Policy-Preferred Paths in AS-level Internet Topology Graphs

Mehmet Engin Tozal *

Abstract

Using Autonomous System (AS) level Internet topology maps to determine accurate AS-level paths is essential for network diagnosis, performance optimization, security enforcement, business policy management and topology-aware application development. One significant drawback that we have observed in many studies is simplifying the AS-level topology map of the Internet to an undirected graph, and then using the hop distance as a means to find the shortest paths between the ASes. A less significant drawback is restricting the shortest paths to only valley-free paths. Both approaches usually inflate the number of paths between ASes; introduce erroneous paths that do not conform to economic policies; and generate symmetric paths, which in reality is not a rule. As a result, the derived conclusions might be greatly misleading. In this study we introduce a single-destination, policy-preferred path enumeration algorithm which discovers policy consistent paths from all ASes to a destination AS in an edge-labeled, finite, directed graph representing the Internet topology. Considering that our algorithm's run time complexity is the same as Dijkstra's shortest paths algorithm, we believe that the proposed algorithm will notably enhance the future works that leverage AS-to-AS paths in Internet topology graphs.

1 Introduction

The Internet is a highly engineered, large scale complex system which has no central governance. The global communication infrastructure is formed by tens of thousands of autonomous networks connecting various organizations and individuals together. These autonomous networks are owned and operated by a diverse set of organizations including companies, network service providers, cloud providers, telecommunications companies, universities and government agencies all around the world.

A group of networks managed by one or more network operators under a well defined routing policy is called an *Autonomous System (AS)* in the Internet [14]. Autonomous Systems (ASes) are identified by unique AS numbers and they connect to each other in different forms to attain the "global" Internet communication [32]. Individual users, small businesses and ASes located at the edge of the Internet participate in the global infrastructure

*School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70504 USA
metozal@louisiana.edu

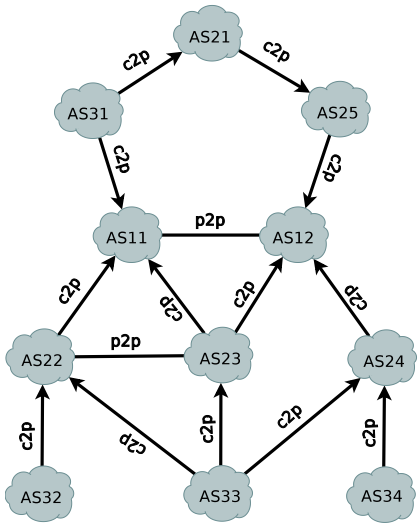


Figure 1: An illustrative AS-level Internet topology map consisting of 11 ASes (clouds) and 15 relations (links) among them.

by means of other ASes called Internet Service Providers (ISPs). Typically, ISPs are business entities providing Internet access service to their customers while getting the same service from one or more upstream ISPs. At the core of the Internet, a small number of ISPs peer with each other through settlement-free interconnections to enable the global communication infrastructure.

ASes in the Internet adhere to the Border Gateway Protocol (BGP) [26] to exchange and propagate inter-AS routing and reachability information via neighbor-to-neighbor BGP advertisements. Essentially, the reachability information consists of an IP address block (routing prefix) in CIDR notation [9], one or more AS paths to reach the routing prefix and a set of AS path attributes. An AS receiving a prefix advertisement, independently decides to employ, drop and/or re-advertise the prefix to its neighbors. These routing decisions largely depend on economic incentives and business relations between ASes rather than performance metrics. Therefore, inter-AS routes in the Internet are typically congruent with the business relations among the ASes [20].

Traditionally, business relations among ASes are categorized as customer-to-provider (c2p), peer-to-peer (p2p) and sibling-to-sibling (s2s) [10]. In a c2p relation, the provider AS provides global Internet access service to the customer AS, in return the customer AS pays to the provider AS for the traffic exchanged between them. To enable the traffic exchange, the provider advertises its full view (all known prefixes including its own prefixes) to the customer and the customer advertises its own prefixes as well as the prefixes that it learned from its customers to the provider AS. For example, in Figure 1 $AS24$ advertises its own prefixes and the prefixes learned from its customers, $AS33$ and $AS34$, to its provider, $AS12$. $AS12$ on the other hand, announces its full view to its customer, $AS24$. In a p2p relation, two peering ASes provide Internet access service limited to each other and each others customer ASes, recursively. Peer ASes typically engage in settlement-free business agreements which means that neither party pays to the other for the traffic exchanged. To enable the traffic exchange, peer ASes advertise their own prefixes as well as the prefixes

that they learned from their customers to each other. For example, in Figure 1 $AS22$ advertises its own prefixes and the prefixes learned from its customers, $AS32$ and $AS33$, to its peer, $AS23$. Similarly, $AS23$ announces its own prefixes and the prefixes learned from its customer, $AS33$, to its peer, $AS22$. In the less frequently observed s2s relation, two ASes provide full reachability to each other because they are operated by the same or sibling organization(s). More complex relations such as hybrid relations where two ASes engage in different relations at different interconnection points and partial relations where a provider AS limits its transit services to its peers and customers are reported in the Internet as well [11]. However, customer-to-provider (c2p) and peer-to-peer (p2p) relations abstract the majority of the business agreements between ASes for practical purposes [20].

In the last two decades researchers have developed many techniques to build AS-level Internet topology maps using various datasets [10, 17, 8, 37, 34, 33]. Employing AS-level Internet topology maps to determine accurate AS-level paths is essential for network diagnosis, performance optimization, service improvement, business policy management and topology-aware application development. To illustrate, inferring AS-level paths allows us (i) to detect congestion points which mostly occur on the links between ISPs [1]; (ii) to control effective neighbor selection in P2P networks [19]; (iii) to leverage the quality of VoIP services and reduce the traffic overhead [27]; (iv) to develop techniques for passive network delay estimation [18]; (v) to optimize server deployment in content delivery networks [13]; (vi) to analyze failures and determine reliability bottlenecks in the Internet [7]; and (vii) to generate synthetic network topologies [2].

On the other hand, a significant drawback in many studies is simplifying the AS-level topology map of the Internet into an undirected graph, and then using the hop distance as a means to find the shortest paths between ASes [2, 19, 27, 5, 13]. Although such an approach facilitates the discovery of shortest paths between ASes for practical purposes, the derived conclusions might be greatly misleading. An AS in fact, may prefer a longer path to reach another AS, if the longer path is economically more advantageous.

A less significant drawback is restricting the shortest paths to only valley-free paths [7, 23, 16, 18]. A valley-free AS path consists of zero or more c2p links followed by zero or one p2p link, and then zero or more p2c links. Although this approach is comparably better, the valley-free property is not solely enough to reveal the policy consistent paths in an AS-level topology graph. An equivalently important property is the AS relation expressed at the first hop link of a path. Assuming that there are multiple valley-free paths from a source AS to a destination AS, the source AS prefers a path starting with an edge oriented to a customer AS compared to an edge oriented to a peer AS. Similarly, the source AS prefers a path starting with an edge oriented to a peer AS compared to an edge oriented to a provider AS. In fact, any intermediate AS on the path makes its decision according to the relation expressed at the first hop link of the sub-path toward the destination. The incentive is again economics: often, a customer pays to a provider and peers do not pay to each other for the exchanged traffic.

As a result, both approaches usually inflate the number of paths between ASes; introduce erroneous paths that do not conform to economic policies; and/or generate symmetric paths, which in reality is not a rule.

In this study we extend our earlier work [31] that introduces a single-destination, policy-preferred path enumeration algorithm which discovers policy consistent paths from all ASes

toward a given destination AS in an AS-level Internet topology graph. We present rigorous analyses on its soundness, completeness, time complexity and space complexity as well as provide a detailed empirical analysis. Our algorithm provides a holistic solution to the AS-level path enumeration problem by incorporating common practices and incentives in the inter-AS routing including shortest-distance preferred paths, valley-free preferred paths and first-hop-edge policy preferred paths. Given an AS-level Internet topology graph and a destination vertex, the algorithm starts from the destination vertex and incrementally builds AS paths in backwards from source vertices toward the destination vertex. At each iteration, a new vertex is joined to the subgraph of the established, policy-preferred paths toward the destination vertex via one or more edges. At the end, the algorithm returns a rooted, directed, acyclic subgraph (r-DAG) of the input graph, which is formed by policy-preferred paths from the source vertices toward the destination vertex. Our algorithm runs in time $O((|V| + |E|)log|V|)$ where $|V|$ is the number of vertices and $|E|$ is the number of edges in the AS-level Internet topology graph. The time complexity of our algorithm is the same as Dijkstra's shortest paths algorithm with a priority queue implementation. Our experimental results on CAIDA's real world dataset [3] show that at least one third of the paths found by the shortest paths algorithm violate AS policy consistency.

We believe that the proposed algorithm will notably affect the future endeavors that leverage AS-level Internet paths for network diagnosis, performance optimization, service improvement, business policy management and topology-aware application development. Yet, the scope of this paper is limited to the study of Policy-Preferred AS paths algorithm as well as its theoretical and empirical analyses.

The rest of the paper is organized as follows. Next section introduces the related work. Section 3 presents an alternative graph model to represent AS-level Internet topology maps. We introduce the definition of the policy-preferred AS paths problem and a solution to the problem in Sections 4 and 5, respectively. Sections 6, 7 and 8 present rigorous proofs for the soundness and completeness of the solution as well as analyses on its time and space complexity. In Section 9 we present an empirical analysis of our solution using real world data. Finally, Section 10 concludes the paper.

2 Related Work

AS-level Internet topology mapping, AS relation inference and AS-level path inference have been active research fields in the last two decades [6, 20, 30, 25].

