden-set Policy Routing

Justin Thaler

1. Introduction

The Internet is composed of thousands of interconnected yet independent entities known as Autonomous Systems (ASes), each of which has a unique, well-defined routing policy. Examples of ASes include Internet Service Providers, college campuses, and corporations. The process of establishing connectivity between ASes is known as interdomain routing. As independent and often competing economic entities, ASes must be able to express diverse routing preferences to optimize the routing outcome achieved, but they are not willing to reveal private information about their preferences or internal topology. Moreover, for reasons of trust, scale, and robustness, the Internet does not contain a centralized facility for computing routing outcomes. Therefore, any routing protocol must satisfy the following networking requirements: it must allow routes to be computed efficiently by the ASes themselves in a distributed fashion; it must be guaranteed to produce loop-free routes even when ASes make autonomous decisions about route preference; and it must accommodate changes in the AS graph [1].

There is one more design requirement that we will be concerned with here: convergence. The routing protocol should eventually enter a stable state in which every node prefers its currently chosen route to all others in its routing table, and all routing tables reflect the current route choices of its neighbors. Moreover, we would like the protocol to be robust under link and node failure, converging for every AS graph obtained by removing any set of nodes and links from the original instance [1].

The standard interdomain routing protocol in use today is the Border Gateway Protocol (BGP). Because BGP computes routes for each destination independently, we can consider a particular destination AS d. In BGP, adjacent ASes communicate by sending update messages to announce newly chosen routes to the destination node d. BGP is a path-vector protocol, which means that, when an AS announces a new path, its update messages contain a list of all ASes in the path. This is necessary so that ASes can make autonomous routing decisions and still avoid routes that contain loops.

In BGP, the routing process at a particular node k occurs in three stages. First, k imports routes by receiving update messages from its neighbors and decides which routes to consider; this decision reflects k’s “import policy.” Next, node k selects a route from those that satisfied its import policy. Finally, node k decides which of its neighbors can route to d through it and exports its best route to those neighbors; this decision reflects k’s “export policy.” Whenever there is a change to k’s best route, it sends update messages to the neighbors that satisfy its export policy [2]. AS autonomy is expressed through the choices node k makes in each of the three stages [1]. Note that we cannot assume ASes send update messages in a synchronized manner. This is important because the order in which update messages are sent often affects BGP convergence and makes convergence difficult to analyze [3, 8].

1.1 Previous Work on BGP Convergence Conditions

In contrast to routing protocols like OSPF and IS-IS, which are primarily used to establish connectivity within a single autonomous system, BGP allows for the use of semantically rich routing policies, and this has been instrumental in its success. Unfortunately, this local policy expressiveness brings with it the risk of global routing anomalies. In the absence of local restrictions on policy expressiveness or global restrictions on the topology of the AS graph, BGP does not necessarily converge. As a result, establishing conditions under which BGP is guaranteed to converge, and identifying routing configurations in which it doesn’t, has been the subject of intense research in recent years.

Varadhan et al. [9] were the first to observe that general policy routing could lead to route oscillations and non-deterministic routing. Moreover, they found that divergence is not due to the policies of a single AS alone, but to the complex interaction of the policies of multiple ASes.
Gao and Rexford addressed this problem in [4, 5] by devising reasonable limitations on expressiveness and autonomy (based on the current commercial relationships between ASes) that guarantee robust BGP convergence. However, these constraints are not enforceable and minor changes in business relationships or misconfiguration is enough to lead to routing divergence [10]. Moreover, they may sometimes be too strict. In [10], Feamster et al. give a real-life example from 2001 in which, a tier-1 AS (Verio) most likely did not meet the Gao-Rexford conditions after establishing a new business relationship with another tier-1 AS (AboveNet).

Griffin, Shepherd, and Wilfong [3] provide several examples of AS graphs and route preferences in which BGP does not converge at all or does not converge robustly. Moreover, they establish a sufficient condition called “no dispute-wheel” for the existence of a stable solution. However, “no dispute-wheel” is not a necessary condition for the existence of a stable solution, and Griffin et al. show that, given an AS graph and a set of arbitrary route preferences, it is NP-complete to determine whether a set of stable paths exists; thus a solution requiring global coordination is computationally infeasible.

Most results on finding local conditions that ensure the existence of a stable solution have been negative. Sobrinho [6] and Griffin et al [7] showed that in order to guarantee that the condition “no-dispute-wheel” holds, ASes must be limited to a generalization of lowest-cost routing. Feamster et al. [10] extend the notion of a dispute wheel to consider routing anomalies that can occur as a result of route filtering, the choice to hide available routes from neighboring ASes. They view robust convergence under link and node failure as a special case of filtering: Removing an edge can be achieved if the ASes incident to that edge filter all routes through that edge, and removing a node entails having all ASes filter all routes through that node. They give a special case of a dispute wheel, called a “dispute ring,” that is a necessary condition for robust convergence under filtering and show that, when arbitrary filtering is allowed, ASes must limit themselves to a policy nearly equivalent to shortest path routing to guarantee that no persistent oscillations can occur.

Most of the negative results just described exclude policies that create any possibility of inducing anomalies, but there have been proposals for protocols that dynamically detect and correct policy disputes in a particular network. In [12], Griffin and Wilfong present a safe path-vector protocol using the model established in [3] that detects and suppresses policy-based oscillations. Ee et al. [11] propose a protocol that achieves the same goal with minimal impact, allowing ASes to exercise full autonomy unless the particular policies and network topology cause an oscillation.

It should be noted that the model used in [3] “abstracts away many of the nitty-gritty details of BGP.” In particular, it assumes that there exists a total preference order of routes at each node. In reality, a single AS is often made up of several different routers, and cannot be represented by a single node in the AS graph with a single routing policy, and as a result many commonly used routing policies, such as use of the Multi-Exit Discriminator (MED), make it impossible to say that a given route is always better or worse than another [3]. Therefore, the model in [3] ignores certain anomalies that can arise in practice. Aaron Jaggard and Vijay Ramachandran address this issue by presenting a framework that is able to model more general policies and develop the notion of a “generalized dispute wheel” which extends the concept of a dispute wheel first developed in [3].

In this paper, we will use the model in [3] rather than any generalized model. Although the behavior of BGP is more complex in practice than that our model, all of our results remain valid and provide lower bounds for real-world BGP [16].

1.2 Related Work on Mechanism Design and Subjective-Cost Routing

Recently, researchers have taken a mechanism-design approach to policy routing. Feigenbaum et al. [13] extended the model of [3] by including cardinal preferences instead of just preference orderings. Specifically, each AS assigns a particular value to any given route. The advantage of this framework is that policy oscillations can be avoided by paying ASes to use less valuable paths when necessary. The mechanism-design approach thus seeks to use payments to achieve a stable, globally optimal routing tree.
However, Feigenbaum et al. [13] showed that for arbitrary preferences, finding a globally optimal routing tree is NP-hard, and it is even hard to obtain an approximation to within a factor of $n^{1/4 - \epsilon}$, where $n$ is the number of nodes in the network.

One natural approach to get around this hardness result is to consider restricted classes of policy preferences. For example, Feigenbaum et al. [13] showed that a globally optimal routing tree can be computed in polynomial time when nodes are restricted to next-hop preferences, a very natural class of preferences that can, for example, model the commercial relationships used by Gao and Rexford [4, 5].

But many useful policies cannot be expressed using next-hop preferences. For example, as presented in [15], an AS $i$ might wish to avoid any route that goes through AS $k$, either because it perceives $k$ to be unreliable or because $k$ is a malicious competitor who would like to drop all of $i$’s traffic. This leads to the forbidden-set class of routing policies: For each AS $i$, there is a set of forbidden nodes such that $i$ prefers any route that avoids these nodes over any route that uses a node in $S_i$. We can then ask the following questions: (1) If each node uses a forbidden-set routing policy, will BGP converge to a set of stable paths?, and (2) Can we find a welfare-maximizing routing tree, i.e., a set of confluent routes that maximizes the number of nodes $i$ whose routes do not intersect the sets $S_i$? If the latter optimization problem were tractable, then this class of routing policies would be a candidate for a mechanism-design solution as in [14].

In [15], Feigenbaum, Karger, Mirrokni, and Sami consider a more general class of policies, subjective-cost routing, of which forbidden set routing is a subclass. Each AS $i$ assigns a cost $c_i(k)$ to every other AS $k$, then, the “cost” perceived by AS $k$ for a route $P$ is $\sum_{c_i(k) \in P} c_i(k)$: AS $i$ prefers routes with lower subjective cost. Subjective-cost routing is a natural generalization of lowest-cost routing (in which there is a single objective measure of cost that all ASes agree upon). Note that forbidden-set routing can be formulated in terms of 0-1 subjective costs. Each node assigns its forbidden nodes a subjective cost of 1 and all other nodes a subjective cost of 0. Then, for any AS $i$, any route that avoids $i$’s forbidden nodes is preferred to a route that does not.

