T U M

I N S T I T U T F Ü R I N F O R M A T I K

Acyclic Type-of-Relationship Problems on the Internet: An Experimental Analysis

Benjamin Hummel Sven Kosub

TUM-I0709
Februar 07

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-02-I0709-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Acyclic Type-of-Relationship Problems on the Internet: An Experimental Analysis

Benjamin Hummel *Sven Kosub*
Fakultät für Informatik, Technische Universität München,
Boltzmannstr. 3, D-85748 Garching, Germany
{hummelb,kosub}@in.tum.de

Abstract

An experimental study of the feasibility and accuracy of the acyclicity approach introduced in [16] for the inference of business relationships among autonomous systems (ASes) is provided. We investigate the maximum acyclic type-of-relationship problem: on a given set of AS paths, find a maximum-cardinality subset which allows an acyclic and valley-free orientation. Inapproximability and NP-hardness results for this problem are presented and a heuristic is designed. The heuristic is experimentally compared to most of the state-of-the-art algorithms on a reliable data set. It turns out that the proposed heuristic produces the least number of misclassified customer-to-provider relationships among the tested algorithms. Moreover, it is flexible in handling pre-knowledge in the sense that already a small amount of correct relationships is enough to produce a high-quality relationship classification. Furthermore, the reliable data set is used to validate the acyclicity assumptions. The findings demonstrate that acyclicity notions should be an integral part of models of AS relationships.

1 Introduction

Interrelationship analysis of autonomous systems (ASes) has recently attracted much attention in both theoretical and practical research on Internet inter-domain routing with BGP (see, e.g., [9, 23, 22, 8, 5, 16, 7]). This interest is motivated by insights how routing stability and quality in Internet is influenced not only by physical connections between ASes but heavily by their business relationships. As business contracts are subject to privacy, computational tools and techniques are required to infer close-to-reality relationship classifications from publicly available resources such as WHOIS databases, the Internet Routing Registry [17], or BGP beacons (e.g., [21, 25]).

A useful approach is the interpretation of observable BGP routes. Techniques based on this approach usually work as follows: collect a set of AS paths from BGP routers, obtain an AS-level connectivity graph (the *AS graph*) by merging all AS paths, and label the AS graph with business relations such that all—actually, as many as possible—collected AS paths are valley-free. Valley-freeness is a characteristic property of AS paths based on economic

rationality, which in a simplified version says that, in the direction of traffic, a customer-to-provider link should never follow a provider-to-customer link. The implementations ranges from purely combinatorial (e.g., [5, 16]) to purely heuristical one's (e.g., [9, 23, 27]). Though there has been some criticism of unrealistic classifications [8], the empirical findings are encouraging for further developments.

In [16], acyclicity has been added to valley-freeness as another structural condition of AS graph labelings. The rationale for acyclicity is that it is unlikely the case that an AS A is a provider of an AS B, AS B is a provider of an AS C, and AS C is a provider of AS A. It has turned out that finding labelings which are both valley-free on the path set and acyclic in the AS graph is easy, even if we want to respect explicit pre-knowledge (following the so-called partialness-to-entireness methodology [27]). However, this theoretical feasibility and plausibility of the acyclicity approach to AS relationship inference has not been supported with empirical evidence in [16]. In this paper, we close this gap.

1.1 Contributions of the Paper

We contribute to an experimental analysis of the acyclic type-of-relationship problems in three ways.

First, we operationalize acyclicity. In [16], algorithms have been presented to test whether all paths of a given path set allow acyclic and valley-free orientations. In practice, collected path sets are expected to fail this test. Thus, these algorithms are of theoretical interest only. We consider the problem to find acyclic orientations which maximize the number of valley-free paths. We provide lower bounds on the approximability of this problem (which implies the NP-hardness as well) and we design a fast heuristic, which is refered to as AHeu, for finding acyclic orientations which are valley-free on a large part of a given path set.

Second, we validate the acyclicity assumptions from [16]. In doing so, one issue is obtaining a reliable data set. We report on several techniques employed. The graph we receive from the reliable data set when only using customer-to-provider relationship is acylic. Including peer-to-peer relationships is more problematic. It seems that we do not yet have the right understanding how peer-to-peer relationships affect the graph-theoretical structure of BGP routes and the AS graph.

Third, we compare the inference quality of our heuristic to a set of standard algorithms from the literature. On the reliable data set, AHeu produces the lowest number of misclassified customer-to-provider edges among all algorithms tested (in numbers: approximately 0.3% of all edges in the reliable data set are misclassified). We further test how the inference quality of the algorithms depend on inital pre-knowledge. Here, it is observed that for AHeu the relative number of misclassified customer-to-provider edges is nearly independent of the amount of pre-knowledge. That is, already a small amouth of correct relationships is enough to infer a high-quality classification among the ASes.

All in all, the findings of this paper indicate that AHeu is a feasible and flexible heuristic with excellent inference quality (at least with respect to customer-to-provider relationships) and that in general, acyclicity should be an integral part of any further accuracy improvements.

1.2 Related Work

Several algorithms have been proposed to infer relationship types from AS paths. The first attempt was made in [9] where the valley-free path model was introduced and a heuristic was designed based on statistical properties of a given path set. This approach was pushed further in [23, 27] to combine valley-free path labelings obtained from different observation points and from sources other than AS paths. In [5] (and some precursor papers), a combinatorial approach is developped based on expressing valley-freeness of paths in terms of the 2SAT problem. A combination of the 2SAT-based formulation of valley-freeness and the statistical properties of path set in terms of mathematical programming has been proposed in [8, 7]. A detailed description of the standard algorithms is given in Section 3. The acyclicity approach to interrelationship analysis is from [16]. We explain it in detail in Subsection 2.3 (formal definitions of acyclicity) and Section 4 (algorithmic ideas) where we devise our heuristic algorithm. Computational techniques not based on AS path interpretation have been proposed and discussed in, e.g., [22, 27, 6, 7]. Some of them are used in this paper to obtain a reliable data set. We mention them if we discuss our methods.

1.3 Organization of the Paper

The rest of the paper is organized as follows: Section 2 contains a description of an abstract model of BGP setting our context. It also contains a formal introduction to the valley-free path model and the acyclicity assumptions. In Section 3 existing algorithms for solving the interrelationship inference problems are discussed. In Section 4 we introduce the maximum acyclic type-of-relationship problem, prove some (in)approximability results, and describe a heuristic approach to solve it. Section 5 explains how we obtained our data sets for experimental tests. In Section 6 we present data for validating the acyclicity assumptions. Finally, in Section 7 we present and discuss our experimental findings. We conclude with a summary and open problems.

2 Preliminaries

We briefly describe a simple, abstract model of inter-domain routing in the Internet using BGP (see, e.g., [26, 20, 11, 9]).

2.1 The Selective Export Rule

The elementary entities in our Internet world are IP adresses, i.e., bit strings of prescribed length. An autonomous system (AS) is a connected group of one or more IP prefixes (i.e., blocks of contiguous IP adresses) run by one or more network operators which has a single and clearly defined routing policy [12]. An AS aims at providing global reachability for its IP adresses. To achieve this goal, ASes having common physical connections exchange routing informations as governed by their own local routing policies. BGP is the *de facto* standard protocol to manage data traffic between ASes for inter-domain routing as well as for route propagation.

AS v exports to	provider	customer	peer	sibling
own routes	Yes	Yes	Yes	Yes
customer routes	Yes	Yes	Yes	Yes
provider routes	No	Yes	No	Yes
peer routes	No	Yes	No	Yes

Figure 1: The Selective Export Rule.

Reachability in the Internet depends on (physical) connectivity and contractual relationships between ASes. The most fundamental binary business relationships are customer-to-provider—where the provider sells routes to the customer—, peer-to-peer—where the involved ASes provide special routes to their customers but no transit for each other—, and sibling-to-sibling—where both ASes belong to the same administrative domain. Evidently, sibling-to-sibling relations are transitive. More peculiar relationships appear in the real world (see, e.g., [9]). We restrict ourselves to the three mentioned types of relationships.

More specifically, let V be a set of AS numbers. For any $v \in V$, let $N(v) \subseteq V$ denote the set of its neighbor ASes, i.e., all numbers of ASes sharing a physical connection with v. The undirected graph $G = (V, E)$ where $E = \{ \{u, v\} \mid v \in N(u) \}$ is called a *connectivity graph* (at the AS level) or simply *AS graph*. Let $v \in V$ be any AS. According to the business relationship we divide the neighbors of v into the sets $\mathrm{Cust}(v)$ of all customers of v, $\mathrm{Prov}(v)$ of all providers of v, $\mathrm{Sibl}(v)$ of all siblings of v, and $\mathrm{Peer}(v)$ of all peering partners of v. Some of the sets may be empty. We let $\mathrm{Sibl}(v)$ contain v as well. Let $R(v)$ denote the set of all currently *active* AS paths in the BGP routing table of v, i.e., all AS paths that have been announced from neighboring ASes at a certain time and never been withdrawn. Assumed that there are no misconfigurations of BGP, all AS paths in $R(v)$ are loopless and not including v. Here, we say that an AS path is *loopless* whenever between two sibling ASes on the path, no non-sibling AS is passed. Based on the neighborhood classification, we further divide $R(v)$ into four categories. A loopless AS path $(u_1, \ldots, u_r) \in R(v)$ is

$$\begin{aligned}
\text{a } \textit{customer route} \text{ of } v &\iff_{\text{def}} \text{ leftmost } u_i \notin \mathrm{Sibl}(v) \text{ lies in } \mathrm{Cust}(v),\\
\text{a } \textit{provider route} \text{ of } v &\iff_{\text{def}} \text{ leftmost } u_i \notin \mathrm{Sibl}(v) \text{ lies in } \mathrm{Prov}(v),\\
\text{a } \textit{peer route} \text{ of } v &\iff_{\text{def}} \text{ leftmost } u_i \notin \mathrm{Sibl}(v) \text{ lies in } \mathrm{Peer}(v),\\
\text{an } \textit{own route} \text{ of } v &\iff_{\text{def}} \text{ for all } 1 \leq i \leq r,\ u_i \in \mathrm{Sibl}(v).
\end{aligned}$$

Now, typically (at least, recommendably), ASes set up their export policies according to the Selective Export Rule [1, 13, 9] as described in Figure 1. In our simplified model, the receiving AS gets from an AS those (locally preferred) routes destined for it and prolongated with the number of the sending AS as the new leftmost AS number in the path.

