
**Master's Thesis**

Submitted to the Department of Computer Science
Technische Hochschule Ingolstadt

# BGPsyche: AS Path Predictions beyond Valley-Free Routing

Author: **Dominik Stahmer**

Issued on: 2024-01-31
Submitted on: 2024-07-30
First Examiner: Prof. Dr. rer. nat. Franz Regensburger
Second Examiner: Dr. rer. nat. Johann Schlamp

# Acknowledgments

This page is dedicated to the people without whom this thesis would not exist. There are too many people to list here that helped me with their expertise, support or even a simple joke or act of kindness at the right time. Still, I would like to specifically mention, in no particular order, my advisor Johann Schlamp for his patience in discussing and explaining the intricacies of Internet routing and overall guidance, my family for their love and continued support of my studies and finally my good friend Philip Gaber for helping with proof-reading and being an awesome human being.

## Abstract

The BGP routing protocol is one of the essential building blocks of the modern Internet. It allows networks to discover paths to one another on a global scale. While individual BGP participants know how to reach every other participant, there is no way to know the AS path between two arbitrary networks. Network operators however have a need to know these paths to, for example, deploy security mechanisms and optimize peer to peer applications. Over the years, many heuristics have been developed to predict these inter-domain paths. However, all of them still fall short of being truly reliable, which justifies further research in the field. In this thesis, we develop a new AS path prediction system *BGPsyche*. To our knowledge, it is the first of its kind to be based on deep learning. BGPsyche can find the correct path in the set of its top three selected candidates **85 %** of the time and selects the correct path as its top candidate with an accuracy of 66 %. This makes it about as accurate as most of its competition, with only one system yielding a superior accuracy. Because of the easy extensibility of BGPsyche, we think that there is still more potential in our approach. In building BGPsyche, we also analyze many features of ASes, links and paths from the perspective of a machine learning application, which may serve as a basis for future research. Notably, we show that including the widely used valley-free constraint in our training data worsens the result, indicating that BGPsyche has learned something more nuanced on its own.

## Kurzfassung

Das BGP Routing Protokoll ist einer der grundlegenden Bausteine des modernen Internets. Es erlaubt Netzwerken auf einer globalen Ebene Pfade zueinander zu finden. Individuelle BGP Teilnehmer wissen zwar, wie von ihnen aus jeder andere Teilnehmer zu erreichen ist, es ist aber nicht möglich, AS Pfade zwischen zwei beliebigen Netzwerken mit Gewissheit zu berechnen. Netzwerkbetreiber haben allerdings ein berechtigtes Interesse diese Pfade zu kennen, um zum Beispiel Sicherheitsmechanismen zu installieren oder Peer-To-Peer Anwendungen zu optimieren. Daher wurden über die Jahre viele Heuristiken entwickelt um AS Pfade vorherzusagen. Allerdings motiviert die mangelnde Zuverlässigkeit dieser Heuristiken weitere Forschung in diesem Gebiet. Deshalb entwickeln wir in dieser Arbeit *BGPsyche*, ein neues System zur Vorhersage von AS Pfaden. Nach unserem Kenntnisstand ist es das erste seiner Art, welches auf Deep Learning aufsetzt. BGPsyche kann den korrekten Pfad in einer berechneten Menge von drei Pfad-Kandidaten mit einer Wahrscheinlichkeit von **85 %** finden. Außerdem findet es in 66 % der Fälle den exakt richtigen Pfad. Damit erreicht es im Vergleich mit den meisten konkurrierenden Systemen eine ähnliche Zuverlässigkeit; Nur ein System ist genauer. Aufgrund der einfachen Erweiterbarkeit von BGPsyche sind wir der Meinung, dass in unserem Ansatz noch deutlich mehr Potenzial steckt. Um BGPsyche zu entwickeln, untersuchen wir außerdem viele Attribute von ASen, Links und Pfaden aus der Perspektive des Machine Learning, was als Grundlage für zukünftige Forschungsarbeit dienen kann. Insbesondere zeigen wir, dass die Verwendung der weit verbreiteten valley-free Regel in unseren Trainingsdaten das Ergebnis verschlechtert, was darauf hinweist, dass BGPsyche selbst eine nuanciertere Regel lernen konnte.

# Contents

# 1. Introduction

The Border Gateway Protocol (BGP) is a fundamental building block of the modern Internet, allowing routers to announce and discover paths between networks on a global scale. While a router participating in BGP will have an understanding of how to route an incoming packet from its position, it has no inherent knowledge of how a packet from an *arbitrary starting position* might be routed. This thesis seeks to make a contribution towards finding an accurate heuristic to predict these BGP paths.

## 1.1. Motivation

Network operators have a need to know arbitrary BGP paths to improve their routing policies to, for example, deploy security mechanisms [1, 2] and optimize peer to peer applications [3, 4].

It is only possible to know real inter-domain paths with access to the source network. Many big networks allow the public to inspect their routing database by hosting a looking glass server[1], but most smaller networks do not offer such a service. Additionally, some networks participate in a service like RIPE Atlas[2], which allows users to run 'traceroute' from any member network. However, even when an IP level path is discoverable through such a service, it is still far from trivial to translate that to a BGP path. Therefore, network operators cannot rely on discovering paths using traceroute or looking glass servers for every path they wish to know. (Looking glass servers do allow a direct lookup of BGP routes instead of running traceroutes, but the source network of interest may not provide such a service.)

Thus, a solution to heuristically infer BGP paths is needed. Finding a good heuristic has been the topic of research for many years [5, 3, 6, 7, 4, 8, 9, 10, 11, 12]. These works will be discussed further in Chapter 3. For now, we ask the reader to accept that these existing systems, while often impressive, have shortcomings that make them deserving of competition. To summarize briefly, some of them can only predict certain kinds of paths, others rely on infrastructure that is no longer available and can be difficult to set up. Notably, there is a surprising lack of research on the utility of neural networks or even machine learning in general for path prediction.

We are therefore motivated to contribute both an alternative to existing BGP path prediction solutions and develop an understanding of the utility of neural networks in this task.

[1] For example `https://lg.he.net`, `https://lookingglass.telekom.com`
[2] `https://atlas.ripe.net`

## 1.2. Key Contributions

This thesis discusses the design and evaluation of *BGPsyche*, a novel BGP path prediction system based on machine learning. The system is trained on publicly available data and can ingest data on ASes, links and entire paths. We compare the accuracy of BGPsyche to existing systems and discuss open questions for further research. We have published the source code of a prototype implementation on GitHub [13].

We find that:

- BGPsyche finds the correct path between an arbitrary source and destination network in its set of *top three* selected candidates **85 %** of the time and selects the correct path as its *top* candidate in **66 %** of cases. The average edit distance between the selected path and the real path is 0.5. This performance is roughly in line with most state-of-the-art BGP path prediction engines.
- The system is easily extensible in the sense that developers and researchers can easily add new features about ASes, links and paths to potentially improve performance.
- Once trained, BGPsyche can run even on relatively low-end hardware. A single core of a typical laptop CPU can predict a single path in about 3 s.

Additionally, we have analyzed the nature and utility of many data points in our pursuit to build BGPsyche. This could serve as a starting point for potential future research into the use of machine learning algorithms for BGP path prediction. Our results also suggest that this research is likely worth doing, as we have arguably reached state-of-the-art performance and still have immediate and easy to implement ideas for potential further improvement.

# 2. Background

This chapter introduces some the core technologies and ideas necessary to discuss the implementation of BGPsyche. Specifically, we give an introduction to BGP routing, path prediction, as well as relevant datasets.

## 2.1. BGP Routing

BGP as defined in RFC4271 [14] (and many others) is a somewhat notoriously complex protocol. This thesis will largely not mention the many edge-cases and extensions to the protocol, and so this section will give an overview of just the basic properties of BGP.

The Internet relies on BGP to exchange routing information between networks. BGP is also sometimes spoken internally within a network, in which case it is denoted as iBGP [14]. This thesis however concerns itself with the use of BGP between networks (eBGP). In all further sections, the acronym BGP should be understood as a shorthand for eBGP unless explicitly stated otherwise.

BGP speaking networks are referred to as Autonomous Systems (ASes). Each AS is assigned a 32-bit (or legacy 16-bit) unsigned integer, the ASN. These are assigned by Regional Internet Registries (RIR) [15], such as the RIPE NCC (Réseaux IP Européens Network Coordination Centre) in Europe.

RIRs are also responsible for the delegation of IP addresses, grouped into IP *prefixes*. Prefixes are usually represented in CIDR (Classless Inter-Domain Routing) [16] notation, consisting of an address and the $log_2$ of the network length. For example, the prefix `192.168.178.0/24` refers to the group of all IP addresses from `192.168.178.0` to `192.168.178.255`. The ultimate goal of BGP is to distribute information about how to reach every publicly routed prefix.

To participate in BGP routing, each AS $a$ sets up a router with a direct physical connection to some (often small) set of other ASes $P$. We say that $a$ is a BGP *peer* of all ASes in $P$. Each AS announces a list of AS paths, including destination IP prefixes, to its peers. This list also includes the IP prefixes that $a$ is configured to announce itself. These peers then decide, based on a configured policy, whether to use this path for their own routing and whether to further announce this path to their peers (after appending their own ASN to the path). Because of this behavior, BGP is often referred to as a path-vector routing protocol [17].

While routers still generally prefer shorter paths and a longer prefix match, the protocol allows for other priorities to be expressed in configuration. An Internet Service Provider (ISP) generally has a financial interest in routing traffic over its customers rather then its upstream ISPs and can configure its policies accordingly. Any attempt at BGP path prediction must attempt to approximate these policies, which are often not publicly disclosed.

BGP peerings are often categorized according to the nature of the underlying intentions and business relationship of the participating ASes. For the purposes of this thesis, we must differentiate between two categories:

- Two ASes are said to have negotiated a *peering agreement* when they share access to some amount of their downstream customer base with each other free of charge for mutual benefit. This is also often referred to as a peer-to-peer relationship [18], which we abbreviate to *p2p*. It is not to be confused with the general process of establishing a BGP session, which is also often referred to as a peering. Peering agreements can be further divided into public and private peerings. Public peering is done at an Internet Exchange Point (IXP), usually between many parties at once. A private peering is negotiated between two parties directly.
- In contrast, a BGP session is said to be a *transit* connection when one AS $a$ pays the other AS $b$ for access to some of $b$'s downstream customer base. This is usually done when $b$ has access to a significantly larger portion of the Internet then $a$ and therefore stands to gain relatively little from gaining access to $a$'s customers. A transit agreement is also often referred to as a customer-to-provider [18] relationship, which we again abbreviate to *c2p*.

The proverbial top of the food chain of ASes is occupied by so called "Tier 1" or "clique" [19] ASes. These do not need to pay another provider AS to access the entire Internet and can instead rely on mutual peering agreements with other Tier 1 ASes.

## 2.2. The Utility of Path Prediction

Outside of pure academic interest, the ability to accurately predict AS path would be very useful in a variety of scenarios. This section therefore serves to further motivate the implementation of BGPsyche. To that end, we list three use-cases of path prediction:

1. **Optimizing Peer-To-Peer Applications:** A participating "client" in a peer-to-peer application often has to choose which peer or relay server to connect to in order to ensure a fast and stable connection. A common example for such an application are centrally managed, but still distributed Voice-over-IP services (e.g. Signal[1]). If a direct peer-to-peer connection cannot be established, a relay server is chosen to make the call. In order to determine which of the available relay servers will likely provide the optimal performance, the application first has to predict the paths toward these relay servers and estimate various performance metrics on these paths. Doing this on-demand by running traceroutes however would be so slow, that it would dwarf the performance gain of choosing the best path. Thus, a fast heuristic for predicting paths is desirable [4, 3].
2. **(Heuristic) Route Origin Validation:** BGP by default does not provide a security mechanism to prevent the operator of an AS to announce an IP prefix has not been allocated to them. Doing so will result in invalid AS paths and traffic to that prefix being incorrectly routed to the offending AS. When this is done intentionally, it is typically referred to as route/prefix

[1] `https://signal.org/blog/signal-video-calls/`

hijacking. When done by accident, we refer to it as mis-origination. In any case, the long term solution to combat this issue is the use of the Resource Public Key Infrastructure (RPKI). RPKI allows network operators (among other use-cases) to apply for a digital certificate from their RIR, proving that they do in fact have the authority to originate a given prefix. Other network operators can then verify this certificate when filtering incoming routes. While the number and completeness of RPKI deployments is increasing, there are still only around 12 % of ASes that have fully implemented the technology as of 2023 [1]. As an interim solution, one could imagine security-conscious network operators deploying a path prediction engine that can alert them to a set of paths that seem unlikely to be "correct".

3. **Mitigating Tor Traffic Correlation Attacks:** The Tor network is commonly used by Internet users who whish to remain anonymous. An adversary with access to a sufficiently large AS can deanonymize a user if the path chosen by the users Tor client both enters and exits Tor through their AS. This attack is referred to as traffic correlation, which Nithyanand et al. [2] attempt to solve by predicting BGP paths and choosing the entrypoint into the network accordingly. In fact, they partially base their approach on Routing Tree [5] by Gill et al., which we will cover in more depth in Section 3.2.

## 2.3. Valley-Free Routing and its Limitations

Whether an AS path is valley-free is a function of the business relationships the ASes on the path have. Gao [18] defines valley-free thusly: "After traversing a provider-to-customer or peer-to-peer edge, the AS path cannot traverse a customer-to-provider or peer-to-peer edge."

Many paths on the Internet follow this pattern, because it should be financially beneficial for network operators to do so: Typically one wants to route as much incoming traffic as possible over customer ASes (where one is paid for the traffic), followed by ASes with peering agreements (where traffic can flow free of charge) and only then over provider ASes (where on has to pay for the traffic) [20].

There are several potential issues when simulating AS paths based on valley-free routing though:

- Computing whether a path is valley-free relies on a dataset of AS relationships. Inferring these accurately is not trivial and the current state of the art is not perfect. For example, according to Shapira and Shavitt [21], state-of-the-art AS relationship computation only reaches 95 % accuracy. Therefore, one cannot say with absolute certainty whether a given path is valley-free.
- In reality, relationships between ASes can not be neatly put into the three categories of c2p/p2p/p2c. For example, an AS $A$ may have a peering agreement with AS $B$ to freely exchange traffic for some portion of the Internet but still has to pay $B$ for access to other prefixes [20].
- The approach ignores the possibility of configuration errors.

Overall, whether a predicted path is valley-free is still a useful metric to determine its likelihood. It would be unwise, however, to strictly require all predicted paths to be classified as valley-free, since the classification is imperfect and the reality of configuration errors is ignored.

## 2.4. Overview of Datasets

In this section, we will list and given an overview of some datasets that commonly used in BGP research. More specifically, these are also essential for many path prediction systems, including BGPsyche. While we do mention private data in this section, we want to stress now that BGPsyche will be trained only on publicly available data to ensure better reproducibility. Private data will only be used for evaluation and examination.

### 2.4.1. RouteViews and RIPE RIS

A Route Collector is a BGP speaking router that does not route any traffic, but allows any third party to announce their routes to it [22, 23]. The collector then makes its datawbase of paths publicly available for researchers and network operators.

Currently, the most relevant route collectors are operated by RIPE's RIS[2] and the University of Oregon's RouteViews[3] projects. BGPsyche uses paths from both RIS and RouteViews extensively in building the training dataset and in its evaluation. It should be noted that the word collector is often used interchangeably to describe projects/organizations like RIPE RIS or individual routers belonging to them.

### 2.4.2. CAIDA's AS Relationship Dataset

The Center for Applied Internet Data Analysis (CAIDA) in their own words "conducts network research and builds research infrastructure to support large-scale data collection, curation, and data distribution to the scientific research community" [24]. Among countless other contributions, they have published a dataset of AS business relationships (c2p/p2p/p2c, as discussed in Section 2.3), which we shall refer to as "asrel" for short, in referene to the filename [25]. Luckie et al. detail the process of inferring these relationships in [19]. In essence, they take routing data from RouteViews and RIPE RIS, construct a graph of ASes and links and run various domain-specific graph-analysis algorithms. When validating their computed relationships from RPSL ("whois") and BGP community data, they find that they are 99 % accurate. However, Shapira and Shavitt [21] find that this accuracy is far lower (90 %) when using different validation data (from BGProtect), which suggests that there are likely many edge-cases that are not classified properly. In addition, as we pointed out previously, the relationship between ASes is often more complex then what the rigid dataset implies. Nonetheless, the asrel dataset is one of the most easily accessible and

[2] `https://ris.ripe.net`
[3] `https://www.routeviews.org/routeviews`

battle-tested datasets on AS relationships, which is why we choose to follow many other works and use it as one of the core building blocks of BGPsyche.

For perspective, using these relationships and all BGP paths from full tables of RIPE RIS from all collectors at `2023-05-01T00:00Z`, we find that out of a total of 3.5 million paths or 87 % are valley-free, 3 % are not and 10 % contain links that are not accounted for in the dataset and can therefore not be verified.

### 2.4.3. RIPE Atlas, CAIDA Ark, PlanetLab

RIPE Atlas[4] is a service that allows running traceroutes from a set of nodes to arbitrary destinations. It runs on a community oriented incentive structure: Anyone can set up an Atlas node in a network they manage and they will receive credits to run traceroutes on other nodes for the continued operation of their node. Alternatively, one can "purchase" credits by making a donation. All traceroutes run on the Atlas platform are logged and publicly available to download in bulk.

CAIDA Ark[5] is similar enough to Atlas for the purposes of this thesis, except that they are currently restricted to academic researchers and do not require the use of credits[6].

Finally, PlanetLab[7] *was* a again rather similar, albeit more centrally managed. The service was shut down in 2020 [26].

### 2.4.4. Tier 1 Routing Tables

The research in this thesis was aided by a private dataset of Tier 1 routing tables provided by Leitwert GmbH[8], which they acquired by negotiating direct BGP sessions with all major Tier 1 ASes.

In practice, this means that we have access to a dataset of the same shape as RouteViews or RIPE RIS, but it shows the Internet from a different perspective. Rather then seeing all routes voluntarily announced to a collector, we see how a Tier 1 AS would actually reach all other ASes, giving us what is essentially a top-down view of the internet.

## 2.5. Mapping Traceroutes to AS Paths

An AS may contain any arbitrary amount of prefixes. An accurate mapping from ASes to prefixes is also not difficult to compute given enough BGP routing data from services like RIPE RIS or RouteViews. Technically, obtaining the inverse mapping from prefixes to ASes is therefore a

[4] `https://atlas.ripe.net`
[5] `https://www.caida.org/projects/ark/`
[6] `https://www.caida.org/projects/ark/siteinfo/`
[7] `https://planetlab.cs.princeton.edu/`
[8] `https://www.leitwert.net`

trivial computation. In practice however, mapping IP prefixes *as displayed by traceroute* to ASes is complex.

Consider the example given in Figure 2.1: Three BGP speaking routers are peering along the simple path AS1 → AS2 → AS3. In both cases shown, the AS path remains the same. However, in the second case depicted, AS2 and AS3 have chosen to configure their BGP session differently. Specifically, AS2 provides the IP address from its prefix to AS3s router instead of the other way around. This affects the output of a traceroute command run towards the same "target.com" endpoint. When attempting to map the IP level path given by traceroute to an AS path, it is possible to miss an AS entirely that in actuality really is on the path.

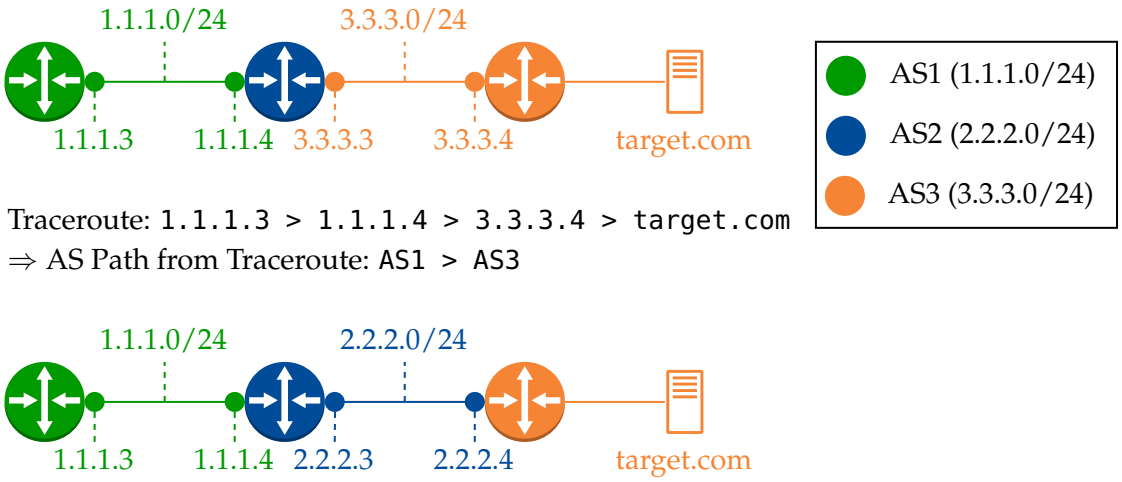



Figure 2.1.: Example of the difficulty of mapping traceroute data to BGP paths: A traceroute run towards the same target will produce different output depending on the way BGP was configured, despite the AS path remaining the same. A mapping of traceroutes to BGP paths can therefore not be done accurately by using a simple mapping of IP prefixes to ASes.