[15] shows that, even when restricted to subjective-cost routing, it is still NP-complete to determine whether a stable solution exists and that the hardness result only requires subjective costs in the set $\{0, 1, 2\}$. Moreover, even if all subjective costs are restricted to 0 or 1 (i.e., forbidden-set routing), it is NP-hard to find a globally optimal routing tree, or even to approximate maximum welfare within any factor. Finally, if the subjective costs are restricted to lie in the range $[1, 2]$, the problem of finding a confluent tree with minimum total subjective cost is APX-hard; thus finding a solution that is within a $1 + \epsilon$ factor of optimal is intractable.1

1.3 Results

In this paper, we extend the work of [15] by considering the simplest subset of forbidden-set routing: We restrict nodes to forbidden sets of size at most 1. We refer to this restriction as FS-1. We show that even in this extremely limited instance, it is NP-hard to find a Minimum Subjective-Cost Routing Tree, and therefore this class of routing policies is not a candidate for the mechanism-design approach of [14]. Moreover, we prove that this problem is APX-hard. However, we also show that when nodes are limited to FS-1 policies, every network has a stable solution. This is in stark contrast to the result of [15] that, when restricted to subjective-cost routing with costs in the set $\{0, 1, 2\}$, determining whether a stable solutions exists is NP-complete. We show, however, that not all networks in which nodes are limited to FS-1 policies are safe, meaning that even though one or more stable solutions exist, the protocol might not find such a solution.

The organization of this paper is as follows: In Section 2, we present our hardness result for finding a welfare-maximizing routing tree. Section 3 contains the formalization of the Stable-Paths Problem established in [3] and our proof that, when nodes are limited to forbidden sets of size at most 1, every network has a stable solution. In Section 4 we present an example of an unsafe network in which nodes are limited to FS-1 policies.

1 Note that this explanation of subjective-cost and forbidden-set routing is taken more or less directly from [15].
2. Hardness Results for Subjective-Cost Minimization

We now present our formulation of the general Subjective-Cost Minimization Problem, identical to that provided in [15], before presenting our hardness results for the special case when nodes are limited to FS-1 policies. In the Subjective-Cost Minimization Problem, we assume that for each pair of ASes v and w, the subjective cost $c_v(w)$ is measured in the same unit across all ASes, and we would like to find a routing outcome that minimizes the sum of subjective costs among all nodes in the network. However, because the packets are sent by forwarding, we require the set of routes to form a confluent tree rooted at the destination i.e. if $\pi(v)=vuP$, then $\pi(u)=uP$ for all nodes v. This constraint applies independently to each destination, so we can restrict our attention to a single destination $d$. Thus, we can frame the Subjective-Cost Minimization problem as:

**Subjective-Cost minimization:** We are given a graph $G=(V, E)$, a set of cost functions $\{c_v(.)\}$, and a specific destination $d$. We want to find a set of routes $\{P_v\}$ such that each AS $v$ has a nonempty route $P_v$, and:
1. The routes $\{P_v\}$ form a tree rooted at $d$.
2. Among all such trees, the selected tree minimizes the sum $\sum_v \sum_{w \in P_v} c_v(w)$.

In [15], Feigenbaum et al. showed that when nodes are limited to forbidden-set policies, the Subjective-Cost Minimization Problem is NP-complete. We obtain the stronger result that when nodes are limited to forbidden-set policies with forbidden sets of size at most 1, the problem is still NP-complete. This hardness result is of particular interest because FS-1 policies can be viewed as the simplest case of forbidden-set routing. We refer to the minimum-cost spanning tree problem limited to forbidden sets of size 1 by MSCT-FS1.

We assume that all nodes are limited to FS-1 policies and that each node incurs the same cost for routing through its forbidden node. Formally, we have the following definition:

**MSCT-FS1:** We are given a graph $G=(V, E)$, a specific destination $d$, and a number $k$. We assume that, for each vertex $v$ in $V$, there is a cost function $\{c_v: V \rightarrow \{0, 1\}|v \in V\}$ such that there is at most one node $u$ with $c_v(u)=1$, and for all other nodes $u \neq u$, $c_v(u')=0$. We want to know if there is a set of routes $\{P_v\}$ such that each AS $v$ has a nonempty route $P_v$, and:
1. The routes $\{P_v\}$ form a tree rooted at $d$.
2. The sum $\sum_v \sum_{w \in P_v} c_v(w) \leq k$.

For each node $v$, if there is a node $u$ such that $c_v(u)=1$, we refer to $u$ as $v$’s forbidden node.

Figure 1: The reduction from set cover to MSCT-FS1. Identical to Figure 3 in [15].

**Theorem 1:** MSCT-FS1 is APX-hard.

**Proof:** We first prove the problem is NP-hard and then modify the reduction to prove APX-hardness. To prove NP-hardness, we give a reduction from the set-cover problem to MSCT-FS1. Consider an instance $I$ of the set-cover problem with $n$ elements $\{E_1, E_2, \ldots, E_n\}$, $m$ sets $\{S_1, S_2, \ldots, S_m\}$, and a number $k$. We would like to know if there is a set cover of size $k$ in $I$. We
construct an instance $J = (G, r, \{c_v : V \rightarrow \{0, 1\} | v \in V\}, k)$ of MSCT-FS1 problem as follows: Vertices $s_j$ and $p_i$ for $1 \leq j \leq m$, corresponds to set $S_j$ in $I$. Vertex $e_i$ for $1 \leq i \leq n$, corresponds to the element $E_i$ in $I$. There are two other vertices, the root $r$ and a helper vertex $h$. Thus, $V(G) = \{r, h\} \cup \{s_j, p_i | 1 \leq j \leq m\} \cup \{e_i | 1 \leq i \leq n\}$. Vertex $e_i$ is connected to all vertices $s_j$ such that $E_i \in S_j$ in $I$. There is an edge between $s_j$ and $p_i$, for all $1 \leq j \leq m$, and there is an edge from each $s_j$ to the helper vertex $h$. All vertices $p_i$, for $1 \leq i \leq m$, and vertex $h$ are connected to $r$.

We also set $c_i(h) = 1$, for all $1 \leq i \leq n$ and $c_i(p_i) = 1$. For all other $v$ and $u$, $c_i(u) = 0$. Note that for each vertex $v$ in $G$, there is at most one vertex $u$ such that $c_i(u) = 1$, so this is indeed an instance of MSCT-FS1. Graph $G$ is depicted in Figure 1 above.

We claim that there is a set cover of size $k$ in $I$ if and only if there is tree with total subjective cost $k$ in $J$.

If there is a family of sets $F$ of size $k$ that covers all the elements in $I$, then we can construct the following solution in $J$. If $S_j \in F$ then connect $s_j$ to $p_j$ to $r$. Each $e_i$ is connected to some $S_j \in F$ in the tree; because all elements are covered in the set cover, all vertices $e_i$ will be included in the tree. For any vertex $s_j$ such that $S_j$ is not in $F$, connect $s_j$ to $h$, and finally, connect $h$ to $r$. It is straightforward to check that the subjective cost of this tree is exactly $k$, since nodes $e_1, \ldots, e_n, p_1, \ldots, p_m$, and $h$ each contribute 0 to the total cost, the $s_j$’s not in $F$ (all of whom have been assigned routes through $h$) also contribute 0 to the total cost, and the $k$ $s_j$’s in $F$ that each route through $p_i$, each contribute 1 to the total cost.

Conversely, if there is a tree $T$ of cost $k$ in $J$, then there is a set cover of size $k$ in $I$. First we can assume that all the edges to $(p_i, r)$, $1 \leq i \leq m$ are in $T$, because, if there is a $p_i$ such $(p_i, r)$ is not in the tree then directed edge $(p_i, s_j)$ must be in the tree since $s_j$ is $p_i$’s only other neighbor, and we could add edge $(p_i, r)$ and remove edge $(p_i, s_j)$ from $T$ and maintain or lower the total subjective cost.

We can also assume the edge $(h, r)$ is in $T$, because if $(h, r)$ isn’t in $T$ then edge $(h, s_j)$ must be in $T$ for some $j$, and we could add $(h, r)$ and remove $(h, s_j)$ without increasing the total cost. The reason for this is as follows. Suppose $(h, r)$ is not in $T$ and there exists an $s_j$ that routes through $h$. Then removing edge $(h, s_j)$ and adding edge $(h, r)$ clearly does not cause $s_j$ to incur any additional cost since neither $h$ nor $r$ is $s_j$’s forbidden node. If there exists an $e_i$ that routes through $s_j$, then removing edge $(h, s_j)$ and adding $(h, r)$ does not cause $e_i$ to incur any additional cost either since $h$ is $e_i$’s forbidden node so either way $e_i$ incurs cost 1. Since no other nodes can route through $h$, it follows that adding $(h, r)$ and removing $(h, s_j)$ does not increase the total cost.

