Acyclic Type of Relationships Between Autonomous Systems

Rami Cohen
Computer Science Department, Technion, IIT
Email: ramic@cs.technion.ac.il

Danny Raz
Computer Science Department, Technion, IIT
Email: danny@cs.technion.ac.il

***Abstract*— The Internet connectivity in the Autonomous System (AS) level reflects the commercial relationship between ASes. A connection between two ASes could be of type *customer-provider* when one AS is a provider of the other AS, or of type *peer-peer*, if they are peering ASes. This commercial relationship induces a global hierarchical structure which is a key ingredient in the ability to understand the topological structure of the AS connectivity graph. Unfortunately, it is very difficult to collect data regarding the actual type of the relationships between ASes, and in general this information is not part of the collected AS connectivity data. The Type of Relationship (*ToR*) problem attempts to address this shortcoming, by inferring the type of relationship between connected ASes based on their routing policies. However, the approaches presented so far are local in nature and do not capture the global hierarchical structure.**

In this work we define a novel way to infer this type of relationship from the collected data, taking into consideration both local policies and global hierarchy constrains. We define the Acyclic Type of Relationship *AToR* problem that captures this global hierarchy and present an efficient algorithm that allows determining if there is a hierarchical assignment without invalid paths. We then show that the related general optimization problem is NP-complete and present a $\frac{2}{3}$ approximation algorithm where the objective function is to minimize the total number of local policy mismatches. We support our approach by extensive experiments and simulation results showing that our algorithms classify the type of relationship between ASes much better than all previous algorithms.

I. Introduction and Related Work

The current Internet consists of over 20000 Autonomous Systems (ASes) interconnected by a set of thousands links. Each AS is a collection of routers under a single administrative authority, and routing between ASes is done using the Border Gateway Protocol (BGP) [1]. One of the well appreciated advantages of BGP is its ability to use policy based routing where each AS defines its own local policy. In practice, the policy of an AS reflects its commercial relationship with other ASes. Thus, the AS connectivity graph has a hierarchical structure in which connected ASes have *customer-provider* relationship if a small AS is connected to a larger AS, and they have *peer-peer* relationship if they have comparable size (other types of relationship such as *sibling-sibling* also exist, but they apply to less than 2% of the connections) [2], [3].

Despite the increasing effort to reveal and characterize the topological structure of the Internet by several projects that collect real up-to-date data, the hierarchical structure, induced by the commercial relationship between connected ASes, is typically not part of the collected information. Thus, in order to get this more complete view, one should infer these relationships from the collected information. This is usually done using guidelines and assumptions regarding the policy used and knowledge regarding the gathered information. For instance, the Internet Routing Registry [4] is a union of world-wide routing policy databases that use the Routing Policy Specification Language (RPSL) [5] . These databases contain, among other things, the local connectivity and the local import/export policy of the registered ASes. In [6] and [7] the authors analyzed the RPSL policies of ASes in the IRR and inferred the type of relationship between registered ASes. Nevertheless, using the IRR database to infer the hierarchical structure of the AS connectivity map has several drawbacks. First, in some cases, entries in the IRR may be invalid and contain out-of-date data [8]. Second, this database is not complete enough. In particular, only 36000 links, most of them are located in Europe, out of over 130000 [7] are fully covered by this database.

While the IRR database contains the local policy of registered AS, it is not part of the information gathered by other projects. In these cases, other techniques should be used in order to infer the type of relationship. The Route-Views project [9] is a BGP based database that collects a snapshot of the Internet AS level topology on a daily basis, based on BGP routing tables from about 60 different sources. The DIMES project [10] samples the Internet using distributed agents located at thousands hosts around the world, performing periodic *traceroute* to a set of IP addresses. While these projects gather information using different methods, the collected data of both databases consists of a set of routing paths (between ASes) that reflect the routing policy of these ASes. In order to infer the type of relationship from such routing paths, one should understand how the policy in the AS level affects these paths.

According to the guidelines presented in [3] and in [2] an AS usually exports its routes and its customer routes to its providers and peers, but it does not export its provider or peer routes to other providers or peers. In contrast, an AS usually exports its routes and its customer routes, as well as all its provider or peer routes to its customers and sibling. This policy indicates that BGP paths are valley-free, and step-free, i.e. after traversing a *provider-customer* or a *peer-peer* link, the path cannot traverse a *customer-provider* or *peer-peer* link [11].

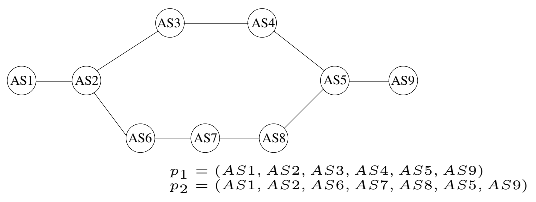

AS1
AS2
AS3
AS4
AS5
AS6
AS7
AS8
AS9
$p_1 = (AS1, AS2, AS3, AS4, AS5, AS9)$
$p_2 = (AS1, AS2, AS6, AS7, AS8, AS5, AS9)$

Fig. 1. A *ToR* instace with two paths

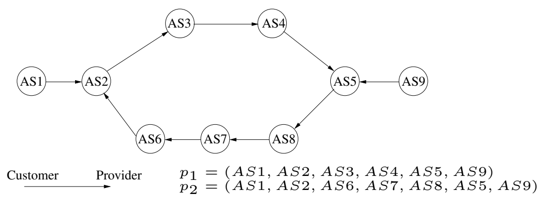

AS1
AS2
AS3
AS4
AS5
AS6
AS7
AS8
AS9
Customer
Provider
$p_1 = (AS1, AS2, AS3, AS4, AS5, AS9)$
$p_2 = (AS1, AS2, AS6, AS7, AS8, AS5, AS9)$

Fig. 2. A *ToR* instace - A possible solution

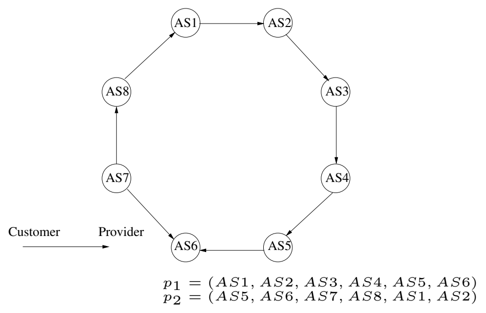

AS1
AS2
AS3
AS4
AS5
AS6
AS7
AS8
Customer
Provider
$p_1 = (AS1, AS2, AS3, AS4, AS5, AS6)$
$p_2 = (AS5, AS6, AS7, AS8, AS1, AS2)$

Fig. 3. A *ToR* instace - every solution contains cycle

Gao [11] was the first to infer the AS relationships from BGP routing tables, based on this valley-free nature of the routing paths. She developed a heuristic algorithm assuming that typically a provider has a larger size than its customer, and the size of an AS is usually proportional to its degree in the AS level connectivity graph. Thus, for each routing path, the AS with the highest degree is set as a *top provider* with respect to the path, inducing *customer-provider* relationship to preceding and subsequent links in the path. The experimental results of [11] indicate that 90% of the links in the Route-Views database are of type *customer-provider*, 8% are of type *peer-peer*, and 1.5% are of type *sibling-sibling*.