It is beyond the scope of this thesis to deeply consider this problem. In our implementation, we follow the common approach to simply use the "prefix2as" dataset from CAIDA[9] and map each IP to the corresponding ASN. Our intention in writing this section is merely to show that we are aware of the challenges of using such datasets and alert the reader that attempting to infer anything about BGP from IP routing data is imperfect.

[9] https://www.caida.org/catalog/datasets/routeviews-prefix2as/

# 3. Related Work

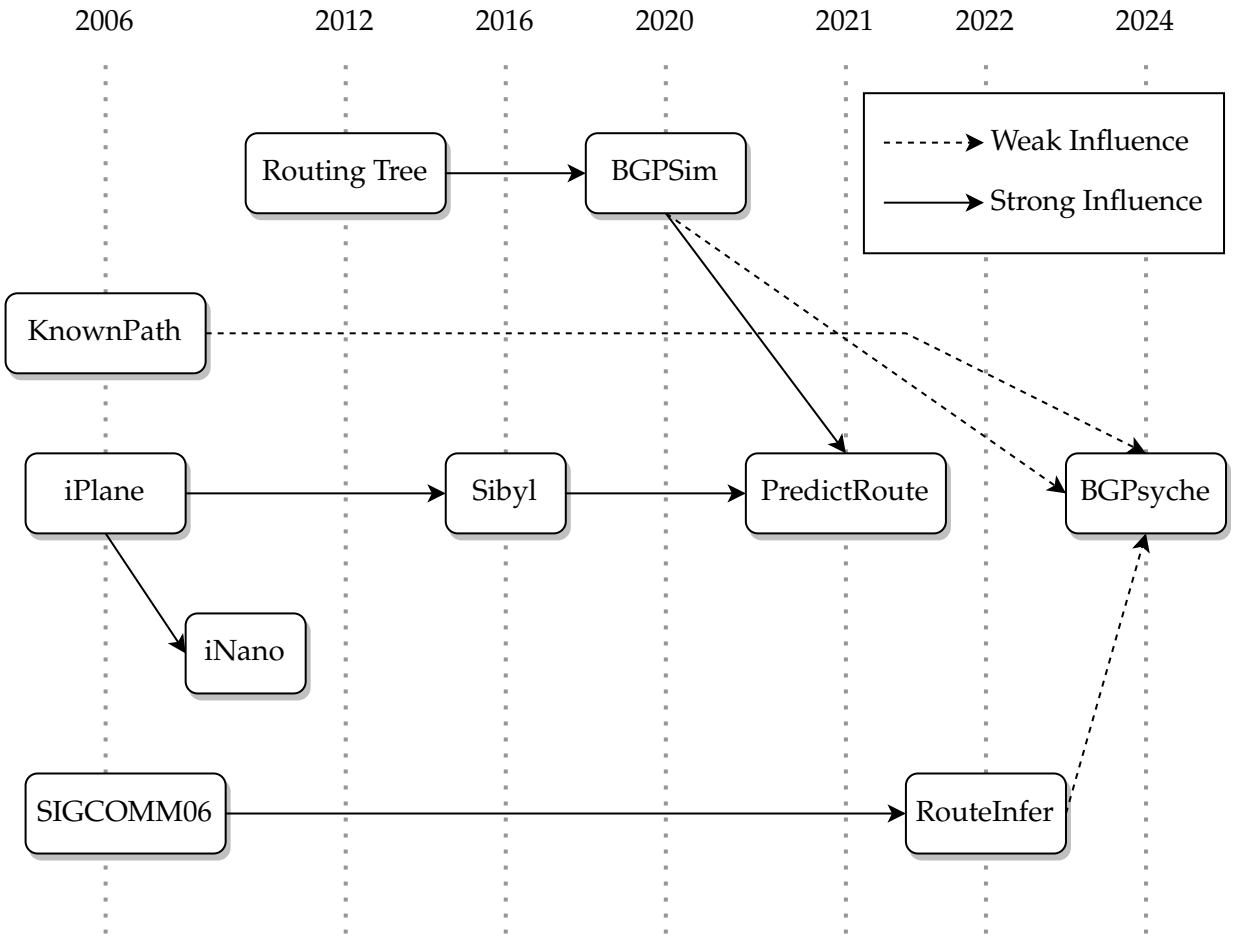



Figure 3.1.: Lineage of AS Path Prediction

This chapter intends to give an overview of the current state of both academic literature and associated software on AS path prediction. To that end, the following sections will elaborate further on the lineage depicted in Figure 3.1. Note that BGPsyche differs from all of these systems in its basic architecture. Even though it builds on many existing datasets and discoveries, it cannot really be understood as a direct successor of any of the works mentioned in this chapter.

It is worth noting here that the only system we were able to run was BGPSim [12]. We will list the practical issues in running the other systems in the following sections.

## 3.1. Shortest Valley-Free

To establish a baseline, we start this chapter by considering the arguably most simple AS path prediction algorithm: To begin, we construct an AS-level graph from real BGP routing data from RIPE RIS and RouteViews, using full tables from all collectors at `2023-05-01T00:00Z`. Then, we run a generic path-finding algorithm, which always finds the shortest paths first (specifically *k-shortest-path* [27]), from the given source to the given destination and finally filter out all paths not considered valley-free according to the publicly available CAIDA asrel dataset. More detailed characteristics of the constructed graph and algorithm performance will be discussed later in Section 4.3.

We find that the correct path is at the top of the list of found paths 20 % of the time. The selected path has an average edit distance of 1.4 to the real path. For comparison, picking one of the shortest paths at random without filtering non valley-free paths yields only slightly worse results with an accuracy of 17 % and an average edit distance of 1.6.

In any case, the poor performance of this baseline shows the need for less naive AS path prediction algorithms. It also highlights that simply filtering for valley-free paths is nowhere near sufficient to achieve a reasonable accuracy.

## 3.2. BGPSim and Routing Tree

While *Routing Tree* [5] was not the earliest of the algorithms discussed in this chapter, it is arguably the simplest. It can be described as a modified breadth-first search which respects the valley-free constraint at every step. While the authors do not provide an implementation directly, BGPSim [12] is, upon close inspection, the same algorithm.

Fundamentally, these algorithms produce a *subset* of available valley-free paths from all source ASes to one given destination AS. The authors of Routing Tree do not give results as to the performance of the algorithm for path prediction, as this was not the focus of the article. When running BGPSim ourselves, we find that it correctly predicts only 16 % of paths, with an average edit distance from the chosen path to the real path of 1.8. Perhaps surprisingly, this is actually worse then our established baseline of searching the shortest valley-free path.

Confusingly, the authors of RouteInfer [11] claim to have measured the accuracy of Routing Tree to be around 24 %. They do not mention if they got access to the original source code or attempted their own re-implementation. This puts Routing Tree above our baseline shortest valley-free algorithm.

The authors of Routing Tree claim to have achieved a runtime of only 10 *ms* for a single iteration. BGPSim on the other hand, needs around 3 *s* in our experience. Most likely, the implementation of Routing Tree was significantly more optimized in some way. Theoretically, some feature of the chosen programming language could also be responsible. Routing Tree was written in C#, while BGPSim is written in Python – although we ran it with the much more performant JIT compiled PyPy.

## 3.3. KnownPath

KnownPath [7] is the earliest algorithm in this overview, written in the year 2006. It is also a modification of an existing, relatively simple path finding algorithm (the "Bellman-Ford" algorithm) and is also limited by producing only valley-free paths, but it has a fundamentally different approach to path finding. KnownPath attempts to construct a path from known path snippets from real routing data, spliced together with artificial valley-free snippets when necessary.

At the time, the authors opted to not share their source code, choosing instead to host KnownPath as a web service. Regrettably, the service is no longer operational[1]. KnownPath is not the most complex algorithm and could likely be re-implemented in a reasonable amount of time.

The authors achieve an average accuracy of 60 %, ranging from 16 % to 95 % depending on the destination AS and training data. Additionally, the authors of RouteInfer [11] claim to have run KnownPath themselves, rating it to have an accuracy of around 59-64 %.

## 3.4. iPlane

In 2006, Madhyastha et al. proposed iPlane [4], a centralized "information plane" service intended primarily to provide developers and network operators with tools to predict path performance for use in distributed (peer to peer) services. In order to estimate performance metrics for a given path, they first had to implement an algorithm to compute possible paths in the first place. Fundamentally, they developed the path prediction algorithm before in [28]. However, since they did not give the algorithm a name, we shall refer to it here as iPlane.

The system starts by first building an atlas of the Internet by running traceroutes on the then ca. 300 vantage points on the PlanetLab service, as well as various looking glass servers. PlanetLab is no longer operational [26], but in principle one could implement iPlane on top of RIPE Atlas or CAIDA Ark. They also go into some detail as to how they select the target destinations of these traceroute measurements and how often they update the atlas.

To predict paths then, iPlane attempts to rely as much on real observed paths as possible. Given a prediction query from AS $A$ to $B$, the system will first check if it ran a traceroute from $A$ to $B$ and, if so, return that path unmodified. Otherwise, it looks for routes from other vantage points towards $B$ that intersect with a path originating from $A$ as close to $A$ as possible. A more detailed visual explanation can be found in Figure 1 in [28]. Additionally, they use latency estimations to minimize the latency from $A$ to the point where the predicted path exits the the first-hop AS, which is intended to model the likely default early-exit (or hot-potato) routing policy.

In [28], the authors claim to find around 83 % of real paths in their atlas and predict around 65 % of them correctly, which we read to result in a final accuracy of $83\,\% \cdot 65\,\% \approx 54\,\%$. Confusingly, they claim in [4] that their approach exactly predicts around 70 % of the paths evaluated.

[1] `http://rio.ecs.umass.edu/cgi-bin/asinfer.cgi`

The source code for iPlane was never made public. However, the website[2] is still operational and even contains some technical documentation, suggesting that it may be possible to still acquire the code for research purposes, refactor the implementation to use RIPE Atlas or CAIDA Ark instead of PlanetLab and test its performance in the modern Internet.

## 3.5. iPlane Nano

Sharing its first name with its older brother iPlane [4], iPlane Nano or *iNano* [3] builds on some of the ideas of iPlane, but is architected to be much more decentralized and scalable. In contrast to iPlane, iNano uses a centralized server only to build an initial atlas and for some coordination, with participating clients providing measurements instead of centrally managed vantage points like those on PlanetLab. This makes the cost of deploying iNano for a centrally managed peer to peer application – such as many Voice-over-IP implementations – much lower.

Despite some similarity in other areas, iNano uses a very different approach to iPlane in path prediction. On the most basic level, iNano uses an algorithm similar in structure to Dijkstra's shortest path algorithm. The authors then extend this algorithm to account for late/early exit policies and to respect the valley-free constraint, resulting in an algorithmic structure similar to BGPSim [12] and Routing Tree [5]. Additionally, iNano uses its training dataset of routes to approximate asymmetric routing and maximize the amount of known AS triplets on the predicted path. Finally, they also model destination-based routing from the training dataset by recalling how often one link was chosen over another for each destination AS.

Madhyastha et al. [3] claim to predict 70 % of paths correctly from a relatively small set of 2800 paths, while the authors of RouteInfer [11] have measured an accuracy of 38-52 %.

Similar to iPlane, there is no publicly available source code, executable, web service or other way to test iPlane Nano. Since it is made by some of the same authors, it may again still be possible for others to reach out to the authors and test the system.

One would still have to determine at what scale to test iNano though, as it requires the participation of at least some amount of client systems to operate. The decentralization of iNano may be an advantage for its primary purpose of determining performance metrics for routes in peer to peer applications, but it also greatly increases the cost of any single organization that wishes to run the system for its own purposes.

## 3.6. Sibyl

Sibyl [8] can in some ways be seen as a successor to iPlane (not to iPlane Nano). However, it has a different focus: Sibyl primarily concerns itself with path predictions, whereas iPlane is primarily intended to predict path performance metrics, for which it has to predict sensible paths first.

[2] `https://web.eecs.umich.edu/~harshavm/iplane/`

It is similar to iPlane in two important ways: It also builds an atlas of the Internet through traceroutes from measurement platforms and even uses the same basic splicing mechanism to predict paths that have not been observed exactly.

On the other hand, it differs from iPlane in two key ways: (1) Sibyl provides the user with a powerful query interface. Instead of "only" being able to query for a set of paths from a source to a destination, Sibyl can predict a set of paths passing through a given set of links, ASes, or countries. (2) Additionally, it uses the supervised machine learning library RuleFit [29, 30] to compute confidence scores for spliced paths. It should be noted that the features used to train this model are highly specific to the splicing algorithm and thus not applicable to the chosen architecture of BGPsyche.

The authors of Sibyl do not provide a simple percentage of how many paths from a given source to a given target were predicted accurately. Instead they compute scores for how well a series of more complex queries were answered. The authors of PredictRoute [10] have reimplemented Sibyl and reached out to the original authors for their training data. They find that Sibyl improves the overall prediction accuracy of iPlane [4] (54 % [28], 68 % [8] or 70 % [4], depending on the source) by some unspecified amount, although it does not improve the edit distance between the real and predicted path (see Figure 12 in [10]).

## 3.7. Mühlbauer et al., SIGCOMM06

Mühlbauer et al. describe another more simulation based approach [6], which, lacking a name from the authors, we shall refer to by the name of the conference in which the paper was published, SIGCOMM06.

They strongly emphasize the need to treat ASes not as single nodes in a graph, but rather as a set of routers, each making different decisions. To account for this, as well as to model policies beyond simple valley-free constraints, they build topology models using *quasi-routers* that can be fed into the *C-BGP*[3] simulator. C-BGP allows for a relatively complete simulation of BGP packets given a topology and policies, terminating when simulated routers converge on a final routing database. Quasi-routers derive their name from not necessarily mapping directly to real routers in a given AS, but nonetheless being required to model routing policies of that AS.

The system is trained iteratively using the following basic loop:

1. Build an initial topology model from RouteViews and RIPE RIS.
2. Run the topology model in the C-BGP simulator.
3. Compare simulated paths to real paths and adjust the topology heuristically to match the real paths more closely by adding quasi-routers or changing policies.
4. Repeat from step 2. until an acceptable ratio of simulated paths are correct.

[3] `https://c-bgp.sourceforge.net/`

The authors claim to achieve 63 % accuracy of exact path matches, as well as an 80 % rate of finding the correct path in the small set of matching paths in the simulated RIB-Out table of the router in question. The authors of RouteInfer [11] on the other hand claim to measure an accuracy of exact matches of around 50-58 %. As there is no source code available for SIGCOMM06 and a re-implementation is out of scope, we will not be able to benchmark the system ourselves.

## 3.8. PredictRoute

PredictRoute [10] builds on BGPSim, RIPE Atlas and CAIDA Ark to predict paths by simulation and active measurement.

It allows path prediction on both the level of ASNs and IP prefixes. Since BGPsyche is only really in competition with the BGP level predictions, this section will not concern itself with prefix level predictions. The following two paragraphs will summarize the general approach to path prediction in PredictRoute:

**Setup:** (1) Bootstrap a BGP topology graph from bulk downloads of traceroutes from Atlas and Ark, as well as simulated routes from BGPSim. More specifically, they build one graph per destination prefix (since BGP routing is fundamentally destination-based) and compute probabilities for each link between ASes in the graph, based on how often the link was seen in the ingested traceroutes. (2) Compute the optimal set of Atlas/Ark nodes to run traceroutes from to enhance the existing per destination graphs / markov chains for a given budget. Notably, this means that there is a cost associated with the operation of PredictRoute.

**Query Response:** Given a user query, use the constructed BGP topology markov chains to predict as much of the desired path as possible. When the source ASN is not in the computed markov chain for the destination AS, BGPSim is used to simulate paths from the source AS towards the destination AS. PredictRoute then picks the simulated path that leads into an ASN contained in the destinations markov chain the quickest, at which point it uses said chain to predict the rest of the path.

The authors do not provide the accuracy of predicted paths as a simple percentage of correctly predicted paths. The figure they use to describe PredictRoute most commonly is that the predicted path differs from the real path by at most one hop, 75 % of the time. It also compares favorably to BGPSim, Sibyl [8] and iPlane Nano [4]. Again, the authors of RouteInfer have benchmarked PredictRoute and claim an accuracy of around 45-60 %.

The source code for a prototype implementation of PredictRoute is publicly available[4]. We were not able to run this code to benchmark the system ourselves, as the prototype depends on the presence of several files and services, a lot of which are not documented.

[4] `https://github.com/racheesingh/predictroute-code/`

```
// destination prefix policy:
// ------------------------------------------------------------------------
aut-num: AS4670
import: from AS3561 action pref=100 accept {204.70.0.0/16}

// destination AS policy:
// ------------------------------------------------------------------------
aut-num: AS18060
import: from AS4739 action pref=10 accept AS473

// neighbour policy:
// ------------------------------------------------------------------------
aut-num: AS17843
import: from AS3786 action pref=10 accept ANY
```

Listing 3.1: RouteInfer 3-Layer Policy Examples

## 3.9. RouteInfer

RouteInfer [11] is fundamentally a very similar system to SIGCOMM06. It conceptually extends SIGCOMM06 with two major contributions:

First, SIGCOMM06 only models prefix policies. However, real policies can also refer to AS or neighbor policies. To illustrate, Listing 3.1 shows the same example that the authors of RouteInfer used (verbatim, in the RPSL format typically returned by the `whois` command line tool). The modeling of these additional types of policies allows RouteInfer to simulate routing behavior more accurately by not being overly specific and allowing more general policies.

Second, for those ASes where RouteInfer cannot infer any policies, RouteInfer uses a Learning To Rank (LTR) machine learning model to choose between available paths instead of relying on the valley-free constraint or simply picking the shortest path. This is in principle not so different from what BGPsyche aims to do, except RouteInfer uses machine learning to pick from only a handful of simulated routes, while BGPsyche makes this the central mechanism for choosing a route from all available candidate routes. One could even imagine fusing RouteInfer with BGPsyche, such that BGPsyche is used in place of the LTR model.

The authors claim to reach an impressive *82 % accuracy* of exactly matching paths. This presents RouteInfer as the *state-of-the-art solution* for path prediction.

The source code for RouteInfer is in theory publicly available[5]. However, the repository is clearly missing some parts of the implementation. There are undefined variables[6] and the LTR library – a custom LambdaMART [31] implementation – is not actually imported or used anywhere.

[5] `https://github.com/DiceWu/RouteInfer`
[6] `https://github.com/DiceWu/RouteInfer/issues/1`

# 4. Architecture and Features

We design and implement a novel BGP path prediction system, BGPsyche, both to explore the potential of an approach based almost purely on deep learning and to hopefully challenge the status quo. This chapter covers architecture, algorithmic path discovery and feature engineering. An evaluation of the model based on general and domain-specific metrics, feature selection and hyperparameter optimization will follow in the next chapter.

## 4.1. Architecture Overview

BGPsyche, as shown in Figure 4.1, can be understood to operate in three stages:

1. *Discover* (Section 4.3): Acquire a list of candidate paths between the given source and destination ASNs. These may be computed by various algorithms or sampled from real paths from e.g. RouteViews.
2. *Enrich* (Section 4.4): Here BGPsyche enriches the previously acquired list of candidate paths with metadata from various sources. These may include attributes from existing databases or values BGPsyche computes itself.
3. *Rank* (Section 4.6): Finally, BGPsyche ranks the candidate paths based on determined metadata. The ranking is done through a custom neural network.

Before operation, it is necessary to train the path ranking neural network, using a dataset created from the same metadata sources and a set of real AS paths.

## 4.2. Training Paths and Date

To ensure the data we use is coherent and fits together, all datasets we mention in further sections are dated as close as possible to the 1st of May, 2023 at midnight, UTC (`2023-05-01T00:00Z`).

We will often mention using paths from RIPE RIS and RouteViews. If not explicitly stated otherwise, we mean all full table files (no update files) from all collectors at exactly `2023-05-01T00:00Z`.