2.2 The Valley-Free Path Model

Valley-freeness is a graph-theoretical consequence of the Selective Export Rule. Let $G = (V, E)$ be an undirected (simple) graph. We assume that $(u, v) \in E \Leftrightarrow (v, u) \in E$. A (mixed) *orientation φ of G* is a mapping from E to T where T denotes a set of possible edge-types. For instance, a directed graph is a graph oriented with type set $T = \{\leftarrow, \rightarrow\}$. We consider type

sets having the following edge-types and interpretations:

$\rightarrow$	indicating a customer-to-provider relationship
$\leftarrow$	indicating a provider-to-customer relationship
$-$	indicating a peer-to-peer relationship
$\leftrightarrow$	indicating a sibling-to-sibling relationship

Throughout this paper, we only consider orientations φ that are consistent with respect to $\rightarrow$. That is, for all $(u, v) \in E$ if $\varphi(u, v) = \leftarrow$ then $\varphi(v, u) = \rightarrow$ and if $\varphi(u, v) = \rightarrow$ then $\varphi(v, u) = \leftarrow$. Thus, if we allow $\rightarrow$ as a possible edge type, then we immediately allow $\leftarrow$ as a possible edge type as well. Instead of $\varphi(u, v) = \rightarrow$ for an edge $(u, v) \in E$ we also write $u \rightarrow v$.

We extend φ from edges to walks homomorphically. Let $(v_0, v_1, \ldots, v_m)$ be any walk in a graph G. Then $\varphi(v_0, v_1, \ldots, v_m)$ is defined to be $\varphi(v_0, v_1)\varphi(v_1, v_2)\ldots\varphi(v_{m-1}, v_m)$, i.e., in our setting generally, a word in $\{\leftarrow, \rightarrow, -, \leftrightarrow\}^*$. We will typically use regular expressions to describe walk types given an orientation. An important property of orientations is valley-freeness, which we state here in terms of regular patterns of paths.

Definition 1 ([9]). *Let G be any graph, and let $\varphi(G)$ be an orientation of G. A loopless path $(v_0, \ldots, v_m)$ is said to be* valley-free *in $\varphi(G)$ if and only if $\varphi(v_0, \ldots, v_m)$ belongs to*

$$\{\rightarrow, \leftrightarrow\}^*\{\leftarrow, \leftrightarrow\}^* \quad \cup \quad \{\rightarrow, \leftrightarrow\}^* - \{\leftarrow, \leftrightarrow\}^*.$$

The valley-freeness of paths abstracts the condition that autonomous systems never route data from one of their providers to another of their providers because instead of earning money, they would have to pay twice for these data streams.

Theorem 2 ([9]). *Let $G = (V, E)$ be an AS graph. Let P be any subset of AS paths of all BGP routing tables, i.e., $P \subseteq \bigcup_{v \in V} R(v)$. If all ASes export their routes according to the Selective Export Rule, then there exists an orientation of P such that all AS paths in P are valley-free.*

2.3 The Acyclicity Assumptions

Following [16], we summarize reasonable acyclicity structures within a connectivity graph, i.e., patterns of oriented cycles which are forbidden to be contained in the graph. An oriented cycle (in its simplest form) can be interpreted as someone being its own provider and customer.

The following formal definition of an oriented cycle has been proposed in [16].

Definition 3 ([16]). *Let G be any graph, and let $\varphi(G)$ be an orientation of G. Let C be any minimal cycle of G, i.e., a cycle that does not contain a vertex twice. C is said to be an* oriented cycle of $\varphi(G)$ *if and only if $\varphi(C)$ belongs to*

$$\{-, \leftrightarrow\}^* \rightarrow \{\rightarrow, -, \leftrightarrow\}^* \quad \cup \quad \{-, \leftrightarrow\}^* \leftarrow \{\leftarrow, -, \leftrightarrow\}^* \quad \cup \quad \leftrightarrow^* - \leftrightarrow^*.$$

The minimality of cycles is required since we exactly count occurrences of peer-to-peer edges in oriented cycles.

To exemplify the definition, Figure 2 shows the 16 non-isomorphic triads of the 64 possible orientations of a complete graph on three vertices. Half of them are oriented cycles according to Definition 3 and half of them are not.

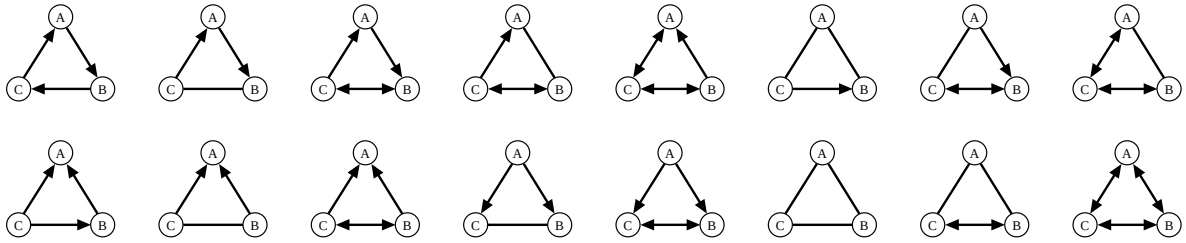

Figure 2: Non-isomorphic triads. All triads in the upper row are forbidden and all triads in the lower row are allowed.

Note that in the case that φ does not exhaust the full type set $\{\rightarrow, —, \leftrightarrow\}$, the patterns of oriented cycles simplifiy. For instance, if the type set is $\{\rightarrow\}$, then we obtain that a minimal cycle C is an oriented cycle if and only if $\varphi(C)$ belongs to $\rightarrow^*$ or $\leftarrow^*$ which is the usual understanding of a cycle. As a second example, if the type set is $\{\leftrightarrow, —\}$, then a minimal cycle C is an oriented cycle if and only if $\varphi(C)$ belongs to $\leftrightarrow^* — \leftrightarrow^*$.

The plausibility of Definition 3 is based on the following size rules (see, e.g., [10, 9, 18, 23, 16]):

1. If AS u is a customer of AS v, then AS u has much smaller size (i.e., number of routers) than AS v.

2. If AS u is a peering partner of AS v, then AS u and AS v are roughly of the same size. Here, the binary relation *roughly the same size* is considered to be transitive.

3. If AS u and AS v are siblings, then they count as one AS, i.e., we assume that the size of u is determined by its own number of routers and the number of routers of all its sibling ASes.

As the definition above is logically implied by these rules, objections to the definition should be traced back to objections to some of these rules.

We call an orientation *acyclic* if it contains no oriented cycles. Testing whether an orientation is acyclic can be done fastly by standard techniques (see, e.g., [4, 16]).

3 Existing Inference Algorithms

The problem of inferring business relationships from AS paths has been studied before. More precisely given a (multi)set of AS paths (which imply an AS graph) we try to find a classification of the edges of the AS graph, such that as many paths as possible are valley-free. Following we will present some of the algorithms for solving this problem.

3.1 Gao's Heuristic

The inference of relationships classified into customer-to-provider, peer-to-peer, and sibling-to-sibling links was originally formulated in [9]. This paper also gives the first heuristic for the problem, based on two assumptions:

1. For a pair of ASes which are connected, always the larger one is the provider for the smaller one. (This is the first size rule from Subection 2.3.)

2. The size of an autonomous system is proportional to its degree in the AS graph.

The algorithm operates in three phases. At first the AS graph is constructed from the set of AS paths by inserting an edge for each pair of ASes adjacent on any path. This graph is also used to calculate the degree of an autonomous system. The second phase finds for each path the AS with maximum degree (the *top provider*) and records for each edge on the path the direction towards the top provider as preferred. Finally the third phase iterated over all edges of the AS graph and compares how often each of both possible directions was preferred. If the vote is clear, the edge is oriented as customer-to-provider in the implied direction, in case of a tie a sibling-to-sibling edge is inferred. The decision whether the vote is clear is based on a threshold L which is recommended to be set to 1 (for details consult [9]).