Also we can assume that, if there is an edge $(e_i, s_j)$ in $T$, then $(s_j, p_j) \in E(T)$, because otherwise $(s_j, h)$ would be in $E(T)$, and we could remove the edge $(s_j, h)$ and add $(s_j, p_j)$ to $T$ to get another tree with lower or equal subjective cost since $h$ is $e_i$’s forbidden node, and this change would allow $e_i$ to avoid $h$. Knowing these properties of the tree $T$, we can easily construct the set cover of size $k$ from $T$: There must be at most $k$ edges of the form $(s_j, p_j)$ in $T$, and for each $j$ such that $(s_j, p_j)$ is in $T$, include set $S_j$ in the set cover. This must indeed constitute a set cover of size $k$, since each $e_i$ has a path to the root in $T$, so each corresponding element $E_i$ in the set cover problem is included in the set cover. This completes the NP-hardness proof.

To prove APX-hardness, we note that the special case of the Set Cover Problem in which each element occurs at most twice and each set is of size at most 3 is APX-complete [17]. Clearly, then, the general Set Cover Problem is APX-hard. We note that a $1 + \varepsilon$ approximation for the MSCT-FS1 problem gives a $1 + \varepsilon$ approximation for the Set Cover Problem, as we have shown that, given an instance $I$ of the Set Cover Problem, we can construct an instance $J$ of MSCT-FS1 that is polynomial in the size of $I$, such that there is a set cover of size $k$ in $I$ if and only if there is tree with total subjective cost $k$ in $J$. Thus, MSCT-FS1 is APX-hard.

Note that our proof of Theorem 1 is extremely similar to the proof of Theorem 3 in [15].
3. The Stable Paths Problem

We begin this section by presenting our formalization of the Stable-Paths Problem (SPP), closely related to that in [3] and [11]. We represent the network as the undirected AS graph \( G = (V, E) \), where each node \( v \in V \) corresponds to one AS, and each edge \( \{u, v\} \in E \) corresponds to two ASes that are physically connected and share routing information with each other. We ignore issues related to IBGP, and therefore condense all the internal routers of each AS into a single node.

A path \( P \) is either the empty path, or a sequence of distinct nodes \( v_1 v_2 \cdots v_k \) such that \( \{v_i, v_{i+1}\} \in E \). Paths may be concatenated with other nodes or paths: If \( P = v_1 v_2 \cdots v_j Q = v_j v_{j+1} \cdots v_k \), then \( PQ \) is the path \( v_1 v_2 \cdots v_k \). We denote the set of all paths in \( G \) by \( P(G) \) and the set of all paths beginning at node \( v \) by \( P^v \).

A path assignment is a function \( \pi: V \to P(G) \) that assigns to each node \( v \) in \( V \) a valid loop-free path in \( P \). We assume that for all path assignments \( \pi \) of \( G \), \( \pi(d) = (d) \), the trivial path. If \( \pi(v) \) is the empty path, we interpret this to mean that \( v \) has not been assigned a path to the destination.

We assume that for each node \( v \), there is a non-negative integer-valued cost function \( c_v \) defined over \( P^v \) that represents how \( v \) ranks its available paths. We assume that \( c_v \) assigns the empty path the highest cost. This cost function induces a ranking function \( \lambda_v \) defined on \( P^v \) such that \( \lambda_v(P) < \lambda_v(P') \) if and only if \( c_v(P) > c_v(P') \) and \( \lambda_v(P) = \lambda_v(P') \) if and only if \( c_v(P) = c_v(P') \). In the formalization of [3], Griffin et al. make the additional assumption of strictness: If \( P_1 \) and \( P_2 \) are both in \( P^v \) and \( P_1 \neq P_2 \), and \( \lambda_v(P_1) = \lambda_v(P_2) \) then there exists a node \( u \) such that \( P_1 = (v, u)P_1' \) and \( P_2 = (v, u)P_2' \) (the paths and have the same next-hop). This assumption does not make sense in forbidden-set routing, as \( v \) should be free to value two paths equally if they both avoid \( v \)'s forbidden node(s) or if they both do not avoid \( v \)'s forbidden node(s); so we do not assume strictness here.

A path assignment \( \pi \) is consistent if the set of routes forms a confluent tree rooted at the destination i.e. if \( \pi(v) = vuP \), then \( \pi(u) = uP \) for all nodes \( v \) in \( V \). A path assignment is stable if, for every node \( v \), \( \pi(v) \) is \( v \)'s most preferred among the routes available to \( v \). Formally, a stable path assignment \( \pi \) is one in which, for each node \( v \), if \( \pi(v) = vu \pi(u) \) then \( \lambda_v(vu \pi(u)) < \lambda_v(vu \pi(w)) \) for all neighbors \( w \) of \( v \). An instance \((G, d, V(G))\) of the Stable-Paths Problem is solvable if there exists a consistent, stable path assignment.

Griffin et al [3] introduced the Stable Paths Problem as a formal model for BGP and showed that when arbitrary routing policies are allowed, it is NP-complete to determine whether the Stable-Paths Problem has a solution. They also provided a sufficient, but not necessary, condition for the existence of a stable solution, called no dispute wheel. A dispute wheel represents a conflicting set of routing preferences that might not be possible to satisfy simultaneously. Formally, we have the following definition:

**Definition 2**: A dispute wheel is a triple \((U, Q, R)\) where \( U \) is a sequence of distinct pivot nodes \( u_0 u_1 \ldots u_{k-1} \) (all subscripts are modulo k), and \( Q \) and \( R \) are sequences of \( k \) non-empty paths such that 1) at each pivot \( u_i \), there exists a spoke path \( Q_i \) from \( u_i \) to the destination; 2) at each pivot \( u_i \), there exists a rim path \( R_i \) to the next pivot \( u_{i+1} \); 3) each pivot \( u_i \) prefers the path \( R_i Q_{i+1} \) over the path \( Q_i \).

See Figure 2 for a depiction of a dispute wheel. Note that the nodes in a dispute wheel need not be distinct. Figure 3 gives an example in which nodes in the dispute wheel are “repeated.” Griffin et al showed that the absence of a dispute wheel guarantees that BGP converges to a unique solution but that the presence of a dispute wheel does not necessarily guarantee that a stable solution does not exist.

We refer to the Stable-Paths Problem when limited to forbidden-set routing with forbidden sets of size at most 1 as SPP-FS1. Let \( G = (V, E) \), where \( V \) is the set of vertices in \( G \) and \( E \) is the set of edges, and let \( S = (G, d, \{c_v: V(G) \to \{0, 1\} \mid v \in V(G)\}) \) be an instance of SPP-FS1, where \( d \) is the unique destination node. For a node \( v \) in \( G \), we refer to \( v \)'s forbidden node as \( v^* \), and assume \( c_v(v^*) = 1 \) and \( c_v(w) = 0 \) for all \( w \neq v^* \). We further assume that, for each node \( v \), \( v^* \neq d \), as it does not make sense for a node to try to reach a forbidden destination.
We first develop a greedy heuristic presented in [3] that always outputs a stable solution to any SPP instance with arbitrary routing policies if no dispute wheel exists. Suppose \( V' \) is contained in \( V \), such that \( d \in V' \). A partial path assignment \( \pi \) for \( V' \) is a path assignment such that for every \( u \in V' \), every node in \( \pi(u) \) is in \( V' \). The algorithm constructs a sequence of subsets of \( V \), \( \{d\} \supseteq V_0 \supseteq V_1 \supseteq V_2 \ldots \) together with a sequence of partial path assignments \( \pi_0, \pi_1, \pi_2, \ldots \) where each \( \pi_i \) is a partial path assignment for \( V_i \). For each \( \pi_i \) define \( \pi'_i \) to be the path assignment for \( V_i \), where \( \pi'_{i}(u) = \pi_i(u) \) for \( u \in V_i \) and \( \pi'_{i}(u) \) is the empty path for \( u \notin V_i \). Conversely, if we are given a consistent path assignment \( \pi'_i \), some nodes assigned the empty path, we associate with \( \pi'_i \), the partial path assignment \( \pi_i \), obtained by setting \( \pi_i(u) = \pi'_i(u) \) if \( \pi'_i(u) \) is not the empty path. The partial path assignment \( \pi_i \) is stable for \( V' \) if \( \pi'_i \) is stable for each \( u \in V_i \).

If \( u \in V-V_i \), then a path \( P \) is said to be consistent with \( \pi_i \) if it can be written as \( P = P_1(u_1, u_2)P_2 \) where \( P_1 \) is a path in \( V_1 \), \( u_2 \in V_i \), \( P_2 = \pi_i(u_2) \), and \( (u_1, u_2) \in E \). Such a path is called a direct path to \( V_i \) if \( P_1 \) is empty. Let \( D_i \) be the set of nodes \( u \in V-V_i \) that have a direct path to \( V_i \). Without loss of generality, each node has a non-empty path to the origin (because we assume \( G \) is connected) so if \( V-V_i \) is not empty than \( D_i \) is not empty. Let \( H_i \) be the set of nodes \( u \in D_i \) whose highest-ranked path (we specify how to break ties later) consistent with \( \pi_i \) is a direct path (that is, who have a direct path consistent with \( \pi_i \) that avoids \( u^* \) or whose only paths consistent with \( \pi_i \) are direct and do not avoid \( u^* \)). Denote this path as \( B^n_i \). If \( H_i \) is not empty, let \( V_{i+1} = V_i \cup H_i \). Define the partial path assignment \( \pi_{i+1} \) on \( V_{i+1} \) as \( \pi_{i+1}(u) = B^n_i \) if \( u \in H_i \) and \( \pi_{i+1}(u) = \pi_i(u) \) if \( u \in V_i \). This process continues until for some \( k \), 1) \( V = V_k \) or 2) \( V \neq V_k \) and \( H_k \) is empty. In the first case, \( \pi_k \) is clearly a stable assignment and we are done. In the second case we are stuck.

