UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Toward realistic network topologies

Dissertation presented by
Maxime De Mol , Gauthier Feuillen

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Olivier Bonaventure

Reader(s)
Steven Gay, Stefano Vissicchio

Academic year 2015-2016

Acknowledgements

We would like to express our gratitude to our advisor Olivier Bonaventure for its time and useful comments. Furthermore we would like to thank Cyrille Dejemeppe, Sascha van Cauwelaert and Steven Gay for taking part in the design of the second solver. We would like to thank CAIDA for their publicly accessible topology data, without which this work would have been a lot more complicated.

Abstract

Obtaining realistic network topologies is crucial when designing routing protocols or trying to study the structure of the internet. They are usually not freely available for various reasons.

In this context, we present Hydrogen, a new engine capable to infer network topologies from external measurements. Our aim is to provide the scientific community with a new tool capable of generating a router-level topology with link weights. This tool source code is freely available at `https://bitbucket.org/gfeuillen/hydrogen/overview`.

In this thesis, we first explain how to infer router topologies, we then describe the link weights inference and then provide the topologies obtained by combining the two firsts parts.

Throughout this thesis we provide quality measures of our implementation in order to prove its efficiency or find its weaknesses. More specifically, these measures allow us to prove that our topologies sometimes present inaccuracies (mostly due to shortcomings in the external measurements) while the inference of link weights is able to produce weights correctly characterizing the routing.

Contents

Lexicon

- **AS:** An Autonomous System, or a collection of nodes/routers under the control of a network operator.
- **ISP:** An Internet Service Provider is an organization that provides internet access to its users.
- **ASN:** Each AS possesses a unique identifier, the Autonomous System Number.
- **AS path:** A list of AS through which a packet has passed to get to a specified router.
- **IP:** The Internet Protocol is the principal communication protocol in the suite of internet protocols for relaying packets across networks.
- **IPv4:** Internet Protocol version 4, is the fourth version of IP. IPv4 uses 32-bit addresses to identify routers and hosts. IPv4 addresses are written as 123.234.123.234, where each group corresponds to 8 bits, and can take any value between 0 and 255.
- **IPv6:** Internet Protocol version 6 is the sixth and most recent version of the Internet Protocol. IPv6 deals with the address exhaustion issue that is facing IPv4. IPv6 uses 128-bit addresses which are written as `2001:0db8:0000:0000:0000:ff00:0042:8329`, where each group of 4 hexadecimals characters corresponds to 16 bits. In an IPv6 address, leading zeroes can be omitted, leading to `2001:db8:0:0:0:ff00:42:8329`. IPv6 can even go a step further by eliminating consecutive sections of zeroes, resulting a compressed address like `2001:db8::ff00:42:8329`.
- **IP prefix:** An IP address prefix, written as `1234:5678::/n`, is a pattern that matches the first n binary bits of an IP address. Example: `2001:468::/32` is the 32 bits prefix of `2001:468:2:209::1`
- **Packet:** A network packet is the base unit of data carried by a network. A packet consists of control information, known as the header, and user data, known as the payload. The header contains important informations like the source address and destination address.
- **Traceroute:** A diagnostic tool used to display the route taken by a packet to reach its destination address. The term traceroute also designates those paths. A more detailed description of traceroute is available in the Introduction.
- **DNS:** Domain Name System, is a hierarchical naming system for computers and resource connected to the internet. It provides translation from the IP addresses to more human adapted addresses. For example, `google.com` is easier to process and remember by us than `2001:6a8:3081:6f01:8ca:a12d:a2c6:d22e`.

- **BGP:** The Border Gateway Protocol, and more specifically eBGP (*e* stands for exterior), is a standardized exterior gateway protocol designed to exchange routing information among ASes. Exterior means it's used to route packets between ASes, as opposed to internal routing that happens inside an AS.
- **BGP table:** A routing table used by BGP routers to route packets to their destination. An BGP table entry consists in a destination prefix, and a list of possible AS paths to reach this destination.
- **Load balancing:** The distribution of the workload across multiple computing resources. In the particular case of networking, load balancing designates the distribution of the bandwidth across all available paths.
- **ICMP:** Internet Control Message Protocol, used by routers to send errors messages, like unreachable host or dropped message errors, back to the source router.
- **ECMP:** Equal-Cost Multi-Path, a routing strategy where next-hop packet forwarding to a destination can occur over multiple "shortest paths". The path length is calculated using the sum of link's "weights", a value that describe a link quality.
- **TTL:** The Time To Live, sometimes called hop limit, is a value that defines the lifespan of a packet. The TTL is decremented by each router the packet visits. When this limit is reached, the packet is dropped.
- **RTT:** Round-Trip Time, the length of time a packet takes to reach its destination plus the time the acknowledgement takes to arrive to the source of the packet.
- **Ingress router:** The first router belonging to the ISP of interest encountered on a traceroute.
- **Egress router:** The last router belonging to the ISP of interest encountered on a traceroute.
- **Interface:** Routers can have many different connectors and types of connectors. These connectors are designated as interfaces. Different interfaces usually have different IP addresses.
- **Alias:** The different interfaces of a router, with different IP addresses are called aliases.
- **Alias resolution:** Resolving which aliases belong to the same physical router.
- **Inter-Routing:** Routing packets between different ASes. BGP is an inter-routing protocol.
- **Intra-Routing:** As opposed to inter-routing, intra-routing designate the routing of packets inside an AS. OSPF, IS-IS and MPLS are intra-routing protocols.
- **OSPF:** Open Shortest Path First is a interior routing protocol. This means that it is used to route packets within an AS. Every OSPF router exchange informations about the links it shares with neighboring routers, with the rest of the routers in the network. They are then able to compute the whole network topology, and use this topology to generate routing tables, by using Dijkstra algorithm [2] to compute the shortest paths.
- **IS-IS:** Intermediate System to Intermediate System is another routing protocol. It is very similar to OSPF. While OSPF is IP version sensitive (which means that to support IPv6, a new version of OSPF, called OSPF v3 had to be developed), IS-IS is IP version neutral which allowed it to be easily used to support IPv6. IS-IS is also more modular and faster

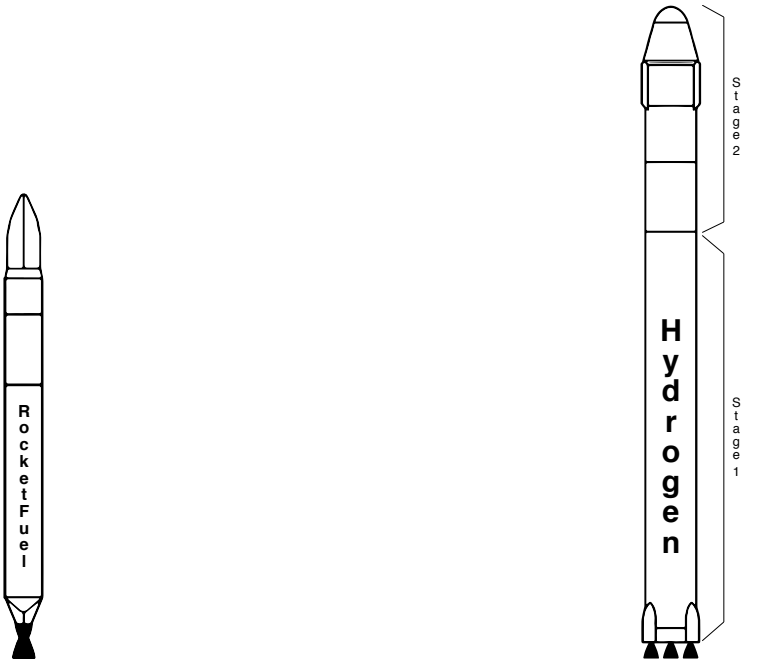
to converge (time needed for all the routers to receive complete information about the topology) than OSPF.

- **MPLS:** MultiProtocol Label Switching is another intra-routing protocol, for high-performance networks. MPLS uses short labels for inter-router labels instead of long addresses, in order to simplify the routing tables and avoid complex lookups. Labels also permit the setup of completely arbitrary paths set by the network administrators, instead of more deterministic protocols like shortest path.
- **Weathermap:** Network diagnostic tool used to display, in a graphical way, the load on the links of a given network.
- **Regular expression:** Sequence of characters that define a search pattern. It is mainly used in pattern matching, for example in strings.
- **Machine learning:** The study and construction of algorithms capable of learning from data and making prediction based on the data.

Introduction

RocketFuel and Hydrogen

In February 2002 a team lead by Neil Spring of the University of Washington published a paper about measuring ISP topologies through traceroutes [3]. The objective of this project, called *Rocketfuel*, was to give the research community access to a resource that was unavailable at that time: realistic ISP topologies, that would be as close to the reality as possible.
The paper presents techniques that significantly reduce the number of traceroutes required to measure accurately a router-level ISP topology, compared to a brute-force approach, and how traceroute collection and graph inference can be done. This approach lays the bases of a method to accurately and efficiently produce ISP maps, and enable us to easily reproduce those results.



(a) RocketFuel (b) Hydrogen

Figure 1: RocketFuel vs. Hydrogen

But RocketFuel was imperfect, and topologies inferred by this tool proved to be less accurate than the paper originally suggested [4], notably in the American *Sprint* network. Moreover, over 14 years later RocketFuel is rapidly becoming obsolete. The internet routing landscape has changed, new routing strategies have appeared, IPv6 has started to replace IPv4, and resources wildly available at the time of RocketFuel's creation (such as public traceroute vantage points)

Listing 1: Traceroute Example

```
   traceroute from 2001:48d0:101:501::247 to 2620:149:f8::1
1  2001:48d0:101:501::18  3.802 ms
2  2001:468:e00:c48::1  2.847 ms
3  2607:f380::108:9a41:af60  3.349 ms
4  2607:f380::118:9a40:b011  11.378 ms
5  2620:149:bb:8220::5  11.931 ms
6  2620:149:bb:2a42::a2  13.235 ms
7  2620:149:bb:20f::42  19.897 ms
8  2620:149:bb:1c20::62  20.177 ms !P
```

have now become rare or even unavailable. A total ground-up reimplementation was needed, and from this need was born *Hydrogen*

Hydrogen is a more modern tool, and takes account of the extensive changes in network technology. For example, Instead of only focusing on topologies like RocketFuel, Hydrogen also has a dedicated module to infer realistic network link weights using Mixed Integer Programming. Hydrogen works in two stages. The first one analyses traceroutes and builds a topology, and the second one infers links weights for this topology. They work in tandem to generate more realistic and accurate network topologies.