3.2 A Simple Approximation Algorithm

In [5] a simple approximation algorithm based on a random orientation of the AS graph is presented. The idea is to orient each edge either as customer-to-provider or provider-to-customer (with probability 1/2). Then a path of length ℓ is valley-free with probability $\frac{\ell+1}{2^\ell}$, as there are only $\ell + 1$ valley-free configurations out of 2^ℓ possible orientations. Thus if the length of the input paths can be bounded by ℓ, it is expected that a fraction of $\frac{\ell+1}{2^\ell}$ of all given paths is valley-free. Using conditional probabilities this algorithm is derandomized yielding the desired approximation algorithm with approximation factor $\frac{\ell+1}{2^\ell}$. We will refer to this algorithm as APX. The mentioned approach can be improved by means of semidefinite programming (SDP) as described in the paper.

3.3 2SAT-based DPP* Heuristic

Another heuristic approach based on the 2SAT problem is also given in [5]. They start with an arbitrary orientation of all edges of the AS graph as either customer-to-provider or provider-to-customer. To each edge a boolean variable is assigned which is true if this edge should be reversed and false otherwise. The crucial observation is that valley-freeness can be checked for locally by inspecting edge pairs adjacent in an AS path. For a single pair of edges there is exactly one situation which is forbidden (both edges pointing away from each other). If an orientation is found where this forbidden situation is avoided for all edge pairs, all AS paths are automatically valley-free. These forbidden edge configurations are formulated as a boolean expression over the edge variables introduced before. As only two edges are involved in each forbidden pattern, the conjunction of all these expressions yields a 2SAT instance which can be solved in linear time. From this 2SAT solution the orientation for the AS graph is easily found by flipping all edges whose variables are true.

The algorithm just described can answer the question whether it is possible to find an orientation making *all* paths valley-free. To get an algorithm for maximizing the number of valid paths, above algorithm is applied on a set of AS paths. If it finds a suitable orientation, all paths are valey-free. Otherwise the set of "blocking" variables (and thus edges) is calculated (this information is easily available from the way the 2SAT problem is solved). From these edges

the one contained in the least number of paths is selected and all paths containing this edge are removed. With this reduced path set the procedure is restarted until finally an orientation for which all remaining paths are valid is found. It should be pointed out that using suitable data structures the described process can be performed incrementally without having to reconstruct the 2SAT instance in each iteration.

After this first phase the heuristic tries to reinsert the removed paths. In [5] two methods for this are mentioned. The first one (called DPP) simply adds each single path and tests if there is still a valley-free orientation. If not, the path is rejected again. The second method (DPP*) considers each edge not yet oriented and counts how many of the removed paths would benefit from each of the possible orientations of this edge. Then the edge is oriented according to the majority of these "votes". As this second approach is reported to be both faster and better in terms of the number of valley-free paths we only use DPP* here.

3.4 Handling Pre-knowledge

In this paper we also explore the influence of previous knowledge on the quality of the induced relationships. As none of the mentioned algorithms has support for this kind of information, we are using a simple preprocessing algorithms in conjunction with them when calculating inference results for given pre-knowledge. This algorithm calculates a partial orientation of the AS graph and a set of remaining AS paths. The final result is then obtained by applying one of the presented inference algorithm on the remaining paths and merging the orientations from the inference algorithm with those from the preprocessor.

Algorithm 1: Preprocessing algorithm for handling pre-knowledge

Input: AS path set P, set of known customer-to-provider edges E
Output: A reduced set of AS paths usable to infer the edges which could not be inferred from E

```
1  P' := P, P := ∅
2  while P ≠ P' do
3      P := P', P' := ∅
4      foreach path p ∈ P do
5          let p = p_1, p_2, ..., p_n
6          up := max{i | p_{i-1} → p_i is in E} ∪ {0}
7          down := min{i | p_i ← p_{i+1} is in E} ∪ {n}
8          if up > down then
9              drop this path, as it cannot be valley-free for E
10         else
11             for i := 2 to up −1 do
12                 insert edge p_{i-1} → p_i into E unless edge p_{i-1} ← p_i exists
13             for i := down +1 to n − 1 do
14                 insert edge p_i ← p_{i+1} into E unless edge p_i → p_{i+1} exists
15             if up < down then insert path p_up, ..., p_down into P'
16 return P
```

Our simple algorithm only handles customer-to-provider edges in the pre-knowledge. It exploits the fact that a single edge on an AS path fixes the orientation of many other edges on this path. These edges are added to the pool of known edges. We repeat this step for all paths until we gain no new information. The details are outlined in Algorithm 1. Two kinds of errors can occur during the application of the algorithm which both indicate errors in the data (partially from BGP misconfigurations). On the one hand a path might be not orientable valley-free with respect to the already known orientations, on the other hand newly found orientations can be in conflict with the existing knowledge. We implemented no advanced correction scheme for these situations but simply react with discarding the new conflicting information.

4 A Heuristic Algorithm for the Maximum Acyclic Type-of-Relationship Problem

An algorithm for finding an acyclic and valley-free orientation of an AS graph for a given path set is presented in [16]. If there is no such orientation the algorithm reports this. We will show how to modify this algorithm to find an acyclic orientation which is valley-free for a *large part* of the AS paths followed by some improvements to the basic scheme. Formally, we consider the following optimization problem (see [3] for notation):

Problem:	Maximum Acyclic ToR
Input:	AS path set P (with possibly multiple occurrences of paths)
Solution:	A subset $P' \subseteq P$ allowing an acyclic and valley-free orientation
Measure:	The cardinality $\|P'\|$ of the subset P'

Later it will be more convenient (in particular, for algorithms) to have also the AS graph induced by P as part of the input. The goal is to design algorithms that find a solution on a given P with the cardinality as high as possible.

4.1 Approximation Guarantees

We start by analyzing the existence of feasible algorithms for Maximum Acyclic ToR having provable approximation guarantees.

On the negative side, we rule out the existence of polynomial-time approximation schemes, unless P = NP. A polynomial-time approximation scheme (for a maximization problem) is an algorithm which computes in time polynomial in the size of the instance (x, ε), $\varepsilon \leq 1$, a solution which is within a factor ε of the optimal solution for the input x (see, e.g., [3]). We feel that the achievable approximation quality is much poorer than stated here but we do not have a proof for this.

Technically, we prove that Maximum Acyclic ToR is hard for the class APX. APX is the class of optimization problems admitting algorithms with a constant approximation ratio (see, e.g., [3]).

Theorem 4. Maximum Acyclic ToR *is* APX*-hard.*

Proof. Maximum Acyclic Subgraph is the problem to compute, on a given directed graph $G = (V, E)$, a maximum-cardinality edge subset $E' \subseteq E$ such that (V, E') is acyclic. It is known that Maximum Acyclic Subgraph can be approximated with in a factor $\frac{1}{2}$ of the optimum but does not admit a polynomial-time approximation scheme, i.e., it is APX-complete [19]. We describe an L-reduction from Maximum Acyclic Subgraph to Maximum Acyclic ToR (see, e.g., [3] for a definition of L-reductions). Given a directed graph $G = (V, E)$ (without loops) let f be the function defined by

$$f(G) =_{\text{def}} \left\{ (u, v, (u, v)^-, (u, v)^+), (u, v, (u, v)^+, (u, v)^-) \;\middle|\; (u, v) \in E \right\}.$$

Here, $(u, v)^-$ and $(u, v)^+$ are new vertices which are associated with the directed edge (u, v) (and thus are different to vertices $(v, u)^-$ and $(v, u)^+$ if the edge (v, u) also belongs to E) and which are not contained in V. Note that each acyclic and valley-free orientation of a path set containing both paths $(u, v, (u, v)^-, (u, v)^+)$ and $(u, v, (u, v)^+, (u, v)^-)$ orients u towards v. Therefore, if G is acyclic then $f(G)$ allows an acyclic and valley-free orientation. For any path set $P \subseteq f(G)$ define the corresponding edge set $g(G, P)$ by

$$g(G, P) =_{\text{def}} \left\{ (u, v) \in E \;\middle|\; (u, v, (u, v)^-, (u, v)^+) \in P \text{ and } (u, v, (u, v)^+, (u, v)^-) \in P \right\}$$

Clearly, f and g are computable in polynomial time. Note that $\|f(G)\| = 2\|E\|$ and $\|g(G, P)\| \geq \|P\| - \|E\|$. Let E^* be an optimal solution for $G = (V, E)$ and let P^* be an optimal solution for $f(G)$. Then, we easily obtain that

$$\|P^*\| \leq \|f(G)\| = 2\|E\| \leq 4\|E^*\|.$$

For the latter note that for each $G = (V, E)$ there is always an acyclic subgraph having at least $\frac{1}{2}\|E\|$ edges [19]. On the other hand, since (V, E^*) is acyclic, there is a topological ordering $\pi : V \to \{1, \ldots, \|V\|\}$ for (V, E^*), i.e., π is bijective and satisfies that if $(u, v) \in E^*$ then $\pi(u) < \pi(v)$. Moreover, since E^* is maximal, for each $(u, v) \in E \setminus E^*$ we have $\pi(u) > \pi(v)$. Thus, if we reverse all edges $(u, v) \in E \setminus E^*$, then we obtain an acyclic graph witnessed by the same topological ordering π. So, the path set

$$P(E^*) =_{\text{def}} f(G) \setminus \left\{ (u, v, (u, v)^-, (u, v)^+) \;\middle|\; (u, v) \in E \setminus E^* \right\}$$

allows an acyclic and valley-free orientation. Hence, we conclude that $\|P^*\| \geq \|P(E^*)\| = \|E^*\| + \|E\|$. Thus, the following bound on the absolute error holds:

$$\|E^*\| - \|g(G, P)\| \leq \|P^*\| - \|E\| - (\|P\| - \|E\|) = \|P^*\| - \|P\|$$

Consequently, Maximum Acyclic Subgraph L-reduces to Maximum Acyclic ToR via $(f, g, 4, 1)$. Since Maximum Acyclic Subgraph belongs to APX, we obtain the APX-hardness of Maximum Acyclic ToR (see [3, Lemma 8.2]). □

The decision version of Maximum Acyclic ToR consists of all instances (P, k) such that P is a (multi)set of AS paths containing a subset $P' \subseteq P$ which has at least k paths and which allows an acylic and valley-free orientation.

