International Journal of Hybrid Information Technology
Vol.3, No.1, January, 2010

An Efficient Algorithm for AS Path Inferring

Yang Guoqiang and Dou Wenhua

National Univernity of Defence Technololy, China
yanggq@nudt.edu.cn

Abstract

Discovering the AS paths between two ASes are invaluable for a wide area of network research and application activities. The traditional techniques for path discovery require direct access to the source node. Recently with more accurate AS relationship inferring algorithm and publicly available AS topology data, it is possible to infer AS paths without accessing the source. This paper proposes an efficient algorithm for inferring all pair shortest AS paths in a relationship annotated AS graph. The running time of the algorithm is O (NM), where N is the number of nodes and M is the number of edges in AS graph. The algorithm bases on the bread-first-search (BFS) algorithm, and experimental results show that it reduces running time dramatically compared with the existing algorithm whose running time is O (N^3).

Keywords: *AS-level path, Network Topology, AS relationships, Internet Routing*

1. Introduction

The Internet is composed of thousands of Autonomous Systems (ASes) controlled by different administrative domains. Examples of administrative domains range from college campuses and corporate networks to large Internet Service Providers(ISPs). Discovering the AS paths between two ASes is invaluable for a wide area of network research and application activities like network diagnosis, routing behavior analysis, proper server siting and high-performance overlay routing designing. One solution to this problem is to launch AS level traceroute tool [1] from the source or to directly access the BGP tables where the source situates. The problem with this solution is that in most cases we do not have privileged access to the source or the BGP tables. The other problem is that, due to asymmetric routing and multihoming, it will be difficult for the solution to find the accurate AS paths[2].

The routing path between two ASes is determined by the interdomain routing protocols such as Border Gateway Protocol(BGP). Interdomain protocols allow each AS to select its own routing policy, and the most important factor in determining routing policies is the commercial relationships between administrative domains. These relationships result in the valley-free routing model which states that every AS path has a hierarchical structure[3]. It means that, provided the AS graph annotated with AS relationships, the AS paths between two ASes can be inferred by finding all shortest policy paths that follow the valley-free model[3]. With this approach, the AS paths can be figured out without having to access the source.

Mao[2] proposes the first algorithm in terms of this idea, which is a modified version of Dijkstra's algorithm. To find the shortest policy paths between a given node pair, the algorithm enumerates all intermediate nodes that can bridge the two endpoint nodes to

form valid AS paths. Therefore, its running time to discover all pair shortest policy paths in the graph is $O(N^3)$, where N is the number of nodes in AS graph.

In this work, we introduce an efficient algorithm for inferring all pair AS paths, whose running time is O(NM), where M is the number of links in AS graph. Unlike Mao's algorithm, our algorithm bases on the bread-first-search(BFS) algorithm. There are two steps in our algorithm: traversal and traceback. In the traversal step, the algorithm traverses the graph from a given source in BFS order, and forms a subgraph which only contains links that are in one of the shortest AS paths from the source. And in the traceback step, the algorithm traces from each node back to the source in the subgraph to find all shortest AS paths from source. We evaluate our algorithm using the AS data from CAIDA [4], and find that it achieves comparable accuracy with Mao's algorithm in less time.

The rest of this paper is structured as follows. Section 2 gives a brief introduction to related work. We detail our algorithm in section 3 and evaluate it in section 4. Finally we conclude this paper in section 5.

2. Related Work

The Internet has experienced a tremendous growth since its commercialization. Understanding Internet topology is important for network researchers to study routing behavior, evaluate networking protocols and optimize network performance. The previous work can be classified into two categories: path discovery and topology discovery. In this work, we focus on the AS level path discovery.

The most widely used tool to obtain network path is traceroute [1]. To determine the network path from source node to target node, it sends a sequence of TTL-limited packets to target node at source node. The paths obtained by traceroute are route level paths. [1] provide an algorithm for converting the route level paths to AS level paths by mapping IP to AS number. [5] proposes a more accurate algorithm to map a given IP to AS number by identifying AS border routers. The main problem with traceroute is that, it has to access the source node when obtaining network path.

The fact that commercial AS relationship plays a significant role in shaping the Internet structure motivates the efforts for AS relationship inference. Gao's work [3] is the first to define AS relationships and valid AS path pattern. She also proposes a heuristic algorithm to infer AS relationships based on the AS paths derived from BGP table. After that, many efforts have been made to more accurately define [6-9] and infer [10-14] AS relationships.