Many techniques in these fields can be categorized according to the source(s) that they employ in topology mapping and relation inference. Path trace based approaches use `traceroute`-like tools to collect path traces from multiple vantage points and employ IP address to AS number mapping techniques to build the links between ASes. Chang et al. proposed a method for discovering AS-level connectivity by inferring individual connections from the router-level topology of the Internet [4]. Their method involves mapping each router-level node to its AS, and then consolidating the nodes mapped to the same AS into an AS-level node. Mao et al. suggested heuristics to fix inaccurate IP-to-AS mappings by using BGP and `traceroute` paths collected from multiple vantage points [22]. Later, they proposed a new approach based on dynamic programming to iteratively improve IP-to-AS

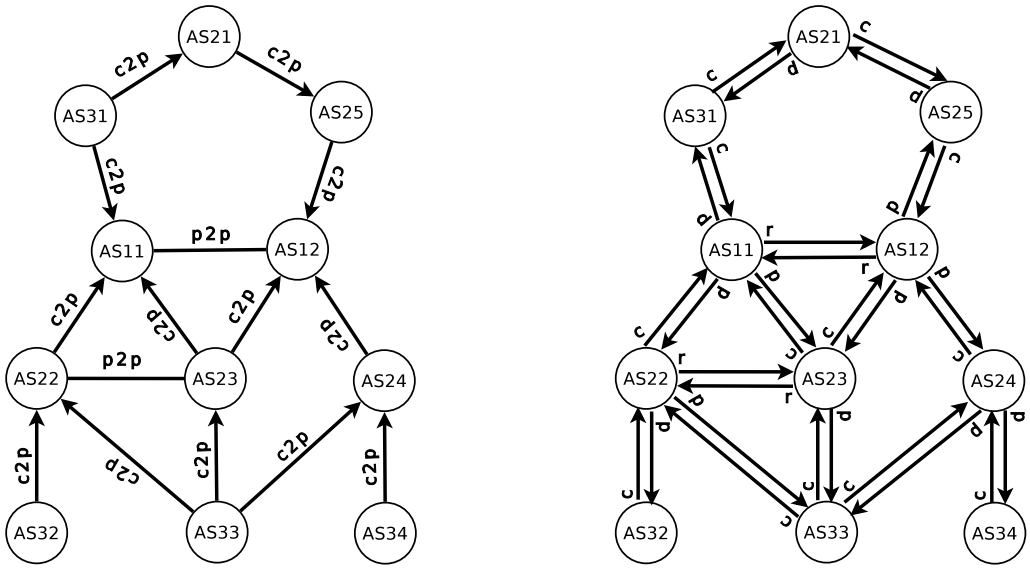
mappings [21]. Chen et al. suggested a set of heuristics for inferring AS-level paths using `traceroute` data collected from peer-to-peer (P2P) users worldwide [4]. More recently, Faggianni et al. showed that using multiple `traceroute` infrastructures helps with improving the completeness of AS-level topology maps [8].

Internet Routing Registry (IRR) databases and BGP looking glasses (LG) are usually used to augment AS-level Internet topologies. He et al. developed an approach which identifies additional AS links by cross-reference and synthesis of BGP routing tables, `traceroute` paths and IRRs [15]. Khan et al. constructed an AS-level Internet topology map using BGP looking glasses [17]. In their work, they were able to identify additional ASes and links that are not reported in BGP based AS-level topologies.

BGP routing table based approaches passively collect BGP updates and use the advertised paths to construct an AS-level topology map of the Internet [35, 34, 33]. Most of the studies in this category focus not only on mapping the Internet at the AS-level but also inferring the types of business relations between the ASes. In her influential work [10], Gao classified business relations between ASes into three groups (c2p, p2p, s2s) based on the assumption that AS-level paths are valley-free, i.e., hierarchical. The algorithm she suggested maximizes the number of valley-free paths by assuming that the high degree ASes at the backbone are connected to each other via p2p links. Later, Xia and Gao proposed an improvement that uses a set of ground truth relations to initiate the inference process [36]. Subramanian et al. used Type of RelationShip (ToR) to formalize Gao's heuristic as a combinatorial optimization problem and suggested a heuristic solution. Their formulation is based on assigning either c2p or p2p to a link such that the number of valley-free paths of a BGP derived graph is maximized. Di Battista et al. proved that the ToR formulation is NP-Complete and fails to capture p2p links and introduced solutions to infer c2p links. Zhang et al. suggested a method based on the assumptions that the ASes at the core of the Internet have large transit degree, form a clique among each other and do not have any providers [37]. Their algorithm first constructs the core AS clique and then assigns c2p relations to these links revealed by the core ASes and assigns p2p to the remaining links. Oliveira et al. suggested a similar approach based on the assumption that a list of core ASes is known [24]. Later, Gregori et al. proposed a similar approach to [24] which identifies the AS relations based on the life span of BGP paths [12]. More recently, Giotsas et al. suggested a new algorithm to infer complex relations between ASes [11]. Their algorithm detects hybrid relations where two ASes engage in different relations at different interconnection points and partial relations where a provider AS limits its transit services to its peers and customers.

Qiu and Gao introduced an algorithm for finding AS paths from any source to a destination prefix [25]. Their algorithm incrementally builds new paths by extending a path set which is directly obtained from BGP routing tables. Mao et al. presented `RouteScope` algorithm which infers AS-level paths between two ASes [23]. Their algorithm restricts the shortest paths to valley-free paths. In a more recent work, Tao et al. suggested a new AS-level path inference method by using hyperbolicity metric assuming that AS-level Internet graphs have low hyperbolicity, i.e., tree-likeness [30].

In this study we propose a novel algorithm to enumerate policy consistent paths in an AS-level Internet topology graph. Our work is unique in terms of the proposed approach and also complementary to existing efforts in the sense that it leverages the Internet topology maps collected, constructed and annotated by the existing projects.



(a) Traditional Internet Graph (b) Alternative Internet Graph

Figure 2: Different graph representations of the example AS-level Internet Topology map given in Figure 1

3 AS-level Internet Graph Representation

Traditionally, an AS-level Internet topology map is modeled as a mixed, finite graph $G = (V, \vec{E}, \bar{E})$ comprising of a set of vertices $V = \{v_1, v_2, v_3, \ldots, v_{|V|}\}$ that represent the autonomous systems; a set of directed edges $\vec{E} \subseteq \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ denoting the customer-to-provider relations; and a set of undirected edges $\bar{E} \subseteq \{(v_k, v_l) \mid v_k, v_l \in V, k \neq l\}$ denoting the peer-to-peer relations between ASes. Another traditional approach is to represent the map as an undirected graph and label the edges by policy relations. Note that AS-level Internet topology graphs abstract the connections between ASes by AS relations which are congruent with inter-AS routing. In our work we found traditional representations restrictive due to the complexity of handling mixed or undirected edges.

In this study we model the AS-level Internet topology map as an edge-labeled, finite, directed graph $G = (V, E, \Phi)$ comprising of a set of vertices $V = \{v_1, v_2, v_3, \ldots, v_{|V|}\}$ that represent the autonomous systems; a set of directed edges $E \subseteq \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ denoting a logical connection from v_i to v_j; and an edge labeling function $\Phi : E \to \mathcal{L} = \{customer, provider, peer\}$ associating a business policy label to an edge (v_i, v_j) which reflects the business relation appointed by v_i between v_i and v_j. To illustrate, an edge labeled as *customer* connects a customer AS to a provider AS, whereas an edge labeled as *provider* connects a provider AS to a customer AS. Note that the function Φ is not necessarily symmetric. For instance, a customer-to-provider relation between vertices v_i and v_j is encoded asymmetrically as $\Phi((v_i, v_j)) = customer$ and $\Phi((v_j, v_i)) = provider$ whereas a peer-to-peer relation between v_i and v_j is encoded symmetrically as $\Phi((v_i, v_j)) = peer$ and $\Phi((v_j, v_i)) = peer$.

Figure 2a shows the traditional representation of the example AS-level Internet topology given in Figure 1. In the figure vertices correspond to ASes and different types of relations

between ASes are represented by mixed edges, e.g., directed edges for customer-to-provider (c2p) connections and undirected edges for peer-to-peer (p2p) connections.

Figure 2b shows our model for the same AS-level Internet topology map given in Figure 1. In the figure vertices correspond to ASes and the relations between ASes are represented only by directed edges. Any edge appearing in Figure 2a is encoded by two directed edges in Figure 2b. Furthermore, each directed edge has an associated policy label which is assigned by the predecessor (first) vertex of the edge. The policy labels simply reflect the role of the predecessor vertex in the relation. For example, the customer-to-provider relation between $AS31$ and $AS21$ is represented by a *customer* (c) edge from $AS31$ to $AS21$ and a *provider* (p) edge from $AS21$ to $AS31$ in Figure 2b. The label of edge $(AS31, AS21)$, *customer* (c), denotes the role of $AS31$ in the relation and the label of edge $(AS21, AS31)$, *provider* (p), denotes the role of $AS21$ in the relation. Similarly, the peer-to-peer relation between $AS11$ and $AS12$ is represented by a *peer* (r) edge from $AS11$ to $AS12$ and another *peer* (r) edge from $AS12$ to $AS11$ in Figure 2b.

In the rest of the manuscript, we develop our arguments using the AS-level Internet topology map representation, $G = (V, E, \Phi)$, demonstrated in Figure 2b.

4 Policy-Preferred AS Paths Problem

Let $G = (V, E, \Phi)$ be an edge-labeled, finite digraph comprising of a set of vertices $V = \{v_1, v_2, v_3, \ldots, v_{|V|}\}$ representing the autonomous systems, a set of labeled edges $E \subseteq \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ denoting a directed edge from v_i to v_j and an edge labeling function $\Phi : E \rightarrow \mathcal{L} = \{customer, provider, peer\}$ associating a policy label to an edge (v_i, v_j) which reflects the business relation appointed by v_i between v_i and v_j. A simple path, P, of hop length k from a source vertex v_s to a destination vertex v_d in G is defined as a sequence of connected, alternating edges which starts at vertex v_s and ends at vertex v_d, i.e., $P = \{(v_s, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_d)\}$ where $P[1][1] = v_s$, $P[k][2] = v_d$, $v_i \neq v_j$ and $P[i][2] = P[i+1][1]$ for $i < k$. We introduce the double square brackets "[][]" operator on a path. The first index of the operator refers to an edge at a particular hop and the second index refers to the predecessor/successor (first/second) vertex of the edge. To illustrate, $P[5][2]$ refers to the successor (second) vertex of the edge located at hop 5 on path P. Note that the edge index of the double brackets operator can inclusively take an integer between 1 and $|P|$ and the vertex index can only take either 1 or 2.

A policy-preferred path, P, in an AS-level Internet topology graph is a simple path which conforms to the following four assumptions that exist in practice [23]. Although deviations from these assumptions are reported [11, 29], they capture the most common cases in practice [20].

Explicit Business Relations: Two ASes connected to each other form a business agreement which is typically consistent with their routing policies [10, 6, 20].

Valley-free AS Path: A simple path is called *valley-free* if the path consists of zero or more *customer* edges followed by zero or one *peer* edge, and then zero or more *provider* edges. A valley-free path is congruent with the business agreements among ASes. That is, customer ASes get reachability service from their provider ASes and peer ASes provide mutual reachability only to each other and their customers, recursively. In Figure 2b the

simple path $P = \{(AS32, AS22), (AS22, AS11), (AS11, AS12), (AS12, AS24)\}$ is a valley-free path from $AS32$ to $AS24$. However, the path $P = \{(AS32, AS22), (AS22, AS33), (AS33, AS24)\}$ is not valley-free because it makes a valley via the edges $(AS22, AS33)$ and $(AS33, AS24)$. Technically, $AS33$ does not re-advertise the routes that it learned from one of its providers, $AS24$, to another provider, $AS22$, because it avoids to relay the traffic between its providers. Similarly, the simple path $P = \{(AS32, AS22), (AS22, AS23), (AS23, AS12), (AS12, AS24)\}$ is not valley-free because it makes a plateau via the edges $(AS22, AS23)$ and $(AS23, AS12)$. On a similar note, $AS23$ does not re-advertise the routes that it learned from its provider, $AS12$, to its peer, $AS22$, because it avoids to relay the traffic between its provider and peer.