Traceroutes

In network topology, traceroutes are the base unit of data measurements. Traceroute, and more specifically Paris-Traceroute [5], is a computer network diagnostic tool for displaying the route, or path, which packets follow within the network, from a source address to its destination. Paris-Traceroute is a newer version of the original traceroute tool, and addresses the previous inability to process correctly routers that employed load-balancing, which led to inaccurate and incomplete paths. Paris-Traceroute, on the other hand, does not have issues with load-balancing and obtains a more precise picture of the actual path taken by the packets. Traceroutes are the outputs of this tool. An example is represented in Listing 1. A traceroute is composed by a source address, from which the traceroute is collected, a destination address, and a list of hops, in order of traversal, from the source to the destination. A hop consists in an address, and usually some other information like the domain name or, in this case, the Round-trip delay time of each hop relative to the source address.

Traceroute works by sending packets with gradually increasing TTL value, starting with a TTL of one. When a router receives a packet, it decreases the TTL value, and drops the packet if this value reaches zero. When a router drops a packet, it sends an ICMP `time-exceeded` message back to the source. This way traceroute is able to lists the routers on the path from the source to the destination.

The original RocketFuel implementation executed traceroutes itself, through publicly available traceroute servers. This is no longer possible because these vantage points are in majority no longer available. Instead we rely on publicly available traceroute data on Caida's [6] repositories, the *Center for Applied Internet Data Analysis*. This institution collects traceroutes every 48 hours through 150 monitors spread across the world, to all announced (advertised by the different ISPs) /48 IPv6 prefixes. This represents more than 6 millions traceroutes, 60 millions hops, showing a snapshot of the state of the World Wide Web.

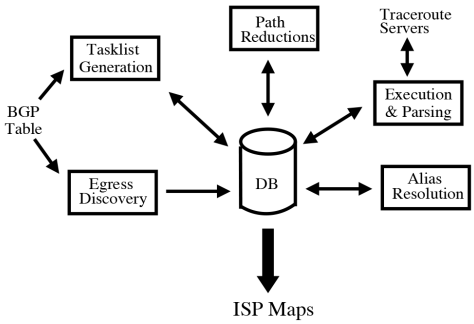


Figure 2: RocketFuel architecture. Source: [3]

RocketFuel

The most notable difference between Hydrogen and RocketFuel is that, while Hydrogen uses external data, RocketFuel needed to produce the traceroutes itself. RocketFuel's creators used a technique called *directed probing*, that used vantage point's BGP [7] tables to identify interesting prefixes that might "hide" behind the targeted ISP, and then probed from these vantage points only to interesting prefixes. For example, in the Listing 2, if we were mapping the AS number 7, a very interesting prefix to trace would be 4.5.0.0/16, because all AS-path traverse the mapped AS. Such a prefix is called *dependent*. Unfortunately, many of those vantage points are no longer available. It is important to note that inferring a topology with RocketFuel was a lengthy process. Executing a traceroute is not instantaneous. It takes in average 5 to 15 seconds for the trace to complete, sometimes substantially more. According to the paper, between 0.5 and 10 millions of traceroute were computed for one ISP with directed probing. With 750 public vantage points, and an average of 15 seconds to send a request to a traceroute server, execute the trace and get it back, it takes no less than 20 thousands seconds for one million traces, or a little more than 5 hours. Of course this result might be underestimating the real time, but when mapping a major ISP like Sprint, which requires approximately 10 millions traceroute, according the paper, RocketFuel could take days. This delay may introduce errors in the inference due to topology changes during the probing time. For example, a link might be down for a while and back up before the end of the inference, which could cause anomalies with the mapping.

RocketFuel also differs in its architecture. It used several asynchronous processes, articulated around a unique central database, to compute the different steps of the inference process(see Figure 2). This design was most likely created to deal with the duration of the collecting process. To reduce the time needed for the total computation, every process monitored the other ones, and changed the objectives based on the obtained results. The tasklist generation process generated a list of directed probes using BGP tables from vantage points. The dependent prefixes found in the tables were then replaced by their egress found by the egress discovery process. This way RocketFuel only traced to the egress, and not beyond. The path reduction process

Listing 2: Sample BGP table snippet with destination prefixes on the left and AS-paths on the right

```
1.2.3.0/24      13 4 2 5
                6 9 10 5
                11 7 5

4.5.0.0/16      3 7 8
                7 8
```

applied, like its name would suggest, the path reduction techniques, explained in section 1.2. The execution process collected the traceroutes, and finally alias resolution was done. Information about traceroutes executed in the past was used by the path reduction module to perform further reductions. This design induced some overhead due to the process synchronization or I/O operations with the database , and the "Database-as-IPC"[1] pattern was known as a bad example of IPC, and even an anti-pattern [8].

Motivations

Realistic network topologies are of great help to the network research community. They help to improve routing protocols [9], identify vulnerabilities and respond to attacks [10]. However, commercial network topologies are not publicly available because ISP's treat them as an industrial secret. RocketFuel is totally inadapted to the current age of telecommunications (due to, for example, the rise of IPv6, new versions of routing protocols and the disappearance of public traceroute servers). Our goal is to continue the work of RocketFuel and adapt it to these new technologies, while introducing a new functionality: link weights inferring. The first main contribution of this work are the creation of a new topology inference engine, based on the principles of RocketFuel but usable and adapted to current technologies. The second main contribution is the development of a fully fledged link weights inferring module, alongside a complete testing environment. In addition, we laid the foundation of an improved, more precise model for this task. Our whole code base has been made available at `https://bitbucket.org/gfeuillen/hydrogen`.

[1] Inter Process Communication

Chapter 1

Selecting Measurements

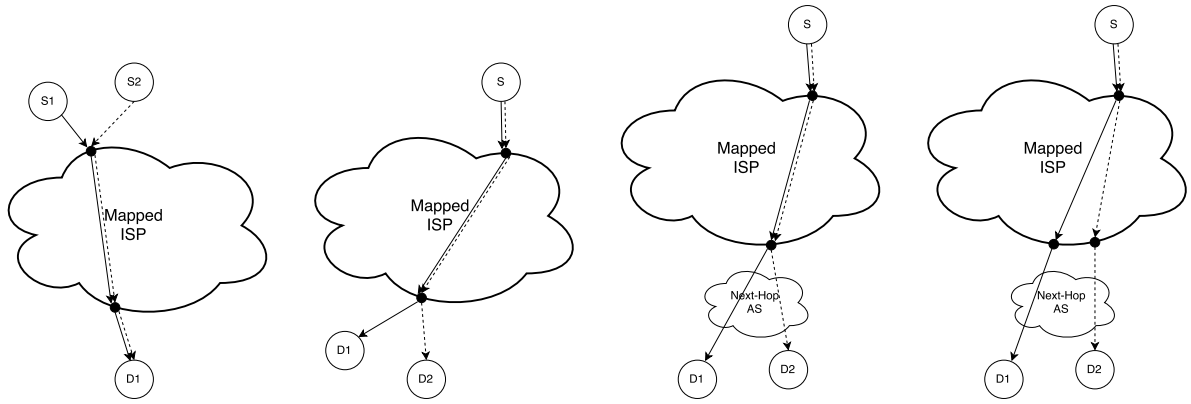
At the heart of the RocketFuel paper and tool lie path reduction techniques. They are the cornerstone of this work as well as the means to reduce the huge number of available traceroutes [11] to a more manageable amount. The objective is to discard all the unneeded data and thus reduce the number of selected traceroutes for both the graph and the weights inference, without loosing precision. But selecting the right measurements is not enough, we also needed to supplement them with additional information, such as routers DNS names. This chapter will first talk about a very inconvenient protocol, MPLS. Then describe the data selection techniques, show how they help us to reduce the huge amount of traceroutes to a more manageable size, and how we proceed to router identification. Finally some words about possible improvements.

1.1 The danger of MPLS

Multiprotocol Label Switching, a relatively new routing protocol that appeared in 2001 [12], is being deployed by operators on more and more networks for more than a decade. It is a routing protocol designed with performance in mind, replacing lengthy addresses and other routing information with short, 20-bit labels in order to shorten routing tables and lookup times. Furthermore it enables operators to easily perform traffic engineering on their networks [13]. It is considered as Multiprotocol because it can run of top of any other protocol, simply by stacking the MPLS header(s), called Label Stack Entries, or LSE, atop the packet header.

We will not explain the inner working of MPLS here, but it is extremely important to underline that the label definition is done by the network operators, and can be totally arbitrary. MPLS packets can follow any path they want, they are not limited to the fastest or shortest one. The impact of MPLS on topology measurements is not well known, but it is possible for some MPLS configurations to perturbate traceroute data, leading to false topology maps.

The biggest issue with MPLS is the `ttl-propagate` option, which decides of the behavior of the TTL field. As the reader may remember from the Introduction, Traceroute works by sending packets with increasing TTL values, and listening to ICMP `time-exceeded` responses. LSE also have a TTL, so let us call it LSE-TTL, and the original IP-TTL. MPLS routers send an ICMP `time-exceeded` message when the LSE-TTL expires exactly as would do a normal router when the TTL reaches zero. So traceroute should be able to follow the packets inside the Label Switched Path, a.k.a. LSP, without problems, at the sole condition that IP-TTL is copied to LSE-TTL when the packets enter the LSP. And this is the case when `ttl-propagate` is set. Unfortunately this is not always the case, and in 2012 researchers evaluated that 30% of traceroutes traverse MPLS tunnels, and 50% of those tunnels do not use `ttl-propagate` [14].