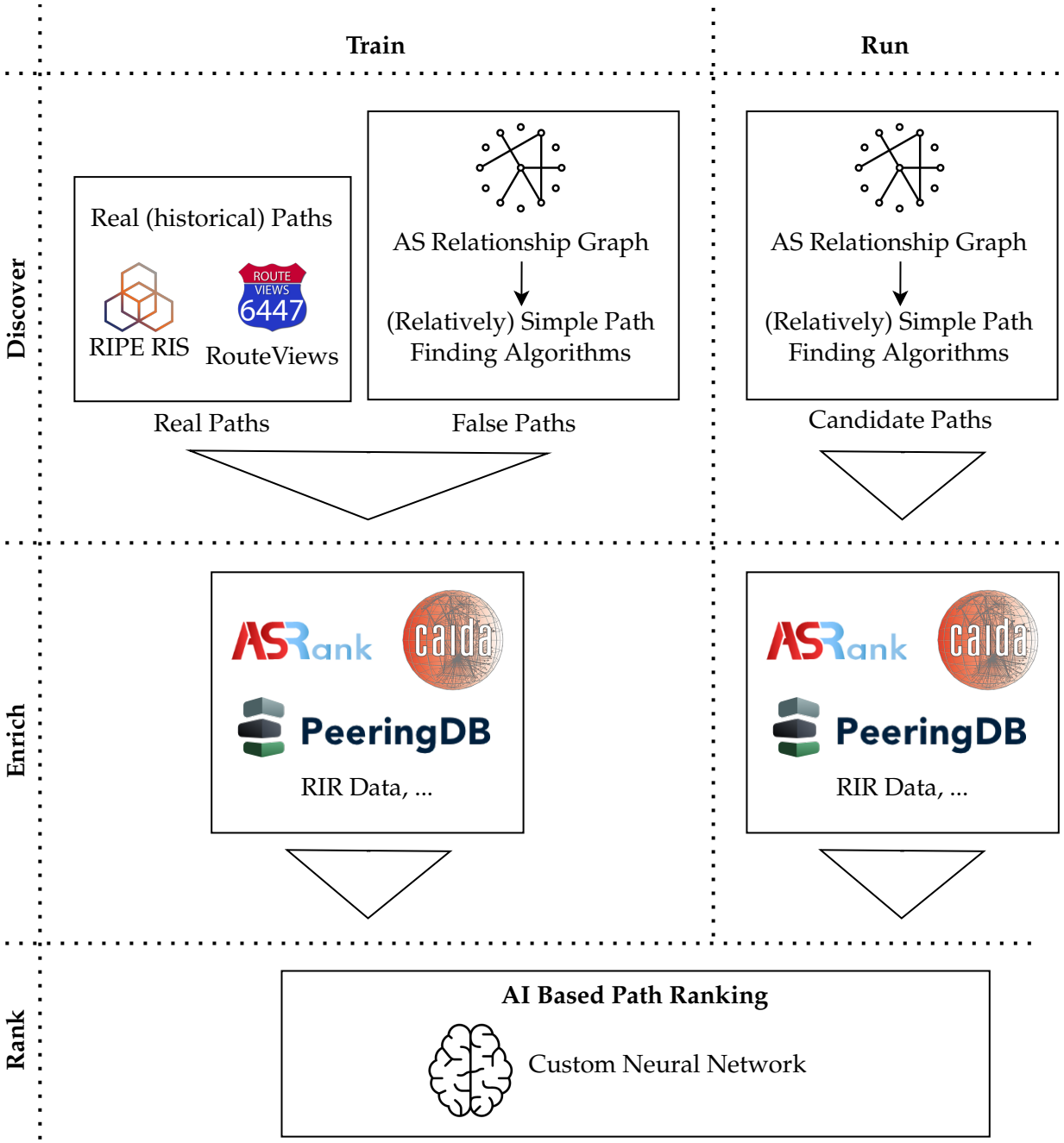



Figure 4.1.: BGPsyche Architecture

## 4.3. Path Discovery

In this first stage of BGPsyche, we compute a set of candidate paths to be enriched and ranked later. We use both off the shelf and custom graph search algorithms for this purpose. Other types of algorithms, such as a splicing algorithm that combines existing path snippets into candidate paths are conceivable, but we deem them not necessary. Our intention is to let the Machine Learning (ML) model judge the probability of a path. Therefore we do not attempt to encode too much domain knowledge into the path search algorithms; We only ensure that they return a satisfactory set of paths that should hopefully include the real one.

Given a list of real paths from sources like Route Collectors, it is not difficult to build a graph

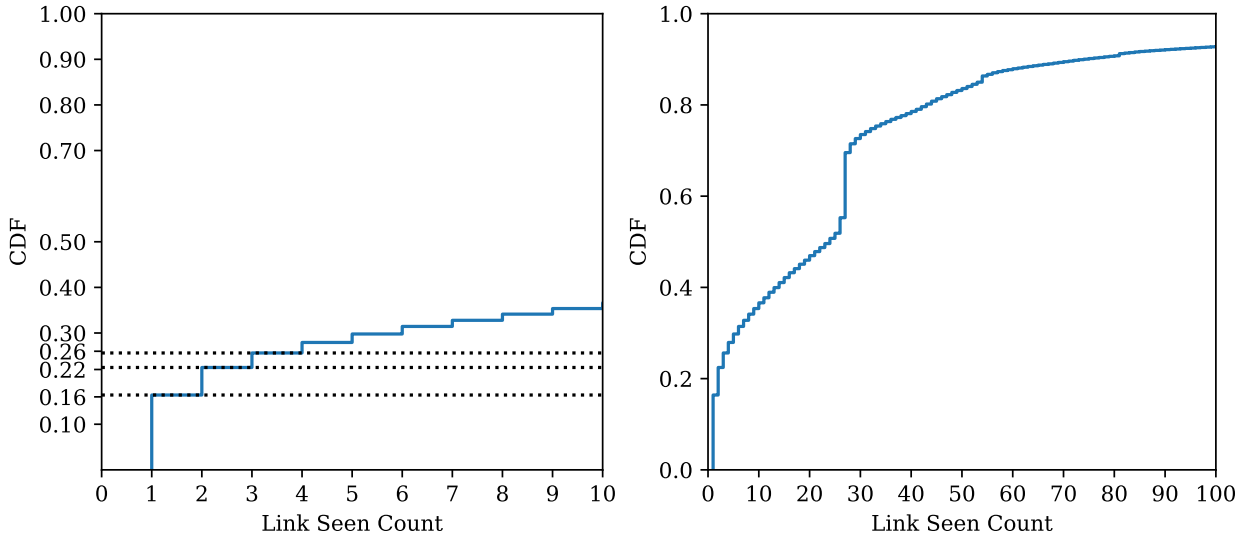
representing the Internet from the perspective of BGP. Whether we can find the correct candidate depends on the completeness of this graph and the time/space complexity of the chosen path search algorithms and thus this question needs to be answered by experiment. This section therefore first establishes the chosen methods to build the graph and algorithms for path searching before evaluating them.

### 4.3.1. Building the Graph

To save time and enhance computational efficiency, we take the arguably most straightforward approach to building a BGP graph: We use all paths from RouteViews and RIPE RIS as outlined in the previous section. It may be possible to slightly improve results by using data collected over a longer time frame or using additional sources like RIPE Atlas. This certainly may be a reasonable thing to do when running BGPsyche in production. However, as the coming sections will show, we will find a satisfactory percentage of real paths using this limited dataset. We take this and the much more reasonable computational requirements as justification to not dwell on optimizing this aspect any longer.

In addition, we have run some experiments that suggest the resulting graph is complete enough for the intended use case:

- When building a graph, we find that it contains 98 % of paths from all Tier 1 RIBS (see Section 2.4.4) from the same day. Therefore, the usage of the private Tier 1 dataset would not add many links to the resulting graph. To make our research more reproducible, we therefore choose to only use the publicly available data from RouteViews and RIPE RIS, although we will use Tier 1 routing data for the evaluation.
- The graph contains "only" 15 % of links which are only found once in the input paths (as shown in Figure 4.2). This is perhaps not ideal, but it does show that a strong majority of links are visible multiple times.




(a) Start of Distribution (Zoomed)

(b) Full Distribution

Figure 4.2.: Link Seen Counter in BGP Graph Input Paths

The constructed graph contains 82,000 ASes with 250,000 links. This number may be of interest to some, but it can not easily be used as an indicator of completeness. While it is easy to query RIR data and conclude that around 110,000 ASNs have been assigned [32], not all of these must necessarily be in active use.

### 4.3.2. Generic Shortest Path Search

To bootstrap an initial set of candidate paths, BGPsyche uses a simple breadth first style path search algorithm. To save time and not worry about optimization, we choose to use the `shortest_simple_paths` function from the `networkx` Python library[1]. This algorithm is more specifically based on the *k-shortest-paths* algorithm by Yen [27], which grants us the guarantee of finding the shortest paths first. This is useful, since BGP paths are generally more likely to be as short as possible.

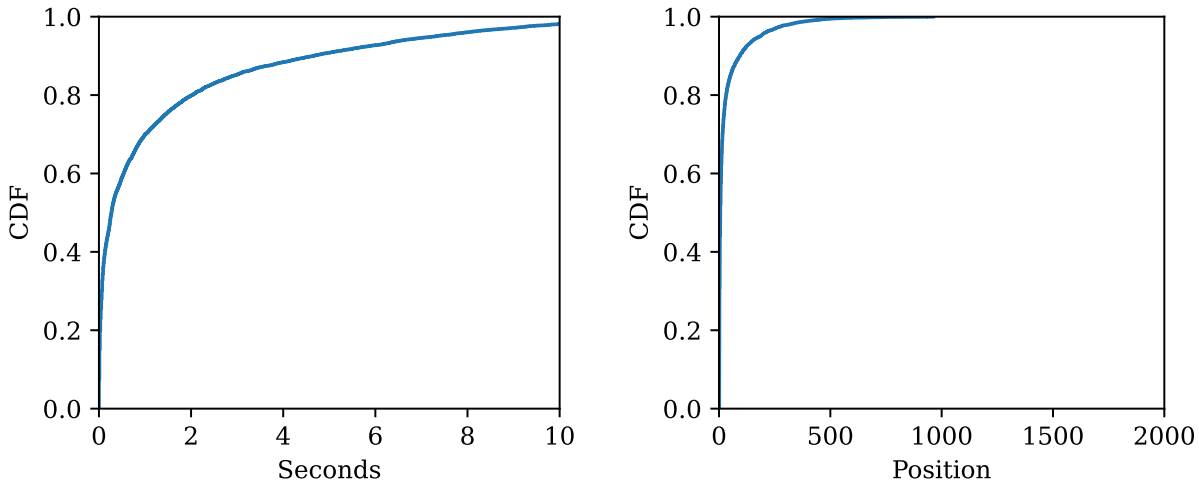
Running `shortest_simple_paths` on the BGP graph is not sufficient to find a satisfying amount of real paths. Perhaps unsurprisingly, given the graph size of 80,000 ASes, the algorithm quickly runs into time complexity issues and will typically cease to find any additional paths after some amount of time. If the real path is not in that initial set of shortest paths, it will not be found.

To make reasonable use of this algorithm then, we need to set a timeout. To determine this timeout, we run the algorithm and, each time it does find the real path, take note of how long it took to do so. This requires us to wait for many seconds for each processed path, making this a rather time intensive (although parallelizable) task. The experiment is therefore limited to a somewhat arbitrary 10,000 random paths from RIPE RIS.

[1] `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.simple_paths.shortest_simple_paths.html`

The results indicate that `shortest_simple_paths` is able to find 71 % of real paths in the given BGP graph. Of those found real paths, 97 % were found after 10 seconds of searching, as shown in Figure 4.3a. This leads us to initially configure the algorithm for a 10 second timeout.

Additionally, it seems unwise to needlessly confuse later stages of the BGPsyche pipeline with paths that are very likely incorrect anyway. To account for this, we also plot the position of found real paths in the returned set of candidates in Figure 4.3b. The plot shows that there close to zero real paths are found roughly after the first 750 candidates were returned. BGPsyche is therefore configured to terminate the algorithm after 800 found candidate paths.




(a) Time Until Real Path Found

(b) Position of Real Path

Figure 4.3.: Generic Shortest Path Search Characteristics

Lastly, we check our intuition that the algorithm primarily fails to find *longer* paths by plotting the distribution of the lengths of those paths that were not found in Figure 4.4. Most paths that were not found seem to be 5 ASes long, which would seem to confirm the suspicion.

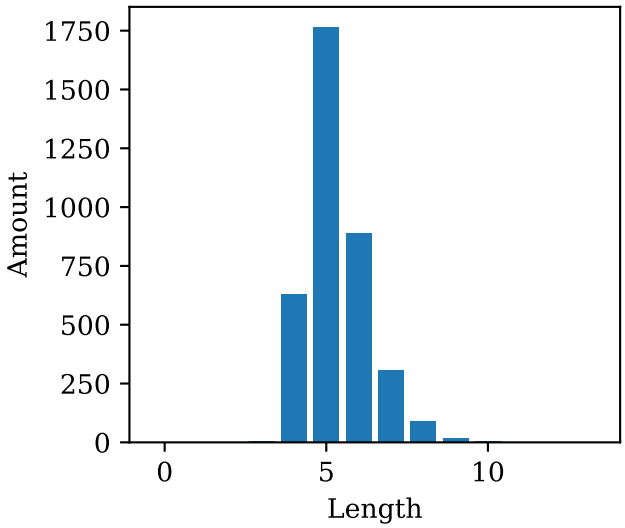



Figure 4.4.: Length of Paths not Found by Generic Shortest Path Search

### 4.3.3. ConeSearch

Since attempting to list all possible paths is clearly computationally too expensive, BGPsyche uses an additional algorithm to find path candidates. This algorithm attempts to heuristically limit itself to "reasonable looking" BGP paths, thereby finding less paths overall but doing so a lot more quickly, especially for longer paths. We shall refer to the algorithm as *ConeSearch*.

Given a sufficiently large list of routes and a dataset of AS relations as discussed in Section 2.4.2, it is possible to write an algorithm that will produce a mapping from every ASN to a list of all transitive upstream ASNs (as in Figure 4.5).

ConeSearch then constructs a sub-graph of the entire BGP graph, which only includes the source and destination ASes of the search query, as well as all their respective transitive upstream ASes. Finding paths in this much smaller graph can then be done using any conventional path finding algorithm. In our implementation, we again use `shortest_simple_paths` from `networkx`, but given the limited size of the sub-graph it really should be possible for any path finding algorithm to successfully find all possible paths within a negligible amount of time.

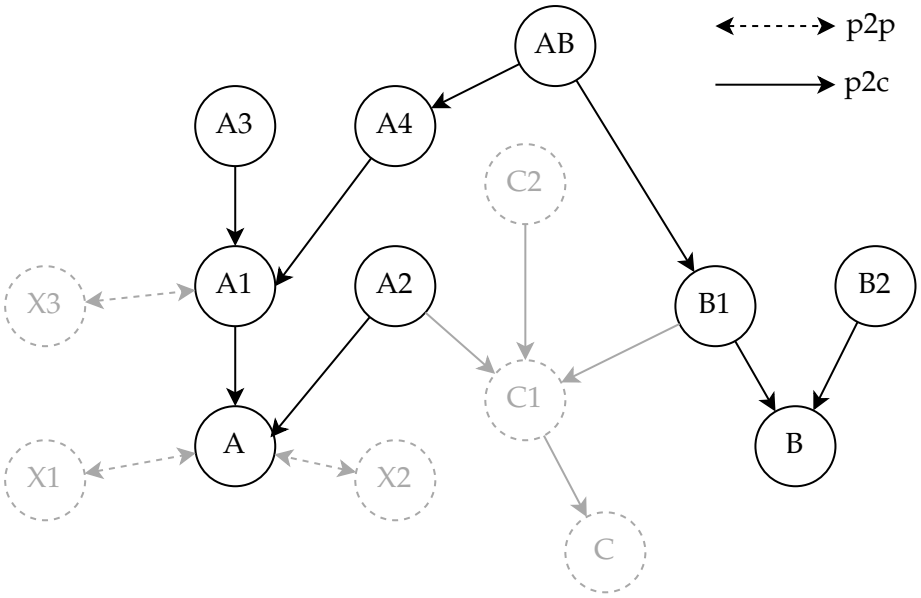
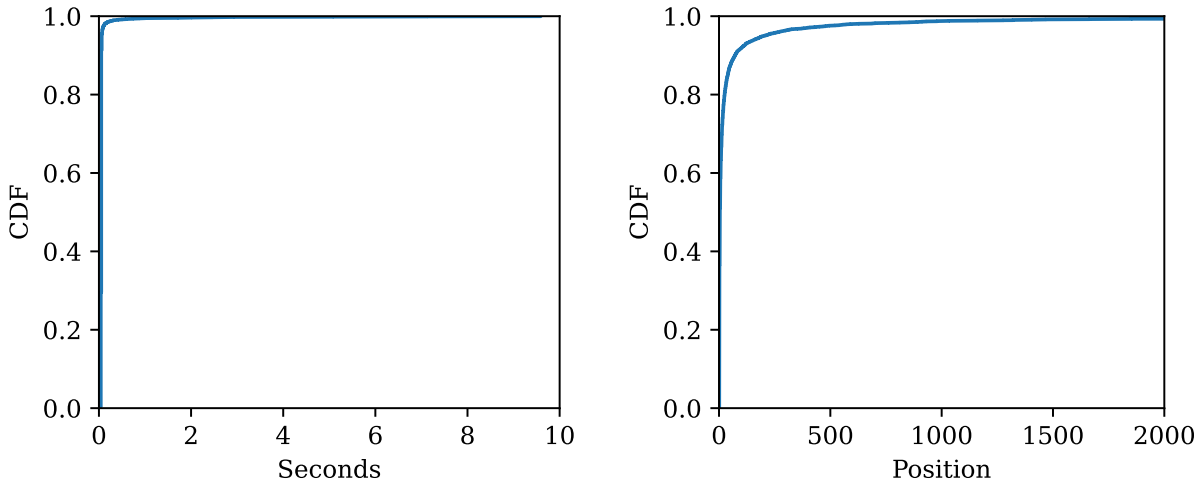



Figure 4.5.: ConeSearch first produces a mapping of every ASN to a list of all transitive upstream ASNs. For example, given the topology in this Figure, it would produce the mapping $\{A \to [A1, A2, A3, A4, AB], B \to [B1, B2, AB], C \to [C1, A2, B1, AB, C2]\}$. For a path search query $A \to B$, ConeSearch then only considers the ASes are in the mapping for $A$ and $B$, highlighted in the Figure by not being grayed out.

Running the same experiment as with the generic path search from the prior section, we find that ConeSearch finds 81 % of real paths in our BGP graph within the first 1,000 candidates and usually takes at most one seconds to do so (see Figure 4.6). BGPsyche therefore terminates ConeSearch after one second and after 1,000 found candidates were returned.




(a) Time Until Real Path Found

(b) Position of Real Path

Figure 4.6.: ConeSearch Characteristics

### 4.3.4. Combining the Algorithms

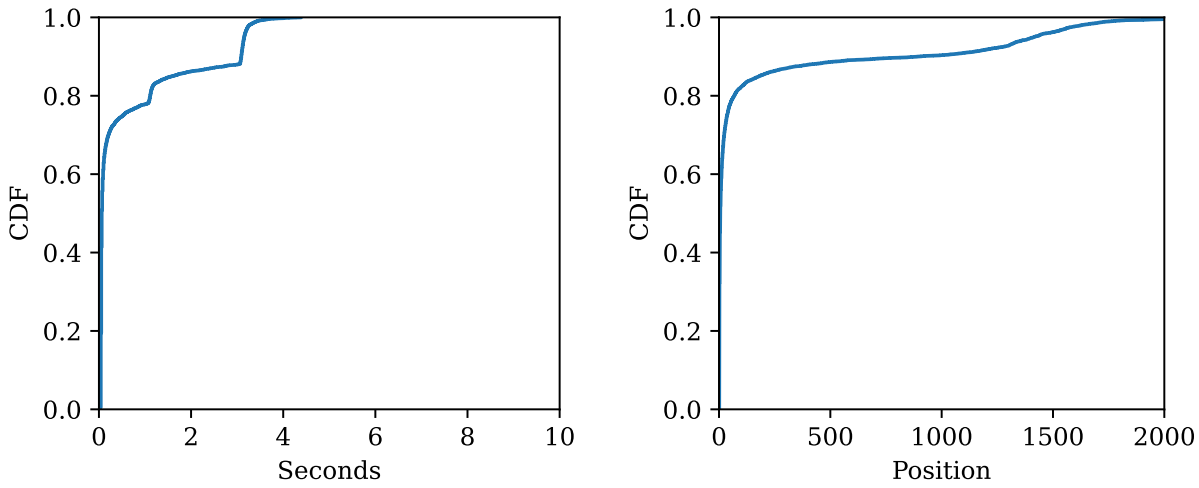
BGPsyche uses both the generic path search and ConeSearch together to find path candidates. Running the same experiment of attempting to find 10.000 RIPE RIS paths again, we are able to find 95 % of the input paths.

Before committing to the algorithm, we make one last attempt at optimization: We set the timeout for the generic path search from 10 $s$ to 3 $s$. This is not merely an attempt to make running and training BGPsyche easier, but follows an intuition about the behavior of the two algorithms. In general, we except ConeSearch to be able to return longer paths more easily since it attempts to limit itself to "reasonable looking" BGP paths. It stands to reason then, that leaving the generic path search running for a long time might be inefficient, because it would hopelessly attempt to compete with ConeSearch in the area it is specifically designed to excel in. Indeed, we confirm this intuition by running the same experiment again and finding 94 % instead of 95 % of paths. We deem this to be a reasonable tradeoff and set the timeout for the generic path search to 3 $s$. The final configuration and use of search algorithms is summarized in Table 4.1.

| Algorithm | Timeout | N Results Cutoff |
|---|---|---|
| Generic Path Search (Section 4.3.2) | 3 $s$ | 800 |
| ConeSearch (Section 4.3.3) | 1 $s$ | 1000 |

Table 4.1.: BGPsyche uses multiple algorithms to discover BGP paths in a given AS graph. This table lists the algorithms used and their main parameters.

Running the final algorithm, we also compute the same plots for search time and candidate positions in Figure 4.7b. These are perhaps no longer essential, as they have no influence on the further setting of some configuration variables, but they do display the behavior of the algorithm, which may be of interest to some readers. Note that the algorithms are run in sequence: We find all paths after four seconds because $1$ *(ConeSearch)* $+ 3$ *(Generic Path Search)* $= 4$.