First-Hop-Edge Label Preferred AS Path: If there are multiple valley-free paths between a source and a destination AS, the source AS prefers one or more of these paths over the others according to the policy labels (*customer*, *provider*, *peer*) of the first hop edges. Specifically, a path starting with a *provider* edge is preferred over a path starting with a *customer* edge because a provider AS usually charges its customer AS for the traffic exchanged. A path starting with a *peer* edge is preferred over a path starting with a *customer* edge because peer ASes typically engage in settlement-free business agreements. On the other hand, a customer AS pays to its provider AS for the traffic exchanged. A path starting with a *provider* edge is preferred over a path starting with a *peer* edge because while peer relations are usually settlement-free, a provider AS charges its customer AS for the traffic exchanged. Briefly, one or more valley-free paths are preferred among multiple valley-free paths according to the policy labels of their first hop edges. Moreover, the set of edge policy labels $\mathcal{L} = \{customer, provider, peer\}$ is linearly ordered according to their preferabilities.

Definition 4.1. *"$\succ$" (succeeds or higher than). The set of edge labels $\mathcal{L} = \{$customer, provider, peer$\}$ is a linearly ordered set under strict total order relation "$\succ$" (succeeds or higher than) where provider $\succ$ peer $\succ$ customer.*

To illustrate, in Figure 2b there are two valley-free paths from $AS12$ to $AS31$ namely, $P_1 = \{(AS12, AS25), (AS25, AS21), (AS21, AS31)\}$ and $P_2 = \{(AS12, AS11), (AS11, AS31)\}$. P_1 starts with a *provider* edge, $\Phi(P_1[1]) = p$, and P_2 starts with a *peer* edge, $\Phi(P_2[1]) = r$. $AS12$ prefers P_1 over P_2 because $\Phi(P_1[1]) \succ \Phi(P_2[1])$. The incentive behind $AS12$ preferring P_1 over P_2 is that the customer AS, $AS25$, on P_1 pays for the traffic exchanged, whereas it has a settlement-free agreement with the peer AS, $AS11$, on P_2. Note that although P_1 is longer than P_2 in terms of hop distances, $AS12$ prefers P_1 over P_2 for economical reasons.

Shortest Hop Distance Preferred AS Path: If there are multiple valley-free paths starting with the same edge policy label from a source AS to a destination AS, the source AS prefers the path with the shortest length in terms of the hop distance. In Figure 2b there are two valley-free paths from $AS31$ to $AS12$ namely, $P_1 = \{(AS31, AS21), (AS21, AS25), (AS25, AS12)\}$ and $P_2 = \{(AS31, AS11), (AS11, AS12)\}$ where $\Phi(P_1[1]) = \Phi(P_2[1]) = c$. Since both paths start with the same edge policy label, c, $AS31$ prefers P_2 over P_1 because both providers $AS21 \in P_1$ and $AS11 \in P_2$ charge $AS31$ for the traffic exchanged, yet P_2 is shorter than P_1 in terms of hop distance, i.e., $|P_1| < |P_2|$. *Note that since the details of business agreements between ASes are confidential, it is common to assume that the cost of sending traffic through provider AS $AS21$ and provider AS $AS11$ are the same.* Another important observation is that the preferred paths between ASes in an AS-level

Internet topology graph is not necessarily symmetric as $\{(AS31, AS11), (AS11, AS12)\}$ is the preferred path from $AS31$ to $AS12$ and $\{(AS12, AS25), (AS25, AS21), (AS21, AS31)\}$ is the preferred path from $AS12$ to $AS31$.

5 Solution to the Policy-Preferred AS Paths Problem

In the following, we first present an overview of the Policy-Preferred AS Path Enumeration Algorithm using illustrations. Then, we provide the pseudocodes of the main algorithm as well as the procedure to enforce the valley-free path property.

5.1 Algorithm Overview

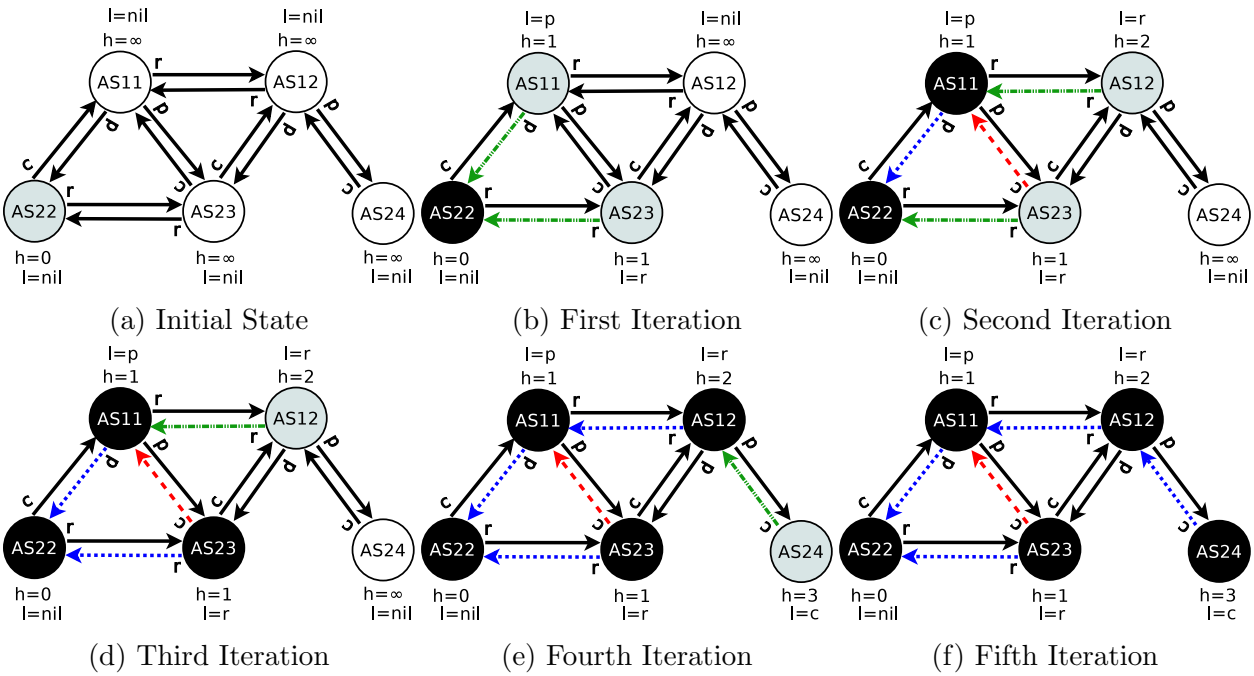
In this part we present an overview of the single-destination policy-preferred AS path enumeration algorithm which *finds all policy consistent paths to a single destination vertex from every source vertex in an AS-level Internet topology graph.*

The algorithm distinguishes the vertices into three disjoint colors. Black vertices are fully explored vertices that already have one or more policy-preferred paths toward the destination vertex. Gray vertices are discovered but not fully explored vertices. White vertices are undiscovered vertices, yet. Initially only the destination vertex is gray and the remaining vertices are white. Additionally, the algorithm tracks two variables for each vertex: `join label` and `join hop-distance`. The `join label` of a vertex reflects the policy label of the tentative edge(s) connecting the vertex to the subgraph of the established, policy-preferred paths. The `join hop-distance` of a vertex reflects the vertex's tentative hop distance to the destination.

The algorithm starts from a designated destination vertex and incrementally builds paths in backwards from source vertices toward the destination vertex. At each iteration, the most preferable gray vertex (highest join edge label and shortest join hop distance) is joined to the subgraph of the established, policy-preferred paths toward the destination vertex via one or more edges. When a gray vertex is selected, its incoming edges that preserve the valley-free property and induce on a gray or white vertex are relaxed. That is, the join edge labels and join hop distances of the white and gray neighbors of the selected vertex are updated if a higher edge policy label or a shorter distance is revealed via the selected vertex. Finally, the selected vertex is colored black to denote its full exploration.

Figure 3 illustrates the workings of our algorithm using a simple AS-level Internet topology graph extracted from the example graph given in Figure 2b. The algorithm starts from the destination vertex, $AS22$, and incrementally builds policy consistent paths in backwards from any source vertex toward the destination vertex. In the beginning (Figure 3a), all vertices but the destination vertex have a join hop-distance infinity, $h = \infty$, a void join label, $l = nil$, and colored white. The destination vertex is colored gray and has zero join-hop distance to itself.

The algorithm starts with selecting the destination vertex which is the only gray vertex and the most preferable among all vertices, i.e., equal join label but shorter hop distance (Figure 3b). At this step all incoming edges, $(AS11, AS22)$ and $(AS23, AS22)$, of the destination vertex are relaxed and the join labels and hop-distances of the neighboring vertices are updated according to the taken edge. The join hop-distances, h, of $AS11$ and $AS23$



(a) Initial State (b) First Iteration (c) Second Iteration

(d) Third Iteration (e) Fourth Iteration (f) Fifth Iteration

Figure 3: An illustrative overview of the Policy-Preferred AS Path Enumeration Algorithm. The algorithm gradually builds policy-preferred paths toward the destination vertex $AS22$ from all vertices in backwards. White vertices are undiscovered vertices, gray vertices are discovered but not fully explored vertices and black vertices are fully explored vertices. Green (dashed-dotted) edges are candidate preferable edges, blue (dotted) edges are established, preferred edges toward the destination vertex and red (dashed) edges are ineffective relaxations. Latest join labels and join hop-distances are shown with l and h, respectively.

are replaced by 1 because they can reach to the destination vertex by one hop. The new join hop-distances are calculated by adding one to the join hop-distance of the selected, gray vertex. The join label, l, of $AS11$ is replaced by *provider* (p) because the taken edge, $(AS11, AS22)$, has policy label *provider*. Similarly, the join label of $AS23$ is replaced by *peer* (r). The tentatively preferable edges toward the destination vertex are shown in green (dashed-dotted).

In the second iteration (Figure 3c) vertex $AS11$ is selected because it is the most preferable (highest join edge label) vertex among all gray vertices. The preferable edge connecting $AS11$ to $AS22$ becomes an established, preferred edge, shown in blue (dotted), toward the destination vertex from $AS11$. At this step all incoming edges of the selected vertex, $AS11$, that form a valley-free path are relaxed. The join hop-distance and join label of the neighboring vertex $AS12$ is updated because the algorithm found a more preferable edge from $AS12$ toward the destination vertex. On the other hand, relaxing edge $(AS23, AS11)$ remains ineffective because $AS23$ has a more preferable join label, *peer* (r), than the suggested one in the relaxation, *customer* (c). The edge involved in the ineffective relaxation is shown in red (dashed).