(a) Ingress reduction: only one traceroute must be selected when 2 sources share the same ingress router and destination

(b) Egress reduction: only one traceroute must be selected when 2 destination share the same source and egress router

(c) Next-Hop AS reduction: only one traceroute must be selected when 2 destinations share the same egress router and Next-Hop AS

(d) Next-Hop AS reduction issue: different paths inside mapped AS

Figure 1.1: Path Reductions

When `ttl-propagate` is not set, LSE-TTL is initialized with an arbirtray value, e.g. 255, and all the routers inside the LSP remain invisible for Traceroute. In topology data this yields links between routers that should not exist. A fingerprint technique was developed [14] to identify those invisible tunnels, but it is not yet deployed on caida, and thus the data we use contains these MPLS anomalies.

1.2 Path reductions

With up to 6 millions traceroutes available, it is clear that a large fraction of these will never cross the network that we are trying to map, and thus are useless for the topology inference. A first reduction is to select only the meaningful traceroutes, i.e. which cross the AS in at least one point with certainty. In order to achieve this, we keep only traceroutes that display at least one address exhibiting the prefix of the mapped network. All of those techniques are described in the RocketFuel paper [3].

1.2.1 Ingress Reduction

Paths taken by packets inside a network are usually destination specific. This means that two traceroutes having two different sources and a common destination enter the mapped network at the same point. They will most likely follow the same path inside the ISP. This is illustrated in figure 1.1a. This observation reveal that all traceroutes, sharing the same destination and *ingress router*[1] are redundant, thus only one is needed to characterize the route.

1.2.2 Egress Reduction

As for ingress reduction, we can deduce from figure 1.1b that two different traceroutes sharing the same source and *egress router*[2] follow the same path inside the mapped ISP, only one of them is necessary.

[1] First router encountered on the trace belonging to the mapped AS
[2] Last router on the trace belonging to the mapped AS

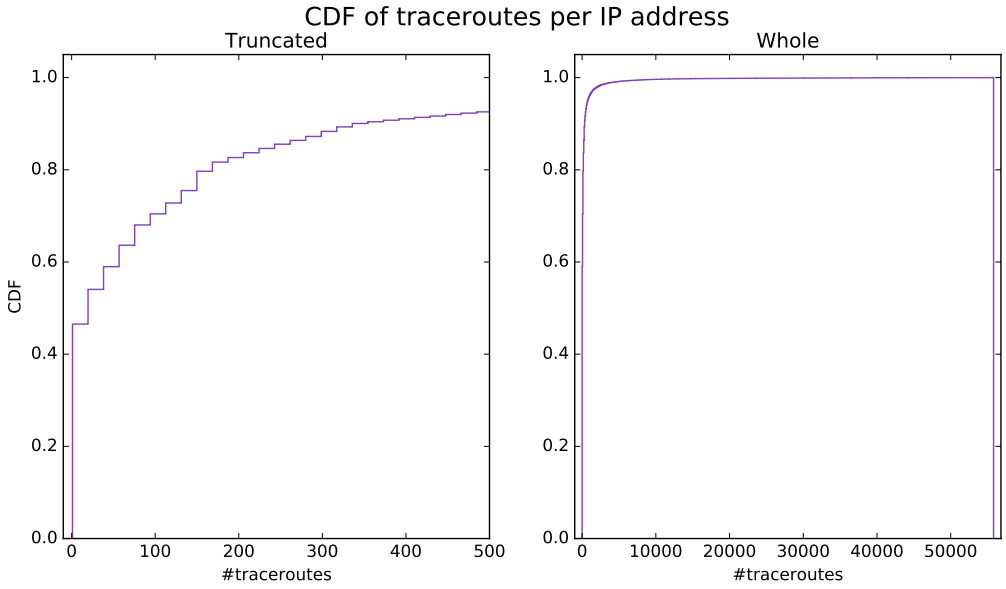


Figure 1.2: CDF of traceroutes per IP

1.2.3 Next-Hop AS (NHA)

NHA was presented as a better definition of egress reduction in the original paper [3]. Instead of using the egress router of different traceroutes as reduction factor, this time we use the Next-Hop AS, visited directly after the AS that we are mapping. Figure 1.1c illustrates this principle. Unlike egress reduction, which assumes there is only one egress router per destination prefix, traces sharing the same Next-Hop AS can exit the network in several places. The authors of RocketFuel [3] decided to use NHA instead of egress reduction because it eliminated even more traceroutes than the latter. Due to the possible difference in accuracy, we decided to discard Next-Hop AS and use the more straightforward and intuitive egress reduction. The reason was, as we can observe on Figure 1.1d, different paths can be taken inside the mapped AS even if the ingress router and NHA are the same.

1.2.4 Data

The data we used[3] counts 5 825 879 different traceroutes, counting a total of 112 427 different IPv6 addresses appearing 54 933 909 (source, destination and hops cumulated). Figure 1.2 shows de cumulative distribution function of traceroutes per IP address. We can see that 80% of IP addresses appear less than 200 times, but a small percentage of them is present in even 55 000 different traceroutes.
Figure 1.3 represents the CDF of traceroute per ISP. We count every unique IPv6/32 prefix as an individual ISP [15]. 50% of ISPs have between 160 and 200 traces passing through them. After this point, the CDF steadily grows, and biggest ISPs appear in almost 2 500 000 different traceroutes. This shows a great variety in ISP's with regard to their presence in the traceroutes. When inferring a topology, the more information we collect about the ISP through the traceroutes, the more accurate the generated router-level map will be. Consequently, the more traceroutes per ISP we collected, the better the accuracy of the inferred topology will be. As seen on figure 1.3, a large fraction of ISP possesses less than 200 traceroutes. This means that inferred topologies have a substantial chance to be inaccurate. This ituition is later confirmed in chapter 5

[3]The CAIDA UCSD IPv6 Topology Dataset - <17/05/2016 - 24/05/2016>, `http://www.caida.org/data/active/ipv6_allpref_topology_dataset.xml`

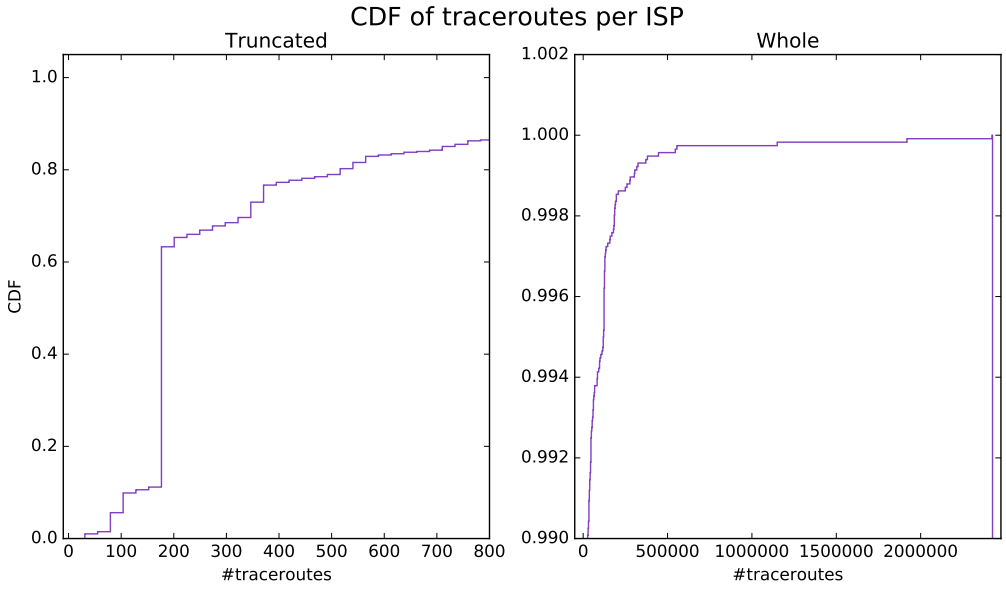


Figure 1.3: CDF of traceroutes per ISP

1.2.5 Reductions' effect on data

The importance of reduction is illustrated in figure 1.4, where a decrease of several orders in magnitude can be observed between the number of traceroutes before and after the reduction. For example, with the ISP Internet2 we started with whole traceroute collection: 5 825 879 traces. When considering only traces that contain Internet2's prefix, we decreased this number to 308 799, which is 5.3% of the original number. The next steps, including the elimination of duplicates and DNS (62 437), ingress reduction (44 535), and egress reduction took this number down to 372, or 0.005% of the original number.

1.3 Router identification and annotation

All aforementioned reduction techniques depends on the proper identification of the routers. We needed a positive way of telling whether a router belongs to a certain ISP. The first method we used was ASN identification. The ASN is an identification number which is easily retrieved for each router by the *whois*[16] command. But this approach proved to be pretty inaccurate due to the ambiguous ASN attribution of border gateaway routers [7].
A more accurate way was to rely on the routers prefixes and DNS names. Each ISP is registered under one /32 IPv6 prefix, and sometimes several smaller prefixes. This prefix is used to pre-filter the traceroutes in order to save off some work from the DNS resolver. We simply selected only the routes that contained at least one address containing the prefix, or list of prefixes, belonging to the targeted ISP.
Then we considered the DNS name resolution, which is essential for proper router identification and annotation. This resolution aimed to replace the router's IPv6 address with its DNS name which carries a lot more information than the address itself. For example, `2001:468:ff:209::2` resolves to `et-10-0-0.106.rtr.kans.net.internet2.edu`, which is a lot more readable. We could easily deduce that this node was a router (`rtr`) located in Kansas (`kans`) and belonging to the AS *Internet2* (`internet2.net`).