Subsequently, the Type of Relationship *ToR* problem was formally defined in [12] as a maximization problem:

Definition 1.1: Given an undirected graph $G = (V, E)$, and a set of paths P, label the edges in E as either $-1, 0$ or $+1$ to maximize the number of valid paths in P, where a valid path can be one of the following types for $M, N \geq 0$:

1) $-1, ...$ (N times), $+1, ...$ (M times).
2) $-1, ...$ (N times), $0, +1, ...$ (M times).

Here -1 indicates a *customer-provider* edge, 0 indicates a *peer-peer* edge, and $+1$ indicates a *provider-customer* edge.

For example, consider the instance of the *ToR* problem depicted in Fig. 1. This instance consists of nine ASes and two paths. A possible solution to that instance, containing only valid paths, is depicted in Fig. 2[1]. In this solution both paths are of type 1, namely they consist of several *customer-provider* links followed by several *provider-customer* links.

The technique proposed in [12] to solve the *ToR* problem combines data from multiple vantage points, where each BGP routing table gives partial view of the Internet from one AS. This technique does not rely on the degree of the ASes. The authors ranked the ASes based on their position in the graph, induced by a single BGP routing table. Then they infer the relationship by comparing the ranks of ASes as it derived from multiple sources.

[1]A directed edge in the graph going from node v to node u means that v is a customer of u.

In [13] the authors showed that the decision version of the *ToR* problem is NP-complete in the general case. Moreover, they presented a linear time algorithm that determines if there is a fully valid solution (i.e without any invalid path). This algorithm maps a *ToR* instance into a *2SAT* formula by converting every two consecutive edges in any of the paths into a clause with two literals. Finding a Truth assignment to this formula induces a valley-free solution, while if the formula cannot be satisfied the *ToR* instance must contain at least one valley. They also proved that the maximum version of the problem (i.e. maximizing the number of valid paths) cannot be approximated within $1/n^{1-\epsilon}$ (for any $\epsilon > 0$) for general instances with n paths unless NP=co-RP. This is done using approximation-preserving polynomial reduction from the Maximum Independent Set problem [14].

The formal definition of the *ToR* problem indeed captures the fact that BGP paths are valley-free and step-free. However, it has an inherent drawback - it does not consider the hierarchical structure of the AS graph. In particular, in the real Internet the directed graph imposed by the assignment of the *customer-provider* relationship can **not** contain cycles (see [15], [16]). A national AS, for example, that provides services to a regional AS that provides services to a local AS cannot be also the customer of the local AS. Consider the solution to the *ToR* instance depicted in Fig. 2. In this solution both paths are valley-free, however it contains a directed cycle, violating the hierarchical structure of the graph. One can achieve an acyclic solution to this specific instance, by changing the direction of the edge $(AS4, AS5)$. In contrast, every optimal solution to the *ToR* instance depicted in Fig. 3 contains a cycle.

In this paper we address this drawback by defining a new problem, the Acyclic Type of Relationship (*AToR*) problem, taking into account the acyclic structure of the AS connectivity graph. In this case, given a set of routing paths, the objective function is to maximize (or minimize) the number of valid (invalid) paths, keeping the directed graph acyclic. This new problem captures the type of relationship between connected ASes more accurately. Note, that while these two problems look similar, their analysis is quite different. In particular, in the *ToR* problem one should only satisfy local conditions in which every two consecutive edges in all the paths should be valley-free. On the other hand, in the new *AToR* problem, in addition to these local conditions, one should satisfy a more global condition ensuring that the assignment is acyclic. For that reason, techniques and algorithms that have been used with respect to the *ToR* problem, cannot be adopted and used to analyze and solve the *AToR* problem. A very similar problem, also termed AToR, was independently defined and studied by Kosub et al. [17]. This work, done in parallel to ours, studies pure theoretical aspects of this problem.

In Section II we formally define the *AToR* problem. Then, in Section III we present an efficient algorithm determining whether an acyclic solution without invalid paths exists. The general case is discussed in Section IV. We consider a variant of this problem in which the objective function is to minimize the total number of valleys. This variant captures the fact that in some cases the export policy executed by an AS does not follow the export policy presented above. Thus, due to the locality of the policy, paths that traverse such ASes may contain valleys. We show that similar to the original problem (in which the objective function is to maximize the number of valid paths), the decision version of this variant of the problem is NP-hard, and we present $\frac{2}{3}$-approximation algorithm for the maximum version of the problem. In Section V we consider practical aspects of inferring the actual type of relationships between ASes. This includes heuristics to infer also *peer-peer* and *sibling-sibling* relationships. In Section VI we examine our algorithms over real up-to-date data gathered from the Route-Views database, and perform simulations over several random graph. We also compare our algorithms to other approaches presented in [13], [12], [11]. We summarize our work and present some conclusions in Section VII.