Because we do not require strictness, it is possible for a node \( v \) to value equally paths \( P_1 \) and \( P_2 \), where \( v \)'s next-hop on \( P_1 \) is not equal to \( v \)'s next-hop on \( P_2 \). It will be useful in certain situations to require the greedy heuristic to break such ties in a specific deterministic fashion as follows. Suppose we start with a path assignment \( \pi_0 \) on graph \( G \) that is not stable, and we alter \( \pi_i \) in some way, including resetting the paths of some nodes in \( G \) to the empty path, to obtain a path assignment \( \pi_{i+1} \) that is stable for each node not assigned the empty path. Let \( \pi_{i+1} \) denote the partial path assignment corresponding to path assignment \( \pi_{i+1} \). In what follows, we will no longer draw a distinction between \( \pi_{i+1} \) and \( \pi_{i+1} \).

Suppose we then run the greedy heuristic again on \( G \), this time beginning with the partial path assignment \( \pi_{i+1} \), and we reach a point where node \( v \) in \( G \) needs to break a tie. Specifically, suppose at iteration \( k \) of the greedy heuristic, \( v \) has direct paths \( P_1 \) and \( P_2 \) consistent with \( \pi_{j+k+1} \), \( \lambda_{\pi_i}(P_1) = \lambda_{\pi_i}(P_2) \), and for all other paths \( P \) in \( P' \) consistent with \( \pi_{j+k+1} \), \( \lambda_{\pi_i}(P) < \lambda_{\pi_i}(P_1) \). If \( v \)'s next-hop in \( P_1 \) is the same as \( v \)'s next-hop in \( \pi_i(v) \) (i.e. the same as \( v \)'s next-hop in the original unstable path assignment \( \pi_i \), we would like \( v \) to break the tie by selecting path \( P_1 \). The usefulness of this method will become apparent later. This completes our development of the greedy heuristic.

We now propose a method for dealing with instances in which we get stuck, given that there is at most one dispute wheel in the network. First we formalize the notion of having “at most one dispute wheel” in a network. Formally, given a dispute wheel \( W = (U, \mathbf{Q}, \mathbf{R}) \), we may suppress pivot node \( u_j \) if \( \lambda_{\pi_i}(R_{j+1}Q_{j+1}) \geq \lambda_{\pi_i}(Q_j) \) by removing \( u_j \) and \( Q_j \) from \( U \) and \( \mathbf{Q} \) respectively and redefining \( R_{j+1} \) to be the concatenation \( R_{j+1}R_j \), where \( \lambda_{\pi_i} \) denotes the ranking function for pivot node \( u_{j+1} \). We call any dispute wheel obtained from \( W \) by a (possibly empty) sequence of such operations a subwheel of \( W \). Since \( \lambda_{\pi_i}(R_{j+1}Q_{j+1}) \geq \lambda_{\pi_i}(Q_j) \), the subwheel obtained by suppressing \( u_j \) is itself a dispute wheel. If \( W \) is a subwheel of \( W' \) obtained by a non-empty sequence of suppression operations, we call \( W \) a proper subwheel of \( W' \). We note that when nodes are limited to FS-1 policies and there are no repeated nodes in the dispute wheel, we may suppress any pivot node \( u_j \) because for any \( u_j, u_{j+1} \in \mathbf{Q}_{j+1} \) and \( u_{j+1} \) is not an element of \( R_{j+1}Q_{j+1} \). We now formalize the notion of a maximal dispute wheel.
**Definition 3:** A dispute wheel \( W \) is maximal if there does not exist a dispute wheel \( W' \) such that \( W \) is a proper subwheel of \( W' \).

In what follows, we first show that, if a given network has a unique maximal dispute wheel and nodes are limited to FS-1 policies, then a stable solution always exists, and therefore the SPP instance is always solvable. In fact, we give an algorithm to find such a solution. We then extend this algorithm to the general case, when multiple dispute wheels may exist, and therefore prove that all instances of SPP-FS1 are solvable. This is in sharp contrast to the result by Feigenbaum, Karger, Mirrokni, and Sami in [15] that the Stable-Paths Problem is NP-complete when restricted to subjective cost routing with costs in the set \( \{0, 1, 2\} \).

**Proposition 4:** If we get stuck at iteration \( j \), the nodes in \( D_j \) constitute the pivot nodes of a single dispute wheel \( W \), which by assumption is the subwheel of the unique maximal dispute wheel \( W' \) in the network. Each pivot node of \( W' \) is either a pivot node in \( W \) or a rim node in \( W \).

Proof (similar to Theorem V.3 from [3]): Let \( u_0 \) be any node in \( D_j \) and let \( Q_0 \) be a direct path for \( u_0 \). Note that there must be a path \( P_0 \) consistent with \( V_j \) which has higher rank than \( Q_0 \), or else \( u_0 \) would have chosen the direct path \( Q_0 \). Since \( P_0 \) is consistent with \( V_j \), it has the form \( P_0 = R_0(u_1, v_1)Q_1' \) where \( R_0 \) is a path from \( u_0 \) to \( u_1 \) in \( V-V_j \), \( v_1 \in V_j, \pi_j(v_1) = Q_1' \), and \( (u_1, v_1) \in E \). Note that \( u_1 \in D_j \) since it has the direct path \( Q_1' = (u_1, v_1)Q_1' \) available to it. Since \( H_j \) is empty we can repeat this process with \( u_1 \). If we continue in this manner it is clear that we will eventually form a dispute wheel \( W \) in which each pivot node \( u_i \) is an element of \( D_j \). Since there is only one dispute wheel in the network by assumption, this must be a subwheel of the unique maximal dispute wheel. Moreover, in the construction above, we may select any node in \( D_j \) to be \( u_0 \) and therefore, since there is only one maximal dispute wheel, all the nodes in \( D_j \) must be pivot nodes in \( W \) for otherwise they would form two or more distinct dispute wheels. Because we have assumed that there exists a unique maximal subwheel \( W' \) in the network, we know that \( W \) is a subwheel of \( W' \). Therefore, \( W \) can be obtained from \( W' \) by a finite sequence of suppression operations. Suppose \( W \) is obtained from \( W' \) by \( n \) such suppression operations and let \( W' = W_0, W_1, \ldots, W_n = W \) be the intermediate wheels obtained by applying each suppression operation in order. The \( i \)’th suppression operation effectively turns a pivot node of \( W_{i-1} \) into a rim node of \( W_i \). Thus we see that if \( v \) is a pivot node in \( W' \), \( v \) is either a rim node or a pivot node of \( W \).

A dispute wheel is depicted in Figure 2. Note that since each pivot node \( u_i \) is in \( D_j \) but not \( H_j \) when our greedy heuristic gets stuck at iteration \( j \), \( u_i \) does not prefer its direct path through \( v_i \) and hence \( u_i \)’s forbidden node \( u_i^* \) lies somewhere on the path \( Q_i \).

![Figure 2: Example of a dispute wheel, using the notation from the proof of Proposition 1. For each pivot node \( u_i \), \( Q_i = (u_i, v_i)Q_i' \) and \( u_i \) prefers the path \( R_0(u_i, v_i)Q_i' \) over its direct path \( Q_i \).](image-url)
Figure 3: A dispute wheel in which a rim node \( r \) appears more than once. Node \( u_1^* \) is the forbidden node of \( u_1 \) and node \( u_2^* \) is the forbidden node of \( u_2 \). Therefore, \( u_1 \) prefers the path \( u_1ru_2u_2^*d \) over \( u_1u_1^*d \) and \( u_2 \) prefers path \( u_2ru_1u_1^*d \) over \( u_2u_2^*d \). Node \( r \) appears twice as a rim node on the right when the dispute wheel is presented in “untangle” form.

**Observation 5:** If we get stuck at iteration \( j \), all nodes in \( V-V_j \) cannot avoid the dispute wheel. That is, for each node \( w \in V-V_j \) and for each path \( P \) available to \( w \) and consistent with the partial path assignment \( \pi_j \), \( P \) must contain at least one pivot node \( u_i \) from the dispute wheel.

**Proof:** This follows immediately from the fact that \( D_j \), the set of nodes with direct paths at iteration \( j \), constitutes the pivot nodes of the dispute wheel. Obviously each node in \( V-V_j \) that does not have a direct path at iteration \( j \) must eventually select a path through a node that does have a direct path at iteration \( j \). The proposition follows.