Corollary 5. *The decision version of* Maximum Acyclic ToR *is* NP*-complete.*

Proof. The containment in NP is obvious. The hardness follows by reduction from the decision version of Maximum Acyclic Subgraph. The decision version of Maximum Acyclic Subgraph consists of all pairs (G, k) such that $G = (V, E)$ is a directed graph containing an acyclic subgraph with at least k edges. This problem is known to be NP-complete [14]. For the reduction we use the function f from the proof of Theorem 4 and prove the following: Let $G = (V, E)$ be a directed graph with m edges. The pair (G, k) belongs to the decision version of Maximum Acyclic Subgraph if and only if the pair $(f(G), k+m)$ belongs to the decision version of Maximum Acyclic ToR. For the direction from left to right let $E' \subseteq E$ be an edge set such that $\|E'\| \geq k$ and (V, E') is acyclic. Define $P(E')$ as in the proof of Theorem 4. Then, by the same arguments as there, we obtain that $P(E')$ has an acyclic and valley-free orientation and $\|P(E')\| = k + m$. For the direction from right to left let $P \subseteq f(G)$ be a path set such that $\|P\| \geq k + m$ and P has an acyclic and valley-free orientation. Define $E' =_{\text{def}} g(G, P)$ for the same g as in the proof of Theorem 4. We conclude that (V, E') is acyclic and $\|E'\| \geq \|P\| - m \geq k$. This proves the NP-hardness of the decision version of Maximum Acyclic ToR. □

On the positive side, we do not know any non-trivial bound on the approximation quality of Maximum Acyclic ToR. However, if we restrict the allowed path lengths, then we obtain a constant approximation ratio. Let Maximum Acyclic ToR$_k$ denote the problem to find, given a set of paths of length at most k, a maximum-cardinality subset which allows an acyclic and valley-free orientation.

Theorem 6. *For each $k \in \mathbb{N}_+$, the problem* Maximum Acyclic ToR$_k$ *can be approximated within a factor of $\frac{2^k}{(k+1)!}$ of the optimum in polynomial time.*

Proof. The proof is by induction over k. For the base of induction, suppose $k = 1$. Obviously, each path set consisting of paths of length one, i.e., edges, allows an acyclic and valley-free orientation. So, the optimum can be achieved which implies an approximation ratio 1. As to the induction step, let $k > 1$. Suppose we are given a path set P. For a vertex $v \in V(P)$, let $\beta_P(v)$ denote the number of paths of P such that v is located at the boundary and let $\mu_P(v)$ denote the number of paths of P such that v is located in the middle. Let $P(v) \subseteq P$ be the set of all paths containing the vertex v. By the pigeon-hole principle, there is always a vertex $v \in V(P)$ such that $\beta_P(v) \geq \frac{2}{k-1} \cdot \mu_P(v)$. Fix such a vertex v_0. Iteratively find such vertices in the path set $P \setminus P(v_0)$. Finally, this produces a sequence of vertices $v_0, v_1, \ldots, v_r$ such that

1. $P = \bigcup_{i=0}^{r} P_i$, where $P_0 =_{\text{def}} P(v_0)$ and for $1 \leq i \leq r$, $P_i =_{\text{def}} (P \setminus \bigcup_{j=0}^{i-1} P_j) \cap P(v_i)$, and
2. for all $0 \leq i \leq r$ it holds that $\beta_{P_i}(v_i) \geq \frac{2}{k-1} \cdot \mu_{P_i}(v_i)$.

Let P_i' be the set of paths obtained from the path set P_i as follows: remove completely all paths with v_i in the middle. In the remaining paths of P_i remove the vertex v_i from the path (which is a boundary vertex). Clearly, the paths of P_i' have length at most $k-1$. Define $P' =_{\text{def}} \bigcup_{i=0}^{r} P_i'$ by multiset union. That is, the multiplicity of a path $p \in P'$ corresponds to the number of v_i's such that $p \in P_i$. Then, P' consists of paths of length at most $k-1$ and can be computed in polynomial time. Moreover, we obtain

$$\|P\| = \sum_{i=0}^{r} \|P_i\| = \sum_{i=0}^{r} \beta_{P_i}(v_i) + \mu_{P_i}(v_i) \leq \left(\frac{k-1}{2} + 1\right) \cdot \sum_{i=0}^{r} \beta_{P_i}(v_i) = \frac{k+1}{2} \cdot \|P'\|.$$

By induction hypothesis, there is an algorithm outputting a path set $L' \subseteq P'$ such that L' allows an acyclic and valley-free orientation and $\|L'\| \geq \frac{2^{k-1}}{k!} \cdot \|P'\|$. We easily obtain from L' a path set L by attaching the vertices v_i back to the corresponding paths and orienting the edges from v_i towards the middle, i.e., $L =_{\text{def}} \{(v_i, u_0, \ldots, u_\ell) \in P | (u_0, \ldots, u_\ell) \in L'\}$ where we assume that each path occurs with maximum multiplicity prescribed by P. It is easily seen that if we fix any acyclic and valley-free orientation of L', then this orientation of L is acyclic and valley-free. Furthermore,

$$\|L\| \geq \|L'\| \geq \frac{2^{k-1}}{k!} \cdot \|P'\| \geq \frac{2^{k-1}}{k!} \cdot \frac{2}{k+1} \cdot \|P\| = \frac{2^k}{(k+1)!} \cdot \|P\|.$$

This proves the theorem. □

Observe that the proof of the theorem shows that each set of paths of length at most k contains at least a $\frac{2^k}{(k+1)!}$ fraction which allows an acyclic and valley-free orientation. Unfortunately, these fractions decrease very quickly as path length increases, e.g., for path length 2 the fraction is $\frac{2}{3} \approx 66.7\%$, for length 3 it is $\frac{1}{3} \approx 33.3\%$, for length 4 it is $\frac{2}{15} \approx 13.3\%$, and for length 5 it is already $\frac{2}{45} \approx 4.4\%$.

4.2 The Basic Heuristic

We turn to algorithms without proven approximation guarantees.

The algorithm from [16] for testing whether a path set allows acyclic and valley-free orientations is based on the observation that a leaf AS (i.e., one that itself has no customers) cannot be in the middle of any AS path (a fact which is also used in the proof of Theorem 6. We will describe it combined with the extension for discarding an interfering path, but before digging into the details we should fix some notation.

During its execution the algorithm will remove ASes which have been finished. To avoid having to change all the paths, we manage a set R of *removed* nodes. Given such a set, a node v in a path p is called *inner node of p relative to R* if it is surrounded in p by nodes u and w with $u, w \notin R$. A node not in R that is not an inner node for all paths of the paths set is called *free*.

In the algorithm (details in Algorithm 2) we count for each node v in count(v) the number of paths for which v is an inner node. The set F of free nodes is then easily initialized by all nodes with count = 0. The main loop (lines 9–28) can be separated into two phases. The first phase (lines 10–18) is taken from the algorithm in [16]. While there is a free node u in F we interpret it as a leaf AS and thus orient the edges to its neighbors (not yet in R) as customer-to-provider. As it is removed afterwards (*i.e.*, put into R) we adjust the count variables accordingly to find nodes which are now freed. If we can remove all ASes this way ($R = V$) we know that the orientation is valley-free and acyclic, as the nodes have been removed in topologically sorted order (each node had indegree 0 when it was removed).

If the first phase ran out of free nodes before all ASes could be removed, we need to create additional free nodes by discarding paths (starting from line 21). As we want to discard as little paths as possible, we select a node v which is an inner node for the minimal number of paths (as indicated by count(v)). By removing all those paths, v becomes a free node and we continue with the first phase.

Algorithm 2: Heuristic for Maximum Acyclic ToR

Input: AS path set P, AS graph $G = (V, E)$ for P
Output: A set N of discarded paths and an orientation of G with customer-to-provider edges such that the orientation is acyclic and valley-free for all paths in $P \setminus N$

```
1  foreach v ∈ V do count(v) := 0
2  foreach p ∈ P do
3      foreach inner node v of p relative to ∅ do
4          count(v) := count(v) + 1
5  F := ∅, R := ∅, N := ∅
6  foreach v ∈ V with count(v) = 0 do
7      F := F ∪ {v}
8  done := false
9  while ¬ done do
10     while F ≠ ∅ do
11         remove vertex u from F
12         foreach v ∈ V \ R with {u, v} ∈ E do
13             orient {u, v} as customer-to-provider
14             foreach p ∈ P containing u and v as neighbors do
15                 if v is an inner node of p relative to R then
16                     count(v) := count(v) − 1
17                     if count(v) = 0 then F := F ∪ {v}
18         R := R ∪ {u}
19     if R = V then
20         done := true
21     else
22         v := argmin_{u∈V\R} count(u)
23         foreach path p ∈ P with v ∈ p do
24             if v is an inner node of p relative to R then
25                 N := N ∪ {p}
26                 foreach inner node u of p relative to R do
27                     count(u) := count(u) − 1
28                     if count(u) = 0 then F := F ∪ {u}
29 return N
```

We want to point out that the algorithm can easily be modified to work for a weighted path set, where the goal consists of minimizing the overall weight of the dropped path. The count(v) variables then would not store the number of path for which v is an inner node, but the overall weight of those paths. All steps updating count(v) are adjusted accordingly for this.