II. Model and Problem Definition

The AS connectivity map is modelled by a graph $G = \{V, E\}$. A node in the graph represents an AS, and an edge represents a peering relation between two ASes. Assigning an orientation to a particular edge in the graph indicates the business relationship between its corresponding ASes. Thus, an edge (v, u) is directed from v to u if v is a customer of u (and respectively u is a provider of v). On the other hand, an undirected edge indicates that the corresponding ASes are connected by *peer-peer* relationship. As explained in Section I, a directed cycle that contains at least one directed edge (i.e. a *customer-provider* edge) violates the hierarchical structure of the graph. On the other hand, ASes from the same hierarchy level can be connected by *peer-peer* links. Thus, cycles that consist of undirected edges alone (i.e. all the links composing these cycles are of type *peer-peer*), implying that the participants ASes are in the same hierarchy level, are permitted. With respect to this observation, we use the term *hierarchical cycle* to describe a directed cycle that contains at least one directed edge. We use the term *valid path* to indicate that the path does not contain valleys nor steps. Thus, $p = \{v_1, v_2, ..., v_n\}$ is a valid path if for all $1 < i < n$, the edge (v_{i-1}, v_i) is directed from v_{i-1}, to v_i or the edge (v_{i+1}, v_i) is directed from v_{i+1}, to v_i. A path is *invalid* if it is not *valid*. Considering the hierarchy structure of the AS graph and using these policy guidelines, we define the *AToR* (Acyclic Type of Relationship) problem as follows:

Definition 2.1: Given an undirected graph G and a set of paths P, assign orientation to some of the edges of G such that the directed graph does not contain *hierarchical cycles* (i.e. directed cycles that contains at least one directed edge), and the number of valid paths is maximized.

One may observe that *sibling-sibling* edges are not part of this definition. In Section V we show how to refine our algorithm to include such edges. In some cases an instance to the *AToR* problem includes only the set of paths P while the graph G is omitted. In such cases, one may consider a graph $G' = (V', E')$ that is imposed by the set of paths, namely $V' = \{v | v \in P\}$, and $E' = \{e | e \in P\}$. The decision version of the problem, *k-AToR*, is defined as follows:

Definition 2.2: Given an undirected graph G, a set of paths P, and an integer k, test if it is possible to give orientation to some of the edges of G such that the directed graph does not contain *hierarchical cycle*, and the number of invalid paths is at most k.

In sections III and IV we present theoretical analysis both for the *AToR* and the *k-AToR* problems. In these analyses we consider solutions that contain only directed edges (i.e. edges of type *customer-provider*). One can argue that this requirement is stricter than necessary and therefore does not reflect the real practical problem. Yet, as we prove in the next lemma, every solution that contains *peer-peer* links can be converted to a solution that contains only *customer-provider* links by giving an orientation to the *peer-peer* links. Clearly, in a solution that contains only directed edge, every directed cycle is a *hierarchical cycle* and acyclic solution is a solution that does not contain *hierarchical cycles*.

Lemma 2.1: Given an instance (G, P) to the *AToR* problem and a solution S that assigns an orientation to some of the edges of G, such that the directed graph does not contain *hierarchical cycles* and the number of valid paths is k, there is a solution S' that assigns an orientation to **all** the edges of G, such that the directed graph does not contain directed cycles and the number of valid paths is at least k.

Proof: Denote by E_1 the set of edges that have an orientation with respect to S, and denote by $G_1 = (V, E_1)$ the graph imposed by this set of edges. Clearly, G_1 does not contain directed cycles. We build S' gradually by assigning orientation to the set of undirected edges $E \backslash E_1$ (i.e. converting the set of *peer-peer* links into *customer-provider* links). We show that each such step does not reduce the number of valid paths and preserves the acyclic property of the graph.

Consider a *peer-peer* link $e = (v, u)$, namely $e \in E \setminus E_1$. The graph G_1 does not contain directed cycles therefore, if there is a directed path $p = (v, ..., u)$ then there is no directed path $p' = (u, ..., v)$ (otherwise, their concatenation is a directed cycle in contradiction to the assumption)[2]. Thus, if there is a directed path from v to u we assign e from v to u. Otherwise, we assign e from u to v. In both cases, after

[2]Note that p and p' are not necessarily in P.

adding e to E_1, the graph G_1 is still acyclic.

Next, we show that every valid path remains valid by assigning direction to a link e. Clearly, paths that do not traverse e are not affected so we need to show that every valid path that traverses e remains valid. Consider a valid path that traverses e. According to the discussion above, this path consists of N *customer-provider* links followed by e which is a *peer-peer* link followed by M *provider-customer* links (where $N, M \geq 0$). If e is assigned to be a *customer-provider* links then the new path consists of $N+1$ *customer-provider* links followed M *provider-customer*. On the other hand if e is assigned to be a *provider-customer* links then the new path consists of N *customer-provider* links followed $M+1$ *provider-customer*. In both cases the new path is valid. ■

III. The *0-AToR* Problem

In this section we present an efficient algorithm for the *0-AToR* problem. In other words, given an instance (G, P) to the *AToR* problem, the algorithm determines if there is a solution without any invalid paths. In such case, the algorithm finds such a solution. For simplicity, we present and analyze the algorithm over a special case of the *0-AToR*, called the *0-AToR2* problem, in which the length of all paths is exactly two links. Although this version seems to be simpler than the *0-AToR* problem, it does not. In particular, in the following lemma we show that the complexity of the *0-AToR2* problem is identical to the complexity of the *0-AToR* problem[3].

Definition 3.1: Given an instance (G, P) to the *0-AToR* problem, we say that there is a *Satisfying* assignment to (G, P) if there is an orientation to the edges such that the directed graph is acyclic and all the paths are valley-free.

Lemma 3.1: Given an instance (G, P) to the *0-AToR* problem, there is an instance (G, P_2) to the *0-AToR2* problem such that there is a *Satisfying* assignment to (G, P) if and only if there is a *Satisfying* assignment to (G, P_2).

Proof: P_2 is generated in the following way: for each $p = \{AS_1, AS_2, ...AS_n\} \in P$ produce $n-2$ paths $p_1, ...p_{n-2}$ such that $p_i = \{AS_i, AS_{i+1}, AS_{i+2}\}$. For each assignment, if the path p_i is invalid than the path p is invalid as well (it contains valley in AS_i, AS_{i+1}, AS_{i+2}). On the other hand, if the path p is invalid, at least one of the paths $p_1, ...p_{n-2}$ is invalid (if p contains valley at $(AS_i, AS_{i+1}, AS_{i+2})$ then the path p_i contains a valley as well). Clearly, an acyclic assignment to (G, P), induces an acyclic assignment to (G, P_2) and vice versa, since both instances consist of the same graph. ■