Observation 5 motivates us to propose the following method for dealing with the existence of a dispute wheel: Assign pivot node \( u_0 \) its direct path \( Q_0 \) and have every other pivot node choose routes that go clockwise around the dispute wheel and through the path \( Q_0 \). Then all nodes in \( V-V_j \) that do not have direct paths when we get stuck have no “choice” but to accept a path that traverses \( u_0 \).

Formally, suppose there are \( n \) nodes in \( D_j \), called \( u_1 \ldots u_n \), and let \( W=(D_j, \{Q_i\}, \{R_i\}) \) be a dispute wheel in which each \( u_i, 1<i<=n \), is a pivot node. If a path \( P=v_1v_2\ldots v_n \) then let \( P|(v_i, v_j) \) denote the subpath of \( P \) beginning at \( v_i \) and ending at \( v_j \) and let \( P|v_i \) denote the subpath of \( P \) beginning at \( v_i \) and ending \( v_n \). For example, if \( R_i= u_ir_1r_2\ldots r_mu_{i+1} \), with \( m>=6 \), then \( R_i|(r_3, r_6)= r_3r_4r_5r_6 \) and \( R_i|r_3= r_3r_4r_5\ldots r_mu_{i+1} \). Then if we get stuck at iteration \( j \), we execute the following submethod:

```plaintext
fix_dispute(Path assignment \( \pi_j \)){
\quad \pi_{j+1}(v)= \pi_j(v) \text{ for all } v \text{ in } V_j
\quad \pi_{j+1}(w) \text{ is the empty path initially for all } w \text{ in } V-V_j
\quad \pi_{j+1}(u_0)=Q_0
\quad \text{For}(j=n; j>=1; j--){
\quad \quad \text{For each node } x \text{ in } R_j, \text{ if } x \text{ has not already been assigned a path (i.e. }
\quad \quad \quad \pi_{j+1}(x) \text{ is the empty path), let } y \text{ be the downstream node closest to } x \text{ in } R_j|x \text{ such that }
\quad \quad \quad \pi_{j+1}(y) \text{ is not the empty path. If no such } y \text{ exists, set } \pi_{j+1}(y)= (R_j|x)\pi_{j+1}(u_{i+1}).
\quad \quad \quad \text{Otherwise, set } \pi_{j+1}(y)= (R_j|(x,y))\pi_{j+1}(y).
\quad \quad }
\quad \text{return } \pi_{j+1}
}
```

It is clear that the partial path assignment constructed by \textbf{fix_dispute} is consistent, i.e. it forms a confluent tree, even when the dispute wheel contains repeated nodes. If no nodes are repeated in the dispute wheel, \textbf{fix_dispute} will assign all nodes in \( R_n \), the path through \( u_0 \), assign \( u_n \) the path \( R_n(u_0, v_0)Q_0 \) and continue this process until we assign \( u_1 \) the path \( R_1R_2\ldots R_n(u_0, v_0)Q_0 \).

It follows from Observation 5 that all nodes in \( V-V_j \) must also choose routes that contain \( u_0 \) because the paths of all pivot nodes contain \( u_0 \). If we now run the greedy heuristic again beginning with the partial path assignment \( \pi_{j+1} \), it must run to completion, because by Proposition
all pivot nodes in the dispute maximal dispute wheel $W'$ are either pivot or rim nodes in $W$ and therefore have already been assigned nonempty paths by the method fix_dispute. Getting stuck again would therefore imply the existence of a second dispute wheel distinct from $W'$, a contradiction to our assumption that $W'$ is the unique maximal dispute wheel.

Let $\pi$ be the complete path assignment when the greedy heuristic completes. Because the greedy heuristic never assigns a node $v$ a path through $v^*$ if there is a $v^*$-avoiding path consistent with the partial path assignment at the time, it follows that when the greedy heuristic finishes, if $v$ is not a node in the dispute wheel and $\pi(v) = v\pi(u)$ then $\lambda_v(\pi(v)) = \lambda_u(\pi(u))$ for all neighbors $w$ of $v$.

Therefore, the only challenge with the approach we have developed is the possibility that there exist one or more rim nodes $r$ such that $r^* \in \pi(r)$ and there exists a neighbor $b$ or $r$ such that $\pi(b)$ avoids $r^*$. We will refer to such a situation as a conflict. Formally, two nodes $x$ and $y$ experience a conflict in a path assignment $\pi$ if $x$ and $y$ are neighbors, $x^* \in \pi(x)$, and $x^*$ is not in $\pi(y)$ (we call $\pi(y)$ an $x^*$-avoiding path). Clearly, if a path assignment $\pi$ is conflict-free, then $\pi$ is stable. We now propose a submethod that will correct all conflicts in the path assignment returned by fix_dispute if one or more conflicts exist.

**Fix_conflicts**(Path assignment $\pi_j$) {

While(there exists a rim node $r$ that experiences a conflict in $\pi_j$) {

j+1.

STEP 1: Let $r$ be a rim node that experiences a conflict. $r$ must have a neighbor $b$ that has an $r^*$-avoiding path $P=\pi_i(b)$. Assign $r$ the path $\pi_{j+1}(r)=rb\pi_j(b)$. For all nodes $x$ such that $r\in\pi_j(x)$, assign $x$ the path $\pi_{j+1}(x)=(\pi_j(x)|(x,r))\pi_{j+1}(r)$. For any nodes $v$ not in the dispute wheel such that there is no rim node $r$ with $v\in\pi_j(r)$, set $\pi_{j+1}(v)$ to the empty path. For all other nodes $v$ in $G$, set $v$’s path to $\pi_{j+1}(v)=\pi_j(v)$.

$\pi_{j+1}=\text{greedy_heuristic}(\pi_{j+1});$

Note again that we cannot get stuck because this would imply the existence of a second maximal dispute wheel.

STEP 2) For each rim node $r$ whose next-hop is not in the dispute wheel, let $r'$ denote the rim or pivot node closest to $r$ in $\pi_{j+1}(r)$ and let $g_r$ denote the last node in $\pi_{j+1}(r)|(r, r')$ before $r'$. Then $g_r$ is not in the dispute wheel and all nodes between $r$ and $g_r$ are not in the dispute wheel either.

While(there exists a rim node $r$ such that a node $w$ in $\pi_{j+1}(r, g_r)$ experiences a conflict) {

Let $x$ denote the node closest to $r$ in $\pi_{j+1}(r)|(r, g_r)$ that experiences a conflict with a neighbor $b$. Assign $x$ the path $\pi_{j+2}(x)=xb\pi_{j+1}(b)$ and assign any nodes $y$ other than $x$ in $\pi_{j+1}(r)|(r, x)$ (note this includes node $r$) the path $\pi_{j+2}(y)=(\pi_{j+1}(y)|(y, x))\pi_{j+2}(x)$.

For each node $z$ in the dispute wheel such that $r\in\pi_{j+1}(z)$, assign $z$ the path $\pi_{j+2}(z)=(\pi_{j+1}(z)|(z, r))\pi_{j+2}(r)$.

For any nodes $v$ in the set $\{v\in V : v$ is not in the dispute wheel and there is no rim node $r$ with $v\in\pi_{j+1}(r)\}$, set $\pi_{j+2}(v)$ to the empty path.

For all other nodes $v$ in $G$, set $v$’s path to $\pi_{j+2}(v)=\pi_{j+1}(v)$.}
\[ \pi_{j+2} = \text{greedy heuristic}(\pi_{j+2}); \]
\[ j += 1. \]

}\}

Return \( \pi_j \)
\}

In summary, our algorithm is as follows:

**find_stable_solution** (\( \pi \))
\\{
    Line 1: Initialize \( \pi \) to be the empty partial path assignment
    Line 2: \( \pi = \text{greedy heuristic}(\pi) \). If we do not get stuck, then \( \pi \) is complete and we’re done.
    If we do get stuck:
    Line 3: \( \pi = \text{fix dispute}(\pi) \);
    Line 4: \( \pi = \text{greedy heuristic}(\pi) \);
    Line 5: \( \pi = \text{fix conflicts}(\pi) \);
    Line 6: Return \( \pi \).
\\}

We say that a node \( w \)'s path is **changed at iteration** \( j \) if \( \pi_{j+1}(w) \neq \pi_j(w) \) and we will say that \( w \) **experiences a conflict at iteration** \( j \) if there is a conflict involving \( w \) in \( \pi_j \). Finally, we will say \( w \)'s **next-hop changes at iteration** \( j \) if \( w \)'s next-hop in \( \pi_j(w) \) is not equal to \( w \)'s next-hop in \( \pi_{j+1}(w) \).

**Observation 6**: **fix_conflicts** always maintains a consistent routing tree

**Observation 7**: No pivot node will ever experience a conflict over the course of the algorithm. It follows that the next-hop of a pivot node will never change over the course of **fix_conflicts**.