4.3 Handling Pre-knowledge

If we already have partial information on the AS relationships we would like to incorporate this knowledge thereby improving the results of the algorithm. In [16] the influence of pre-knowledge on the complexity of testing whether an acyclic and valley-free orientation consistent with the pre-knowledge exists is discussed and an extension for the acyclic inference algorithm for handling known customer-to-provider edges is presented. As our heuristic is a modification of the algorithm given there, we can easily transfer this extension.

The idea is to introduce for each known customer-to-provider edge $u \to v$ a new path $(u, v, \perp)$, where $\perp$ is an artificial AS with count$(\perp) = \infty$. So the only way to make v a free node is to remove u which includes the introduction of an edge $u \to v$ as desired. Of course these new AS paths need not be constructed explicitly, but can be handled implicitly by modifying the heuristic above.

As the heuristic is allowed to drop paths hindering a consistent orientation, interpreting known edges as paths also allows dropping these edges in case of a conflict. Often however the explicit pre-knowledge is trusted more than the set of AS paths. For this case we can increase the weight of the (virtual) paths introduced in this step. In our implementation all AS paths are weighted with 1 and all paths originating from known edges are assigned the same weight W. Thus a customer-to-provider edge may only be discarded, if we can "save" at least W AS paths instead.

4.4 Readding

After finding an AS path set which can be oriented valley-free, the DPP* heuristic from [5] enters a second phase where edges which had to be dropped before are readded to the path set if possible. Two approaches for this are considered there. One is the voting process already described in Subsection 3.3, the other one is for each single path to add it to the path set and only keep it if a valley-free orientation is still possible.

As the readding stage reduces the number of invalid path by some extent we adapted this method for our heuristic. Unfortunately the simple and fast voting strategy will not work for our case as it does not necessarily preserve the global acyclicity. The alternative approach of adding paths one by one and retesting orientability does work but is quite expensive if the number of dropped paths is high (as already observed by [5]). Therefore we adjusted readding as follows.

The first run of the heuristic returns a set N of discarded paths, from which we can determine the set of orientable paths P. We decide on the number $k > 1$ of additional runs we are willing to spend and partition N arbitrarily into k sets $N_1, \ldots, N_k$. For each such set N_i we then run the heuristic above for the path set $P \cup N_i$ and again receive a set of discarded paths N'. If $|N'| < |N_i|$ we could reduce the number of dropped paths and use $P := (P \cup N_i) \setminus N'$ from now

on, otherwise we stick with the original P. As we treat edges from the pre-knowledge as AS paths as described before, this strategy will work for readding those discarded edges as well.

5 Obtaining Real-World Data

To run the inference algorithms we need valid AS paths used in the Internet. Additionally we are interested in at least partial information on the business relationships between autonomous systems, to both verify our inference results as well as using them as previous knowledge for the inference algorithms. Unfortunately there is no single exhaustive source listing those relationships, so we have to use other publicly available information and try to extract them from it. We will discuss the sources and methods used to retrieve the data used herein in the following subsections. All data was collected on 3/31/2006. In case of multiple data samples for this day (as offered by some archives) the latest one available for the day was used.

5.1 AS Paths

We extracted AS paths from the routing tables available through the route collectors from RIPE RIS [21] and the Route Views project [25]. Both sites operate several *BGP beacons*, modified BGP routers which exchange routing information via peering sessions with other routers but do not participate in actual routing (i.e., they announce no routes). Both sites provide an archive with the routing tables of these beacons from which AS paths can be extracted. We used the last routing table available for 3/31/2006 from each beacon available (for RIPE RIS these are rrc00 to rrc07, and rrc10 to rrc15, for RouteViews these are RouteViews2, EQIX, ISC, KIXP, LINX, and WIDE).

The second source for AS paths are the public route servers listed at [15]. We established a telnet session and obtained their routing tables using the `show ip bgp` command. Due to technical issues (time-outs, etc.) not all of them could be collected. The routing tables used are listed in Figure 3.

By design the AS paths used in BGP contain no cycles (i.e., duplicate AS numbers on the same path). The only exception is *prepending* where the same AS appears multiple times at *consecutive* positions. As prepending is only relevant for routing decisions but not for the inference algorithms discussed here, we preprocessed the paths by discarding all but the first AS from a sequence of identical AS numbers. Even after this preprocessing there are still paths containing cycles or ASes from the private range (AS64512 to AS65534), probably due to configuration errors. Such paths were completely removed from the data pool.

Finally we normalized the path set to include only those paths that were not already implied by other AS paths. More formally, an AS path is dropped if itself or the reversed path is a substring (including identity) of another AS path in the data pool. Using the source presented above and the preprocessing and normalization procedure we received a set of 2002714 AS paths with an average path length of 3.44.

5.2 RPSL Information from WHOIS

Our primary data source for relationships are the WHOIS databases which besides other information contain structured records for autonomous systems (`aut-num`). A list of avail-

AS553 (BelWue)
AS852 (Telus - East Coast)
AS852 (Telus - West Coast)
AS3257 (Tiscali)
AS3561 (Savvis Communications)
AS3741 (Internet Solutions)
AS4323 (Time Warner Telecom)
AS5388 (Planet Online)
AS5511 (Opentransit)
AS6539 (GT Group Telecom)
AS6648 (Bayantel Inc.)
AS6667 (Eunet Finland)
AS6730 (Sunrise)
AS6939 (Hurricane Electric)
AS7474 (Optus Route Server Australia)
AS7911 (Wiltel)
AS8220 (Colt Internet)
AS9132 (Broadnet Mediascape Communications AG)
AS12312 (Tiscali Germany)
AS13645 (Host.net)
AS15290 (Allstream - East)
AS15290 (Allstream - West)

Figure 3: The routing tables extracted from routers listed at [15]

```
aut-num:  AS123
import:   from AS456 accept ANY
export:   to AS456 announce AS456

aut-num:  AS456
import:   from AS123 accept AS123
export:   to AS123 announce ANY
```

Figure 4: A simplified WHOIS record

able databases can be found at [17]. We decided to use the databases from APNIC, ARIN, LEVEL3, RADB, RIPE, and VERIO. Other approaches for extracting business relationships from WHIOS data are described in [6, 22]. As we are only interested in a set of relationships to test our inference results with and not in a complete view of the Internet, we use a simpler approach which is motivated by those mentioned before and described next.

The most valuable information in these aut-num records are the import and export rules which encode the routing policy of the autonomous system using RPSL (Routing Policy Specification Language [2]). As RPSL is a very rich language (the Bison grammar given in [2] covers 8 pages) we do not try to understand all of it but discard all rules not of the most primitive format which is the number of the AS these rules apply to, followed by a list of autonomous systems (possibly including `as-sets`), routes to which are imported respectively exported. To indicate the lack of filtering the list can be replaced by the `ANY` keyword. An example of such rules is given in Figure 4. As most rules are using this simple format, we are not losing to much information. To get rid of conflicting entries, all records appearing in multiple databases are skipped. Additionally we discard all records not modified since 1/1/2005 to get rid of unmaintained entries.

When processing the record of some AS A we check each AS B for which both import and export policies have been parsed. These policies can be roughly categorized into *unfiltered* (if `ANY` is used) and *filtered* (if a list of AS numbers follows). The only rules used in the valley-free path model are to export anything or to export only own and customer routes. These map nicely with our filtered and unfiltered policies, so having both the import and export rules between A and B we can directly infer one of the four relationship types of the valley-free path model. In the example given in Figure 4 AS123 exports only own routes to AS123 but imports everything from it, so the export is filtered while the import is unfiltered. Thus we conclude AS456 to be a provider of AS123. If both were filtered, we would infer a peer-to-peer relationship, and unfiltered import and export indicate sibling-to-sibling relationship. However as we do not expect ISPs to document internal routing policies in public registries, such sibling-to-sibling links are considered configuration errors (or documentation inconsistencies) and discarded.

As the information in the WHOIS tables is usually provided voluntarily and often outdated, we can not expect the results from this approach to be absolutely correct. Siganos and Faloutsos [22] estimate the correctness of their refined approach to be higher than 83% due to errors in the WHOIS tables. On the other hand they are analyzing more complicated rules and try to get a more complete picture of the Internet relationships than we do here. As we only need partial information and may drop everything not fitting with our assumptions we expect to have more accurate results.

```
aut-num:  AS8437
as-name:  UTA-AS
  ...
remarks:  Communities tagged on inbound routes
remarks:  ====================================
remarks:
remarks:  8437:1000 Customer Routes
remarks:  8437:1004 Peering Routes
```

Figure 5: Example for documenting communities in the aut-num record

One step towards improving the quality of the extracted relationship information is exploiting the symmetry of the import and export rules. In Figure 4 we concluded from the record for AS123 that AS456 is a provider for it. From the record for AS456 we can see AS123 to be its customer which is consistent with this information. Ideally for each pair of ASes we have two characterizations of their relationships. If they are consistent (both peer-to-peer or customer-to-provider with provider-to-customer) our trust in this information increases, otherwise we discard the information for this AS pair. As many autonomous systems do not disclose their routing policy, we often only find one characterization for a relationship. We keep this information separate and use it for testing the robustness of the inference algorithms on erroneous data.