As discussed above, the directed graph imposed by a solution to an instance of the *0-AToR2* problem does not contain directed cycles. Thus, one can present this solution as an ordering π of the vertices of a graph such that for each directed edge $(v_i, v_j) \in E$, going from v_i to v_j, $\pi(v_i) < \pi(v_j)$. Clearly, the directed graph induced by such ordering is acyclic. Moreover, this solution does not contain valleys, namely for each path $p = (v_i, v_j, v_k)$ $\pi(v_j) > \pi(v_i)$ or $\pi(v_j) > \pi(v_k)$. The following algorithm determines if such an ordering exists.

```
Algorithm ATOR($G = (V, E), P = (p_1, ..., p_n)$)

 1.  $P_2 = \phi$
 2.  For all $p_i = \{AS_1, AS_2, ...AS_m\} \in P$.
 3.   for($i = 1$ to $m - 2$)
 4.     $P_2 = P_2 \cup \{AS_i, AS_{i+1}, AS_{i+2}\}$

 5.  i=1
 6.  For all $v \in V, \pi(v) = -1$.

 7.  while $P_2 \neq \phi$ do
 8.    Find $v \in V$ such that
            $\forall p = (v_i, v_j, v_k) \in P_2$, $v \neq v_j$
 9.    If such $v$ does not exist
         return NO SOLUTION.
10.    set $\pi(v) = i$.
11.    i=i+1.
12.    $P_2‘ = \{p | v \in p\}$.
13.    $P_2 = P_2 \setminus P_2‘$

14.  while $i \leq |V|$
15.    if $\pi(v) = -1$, set $\pi(v) = i$.
16.    $i = i + 1$.

17.  return $\pi$.
```

In the first stage (steps 1 to 4) the algorithm generates a set of paths P_2 according to the construction described in Lemma 3.1, such that the length of each path in P_2 is exactly two and (G, P) has a *Satisfying* assignment if and only if (G, P_2) has a *Satisfying* assignment. Steps 5 and 6 are for initialization. Then, in every iteration the algorithm finds a vertex that does not appear in the middle of any path. Giving this vertex the current lowest value in the ordering insures that the associated paths are valid. On the other hand, if such vertex does not exist, it means that at least one path contains this vertex in the middle. Thus, giving this vertex the current lowest value in the ordering makes this path invalid.

In each iteration at least one path is removed (steps 12 and 13), and thus at most $|P_2|$ iterations are performed. In each iteration, the algorithm goes through the vertices and pick one vertex (Step 8). Moreover, $|P_2| = |P| \cdot (N-2)$ where N is the average length of a path in P. Therefore, the time complexity of the algorithm is $O(|P| \cdot N \cdot |V|)$.

If the algorithm finds a solution (i.e. it does not return *NO SOLUTION*) the peering relationships are induced as follow: For each edge (v, u) in the graph, v is a customer of u (and respectively u is a provider of v) if and only if $\pi(v) < \pi(u)$. Next we show the correctness of the algorithm.

If an instance (G, P) of the *0-AToR* problem has a *Satisfying* assignment, clearly a sub-instance (i.e. an instance that consists of a subset of P) has a *Satisfying* assignment as well. Thus,

Observation 3.1: Given an instances (G, P) and (G, P') to the *0-AToR* problem, such that $P' \subseteq P$. If (G, P') does not have a *Satisfying* assignment then (G, P) does not have a

[3]Recal that in this section we consider solutions that assign an orientation to all the edges. In this case, a valid path is a valley-free path (i.e. a path that does not contain valleys).

Satisfying assignment.

Given an instance $(G = (V, E), P)$ of the *0-AToR* problem, denote by $G_p = (V_p, E_p)$ the graph imposed by P (i.e. $V_p = \{v | v \in P\}$ and $E_p = \{e | e \in P\}$). Clearly, the ordering of any vertex v such that $v \in V \setminus V_p$ does not affect the validity of any path (since this vertex does not appear in any path), thus:

Observation 3.2: Given an instance (G, P) to the *0-AToR* problem. (G, P) has a *Satisfying* assignment if and only if (G_p, P) has a *Satisfying* assignment.

Theorem 3.1: Given an instance (G, P) to the *0-AToR* problem, if Algorithm *ATOR* returns ordering π of the vertices of a graph, then this ordering induces a *Satisfying* assignment.

Proof: Clearly, the directed graph induced by the ordering is acyclic. We show that this directed graph is valley-free with respect to the set of paths P_2. Without loss of generality, assume that $p = (v_x, v_y, v_z)$ is removed from P_2 in the i'th iteration. According to steps 8 and 10 of the algorithm $\pi(v_x) = i$ or $\pi(v_z) = i$. Moreover, $\pi(v_y)$ was not set yet, thus $\pi(v_y) > i$ and therefore p is valley-free.

Now, recall that P_2 is constructed according to Lemma 3.1, thus the directed graph imposed by the ordering π, is valley-free with respect to the set of paths P as well. ■

Theorem 3.2: Given an instance (G, P) of the *0-AToR2* problem, if Algorithm *ATOR* returns *NO SOLUTION*, then there is no *Satisfying* assignment.

Proof: Without loss of generality, assume that the algorithm returns *NO SOLUTION* in the i'th iteration and the set of paths that was removed in the first $i - 1$ iterations is P_2''. Denote by P_2' the remaining paths, namely $P_2' = P_2 \setminus P_2''$. We show that the instance (G', P_2'), does not have a *Satisfying* assignment, where $G' = (V', E')$ is a subgraph of G in which $V' = \{v | v \in P_2'\}$ and $E' = \{(v, u) | (v, u) \in E \bigcap v, u \in V'\}$. Assume that there is an ordering π that induces a *Satisfying* assignment. Denote by v the node with $\pi(v) = 1$. Recall that $v \in V'$, therefore $v \in P_2'$. According to steps 8 and 9 in the algorithm, $\forall v \in P_2', \exists p = (v_x, v_y, v_z) \in P_2'$ such that $v \equiv v_y$. Thus, since $\pi(v_x) > 1$, $\pi(v_z) > 1$, and $\pi(v_y) = 1$, p contains a valley, so (G', P_2') does not have a *Satisfying* assignment. According to Observation 3.2, (G, P_2') does not have a *Satisfying* assignment and according to Observation 3.1 (G, P_2) does not have a *Satisfying* assignment. Again, recall that P_2 is constructed according to Lemma 3.1, thus the instance (G, P) does not have a *Satisfying* assignment as well. ■