This approach is a lot more accurate than the ASN method, but depends on the manual

[4]The CAIDA UCSD IPv6 Topology Dataset - <19/05/2016-24/05/2016>, `http://www.caida.org/data/active/ipv6_allpref_topology_dataset.xml`.

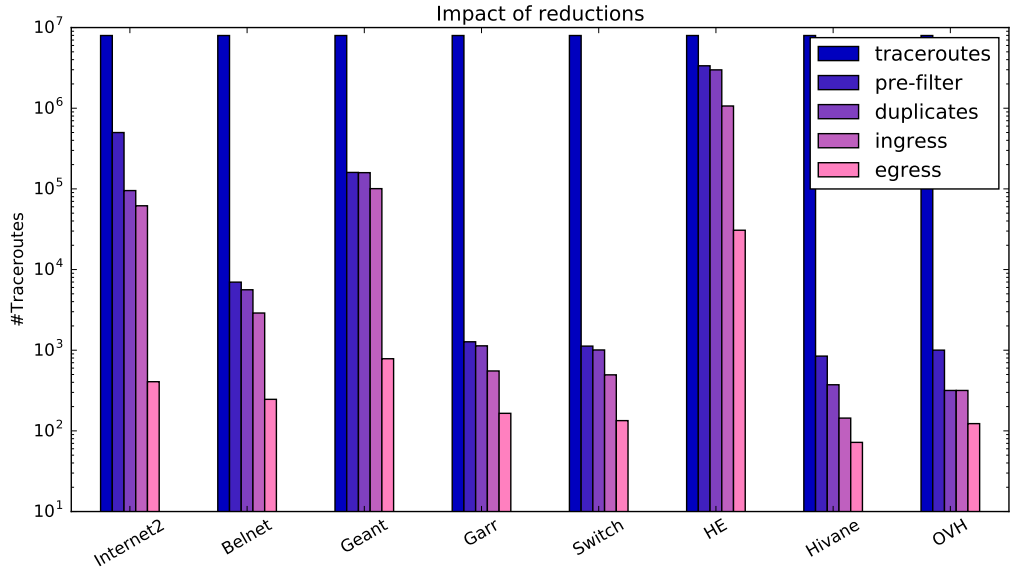


Figure 1.4: Impact of reductions [4]

analysis of each mapped ISP naming convention for proper identification.

1.4 A few words about alias resolution

The RocketFuel paper cited the existence of multiple interfaces for each router as an issue in map inference from traceroutes. This issue is fixed by determining which interfaces IP address belong to the same router. This is known as alias resolution [17]. This approach prevents the appearance of aliases in the map as redundant nodes and edges.

Alias resolution techniques are divided into three main categories.

- Fingerprint techniques: these techniques analyse routers responses to probe packets, and try to find recurring patterns in those response which act as the identifier of the router. Two routers with similar "fingerprints" are marked as aliases. Fingerprint techniques are usually accurate, but tend to be incomplete because many routers do not respond to probe packets.

- Analytical techniques: these techniques focus on a topology map, and try to discover aliases by comparing and finding similarities (same edges, same neighbors) between the nodes of the graph. They are less accurate than fingerprint techniques but they can work on non-responding routers.

- DNS techniques: when ASes uses some convention when assigning DNS names to router interfaces, that clearly defines the interface and the router, we can extract information from those names and use it to identify aliases. It does not require any probes, and thus works on unresponsive routers. But this method presents a major drawback because it requires a human to identify the naming convention used by the ISP, and write rules adapted to each naming convention, if one can be found.

We decided to use Caida's Speedtrap[18], a fingerprint based tool, included in Caida's Scamper, as our alias resolution engine. Speedtrap induces fragmented IPv6 responses from router aliases in a particular temporal pattern that produces distinguishable, per-router responses, that act as fingerprints which identify a router.

Listing 1.1: Two interfaces of a single router

```
et-10-0-0.106.rtr.kans.net.internet2.edu
et-5-0-0.109.rtr.kans.net.internet2.edu
```

However tests with alias resolution revealed that Speedtrap was identifying very few routers, often none, with several interfaces when mapping the networks. In addition it was using a substantial amount of time doing so.
We then turned to DNS techniques. An example of such a DNS alias resolution is listing 1.1. Both addresses shares the exact same domain name. The only difference is the first part, which suspiciously resemble an ethernet interface identifier. We could then assume that those both addresses were aliases of the same router. But this particular example, attached to a particular ISP using a particular convention could not be extended to other ISPs. Every new ISP requires significant human input, and this is ineffective on routers without an identifiable naming convention.

1.5 Further work

The biggest issue encountered while designing Hydrogen was MPLS. As explained in section 1.1, if networks operators decide to not propagate packets's TTL values inside their MPLS tunnels, packet traffic inside those tunnels becomes completely opaque to Traceroute. Opaque tunnels do not appears in Caida's traceroutes and, by extension, in the generated topologies. This translates into missing links and routers, and "jumps" between routers (long links that connect faraway routers). In the worst cases, entire topologies can be hidden from Traceroute, rendering our inference totally useless for studying some ISP's. This is notably the case for the Belgian provider *Proximus* [19] or French service and access provider *OVH*.

Unfortunately there are very few solutions to this major issue. Network operators cannot be forced to configure their MPLS infrastructures to use `ttl-propagate`. It is their right to protect their topologies, and nothing can be done about it. We can only hope that one day Caida or Paris-Traceroute will use techniques designed to reveal MPLS tunnels [14].

Chapter 2

Inference Engine Implementation

The original RocketFuel implementation [3] was designed in 2002, 14 years ago. Some key points of its implementation, like directed probing and manual traceroute collection from public vantage points, are no longer possible due to changes in the networking landscape, and the disappearance of public traceroute servers.
We decided to build Hydrogen Stage 1 from scratch. We did it by rethinking the dataflow of the application, and taking advantage of publicly accessible traceroute's databanks instead of relying on manual collection of traceroutes.
While RocketFuel was using a centralized dataflow, see Figure 2.1a, with a database serving as interprocess communication, and asynchronously running processes, we opted for a more streamlined, linear dataflow, see Figure 2.1b, with different steps running sequentially.

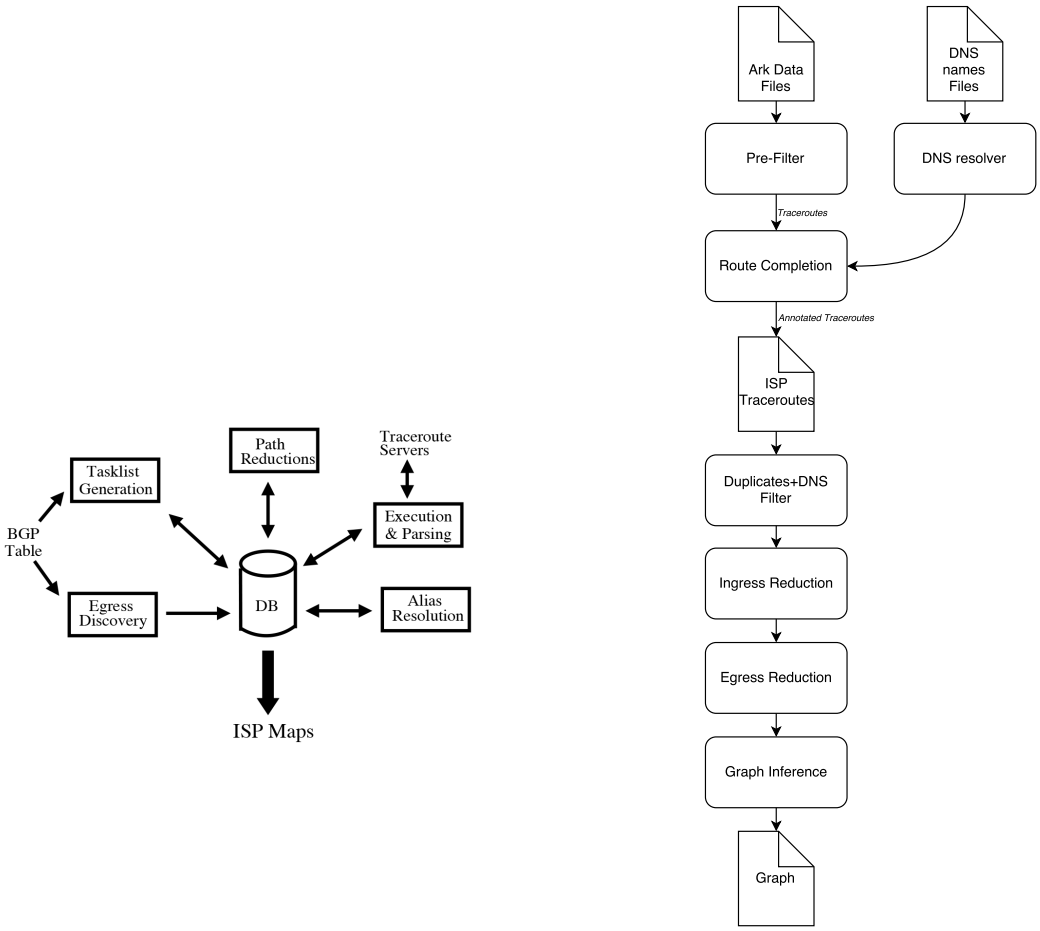
An overview of our implementation is presented in Figure 2.1 and is described in more detail in the following sections. Section 2.1 explains the inputs to our program and section 2.2 depicts the role of the pre-filter module. The DNS resolver is detailed in section 2.3, and section 2.4 specify the route completion block. Intermediary results are saved in the Intermediary ISP file as explained in section 2.5. The Duplicate and DNS filter (section 2.6) read the file, and then the reductions are achieved by both the Ingress filter and Egress filter, described in section 2.7. The Graph inference module which produces the graph is depicted in section 2.8.