In the third iteration (Figure 3d) vertex $AS23$ is selected because it is the most preferable (equal join edge label but lower hop distance) vertex among all gray vertices. The preferable

edge connecting $AS23$ to $AS22$ becomes an established, preferred edge toward the destination vertex, shown in blue (dotted). At this step none of the incoming edges to the selected vertex, $AS23$, is relaxed. Edge $(AS11, AS23)$ is not relaxed because the incident vertex $AS11$ is black, i.e., it already has an established, preferred edge toward the destination vertex. Edge $(AS12, AS23)$ is not relaxed because taking edge $(AS12, AS23)$ from $AS12$, and then taking the newly established, preferable edge $(AS23, AS22)$ violates the valley-free AS path property. That is, the path $P = \{(AS12, AS23), (AS23, AS22)\}$ makes a plateau by connecting a *provider* edge to a *peer* edge and it is not allowed by the algorithm.

In the fourth iteration (Figure 3e) vertex $AS12$ is selected because it is the only hence, the most preferable gray vertex. The preferable edge connecting $AS12$ to $AS11$ becomes an established, preferred edge toward the destination vertex, shown in blue (dotted). At this step incoming edge $(AS24, AS12)$ of the selected vertex, $AS12$, is relaxed. Moreover, the join label and the hop distance of vertex $AS24$ is updated accordingly. However, incoming edge $(AS23, AS12)$ is not relaxed because the incident vertex $AS23$ is black, i.e., already explored.

In the last iteration (Figure 3f) vertex $AS24$ is selected because it is the only therefore, the most preferable gray vertex. The preferable edge connecting $AS24$ to $AS12$ becomes an established, preferred edge toward the destination vertex, shown in blue (dotted). At the end of this step our algorithm completes processing all vertices. The subgraph shown in blue (dotted) comprises all policy-preferred paths from any vertex to the destination vertex, $AS22$. In Section 5.2 we prove that the output of the policy-preferred AS path enumeration algorithm is, a rooted, directed, acyclic graph (r-DAG).

The core of the illustrated algorithm lies in the step which selects the most preferable vertex among all gray vertices. At each iteration, the decision is made according to both join edge labels and join hop-distances of the gray vertices. The set of join labels, $\mathcal{L} = \{customer, provider, peer\}$, is an ordered set under strict total order relation "$\succ$" (succeeds or higher than) where $provider \succ peer \succ customer$ (Definition 4.1). The set of join hop-distances, $\mathbb{Z}_{\geq 0}$, is also a linearly ordered set under the usual "$<$" (less than) relation. As a result, a new relation "$\succ^*$" (more preferable) which is defined on the Cartesian product of $\mathcal{L}$ and $\mathbb{Z}_{\geq 0}$ is a strict total order relation.

Definition 5.1. *"$\succ^*$" (more preferable). A vertex with join label l and join hop-distance h is more preferable compared to another vertex with join label l' and join hop-distance h' if and only if l succeeds l' or l is equal to l' and h is less than h'. That is, $(l, h) \succ^* (l', h') \iff (l \succ l') \vee (l = l' \wedge h < h')$ where $l, l' \in \mathcal{L}$ and $h, h' \in \mathbb{Z}_{\geq 0}$.*

5.2 Policy-Preferred AS Path Enumeration Pseudocode

Our implementation of the single-destination, policy-preferred AS path enumeration algorithm involves the following function definitions and data structures:

`in_neighbor`($\mathtt{v_i}$): Provides an unordered list of vertices that have an incoming edge to vertex v_i.

$\Phi((\mathtt{v_i}, \mathtt{v_j}))$: Provides the v_i appointed edge policy label for the edge (v_i, v_j).

`color`($\mathtt{v_i}$): Provides the track of progress in the algorithm for vertex v_i. Specifically, *white* means undiscovered vertex, *gray* means discovered but not fully explored vertex and *black*

Algorithm 1 Policy-Preferred AS Path Enumeration

Input: $G = (V, E, \Phi)$ ▷ AS-level Internet topology graph
Input: v_d ▷ destination vertex, $v_d \in V$
Output: π ▷ list of successor vertices towards v_d for each vertex

▷ Initialize all vertices
for all $v_i \in V$ **do**
    $color(v_i) \leftarrow white$
    $join\_edge\_label(v_i) \leftarrow nil$
    $join\_hop\_dist(v_i) \leftarrow \infty$
end for

▷ Initialize the destination vertex v_d
$color(v_d) \leftarrow gray$
$join\_hop\_dist(v_d) \leftarrow 0$

$Q(\succ^*)$ ▷ A priority queue with "more preferable" relation $\succ^*$
$Q.push(v_d)$

while Q is not empty **do**
    $v_s \leftarrow Q.pop()$ ▷ selects the most preferable vertex in Q
    for all $v_j \in in\_neighbor(v_s)$ **do**
        if $is\_valley\_free(\Phi((v_j, v_s)), join\_edge\_label(v_s))$ **then**
            if $color(v_j)$ is $white$ **then**
                $join\_edge\_label(v_j) \leftarrow \Phi((v_j, v_s))$
                $join\_hop\_dist(v_j) \leftarrow join\_hop\_dist(v_s) + 1$
                $\pi(v_j).insert(v_s)$
                $color(v_j) \leftarrow gray$
                $Q.push(v_j)$
            else if $color(v_j)$ is $gray$ **then**
                if $\Phi((v_j, v_s)) \succ join\_edge\_label(v_j)$ **then**
                    $join\_edge\_label(v_j) \leftarrow \Phi((v_j, v_s))$
                    $join\_hop\_dist(v_j) \leftarrow join\_hop\_dist(v_s) + 1$
                    $\pi(v_j).clear()$
                    $\pi(v_j).insert(v_s)$
                else if $\Phi((v_j, v_s)) = join\_edge\_label(v_j)$ **then**
                    if $join\_hop\_dist(v_j) > join\_hop\_dist(v_s) + 1$ **then**
                        $join\_hop\_dist(v_j) \leftarrow join\_hop\_dist(v_s) + 1$
                        $\pi(v_j).clear()$
                        $\pi(v_j).insert(v_s)$
                    else if $join\_hop\_dist(v_j) = join\_hop\_dist(v_s) + 1$ **then**
                        $\pi(v_j).insert(v_s)$
                    end if
                end if
            end if
        end if
    end for
    $color(v_s) \leftarrow black$
end while

return π

means fully explored vertex.
`join_edge_label`($\mathtt{v_i}$) and `join_hop_dist`($\mathtt{v_i}$): Respectively provide the preferable join edge label and join hop-distance of vertex v_i during the course of the algorithm. Both join edge label and join hop-distance are subject to change after edge relaxations for white and gray vertices whereas, they are final for black vertices.
$\mathtt{Q}(\succ^*)$: A priority queue holding the gray vertices according to the "$\succ^*$" (more preferable) relation given in Definition 5.1. The priority queue, Q, supports `push` and `pop` operations to insert a new element and retrieve the top element, respectively.

The policy-preferred AS path enumeration algorithm expects an AS level Internet topology graph, $G = (V, E, \Phi)$, and a destination vertex, $v_d \in V$, as input. The output of the

algorithm is a list of successor vertices, π, toward the destination vertex v_d for each vertex in the topology graph. Note that the algorithm computes the policy consistent AS paths. Printing the paths requires starting from a source vertex and following the list of successor vertices until reaching the destination vertex.

Lines 2-6 in Algorithm 1 initializes the vertices by setting their colors to white, join labels to *nil* and join hop-distances to *infinity*. Lines 9 and 10 set up the destination vertex, v_d, by setting its color to gray and its join hop-distance to zero, respectively. Line 12 declares a priority queue of gray vertices ordered by the "$\succ^*$" (more preferable) relation and line 13 inserts the destination vertex v_d into the priority queue.

The algorithm processes the vertices of the topology graph until the loop at line 15 ends. At each iteration the most preferable, gray vertex in the priority queue is selected (line 16). If there is a tie, it is broken arbitrarily. Then, the incoming edges of the selected vertex are relaxed in a for-loop (line 17) with the condition that joining the edge to the selected vertex does not violate the valley-free property (line 18). The details of the valley-free validation procedure is presented in the next sub-section.

While relaxing the incoming edges of the selected vertex if the neighboring vertex is unexplored, i.e. it is white, then its join label is updated with the edge label of the relaxed edge; its join hop-distance toward the destination is updated by adding one to the join hop-distance of the selected vertex; its successor list is updated by the selected vertex; its color is set to gray; and finally it is added to the priority queue as shown at lines 20-24.

If the neighboring vertex is not fully explored yet, i.e. it is gray, and the relaxed edge's policy label succeeds the neighboring vertex's current join label, then a more preferable edge is found to connect the neighboring vertex to the subgraph of the established, policy-preferred paths toward the destination vertex. Hence, the join label of the neighboring vertex is replaced by the label of the relaxed edge; its join hop-distance is updated according to the join hop-distance of the selected vertex; its successor list is cleared and the selected vertex is added to the successor list as shown at lines 27-30.

On the other hand, if the relaxed edge's policy label is equal to the neighboring vertex's current join label and joining the neighboring vertex through the relaxed edge has a shorter hop distance then, again a more preferable edge is found to connect the neighboring vertex to the subgraph of the established, policy-preferred paths toward the destination vertex. Hence, the neighboring vertex's join hop-distance is replaced according to the selected vertex; its successor list is cleared and the selected vertex is added to the successor list as shown at lines 33-35.

Lastly, if the relaxed edge's policy label is equal to the neighboring vertex's current join label and joining the neighboring vertex through the relaxed edge has the same join hop-distance, then an additional edge is found to connect the neighboring vertex to the subgraph of the established, policy-preferred paths toward the destination vertex. Hence, the selected vertex is simply added to the neighboring vertex's successor list as shown at line 37.

Finally, line 43 sets the color of the selected vertex to black to denote that it is fully explored and its policy consistent path(s) toward the destination vertex have been successfully computed. Note that if the destination vertex is not reachable from a source vertex via a valley-free path, its join label remains as *nil* and its join hop-distance remains as *infinity*.

In summary, Algorithm 1 starts from the destination vertex, v_d, and incrementally builds policy consistent paths in backwards from a source vertex, v_s, toward the destination vertex.

At each iteration, the most preferable gray vertex under the total order relation "$\succ^*$" (more preferable) is joined to the subgraph of the established, policy-preferred paths toward the destination vertex via one or more preferred edges. Put in other words, the edges connecting a gray vertex to one or more black vertices with the highest policy label are joined to the subgraph of the established, policy-preferred paths. In case there are multiple edges with the same policy label, the ones that are closest to the destination vertex are joined.