From the WHOIS databases listed above, we extracted 4438 edges which are confirmed twice by the RPSL statements and 33828 that were only encountered once. Because the inference algorithms only find relationships for AS pairs adjacent on at least one path, we removed all AS pairs from above edge sets not neighbored on any path. After this cleaning process there are 964 customer-to-provider and 598 peer-to-peer relationships in the first set and 6564 customer-to-provider and 3988 peer-to-peer edges in the second set.

5.3 Using BGP Communities

In [27] a different approach for extracting relationships using the BGP communities attribute (introduced in [24]) is described. BGP communities are arbitrary numbers that are tagged to routes and can be used in routing decisions. To make these numbers unique, the AS number of the AS which added the community is included in the community number. Besides influencing other routing parameters (such as local-pref and prepending) communities are often used to document the origin of a route including whether they were received from a provider or from a peer. Unfortunately there is no general agreement on the usage of community numbers, so their meaning depends on the originating AS. Furthermore there is no unique way of documenting them, however some ISPs are using the `remarks` entries in the WHOIS aut-num records for this purpose (an example is given in Figure 5).

As the contents of the remarks section are not standardized it is complicated to automatically extract the meaning of documented communities, so we performed this step manually. Although harvesting the entire WHOIS databases for this information by hand is nearly impossible, limiting this search to remark lines containing an additional colon and one of the phrases `peer, customer, provider, uplink, upstream` makes this a manageable task.

Now if we are given an AS path tagged with a community from our list, we can infer the relationship for one AS pair. For example let the path be `AS1 AS2 AS8437 AS3 AS4` and the communities contain `8437:1000` then we know that AS8437 received the path `AS3 AS4` from its customer AS3, so we can infer a customer-to-provider edge. With a list of known communities extracted from all WHOIS databases above, we used this approach on all the routing tables collected from RIPE RIS and Route Views. The routing tables obtained using `show ip bgp` cannot be used as they lack information about communities. Although many communities are discarded by routers on the path, enough of them are in the routing tables to allow us finding 1882 distinct customer-to-provider and 1495 peer-to-peer edges. Due to the method used, they are automatically on an AS path.

5.4 Sibling Relationships

The approaches presented so far find no sibling-to-sibling edges, as ISPs only document their *external* routing behavior. To get around this we assume that all autonomous systems belonging to the same organization are siblings of each other, which is a sensible assumption under economic aspects. Further we expect a company to have the same contact persons for all its autonomous systems, so aut-num records having different entries in one of the `mnt-by`, `admin-c`, `tech-c` fields are considered not to be siblings. From the remaining pairs of "sibling candidates" we only keep those sharing a long common prefix or suffix in the name or the description of the ASes. This gives a set of potential sibling clusters which are then manually verified in a final step.

It turns out that using only the information from WHOIS we sometimes cannot decide whether two autonomous systems are siblings or if one is a customer AS which is only maintained by its provider. As only small ASes at the border of the Internet are likely to delegate their administration, and introducing a wrong sibling-to-sibling relationship in this case does not do too much harm, we decide for keeping the sibling-to-sibling edge when in doubt. This way we obtain 5301 sibling-to-sibling edges from 360 sibling cliques.

Contrary to the customer-to-provider and peer-to-peer edges the sibling-to-sibling edges are not used to validate the results of the algorithms but rather to preprocess the input data by replacing each AS number with a representative sibling in all paths and edges.

5.5 Summary of Data Used

We want to summarize the data used for the experiments presented later. For each experiment we used the complete AS path set as described before. The edges are separated into two sets, which we call the *reliable set* and the *full set* respectively. In the reliable set there are the customer-to-provider and peer-to-peer edges obtained from BGP communities and those appearing twice in the RPSL extracted edges. We expect this set to contain only little errors. The full set consists of the reliable set and additionally all edges appearing only once in the RPSL edges. This set is especially useful for testing the algorithms with uncertain preknowledge, as probably 5 to 20 percent of the information contained there is incorrect.

As a preprocessing step for each experiment we used the sibling edges obtained before to replace all sibling ASes with a single AS. This slightly influences the numbers reported before, as some paths and edges had to be dropped due to being malformed after this step.

AS paths	2002680 paths (average length 3.43) containing 21862 ASes and inducing 56922 AS pairs
reliable edge set	2739 customer-to-provider edges 2000 peer-to-peer edges
full edge set	8850 customer-to-provider edges 5434 peer-to-peer edges

Figure 6: The data used for the experiments after all preprocessing steps (path normalization, dropping unused edges, replacing siblings)

Additionally inconsistent edges (e.g., a peer-to-peer and customer-to-provider relationship for the same edge) were removed. The details for the final path and edge sets ultimately used as input and reference for the algorithms are shown in Figure 6.

6 Validating the Acyclicity Assumptions

Using the data from the previous section we intend to compare our acyclicity model to the real Internet. Therefore we build graphs from the edges collected and check if they are acyclic according to our formulation.

The graphs we receive when only using customer-to-provider relationships are acyclic both for the reliable and the full data set. However including peer-to-peer edges into these graphs creates cycles in both cases. To get an impression of how much acyclicity is violated we tested every triangle in the graphs whether it was a directed circle or not. For the reliable edge set the graph contains 2826 triangles from which 253 (9%) induce a directed cycle. In full edge set there are 18621 triangles, 2427 (13%) of them cyclic.

We take these results as an indication that indeed the overall structure of AS relationships is acyclic but our model of acyclicity is still imprecise when it comes to peer-to-peer relationships. This is probably mostly due to assuming the relation "roughly of the same size" to be transitive when interpreting peer-to-peer edges.

7 Experimental Findings

To compare our acyclic inference heuristic (to which we refer as AHeu here) to existing algorithms we executed all of them on the path set from Section 5 and compared the resulting edge classification to the reliable and full edge set described there. We are interested in both the number of paths which are not oriented correctly (i.e., are not valley-free) and the number of customer-to-provider edges that were not inferred as such. As none of the algorithms used is capable of identifying peer-to-peer relationships we do not compare the inferred results to our known peer-to-peer edges. An exception is Gao's algorithm as it introduces sibling-to-sibling edges which we do not want to include in the inferred results, thus paths containing a sibling-to-sibling edge are counted as invalid as well.

The detailed results of our experiments is shown in Figure 7. As seen our heuristic has the lowest number of invalid paths as well as the least number of errors when compared to the reference data. Additionally it is exemplified how using the readding strategy we can

Algorithm	Invalid paths	Misclassified c-to-p for reliable edge set	Misclassified c-to-p for full edges set
Gao	27.366% (249 not valley-free) (547811 with s-to-s edge)	1.387% (4 as p-to-c) (34 as s-to-s)	6.814% (484 as p-to-c) (119 as s-to-s)
APX	4.483% (89775 not valley-free)	5.330% (146 as p-to-c)	9.458% (837 as p-to-c)
DPP*	0.519% (10391 not valley-free)	0.913% (25 as p-to-c)	6.915% (612 as p-to-c)
AHeu ($W = 10$)	0.483% (9666 not valley-free)	0.292% (8 as p-to-c)	5.073% (449 as p-to-c)
AHeu ($W = 10$) readd($k = 10$)	0.413% (8278 not valley-free)	0.329% (9 as p-to-c)	5.469% (484 as p-to-c)

Figure 7: Results for inferring only customer-to-provider relationships

lower the number of invalid paths even more but at the cost of reliability of the resulting edge classifications.

Another aspect we are interested in is the behavior of these algorithms when having initial pre-knowledge of some of the edges. Therefore we repeated the experiment described before but provided the algorithms with a certain fraction of the edges used for comparison later. As the choice of edges provided to the algorithm has some influence on its results we averaged all results over 5 random permutations of the edge set. Additionally the same permutations were used for all of the algorithms. Our heuristic is the only one explicitly supporting pre-knowledge, so we used the preprocessing described in Section 3.4 to augment the remaining algorithms accordingly. This should be kept in mind when comparing the results as thus our heuristic is the only one capable of "trading" edge errors (i.e., violated pre-knowledge) for violated paths.

An impression on how much of the classification is already implied by the pre-knowledge Figure 8 shows the number of edges fixed by our preprocessing step when using the reliable data set. Obviously a small number of known edges already determines a rather large part of the final classification while on the other hand increasing the pre-knowledge seems to have little influence. On the other hand providing additional knowledge might even lead to inconsistencies as is shown by Figure 9 which lists the number of conflicts occurring during the preprocessing phase.

As stated before we are interested both in the number of misclassified edges and the number of invalid paths. These numbers are given in Figures 10 resp. 11 using the reliable edge set both as pre-knowledge and for comparison. According to these plots the performance of our heuristic is hardly influenced by the amount of pre-knowledge available which we take as an indication for the high quality of the inferred results.

We provide the analogous plots for the full data set in the appendix. They show the same trend with the difference of a higher number of overall errors due to the inexact data in this set. Figure 14 illustrates nicely how our heuristic keeps giving good results even in the presence of (partially) invalid pre-knowledge while the other algorithms (due to the inflexible preprocessing step) have to trust this knowledge at the cost of a huge number of invalid paths.