Discussion: One may notice that the *AToR2* problem resembles to the well known *BETWEENNESS* problem, studied in the field of Computational Biology [18]. The input to the *BETWEENNESS* problem consists of a set of points $S = \{x_1, x_2, ..., x_n\}$ and a set of constraints, where each constraint is a triplet $\{x_i, x_j, x_k\} \in S \times S \times S$. A solution to the problem is a total order on its points such that every constraint $\{x_i, x_j, x_k\}$ satisfies $x_i < x_j < x_k$ or $x_k < x_j < x_i$, namely x_j is *between* x_i and x_k (see [18]). With respect to our problem, the set of triplet corresponds to the set of paths with length two. While in the *BETWEENNESS* problem x_j must be between x_i and x_k, in the *AToR* problem x_j can be between x_i and x_k or bigger than x_i and x_k (i.e. x_j must not be lower than x_i and x_k). Despite the appeared similarity between the problems their complexity is completely different. Thus, while determining if there is a *satisfying* assignment to the *AToR2* problem (and finding such an assignment) can be done in polynomial time (using the algorithm described above), determining if there is a *satisfying* assignment to the *BETWEENNESS* problem is NP-hard [19]. The intuition behind this difference is the fact that in the *AToR* problem, determining the ordering of one edge of a path satisfies this path (i.e. the path is valid). On the other hand determining the ordering of single point in a triplet, does not necessarily satisfy this triplet, in the *BETWEENNESS* problem.

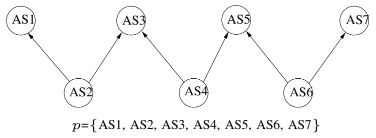


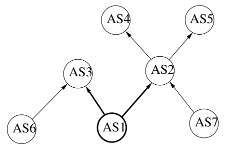
Fig. 4. One invalid path - Multiple valleys

IV. THE *k-AToR* PROBLEM

While determining if there is a solution without invalid paths (i.e. finding a solution to the *0-AToR* problem) can be done in polynomial time, the general case is much more complex. In particular, the reduction presented in [13] for the *ToR* problem holds for the *AToR* problem. Thus, the general decision version (called the *k-AToR* problem) is NP-hard and the maximum version of the problem (i.e. maximizing the number of valid paths) cannot be approximated within $1/n^{1-\epsilon}$ (for any $\epsilon > 0$) for general instances with n paths unless NP=co-RP.

In this section we consider a variant of the problem in which the objective function is to minimize the total number of valleys. This problem is different from the original problem since one invalid path may contain more than one valley (see Fig. 4), and on the other hand, if several paths traverse a common AS, one valley (in the common AS) may cause several invalid paths (see Fig. 5). The objective function of this variant of the problem is motivated by the fact that in some cases ASes (mistakenly or purposely) do not follow the export policy discussed in Section I. In particular, in such cases, an AS may export its provider routes to other providers. Thus, due to the locality of the policy, paths that traverse such ASes may contain valleys. In the extended version of this paper we prove that the decision version of this variant of the problem is NP-hard using a reduction from the Feedback Vertex Set (FVS) problem [20], [14]. Here we present a $\frac{2}{3}$-approximation algorithm to the maximum version of the problem.

Consider an instance of the *k-AToR* problem in which all the paths contain exactly two edges (we refer this problem as *k-AToR2* problem). In this case an invalid path contains exactly one valley and therefore if there is no duplicated paths, every valley corresponds to one invalid path. Thus, given a graph G and a set of paths P, we build an instance (G, P_2) to the *k-AToR2* such that for any assignment, the number of



p_1 ={AS7, AS2, AS1, AS3, AS6}
p_2 ={AS5, AS2, AS1, AS3}
p_3 ={AS4, AS2, AS1, AS3}

Fig. 5. One valley - Multiple invalid paths

valleys with respect to the set of paths P is equal to the number of invalid paths with respect to the set of paths P_2. Thus, for each $p = \{AS_1, AS_2, ...AS_n\} \in P$ we produce $n-2$ paths $p_1, ...p_{n-2}$ such that $p_i = \{AS_i, AS_{i+1}, AS_{i+2}\}$, then we remove all the duplicate paths. In this case, given an assignment of the edges in G, every valley in the edges (v_1, v_2) and (v_2, v_3) with respect to P, corresponds to invalid path (v_1, v_2, v_3) with respect to P_2. Based on this observation, henceforth we consider only the *k-AToR2* problem.

Next, we present a $\frac{2}{3}$-approximation algorithm to the *k-AToR2* problem. For each vertex $v \in V$, denote by E_v the set of the paths such that v is an end point of each path in the set. Similarly denote by M_v the set of the paths such that v is in the middle of each path in the set. Formally, $E_v = \{p | p \in P, p = \{v, u, w\} \cup p = \{u, w, v\}\}$, $M_v = \{p | p \in P, p = \{u, v, w\}\}$.

```
Algorithm k-ATOR2($G = (V, E), P = (p_1, ..., p_n)$)

 1.  $P_2 = \phi$
 2.  For all $p_i = \{AS_1, AS_2, ...AS_m\} \in P$.
 3.   for($i = 1$ to $m-2$)
 4.     $P_2 = P_2 \cup \{AS_i, AS_{i+1}, AS_{i+2}\}$

 5.  i=1
 6.  For all $v \in V, \pi(v) = -1$.

 7.  while $P_2 \neq \phi$ do
 8.     Let $v$ be the vertex in $V$ such that
          $\forall u \in V, |M_v|/|E_v| \leq |M_u|/|E_u|$
 9.     set $\pi(v) = i$.
10.     i=i+1.
11.     $P_2‘ = \{p | v \in p\}$.
12.     $P_2 = P_2 \setminus P_2‘$

13.  while $i \leq |V|$
14.     if $\pi(v) = -1$, set $\pi(v) = i$.
15.     $i = i + 1$.

16.  return $\pi$.
```

The algorithm is similar to the algorithm presented in Section III. In particular, the algorithm presented in Section III is a special case of this algorithm in which the vertex, selected in Step 8 must follow $|M_v|/|E_v| = 0$ (i.e. $|M_v| = 0$)[4]. Next we prove that this algorithm is a $\frac{2}{3}$-approximation algorithm.

[4]Note that E_v and M_v are defined with respect to the set of paths P_2 in the algorithm.

Lemma 4.1: The vertex v that is selected in Step 8 fulfils $|M_v|/|E_v| \leq \frac{1}{2}$.