Inspired by the AS relationship inference efforts, Mao [2] proposes an algorithm to inferring AS paths without accessing the source node. The algorithm infers AS paths based on the AS relationships and the valid AS path pattern. The running time for discovering all AS paths in the AS graph is $O(N^3)$, where N is the number of nodes in AS graph. The drawback of the algorithm is that it dose not take into account the sibling-sibling edges.

3. AS Path Inference Algorithm

In this section, we propose an efficient algorithm to infer AS paths. There are two stages in our algorithm: traversal and traceback. In the traversal stage, the algorithm traverses the graph from a given source in BFS order, and forms a subgraph which only contains edges that are in one of the shortest AS paths from the source. And in the

traceback stage, the algorithm traces from each node back to the source in the subgraph to find all shortest AS paths from source.

3.1 AS Relationships and AS Path

AS relationships can be categorized into three types to reflect different business agreements between ASes.

- Customer-provider(c2p): a customer AS pays a provider AS for transiting traffic from the customer and also for delivering traffic to the customer.
- Peer-peer(p2p): two ASes exchange traffic between their customers but do not exchange traffic from or to their providers or peers.
- Sibling-sibling(s2s): two ASes exchange traffic from or to any ASes.

AS relationships can be represented by an annotated AS graph. An annotated AS graph is a partially directed graph in which nodes represent ASes and edges represent the relationships between ASes. Fig. 1 shows an example of an annotated AS graph.

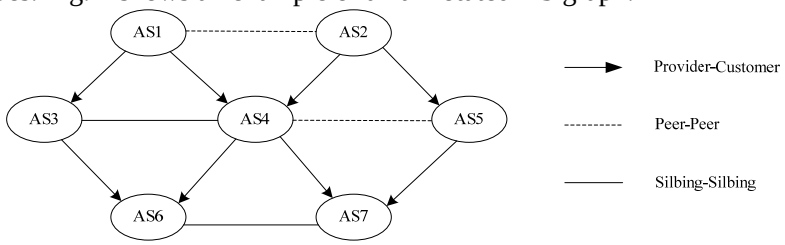

Figure 1. An annotated AS graph

According to the definition of AS relationships, a valid AS path must comply with the following hierarchical pattern: an uphill segment of zero or more c2p or s2s links, followed by a flat segment of zero or one p2p link, followed by a downhill segment of zero or more p2c or s2s links.

For example, in Fig. 1 AS3-AS1-AS4-AS7 is a valid AS path. But AS3-AS6-AS4-AS7 is not a valid AS path, because AS6 is a customer of AS3 and AS6 do not transit traffic from AS3.

We can categorize AS paths into two types: uphill and downhill paths.

1) uphill path, empty or containing only c2p or s2s edges;

2) downhill path, containing at least one p2c or p2p edge.

When we explore the AS graph, the AS path type changes with the type of the edge explored. The transition of AS path type is a state machine illustrated in Fig. 2. For example, if current AS path type is uphill, and the edge type we explored is p2p, then the type of AS path changes to downhill.

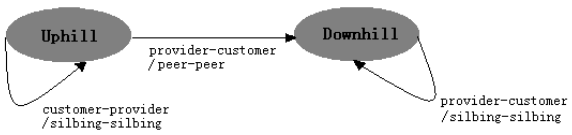

Figure 2. Transition state machine of AS path type, and the initial state is uphill

3.2 Traversal Stage

There are two stages in our algorithm: traversal stage and traceback stage. In the traversal stage, the algorithm traverses the graph from a given source in BFS order, and forms a subgraph which only contains edges that are in one of the shortest AS paths from the source. And in the traceback stage, the algorithm traces from each node back to the source in the subgraph to find all shortest AS paths from source.

To simplify the description of the algorithm, we introduce some notations here. We use $d_{up}(s, u)$ to denote the length of the shortest uphill AS path from source node s to node u. We use $d_{down}(s, u)$ to denote the length of the shortest downhill AS path from source node s to node u. We use *predecessor*(u, s) to denote set of nodes such that for any node p in *predecessor*(u, s), there exists a shortest AS path (s,...,p,u).

In the traversal stage, we explore AS graph in several steps. In the Nth step, we explore nodes that the length of the shortest AS path from source to them is N.