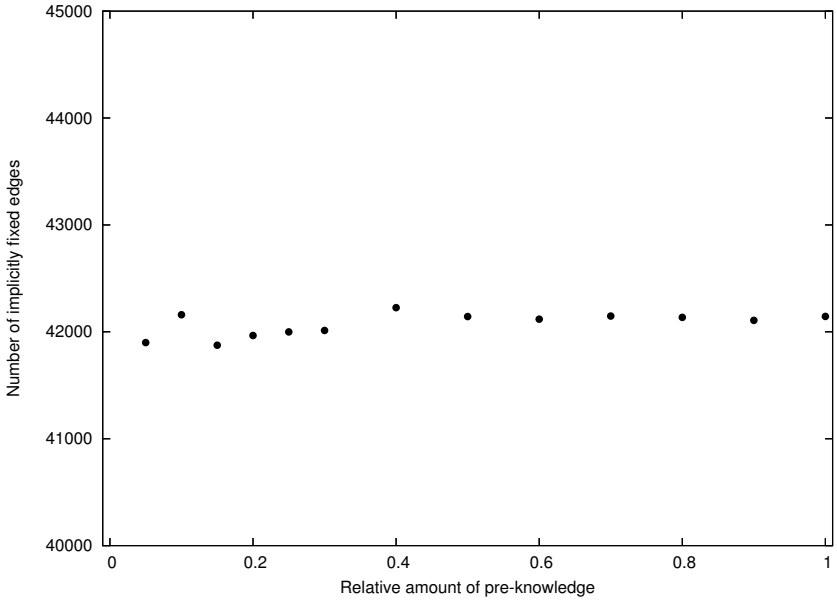

Figure 8: Number of edges implicitly fixed by pre-knowledge (reliable edge set, average over 5 random permutations of the edge set)

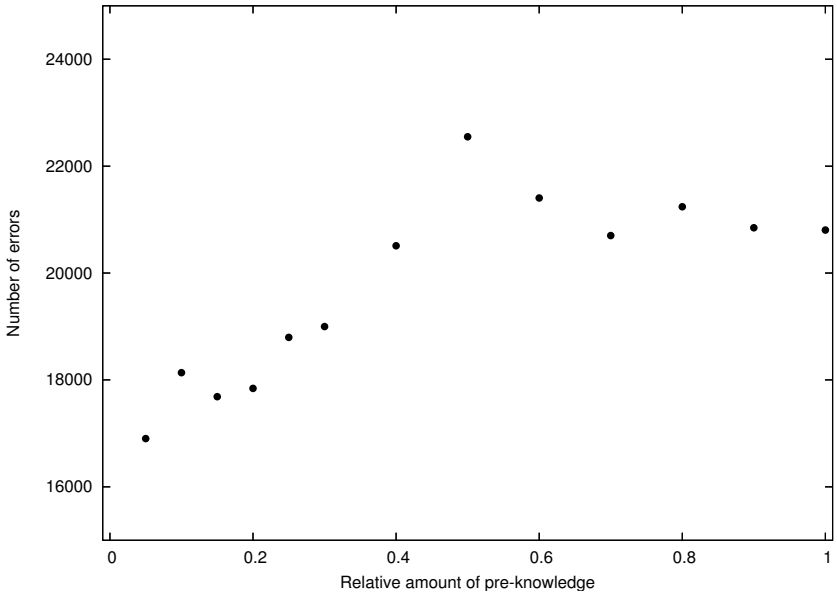

Figure 9: Number of errors encountered during inferring edges from pre-knowledge (reliable edge set, average over 5 random permutations of the edge set)

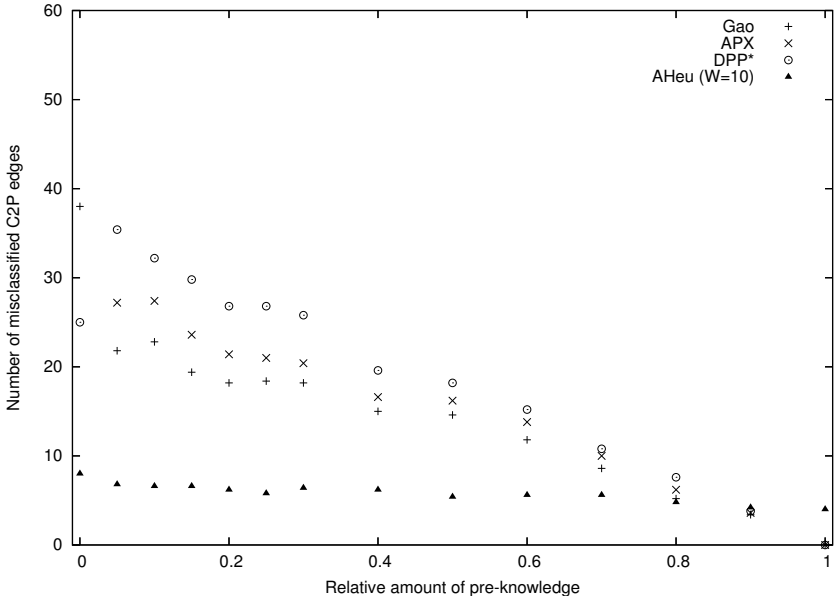

Figure 10: Number of edges misclassified by the algorithms for different amounts of pre-knowledge (reliable edge set, average over 5 random permutations of the edge set)

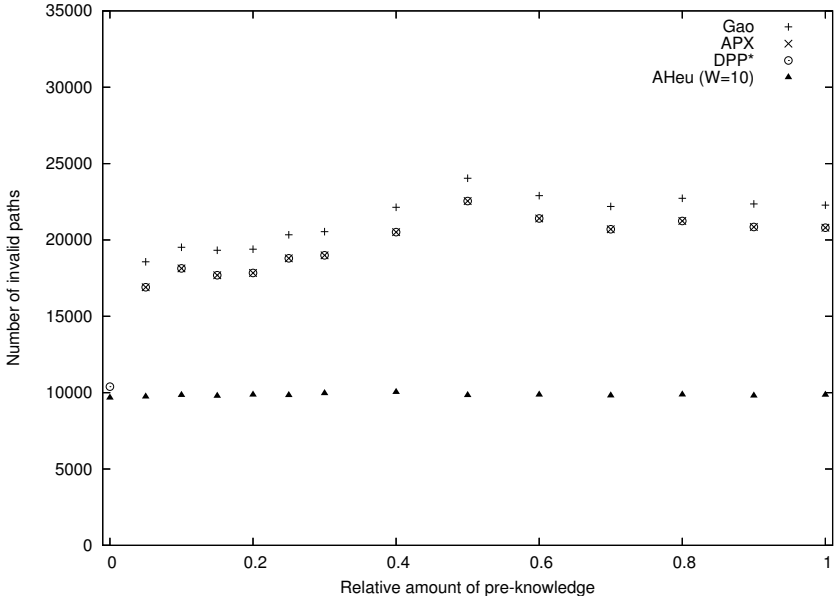

Figure 11: Number of invalid paths for the orientation obtained from the algorithms for different amounts of pre-knowledge (reliable edge set, average over 5 random permutations of the edge set)

8 Conclusion

We studied the acyclicity approach to AS relationship inference introduced in [16] from an experimental point of view. On the one side, we presented both theoretical and practical evidence that this approach is feasible and, in large parts, accurate. The described, heuristic algorithm AHeu turned out to be easily implementable, fast, and flexible with respect to incorporating initial pre-knowledge. On a reliable data set, it outperformed most of the state-of-the-art algorithms proposed in the literature. Moreover, the acyclicity of all customer-to-provider relationships with in the reliable data set could be confirmed. These findings suggest to integrate acyclicity notions in detailed models of AS relationships.

On the other side, we have learned that acyclicity with respect to peer-to-peer relationships is not yet fully captured. The underlying assumption that the roughly-the-same-size relation is transitive seems too much a simplification. We consider finding a more accurate problem formulation involving acyclicity and peer-to-peer relationships as the main open issue of this paper.

Acknowledgement. We thank Wolfgang Mühlbauer for helpful discussions.

References

[1] C. Alaettinoğlu. Scalable router configuration for the Internet. In *Proceedings of the 5th International Conference on Computer Communications and Networks (ICCCN'96)*. IEEE Computer Society Press, Washington, D.C., 1996.

[2] C. Alaettinoğlu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. Routing Policy Specification Language (RPSL). RFC 2622, The Internet Society, 1999.

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, Berlin, 1999.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2nd edition, 2001.

[5] G. Di 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*, 2006.

[6] G. Di Battista, T. Refice, and M. Rimondini. How to extract BGP peering information from the Internet Routing Registry. In *Proceedings of the ACM SIGCOMM 2006 Workshop on Mining Network Data (MineNet'2006)*, pages 317–322. ACM Press, New York, NY, 2006.

[7] X. A. Dimitriopoulos, D. V. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, K. C. Claffy, and G. F. Riley. AS relationships: Inference and validation. *ACM SIGCOMM Computer Communication Review*, 37(1):31–40, 2007.

[8] X. A. Dimitriopoulos, D. V. Krioukov, B. Huffaker, K. C. Claffy, and G. F. Riley. Inferring AS relationships: Dead end or lively beginning? In *Proceedings of the 4th International Workshop*

on Experimental and Efficient Algorithms (WEA'05), volume 3503 of *Lecture Notes in Computer Science*, pages 113–125. Springer-Verlag, Berlin, 2005.