(a) Time Until Real Path Found

(b) Position of Real Path

Figure 4.7.: BGPsyche Path Search Characteristics

## 4.4. Acquiring Path Metadata

In order for the third stage of BGPsyche to rank path candidates, we have to first acquire metadata about these paths. We justify our choice in potential *features* (or attributes) used to train BGPsyche by both statistical methods as well as a discussion of technical intuitions about AS-paths. In the language of machine learning, this section is dedicated to *feature engineering*. The analysis of the measurable impact on model performance of each feature will follow later. In this section, we "only" seek to explore features individually.

BGPsyche is trained on three different types of features: Those relating to ASes, those that are attributes of links between ASes and finally those which concern an entire AS path. This section is divided into subsections according to these types.

### 4.4.1. Correlation

In order to approximate the utility of features we feed to BGPsyche, we will at various points calculate correlations between them. How these are computed and how to interpret them is not trivial without some prior knowledge of statistics, so we give an overview here.

In general, one must first determine the "type" of a feature before proceeding to calcucate a correlation, that is, if the feature in question is *numerical* or *categorical*. Table 4.2 lists commonly used correlation algorithms for different types of data, along with available implementations in Python.

| Feature Types | Algorithm | Python Implementation |
|---|---|---|
| numerical - numerical | Pearson's $r$ | `statistics.correlation`[2] |
| numerical - binary | Pearson's $r$ | `statistics.correlation` |
| binary - binary | Pearson's $r$ | `statistics.correlation` |
| categorical - categorical | Cramer's $V$ | `scipy.stats.contingency.association`[3] |
| categorical - numerical | $\eta$ ('Eta') | `dython.nominal.correlation_ratio`[4] |

Table 4.2.: Overview of commonly used correlation algorithms for various types of data

The additional *binary* type refers to a categorical feature that was split into several binary values. For example a categorical feature with the values $A$, $B$ and $C$ may be split into three separate features $A$, $B$ and $C$, each set to zero or one depending on the original value. This means that we can in practice use Pearson's $r$ everytime we compute a correlation.

How to interpret different values of Pearsons's $r$ depends on the context. For us, we interpret anything less then a 0.8 correlation between two features to mean that they are sufficiently different for our ML model to potentially benefit from the inclusion of both.

### 4.4.2. AS Features

We start by discussing features related to individual autonomous systems. Each feature will be explained and the data explored and visualized to justify its potential inclusion in BGPsyche. The features discussed are:

- Customer Cone and ASRank
- AS Category
- Democracy Index
- Amount of Registered IP Addresses
- Date of Registration
- Geographic Distance to Source AS

After reviewing these features, possible correlations between features are plotted in a correlation matrix. Overall, we find that all features show the potential to genuinely improve model training.

[2] `https://docs.python.org/3/library/statistics.html#statistics.correlation`

[3] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.contingency.association.html`

[4] `https://shakedzy.xyz/dython/modules/nominal/#correlation_ratio`

### Customer Cone and ASRank

The customer cone of an AS $A$ is defined to be the set or number of ASes which $A$ can reach using customer links [19], which in turn are given by some dataset of AS relationships (see Section 2.4.2).

BGPsyche uses customer cone data from CAIDAs ASRank [33]. The only modification we make is to scale it logarithmically, since otherwise it would be a highly uneven distribution, with only very few ASes having massive cones and most having only a small amount of customers. The distribution is plotted in Figure 4.8.

It should be noted that BPGsyche deliberately does not use the ASRank directly, because an increase of the ASRank by one can translate to vastly varying jumps in customer cone size.

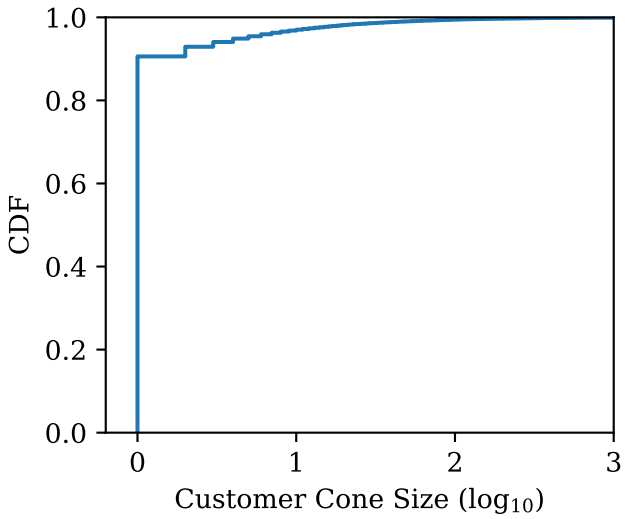



Figure 4.8.: Distribution of ASRank Customer Cones (Logarithmic Scale)

### AS Category

There are many ways in which one could categorize ASes. What we mean to discuss in this section is a categorization that hopefully encodes information that may be relevant to BGP routing decisions.

BGPsyche uses the same categories as CAIDA's AS classification dataset [34], which are almost the same as PeeringDB[5], except that the "NSP" and "Cable/DSL/ISP" are merged into a single "Transit/Access" category, with the intuition that these do not meaningfully differ for routing decisions. The full list of categories is therefore: "Unknown", "Transit/Access", "Content", "Enterprise", "Educational/Research", "Non-Profit", "Route Server", "Network Services", "Route Collector" and "Government".

[5] `https://www.peeringdb.com`

```
asn2cat: dict[int, str] = {}

class ASdbEl(TypedDict):
    asn: int
    layer1_categories: str[]
    layer2_categories: str[]

for asdb_el in load_asdb():
    if 'Computer and Information Technology' in layer1_categories:
        idx = asdb_el['layer1_categories'].index('Computer and Information Technology')
        asn2cat[asdb_el.asn] = 'Computer and IT - ' + asdb_el['layer2_categories'][idx]
    else:
        asn2cat[asdb_el.asn] = asdb_el.layer1_categories[0]
```

Listing 4.1: Flattening ASdb Categories

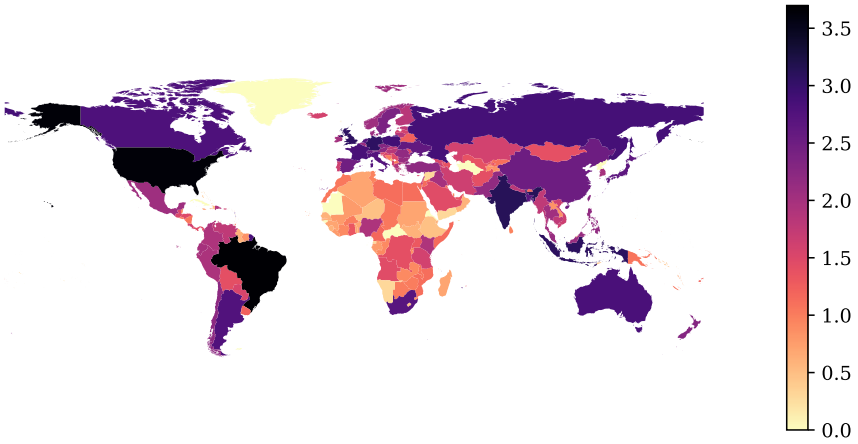
The reason to choose the categories is again similar to CAIDA [34]: BGPsyche uses PeeringDB as ground truth data for AS categories. PeeringDB is a public database where network operators can voluntarily disclose information about their ASes, usually with the goal of "accelerat[ing] the process of finding and connecting with other networks, while supporting a faster and more decisive deployment of [...] network expansion and development plans" [35].

According to their website [35], PeeringDB has information on 30.000, or around a third of all ASes. While impressive, this still means that we want to extend this dataset by other means. Therefore, BGPsyche also considers the categories of "ASdb" [36]. To do so, we must produce a mapping from ASdb to BGPsyche categories. ASdb is significantly more granular then PeeringDB in its categorization: It assigns multiple two-layer categories to each AS. For example, Amazon (AS16509) is classified as "Computer and Information Technology (Hosting and Cloud Provider)" as well as "Retail Stores, Wholesale, and E-commerce Sites", where "Hosting and Cloud Provider" is a subcategory (layer two) of "Computer and Information Technology". BGPsyche maps these multiple layered categories to one single-layer category: When any layer one category of the given AS is "Computer and Information Technology", it will use the layer two category. In any other case, we pick the first layer one category. This is also shown in Listing 4.1 for clarity. Finally, we statically map the left over ASdb categories to BGPsyche categories. The mapping in question is available in Appendix A.1.

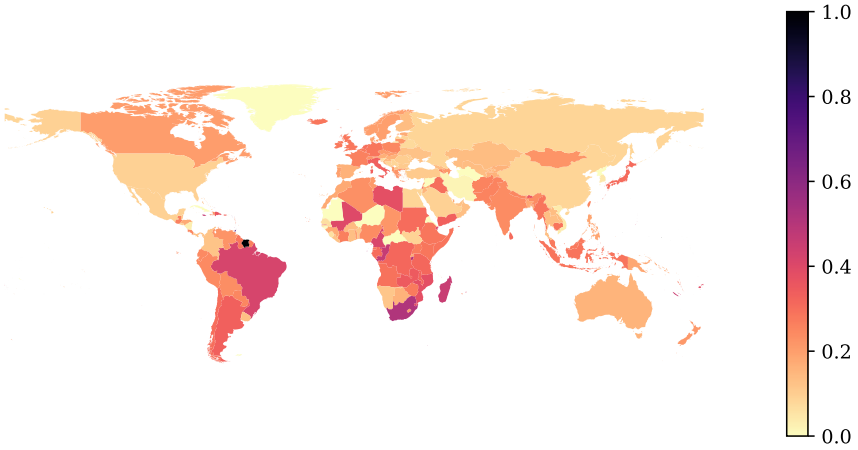
The primary reason for extending PeeringDB categories using ASdb was to have more ASes categorized. Using our mapping (Appendix A.1) and discarding any "Unknown" categories, ASdb covers 96,625 ASes of all 117,138 currently assigned ASNs. Extending the 20,878 ASes covered by PeeringDB yields a total of 98,361 ASes that we can categorize.

We were initially hesitant about using PeeringDB categories, as we suspected that this could bias BGPsyche to prefer routes in western countries, which we thought would contribute to PeeringDB more often. However, this is not actually the case, as shown in Figure 4.9. While there really are many more ASes from western countries participating in PeeringDB *absolutely*, they do not contribute more *proportionally* to the amount of ASes that exist in their countries overall. In fact, western ASes contribute to PeeringDB *less* often on average.

This data was computed by first getting a list of all ASes according to the RIR delegation files[6] as well as all ASes in PeeringDB using their SDK[7] and then mapping all ASNs to countries using the `asn.txt`[8] from RIPE. From there we simply take the absolute amount of ASes in PeeringDB per country or the fraction of ASes in PeeringDB in that country and plot them on a map. The absolute amount of ASes was scaled logarithmically.




(a) Absolute Amount of ASes participating in PeeringDB (Logarithmic Scale)




(b) Percentage of ASes participating in PeeringDB

Figure 4.9.: Distribution of ASes Participating in PeeringDB

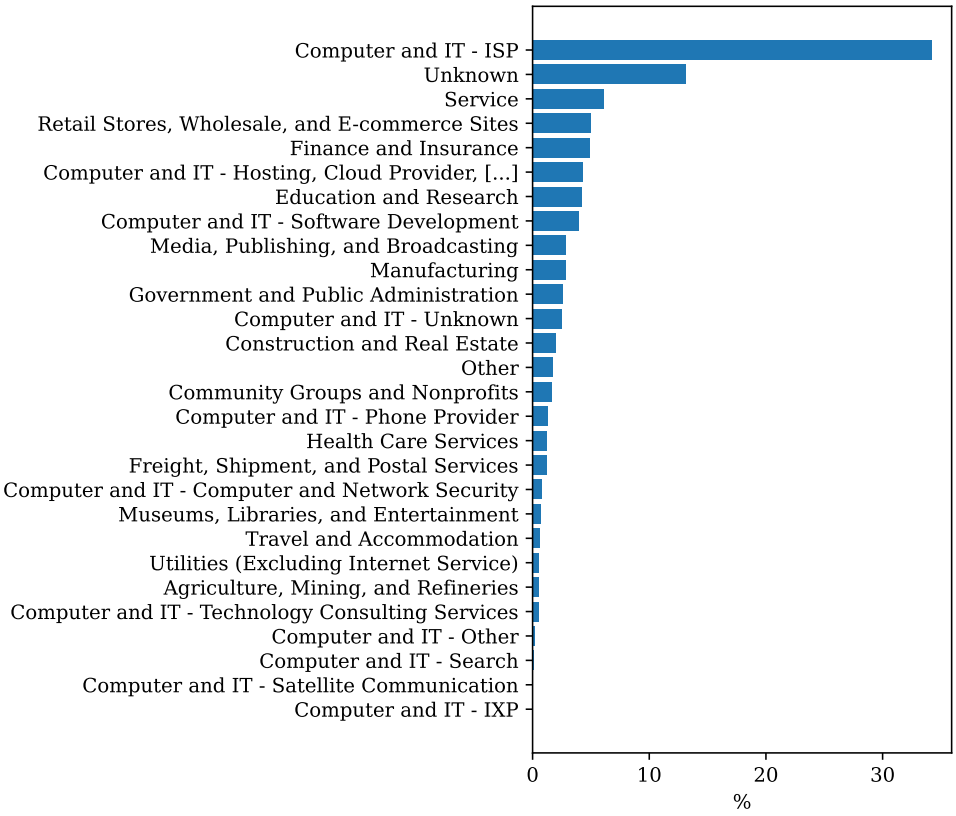
Finally, we provide plots for the distribution of categories in all ASNs allocated by RIRs in Figure 4.10. We make the following observations and adjustments:

- The "Route Collector" category is so uncommon that it is omitted when producing the training data set. This should hopefully make learning slightly easier without sacrificing

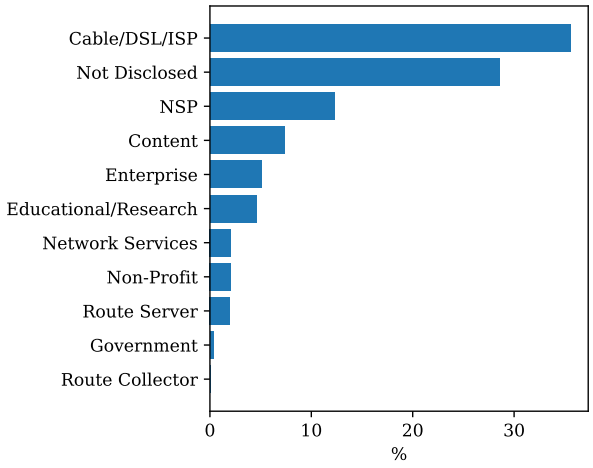
[6] `https://ftp.ripe.net/pub/stats/`
[7] `https://docs.peeringdb.com/howto/peeringdb-py/`
[8] `https://ftp.ripe.net/ripe/asnames/asn.txt`

accuracy.

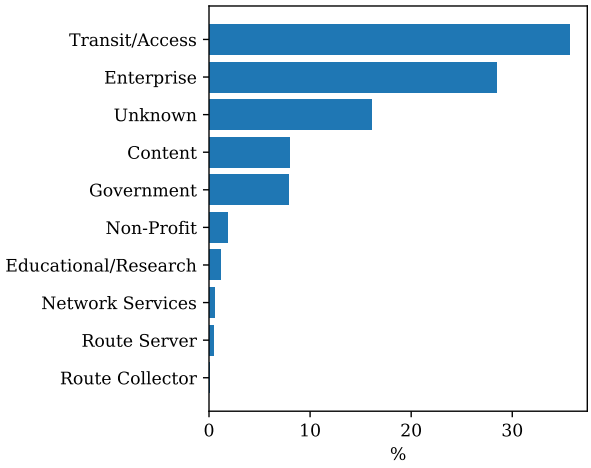
- All three datasets agree that the most common category are Internet Service Providers.
- Route Servers (labeled as IXPs in ASdb) are significantly more common in the PeeringDB dataset. This makes sense when considering that the purpose of PeeringDB is primarily to facilitate peering decisions, which would make an IXP more likely to participate in PeeringDB then the average AS.
- One could make a similar argument for the "Educational/Research" category. In PeeringDB, it is again overrepresented, perhaps because more commercially oriented organizations are on average less interested in contributing data to a public database.
- There are slightly less ASes classified as "Unknown" in ASdb compared to the combined BGPsyche dataset. This is because BGPsyche additionally maps the “Other” category of ASdb to its “Unknown” category.




(a) AS Categories in ASdb




(b) AS Categories in PeeringDB




(c) AS Categories in BGPsyche

Figure 4.10.: Distribution of AS Categories in all ASNs allocated by RIRs

## Democracy Index

AS paths should intuitively be affected by political alliance and tension between countries. For example, one would expect not many AS paths to cross the border between North and South Korea. Numerically modeling political tensions between countries is not our expertise and so we choose to use a "Democracy Index" as an approximation. This would at least properly model the example case given between North and South Korea.

Of course, getting the democracy index for the country of an AS first requires a mapping from ASNs to countries. BGPsyche uses the "asn.txt" dataset from RIPE [37] for this task. It should be noted that a mapping from ASN to countries is necessarily flawed, as an AS may operate any number of IP prefixes in different countries. However, we can accept some flaws in our dataset as long as BGPsyche can learn general trends.

BGPsyche specifically uses a dataset downloaded from "Our World in Data", which gets its data from the "Economist Intelligence Unit" [38, 39]. The index is relatively evenly distributed, as shown in Figure 4.11, so we include it directly as a feature without any further transformation.

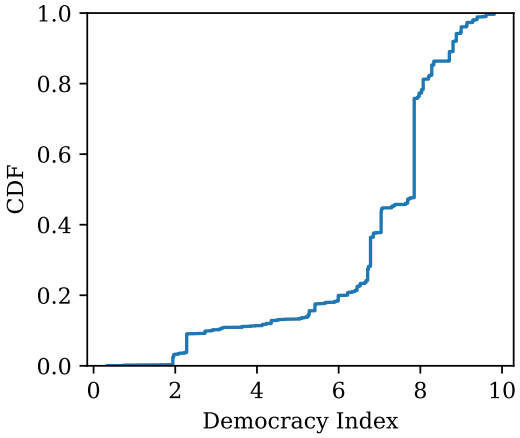
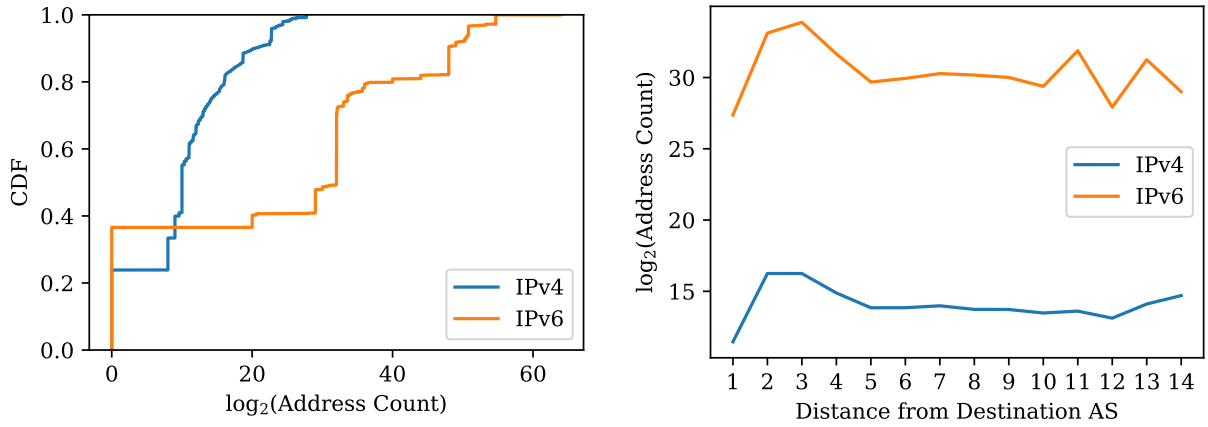



Figure 4.11.: Distribution of Democracy Index over all registered ASNs

### Amount of Registered IP Addresses

As discussed in Section 2.1, the Regional Internet Registries are responsible for both delegating ASNs and IP address ranges to these ASes. They publish this data in a mostly standardized plain text format [40]. BGPsyche parses these files to (among other use cases) get the amount of allocated IP addresses as a feature for its learning process.

Figure 4.12a shows the distribution of the amount of IP addresses over all allocated ASNs. BGPsyche scales this number logarithmically to get a more even and thus easier to learn distribution.




(a) Number of IP Addresses of all allocated ASNs (b) Mean number of IP Addresses of each AS on a path

Figure 4.12.: Distribution of the Number of IP Addresses Allocated to ASes

Additionally, Figure 4.12b shows the mean amount of IP addresses of ASes on real AS paths by the ASes distance from the destination on the path. The values are computed using paths from RIPE RIS. We plot these values to confirm our intuition that there is a tendency for AS paths to follow a learnable "slope" of address counts. For IPv6, the plot may seem initially to go against this intuition. On closer inspection however, it is possible to identify a likely learnable slope just like in the IPv4 plot up to the first 5 ASes an any given path.

It is worth noting that the large amount of ASes that only have a single IP address allocated to them is not really surprising: These are likely "stub"-ASes, with no customers of their own, which mostly participate in BGP to be able to purchase Internet access from multiple providers at the same time as a fail-safe.