We use two node sets S_{up} and S_{down} to store the nodes to be expanded each step. The uphill node set S_{up} stores nodes that one of the shortest AS paths from source to them is uphill path. And the downhill node set S_{down} stores nodes that one of the shortest AS paths from source to them is downhill path. One node can be in both sets simultaneously. In the initial state, node set S_{down} is empty, and node set S_{up} contains only one node: the source node s.

In the Nth step, we expand nodes in two sets according to the state machine illustrated in Fig.2. The detailed process is illustrated in Procedure 1. We execute Procedure 1 for several steps until all nodes in AS graph are explored.

Procedure 1:

clear node set T_{up} *and* T_{down}
for *each node* v in S_{up} **do**
  for *each neighbour* u of v **do**
    if u *has not been added into* S_{up} *yet* ***then***
      $d_{up}(s, u) \leftarrow N$
    end if
    if ($d_{up}(s, u) = N$) ***then***
      add node v *into node set predecessor(u,s)*
      if *type of edge (*v,u*) is c2p or s2s* ***then***
        add u *into node set* T_{up}
      else
        add u *into node set* T_{down}
      end if
    end if
  end for
end for
for *each node* v in S_{down} **do**
  for *each neighbour* u of v **do**
    if u *has not been added into* S_{down} *yet* ***then***
      $d_{down}(s, u) \leftarrow N$
    end if
    if $d_{down}(s, u) = N$ ***then***
      if *type of edge (*v,u*) is p2c or s2s* ***then***
        add v *into node set predecessor(u,s)*

add u *into node set* T_{down}
end if
end if
end for
end for
$S_{up} \leftarrow T_{up}$, $S_{down} \leftarrow T_{down}$

After the traversal stage, for each node *u* in AS graph, $d_{up}(s, u)$ and $d_{down}(s, u)$ have been calculated, and we get a subgraph stores in form of node sets *predecessor*(*u*, *s*). The subgraph only contains edges that are in one of the shortest AS paths from the source. For example, if we apply the traversal stage to the AS graph shown in Fig.1, and the source node is AS1, then Fig.3 shows the traversal steps and the subgraph we get.

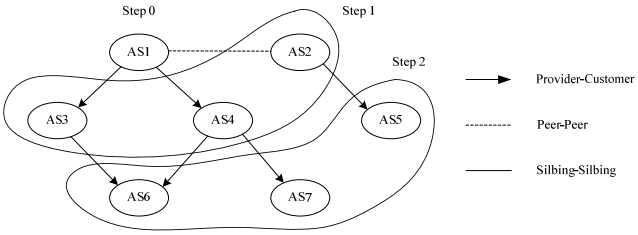

Figure 3. The traversal steps and the sub graph we get after the traversal stage

3.2 Traceback Stage

After the traversal stage, for each node *u* in AS graph, $d_{up}(s, u)$ and $d_{down}(s, u)$ have been calculated, and we get a subgraph stores in form of node sets *predecessor*(*u*,*s*). The subgraph only contains edges that are in one of the shortest AS paths from the source. Then in the traceback stage, we trace from each node back to the source node *s* in the subgraph to find all shortest AS paths from *s*. The detailed process of the traceback stage is illustrated in Procedure 2.

Procedure 2:
for *(each node v)* ***do***
clear stack path
depth $\leftarrow$ *min(*d_{up}*(s, v),* d_{down}*(s, v))*
tracenode*(s,v,*path,downhill,depth*)*
end for
Procedure **tracenode*(*source, target, path, pathtype, depth*)***
push target $\rightarrow$ path
if *(*source *is* target*)* ***then***
output path
return
end if
depth $\leftarrow$ depth *-1*
for *each node u in predecessor(*target,*s)* ***do***
if $d_{up}(s, u) \neq$ depth *and* $d_{down}(s, u) \neq$ depth ***then***
continue
end if
if pathtype=uphill ***then***

```
    if type of edge (u,target) is p2c or p2p then
      continue
    end if
  else
    if type of edge (u,target) is p2p or c2p then
      pathtype←uphill
    end if
  end if
  tracenode(source, u, path, pathtype, depth)
end for
```

4. Simulation and Validation

To evaluate the accuracy and performance of our algorithm, we use the data sets provided by CAIDA[15] to construct AS graph and annotate AS relationships. CAIDA uses the RouteView[16] daily data to extract AS links and the algorithm proposed by Dimitropoulos[13] to infer AS relationships.