2.1 Inputs

Raw traceroutes are the first input. They are provided by Caida and their Archipelago Measurement Infrastructure, a.k.a *Ark* [20] (counting 150 probes) and supplied in Caida's *warts* format. This data, organized into several dozens files, is first sorted by date of creation, and only the most recent files are used. Selected data is converted into parsable text by the Scamper tool [21], and then deserialized into traceroute objects.

2.2 Pre-filter

Only traceroutes that are related to our mapping were of interest to this study. However only a fraction of all traces contained information about our mapped ISP. We discard every traceroute which was not associated with our search. The pre-filter will checks if a traceroute met the right criteria. We checked every IPv6 address of every hop, and tried to match the ISP's IP prefix with the hop. If it matched at least once, the concerned traceroute was kept. Otherwise the trace was discarded.



(a) RocketFuel architecture. Source: [3]

(b) Hydrogen Stage 1 Architecture

Figure 2.1: Architecture: Hydrogen Stage 1 vs. RocketFuel

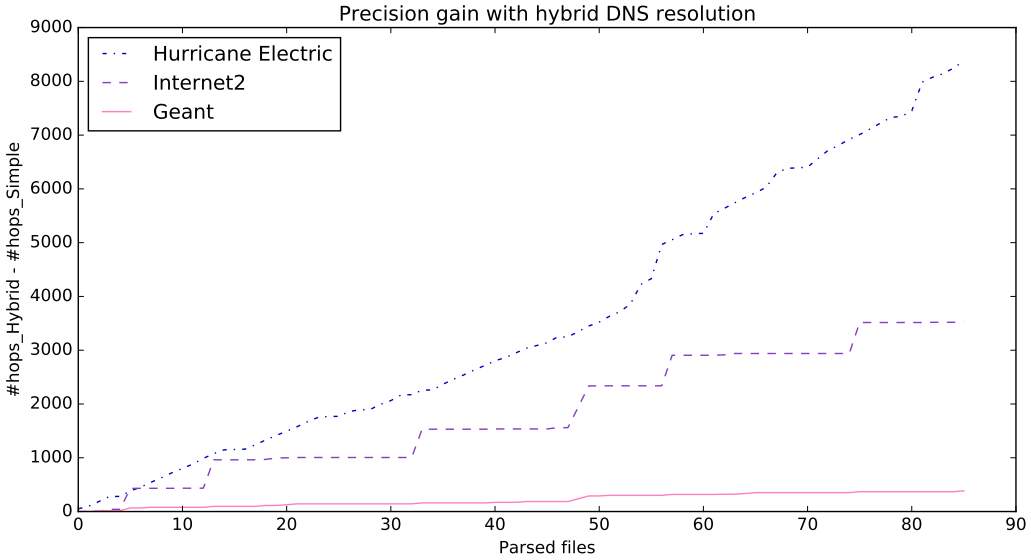


Figure 2.2: Hybrid DNS resolution

2.3 DNS resolver

Resolving the DNS names of the routers was an important task for the proper functioning of both the inference engine and the weights inference. The next steps heavily relied on an accurate annotations of the hops inside the traceroutes, and thus this step must be as accurate as possible, without taking too much time.

Two solution were possible. The first one is the local DNS resolution. It consists in resolving for every IP address locally, in order to get the most accurate, and up to date information as possible. However, local resolution take a substantial amount of time when resolving more than 100 thousands different addresses.

A second solution is the use of Caida's DNS names files. Those are files containing pairs of "IP address - DNS name", for addresses already resolved by Caida. These files consequently speed up the DNS resolver, but are sometimes outdated, and/or contains non-resolved addresses. When comparing local resolution and DNS files, we found out that the number of incorrect DNS resolutions was insignificant, often close to zero. On the other hand, the number of additional resolutions done by the local resolver (i.e. address that were not resolved in Caida's data, but that we could resolve locally) was slightly more substantial, as we can observe on figure 2.2. For example, more than 3 thousand additional addresses were resolved by local resolution for Internet2. This number even reach 9 thousand for a bigger ISP like Hurricane Electric.

To retain the performances of DNS names files, but achieve the accuracy of the local resolution, we used a hybrid method for DNS resolution. We use the DNS files to resolve every possible address, and the use local resolution for addresses that were not resolved with the files.

2.4 Route Completion

Even after the hybrid DNS resolution, some nodes remained nameless. This can prove to be very inconvenient, because those nodes would not be added to the topology. For example, if we encounter a route similar to $node_{ISP} \rightarrow node_{anonymous} \rightarrow node_{ISP}$ we will only register 2 nodes, and miss a part of the topology. We call those nodes *anonymous*.

An anonymous node, is a node displaying the ISP's prefix and located between two resolved ISP's nodes, for which DNS resolution did not succeed. This node should be, by rule, added to

the topology, together with the links it forms with its neighbors.
We attribute an *anonymous* DNS name for those nodes, inspired from the ISP's naming convention. For example, Internet2's anonymous nodes will receive the DNS `ANON.net.internet2.edu`. But this is not enough because this would mean that a topology can have only one anonymous node, which is not always the case. To differentiate between anonymous nodes, we added a suffix to this DNS name, from 0 to n, depending on this node address, prefix and the preceding node.

2.5 Intermediary ISP traceroutes

Since the time required to parse all the data and resolve the DNS names is greater than the time needed for the inference, we decided to copy the results of the previous stages to an intermediary file, so that ulterior inferences on the same ISP (for testing purposes) can be computed much faster. This file follows the same organization as Caida's files, so the same parser can be used on both.

2.6 Duplicates and DNS filter

In the pre-filter, we eliminated all traceroutes not containing the mapped AS's prefix. We take this step a little further by discarding all traceroutes that do not contain the mapped AS' name in their DNS name. This is because an ISP prefix contains not only the ISP's routers, but also customers' machine and networks. DNS resolving for all those networks does not identify these as belonging to the ISP, and thus we can easily distinguish, after the DNS, resolution which traceroutes contains our mapped ISP's routers, and which simply concerns customers. We discard the latter.

2.7 Ingress and egress reduction

Those two filters will apply the two path reduction techniques explained in Section 1.2. Their goal is to prune the set of traceroutes, by rejecting redundant data without sacrificing accuracy. For each reduction, we maintain a set of ingress-destination (resp. source-egress) pair. For each traceroute in the input set, we check if the set already contains an entry for this combination. If it was not the case, we added this new combination into the set, and put the traceroute into a new set containing only reduced traceroutes. The pseudocode for the ingress reduction is presented in Algorithm 1. Egress reduction algorithm is very similar.

```
   input  : set: set of traceroutes
   output: Reduced set of traceroutes
 1 newSet ← new Set;
 2 store ← new Set;
 3 for traceroute ← set do
 4 |   ingress ← traceroute.getIngress;
 5 |   destination ← traceroute.getDestination;
 6 |   if !store.contains((ingress, destination)) then
 7 |   |   store ← store + (ingress, destination);
 8 |   |   newSet ← newSet + traceroute;
 9 |   end
10 |   return newSet
11 end
```

Algorithm 1: Ingress Reduction Pseudocode

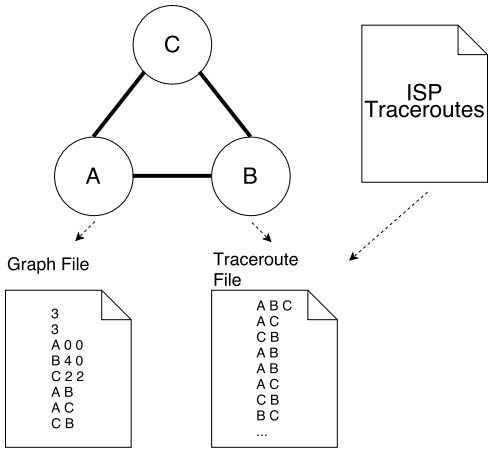


Figure 2.3: Graph is converted into 2 files before Stage 2

2.8 Graph inference

Once the traceroutes have been reduced we can start the graph inference. To represent a network, we use a undirected graph where nodes represent routers, and edges links between routers. The algorithm is pretty simple, we loop through all hops of all traceroutes, adding to the graph nodes that belongs to the mapped ISP. We also use a variable to store the last traversed node. If this node belongs to the ISP, we add a new edge *last — current*. We also use *GraphStream* [22], a graph visualization library, to easily draw and visualize the topologies produced by this process.

```
   input  : set: set of traceroutes, dns: dns names of our ISP
   output: Network topology graph
 1 last ← empty;
 2 graph ← new Graph;
 3 for traceroute ← set do
 4 |   last ← empty;
 5 |   for hop ← traceroute.hops do
 6 |   |   if hop.belongs(dns) and !graph.contains(hop) then
 7 |   |   |   graph.addNode(hop);
 8 |   |   end
 9 |   |   if last.belongs(dns) and !graph.contains(last — hop) then
10 |   |   |   graph.addEdge(last — hop);
11 |   |   end
12 |   |   last ← hop;
13 |   end
14 end
```

Algorithm 2: Graph Inference Pseudocode

2.9 Further Work

Hydrogen acts more as a proof of concept than a fully finalized tool, and some improvements can be done to the architecture and modules to achieve more accuracy and better performance. In its actual state, Hydrogen exchanges set of traceroutes between the different modules. An improvement could be the use of a streams of traceroutes manipulated concurrently by the modules, to reduce the time required to generate a topology.
Another area where improvements could be done is the alias resolution. At the current time it must be manually tuned to accommodate each ISP's naming conventions. The process could

be made more autonomous by the use of more advanced techniques like regular expressions or machine learning.

Chapter 3

Inferring link weights

The aim of the second stage is to assign weights to edges based on the observed traceroutes. In this context, we present the solution provided by [1]. We hereby call this part of the second stage the *Solver* and formalize its operating in terms of inputs and outputs (represented in figure 3.1):

- **Inputs**
  - A topology (or Graph) denoted by G.
  - Traceroutes collected for this topology.
- **Output**: a weight for each edge present in the input traceroutes.