### Date of Registration

From the same RIR delegation files, we can also get timestamps for when the ASNs where initially allocated. Intuitively, an older AS may be more likely to be more "important" in some way to Internet routing. Wether or not this would be better approximated by e.g. the customer cone is up for debate, but we hope that there still is something learnable in the data.

As shown in Figure 4.13, the distribution of allocated ASNs has been increasing over the years as one might expect. We deem the data to be reasonably distributed for BGPsyche to understand and take no further action to scale it.

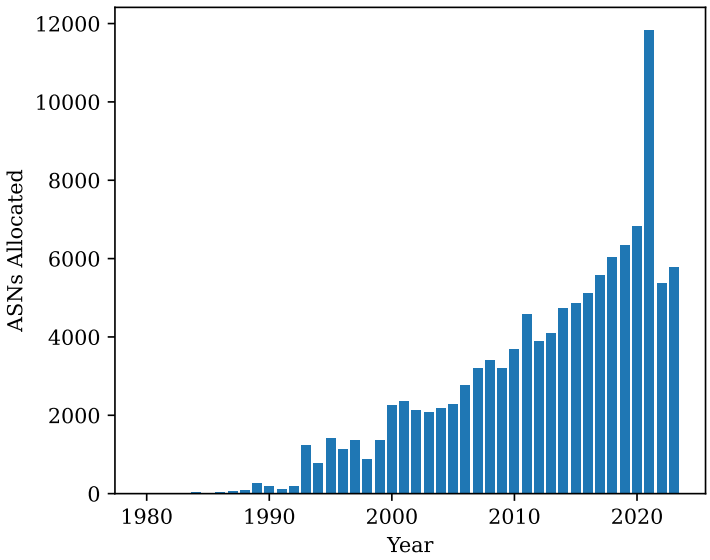



Figure 4.13.: ASNs Allocated per Year

### Geographic Distance to Source AS

Given a mapping from ASNs to countries [37], it is possible to get a (very) rough distance between two ASes by taking the center points of their home countries and computing the distance. For this feature, BGPsyche computes the distance between the AS and the *paths* source AS.

We hope that the inclusion of this feature can encode a rather basic intuition about AS paths: A path is rather unlikely to originate in a country $C$, then go all over the world only to come back to $C$ in the end.

Figure 4.14 plots the distance between the "current" AS on the path from the source AS over the hops to the destination AS. As one might predict, the mean distance to the source AS increases the closer we get to the destination AS. We interpret this to be a learnable distribution justifying the inclusion of the feature.

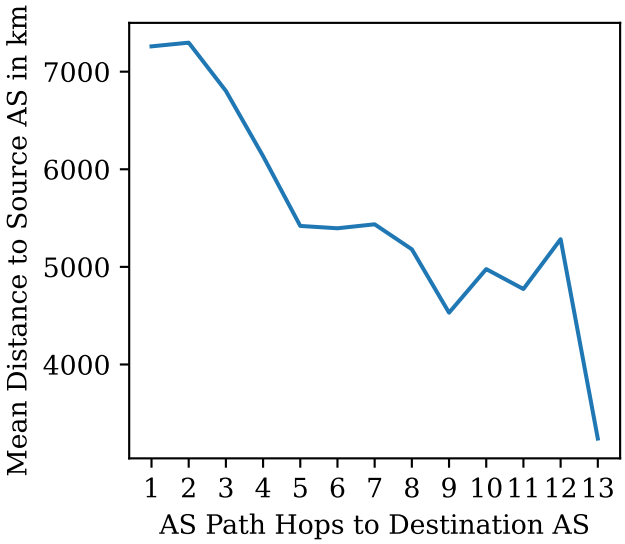



Figure 4.14.: Geographic Distance from each AS on the Path to the Source AS

### Correlation Matrix

To better understand how the features discussed in this section relate to each other, we plot a matrix of correlations between all features in Figure 4.15. Specifically, we compute Pearson's $r$ coefficient (see Section 4.4.1) and split up the AS category into separate dichotomous variables. The correlation between two dichotomous variables belonging to the same "parent" variable does not make sense to compute and so the plot shows these to be zero. To improve legibility, negative correlations are shown as their absolute positive values.

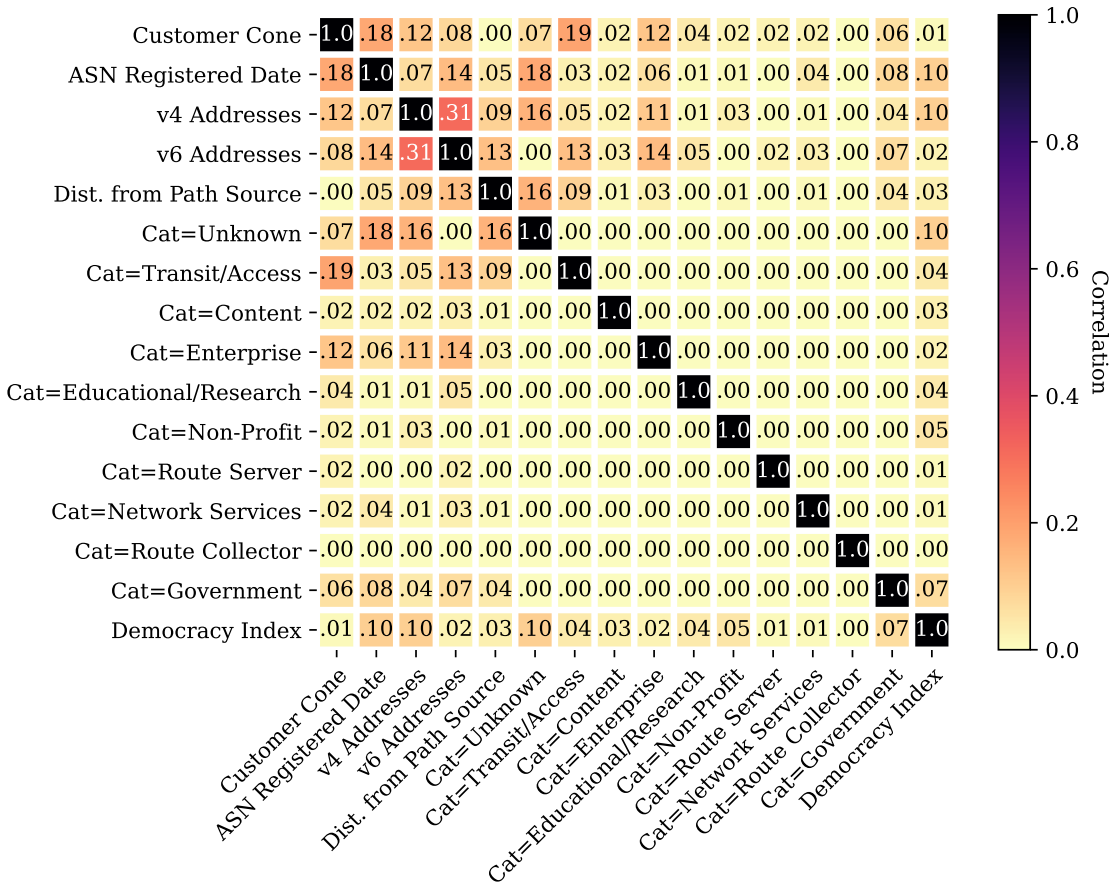



Figure 4.15.: AS Level Feature Correlation

Overall, we can see that while some features are related to each other, no single feature seems to correlate with any other so strongly as to make it obsolete. Even the number of allocated IPv4 and IPv6 addresses is not similar enough to justify excluding one of these features.

### 4.4.3. Link Features

This section covers features associated with links between ASes. We will again plot possible correlations between features in a correlation matrix. All listed features appear to be reasonable candidates for inclusion in the training of BGPsyche. The features discussed are:

- Trade Volume
- Type of Relationship
- Seen Count

#### Trade Volume

With the inclusion of this feature we hope to encode an approximation of political alliance or tension, similar to the previously discussed democracy index feature on the AS level. BGPsyche

uses the "trade in services annual dataset" from the World Trade Organization (WTO) [41], which tracks global trade between countries in USD. To compute the feature, we first map the links source and destination ASNs to countries (again using the RIPE "asn.txt" dataset [37]), get the bidirectional trade volume and normalize it by the countries total imports/exports. This way, we get a number between 0 and 1 that should be a decent indicator for how tightly the two countries cooperate economically.

Figure 4.16 shows the distribution of this feature across all links in RIPE RIS. While this alone does not allow us to judge the true utility of the feature, it does show that the feature is distributed enough to potentially contain something learnable.

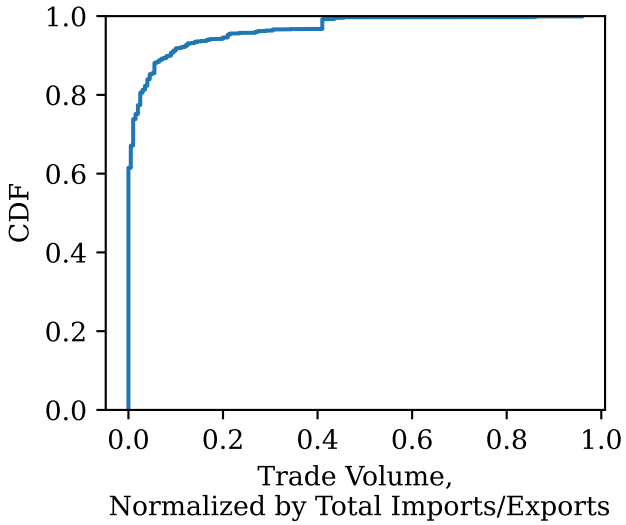



Figure 4.16.: Distribution of Trade Volume between Countries on AS Links, Normalized by Total Imports/Exports

### Type of Relationship

Again, we use the dataset of AS relationships from CAIDA covered in section 2.4.2 to annotate each link as downstream (p2c), peering (p2p) or upstream (c2p). Our hope is that using these relationships, BGPsyche can learn *something* more nuanced then valley-free routing, which, as a boolean value computed from these relationships algorithmically, can hardly predict the much more complex reality of BGP routing (as discussed in section 2.3).

One might object that the relationship between ASes is likely to correlate highly with the AS categories. For example, a link from an AS classified as "Government" to another AS classified as "ISP" is likely not a downstream connection. However, it is still easy to reason that the inclusion of the relationship adds a substantial amount of information: Consider a case, where we have a link between two "ISP" classified ASes. In this case, there is no way to predict the type of relationship. Similarly, having a p2c link may correlate with the source AS being an ISP, but does not even necessitate that. Therefore, the type of relationship is logically distinct from the AS category and worth including.

```python
CONFIDENCE_MIN = 5;

def get_link_confidence(
  destination: int,
  link_counts_at_src: dict[int, int],
) -> float: # [-1;1]
    if len(link_counts_at_src.keys()) == 0: return 0

    counts_sum = sum(link_counts_at_src.values())
    base_prob = 1 / len(link_counts_at_src.keys())

    # probability of link [0;1]
    prob = (
        (link_counts_at_src[dst] / counts_sum)
        if dst in link_counts_at_src else 0
    )

    delta = prob - base_prob

    # scale delta to [-1;1] using base_prob
    if base_prob == 1:
        direction = 1
    else:
        direction = (
            delta * (1 / base_prob)
            if delta < 0
            else delta * (1 / (1 - base_prob))
        )

    # [0;1]: if we have less then N links to compute prob from, this number goes
    # to 0
    counts_confidence = \
        min(counts_sum, CONFIDENCE_MIN) / CONFIDENCE_MIN

    return round(direction * counts_confidence, 2) # [-1;1]
```

Listing 4.2: Computing Link Confidence

### Confidence by Seen Count

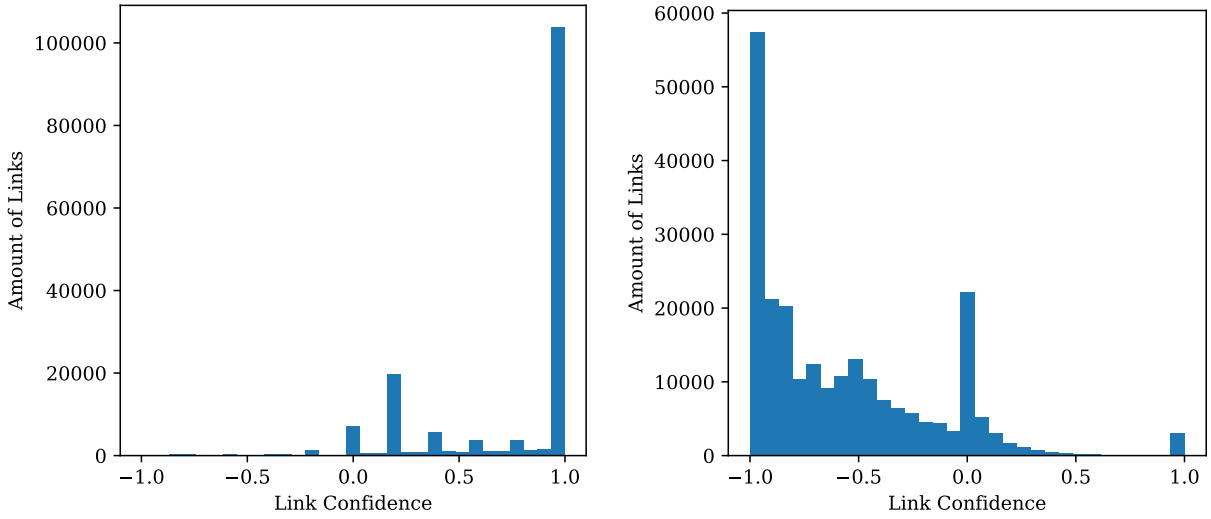
Using routing data from RIPE RIS again, we construct one BGP graph per destination AS of each input path, where each link is annotated with how often it was seen in the input paths. In addition, we construct such a graph without differentiating destination ASes.

From these graphs, BGPsyche can then compute two separate confidence values – one for the per-destination graph and one for the full graph – for each link using the algorithm in Listing 4.2.

The resulting feature is a confidence value from $-1$ to 1, based on how often we saw the given link in the training data *in relation to other links* from the same AS. We acknowledge that using

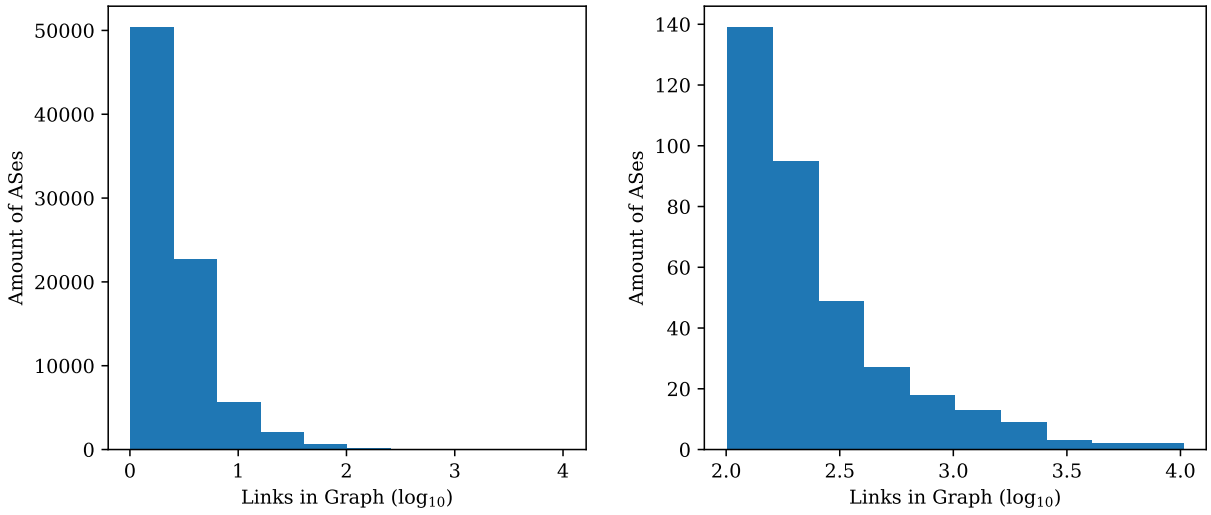
this feature has a high risk of overfitting BGPsyche to its training data. For now, we defer that discussion to the later Section 6.5.

The distribution of this feature over all links in RIPE RIS is shown in Figure 4.17. We observe a completely different distribution when using the per-destination graphs in comparison to using the full graph. Intuitively, the reason for this may be that the per-destination graphs are much more sparse and therefore the links that do exist have a higher confidence value on average. Conversely, there is a small set of ASes in the full graph that account for a large amount of links. We plot the number of links per AS in Figure 4.18. For example, 0.4 % of ASes – those with more than 100 links – account for 34 % of links. Therefore, most links have to "compete" with many other links for their confidence value, which results in the uneven distribution.




(a) Per Destination Graphs (b) Full Graph

Figure 4.17.: Distribution of Calculated Link Confidence




(a) Full Distribution (b) Tail of Distribution

Figure 4.18.: Distribution of the Number of Links per AS in the Full Graph

### Correlation Matrix

In the same we have done already for AS level features, we compute the correlation between all link level features in Figure 4.19. While there is a noticeable correlation between some of them, none correlate to an extend that would make them superfluous.

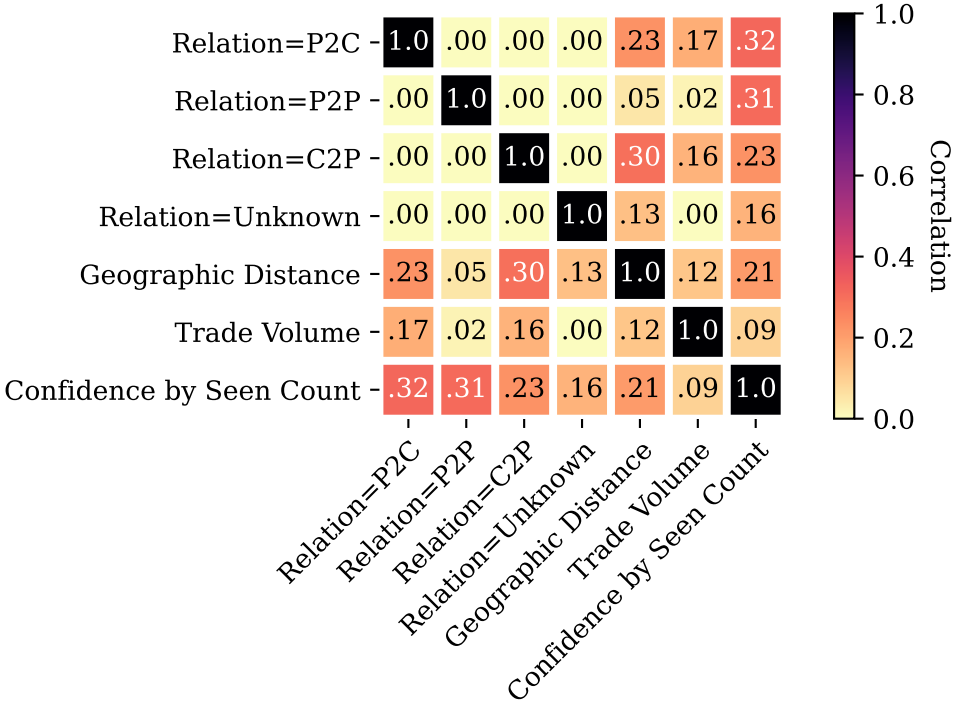



Figure 4.19.: Link Level Feature Correlation

### 4.4.4. Path Features

Finally, BGPsyche also uses some features computed from the entire path. This section will cover the following features:

- Length
- Valley-Free
- Real Snippet Length
- Confidence by Link Seen Count

Overall, we show that all features correlate significantly with the correctness of the path.

### Length

The length of a path should somewhat obviously correlate with its correctness, as BGP is still fundamentally a protocol that attempts to find a shortest path, albeit with many exceptions due to policy configuration. We note that the length has what we would deem a significant correlation of 17 % with the correctness of the path. For the sake of completeness, we plot the distribution of path lengths in Figure 4.20.

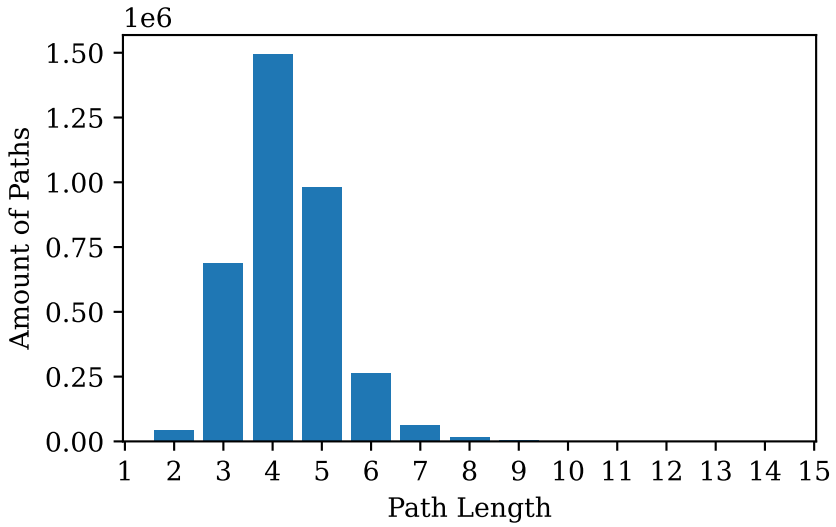



Figure 4.20.: Distribution of AS Path Length

### Valley-Free

We hope for BGPsyche to learn something more nuanced than the valley-free constraint using the types of link relationships discussed in Section 4.4.3. Therefore, this feature will *not be used in the final model*.