Proof: In every path $p = \{v_1, v_2, v_3\}$ two vertices (v_1 and v_3) are end points and one vertex (v_2) is in the middle. Thus, $|P| = \sum_{v \in V} |M_v| = \frac{1}{2} \sum_{v \in V} |E_v|$, and at least one vertex $v \in V$ satisfies $|M_v|/|E_v| \leq \frac{1}{2}$ and therefore the vertex selected in Step 8 fulfil $|M_v|/|E_v| \leq \frac{1}{2}$. ■

Theorem 4.1: Given an instance (G, P_2) of the *k-AToR2* problem, the Algorithm *k-ATOR2* returns an ordering π such that the directed graph induced by this ordering is acyclic and the total number of valley-free paths P_{valid} is at least $\frac{2}{3} \cdot |S^{opt}|$, where S^{opt} is the optimal solution.

Proof: The algorithm returns an ordering over the vertices, thus the directed graph induced by this ordering is acyclic. Assume that $p = (v_x, v_y, v_z)$ is removed from P_2 in the i'th iteration. If $p \in E_v$ then according to Step 9 in the algorithm $\pi(v_x) = i$ or $\pi(v_z) = i$. Moreover, $\pi(v_y)$ was not set yet, thus $\pi(v_y) > i$ and therefore p is valley-free. On the other hand, if $p \in M_v$ then $\pi(v_y) = i$ while $\pi(v_x)$ and $\pi(v_z)$ were not set yet. In this case $\pi(v_x), \pi(v_z) > i$ and p is an invalid path. According to this observation, in each iteration, $|E_v|$ paths turn to be valid paths while $|M_v|$ paths turn to be invalid paths. According to Lemma 4.1 $|M_v|/|E_v| \leq \frac{1}{2}$ therefore the total number of valid paths is at least double the number of invalid paths. In other words, $P_{valid} \geq 2(|P_2| - P_{valid})$ and hence $P_{valid} \geq \frac{2}{3} \cdot |P_2| \geq \frac{2}{3} \cdot |S_{opt}|$. ■

One may observe that the proof is independent of the value of $|S^{opt}|$. Thus, the theorem proves a stronger result. In particular, the algorithm guarantees that at least $\frac{2}{3}$ of the paths are valley-free regardless on the optimal solution.

While we consider an objective function that minimizes the total number of valleys, other objective targets may be considered as well. In particular, one may consider an objective target that minimizes the total number of directed cycles. This objective target may be interesting from a theoretical point of view, but it seems that it does not have a practical interest. While invalid paths and valleys may appear in the internet for several reasons (e.g. export policy misconfiguration) the hierarchical acyclic structure of the internet is a result of the business relationship between ASes. Large ASes always provide services to small AS and this ordering imposes an acyclic structure.

V. Practical Consideration

In this section we show how the theoretical algorithms presented in sections III and IV can be used to solve the practical problem of inferring the correct type of relationships from the collected data. In practice we may have to choose between more than one solution and we also have to consider other type of relationships, namely *peer-peer* and *sibling-sibling* that were not considered in these algorithms.

Consider the algorithms presented in sections III and IV. While these algorithms find a specific ordering, the space of possible solutions may contain more than one ordering. In particular, in Step 8 of both algorithms, more than one vertex

AS1
AS2
AS3
AS4
AS5
AS6
AS7

Fig. 6. Space of Solutions

may satisfy the required condition (i.e. minimizing $|M_v|/|E_v|$) and therefore different orders found by the algorithm may induce different peering relationships. In particular, the ordering found by the algorithm may induce peering relationship that are different from the real relationships. To obtain a more accurate solution, one should use heuristics and guidelines that consider the topological structure of the graph and the gathering process of the data[5].

Considering the fact that the overwhelming majority of ASes are small[6], most of the routing paths obtained by BGP routing tables are between small ASes (i.e. the source and the destination of the path are small ASes). Thus, most of the routing paths consist of several *customer-provider* links follows by several *provider-customer* links. For instance, consider the routing path depicts in Figure 6. One possible solution may determine that $AS(i)$ is a customer of $AS(i+1)$ (i.e. $AS7$ is the top provider). In this case, the vertices are picked by Step 8 of the algorithm in the following order: $AS1, AS2, AS3, AS4, AS5, AS6, AS7$. In a more realistic solution $AS4$ is defined to be the top provider. In this case, the vertices are picked by Step 8 of the algorithm in the following order: $AS1, AS7, AS2, AS6, AS3, AS5, AS4$. Our heuristic is based on this observation and we choose the vertices assigned to a single path in rotation, i.e. after picking a vertex from one end of a path[7] we prefer that the next vertex picked from this path will be from the second end of the path.

A. *Finding* Peer-Peer *Relationships*

The algorithms described in previous sections give orientation to all edges in the graph, and therefore determines a *customer-provider* relationship between adjacent ASes. In practice some of the edges may be of type *peer-peer*. While Lemma 2.1 proved that every *peer-peer* link can be converted to a *customer-provider* link without harming the solution, the opposite does not hold. Namely, not every *customer-provider* link can be converted to a *peer-peer* link.

With respect to this observation, we need to assign *peer-peer* relationship to some of the links reserving the quality of the solution, namely without increasing the number of invalid paths and without violate the hierarchical, acyclic structure of the graph. We handle this task in two stages, during the execution of the algorithm, and after the assignment of the *customer-provider* links.

In the first step, we refine the algorithms by assigning *peer-peer* relationship to some of the links. In Algorithm *k-AToR* we omit Step 15, while in Algorithm *AToR* we omit Step 16 (i.e. $i = i+1$). Thus, all the remaining vertices have the same value in the topological sort, which is greater than the values of all other vertices. The peering relationship is induced as follow: Given an edge (v, u), the edge is of type *peer-peer* if and only if $\pi(v) = \pi(u)$, and v is a customer of u if and only if $\pi(v) < \pi(u)$. Thus, all the edges connecting the remaining vertices are of type *peer-peer* while other edges are of type *customer-provider*. In this case, the graph does not contain *hierarchical cycle*, but cycles that consist of *peer-peer* links alone are possible. Recall that two ASes connected by *peer-peer* link, means that they are in the same hierarchy level. Thus, these kind of cycles are permitted and they do not violate the hierarchical structure of the AS graph . In addition, for each valid path $p = (v_i, v_j, v_k)$, $\pi(v_i) < \pi(v_j)$ or $\pi(v_k) < \pi(v_j)$ thus the following five options are possible:

$$\pi(v_i) < \pi(v_j) = \pi(v_k), \quad \pi(v_i) < \pi(v_j) < \pi(v_k),$$
$$\pi(v_i) < \pi(v_j) > \pi(v_k), \quad \pi(v_i) = \pi(v_j) > \pi(v_k),$$
$$\pi(v_i) > \pi(v_j) > \pi(v_k)$$

One can see that in all cases, the peering relationship that are derived from each assignment induces a valid path.