The graph G is a directed graph, completely described by the tuple $< V, E >$ where V are the vertices and E are the edges. An edge is simply described by a pair of vertices (S, D) where S is the source and D is the destination (it can also be written SD for simplification). We chose to represent topologies as directed graphs to respect that links do not necessarily have symmetric weights.

The traceroutes are paths in the graph. They can be described as a succession of nodes or edges. Traceroutes are executed on the network in a given period. During this period each path might appear several times in the traceroutes.

The rest of this chapter continues as follows: we present the first *Solver*, we then introduce the simulation environment surrounding the *Solver* (which is necessary for testing), we describe the quality measures used to assess its performances, we then provide results combining both the simulation environment and the quality measures and finally we outline further improvements that could be brought.

3.1 Design choices

Before diving into the model we spent some time on design choices that were made. As it was mentioned, we used a constraint based approach as described in [1]. This posed the question of

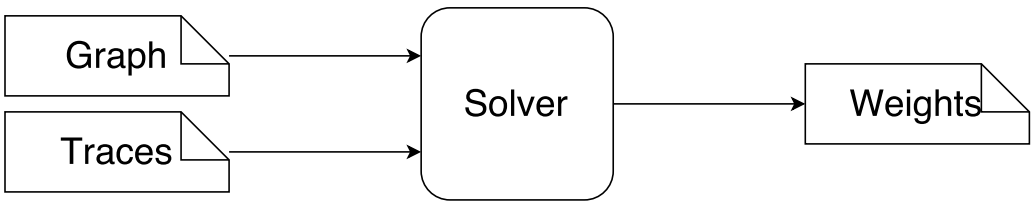


Figure 3.1: Representation of the solver

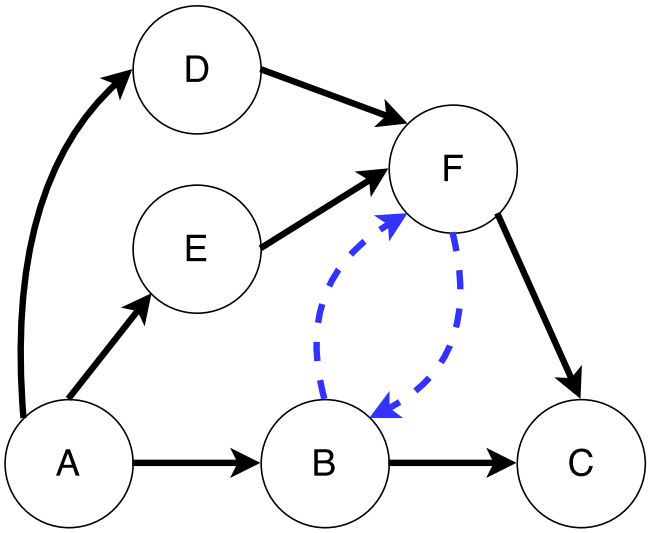


Figure 3.2: Example graph

the type of solver which can support our model. Two technologies are commonly be: constraint programming (*CP*) or mixed integer programming (*MIP*). While both of these options could answer our needs, we chose a *MIP* solver called Gurobi [23]. Without going into the details of each option let us say that MIP solvers perform better on models that can be described using (almost) only linear relationships. As the reader will see across this chapter, this is our case.

We chose to use integers only which placed our model in the category of Integer Programming. This might be limiting in term of computational time because it makes it more difficult for the solver to find the optimal solution. This choice is motivated by the fact that weights in networks running OSPF [24] or IS-IS as intra-domain protocols are conventionally integers. While our inferred weights might always differ from a factor compared to the real ones, the fact that both of them are integers allows for easier comparison (because they both take discrete values). This choice also turned out to be useful when facing problems with the weights returned by Gurobi. Indeed Gurobi acts as a black box and while we ask for integers it might sometimes return values being close to integers (e.g. 1.9999 instead of 2). When this happens, we can simply round the values to the nearest integer.

3.2 First model

Throughout this section we will always use the same example graph shown in figure 3.2. As the reader can see in this graph, the links connecting B to F are shown with dashed lines because these are backup links. These backup links will only be used in the last subsection (3.2.4). In the first three subsections (3.2.1, 3.2.2 and 3.2.3), we consider these links as not being observed in the traceroutes, thus they can not appear in the topology nor can the *Solver* make inference about their weights. In addition we defined the following conventions :

- An edge is written as SD where S is the source and D the destination of the edge.
- The weight of an edge is written as w_{SD}. The weights are positive integers, these constraints are expressed as :

$$\forall S \; \forall D, \; w_{SD} \in \mathbb{Z} \tag{3.1}$$

$$\forall S \; \forall D, \; w_{SD} \geq 1 \tag{3.2}$$

- A path is written P_{SiD} where S is the source, D the destination and i represents a (possibly empty) sequence of intermediate nodes describing the path.

(1)	A B
(2)	A E
(3)	A D
(4)	D F
(5)	E F
(6)	F C
(7)	B C

Table 3.1: Assumed observed traceroutes

- The weight of a path is the sum of the weights of each edge composing the path and is written $W(P_{SiD})$

Before defining the model, an assumption has to be made concerning the traceroutes:

Assumption 1. *If a traceroute between the two points A and B is observed, then at that given time, the shortest path going from A to B was the one observed.*

Although this assumption might not always be true in networks running protocols such as MPLS, where the path chosen might be arbitrary, most common intra domain protocols (OSPF, IS-IS) use link weights to compute shortest paths.

Thus our model will express two types of constraints directly deriving from assumption 1:

- If there are multiple path observed between two nodes, then their weight must be equals.
- The observed path between two points is the one with the smallest weight.

In order to formalize these constraints we made the following assumptions on the traceroutes observed :

Assumption 2. *We have observed every shortest paths between every pair of nodes.*

Assumption 3. *Every observed shortest path is indeed the shortest path used in the network*

Under these assumptions, we can formally describe these constraints as :

$$\forall i \, st. \, P_{SiD} \in Traces, \forall i' \neq i \, st. \, P_{Si'D} \in Traces \Rightarrow W(P_{SiD}) = W(P_{Si'D}) \tag{3.3}$$

And:

$$\forall i \, st. \, P_{SiD} \in Traces, \forall i' \neq i \, st. \, P_{Si'D} \notin Traces \Rightarrow W(P_{SiD}) < W(P_{Si'D}) \tag{3.4}$$

The assumptions 2 and 3 that led us to equations 3.3 and 3.4 are strong assumptions that can not be hold when facing real traceroutes. They are both discarded respectively in subsection 3.2.3 and 3.2.4.

In the rest of this chapter we will assume that the edge linking two vertices can always be considered as a shortest path between them (if such an edge exists). We emulate this in our sandbox graph (figure 3.2) by observing every edge linking two nodes as described in table 3.1.

3.2.1 Naive approach

Equations 3.3 are straightforward to set up: it is a simple matter of adding the weight of every path connecting two nodes and set them as equal. From the implementation point of view, a MIP solver like Gurobi does not need a constraint for every pair of paths connecting similar

(8)	A B C
(9)	A E F

Table 3.2: Additional observed traceroutes

(1)	$W(P_{ABC})$	$<$	$W(P_{AEFC})$
(2)	$W(P_{ABC})$	$<$	$W(P_{ADFC})$
(3)	$W(P_{AEF})$	$<$	$W(P_{ADF})$

Table 3.3: Constraints for the naive approach

nodes, It is sufficient to state that one of them (the first one in our implementation) is equal to every other.

A naive approach would consist in setting up constraints by following equation 3.4 literally. Considering the observed traceroutes described in table 3.2 we get the equations described in table 3.3.

Document [1] makes the following statement: "The basic solution leads to an exponential number of constraints because a constraint is set up for every simple alternate path. This makes it intractable for large networks." This is not desirable for two reasons. First it would have an enormously negative impact on the solving process. Secondly, we might spend more time setting up constraints than actually solving the problem. This issue is addressed in the next subsection.

3.2.2 A polynomial approach

Based on the criterion set out in [1] we are able to get a polynomial number of constraints. We first state and explain the criterion and then use it on our sandbox example. For the remainder of this section, we will only consider the reduced subset of constraints.

Reducing the number of constraints

As described in [1]:

Let P_{SiD} be the observed path between nodes S and D (and thus considered as the shortest path) and $P_{Si'D}$ be an alternate path.

A path P_{SiD} can be decomposed in two paths P_{SjI} and P_{IkD} where I is an intermediate vertex along the path ($I \in i$) (if such an intermediate vertex exists). We denote this decomposition as $P_{SiD} = P_{SjI} \cdot P_{IkD}$.

Then the constraint $W(P_{SiD}) < W(P_{Si'D})$ is redundant if at least one of the two following conditions is true for any intermediate vertex I in the alternate path $P_{Si'D} = P_{Sj'I} \cdot P_{Ik'D}$:

1. The sub-path $P_{Sj'I}$ is not observed.
2. The sub-path $P_{Ik'D}$ is not observed.

Let us fully explain the first case (represented in figure 3.3): let P_{SiD} be the shortest path between S and D and $P_{Si'D}$ be an alternate path. If the first condition is respected, then it means that there exist an alternate sub path $P_{Sj''I}$ (dashed arrow in figure 3.3) such that $W(P_{Sj''I}) < W(P_{Sj'I})$. We can thus describe a second alternate path $P_{Si''D} = P_{Sj''I} \cdot P_{Ik'D}$. If we had used the naive approach described in Subsection 3.2.1, then the following constraints would have been set up:

(1)	$W(P_{SiD})$	$<$	$W(P_{Si'D})$
(2)	$W(P_{SiD})$	$<$	$W(P_{Si''D})$
(3)	$W(P_{Sj''I})$	$<$	$W(P_{Sj'I})$

Where we can clearly observe that constraint (1) becomes useless since it is implied by the other two. Let us obtain (1) from (2) by using (3), more specifically:

$$
\begin{array}{rcccll}
W(P_{SiD}) < W(P_{Si''D}) & \Leftrightarrow & W(P_{SiD}) & < & W(P_{Sj''I}) + W(P_{Ik'D}) & \text{By decomposition} \\
& \Leftrightarrow & W(P_{SiD}) & < & W(P_{Sj'I}) + W(P_{Ik'D}) & \text{Using constraint (3)} \\
& \Leftrightarrow & W(P_{SiD}) & < & W(P_{Si'D}) & \text{By composition}
\end{array}
$$

A similar proof can easily be done for the second case. This exercise is left to the reader.