**Proof**: We first consider the case that \( u_i^* \) is in \( Q_0 \). By **Observation 5**, if we got stuck at iteration \( j \), all nodes in \( V - V_j \) (the nodes that do not have direct paths when we get stuck) have no “choice” but to accept a path that traverses \( u_0 \) and hence also \( Q_0 \). Therefore, if \( u_i^* \) is in \( Q_0 \), \( u_i \) will never have a neighbor that establishes a \( u_i^* \)-avoiding path over the course of the entire algorithm.

We now handle the case that \( u_i^* \) is not in \( Q_0 \). Since \( u_i^* \) is in \( Q_i \) for each pivot node \( u_i \), \( u_i^* \) is in \( V_j \). And it should be clear that at any point in the algorithm, if \( x \) is a node in \( V - V_j \) and \( y \) is a node in \( V_j \) in the path assigned to \( x \), then \( y \) must be an element of \( Q_0 \). Therefore, if \( u_i^* \) is not in \( Q_0 \), then the path assigned to \( u_i \) at any point in the algorithm must avoid \( u_i^* \).

**Observation 8**: If there is a rim node \( r \) such that \( w \in \pi_j(r) \), then \( w \)'s path is changed at a given iteration \( j \) in **fix_conflicts** if and only if \( w \) experiences a conflict itself in \( \pi_j \) that is corrected during iteration \( j \) or if a node downstream of \( w \) in \( \pi_j(w) \) experiences a conflict that is corrected during iteration \( j \). \( w \)'s next-hop in \( \pi_{j+1}(r) \) is different than \( w \)'s next-hop in \( \pi_j(r) \) if and only if \( w \) experienced a conflict that was corrected at iteration \( j \).

**Observation 9**: Suppose \( w \) is a node that is not in the dispute wheel and there does not exist a rim node \( r \) such that \( w \in \pi_j(r) \). If \( w \)'s next-hop changes in iteration \( j \) from node \( b \) to node \( c \), then one of two cases is possible:

1) \( \pi_j(b) \) and \( \pi_{j+1}(b) \) are both non-\( w^* \)-avoiding path and \( \pi_{j+1}(c) \) is a \( w^* \)-avoiding path that was not available during iteration \( j-1 \), or
2) \( \pi_j(b) \) is a \( w^* \)-avoiding path but \( \pi_{j+1}(b) \) is not, while \( \pi_{j+1}(c) \) is.

This observation follows immediately from our tie-breaking method described during our presentation of the greedy heuristic.
If w’s next hop in iteration j is uo, then w cannot experience a conflict in any iteration \( j \geq j \). The reason for this is identical to that in Observation 7: if w* is in \( Q_0 \) then w will never find a neighbor that develops a w*-avoiding path and if w* is not in \( Q_0 \) then \( \pi_j(w) \) is already a w*-avoiding path.

**Proposition 10:** No rim node r will be assigned the same path on two separate occasions. That is, if \( P \neq P' \), there does not exist a rim node r and numbers \( k < k' \) such that \( \pi_k(r) = P \), \( \pi_{k'}(r) = P' \), and \( \pi_{k''}(r) = P \). Since there are only finitely many paths available to each rim node, it follows that the algorithm must terminate.

**Proof:** Suppose there exists a rim node r and numbers \( k < k' \) such that \( \pi_k(r) = P \), \( \pi_{k'}(r) = P' \), and \( \pi_{k''}(r) = P \). Since r was assigned the path \( P \neq P' \) after it had previously been assigned P, there must be at least one rim node (possibly r itself) in P that experienced a conflict that is corrected, or else r’s path would never have changed in the first place by Observation 8. Let v be the node farthest away from r in P that changes its next-hop during some iteration \( j, k < j < k'' \). Since no nodes downstream of v in \( P|v \) change their next-hops between iterations \( k \) and \( k'' \), it follows that no nodes downstream of v change their paths at all between iterations \( k \) and \( k'' \). Thus, if w is a node downstream of v in \( P|v \), w’s path remains \( P|w \) between iteration \( k \) and \( k'' \).

Therefore, if the next-hop of v in \( P|v \) was b, we refer to Observation 9 and note the Case 2 cannot apply because b did not alter its path at any point between iteration \( k \) and \( k'' \). Thus, \( P|v \) must be a non-v*-avoiding path.

We now show v cannot get reassigned path \( P|v \) at any point in the algorithm after it was assigned \( \pi_j(v) \neq P|v \) in iteration \( j \). Suppose v was reassigned the path \( P|v \) in iteration \( j' \), \( j < j' \leq k'' \), i.e. \( \pi_{j'+1}(v) = P|v \) but \( \pi_{j'}(v) = P|v \). Since v’s next hop in \( \pi_{j+1}(v) \) was not b, there must be some iteration \( j'' \), \( j < j'' < j' \), in which v’s next-hop was changed back to b. But this is impossible by Observation 8, because \( P|c \) is a non-v*-avoiding path. Therefore we have shown that it is impossible for r to be reassigned the path P, as v will never reselect the path \( P|v \).

**Proposition 11:** When the algorithm terminates, it will return a stable path assignment \( \pi \).

**Proof:** Consider the last call to the greedy heuristic before the algorithm terminates. It is clear that all nodes whose paths were empty when this last call was made will have stable paths because the greedy heuristic only assigns a node v a non-v*-avoiding path if there are no v*-avoiding paths consistent with the path assignment \( \pi \). Therefore, we only need to show that no nodes with non-empty paths when the last call to the greedy heuristic was made will develop conflicts.

The nodes with non-empty paths immediately before the final call to the greedy heuristic are nodes that are in the dispute wheel, that lie on the path of a node that is in the dispute wheel, or that were assigned paths by the greedy heuristic before we got stuck. It is clear that any nodes that had already been assigned nonempty paths by the greedy heuristic when we got stuck at iteration \( j \) do not experience any conflicts, and pivot nodes do not experience any conflicts by Observation 7. Finally, fix_conflicts will not return until there are no nodes rim nodes r or nodes w in the path of a rim node r that experience a conflict. Therefore we have shown that the algorithm will not terminate until there are no conflicts in the path assignment. By Proposition 10 the algorithm always terminates, so the path assignment is stable.

**Theorem 12:** If an instance of the Stable-Paths Problem S with nodes limited to FS-1 routing policies contains at most 1 maximal dispute wheel, then S is solvable. Moreover, our algorithm will find and return such a stable solution.

It is worth noting that our algorithm will find a stable solution on many instances that contain more than one maximal dispute wheel. Figure 4 provides an extremely simple example of such a situation and Figures 5 and 6 together provide a much more complex example. All we actually require for our algorithm to work as proven above is:
Condition 1: If the greedy heuristic first gets stuck at iteration $j$, all nodes in $D_j$ are involved in a single dispute wheel. This ensures that Observations 5 and 7 hold.

Condition 2: The greedy heuristic never gets stuck after we call $\text{fix\_dispute}$.

These two conditions may be met even when more than one maximal dispute wheel exists, as Figures 4, 5, and 6 demonstrate.

Figure 4: A simple example of a situation in which our algorithm works despite the presence of more than one maximal dispute wheel. The entire graph is depicted in A, with a dispute wheel $W$ indicated in A through labels. Two additional dispute wheels that are clearly not subwheels of $W$ are depicted in B and C.

Suppose the dispute wheel $W$ depicted in A is the dispute wheel selected by $\text{fix\_dispute}$ when the greedy heuristic first gets stuck. All nodes on the dispute wheel are initially assigned the paths in the direction of the arrows by $\text{fix\_dispute}$. In particular, $r$ is assigned the path through $r^*$. Then $r$'s neighbor $b$ is assigned the $r^*$-avoiding path $bcu_0Q_0$ when the greedy heuristic is run again, so $r$ and $b$ are in conflict. To correct this conflict, $\text{fix\_conflict}(\pi)$, in the first iteration of its while loop, assigns $r$ the path $rb\pi(b)$ and assigns $u_1$ the new path consistent with $r$, i.e., $u_1rbcu_0Q_0$. The path assignment is now complete, consistent, and stable and therefore the algorithm terminates.

Figure 5: An extended example in which our algorithm finds a stable solution even though more than one maximal dispute wheel is present. After Line 4 in $\text{find\_stable\_solution}$ executes, all nodes have been assigned the paths indicated by the bold arrows. Note in particular that $e$ was
assigned the path $\pi(e)=egmhuQ_0$ even though this path does not avoid $e^*$ because both paths available to $e$ consistent with $\pi$ (the other being the path through $e$'s neighbor $p$) contained $e^*$.