In the second step, we convert *customer-provider* links into *peer-peer* similar to the way described in [21]. First, we find a set of *peer-peer* candidates. A *customer-provider* link is added to this set if it can be converted to *peer-peer* link without violate the hierarchical structure of the graph and without increasing the number of invalid paths. While converting any single link from the candidate set is permitted, converting simultaneously more than one link may violate the hierarchical structure or increase the number of invalid paths. Thus, we convert the links in the set one after the other, preferring links that connect nodes that their vertex degrees are on the same order, testing each time if the graph remains acyclic and if the number of invalid path is not increased.

B. *Finding* Sibling-Sibling *Relationships*

A solution of the Algorithm *k-AToR* presented in Section IV may contain invalid paths. In particular, when $|M_v|/|E_v| > 0$ with respect to Step 8 in the algorithm, it means that at least one valley traverses vertex v. While these kind of anomalies can be explained by an unexpected policy, it may reflect a *sibling-sibling* relationship between connected ASes. Thus, invalid paths may be settled by assigning *sibling-sibling* relationship to one of the edge on each valley. Similar to the discussion regarding *peer-peer* links in Section V-A, converting a *customer-provider* link into *sibling-sibling* link may violate the hierarchical structure of the graph. We convert *customer-provider* link into *sibling-sibling* in the following way. First, we find a set of *sibling-sibling* candidates. This set consists of all the edges that are traversed by at least one valley. Then we convert the links in the set one after another, preferring links that connect nodes that their vertex degrees are on the same order, until the set is empty or until all the paths become valid, testing every step if the graph remains acyclic.

VI. Experiments and Simulation Results

In this section we examine practical versions of the algorithms presented above over real data gathered from the

[5]Note that enumerating all possible solutions is computationally infeasible.

[6]The term small AS refers to the hierarchy level of the AS

[7]In this case, we refer to the entire long path before it was sliced into short paths according Lemma 3.1.

Route-Views project [9]. We also simulate the algorithms over several random graphs, and compare the obtained results to other approaches presented in [11], [12], [13].

As mentioned in Section I, the Route-Views database consists of 54 BGP routing tables from different sources. Each table contains a set of AS paths from a single source to almost all other ASes in the AS connectivity map. We use data from April 2006 that consists of 21505 ASes, 45783 links and over 1,500,000 paths. First, we execute Algorithm *AToR* over the entire database. The algorithm returns *NO SOLUTION*, namely every assignment contains cycles or invalid path. Moreover, we found out using the algorithm presented in [13] (and similar to the results presented in [13] over older data), that the simpler *0-ToR* problem (i.e. regardless the existence of cycles), does not have a valley-free solution as well[8].

Next, we execute the different algorithms over the entire Route-Views database. To verify and evaluate the results obtained by the algorithms, one needs to compare these results against actual data, namely one needs to obtain information regarding the real type of relationship of links in the AS connectivity graph, and compare the relationships inferred by the algorithms against the relationships of these links. In contrast to previous work that validate the results against the type of relationships of few ASes (usually contacting the network administrators of these ASes and asking for internal information regarding the actual type of relationships of each AS [11], [21]), we use a general and much more extensive method consisting of a data, collected from the IRR database [4]. This database contains among other things, the export policy of registered ASes. We use the methods presented in [7] and [6] to infer the peering relationships between these ASes. While the IRR database contains 36,000 links and the Route-Views database contains 49,500 links, the intersection of these databases consists of 10,000 links (i.e. 10,000 links appears in both databases). Thus, we compare the solution, obtained by the algorithms to the orientation induced by the IRR database with respect to these intersected links.

Result analysis and measurement: Given an edge (u, v), the export policy of u to v and the export policy of v to u determines the type of relationship of this edge [3], [2]. In particular, an AS may export its routes and its customer routes or it may export its routes and its customer routes, as well as all its provider or peer routes. We denote the first case as *Type I* and the second case as *Type II*. Table I show how the export policy of an edge (u, v) is derived from this export policy. For example, if u exports to v its routes and its customer routes (i.e. *Type I*) and u exports to v its routes and its customer routes, as well as all its provider or peer routes (i.e. *Type II*, it means that u is a customer of v.

If the orientation of an edge, assigned by a specific algorithm is different from the actual orientation of the edge, it means that there is a mismatch between the export policy inferred by the algorithm to the actual export policy of one or both nodes. We say that the orientation of an edge is *fully mismatched* if the export policy of both nodes is incorrect. We say that the orientation of an edge is *partly mismatched* if the export policy of one node is incorrect.

Type of relationship $u - v$	Export policy of u to v	Export policy of v to u
customer-provider	Type I	Type II
provider-customer	Type II	Type I
peer-peer	Type I	Type I
sibling-sibling	Type II	Type II

TABLE I
EXPORT POLICY AND TYPE OF RELATIONSHIP

Table II presents the number of partial and full mismatch edges of each graph and each algorithm (*CR* - our algorithm, presented in Section IV; *BPP* - the algorithm presented in [13]; *Gao* - the algorithm presented in [11]; and *SARK*- the algorithm presented in [12]). The table also depicts the number of policy mismatch, i.e. the total number of cases in which the export policy derived by the algorithm is different from the actual policy[9]. One can see that the results, obtained by our algorithm are better compared to the results obtained by other algorithms. Moreover, while the solution obtained by *CR* is acyclic, the solutions obtained by *BPP*, *SARK*, *Gao*, contain directed cycles. An interesting result is that the number of partial mismatch (in all algorithms) is extremely high compared to the number of full mismatch which may emphasize the difficulty to identify *peer-peer* relationship (a partial mismatch is usually caused by confusing between *peer-peer* and *customer-provider* relationships).

Algorithm	Partial Mismatch	Full Mismatch	Policy Mismatch
CR	1463	93	1649
BPP	1526	104	1734
SARK	2329	192	2713
Gao	1610	109	1828

TABLE II
EXPERIMENTS RESULTS ROUTE-VIEWS VS. IRR

While the experiment presented above is performed over real data, its validation using the IRR may lead to biased validation results. In particular, as mentioned in Section I, in some cases entries in the IRR may be invalid and contain data that is out-of-date or not updated. Second, as pointed out in [7], in some cases the interpretation of the export policy may be ambiguous. For that reason we perform simulations, examining the different algorithms over several random graphs. This process consists of the following steps. First, we generated a random graph and assign a type of relationship to its edges. Then, we simulate the gathering process of BGP routing tables, by modelling each BGP routing table as a policy-based shortest path tree. This modelling is motivated by the assumptions that each BGP routing table contains paths from a single source to the entire ASes, and under the policy discussed so far routing is done along shortest paths[10]. Finally, we

[8]We refer to the *0-ToR* problem as the decision version of the *ToR* problem that determines if there is a solution without any invalid path.