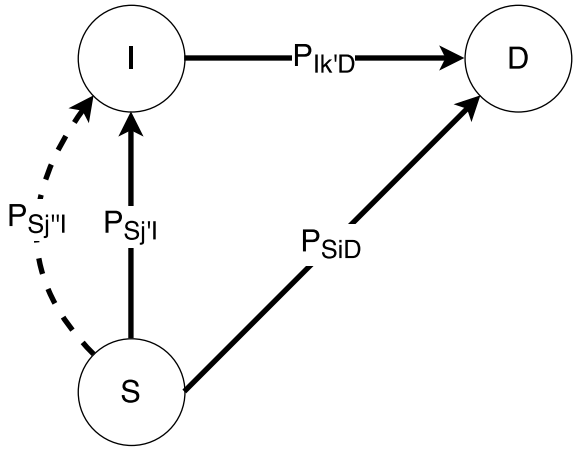


Figure 3.3: Representation of the criterion's first case

Thus we get a polynomial number of constraints. For a graph containing n vertices, the number of source and destination couples is equal to n^2, for each intermediate node a constraint might be set up hence resulting in $\mathcal{O}(n^3)$ constraints in the worst case.

On our sandbox example

Using the same traces as the one used in the naive model (see table 3.2), the constraint number 2 (see table 3.3) is not useful any more.

3.2.3 Not enough traceroutes

A first problem that might arise when using traceroutes is that they might not contain every shortest paths. For example not every router in the topology could be used as a vantage point.

Let us take a look at our sandbox example (figure 3.2). We observed the traces AD and DF but not ADF. Since our model uses strict inequalities, the path ADF will not be part of the shortest paths. But what if it actually was one and we just were not able to observe the ADF trace ? Using non-strict equalities '$\leq$' instead of '$<$' solves this issue by letting the chance for ADF to be predicted as a shortest path.

3.2.4 Inaccurate traceroutes

What happens if there are inaccurate traceroutes (traces observed that are not shortest paths) fed to the input ? This could happen if a link was down or the network was in some transient state. The answer to the question is then straightforward: our model might be unresolvable. We present this problem on our sandbox graph, then explain how to cope with such shortcomings, show that the issue is resolved on our sandbox example and give a short analysis about the meaning of error terms.

Unresolvable system

Let us assume that our network has been in three different states:

1. A consistent state where traces presented in table 3.1 have been observed.
2. A first transient state where link BC was down. For packets to go from B to C, the backup link BF was used thus leading to observe inconsistent traceroutes such as BFC.
3. A second transient state where link FC was down. For packets to go from F to C, the backup link FB was used thus leading to observe inconsistent traceroutes such as FBC.

We represent only the additional traces in table 3.4 and the associated system in table 3.5.

(8)	B F C
(9)	F B C

Table 3.4: Traces observed

(1)	$W(P_{BC})$	=	$W(P_{BFC})$
(2)	$W(P_{FC})$	=	$W(P_{FBC})$

Table 3.5: Constraints in case of shortcomings in traceroutes

Let us quickly show that this system is unsolvable. Constraints (1) and (2) can be rewritten as:

$$w_{BC} = w_{BF} + w_{FC} \tag{3.5}$$

$$w_{FC} = w_{FB} + w_{BC} \tag{3.6}$$

Injecting w_{BC} from equation 3.5 into equation 3.6 and simplifying common terms yields:

$$w_{BF} + w_{FB} = 0 \tag{3.7}$$

Since our weights are positive integers as stated in equations 3.2 and 3.1 then $w_{BF}+w_{FB} \geq 2$. It is of course impossible to satisfy both this equation and 3.7.

The error terms

To overcome those shortcomings in the data, [1] introduced error terms for each traceroutes observed. "The main idea is to associate error variables with constraints, and minimize the weighted sum of errors. In our approach, we use an error variable per observed path [...] using the number of times the path was observed as the weight" [1]. This error term represents the excess weight of the associated path (thus errors are greater or equal to 0) and "the error variables of non-shortest paths have a non-zero value in the solution" [1]. For our inferred weights to correspond as much as possible to the input traces, we minimize the weighted sum of errors.

We thus define for each path P_{SiD} an error $E(P_{SiD})$. We also use the same notations as the ones used in the previous chapter, the number of time a path P_{SiD} is observed in the traceroutes is denoted by $c(P_{SiD})$. We can now describe the objective function as:

$$\min \sum_{P_{SiD} \in Traces} c(P_{SiD})E(P_{SiD}) \tag{3.8}$$

As we will see in the last subsection, this objective function favours paths observed several times over paths only observed a few times. We think that this is a valid hypothesis since our traceroutes, if measured over a long period, should contains more correct traceroutes than incorrect ones.

A resolvable system

Using the previously stated errors terms, the constraints described in table 3.5 become the ones described in table 3.6.

$$\begin{array}{lrcl} (1) & W(P_{BC}) - E(P_{BC}) & = & W(P_{BFC}) - E(P_{BFC}) \\ (2) & W(P_{FC}) - E(P_{FC}) & = & W(P_{FBC}) - E(P_{FBC}) \end{array}$$

Table 3.6: Constraints with error terms

We can now rewrite equation 3.7 using constraints described in table 3.6 as :

$$w_{FB} + w_{BF} = E(P_{BFC}) + E(P_{FBC}) - E(P_{BC}) - E(P_{FC}) \tag{3.9}$$

As it can be seen in equation 3.9, the system is now solvable. Since we are minimizing the sum of errors which have to be greater or equal to 0, several assignments are possible. Let us take a quick look at the objective function described in equation 3.8 where other terms than E_{BFC} and E_{FBC} are excluded because they can be set to 0. Thus, in our example graph, the objective becomes:

$$\min c(P_{BFC})E(P_{BFC}) + c(P_{FBC})E(P_{FBC}) \tag{3.10}$$

One of the two remaining error terms has to take a non-zero value. Equation 3.9 yields that $E(P_{BFC}) + E(P_{FBC}) \geq 2$ (weights are strictly positive integers). Since this is a minimizing problem the solver chooses to set $E(P_{BFC}) + E(P_{FBC}) = 2$. Until now, we did not bound $c(P_{BFC})$ and $c(P_{FBC})$ to values but it appears clearly (by looking at the objective function, equation 3.10) that if $c(P_{BFC}) > c(P_{FBC})$ then $E(P_{FBC})$ will be set to 2 and inversely. As stated, this model favours as shortest path, traces that have been observed several times over traceroutes only observed a few times.

What happens if $c(P_{BFC}) = c(P_{FBC})$? Gurobi [23] will arrange itself such that $E(P_{BFC}) + E(P_{FBC}) = 2$. It might even choose to assign both errors to 1. This is because neither of the two paths can be favoured and it might be a problem. Indeed, for our model to be as consistent as possible with the observed traceroutes, we would prefer to discard one path and not both. While this could addressed up by changing the objective function such that it takes the number of paths discarded into account, we can reasonably assume that over a long period of measurement, this case should not arise.

Meaning of the error term

What does exactly represents the error term ? We already stated that this term represents the excess weight of the path associated to it. This has several implications:

1. The adjusted weight (the weight diminished by its error) of every path observed between a source and a destination is equal to the weight of a shortest path between these two nodes (due to inequality constraints)

2. All adjusted weights between a source and a destination are equal (due to equality constraints).

3. If the error term is equal to 0 then the weight of the path is equal to its adjusted weight and thus is equal to the weight of a shortest path between its source and destination. In other words, this means that the observed path can be considered as a shortest path.

3.2.5 Summary

Using the polynomial approach described in subsection 3.2.2 along with the shortcomings solutions described in subsections 3.2.3 and 3.2.4 we can describe how constraints are set up in a general manner. We first show those constraints and then explain our implementation.

The equality constraints look like:

$$W(P_{SiD}) - E(P_{SiD}) = W(P_{Si'D}) - E(P_{Si'D}) \tag{3.11}$$

The inequalities look like:

$$W(P_{SiD}) - E(P_{SiD}) \leq W(P_{Si'I}) - E(P_{Si'I}) + W(P_{Ii''D}) - E(P_{Ii''D}) \tag{3.12}$$

There might be several equality constraints for a given path, e.g. if three paths (P_{SiD},$P_{Si'D}$ and $P_{Si''D}$) are observed between S and D. In such a case we do not need to set up equality constraints for every pair of paths. Indeed, in our example, setting it for $(P_{SiD}, P_{Si'D})$ and $(P_{SiD}, P_{Si''D})$ is enough since the third constraint can be derived from the other two. This is what we did in our implementation to avoid spending too much time setting up constraints.

A similar observation can be made for inequalities. Let us assume that in addition to the paths described in equation 3.12 there is a second path $P_{Si''I}$ observed between S and I. Then there exists an equality constraint linking $P_{Si'I}$ and $P_{Si''I}$ and it becomes redundant to set up the following constraint:

$$W(P_{SiD}) - E(P_{SiD}) \leq W(P_{Si''I}) - E(P_{Si''I}) + W(P_{Ii''D}) - E(P_{Ii''D})$$

Indeed this constraint is simply obtained using the equality constraint along with equation 3.12. This observation can easily be extended for every path present in equation 3.12.