Let $H = (V', E')$ be the output graph generated by Algorithm 1. $H = (V', E')$ is a subgraph of $G(V, E, \Phi)$ formed by the policy-preferred paths toward the destination vertex such that $V' \subseteq V$ and $E' \subseteq E$.

Theorem 5.1. *$H = (V', E')$ is a directed, acyclic graph.*

Proof by Induction. Let $H_i = (V'_i, E'_i)$ be the subgraph of the established, policy-preferred paths after the i^{th} iteration of Algorithm 1. After the first iteration the subgraph of policy-preferred paths, H_1, consist of only the destination vertex, so the claim is true. Assume that the claim is true after $i-1$ iterations, i.e., $H_{i-1} = (V'_{i-1}, E'_{i-1})$ is a directed, acyclic graph. Let v_i be the next vertex to be joined to H_{i-1} via one or more edges. Since the paths are built in backwards, any edge, e_i, connecting v_i to H_{i-1} must have v_i as the predecessor vertex, i.e., $e_i = (v_i, u)$ such that $v_i \in V - V'_{i-1}$ and $u \in V'_{i-1}$. Put in other words, v_i is connected to H_{i-1} only via one or more of its outgoing edges. The minimum requirement for v_i to make a directed cycle in H_i is two edges such that the first edge is an outgoing edge from v_i and the second edge is an incoming edge to v_i from a vertex appearing on a path from v_i to the destination vertex. However, v_i is joined to H_{i-1} only via its outgoing edges therefore, joining v_i to H_{i-1} cannot form a directed cycle in H_i. □

Corollary 5.2. *$H = (V', E')$ is a rooted, directed, acyclic graph (r-DAG) which is rooted by the destination vertex, $v_d \in V'$, because ideally v_d is the only vertex which does not have an outgoing edge in H.*

5.3 Valley-Free AS Path Detection Pseudocode

In this part we define the procedure used to enforce the valley-free path property at line 18 of Algorithm 1. By definition, a simple path is called *valley-free* if the path consists of zero or more *customer* edges followed by zero or one *peer* edge, and then zero or more *provider* edges. Note that edge labels are appointed by the predecessor vertices. Hence, an edge labeled as *customer* connects a customer AS to a provider AS, whereas an edge labeled as *provider* connects a provider AS to a customer AS (see Section 3).

We introduce a non-deterministic finite automaton with ε transitions, $(\Sigma, S, s_0, \delta, F)$, recognizing valley-free paths in an AS level graph, $G = (V, E, \Phi)$, in Figure 4. The automaton consists of an input alphabet $\Sigma = \{provider, peer, customer\}$, a set of states $S = \{State_0, State_1, State_2, State_3, State_4\}$, an initial state $s_0 = State_0$, a state transition function $\delta : S \times (\Sigma \cup \varepsilon) \to \mathcal{P}(S)$, and a set of final states $F = \{State_4\}$.

Our implementation of valley-free AS path detection procedure follows the NFA-ε given in Figure 4. Since Algorithm 1 incrementally explores alternative edges by relaxation and a subpath of a valley-free path is also valley-free (see Section 6, Corollary 6.2), it is enough

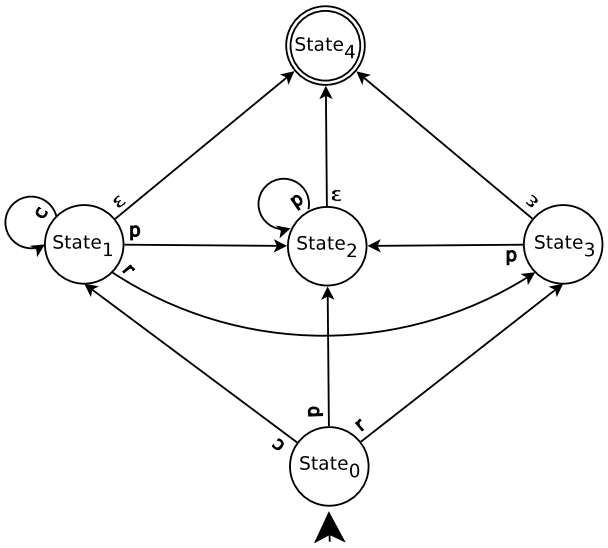


Figure 4: NFA-ε recognizing valley-free AS paths

Algorithm 2 Valley-Free AS Path Detection

Input: $l_p \in \mathcal{L}$ ▷ Previous edge policy label
Input: $l_n \in \mathcal{L}$ ▷ Next edge policy label
1: **procedure** $is\_valley\_free(l_p, l_n)$
2: **return**
3: $(l_p = customer \wedge (l_n = provider \vee l_n = peer \vee l_n = customer))$
4: $\vee(l_p = peer \wedge l_n = provider)$
5: $\vee(l_p = provider \wedge l_n = provider)$
6: $\vee(l_n = nil)$
7:
8: **end procedure**

to check if the last edge label transition preserves the valley-free property before relaxing an edge.

Algorithm 2 provides a procedure which recognizes if joining an edge to the subgraph of the established, policy-preferred paths will violate the valley-free property or not. The algorithm expects a previous edge label and a next edge label as input. Lines 3-5 validates the transition from the previous edge label to the next edge label according to the NFA-ε given in Figure 4. Line 6 is introduced to validate the initial transitions from the destination vertex which has the void join label *nil* in Algorithm 1. Note that Algorithm 1 builds the paths in backwards hence, it arranges the order of the arguments in the procedure call, accordingly at line 18. Moreover, the join edge label of a vertex reflects the policy label of the edge(s) connecting the vertex to the r-DAG, i.e., the next edge policy label toward the destination.

6 Analysis of Soundness

In this part we prove that Algorithm 1 produces a rooted, directed, acyclic subgraph (r-DAG), which is formed only by the policy-preferred paths from source vertices toward the destination vertex. In the following, we first introduce two lemmas that we use as the

building blocks of the main proof.

Let $G = (V, E, \Phi)$ be an edge labeled, finite digraph representing an Internet topology map.

Lemma 6.1. *The edges making up a valley-free path, P, from a source vertex, v_s, to a destination vertex, v_d, form a monotonically increasing sequence in terms of their edge policy labels under the total order relation "$\succ$" (succeeds or higher than). Put in other words, $\Phi(P[i+1]) \succeq \Phi(P[i])$ where the hop index $i \in \mathbb{Z}_{\geq 1}$ is between 1 and $|P|$.*

Proof by Construction. By definition, a simple path from a source vertex, v_s, to a destination vertex, v_d, is called *valley-free* if the path takes zero or more *customer* edges followed by zero or one *peer* edge, and then zero or more *provider* edges. Furthermore, Definition 4.1 states that the set of edge policy labels $\mathcal{L} = \{customer, provider, peer\}$ is a linearly ordered set under the strict total order relation "$\succ$" (succeeds or higher than) where $provider \succ peer \succ customer$. As a result, the sequence of edges making up a valley-free path from a source vertex, v_s, to a destination vertex, v_d, monotonically increases from the first edge towards the last edge. □

Corollary 6.2. *Any subpath of a valley-free path is also a valley-free path, because any sub-sequence of a monotonically increasing sequence is also monotonically increasing.*

Remember that at each iteration, Algorithm 1 expands the r-DAG of the established, policy-preferred paths by joining the most preferable gray vertex under the total order relation "$\succ^*$" (more preferable). A gray vertex with the highest join label and a shorter hop distance to the destination vertex is the most preferable vertex. If the gray vertex has multiple edges with the same edge label and the same hop distance then, all edges are included in the resulting r-DAG. In case there are multiple, most preferred gray vertices, one is selected arbitrarily.

Lemma 6.3. *At any iteration of Algorithm 1, there cannot be an edge connecting a gray vertex to a black vertex which either has a higher edge label than the edges connecting the selected vertex or has the same edge label and a shorter hop distance to the destination vertex.*

Proof by Contradiction. Let v_i be the selected vertex with the highest join label and shortest hop distance at the i^{th} iteration of Algorithm 1. Without loss of generality, let $e_i = (v_i, u)$ be the only edge connecting v_i to the subgraph of the established, policy-preferred paths as shown in Figure 5. Let $e_j = (x, y)$ be an edge which either has a higher edge label than e_i or has the same edge label and a shorter hop distance to the destination vertex, v_d. In the former case the join label of x would be higher than the join label of v_i. Therefore, x must have been the selected vertex from the priority queue (Line 16, Algorithm 1). In the latter case the join label of x would be the same as the join label of v_i but the join hop-distance of x would be shorter than the hop distance of v_i. Therefore, x must have been the selected vertex from the priority queue (Line 16, Algorithm 1). In both cases x is not the selected vertex from the priority queue. □

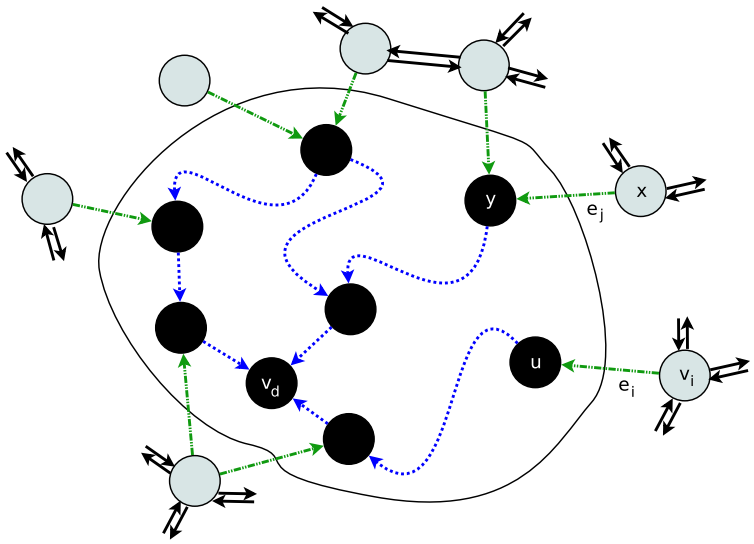


Figure 5: A visual snapshot of Algorithm 1

Theorem 6.4. *Algorithm 1 produces a rooted, directed, acyclic subgraph, $H = (V', E')$, of $G = (V, E, \Phi)$ which is formed only by the policy-preferred paths from any source vertex toward the destination vertex, v_d, under the assumption that v_d is reachable from any source vertex via a valley-free path.*

Proof by Induction. Let $H = (V', E')$ be the r-DAG of policy-preferred paths generated by the algorithm where $V' \subseteq V$ and $E' \subseteq E$. Let $H_i = (V'_i, E'_i)$ be the subgraph of the established, policy-preferred paths after the i^{th} iteration of Algorithm 1. Our proof is based on the fact that each new vertex is joined to the subgraph via one or more edges that form policy-preferred paths toward the destination vertex. Hence, the policy-preferred paths characteristic of the subgraph, $H_i = (V'_i, E'_i)$, is preserved after each iteration i.