When the method \texttt{fix_conflicts} is invoked in Line 5, the conflict between $r$ and $b$ must be corrected at STEP 1, so $r$ is assigned the path $r\pi(b)=rbegmhuQ_0$, $c$ is assigned the path $(\pi(c)|(c,r))\pi(r)=crbegmhuQ_0$, $u_1$ is assigned the path $(\pi(u_1)|(u_1,r))\pi(r)=u_1crbegmhuQ_0$, and the paths of $p$, $f$, $k$, $x$, $y$, and $z$ are all assigned the paths in the direction of the arrows a second time, but $p$ selects the path through next-hop $e$ because $c$'s path no longer avoids $p^*=b$. However, this creates a conflict between nodes $p$ and $e$ because $p$'s path now avoids $e^*=g$. The while loop corrects this conflict by assigning $e$ the path $e\pi(p)=epfkhuQ_0$, $b$ the path $b\pi(p)$, $r$ the path $r\pi(p)$, $c$ the path $cr\pi(p)$, and $u_1$ the path $u_1cr\pi(p)$. Note that this path assignment is stable, even though the graph contains the second dispute wheel pictured below (Figure 6), involving pivot nodes $e$ and $p$, which are not rim or pivot nodes in the first dispute wheel.

![Figure 6](image-url)

For the general case, we cannot assume that Conditions 1 and 2 hold. Because Condition 1 does not hold, we can no longer make use of Observation 7, which stated that pivot nodes will never experience conflicts (see Figure 7 for an example in which a pivot node does experience a conflict in the general case). However, it is easy to see that if we explicitly check for conflicts involving pivot nodes in the method \texttt{fix_dispute} just as we did for rim nodes in our original version of the method, then Proposition 10 applies to both rim nodes and pivot nodes. Finally, because Condition 2 does not hold, we need to handle dispute wheels recursively. These are the only two changes we need to make to remove the assumption that there exists a unique maximal dispute wheel.

//find_stable_solution_general takes as a parameter a stable partial path assignment $\pi$ (that is possibly empty) and returns a solution that is stable for all nodes assigned the empty path in $\pi$

\begin{verbatim}
find_stable_solution_general($\pi$) {
  $\pi=$greedy_heuristic($\pi$);
  If we don’t get stuck, return $\pi$. Otherwise:
  $\pi=$fix_dispute($\pi$);
  //handle additional disputes recursively
  $\pi=$find_stable_solution($\pi$);
}
\end{verbatim}
\[ \pi = \text{fix_conflicts_general}(\pi); \]
Return \( \pi; \)

\[ \text{fix_dispute_general}(\text{Path assignment } \pi_j) \{ \]
Let \( W = (\{ u_i \}, \{ Q_i \}, \{ R_i \}) \) be any dispute wheel in which each \( u_i \) is a node in \( D_j \)
\( \pi_j+1(v) = \pi_j(v) \) for all \( v \) in \( V_j \)
\( \pi_j+1(w) \) is the empty path initially for all \( w \) in \( V - V_j \)
\( \pi_j+1(u_0) = Q_0. \)
For \( j=n; j>1; j-- \) \{
For each node \( x \) in \( R_j \), if \( x \) has not already been assigned a path (i.e. \( \pi_j+1(x) \) is the empty path), let \( y \) be the downstream node closest to \( x \) in \( R_j|x \) such that \( \pi_j+1(y) \) is not the empty path. If no such \( y \) exists, set \( \pi_j+1(y) = (R_j|x)\pi_j+1(u_{i+1}). \)
Otherwise, set \( \pi_j+1(y) = (R_j|(x,y))\pi_j+1(y). \)
\}
return \( \pi_j+1 \)
\}

\[ \text{fix_conflicts_general}(\text{Path assignment } \pi_j) \} \]

While(there exists a rim or pivot node \( r \) that experiences a conflict in \( \pi_j) \) \{
\( j = 1 \).

STEP 1: Let \( r \) be a rim or pivot node that experiences a conflict. \( r \) must have a neighbor \( b \) that has an \( r^* \)-avoiding path \( P = \pi_j(b) \). Assign \( r \) the path \( \pi_{j+1}(r) = rb \pi_j(b) \). For all nodes \( x \) such that \( r \in \pi_j(x) \), assign \( x \) the path \( \pi_{j+1}(x) = (\pi_j(x)|(x,r))\pi_{j+1}(r) \). For any nodes \( v \) not in the dispute wheel such that there is no rim node \( r \) with \( v \in \pi_j(r) \), set \( \pi_{j+1}(v) \) to the empty path. For all other nodes \( v \) in \( G \), set \( v \)'s path to \( \pi_{j+1}(v) = \pi_j(v) \).

//handle additional dispute wheels recursively
\( \pi_{j+1} = \text{find_stable_solution}(\pi_{j+1}) \).

STEP 2) For each rim or pivot node \( r \) whose next-hop is not in the dispute wheel, let \( r' \) denote the rim or pivot node closest to \( r \) in \( \pi_{j+1}(r) \) and let \( g_r \) denote the last node in \( \pi_{j+1}(r)|(r, r') \) before \( r' \). Then \( g_r \) is not in the dispute wheel and all nodes between \( r \) and \( g_r \) are not in the dispute wheel either.

While(there exists a rim or pivot node \( r \) such that a node \( w \) in \( \pi_{j+1}(r, g_r) \) experiences a conflict) \{
Let \( x \) denote the node closest to \( r \) in \( \pi_{j+1}(r)|(r, g_r) \) that experiences a conflict with a neighbor \( b \). Assign \( x \) the path \( \pi_{j+2}(x) = xb \pi_{j+1}(b) \) and assign any nodes \( y \) other than \( x \) in \( \pi_{j+1}(r)|(r, x) \) (note this includes node \( r \)) the path \( \pi_{j+2}(y) = (\pi_j(y)|(y, x))\pi_{j+1}(x). \)

For each node \( z \) in the dispute wheel such that \( r \in \pi_{j+1}(z) \), assign \( z \) the path \( \pi_{j+2}(z) = (\pi_{j+1}(z)|(z, r))\pi_{j+2}(r). \)

For any nodes \( v \) in the set \( \{ v \in V_r: v \) is not in the dispute wheel and there is no rim node \( r \) with \( v \in \pi_{j+1}(r) \} \), set \( \pi_{j+2}(v) \) to the empty path.

For all other nodes \( v \) in \( G \), set \( v \)'s path to \( \pi_{j+2}(v) = \pi_{j+1}(v). \).
\[ \pi_{j+1} = \text{find\_stable\_solution}(\pi_{j+1}). \]

\[ j+=1. \]

} Return \( \pi_{j+1} \)

We note that in the case that the algorithm gets stuck at most once, \textbf{find\_stable\_solution\_general} will behave identically to our original algorithm, \textbf{find\_stable\_solution}. Moreover, Observations 6, 8, and 9 apply to our generalized algorithm just as they apply to our original algorithm.

By our discussion before the presentation of our generalized algorithm, Proposition 10 can be restated for the general case as follows:

\textit{Proposition 13:} No rim node or pivot node \( r \) will be assigned the same path on two separate occasions. That is, if \( P \neq P' \), there does not exist a rim node or pivot node \( r \) and numbers \( k<k'<k'' \) such that \( \pi_k(r)=P, \pi_{k'}(r)=P' \), and \( \pi_{k''}(r)=P \). Since there are only finitely many paths available to each rim node, it follows that the algorithm must terminate.

In our analysis of the original algorithm, Condition 2 allowed us to conclude that, once we got stuck and assigned all nodes in the dispute wheel a non-empty path, all subsequent calls to \textbf{greedy\_heuristic}(\pi) would return a path assignment that was consistent with \( \pi \) and stable for all nodes who were assigned the empty path at the time the greedy heuristic was called. We see that each recursive call to \textbf{find\_stable\_solution\_general}(\pi) serves the same purpose: it finds a path assignment consistent with \( \pi \) that is stable for all nodes assigned the empty path in \( \pi \).

Therefore, for a given level of recursion in the general algorithm, Propositions 11 applies, assuming the partial path assignment at the beginning of the level of recursion is stable. In particular, the propositions apply the first time the greedy heuristic gets stuck. Since all nodes assigned non-empty paths when the greedy heuristic first gets stuck will clearly have stable paths, it follows that Propositions 10 and 11 still apply to the generalized algorithm as a whole. Thus, we have the following generalized version of Theorem 12:

\textit{Theorem 14:} Any instance of the Stable-Paths Problem \( S \) with nodes limited to FS-1 routing policies is solvable. Moreover, our algorithm \textbf{find\_stable\_solution\_general} will find and return such a stable solution.

\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{figure7.png}
\caption{An instance in which a pivot node in a dispute wheel might experience a conflict during the execution of \textbf{find\_stable\_solution\_general}. Suppose when the greedy heuristic first gets stuck and \textbf{fix\_dispute} is called, the dispute wheel \( W \) that is chosen is the one consisting only of pivot nodes \( u_0 \) and \( u_1 \), and \( u_0 \) and \( u_1 \) get assigned the paths indicated by the arrows. Then, during the recursive call to \textbf{find\_stable\_solution\_general}, the greedy heuristic gets stuck again because of the dispute wheel involving \( u_2 \) and \( u_3 \), so \textbf{fix\_dispute} assigns \( u_2 \) and \( u_3 \) the paths indicated by the arrows. Then next call to the greedy heuristic assigns \( w \) the path through \( u_2 \), which creates a conflict for \( u_0 \) that must be corrected.}
\end{figure}
4. Safety

The existence of a stable solution does not guarantee that the routing protocol will reach a stable state. To capture this notion, we use the concept of safety, which guarantees that BGP will always reach a stable state regardless of the initial state of the network and order in which ASes send and receive update messages. In considering safety, we must extend the abstraction used in the previous section in order to model the process by which ASes send and receive advertisements and update their own route decisions. Our model will be identical to that used in [10].