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

[10] R. Govindan and A. Reddy. An analysis of Internet inter-domain topology and route stability. In *Proceedings of the 16th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'97)*, pages 850–857. IEEE Computer Society Press, Washington, D.C., 1997.

[11] T. G. Griffin and G. T. Wilfong. An analysis of BGP convergence properties. *ACM SIGCOMM Computer Communication Review*, 29(4):277–288, 1999.

[12] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an autonomous system (AS). RFC 1930, The Internet Society, 1996.

[13] G. Huston. Interconnection, peering and settlements—Part II. *The Internet Protocol Journal*, 2(2):2–23, 1999.

[14] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, NY, 1972.

[15] T. Kernen. traceroute.org web site. `http://www.traceroute.org`.

[16] 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'06)*, volume 4235 of *Lecture Notes in Computer Science*, pages 98–111. Springer-Verlag, Berlin, 2006.

[17] Merit Network Inc. Internet Routing Registry. `http://www.irr.net`.

[18] W. B. Norton. Internet Service Providers and peering. Equinix White Paper, Equinix, Inc., Foster City, CA, 2001.

[19] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[20] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, The Internet Society, 1995.

[21] RIPE NCC. Routing Information Service (RIS). `http://www.ripe.net/ris/`.

[22] G. Siganos and M. Faloutsos. Analyzing BGP policies: Methodology and tool. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, pages 1640–1651. IEEE Computer Society Press, Washington, D.C., 2004.

[23] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the Internet hierarchy from multiple vantage points. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, pages 618–627. IEEE Computer Society Press, Washington, D.C., 2002.

[24] P. Traina, R. Chandra, and T. Li. BGP community attribute. RFC 1997, The Internet Society, 1996.

[25] University of Oregon. Route Views project page. `http://www.routeviews.org`.

[26] I. van Beijnum. *BGP*. O'Reilly & Associates, Sebastopol, CA, 2002.

[27] J. Xia and L. Gao. On the evaluation of AS relationship inferences. In *Proceedings of the 47th Annual IEEE Global Telecommunications Conference (Globecom'04)*, volume 3, pages 1373–1377. IEEE Communication Society, New York, NY, 2004.

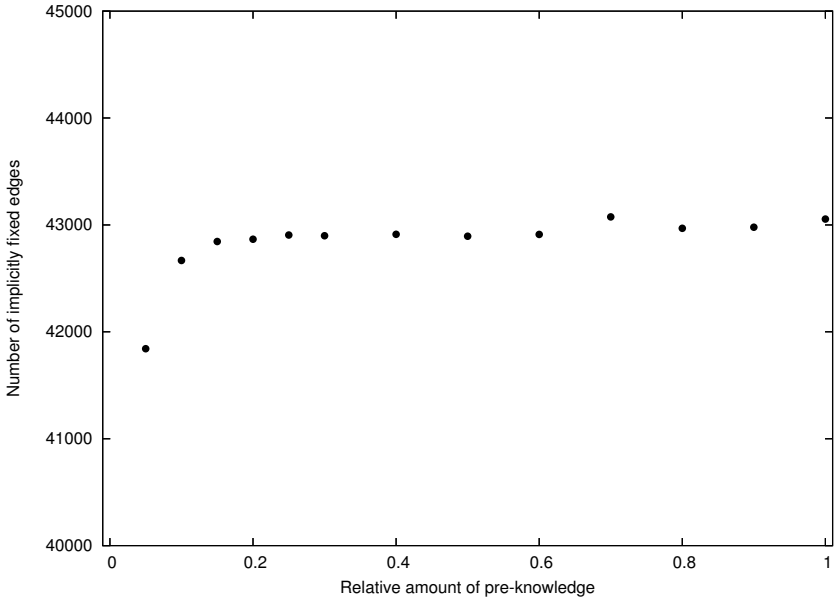

Figure 12: Number of edges implicitly fixed by pre-knowledge (full edge set, average over 5 random permutations of the edge set)

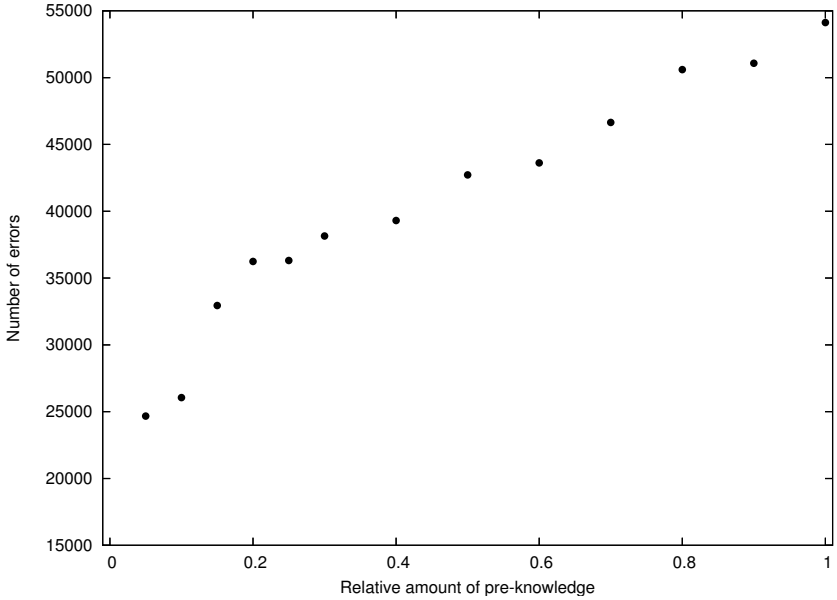

Figure 13: Number of errors encountered during inferring edges from pre-knowledge (full edge set, average over 5 random permutations of the edge set)

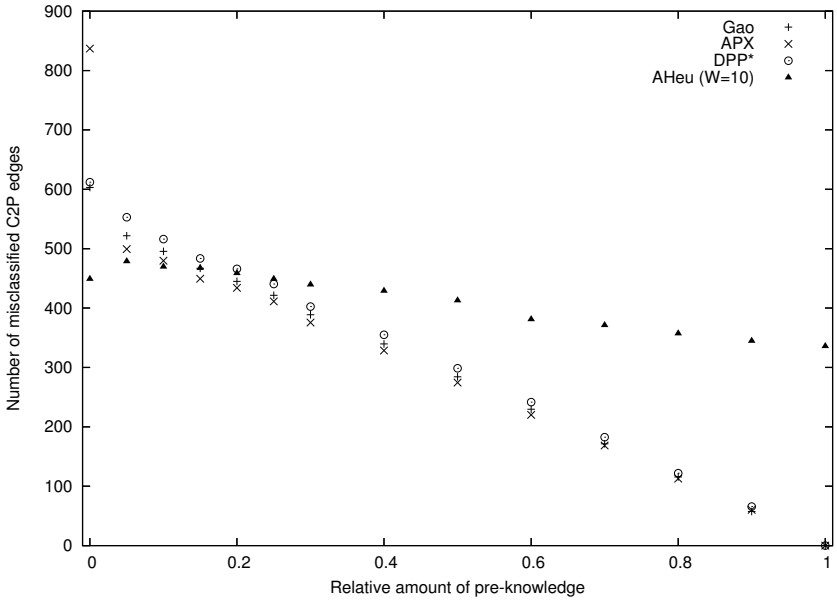

Figure 14: Number of edges misclassified by the algorithms for different amounts of pre-knowledge (full edge set, average over 5 random permutations of the edge set)

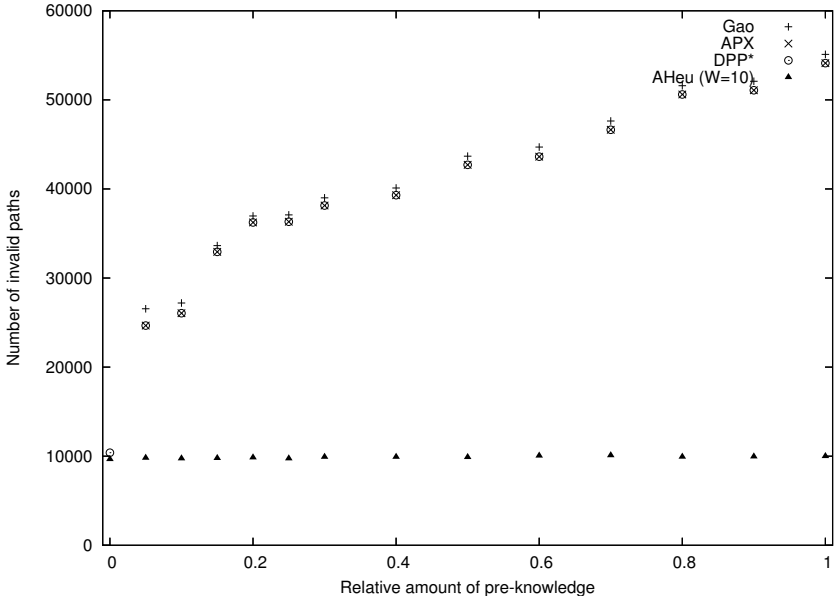

Figure 15: Number of invalid paths for the orientation obtained from the algorithms for different amounts of pre-knowledge (full edge set, average over 5 random permutations of the edge set)