Initially the subgraph of policy-preferred paths, H_1, consists of only the destination vertex, so the claim is true. Assume that after $i-1$ iterations the algorithm successfully built a policy-preferred subgraph, H_{i-1}, for $i-1$ vertices. Without loss of generality, let the algorithm select $v_i \in V - V'_{i-1}$ at the i^{th} iteration as the most preferable vertex and join it to H_{i-1} through edge e_i as shown in Figure 5. Note that Algorithm 1 builds the policy-preferred paths in backwards hence, the directed edge connecting v_i to H_{i-1} must have v_i as the predecessor vertex. That is, $e_i = (v_i, u)$ such that $v_i \in V - V'_{i-1}$ and $u \in V'_{i-1}$. Moreover, the color of v_i must be gray and the color of u must be black. That is, vertex u has already been fully explored. In the following we claim by contradiction that any path from v_i to v_d is policy-preferred hence, the subgraph $H_i = (V'_i, E'_i)$ preserves the policy-preferred paths characteristic.

Assume that the edge, e_i, connecting v_i to H_{i-1} forms a non-policy-preferred path, P, toward the destination vertex. Hence, subgraph H_i does not preserve policy-preferred paths characteristic. That is, either (Case 1) v_i has another edge with a higher policy label than e_i that forms a path P' toward v_d, i.e., $\Phi(P'[1]) \succ \Phi(P[1] = e_i)$, or (Case 2) v_i has another edge with the same policy label as e_i that forms a shorter path P' toward the destination vertex, i.e., $\Phi(P'[1]) = \Phi(P[1] = e_i) \wedge |P'| < |P|$.

Case 1: $\exists P'$ such that $\Phi(P'[1]) \succ \Phi(P[1] = e_i)$
Assuming that P' is a policy-preferred path toward the destination vertex, there must be an edge $P'[j] = (x, y)$ which passes through a gray vertex $x \in V - V'_{i-1}$ and a black vertex $y \in V'_{i-1}$ because H_{i-1} is a policy-preferred subgraph toward the destination. Note that vertices x and v_i as well as y and u are not necessarily different, yet they cannot be the same simultaneously. Since $\Phi(P'[1]) \succ \Phi(P[1] = e_i)$ and the edges of P' monotonically increases (Lemma 6.1), then $\Phi(P'[j]) \succeq \Phi(P'[1]) \succ \Phi(P[1] = e_i)$ where $j \geq 1$. However, the conclusion $\Phi(P'[j]) \succ \Phi(P[1] = e_i)$ cannot be true because it contradicts with Lemma 6.3.
Case 2: $\exists P'$ where $\Phi(P'[1]) = \Phi(P[1] = e_i) \wedge |P'| < |P|$
Assuming that P' is a policy-preferred path toward the destination vertex, there must be an edge $P'[j] = (x, y)$ which passes through a gray vertex $x \in V - V'_{i-1}$ and a black vertex $y \in V'_{i-1}$ because H_{i-1} is a policy-preferred subgraph toward the destination. Note that vertices x and v_i as well as y and u are not necessarily different, yet they cannot be the same simultaneously. Let P'' be the subpath of P' which starts from vertex x toward the destination v_d. Because P' is shorter than P, then its subpath P'' is also shorter than P, i.e., $|P''| < |P|$. Moreover, since $\Phi(P'[1]) = \Phi(P[1] = e_i)$ and the edges of P' monotonically increases (Lemma 6.1), then $\Phi(P''[1] = P'[j]) \succeq \Phi(P'[1]) = \Phi(P[1] = e_i)$ where $j \geq 1$. However, the conclusion $\Phi(P''[1]) \succeq \Phi(P[1] = e_i)$ and $|P''| < |P|$ cannot be true because it contradicts with Lemma 6.3. □

Definition 6.1. *The suffix paths of a simple path P are those subpaths of P which strictly include the last edge of P.*

Corollary 6.5. *Let H be the r-DAG computed by Algorithm 1. Any suffix path of a policy-preferred path in H is also a policy-preferred path. Because H is only formed by policy-preferred paths from source vertices to the destination vertex (Theorem 6.4).*

7 Analysis of Completeness

In this part, we first prove that Algorithm 1 eventually terminates. Next, we show that if a destination vertex is reachable from a source vertex via a policy-preferred path, then the algorithm successfully computes it.

Lemma 7.1. *A gray vertex that is dequeued and fully explored is never enqueued in Q again.*

Proof. Only white vertices can be enqueued as shown at lines 19-24 of Algorithm 1. An enqueued white vertex becomes gray (line 23, Algorithm 1) and a gray vertex is eventually dequeued (line 16, Algorithm 1). A dequeued gray vertex becomes black (line 43, Algorithm 1) and a black vertex stays as black during the course of the algorithm. As a result, a dequeued vertex will never be enqueued in Q again. □

Theorem 7.2. *Algorithm 1 eventually terminates.*

Proof. Since the input topology graph, $G(V, E, \Phi)$, is a finite graph consisting of a finite number of vertices and a dequeued vertex is not enqueued in the priority queue, Q, again (Lemma 7.1), Q eventually becomes empty and the algorithm terminates. □

Lemma 7.3. *Algorithm 1 ensures that any incoming edge of a selected vertex is relaxed with the condition that it preserves valley-free property and induces on a gray or white vertex.*

Proof. The algorithm processes the incoming edges of the selected vertex at line 17. At line 18 the algorithm enforces valley-free path condition to each incoming edge. At lines 19-24 and 25-38 the algorithm relaxes incoming edges if the predecessor vertex is white or gray, respectively. □

Lemma 7.4. *If a selected, gray vertex is connected to multiple black vertices via different outgoing edges which have the same edge label and the same hop distance toward the destination, then Algorithm 1 includes all the edges in the resulting r-DAG.*

Proof. Note that an outgoing edge connecting a gray vertex to a black vertex is considered to be an incoming edge from the perspective of the black vertex. If the selected, gray vertex is connected to multiple black vertices via different outgoing edges, then the black vertices were selected earlier. If the black vertices were selected earlier then all of their incoming edges were already relaxed (Lemma 7.3). Because all these incoming edges have the same edge label and the same hop distance toward the destination, the edges were included in the resulting r-DAG at line 37 of Algorithm 1. □

Theorem 7.5. *If a destination vertex, v_d, is reachable from a source vertex, v_s, via a policy-preferred path $P = \{(v_s, v_1), (v_1, v_2), \ldots, (v_i, v_j), (v_j, v_{j+1}), \ldots, (v_{k-1}, v_d)\}$ in the input topology graph G, then Algorithm 1 successfully computes it. That is, the resulting r-DAG, H, includes the policy-preferred path P.*

Proof by Contradiction. Assume that Algorithm 1 failed to compute the policy-preferred path P, i.e., the resulting r-DAG, H, does not include the policy-preferred path P. Put in other words, one or more edges in P are missing in H. Let $P[j] = (v_i, v_j)$, $1 \leq j \leq |P|$, be the closest missing edge to the destination vertex, v_d. Since the paths are built in backwards, $P[j]$ would be the first edge in P which is missed by the algorithm. In accordance with Corollary 6.5 the policy-preferred suffix path $P_j = \{(v_j, v_{j+1}), \ldots, (v_{k-1}, v_d)\}$ is in H. However, the suffix path $P_i = \{(v_i, v_j), (v_j, v_{j+1}), \ldots, (v_{k-1}, v_d)\}$ is not in H because edge (v_i, v_j) is missed.

There are three cases that may lead Algorithm 1 to miss edge (v_i, v_j) while constructing the policy-preferred path P_i.

Case 1: Algorithm 1 computed a non-policy-preferred path P_i' from v_i to v_d instead of the policy-preferred path P_i.

Theorem 6.4 states that the policy-preferred paths computed by the algorithm are correct. Therefore, the case statement cannot be true.

Case 2: While building policy consistent paths in backwards, Algorithm 1 failed to discover vertex v_i at all.

The policy-preferred path, P_j, from v_j to v_d is in H. Moreover, vertex v_j is discovered before v_i because the paths are built in backwards. Then, the edge (v_i, v_j) must have been relaxed by Algorithm 1 when v_j was selected (Lemma 7.3). Hence, v_i must have been discovered when the edge (v_i, v_j) was relaxed. As a result, the case statement cannot be true.

Case 3: Vertex v_i has multiple edges including (v_i, v_j) which have the same edge policy label and hop distance toward the destination vertex, however Algorithm 1 simply missed edge (v_i, v_j) while including other edge(s).

The policy-preferred path, P_j, from v_j to v_d is in H. Moreover, vertex v_j is discovered before v_i because the paths are built in backwards. Then, the edge (v_i, v_j) must have been relaxed by Algorithm 1 when v_j was selected (Lemma 7.3). Since the edge (v_i, v_j) has the same edge label and hop distance as the included edge(s), it had to be included in H as well (Lemma 7.4). Therefore, the case statement is not true.

The analysis above shows that the supposedly missed edge (v_i, v_j), in fact, cannot be missed by Algorithm 1, hence P_i must be part of H. Applying the same logic to all supposedly missing edges of P in backwards shows that the entire policy-preferred path P must have been included in H. As a result, if there is a policy-preferred path between a source and the destination vertex, Algorithm 1 successfully computes it. □

8 Analysis of Time and Space Complexity

We analyze the time complexity of Algorithm 1 by dividing the algorithm into four parts. Lines 2-6 present the *initialization* part. Lines 9-13 represent the *setup* part. Line 16 is the *vertex selection* part. Lastly, lines 17-41 denote the *edge exploration* part.

The initialization part (lines 2-6) involves initializing all vertices in the graph. Lines 3-5 have constant time complexity $O(1)$ and executed once for each vertex. Hence, the initialization part time complexity is $O(|V|)$.

The setup part (lines 8-13) presents the set-up of the destination vertex and the declaration of a priority queue. Each line in this part has constant time complexity $O(1)$. Therefore, the setup part has constant time complexity $O(1)$.

Since a vertex in V can be enqueued only once and dequeued only once (Lemma 7.1), the `while` loop at line 15 can be executed at most $|V|$ times. The `pop` operation at line 16 has constant time complexity. However, the priority queue needs to be re-organized after a `pop` operation to determine the next top element. Re-organizing a priority queue of size n has $O(logn)$ time complexity. Therefore, the worst case time complexity of the `pop` operation at line 16 is $O(log|V|)$ for each vertex. Since there are $|V|$ vertices in total, the time complexity of vertex selection part (line 16) is $|V| \cdot O(log|V|)$.