[9]This is exactly $(\text{Partial Mismatch} + 2 \cdot \text{Full Mismatch})$.

[10]For further discussion regarding this modelling see Section 2 in [7].

execute the different algorithms and measure their inaccuracy, i.e. in how many edges, each algorithm failed to assign the correct type of relationship.

We consider several random graphs; each of them consists of 22,000 vertices (similar to the number of ASes observed by the Route-Views database). As was suggested in [7] about third of the links in the AS graph are of type *customer-provider* while almost all the rest of the links are of type *peer-peer*. In the first graph we use the guidelines from [20], [6], [21] and divide the set of nodes of the graph into five hierarchical groups, where the number of ASes in each group increases exponentially. Thus, the set of nodes of each graph is divided in the following way: 10 ASes are in level 1, 140 ASes are in level 2, 1350 ASes are in level 3, 3500 ASes are in level 4, and about 17000 ASes are in level 5. According to this hierarchy, ASes from the same groups are connected by a *peer-peer* relationship while ASes from different groups are connected by a *customer-provider* relationship. In order to guarantee the reachability of the nodes in the graph with respect to the policy enforced[11], we build the graph such that each AS has, on the average, two providers. We call this graph *layered graph*. The second graph captures the fact that the vertex degree distribution of the *customer-provider* subgraph follows the power-law while the *peer-peer* vertex degree distribution does not [7]. Thus, in this graph the *customer-provider* subgraph is modelled by a Barabasi-Albert graph [22] while the *peer-peer* subgraph is modelled by a random graph. We call this graph *Barabasi-Albert graph*. The ten top providers in both graph are fully connected by a set of 45 *peer-peer* links.

Barabasi-Albert / Layered graph

Algorithm	Partial Mismatch	Full Mismatch	Policy Mismatch
CR	304 / 221	7 / 0	318 / 221
BPP	730 / 1808	22 / 44	774 / 1896
SARK	9980 / 5590	806 / 35	11592 / 5660
Gao	833 / 417	26 / 1	885 / 419

TABLE III
SIMULATION RESULTS OF THE RANDOM GRAPHS

Table III presents the number of partial and full mismatch edges of each graph and each algorithm. It also presents the number of cases in which the export policy derived by the algorithm is different from the actual policy. Similar to the experiments over the Route-View database, in both graphs the results obtain by our algorithm are much better compared to other algorithms. These results stand out mainly when the number of full mismatch is considered. Likewise, the solutions obtained by our algorithm is the only one that preserve the hierarchical acyclic structure of the graph.

VII. DISCUSSION

In this paper we studied the Type of Relationship problem. We observed that the conventional definition of the problem does not consider the hierarchical acyclic structure of the AS connectivity map. Base on this observation we defined a new problem, the Acyclic Type of Relationship problem, that takes into account this hierarchical structure. We proved that determining if there is a solution without invalid paths can be solved in a polynomial time, and presented an efficient algorithm for this case. We also presented a $\frac{2}{3}$ approximation algorithm for a variant of the problem, considering the total number of valleys. Our experiments and simulation results show that our algorithms classify the type of relationship between ASes much better than all previous approaches.

[11]In general, when routing policy is considered, connectivity does not necessarily mean reachability, namely if two ASes are physically connected via one or more physical paths it does not necessarily means that there is a permitted path with respect to the adopted policy.

REFERENCES

[1] Y. Rekhter, T. Watson, and T. Li, "Border gateway protocol 4," *RFC 1771*, 1995.

[2] G. Huston, "Interconnection, peering and settlement," in *Proc. of INET*, June 1999.

[3] C. Alaettinoglu, "Scalable router configuration for the internet," in *Proc. of IEEE IC3N*, October 1996.

[4] "Internet routing registry," http://www.irr.net.

[5] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra, "Routing policy specification language (rpsl)," *RFC 2622*, 1999.

[6] G. Siganos and M. Faloutsos, "Analyzing bgp policies: Methodology and tool," in *Proc. of IEEE INFOCOM*, 2004.

[7] R. Cohen and D. Raz, "The internet dark matter – on the missing links in the as connectivity map," in *Proc. of IEEE INFOCOM*, 2006.

[8] H. Chang, R. Govindan, S. Jamin, S. J. Shenker, and W. Willinger, "Towards capturing representative as-level internet topologies," *Computer Networks Journal*, vol. 44, no. 6, pp. 737–755, April 2004.

[9] "University of oregon route views projects," http://www.routeviews.org.

[10] Y. Shavitt and E. Shir, "Dimes: Let the internet measure itself." in *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, October 2005, pp. 71–74.

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

[12] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz, "Characterizing the internet hierarchy from multiple vantage points," in *Proc. of IEEE INFOCOM*, 2002.

[13] G. D. Battista, T. Erlebach, A. Hall, M. Patrignani, M. Pizzonia, and T. Schank, "Computing the types of the relationships between autonomous systems," *IEEE/ACM Transactions on Networking*, 2007, to appear.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

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

[16] L. Gao, T. Griffin, and J. Rexford, "Inherently safe backup routing with bgp," in *Proc. of IEEE INFOCOM*, 2001, pp. 547–556.

[17] S. Kosub, M. G. Maaß, and H. Täubig, "Acyclic type-of-relationship problems on the internet," in *Proceedings of the 3rd Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN'2006)*, 2006, pp. 98–111.

[18] B. Chor and M. Sudan, "A geometric approach to betweenness," in *Third Annual European Symposium on Algorithms*, September 1995, pp. 227–237.

[19] J. Opatrny, "Total ordering problem." *SIAM J. Comput.*, vol. 8, no. 1, pp. 111–114, 1979.

[20] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds., 1972, pp. 85–103.

[21] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, kc claffy, and G. Riley, "As relationships: Inference and validation," *ArXiv Computer Science e-prints*, April 2006.

[22] A. L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science,*, vol. 286, pp. 509–512, October 1999.