First, we use the data set of 2007-8-6 provided by CAIDA to evaluate the accuracy of our algorithm. The data set contains 25697 nodes and 105524 edges. We apply our algorithm on the data set to infer all pair shortest AS paths, and compare them with the AS paths inferred by MAO's algorithm. We find that the AS paths inferred by the two algorithms are all the same.

Then, we select seven data sets from 2004-02-02 to 2007-02-05 provided by CAIDA to evaluate the performance of our algorithm. We apply the two algorithms on the seven data sets separately on the same computer and compare the time they use in Table 1. From Table 1, we can see that our algorithm uses much less time to infer all pair shortest AS paths.

Table 1. Performance comparison of the two algorithms

Data set	2004.02.02	2004.08.02	2005.02.07	2005.08.01	2006.02.06	2006.08.07	2007.02.05
Nodes	16493	17665	18911	20037	21343	22706	24142
Edges	66744	72140	75982	80948	86850	92826	98400
Time Used by Our Algorithm(ms)	297	344	328	390	422	485	562
Time Used by Mao's Algorithm(ms)	63735	70594	81641	88140	102547	115360	131422

5. Conclusions

We propose an efficient algorithm for inferring all pair shortest AS paths from a relationship annotated AS graph. The running time of our algorithm is O (NM), where N is the number of nodes and M is the number of links in AS graph. The algorithm bases on the bread-first-search (BFS) algorithm, and our evaluation shows that it reduces running time dramatically compared with previous algorithm.

Reference

[1] A.B. Smith, C.D. Jones, and E.F. Roberts, "Article Title", Journal, Publisher, Location, Date, pp. 1-10.
[2] Z. M. Mao, J. Rexford, J. Wang, and R. H.Katz. Towards an accurate as-level traceroute tool. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. ACM Press, 2003, pp. 365– 378.
[3] Z. Mao, L. Qiu, J. Wang, and Y. Zhang. On as-level path inference. SIGMETRICS, Banff, Alberta, Canada, 2005.
Lixin Gao. On Inferring autonomous system relationships in the internet. IEEE/ACM Trans on Networking, 2001,9(6):733-745.
[4] CAIDA. http://www.caida.org.
[5] H. Chang, S. Jamin, and W. Willinger. Inferring as-level Internet topology from router-level path traces. In Proceedings of the SPIE ITCom, 2001.
[6] R. Cohen, D. Raz. Acyclic Type of Relationships Between Autonomous Systems. In IEEE *INFOCOM*, 2007.
[7] Giuseppe Di Battista, Thomas Erlebach, et al. Computing the Types of the Relationships between Autonomous Systems. IEEE/ACM Transactions on Networking. 15(2):267-280. Apr 2007.
[8] Xenofontas Dimitropoulos, George Riley. Modeling Autonomous-System Relationships. In Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation. 2006.
[9] T. Erlebach, A. Hall, and T. Schank. Classifying customer-provider relationships in the Internet. In Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN), 2002.
[10] X. Dimitropoulos, D. Krioukov, B. Huffaker, kc claffy, and G. Riley. Inferring AS relationships: Dead end or lively beginning? In Proceedings of 4th Workshop on Efficient and Experimental Algorithms (WEA' 05), May 2005.
[11] J. Xia and L. Gao. On the evaluation of AS relationship inferences. In IEEE GLOBECOM, 2004.
[12] Jian Qiu, Lixin Gao. AS Path Inference by Exploiting Known AS Paths. In IEEE GLOBECOM, San Francisco,USA, 2006.
[13] X. Dimitropoulos, D. Krioukov, M. Fomenkov, and B. Huffaker, As relationships: Inference and validation. ACM SIGCOMM Computer Communications Review, vol. 37, no. 1, pp. 29–40, 2007.
[14] S. Kosub, M. G. Maaß, and H. Taubig. Acyclic type-of-relationship problems on the internet. In The3rd Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN'06), volume 4235 of LNCS, pages 98–111. Springer-Verlag, July 2006.
[15] The CAIDA AS Relationships Dataset. http://www.caida.org/data/active/as-relationships/.
[16] Route Views Project. http://www.routeviews.org.

Authors

Yang Guoqiang, born in 1981. Received B.A's and M.A's degree in computer science from the National University of Defense Technology, Changsha, China, in 2003 and 2005 respectively. Since 2005, he has been a Ph.D. degree candidate in computer science from the National University of Defense Technology. His current research interests include Internet topology modeling and Internet routing technology.

Dou Wenhua, born in 1946. He is a professor in the National University of Defense Technology. His main research interests are high performance computing and advanced computer network .