The `for` loop at line 17 explores the incoming neighbors of each vertex. Line 18 involves the verification of valley-free property which has constant time complexity $O(1)$. Lines 19-40 represent the relaxation step for the incoming edges which may also require re-organizing the priority queue if the status of an incoming vertex is updated or a `push` operation is executed. Therefore, the worst case time complexity of lines 19-40 is $O(log|V|)$ for each incoming edge. Since there are $|E|$ edges in total, the complexity of edge exploration part (lines 17-41) is $|E| \cdot O(log|V|)$. Finally, the total time complexity is $O((|V| + |E|)log|V|)$ by the dominating term.

The space complexity of Algorithm 1 hinges on the data structures used to represent the input graph G, the vertex successor list π and the priority queue Q. In our implementation, we used adjacency lists to represent the input graph G. The total space complexity for an adjacency list is $O(|V| + |E|)$. The successor list, π, holds a list of edges representing the

successor vertices of each vertex. Similar to an adjacency list, it has $O(|V| + |E|)$ space complexity. The priority queue, Q, has $O(|V|)$ space complexity. Finally, the total space complexity is $O(|V| + |E|)$ by the dominating term.

9 Empirical Evaluations

In this part we present a comparison of policy-preferred paths and shortest paths on a real world dataset to complement our theoretical analyses as well as to provide the reader with more insight. The Internet topology map used in our experiment is obtained from CAIDA [3]. *Since CAIDA constructs AS-level Internet topology maps using passively collected BGP data from UO routeviews project [35] and RIPE remote route collectors [28], their maps are compliant with the real BGP paths reported in those datasets.* Note that verifying the topology map itself is beyond the scope of this study, nevertheless the accuracy of CAIDA maps is reported to be around 99% based on the collected data [20]. The Internet topology map used in this study consists of 53,195 ASes connected to each other via 215,596 AS relations.

First, we computed all-pairs, policy-preferred paths using the proposed algorithm (Algorithm 1) in this study. The input graph consists of 53,195 vertices (ASes) connected to each other via 431,192 edges (dual AS relations). Algorithm 1 theoretically runs in time $O(|V|(|V| + |E|)log|V|)$ to compute all-pairs, policy-preferred paths. It took the algorithm 5 hours and 44 minutes to finish with 36 concurrent threads on a dual socket Intel Xeon E5-2680 Linux machine. Note that the wall-time execution of the algorithm depends on many factors including the implementation, system CPU, compiler, OS, and the number of processes.

Due to memory overhead we computed all-pairs shortest paths using a textbook implementation of Dijkstra's shortest paths algorithm rather than Johnson's algorithm or Floyd-Warshall algorithm. We used the same Internet topology map without considering the edge policy labels to compute the all-pairs shortest paths. The input graph for the all-pairs shortest paths algorithm consists of 53,195 vertices (ASes) connected to each other via 215,596 edges (undirected links), effectively. It took the algorithm 2 hours and 7 minutes to finish with 36 concurrent threads on the same machine. Algorithm 1 takes longer than the shortest paths algorithm, in practice, because the "$\succ^*$" (more preferable) relation given in Definition 5.1 requires two comparisons and valley-free AS path detection procedure (Algorithm 2) requires an additional function call at each iteration.

Both algorithms support multi-path detection between pairs of ASes and we considered multi-paths in our experiments. Note that multiple paths appearing between the same pair of ASes have the same path length for the shortest paths algorithm as well as for the policy-preferred paths algorithm.

9.1 Experimental Results:

In the following we analyze the lengths (hop distances) of the paths appearing between more than 2.8 billions of AS pairs in our topology graph. The paths computed by the shortest paths algorithm are referred as **SP-paths** and the paths computed by the policy-preferred

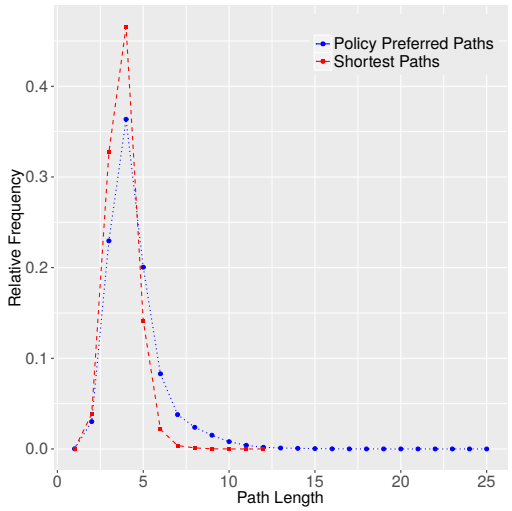


Figure 6: Relative path length frequencies for PPP and SP

paths are referred as **PPP-paths**. It is important to note that direct comparisons of multiple paths appearing between more than 2.8 billions of AS pairs were beyond our computational capacity. Therefore, we compared the path lengths between AS pairs which gives a lower bound on the discrepancy between the algorithms.

The average path lengths for the PPP-paths and the SP-paths are 4.47 and 3.79, respectively. PPP-paths are longer than the SP-paths by 0.68 hops on the average. Since the majority of the paths in an AS level Internet graph are not very long, a difference of 0.68 between the averages of PPP-paths and SP-paths is significant.

Figure 6 shows the relative path length frequencies for both PPP-paths (dotted, blue) and SP-paths (dashed, red). The minimum and maximum path lengths in SP-paths are 1 and 12, respectively. Moreover, more than 99% of the path lengths are between 2 and 6. On the other hand, PPP algorithm generates longer paths since the paths have to adhere BGP policy consistency. The minimum and maximum path lengths in PPP-paths are 1 and 25, respectively. Besides, more than 99% of the path lengths are between 2 and 10.

Evidently, the SP algorithm generates shorter paths compared to the PPP algorithm by opting for "policy inconsistent" and "less preferable" paths.

Figure 7 shows the path length differences calculated by subtracting the SP-path length from the corresponding PPP-path length for each AS pair. In the figure around 67% of the paths found by the SP and PPP algorithms have the same length, i.e., have difference zero. On the other hand, 33% of the paths differ by 1 to 20 hops, meaning that one third of the AS paths found by the SP algorithm violate policy consistency, for certain. Alternatively, at least one third of the paths found by the SP algorithm are the paths that are not employed in the Internet, in reality. Note that the PPP and SP algorithms may show discrepancy for the AS pair paths in the remaining two thirds as well. Specifically, in case there is a mixture of policy consistent (more preferable) and policy inconsistent (less preferable) shortest paths between a pair of ASes, the SP algorithm computes both types of the paths while the PPP algorithm computes only the policy consistent (more preferable) paths.

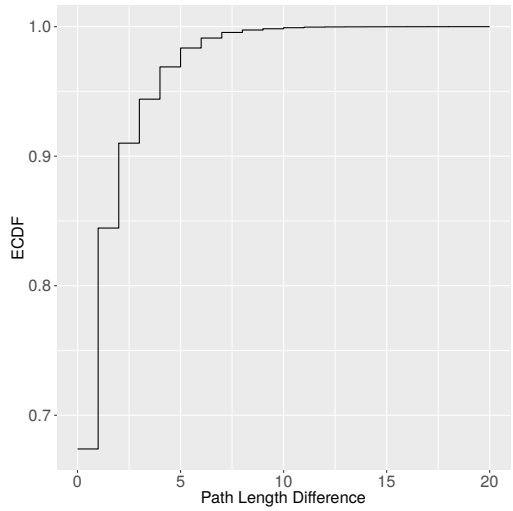


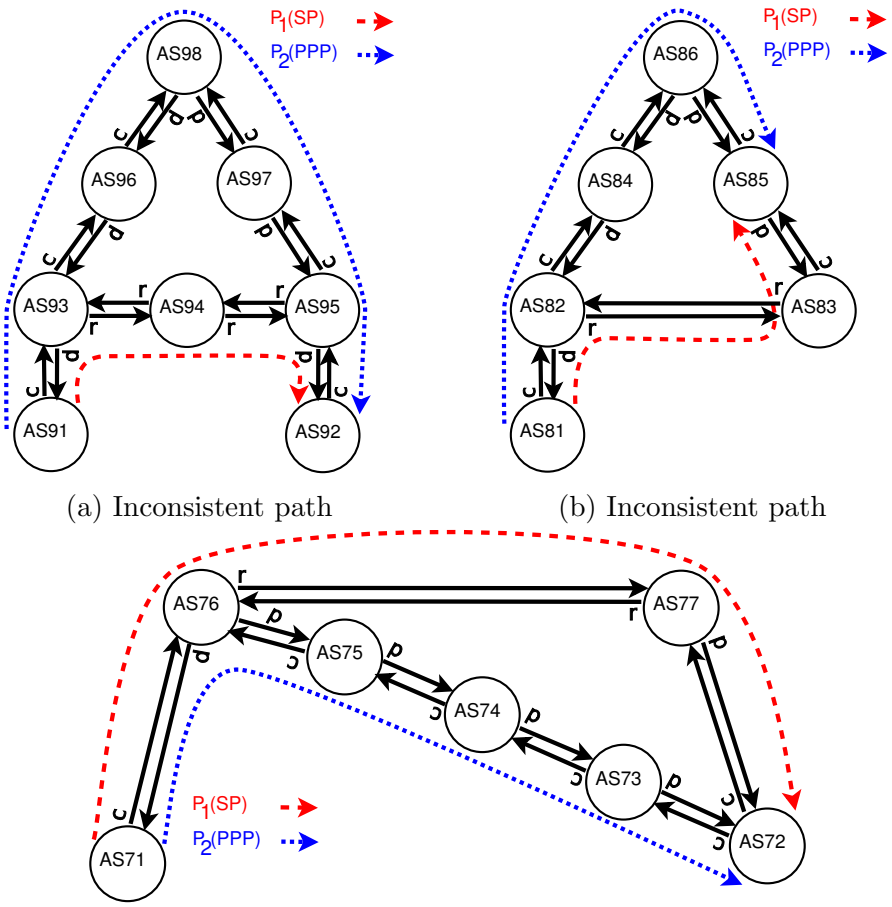
Figure 7: Path length differences between PPP and SP

9.2 Discussions:

The shorter path lengths resulting in the SP algorithm compared to the PPP algorithm can be attributed to (i) policy inconsistent path computation and (ii) less preferable path selection. Figure 8 illustrates the path length discrepancy between the SP algorithm and the PPP algorithm. In the figure, the paths in red (dashed) are the ones computed by the SP algorithm and the paths in blue (dotted) are the ones computed by the PPP algorithm between the same source and destination ASes. Lastly, the AS numbers in the figure are assigned arbitrarily and the lengths of the links in the illustrations do not have any specific meaning. Remember that an AS learns the existence of a path via an associated prefix reachability advertisement from one of its neighboring ASes. For the sake of brevity, we use use "prefix advertisement" and "path advertisement" interchangeably in our discussions.