In our model, exactly one AS will be activated at each time step. When an AS is activated, it will consider all of its incoming route advertisements and make a routing decision. We assume that such an operation occurs atomically, and as a result, exactly one AS (namely, the activated AS) may change its route at any given time step.

Definition 15: Given a graph $G=(V, E)$, the sequence $v_1, v_2, \ldots$ is a fair activation sequence if each node $v \in V$ appears infinitely often in the sequence.

We now formalize the protocol dynamics in our model, which are identical to those in [10]:

Routing Protocol Dynamics: At time $t-1$, the current path assignment is $\pi_{t-1}$; i.e., each node $v$ has currently selected path $\pi_{t-1}(v)$ to the destination $d$. At time $t$, the following steps occur atomically:

1. A given node $v_t$ is activated.
2. $v_t$ updates its path to be its most preferred path consistent with $\pi_{t-1}$. That is, (a) $\pi_{t-1}(v_t)$ is consistent with $\pi_{t-1}$, and (b) $\lambda_{v_t}(\pi_{t-1}(v_t)) \geq \lambda_{v_t}(P) \forall P \in P^t$ such that $P$ is consistent with $\pi_{t-1}$.
3. All other nodes leave their paths unchanged.

We note that at a given time step $t$, the path assignment $\pi_t$ may be inconsistent. If $\pi_t(v)=vuP$, where $\pi_t(u)\neq P$, this simply means that $v$ has not been activated since $u$ last changed its route decision. Until node $v$ is activated again at a time $t'>t$, $v$ will continue to advertise path $vuP$ despite the inconsistency. When activated at time $t'$, $v$ will realize that its path is inconsistent with $\pi_{t'}$ and choose a new path that is consistent with $\pi_{t'}$. We are now able to formally define safety.

Definition 16: A network is safe if for any initial path assignment $\pi_0$ and fair activation sequence $v_1, v_2, \ldots$ there exists a finite $T$ such that $\pi_{t+1}=\pi_t$ for all $t \geq T$.

We note that because we do not require strictness in our model of FS-1 policy routing, it is trivial to give an example of a network that is not safe in which nodes are limited to FS-1 policies: If there is a node $v$ that is indifferent between two paths $P$ and $P'$ such that $v$’s next-hop in $P$ is different than $v$’s next-hop in $P'$, then, assuming that the paths of all other nodes in the network remain invariant, $v$ can oscillate between path $P$ and $P'$ for eternity. However, we now give an example of a network that is not safe, even though nodes are limited to FS-1 policies and all nodes break ties by maintaining the same next-hop whenever possible. The network is depicted below in Figure 8.
Figure 8: An unsafe network. The arrows indicate the initial path assignment described below.

In the chart below, we present an activation sequence and show the route decision for the node that is activated in each time step. For lack of space, we do not include $u^0_0$, $u^1_1$, or $u^2_2$ in the activation sequence, but we consider their inclusion implicit because activating any of these three nodes will never change their paths because they have no forbidden nodes and all nodes break ties by maintaining the same next-hop whenever possible.

Notice that the initial path assignment is identical to the path assignment after the final time step in the chart, and that each node is activated at least once (actually, exactly twice) in the activation sequence described below. Therefore, if we repeat this activation sequence infinitely many times, we would have a fair activation sequence that would never reach a stable state. This demonstrates that the network depicted in Figure 8 is unsafe, even though all nodes break ties by maintaining the same next-hop whenever possible.

<table>
<thead>
<tr>
<th>Activate</th>
<th>$u_0$</th>
<th>$v_0$</th>
<th>$w_0$</th>
<th>$u_1$</th>
<th>$v_1$</th>
<th>$w_1$</th>
<th>$u_2$</th>
<th>$v_2$</th>
<th>$w_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$v_0 w_0 u_0 Q_1$</td>
<td>$v_0 w_0 u_0 Q_1$</td>
<td>$w_0 u_0 Q_1$</td>
<td>$v_1 w_1 u_1 Q_2$</td>
<td>$w_1 u_1 Q_2$</td>
<td>$Q_2$</td>
<td>$v_2 w_2 u_2 Q_0 W_0$</td>
<td>$w_2 u_2 Q_1$</td>
<td>$Q_1$</td>
</tr>
<tr>
<td>$u_1$</td>
<td>$u_1 v_1 w_1$</td>
<td>$u_1 v_1 w_1$</td>
<td>$u_2 v_1 w_1$</td>
<td>$u_2 v_1 w_1$</td>
<td>$u_2 v_1 w_1$</td>
<td>$u_2 v_1 w_1$</td>
<td>$w_0 u_0 Q_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$u_2$</td>
<td>$w_0 u_0 v_0$</td>
<td>$w_0 u_0 v_0$</td>
<td>$w_0 u_0 v_0$</td>
<td>$w_0 u_0 v_0$</td>
<td>$w_0 u_0 v_0$</td>
<td>$w_0 u_0 v_0$</td>
<td>$w_0 u_0 v_0$</td>
<td>$w_0 u_0 v_0$</td>
<td>$w_0 u_0 v_0$</td>
</tr>
<tr>
<td>$w_0$</td>
<td>$v_0 w_0 u_0 v_0 w_1$</td>
<td>$v_0 w_0 u_0 v_0 w_1$</td>
<td>$v_0 w_0 u_0 v_0 w_1$</td>
<td>$v_0 w_0 u_0 v_0 w_1$</td>
<td>$v_0 w_0 u_0 v_0 w_1$</td>
<td>$v_0 w_0 u_0 v_0 w_1$</td>
<td>$v_0 w_0 u_0 v_0 w_1$</td>
<td>$v_0 w_0 u_0 v_0 w_1$</td>
<td>$v_0 w_0 u_0 v_0 w_1$</td>
</tr>
<tr>
<td>$v_0$</td>
<td>$u_0 v_0 w_0 u_0 v_0$</td>
<td>$u_0 v_0 w_0 u_0 v_0$</td>
<td>$u_0 v_0 w_0 u_0 v_0$</td>
<td>$u_0 v_0 w_0 u_0 v_0$</td>
<td>$u_0 v_0 w_0 u_0 v_0$</td>
<td>$u_0 v_0 w_0 u_0 v_0$</td>
<td>$u_0 v_0 w_0 u_0 v_0$</td>
<td>$u_0 v_0 w_0 u_0 v_0$</td>
<td>$u_0 v_0 w_0 u_0 v_0$</td>
</tr>
<tr>
<td>$w_1$</td>
<td>$v_1 w_1 u_1 Q_2$</td>
<td>$v_1 w_1 u_1 Q_2$</td>
<td>$v_1 w_1 u_1 Q_2$</td>
<td>$v_1 w_1 u_1 Q_2$</td>
<td>$v_1 w_1 u_1 Q_2$</td>
<td>$v_1 w_1 u_1 Q_2$</td>
<td>$v_1 w_1 u_1 Q_2$</td>
<td>$v_1 w_1 u_1 Q_2$</td>
<td>$v_1 w_1 u_1 Q_2$</td>
</tr>
<tr>
<td>$v_1$</td>
<td>$w_1 u_1 Q_1$</td>
<td>$w_1 u_1 Q_1$</td>
<td>$w_1 u_1 Q_1$</td>
<td>$w_1 u_1 Q_1$</td>
<td>$w_1 u_1 Q_1$</td>
<td>$w_1 u_1 Q_1$</td>
<td>$w_1 u_1 Q_1$</td>
<td>$w_1 u_1 Q_1$</td>
<td>$w_1 u_1 Q_1$</td>
</tr>
<tr>
<td>$u_1$</td>
<td>$Q_1$</td>
<td>$Q_1$</td>
<td>$Q_1$</td>
<td>$Q_1$</td>
<td>$Q_1$</td>
<td>$Q_1$</td>
<td>$Q_1$</td>
<td>$Q_1$</td>
<td>$Q_1$</td>
</tr>
</tbody>
</table>
References

\[
\begin{array}{|c|c|c|}
\hline
u_2 & w_0 & Q_2 \\
\hline
w_0 & w_0 & Q_1 \\
\hline
v_0 & v_0 & Q_1 \\
\hline
u_0 & u_0 & w_0 & Q_1 \\
\hline
w_2 & v_2 & w_0 & u_1 & Q_1 \\
\hline
v_2 & w_2 & u_0 & v_0 & Q_1 \\
\hline
w_1 & v_1 & w_1 & u_0 & Q_2 \\
\hline
v_1 & v_1 & w_1 & u_0 & Q_2 \\
\hline
\end{array}
\]