This section exists only to point out the correlation of 18 % we found between the correctness of a path and it being valley-free. While this is a significant correlation, it is still arguably lower than one might expect, considering how important this feature is generally seen in the literature.

### Real Snippet Length

Given a list of real paths from RIPE RIS and RouteViews, we can find the longest real path snippet of the given candidate path. This should intuitively correlate strongly with path correctness, since we already know that the snippet in question can be used in a real path. Indeed we find a very high correlation of 67 %.

Figure 4.21 depicts the distribution of this feature over a random selection of 10k paths from RIPE RIS and RouteViews. As one might expect, longer snippets are harder to find in candidate paths.

The plot also shows that there are significantly more candidate paths containing at least one link that we have previously seen than paths which contain only completely foreign links.

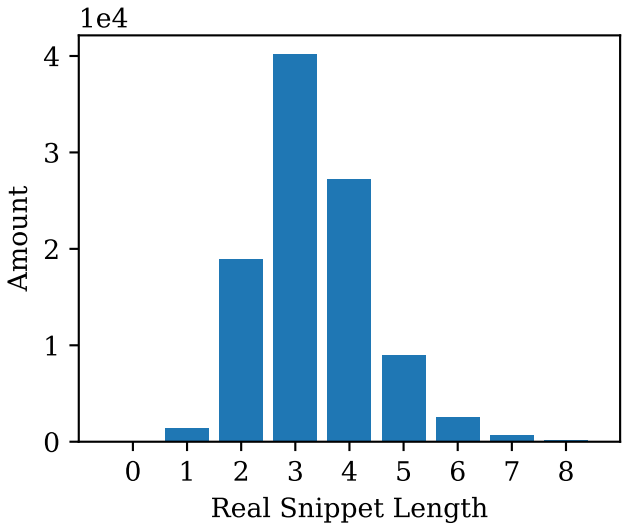



Figure 4.21.: Distribution of Longest Real Path Snippet Length

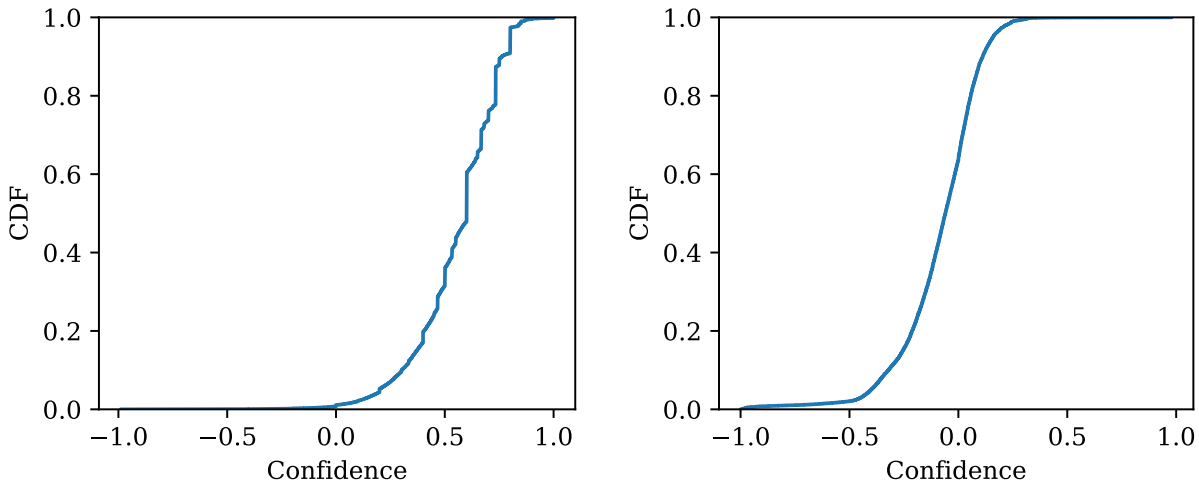
As with the usage of link confidence values in Section 4.4.3, including this feature in the training process has a high risk of overfitting. Again, we defer this discussion to Section 6.5.

For potential future research, we note here that one could imagine more complex algorithms that return a sort of "familiarity" score. For example, one could get the ratio of links in the candidate path that were seen in real paths, or attempt to fit as many real path snippets to the candidate path as possible.

### Confidence by Link Seen Count

Given the per-link confidence values from Section 4.4.3, we can compute confidence values for the entire path by taking the mean confidence value from all links, ignoring links that are not in the given graph.

Figure 4.22 plots the distribution of this confidence value over all paths from RIPE RIS. We note that – especially for the confidence based on the per-destination graphs – there are very few paths that are considered "actively" unlikely (confidence values below zero). This may seem initially as though it would clash with the data in Figure 4.17, where we discovered that most links have a low confidence value. However, logic would dictate that links that are considered unlikely are probably not used much in real AS paths, which explains the distribution seen here.




(a) Per Destination Graphs

(b) Full Graph

Figure 4.22.: Distribution of Path Confidence

## Correlation Matrix

To conclude, we plot a correlation matrix of the path features discussed in Figure 4.23. Since this plot is about path features, we have the luxury of being able to correlate features with the correctness of the path. In this case, a high correlation is desirable. Of course, as with all other features, correlation between features should still not be too high. We observe again that no two features correlate highly enough to make one of them superfluous.

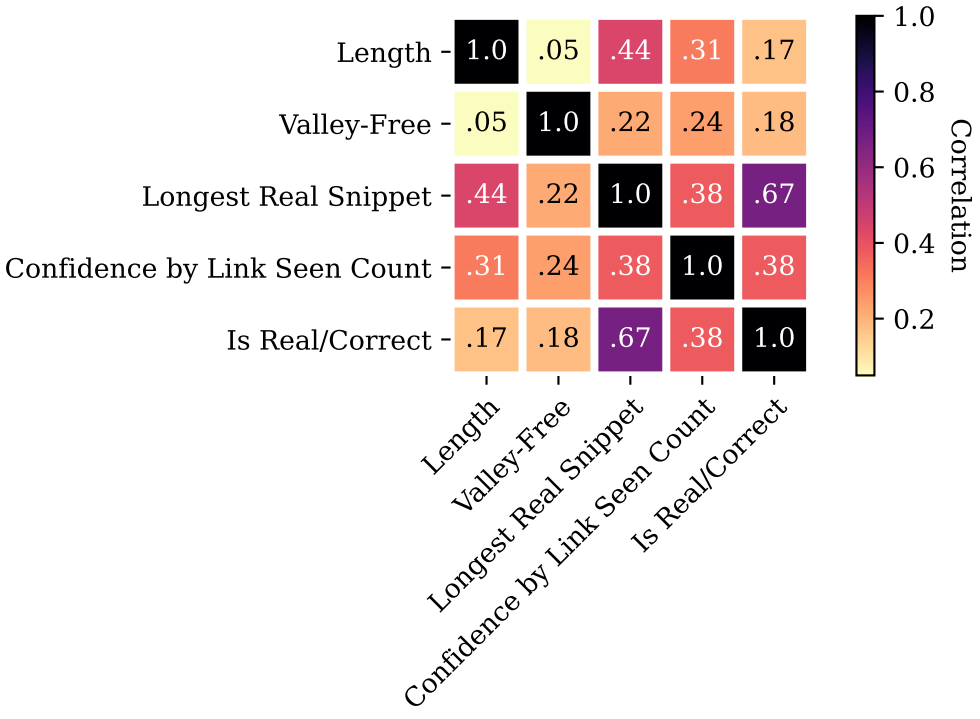



Figure 4.23.: Path Level Feature Correlation

## 4.5. Computing a Test/Training Dataset

Using all of the discussed features, BGPsyche computes a dataset to train on and test its own performance. It does this by looping over a random selection of RIPE RIS paths, getting a list of false candidates for each one real path, retrieving all features for each path and finally "vectorizing" these features.

"Vectorization" refers to the process of turning an arbitrary datastructure into a series of vectors that a machine learning algorithm can understand. There are many ways to do this based on the given input data, but for BGPsyche, we only need to discuss two concepts:

- **One-Hot Encoding**: Given a *categorical* variable with $N$ categories, one can encode that as $N$ binary variables. This can be expressed as a mapping. For example, given the input categories $A$, $B$ and $C$, on-hot encoding is equivalent to the mapping $A \rightarrow [1,0,0]; B \rightarrow [0,1,0]; C \rightarrow [0,0,1]$. There are other ways to encode categorical data, but given the small set of categories that BGPsyche deals with as input features, this simple approach is sufficient.
- **Scaling and Normalization**: Generally speaking, it is wise to scale *numerical* features such that their distribution is spread to a reasonable interval and to normalize them to a common range, such as $[0;1]$. The scaling was already addressed in the sections for the individual features. BGPsyche additionally normalizes all numerical values to the range $[0;1]$.

## 4.6. Path Ranking

The previous sections established various datapoints BGPsyche could use to rank AS paths. There are a number of machine learning approaches that one could use to train on this data. We chose to use two Recurrent Neural Networks (RNNs), combined with two Multi Layer Perceptrons (MLPs). This section attempts to justify that choice and explore some of the alternatives, some of which may be of interest for future research.

BGPsyche uses supervised learning (that is, we have a labeled dataset of correct and incorrect paths). More specifically, the job of the ML model can be seen as either a binary classification or ranking task. Fundamentally, we want the model to sort a list of candidate paths, hopefully putting the correct path at the top of the list.

While binary classification algorithms are often understood to only provide "yes" or "no" as an output, they actually output floating point numbers between 0 and 1. These are typically collapsed into a boolean value, but BGPsyche can use them to sort its list of candidates.

### 4.6.1. Non Deep Learning Approaches

The key reason why we chose to use a deep learning approach was because more traditional machine learning algorithms for supervised binary classification seem to all require a single flat, fixed size vector of input features. This does not map to our problem space, as each AS path

can have a variable number of ASes (and links) on the path, each with its own feature vectors. Therefore, we would have to somehow attempt to collapse the AS- and link-level features into a fixed amount of features or omit them entirely. Consequently, we chose to not pursue non deep learning approaches.

### 4.6.2. BGPsyche Model Architecture

Recurrent Neural Networks form the basis of the BGPsyche model architecture. They allow a sequence of variable length to be the input, while the output is still a fixed size vector. This allows BGPsyche in turn to combine the output of one RNN operating on AS features, another RNN using link features and an MLP processing path features together into one final MLP that produces the output floating point confidence value, as depicted in Figure 4.24.

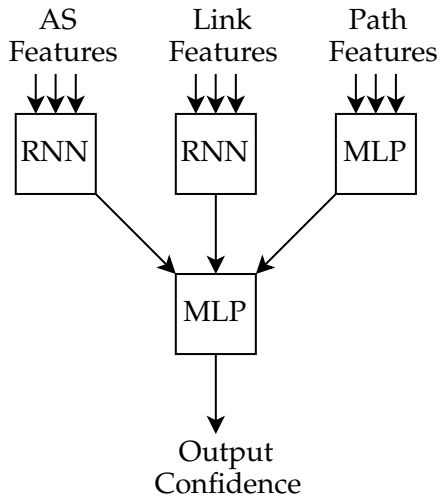



Figure 4.24.: BGPsyche Model Architecture

The way that BGPsyche uses RNNs is similar to the more commonly performed task of *sentiment analysis*, where one feeds a series of words from e.g. a movie review into an RNN to predict for example how positive or negative the review is. Structurally, we do the same thing by feeding a series of AS or link features into an RNN to predict how likely the path is.

### 4.6.3. Alternative Approaches

This section briefly outlines three possible alternative approaches to path ranking and even discovery. The implementation of these approaches would certainly be interesting, but it is out of scope for this thesis.

- **Graph Neural Networks (GNN):** Graph Neural Networks can be understood as being able to learn a mapping from a graph datastructure to a fixed size vector. GNNs have already been shown to be capable of detecting anomalies in BGP routing [42]. One could imagine using a GNN for path prediction in two ways:

  - Construct a BGP "graph" from just the given candidate path and let the GNN estimate its correctness. It is difficult to say whether this would produce better or worse results than our approach, but it should intuitively not differ substantually since it would learn from the exact same feature vectors.
  - Construct a local BGP graph *around* the given candidate path and let the GNN estimate its correctness. This approach has some promise because it gives the GNN the ability to evaluate a path within a wider context. An AS path might make sense when looking at it in isolation, but when taking some neighboring ASes into consideration, it could becomes less sensible.

- **Reinforcement Learning:** Reinforcement learning is a technique where one trains an agent to take actions in a virtual space. One could imagine an AS graph to be the virtual space, where the trained agent decides how to reach a given destination AS from a given source. While perhaps a promising idea, this is out of scope for BGPsyche, as we wish to fundamentally sort a list of candidate paths instead of training such an agent.
- **Learning to Rank (LTR):** Learning to Rank algorithms such as LambdaMART [31] have already been shown to be able to effectively rank small amounts of candidate paths in RouteInfer [11]. While this may seem to make them very promising initially, they can only be used to rank fixed size feature vectors, which makes the encoding of properties of ASes and links difficult. It shares this issue with many machine learning algorithms that we will not list here. LTR is only mentioned specifically to acknowledge the use in RouteInfer.

# 5. Feature Selection and Hyperparameters

In the previous chapter we discussed the fundamental architecture of BGPsyche, as well as many potentially useful features of ASes, links and paths. This chapter justifies how we set specific parameters in the machine learning algorithms we use as well as which features we include or omit.

## 5.1. The Necessity of Intuition and the Silver Model

Ideally, one would train a given machine learning model hundreds or thousands of times, every time systematically testing different values for parameters and combinations of features. When enough compute and time is available or the training dataset is rather small, this is actually also done in practice. However, for this thesis, compute and time are finite resources and we are dealing with a large training dataset. As will be discussed in more detail later, the process of training and evaluating BGPsyche takes us at least 80 minutes. This drastically limits the amount of experiments that can be run in the weeks allotted to parameter optimization in the thesis's schedule. Therefore, we must sometimes rely on reasoning more then empirical evidence to justify our choice in parameters and features. In cases where we do want to provide some empirical reason, we can only train and run BGPsyche a few times to do so.

Since we cannot test all combinations of parameters and features, we have to – somewhat counter-intuitively – first define the final parameters and set of features BGPsyche uses. We shall call the model trained on these the "silver" model. Despite the fact that there may still be some further optimization possible, it is the best configuration we were able to find.

Tables 5.1 and 5.2 list the features and hyperparameters used in the silver model, respectively. In the following sections we will run many experiments where we deviate from the silver model in one feature or parameter value at a time, all of which will justify the values set here.

| Feature | Section | Included |
|---|---|---|
| AS: Customer Cone and ASRank | 4.4.2 | ✓ |
| AS: Category | 4.4.2 | ✓ |
| AS: Democracy Index | 4.4.2 | ✗ |
| AS: Registered IP Addresses | 4.4.2 | ✓ |
| AS: Geographic Distance to Source AS | 4.4.2 | ✓ |
| Link: Trade Volume | 4.4.3 | ✗ |
| Link: Type of Relationship | 4.4.3 | ✓ |
| Link: Confidence (Per Destination) | 4.4.3 | ✗ |
| Link: Confidence (Full Graph) | 4.4.3 | ✗ |
| Path: Length | 4.4.4 | ✓ |
| Path: Valley-Free | 4.4.4 | ✓ |
| Path: Real Snippet Length | 4.4.4 | ✓ |
| Path: Confidence (Per Destination) | 4.4.4 | ✓ |
| Path: Confidence (Full Graph) | 4.4.4 | ✓ |

Table 5.1.: Features Included in the Silver Model

| Hyperparameter | Section | Value |
|---|---|---|
| Balance of Classes | 5.3.2 | 5 |
| Training Duration | 5.3.3 | 5k Epochs |
| Learning Rate | 5.3.4 | 0.01 |
| Activation Functions | 5.3.5 | tanh in RNN, ReLU in MLP |
| Amount of Neurons per Layer | 5.3.6 | 8 layered RNN \| MLP 32, 16, 8, 1 |

Table 5.2.: Hyperparameters Used in the Silver Model

## 5.2. Measuring Model Performance

To discuss how deviations from the silver model affect the results, we first have to address how to measure success. To that end, this section defines the test setup and gives a brief introduction on some common metrics used in machine learning, as well as metrics that are highly specific to the task of ranking AS paths. Lastly, we will also address the issue of training stability (or lack thereof).

### 5.2.1. Test Setup

The metrics discussed in the following two sections are taken in a fixed test setup. The setup is relatively intuitive and follows from the structure and usage of BGPsyche as discussed up to this point.

We start by creating a dataset as discussed in Section 4.5 and splitting it into separate *test* and *training* datasets, where the test dataset contains 20 % of the original records and the remaining 80 % are used for training. Doing this is common practice in machine learning because it allows one to immediately see when the model is *overfitting*: When a model is trained too much on the training data, it will often start to simply "memorize" the training dataset. This improves its performance on the training data, but predictions about other data suffer. To avoid this, we plot the loss on both the test and training dataset during training. When the training loss decreases but the test loss increases, the model is starting to be overfit. By cancelling the training before that happens or by reverting to a previous training snapshot, overfitting can be avoided. In practice, overfitting never occurred in any of the experiments listed in this thesis.

From the training dataset we then train a model, identical to the silver model in all but one aspect, again leaving out the test dataset for evaluation. Additionally, we get a list of random real AS paths from RIPE RIS (as usual, from all full tables of all collectors from a single timestamp) and ask BGPsyche to predict these real paths by only giving it the source and destination ASNs.

### 5.2.2. Accuracy, Precision, Recall, F1-Score

Fundamentally, BGPsyche can be understood to be a *binary classifier* in the sense that, for any given path, it simply has to make a binary decision as to whether it is the correct path or not. Accuracy, Precision, Recall and the F1-Score are used in almost all works of machine learning concerning binary classifiers in some capacity. They give us a sensible foundation to talk about the performance of BGPsyche. Given the ubiquity of these metrics, this section is intended to give the reader an intuition of their meaning rather then provide rigorous definitions. We compute these metrics on the *test dataset*, not the training data.

- **Accuracy** simply tells us how many times a prediction was correct. A high accuracy however is not sufficient to conclude that anything of value has been learned. Consider an example where BGPsyche is given 99 incorrect paths for every real path. It could then learn to simply always judge every path to be incorrect, which would yield a 99 % accuracy. However, if the input dataset is *balanced* – that is, if the classifier only ever has to decide between one true and one false value – the accuracy is actually a really good measure of the models performance.
- **Precision** refers to the rate of true positives. In other words, when BGPsyche predicts that a path is correct, how often is that true?
- **Recall** is the ratio of true values that are also predicted to be true. In other words, if BGPsyche is given 100 incorrect and ten correct paths, how many of those ten correct paths where actually predicted to be correct? Not unlike accuracy, recall is meaningless on its own, as the model could just learn to predict everything to be correct and achieve 100 % recall.
- The **F1-Score** combines precision and recall into a single metric. It is useful to think of the F1-Score as a replacement or proxy for accuracy when the dataset is *not balanced*.

### 5.2.3. Correct Predictions, Edit Distance and Candidate Positioning

To really determine the performance of BGPsyche, it is perhaps generally more useful to look at domain-specific metrics rather then the aforementioned general-purpose metrics for binary classification.

In order to explain these metrics, we must briefly address an important difference between BGPsyche's training process and its actual usage to predict real paths: The training data only contains a small amount of incorrect paths for every correct path, while running a real path prediction produces hundreds or thousands of candidate paths, where by definition only one of those may be correct. The reason for this will be discussed in more detail in Section 5.3.2. What this means though is that computing accuracy, precision, recall and the F1 score on the dataset will not necessarily match the real-world performance of BGPsyche. Therefore, we run BGPsyche the same way a user would, but do so automatically and measure its performance in several domain-specific metrics. To have something to compare against, we ask BGPsyche to predict a random selection of paths from RIPE RIS.

- The amount of **correct predictions** is the amount of paths that were correctly predicted by BGPsyche.
- We also measure the **edit distance** between the predicted and real path.
- Finally, we take note of the **position** of the real path in the list of sorted candidates. A lower average position indicates that the model is better at judging the probability of candidate paths.

### 5.2.4. Training Stability

The optimization of neural networks is generally a task which deliberately includes some level of randomness. This means that the retraining of a model on the same dataset can result in different levels of performance.

Curiously, we find that while the accuracy, precision, recall and F1-score stay almost completely stable when retraining, the domain specific measurements (correct predictions, edit distance and position) often vary significantly.

In fact, we find that the accuracy, precision, recall and the F1 score only change in situations where the real world model performance worsens or improves *very* drastically. For example, a model that during training (on different datasets) showed an F1 score of 80 % (the most typical value we observe) could predict anything from 30 % to 70 % of paths correctly. Because of this, we do not use them to measure model performance in the following sections. For almost every experiment we run, including training the silver model, we get an accuracy of 90 %, a precision of 67 %, a recall of 98 % and an F1 score of 80 %.

The amount of real-world correct path predictions can differ as much as 20 % when retraining on the same dataset. Table 5.3 shows the results of ten iterations of retraining the silver model.