Figures 8a and 8b show two cases of policy inconsistent path computation. In Figure 8a the SP algorithm computes path $P_1 = \{(AS91, AS93), (AS93, AS94), (AS94, AS95), (AS95, AS92)\}$ from $AS91$ to $AS92$ since it is the shortest path from the source to the destination. The SP algorithm violates BGP policy consistency by selecting two peer links, $(AS93, AS94)$ and $(AS94, AS95)$, induced on $AS94$. In reality, $AS94$ does not advertise the reachability information learned from one of its peers, $AS95$, to another peer, $AS93$. Although both subpaths $P_1' = \{(AS91, AS93), (AS93, AS94)\}$ and $P_1'' = \{(AS94, AS95), (AS95, AS92)\}$ are known by $AS94$, $AS94$ does not announce P_1'' to $AS93$. Therefore, $AS93$ cannot advertise $\{(AS93, AS94), P_1''\}$ to $AS91$. As a result, subpath $\{(AS93, AS94), P_1''\}$ does not even exist from the perspective of $AS91$. PPP algorithm, however, computes the accurate, policy consistent path $P_2 = \{(AS91, AS93), (AS93, AS96), (AS96, AS98), (AS98, AS97), (AS97, AS95), (AS95, AS92)\}$ between $AS91$ and $AS92$. Overall, SP algorithm computes a shorter, policy inconsistent path of length 4 compared to the PPP algorithm-computed path of length 6.

Similarly, in Figure 8b the SP algorithm computes the policy inconsistent path $P_1 = \{(AS81, AS82), (AS82, AS83), (AS83, AS85)\}$ from $AS81$ to $AS85$. In reality, $AS83$ does not advertise the reachability information learned from its provider, $AS85$, to its peer, $AS82$.



(a) Inconsistent path

(b) Inconsistent path

(c) Less preferable path

Figure 8: Policy Preferred Paths (PPP) algorithm compared to the Shortest Paths (SP) algorithm

That is, the subpath $\{AS83, AS85)\}$ is not announced to $AS82$ by $AS83$, hence cannot be announced to $AS81$ by $AS82$. Again, P_1 does not even exist from the perspective of $AS81$. On the other hand, PPP algorithm computes the accurate, policy consistent path $P_2 = \{(AS81, AS82), (AS82, AS84), (AS84, AS86), (AS86, AS85)\}$ from $AS81$ to $AS85$.

Figure 8c shows two paths, $P_1 = \{(AS71, AS76), (AS76, AS77), (AS77, AS72)\}$ and $P_2 = \{(AS71, AS76), (AS76, AS75), (AS75, AS74), (AS74, AS73), (AS73, AS72)\}$, sharing a common AS, $AS76$, from $AS71$ to $AS72$. Unlike the previous two examples, both P_1 and P_2 are policy consistent. However, $AS76$ prefers to advertise the longer subpath $P_2'' = \{(AS76, AS75), (AS75, AS74), (AS74, AS73), (AS73, AS72)\}$ to $AS71$ rather than the shorter subpath $P_1'' = \{(AS76, AS77), (AS77, AS72)\}$. Because, $AS76$ locally prefers a first-hop *provider* link, $(AS76, AS75)$, over a first-hop *peer* link, $(AS76, AS77)$, to make profit for the traffic destined to $AS72$. The SP algorithm computes the less preferable subpath P_1'' compared to the more preferable subpath P_2'' at $AS76$ and constructs path P_1 from

$AS71$ to $AS72$. On the other hand, PPP algorithm computes the accurate, more preferable subpath P_2'' and construct the path P_2 from $AS71$ to $AS72$.

In summary, SP algorithm generates shorter paths compared to the PPP algorithm because it either computes policy inconsistent paths or selects less preferable subpaths.

10 Conclusions

Using AS-level Internet topology maps to determine accurate AS-level paths is essential for detecting congestion points in the Internet, reducing the overall traffic in P2P networks, enhancing the quality of VoIP services, estimating network delays and optimizing service deployment in the Internet.

In this study we introduced a novel single-destination, policy-preferred path enumeration algorithm which discovers preferable, policy consistent paths from all ASes to a destination AS in an AS-level Internet topology graph. Additionally, we presented rigorous analyses on its soundness, completeness, time complexity and space complexity. Our algorithm provides a holistic solution to the AS-level path enumeration problem by incorporating common practices and incentives in inter-AS routing. The proposed algorithm runs in time $O((|V| + |E|)log|V|)$ which is the same as Dijkstra's shortest paths algorithm with a priority queue implementation. Our experimental results show that at least one third of the paths found by the shortest paths algorithm violate AS policy consistency.

Finally, a C++ implementation of the presented algorithm is publicly available on our project website at `http://nsrg.louisiana.edu/project/ntmaps/` .

References

[1] A. Akella, S. Seshan, and A. Shaikh. An empirical evaluation of wide-area internet bottlenecks. In *ACM IMC*, Miami, FL, USA, Oct 2003.

[2] T. Bu and D. Towsley. On distinguishing between internet power law topology generators. In *INFOCOM, IEEE*, Jun 2002.

[3] CAIDA. Dataset. http://data.caida.org/datasets/as-relationships/serial-1/20160201.as-rel.txt.bz2.

[4] H. Chang, S. Jamin, and W. Willinger. Inferring as-level internet topology from router-level path traces. In *SPIE ITCom*, Denver, CO, USA, Aug 2001.

[5] R. Cohen, K. Erez, D. Avraham, and S. Havlin. Breakdown of the internet under intentional attack. *Phys. Rev. Let.*, 86:3682–3685, 2001.

[6] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, k. claffy, and G. Riley. AS Relationships: Inference and Validation. *ACM Computer Communication Review*, 37(1):29–40, Jan 2007.

[7] D. Dolev, S. Jamin, O. Mokryn, and Y. Shavitt. Internet resiliency to attacks and failures under {BGP} policy routing. *Computer Networks*, 50(16):3183 – 3196, 2006.

[8] A. Faggiani, E. Gregori, A. Improta, L. Lenzini, V. Luconi, and L. Sani. A study on traceroute potentiality in revealing the internet as-level topology. In *IFIP Networking*, Trondheim, Norway, Jun 2014.

[9] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632 (Best Current Practice), Aug 2006.

[10] L. Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, Dec 2001.

[11] V. Giotsas, M. Luckie, B. Huffaker, and K. Claffy. Inferring Complex AS Relationships. In *ACM IMC*, Nov 2014.

[12] E. Gregori, A. Improta, L. Lenzini, L. Rossi, and L. Sani. Bgp and inter-as economic relationships. In J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, and C. Scoglio, editors, *NETWORKING 2011*, volume 6641 of *Lecture Notes in Computer Science*, pages 54–67. Springer Berlin Heidelberg, 2011.

[13] S. Hasan, S. Gorinsky, C. Dovrolis, and R. Sitaraman. Trade-offs in optimizing the cache deployments of cdns. In *IEEE INFOCOM*, Toronto, Canada, Apr 2014.

[14] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930, Mar 1996.

[15] Y. He, G. Siganos, M. Faloutsos, and S. Krishnamurthy. Lord of the links: A framework for discovering missing links in the internet topology. *IEEE/ACM Trans. Netw.*, 17(2):391–404, Apr 2009.

[16] J. Juen, A. Johnson, A. Das, N. Borisov, and M. Caesar. Defending tor from network adversaries: A case study of network path prediction. *Proc. on Privacy Enhancing Technologies*, 2015(2):171–187, Jun 2015.

[17] A. Khan, T. Kwon, H. Kim, and Y. Choi. As-level topology collection through looking glass servers. In *ACM IMC*, Spain, Oct 2013.

[18] D. Lee, K. Jang, C. Lee, G. Iannaccone, and S. Moon. Scalable and systematic internet-wide path and delay estimation from existing measurements. *Computer Networks*, 55(3):838–855, Feb 2011.

[19] J. Li and K. Sollins. Exploiting autonomous system information in structured peer-to-peer networks. In *ICCCN*, pages 403–408, Oct 2004.

[20] M. Luckie, B. Huffaker, K. Claffy, A. Dhamdhere, and V. Giotsas. AS Relationships, Customer Cones, and Validation. In *Internet Measurement Conference (IMC)*, pages 243–256, Oct 2013.

[21] Z. Mao, D. Johnson, J. Rexford, J. Wang, and R. Katz. Scalable and accurate identification of as-level forwarding paths. In *INFOCOM. IEEE*, volume 3, pages 1605–1615, Mar 2004.

[22] Z. Mao, J. Rexford, J. Wang, and R. Katz. Towards an accurate as-level traceroute tool. In *SIGCOMM*, pages 365–378, New York, NY, USA, 2003.

[23] Z. M. Mao, L. Qiu, J. Wang, and Y. Zhang. On as-level path inference. In *ACM SIGMETRICS*, Alberta, Canada, Jun 2005.

[24] R. Oliveira, D. Pei, W. Willinger, B. Zhang, and L. Zhang. The (in)completeness of the observed internet as-level structure. *IEEE/ACM Transactions on Networking*, 18(1):109–122, Feb 2010.

[25] J. Qiu and L. Gao. As path inference by exploiting known as paths. In *Global Telecommunications Conference, IEEE*, Nov 2006.

[26] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), Jan 2006.

[27] S. Ren, L. Guo, and X. Zhang. ASAP: an AS-Aware Peer-Relay Protocol for High Quality VoIP. page 70, 2006.

[28] RIPE. Routing information service. www.ripe.net.

[29] M. Roughan, W. Willinger, O. Maennel, D. Perouli, and R. Bush. 10 lessons from 10 years of measuring and modeling the internet's autonomous systems. *IEEE Journal on Selected Areas in Communications*, 29(9):1810–1821, 2011.

[30] N. Tao, X. Chen, and X. Fu. As path inference: From complex network perspective. In *IFIP Networking Conference*, pages 1–9, May 2015.

[31] M. E. Tozal. Enumerating single destination, policy-preferred paths in as-level internet topology maps. In *IEEE Sarnoff Symposium*, Newark, NJ, USA, Sep 2016.

[32] M. E. Tozal. The Internet: A system of interconnected autonomous systems. In *IEEE Systems Conference*, Orlando, FL, USA, Apr 2016.

[33] UCLA. Internet research lab. irl.cs.ucla.edu.

[34] UCSD. Center for applied internet data analysis. www.caida.org.

[35] UO. Route views project. www.routeviews.org.

[36] J. Xia and L. Gao. On the evaluation of as relationship inferences. In *Global Telecommunications Conference, IEEE*, volume 3, pages 1373–1377, 2004.

[37] B. Zhang, R. Liu, D. Massey, and L. Zhang. Collecting the internet as-level topology. *SIGCOMM Comput. Commun. Rev.*, 35(1):53–61, Jan 2005.