Usually, the model seems to predict between 60 % and 65 % of paths accurately. In some rarer cases, performance as low as 45 % accuracy is possible.

| Correct Predictions | Mean Edit Distance | Mean Path Position |
|---|---|---|
| **45 %** | 0.9 | 2.9 |
| **66 %** | 0.4 | 0.6 |
| **66 %** | 0.5 | 0.6 |
| **49 %** | 0.9 | 3.9 |
| **61 %** | 0.6 | 1.0 |
| **65 %** | 0.5 | 0.6 |
| **45 %** | 0.9 | 24 |
| **61 %** | 0.6 | 0.8 |
| **65 %** | 0.5 | 0.7 |
| **65 %** | 0.5 | 0.6 |

Table 5.3.: Model Performance after Training on the Same Data Ten Times

When for example testing the inclusion or omission of a feature in the training dataset, we must therefore interpret any change within the ranges shown in Table 5.3 to be within the margin of error. If a feature has only a slight impact on real-world performance, positive or negative, we will not be able to detect it. Strictly speaking, this would mean that we would have to completely ignore changes in accuracy as high as 20 %. However, given that training the silver model *usually* results in a performance of 60 % to 65 %, we will still mention and consider results below 60 % to be a light indicator for worse performance. In order to still provide slightly more stable results, we will train the model twice for many configurations we test.

## 5.3. Hyperparameter Optimization

As stated in Section 4.6.2, we have chosen to use a combination of two RNNs together with two simple MLPs. However, we still need to make many more decisions about the shape and parameters of the model. For example, we need to set a loss function and choose how to incorporate nonlinearity. Additionally, most components of the model are configurable with certain constants (e.g. the amount of layers of artificial neurons). These constants, generally referred to as hyperparameters in the language of machine learning, can have a significant impact on the learning process and the resulting performance of the model. This section therefore is dedicated to empirically justify BGPsyche's model design and hyperparameter values.

### 5.3.1. Loss Function and Optimizer

In the language of machine learning, a loss function allows us to numerically estimate how "different" the actual prediction of a model is from the desired prediction. Consequently, the choice

of a loss function influences the models understanding of its own errors and therefore the training process.

Informal Internet research seems to indicate that "sigmoid binary cross entropy loss" is the most common choice for binary classification. In addition, two other academic works training neural network binary classifiers on BGP routing data also use this loss function in their implementation [42, 43], with good results. We therefore choose to fall in line with what seems to be the accepted default.

The optimizers task is then to attempt to find a more optimal set of weights in response to a lower loss value. Much like with loss functions, we justify our choice in the "ADAM" optimizer by informal Internet research as well as the successful use in prior research related to BGP [43, 42].

### 5.3.2. Balance of Classes

When computing the training dataset (see Section 4.5), we need to decide on how many false candidate paths should be included for every real path. The chosen binary cross entropy loss function must also be configured to account for class imbalance.

Typically, the chosen class imbalance should resemble reality. In our case however, there are up to 2000 candidate paths (see Section 4.3.4) that BGPsyche will have to decide between during runtime. When attempting to train the model with this kind of imbalance, we saw that it was too difficult for the model to learn anything; It simply "decided" to classify every path as false and was content with the 99 % accuracy this resulted in.

From this we can infer that the class imbalance in the training dataset should be much lower, but we still need to run experiments to determine *how much* lower. Table 5.4 shows the results of training BGPsyche using two, three, five and ten false candidates per real path. We do not include the resulting loss, F1-Score or other synthetic benchmarking values here because the changes in class balance make them hardly comparable. Crucially, we find that using a class imbalance of five drastically improves the sorting position of the real path. From this data, we consider a class imbalance of 5 to be adequate and use it in the silver model.

| | Correct Predictions | Mean Edit Distance | Mean Path Position |
|---|---|---|---|
| 2 Candidates per Real Path | **49 %** | 0.8 | 2.3 |
| 2 Candidates per Real Path | **55 %** | 0.7 | 1.9 |
| 3 Candidates per Real Path | **63 %** | 0.6 | 0.7 |
| 3 Candidates per Real Path | **62 %** | 0.6 | 1.0 |
| 5 Candidates per Real Path | **65 %** | 0.5 | 0.6 |
| 5 Candidates per Real Path | **61 %** | 0.6 | 1.0 |
| 10 Candidates per Real Path | **65 %** | 0.5 | 0.7 |
| 10 Candidates per Real Path | **66 %** | 0.5 | 0.6 |
| Silver Model (5 Candidates per Real Path) | **66 %** | 0.4 | 0.6 |

Table 5.4.: Model Performance after Training with Different Class Balances

In order to still fit into memory, we had to use only 50k real paths for the dataset with 10 false candidates per real path. For the others, we used the previously established size of 100k real paths. Again, rewriting the BGPsyche's code to be able to handle larger datasets than what can fit into (video) memory should likely be addressed by potential future research.

### 5.3.3. Training Duration

The training duration can be seen as consisting of two parts: Dataset creation and the actual training of the model. Creating the dataset as discussed in Section 4.5 takes us substantially longer than the actual training on the available hardware. It should be noted that the dataset creation can be parallelized to however many CPU cores are available, while the training must be done on a single GPU/TPU. In any case, creating a dataset of 100k real paths with 5 false candidate paths each (see previous Section) takes us around 50 minutes on a 12 core CPU.

We attempt training BGPsyche on a dataset constructed from 50k, 100k and 125k real paths (with five false candidate paths each) and and show the results in Table 5.5. The results indicate that 100k real paths yield the best results. This seems counterintuitive, as one would probably expect the model performance to plateau at a certain amount of paths instead of degrading significantly as observed here. We think that these results are caused by our training hardware: A dataset of 125k paths just barely fits into our 16 GB of GPU VRAM and also mostly fills up our 16 GB of CPU RAM. Moreover, the OS started to use the allocated swap space a lot more than with the 100k dataset. Since creating the dataset and testing the model depends on the time-sensitive computation of candidate paths, it is likely that the quality and quantity of these candidate paths suffers when swapping, resulting in a severe impact in model performance. This problem may be addressed in future research by rewriting some parts of BGPsyche's code. For now, we respect the results and use a dataset with 100k real paths (and 500k false candidate paths).

| Real Paths in Dataset | Correct Paths Predicted | Mean Edit Distance | Mean Path Position |
|---|---|---|---|
| 50k | 60 % | 0.6 | 1.3 |
| 50k | 58 % | 0.7 | 2.5 |
| 100k | 61 % | 0.6 | 1.0 |
| 100k | 65 % | 0.5 | 0.6 |
| 125k | 66 % | 0.5 | 0.6 |
| 125k | 65 % | 0.5 | 0.6 |
| Silver Model (100k) | 66 % | 0.4 | 0.6 |

Table 5.5.: Model Performance after Training with Different Dataset Sizes

Generally speaking, a deep learning model should be trained until the test loss no longer decreases significantly. This training time is usually not measured in minutes, but in *epochs*, where one epoch refers to the model seeing the entire training dataset once. It is customary to train a model for several epochs, sometimes even quite a large number. When training, one must also take care to not train for too long, lest the model is overfit to the training data, resulting in subpar predictions for inputs that do not precisely match the training data. Usually, overfitting is made visible by an increase in the test loss with a simultaneous decrease of the training loss.

Table 5.6 shows BGPsyche training for 5k, 10k and 20k epochs, with the percentage of correctly predicted paths when evaluating the given model. We do not include plots for the test or training loss because – despite varying model performance – they curiously are always the same: Both the test and training loss always arrive at $0.23$ after $700$ epochs and remain there. We therefore also did not experience any overfitting. In any case, the results indicate that there is no significant improvement to be gained beyond 5k epochs, which we consequently choose to train the silver model with.

| Epochs | Correct Predictions | Mean Edit Distance | Mean Path Position |
|---|---|---|---|
| 5k | 61 % | 0.6 | 0.7 |
| 5k | 65 % | 0.5 | 0.6 |
| 10k | 65 % | 0.5 | 0.7 |
| 10k | 53 % | 0.8 | 16 |
| 20k | 45 % | 0.9 | 8 |
| 20k | 58 % | 0.7 | 0.9 |
| Silver Model (5k) | 66 % | 0.4 | 0.6 |

Table 5.6.: Model Performance after Training for Different Amounts of Epochs

Training for 5k epochs takes about 40 min and the evaluation of 10k random paths another 40 min on our available hardware. This means that all further tests in this section take about 1.2 h to complete.

### 5.3.4. Learning Rate

The learning rate of a training process influences by how much the optimizer will "jump" to new weight values in response to the loss. A learning rate that is set too high will likely result in the model not learning much at all, because the weight values would change to erratically to find a local minimum loss. Conversely, a learning rate that is set too low will result in the model perhaps only optimizing for a local minimum instead of discovering a global minimum, in addition to also increasing the training duration.

Table 5.7 shows the results of training BGPsyche with a learning rate of 0.1, 0.01, 0.001 and 0.0001. From this data, we conclude that a learning rate of 0.01 is adequate for our use.

| | Correct Predictions | Mean Edit Distance | Mean Path Position |
|---|---|---|---|
| Learning Rate 0.0001 | **48 %** | 1.0 | 2 |
| Learning Rate 0.0001 | **36 %** | 1.0 | 19 |
| Learning Rate 0.001 | **61 %** | 0.6 | 17 |
| Learning Rate 0.001 | **63 %** | 0.6 | 1.6 |
| Learning Rate 0.01 | **61 %** | 0.6 | 0.8 |
| Learning Rate 0.01 | **65 %** | 0.5 | 0.7 |
| Learning Rate 0.1 | **17 %** | 1.6 | 173 |
| Learning Rate 0.1 | **18 %** | 1.6 | 170 |
| Silver Model | 66 % | 0.4 | 0.6 |

Table 5.7.: Model Performance after Training with Different Learning Rates

### 5.3.5. Activation Functions (nonlinearity)

Neural Networks are by default linear functions. This means that – in this default state – they are best suited to learn linear problems, such as linear regression. If one wants a neural network to "understand" non-linear problems – or more generally, problems that appear somewhat complex – it is wise to use non-linear *activation functions*. Conceptually, these sit between the layers of the network and control the activation threshold and intensity of neurons.

There are a number of commonly used activation functions. We choose to test the two that are arguably the most common: `tanh` and `ReLU`. `ReLU` can be expressed as $f(x) = \max(0, x)$ and `tanh` refers to the mathematical function of the same name (the hyperbolic tangent). Common wisdom seems to be that a simpler function like `ReLU` is often desirable because it performs well enough while being significantly easier to compute. Hoarau et al. also use a `ReLU` activation function in their model to detect BGP anomalies [42].

Table 5.8 shows the results of training BGPsyche with different sets of activation functions. Because none of the results suggest that one combination of activation function(s) performs clearly better than the other, we choose to use `tanh` for the RNN since it is the default setting of the ML library we use (pytorch) and `ReLU` for the MLP because it was also used in [42].

| | Correct Predictions | Mean Edit Distance | Mean Path Position |
|---|---|---|---|
| ReLU in RNN, ReLU in MLP | **65 %** | 0.5 | 0.6 |
| ReLU in RNN, tanh in RNN | **65 %** | 0.5 | 0.6 |
| tanh in RNN, ReLU in MLP | **65 %** | 0.5 | 0.6 |
| tanh in RNN, tanh in MLP | **61 %** | 0.6 | 0.9 |
| Silver Model | 66 % | 0.4 | 0.6 |

Table 5.8.: Activation Function Performance

### 5.3.6. Amount of Layers and Neurons per Layer

Both the RNN and MLP components of BGPsyche can be configured for an arbitrary amount of layers, each with different amounts of neurons. For this section, we denote the amount of neurons in the MLP component as a comma-separated list: For example, the list "8, 4, 1" describes an MLP with eight neurons in the first layer, four in the second layer and finally one output neuron.

Table 5.9 shows the results of training BGPsyche with various amounts of layers and neurons. From these results, it seems that the amount of layers in the RNNs does not have a great effect on the outcome. However, something about the combination of "16, 8, 4, 2, 1" neurons in the MLP components made BGPsyche perform noticeably worse. Perhaps having this kind of "long tail" of layers slowly becoming smaller is detremental to performance. In any case, we choose to use an 8 layered RNN with a layer configuration of "32, 16, 8, 1" for the MLP for use in the silver model, as there does not seem to be an improvement in performance to be gained by further increasing these values.

| | Correct Predictions | Mean Edit Distance | Mean Path Position |
|---|---|---|---|
| 4 layered RNN \| MLP 16, 8, 4, 2, 1 | **43 %** | 1.0 | 4.2 |
| 4 layered RNN \| MLP 16, 32, 16, 8, 1 | **64 %** | 0.6 | 0.7 |
| 2 layered RNN \| MLP 16, 8, 1 | **59 %** | 0.7 | 1.2 |
| 8 layered RNN \| MLP 32, 16, 8, 1 | **65 %** | 0.5 | 0.6 |
| 16 layered RNN \| MLP 32, 16, 8, 1 | **65 %** | 0.5 | 0.6 |
| 8 layered RNN \| MLP 64, 32, 16, 8, 1 | **63 %** | 0.6 | 0.7 |
| Silver Model | 66 % | 0.4 | 0.6 |

Table 5.9.: Performance of Different Amounts of Layers and Neurons

## 5.4. Feature Selection

Having configured the models hyperparameters, we now need to decide which of the features we proposed and analyzed in Section 4.4 we really want include in the training data. Up until now, we have only built an intuition that these features *may* contain learnable data. Now, we will empirically justify their inclusion/omission in the silver model. We include features that were omitted in the silver model and omit those that were included.

Features may interact and complement one another in complex ways. Because of this, we choose to only omit a feature if it worsens the model performance. This of course means that we may be training on a number of unnecessary features. However, optimizing this further would require testing all possible combinations of features, which is not feasible in a reasonable time frame.

### 5.4.1. AS Features

We start by evaluating the utility of AS level features in Table 5.10. The data suggests that the customer cone and AS category are probably the most relevant features, because their omission worsens model performance the most (specifically the average path position). The amount of registered IPs and date of registration, which we get from RIR data, as well as the geographic distance to the source AS seem to not affect the model negatively, hence why we keep them in our training dataset. Finally, the inclusion of the democracy index does again worsen the result, so we omit it.

| | Correct Predictions | Mean Edit Distance | Mean Path Position | Include |
|---|---|---|---|---|
| Omit Customer Cone and ASRank | **37 %** | 1.2 | 138 | ✓ |
| Omit AS Category | **63 %** | 0.5 | 4.4 | ✓ |
| Include Democracy Index | **58 %** | 0.7 | 3.4 | ✗ |
| Omit Amount of Registered IPs | **64 %** | 0.6 | 0.8 | ✓ |
| Omit Date of Registration | **64 %** | 0.6 | 0.6 | ✓ |
| Omit Geographic Distance to Source AS | **66 %** | 0.5 | 0.6 | ✓ |
| Silver Model | 66 % | 0.4 | 0.6 | |

Table 5.10.: Feature Selection for AS Level Features

### 5.4.2. Link Features

Next, we evaluate the utility of Link level features in Table 5.11. Omitting the distance between the ASes and the type of relationship clearly worsens model performance, so we include them in the training dataset. However, the inclusion of the link confidence surprisingly worsens model

performance substantially, so we omit the feature. Finally, the inclusion or omission of trade volume does not seem to impact performance, so we include it in the dataset.

| | Correct Predictions | Mean Edit Distance | Mean Path Position | Include |
|---|---|---|---|---|
| Omit Trade Volume | **65 %** | 0.6 | 0.5 | ✓ |
| Omit Distance in km | **54 %** | 0.8 | 2.9 | ✓ |
| Omit Type of Relationship | **45 %** | 1.0 | 22 | ✓ |
| Include Confidence (Per Destination) | **16 %** | 1.6 | 169 | ✗ |
| Include Confidence (Full Graph) | **16 %** | 1.6 | 166 | ✗ |
| Silver Model | 66 % | 0.4 | 0.6 | |

Table 5.11.: Feature Selection for Link Level Features

### 5.4.3. Path Features

Finally, we evaluate the utility of Path level features in Table 5.12. Perhaps unsurprisingly, the data suggests that the path length and the length of the longest real snippet are the most crucial features.

What is somewhat puzzling is just how much worse the performance gets when omitting the length of the longest real snippet. A cynical surface level reading of the data would suggest that BGPsyche is only an overly complex proxy for that one single feature. However, the performance also clearly suffers when we omit other features such as the customer cone, link relationship or AS category. We can therefore conclude that BGPsyche is definitely learning something more complex then that. Still, this suggests that in its current state, the system bases a lot of its assumptions on this one feature. We add that in previous versions of BGPsyche, this feature did not exist and we still achieved a performance of roughly 30 % accuracy. However, we are no longer able to reproduce this state in the final version. In summary, this feature seems absolutely essential to the current implementation, but it is also certainly not the only relevant component.

Interestingly, BGPsyche also performs significantly worse if we *include* the valley-free property of paths in the training data. We propose that this may be because BGPsyche is able to learn something more nuanced than valley-free on its own through the use of the type of relationship feature at the link level. A deeper investigation of this phenomenon may be of interest for future research.

| | Correct Predictions | Mean Edit Distance | Mean Path Position | Include |
|---|---|---|---|---|
| Omit Length | **11 %** | 1.9 | 325 | ✓ |
| Include Valley-Free | **28 %** | 1.7 | 86 | ✗ |
| Omit Real Snippet Length | **4 %** | 2.4 | 252 | ✓ |
| Omit Confidence (Per Destination) | **57 %** | 0.7 | 0.9 | ✓ |
| Omit Confidence (Full Graph) | **40 %** | 1.2 | 4 | ✓ |
| Silver Model | 66 % | 0.4 | 0.6 | |

Table 5.12.: Feature Selection for Path Level Features

# 6. Evaluation

This chapter details the accuracy of BGPsyche under various circumstances and in comparison to related works. We also discuss limitations, both inherent to many attempts at path prediction and specific to BGPsyche. Lastly, we justify the inclusion or omission of available features in the training process and briefly discuss cost and scalability.

## 6.1. Accuracy of Predicted Paths

BGPsyche finds the correct path between an arbitrary source and destination network with an accuracy of **66 %**. The set of the *top three* selected candidates contains the correct path **85 %** of the time. Additionally, the average edit distance between the selected path and the real path is 0.5.

## 6.2. Comparison with Related Work

| Algorithm | Accuracy |
|---|---|
| Shortest Valley-Free | 20 % (Section 3.1) |
| KnownPath (2006) [7] | 16-95 %, avg. 60 % [7], 59-64 % [11] |
| iPlane (2006) [4] | 54 % [28], 68 % [8], 70 % [4] |
| iPlane Nano (2006) [3] | 70 % [3], 38-52 % [11] |
| SIGCOMM06 (2006) [6] | 63 % [6], 50-58 % [11] |
| Routing Tree (2012) [5] | 24 % [11] |
| Sibyl (2016) [8] | > iPlane ? [10] |
| PredictRoute (2021) [10] | 45-60 % [11] |
| BGPSim (2020) [12] | 16 % |
| RouteInfer (2022) [11] | **82 %** [11] |
| BGPsyche (2024) | **66 %** |

Table 6.1.: Comparison of Accuracy of AS Path Prediction Systems

In Chapter 3, we gave an overview of various existing path prediction systems. We already mentioned their performance in passing, but we now want to more systematically compare them with BGPsyche. To that end, Table 6.1 summarizes the reported accuracies of each system. Recall from Chapter 3 that these systems were not tested under uniform conditions and sometimes choose to highlight different aspects of their performance. For example, the authors of PredictRoute [10]

never directly state the accuracy of their system, which forces us to rely on the evaluation of the authors of RouteInfer [11].

We make the following key observations:

- The most common accuracy of AS path prediction systems is arguably around 60 %. This is a *very* rough estimate, but it seems to hold in the majority of cases.
- The table shows how much the reported accuracy of a system can differ depending on the reporting authors and test setup. This further adds to the difficulty of making a comparison with BGPsyche.
- Surprisingly, there is little correlation between the age of a path prediction engine and its accuracy.

From the reported accuracies, RouteInfer is clearly the most advanced state-of-the-art route prediction system, achieving an impressive 82 % accuracy. Still, BGPsyche is competitive with all other solutions. A fusion of RouteInfer and BGPsyche is theoretically possible and could provide even better results.

## 6.3. Opaque Model Decisions

Understanding the decisions of a deep learning model is difficult in general. For BGPsyche, this means that when a falsely chosen candidate path is observed, one cannot simply attach a debugger and step through the decision making process to correct a mistake. Instead, one has to attempt to reason about something that *could* make the model decide a certain way. This in turn means the improvement of BGPsyche can involve time consuming amounts of trial and error.

## 6.4. Cost and Scalability

BGPsyche was trained on a single PC with consumer grade hardware. Specifically, it was trained on an *NVIDIA RTX 4060 Ti* with 16 GB of VRAM, 16 GB of regular RAM and a relatively modest *Ryzen 5 3600* 6-core/12-thread CPU. Loading the full dataset of paths into VRAM exhausts most of the 16 GB of VRAM and evaluating the model in 11 parallel processes exhaust both the CPU and the RAM.

Therefore, if one would want to increase the amount of training data or increase the speed of evaluation for faster development, more expensive hardware would be required. However, we are ultimately still talking about a relatively small budget for research when compared to many other ML related tasks.

Loading and running an already trained model can be done even on rather weak hardware. Predicting a single path on a single core of the mentioned CPU (clocked at 3.6GHz) takes around 6 s – discarding a 25 s initial startup time – and uses around 6 GB of RAM. Most of the time and space is spent on path candidate discovery and loading various cached data files. Loading and

running the actual ML model takes almost no time; The saved model parameters would fit on a floppy disk, taking up around 30 kB. Therefore, even a model trained on much more data and with many more parameters can still likely easily be run on relatively weak consumer hardware.

In addition, BGPsyche was trained exclusively on publicly available data. There are no additional cost factors in training it beyond paying for electricity and hardware. This is in contrast to some other path prediction systems like iPlane [4] or PredictRoute [10] which are built on active (traceroute) measurements.

## 6.5. Overfitting and Training Bias

The concept of overfitting was already briefly discussed in Section 5.2.1. However, up until now we have ignored any potential *bias* in the training data: If BGPsyche is trained only on the paths from say RIPE RIS and is then asked to predict routes as they would appear in the private routing table of a Tier 1 provider, it will likely struggle. One could say that in this case, we have overfitted BGPsyche to RIPE RIS.

This is a hard problem to solve for any path prediction system, as they must all in some way rely on available routing data. Even if they are not using any algorithm that is technically considered to be in the category of machine learning, they still fundamentally have a training phase that relies on a set of input paths to then make their predictions afterwards. Even a simple "shortest valley-free" algorithm (Section 3.1) must get its BGP graph and AS relationships from somewhere, which can be considered a training phase.

We can show how this problem affects predictions using our private dataset of Tier 1 routing data (Section 2.4.4). Table 6.2 shows some algorithms trained "only" on RIPE RIS and RouteViews attempting to predict paths from Cogent (AS174) and NTT (AS2914), as well as other paths from RIPE RIS as a baseline. For these two Tier 1 ASes, we observe a roughly 20 % reduction in prediction accuracy.

| | Correct Predictions | Mean Edit Distance | Mean Path Position |
|---|---|---|---|
| NTT | **41 %** | 1.2 | 30 |
| Cogent | **40 %** | 1.3 | 13 |
| RIS (baseline) | **65 %** | 0.6 | 0.5 |

Table 6.2.: Training on RIS and RouteViews to Predict Paths from NTT and Cogent

Regrettably, this sort of data access is not available to many researchers, which means that this analysis is not typically included in articles about other path prediction systems. It is therefore difficult to say to what extent BGPsyche is affected by this type of overfitting compared to other systems.

# 7. Conclusion and Future Work

Despite the high potential utility and a long tradition of research, AS path prediction engines still fall short of being truly reliable, with the state-of-the-art approach RouteInfer [11] reaching an accuracy of 82 %. In this thesis, we have developed BGPsyche, a novel machine learning approach to AS path prediction. BGPsyche in its current implementation finds the correct path in the set of its top three selected candidates 85 % of the time and selects the correct path as its top candidate with an accuracy of 66 %, making it fall in line with most of its competition, eclipsed only by RouteInfer. Due to the relative ease of adding new features to the training dataset (compared to designing and implementing new algorithmic approaches), it may be possible to improve BGPsyche significantly.

BGPsyche also shows that a machine learning approach, or more specifically, one based on neural networks, is feasible for AS path prediction. We think this motivates future research on the performance of other machine learning approaches for the same task, such as graph neural networks or reinforcement learning.

In the process of designing BGPsyche, we have also given an overview and analysis of many data points relevant to (ML based) path prediction and BGP routing more broadly. We think these efforts could be used as a starting point for other ML based path prediction engines.

## 7.1. Improving Path Discovery

In Section 4.3, we discussed the implementation of an algorithm to discover a list of candidate paths between two ASes. We were able to build it such that it can find 94 % of real paths. While we deem this to be acceptable for the purposes of this thesis, there is obviously still some room for improvement. For example, it might be interesting to see how the use of BGPSim [12] or KnownPath [7] would affect this number. Additionally, the performance of BGPsyche may (or may not) be improved by filtering the candidate paths by some basic criteria. For example, one could filter paths with ASNs that were not allocated by IANA to a RIR or from a RIR to an ISP or other organization (also known as *bogon* routes). In any case, the path discovery stage of BGPsyche grants the opportunity to hard-code conventional wisdom for path prediction before entering the often hard to reason about world of machine learning. It should be noted though, that filtering too heavily may decrease overall accuracy, as it may prevent the ML model from learning non-trivial nuances about AS paths which conventional wisdom does not account for.

Taking this approach further, it is also feasible to use BGPsyche only as a “fallback” when more traditional algorithms can no longer decide between candidate paths, much like RouteInfer [11] uses its ML model. In fact, as we hinted at before, a fusion of BGPsyche with specifically RouteInfer seems promising.

## 7.2. A Larger Training Dataset

Because of the way BGPsyche is currently written, the entire training dataset must fit into (V)RAM for training. This limits the amount of paths we could train on using our 16 GB of VRAM to 600k, with 100k real paths and 500k generated false candidate paths (as discussed in Section 5.3.3). Additionally, using a larger training dataset would have made it less feasible to run all of the analyses for Chapter 5. Future work on BGPsyche or similar systems could better explore the effect of a (much) larger training dataset. Aside from limited time and compute, there is no fundamental reason why a system like BGPsyche could not be trained on millions of paths.

## 7.3. Additional Path Metadata

In Section 4.4, we went into detail on many features one can consider when ranking a candidate path. This section lists some additional features that may be worth exploring for future work:

- **Tranco website popularity ranking:** An AS may be strongly associated with some websites. For example, the WHOIS (RIR database) entries of an AS may contain email-addresses with a domain that also hosts a valid website, or the website may be listed in PeeringDB. One could then use a website popularity ranking like `tranco-list.eu` to get a sense of whether the AS is operated by the maintainers of a popular website. This could be somewhat prone to errors though: Consider a business which hosts a popular web service `a.com`, but also hosts another site about its wider activities under `b.com`. It would then likely point PeeringDB and other databases to `b.com`, which would influence the ranking we obtain. Perhaps one could list all websites connected to an ASN by querying the DNS records for every IP in the prefixes the AS announces over BGP and then pick the highest rated website found. Additionally, one could compute the cumulative ranking of all websites in the customer cone of an AS as a separate feature.
- **BGP2Vec:** In 2022, Shapira et al. [21] published a promising paper, where they introduce BGP2Vec, an embedding method similar in principle to the widely used Word2Vec [44]. The word "embedding" can be understood as synonymous to a feature vector (hence the "2Vec"). On a basic level, BGP2Vec allows us to learn a feature vector for any given ASN by training on a list of real paths, instead of manually selecting a set of features to include. In theory, this technique can lead to impressive results and the authors do show promising initial results in their chosen use-case of computing AS relationships. It would be possible to attempt replacing all features on the AS level (Section 4.4.2) with BGP2Vec or even just add it to the existing features.

Thanks to the extensibility of machine learning algorithms in general and BGPsyche in particular, the inclusion these or other features and testing how they impact performance is almost trivial once the data has been acquired. Researchers in the field may be able take advantage of this property by coming up with many other features not listed here.

# A. Appendix

## A.1. Flattened ASdb to BGPsyche Category Mapping

| Flattened ASdb Category | BGPsyche Category |
|---|---|
| Media, Publishing, and Broadcasting | Content |
| Finance and Insurance | Enterprise |
| Education and Research | Government |
| Service | Enterprise |
| Agriculture, Mining, and Refineries (Farming, Greenhouses, Mining, Forestry, and Animal Farming) | Enterprise |
| Community Groups and Nonprofits | Non-Profit |
| Construction and Real Estate | Enterprise |
| Museums, Libraries, and Entertainment | Enterprise |
| Utilities (Excluding Internet Service) | Government |
| Health Care Services | Government |
| Travel and Accommodation | Enterprise |
| Freight, Shipment, and Postal Services | Enterprise |
| Government and Public Administration | Government |
| Retail Stores, Wholesale, and E-commerce Sites | Enterprise |
| Manufacturing | Enterprise |
| Computer and IT - Internet Service Provider (ISP) | Transit/Access |
| Computer and IT - Phone Provider | Transit/Access |
| Computer and IT - Hosting, Cloud Provider, [...] | Content |
| Computer and IT - Computer and Network Security | Enterprise |
| Computer and IT - Software Development | Enterprise |
| Computer and IT - Technology Consulting Services | Enterprise |
| Computer and IT - Satellite Communication | Transit/Access |
| Computer and IT - Search | Enterprise |
| Computer and IT - Internet Exchange Point (IXP) | Route Server |
| Computer and IT - Other | Unknown |
| Computer and IT - Unknown | Unknown |

Table A.1.: Flattened ASdb to BGPsyche Category Mapping

# Bibliography


[1] Weitong Li et al. "RoVista: Measuring and Analyzing the Route Origin Validation (ROV) in RPKI". In: *IMC 2023 - Proceedings of the 2023 ACM Internet Measurement Conference*. 2023. DOI: 10.1145/3618257.3624806.

[2] Rishab Nithyanand et al. *Measuring and Mitigating AS-level Adversaries against Tor*. 2015. arXiv: arXiv:1505.05173. preprint.

[3] Harsha V Madhyastha et al. "iPlane Nano: Path Prediction for Peer-to-Peer Applications". In: *NSDI 2009 - Proceedings of the 2009 USENIX Symposium on Networked Systems Design and Implementation*. 2009. DOI: 1558977.1558987.

[4] Harsha Madhyastha et al. "iPlane: An Information Plane for Distributed Services". In: *OSDI 2006 - Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*. 2006. DOI: 10.5555/1298455.1298490.

[5] Phillipa Gill, Michael Schapira, and Sharon Goldberg. "Modeling on Quicksand: Dealing with the Scarcity of Ground Truth in Interdomain Routing Data". In: *ACM SIGCOMM Computer Communication Review* 42.1 (2012). DOI: 10.1145/2096149.2096155.

[6] Wolfgang Mühlbauer et al. "Building an AS-Topology Model That Captures Route Diversity". In: *SIGCOMM 2006 - Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 2006. DOI: 10.1145/1159913.1159937.

[7] Jian Qiu and Lixin Gao. "CAM04-4: AS Path Inference by Exploiting Known AS Paths". In: *IEEE Globecom*. 2006. DOI: 10.1109/GLOCOM.2006.27.

[8] Italo Cunha et al. "Sibyl: A Practical Internet Route Oracle". In: *NSDI 2016 - Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation*. 2016. DOI: 10.5555/2930611.2930633.

[9] Rachee Singh and Phillipa Gill. "PathCache: A Path Prediction Toolkit". In: *SIGCOMM 2016 - Proceedings of the 2016 ACM SIGCOMM Conference*. 2016. DOI: 10.1145/2934872.2959053.

[10] Rachee Singh et al. "PredictRoute: A Network Path Prediction Toolkit". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5.2 (2021). DOI: 10.1145/3460090.

[11] Tianhao Wu et al. "RouteInfer: Inferring Interdomain Paths by Capturing ISP Routing Behavior Diversity and Generality". In: *Passive and Active Measurement*. Springer, 2022. DOI: 10.1007/978-3-030-98785-5_10.

[12] Italo Cunha. *TopologyMapping/Bgpsim: BGP Route Propagation Simulator*. 2020. URL: https://github.com/TopologyMapping/bgpsim (visited on 10/23/2023).

[13] Dominik Stahmer. *Dominiksta/Bgpsyche*. GitHub. 2024. URL: https://github.com/dominiksta/bgpsyche (visited on 07/18/2024).

[14] Yakov Rekhter, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. Request for Comments RFC 4271. Internet Engineering Task Force, 2006. URL: https://datatracker.ietf.org/doc/rfc4271 (visited on 07/26/2024).

[15] Russ Housley et al. *The Internet Numbers Registry System*. Request for Comments RFC 7020. Internet Engineering Task Force, 2013. URL: https://datatracker.ietf.org/doc/rfc7020 (visited on 07/26/2024).

[16] Vince Fuller and Tony Li. *Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*. Request for Comments RFC 4632. Internet Engineering Task Force, 2006. URL: https://datatracker.ietf.org/doc/rfc4632 (visited on 07/26/2024).

[17] João Luis Sobrinho. "Network Routing with Path Vector Protocols: Theory and Applications". In: *SIGCOMM 2003 - Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 2003. DOI: 10.1145/863955.863963.

[18] Lixin Gao. "On Inferring Autonomous System Relationships in the Internet". In: *GLOBECOM 2000 - Proceedings of the IEEE Global Telecommunications Conference*. Vol. 1. 2000. DOI: 10.1109/glocom.2000.892035.

[19] Matthew Luckie et al. "AS Relationships, Customer Cones, and Validation". In: *IMC 2013 - Proceedings of the 2013 ACM Internet Measurement Conference*. 2013. DOI: 10.1145/2504730.2504735.

[20] Lixin Gao and J. Rexford. "Stable Internet Routing without Global Coordination". In: *IEEE/ACM Transactions on Networking* 9.6 (2001). DOI: 10.1109/90.974523.

[21] Tal Shapira and Yuval Shavitt. "BGP2Vec: Unveiling the Latent Characteristics of Autonomous Systems". In: *IEEE Transactions on Network and Service Management* 19.4 (2022). DOI: 10.1109/TNSM.2022.3169638.

[22] RIPE Network Coordination Center. *Routing Information Service (RIS)*. 2024. URL: https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ (visited on 07/29/2024).

[23] University of Oregon. *FAQ – RouteViews*. 2024. URL: https://www.routeviews.org/routeviews/index.php/faq/ (visited on 07/29/2024).

[24] *About CAIDA*. CAIDA. 2020. URL: https://www.caida.org/about/ (visited on 02/26/2024).

[25] CAIDA. *AS Relationships Dataset*. 2024. URL: https://www.caida.org/catalog/datasets/as-relationships/ (visited on 10/24/2023).

[26] Larry Peterson. *PlanetLab | Farewell Notice*. Systems Approach. 2020. URL: http://www.systemsapproach.org/1/post/2020/03/its-been-a-fun-ride.html (visited on 04/14/2023).

[27] Jin Y. Yen. "Finding the K Shortest Loopless Paths in a Network". In: *Management Science* 17.11 (1971). DOI: 10.1287/mnsc.17.11.712.

[28] Harsha V. Madhyastha et al. "A Structural Approach to Latency Prediction". In: *IMC 2006 - Proceedings of the 2006 ACM Internet Measurement Conference*. 2006. DOI: 10.1145/1177080.1177092.

[29] Jerome H. Friedman and Bogdan E. Popescu. *RuleFit*. 2012. URL: http://archive.today/2017.03.15-113459/http://statweb.stanford.edu/~jhf/R_RuleFit.html (visited on 04/30/2024).

[30] Jerome H. Friedman and Bogdan E. Popescu. "Predictive Learning via Rule Ensembles". In: *The Annals of Applied Statistics* 2.3 (2008). DOI: 10.1214/07-AOAS148.

[31] Qiang Wu et al. "Adapting Boosting for Information Retrieval Measures". In: *Information Retrieval* 13.3 (2010). DOI: 10.1007/s10791-009-9112-1.

[32] Patrick Maigron. *World ASN Statistics by Number*. 2024. URL: https://www-public.imtbs-tsp.eu/~maigron/rir-stats/rir-delegations/world/world-asn-by-number.html (visited on 03/07/2024).

[33] CAIDA. *ASRank: A Ranking of the Largest Autonomous Systems (AS) in the Internet.* 2024. URL: https://asrank.caida.org/ (visited on 04/12/2024).

[34] CAIDA. *AS Classification*. 2015. DOI: https://catalog.caida.org/dataset/as_classification. URL: https://catalog.caida.org/dataset/as_classification.

[35] *PeeringDB*. 2024. URL: https://www.peeringdb.com/about (visited on 04/12/2024).

[36] Maya Ziv et al. "ASdb: A System for Classifying Owners of Autonomous Systems". In: *IMC 2021 - Proceedings of the 2021 ACM Internet Measurement Conference*. 2021. DOI: 10.1145/3487552.3487853.

[37] RIPE NCC. *AS Names Dataset*. 2024. URL: https://ftp.ripe.net/ripe/asnames/asn.txt (visited on 04/19/2024).

[38] Our World in Data. *Democracy Index*. 2022. URL: https://ourworldindata.org/grapher/democracy-index-eiu?time=latest (visited on 04/15/2024).

[39] Economist Intelligence Unit. *Democracy Index*. 2022. URL: https://www.eiu.com/n/campaigns/democracy-index-2022/ (visited on 04/15/2024).

[40] RIPE NCC. *RIR Statistics Exchange Format*. 2022. URL: https://ftp.ripe.net/pub/stats/ripencc/RIR-Statistics-Exchange-Format.txt (visited on 04/15/2024).

[41] World Trade Organization. *Trade Statistics - Services Annual Dataset*. 2023. URL: https://www.wto.org/english/res_e/statis_e/trade_datasets_e.htm (visited on 04/19/2024).

[42] Kevin Hoarau, Pierre Ugo Tournoux, and Tahiry Razafindralambo. "BGNN: Detection of BGP Anomalies Using Graph Neural Networks". In: *ISCC 2022 - Proceedings of the 2022 IEEE Symposium on Computers and Communications*. 2022. DOI: 10.1109/ISCC55528.2022.9912989.

[43] Kevin Hoarau, Pierre Ugo Tournoux, and Tahiry Razafindralambo. "Detecting Forged AS Paths from BGP Graph Features Using Recurrent Neural Networks". In: *CCNC 2022 - Proceedings of the 2022 IEEE 19th Annual Consumer Communications & Networking Conference*. 2022. DOI: 10.1109/CCNC49033.2022.9700668.

[44] Tomas Mikolov et al. *Distributed Representations of Words and Phrases and Their Compositionality*. 2013. arXiv: arXiv:1310.4546. preprint.

# Glossary

**BGP Looking Glass** A (usually public) internet service, allowing a user to look up BGP or IP routes from the looking glass to arbitrary destinations on the internet. 1, 11

**BGPsyche** The novel AS path prediction system based on deep learning we develop in this thesis. iii, VI, VII, 2–4, 6, 7, 9, 13–17, 19–27, 29–33, 35–40, 43–56, 58, 60–65

**Center for Applied Internet Data Analysis** An organization that in their own words "conducts network research and builds research infrastructure to support large-scale data collection, curation, and data distribution to the scientific research community". V, 6

**Internet Exchange Point** A physical location where ASes can enter public peering agreements. V

**Internet Service Provider** An AS that is set up to allow consumers and businesses to pay for access to the internet. V, 3

**PeeringDB** A public database containing information about ASes relevant to negotiating peering sessions. All data is entered by participating ASes voluntarily. VI, 26–30, 64

**Regional Internet Registry** An organization that, among other things, is responsible for assigning ASNs and IP addresses for a region of the world. IV, V, 3, 31

**RIPE RIS** A Route Collector from RIPE. VII, 6, 7, 10, 13, 16, 18, 19, 23, 32, 36–38, 40, 41, 43, 48, 49, 62

**Route Collector** A BGP speaking router that does not route any traffic, but allows (any) third party to announce their routes to it. The collector then makes its database of paths publicly available for researchers and network operators. IV, 6, 7, 17, 48

**RouteViews** A Route Collector from the University of Oregon. VII, 6, 7, 10, 13, 16, 18, 40, 62

**Réseaux IP Européens (Network Coordination Centre)** The Regional Internet Registry for Europe, the Middle East and Central Asia. V

**Tier 1** An AS is said to be Tier 1 if it does not have to pay other ASes for access to the entire internet. Instead, Tier 1 ASes are large and powerful enough to negotiate mutual peering agreements with other Tier 1 ASes for the same access. 4, 7, 18, 62

**traceroute** A piece of software, typically used on the command line, to identify IP addresses of routers on the path to a given destination IP/URL. 1, 4, 7, 8, 11, 13, 14, 62

**valley-free** An AS path is considered valley-free when the type of relationships (upstream/downstream/peering) on each link do not form a valley, visually speaking. A path not being valley-free is often considered a strong indicator that the path is not correct. More in Section 2.3. iii, 5–7, 10–13, 15, 36, 39, 40, 58

# Acronyms

**ASN** AS Number. IV, VI, 3, 8, 14, 16, 19, 21, 22, 27, 28, 30–33, 36, 48, 63, 64

**BGP** Border Gateway Protocol. IV, 1

**CAIDA** Center for Applied Internet Data Analysis. 6–8, 10, 12, 14, 26, 27, 36

**ISP** Internet Service Provider. 3, 36, 63

**IXP** Internet Exchange Point. 4, 29

**ML** Machine Learning. 17, 25, 43, 55, 61–63

**MLP** Multi Layer Perceptron. 43, 44, 47, 50, 55, 56

**RIPE** Réseaux IP Européens (Network Coordination Centre). 3, 7, 11, 12, 14, 18, 28, 31, 36

**RIR** Regional Internet Registry. VI, 3, 5, 19, 28, 30, 32, 57, 63, 64

# List of Figures

# List of Tables



# List of Listings

# Affidavit

I hereby declare that this thesis is my own work, that I have not submitted it elsewhere for examination purposes, have not used any sources or aids other than those indicated, and have marked verbatim and indirect quotations as such.

Ingolstadt, _____

_____

Dominik Stahmer