3.3 Testing environment

The testing environment is what surrounds the solver. It provides manageable inputs and allows for easy testing. We designed it as modular as possible and is described in the following subsections. We first present an overview of the simulation environment, then spend more time describing each of its components and finally explain how tests are conducted.

3.3.1 Overview

An overview of the Testing environment is represented in figure 3.4. We begin by introducing every block (rounded square on the figure) and then come back to the functioning of the whole environment.

The first aim of the testing environment is to provide manageable inputs. The component responsible to provide those inputs is called the *Simulator*. Without going into the details, the *Simulator* creates traceroutes from a given graph, knowing its weights a priori. The *Simulator* is described in subsection 3.3.2.

The more information about the network's routing, the better the predicted weights. In our problem, more information means more traceroutes. Thus a pre-process can be applied to the traces produced by the *Simulator* to enhance the information it contains. The *Pre-processor* is in charge of this enhancement and is described in subsection 3.3.3.

Now that every component has been briefly described, we present the testing environment in figure 3.4. As first input, a graph and its weights from which we can simulate traceroutes using the *Simulator*, then these traceroutes go trough the *Pre-processor* before entering the solver(along with the topology). At the end the *Solver* generates predicted weights that can be compared with the input to assess their correctness. This last comparison step may be quiet difficult as we will see in section 3.4.

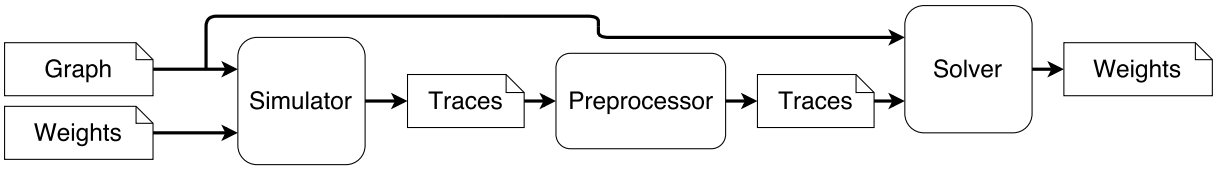


Figure 3.4: Representation of the simulation environment

From an implementation point of view, we adopted this box-design approach to allow for unit testing and modularity. It is indeed very simple to add or even change a step in the process, just change or add a box. Once again, for modularity reasons and data re-usability, we chose to keep every intermediate state in a file.

3.3.2 Simulator

The *Simulator* is shown in figure 3.5. We detail its operations in this Subsection.

Let us assume that we know the graph's weights a priori. Our graph is no longer described as a couple $< V, E >$ but as a triplet $< V, E, w >$ where w is a function calculating the weight of an edge. Then, the problem of creating inputs for the solver is simply reduced to the creation of traceroutes from a given graph. Given the assumption that observed traceroutes were of the shortest paths, the creation of traceroutes is solved by using any shortest path algorithm. The traceroutes created using the *Simulator* should be similar to the one we actually use from CAIDA. We thus identified the following desired behaviours:

- The collection of traceroutes is done over a long period.
- Network failures can happen at any time.
- Not all nodes can be used to collect traceroutes (i.e. the number of source nodes is limited)
- We might not be able to collect traceroutes going to every router (i.e. the number of destination node is limited)

The first behaviour is implemented by collecting several samplings (i.e. computing shortest paths) of the same network. The number of samplings is a parameter given to the *Simulator* as it can be seen on figure 3.5. The last three desired behaviours are described as events that can happen between two samplings. We identified four events:

- Link down: meaning the edge described between two nodes can not be used any more.
- Node down: meaning the node can not be used any more, it is removed from the topology
- Source node unavailable: no traceroute can be completed going from the node (but the node may appear in traceroutes).
- Destination node unavailable: no traceroute can be completed going to the node (but the node may appear in traceroutes).

Link and node down are common failures that can happen in every network and for various reasons. Nodes source/destination unavailable reflect the fact that some nodes in the network just can not be accessed directly, they may appear in traceroutes but can not be used as vantage points. These events are parameters of the *Simulator* as it can be seen on figure 3.5.

From an implementation point of view, we created an abstract class called Event to represent the events described above. Anything can happen in an event, it just needs a trigger, a duration

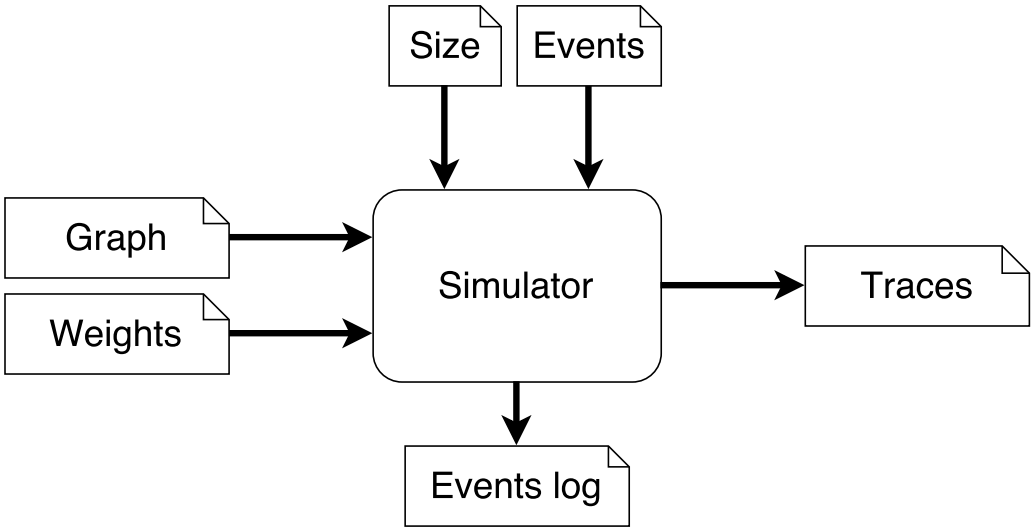


Figure 3.5: Representation of the *Simulator*

and an action to perform and can be added to the simulator as long as it extends our abstract class. Details of the implementation are left in the code but at this point our simulator just needs to keep a list of events, look for triggers, apply/revert failures and do the sampling. Our implementation also provides a log: for each sampling executed on the network we state which events were used. The log is thus an output of the *Simulator* as represented in 3.5.

3.3.3 Pre-Process

As described subsection 3.3.2, we usually do not have all the information about the network's routing. Aside from failures, we do not have every traceroutes for every pair of nodes of the topology. But we might be able to mitigate this shortcoming by using the following two techniques identified by [1]:

1. *Optimal substructure*: "any subset of a shortest path is shortest" [1]
2. *Reverse path symmetry*: "a consequence of both directions of a link having equal weight, the reverse path of a shortest path is shortest." [1]

While [1] states these two techniques as confirmed, we decided to test their impact on our results. We see them as hypothesis that can be made on networks. From an implementation point of view, we added a new box called the *Pre-processor* (represented in figure 3.6). This box simply takes two boolean inputs for each hypothesis and places itself between the simulator and the solver. If the *sub opt* boolean is set to true, then for every path contained in the traces produced by the simulator, we add every sub-path to the traces. If the *reverse opt* is set to true then we create the reversed path for each path and add it to the traces. Both of them can be combined such that if they are both set to true, then not only we create every sub path but we also create the reversed sub path and add it to the traces.

3.3.4 Summary

In this summary, we describe the parameters that can be tuned to run a test. These parameters will be used to characterize every test in the following section:

- **The topology**: the input graph.

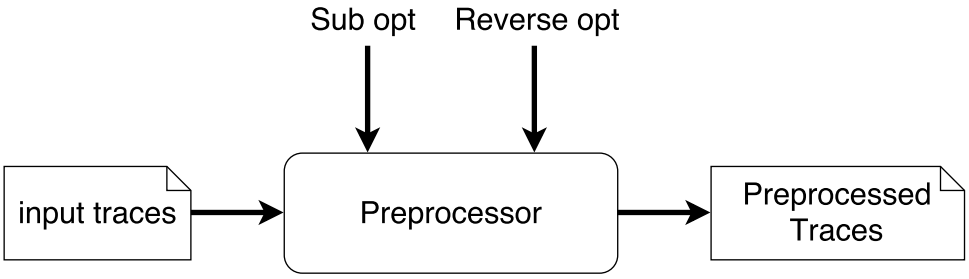


Figure 3.6: Representation of the *Pre-processor*

- **Sampling size**: the number of samples to be done on the graph to simulate the traceroutes.
- **Events**: the events added to the simulator.
- **Sub opt**: boolean to use the *Optimal substructure* technique.
- **Reverse opt**: boolean to use the *Reverse path symmetry* technique.

3.4 Quality measures

In order to understand the quality measures described in this section we introduce the following notations.

1. P_{in} describes the paths (traceroutes) given as input of the solver.
2. Each path $P \in P_{in}$ can be observed more than once, we denote that counter by $c(P)$.
3. P_{out} describes the paths computed using the weights produced by the solver.
4. P_{in} and P_{out} are stored in paths matrices M_{in} and M_{out}.
5. The path matrix M stores at indices (S, D) every path having vertex S as its source and vertex D as its destination.
6. We consider two paths as being equal if they share the exact same sequence of vertices or edges.
7. P_{in} and P_{out} can be considered as sets such that common set operations can be used.

Paths matrices described in notations 4. and 5. are not only a convenient way of storing paths but also help us define all quality measures described in this section. These notations are also consistent with the next chapter (Chapter 4) and are the exact data structure used in our implementation.

In this section, we first discuss the quality measures from [1], we then review them and finally introduce our own quality measures.

3.4.1 Quality measures in [1]

The fraction of observed paths that had least cost

"Each path is weighted by the number of times it was seen." [1]

Using our previously defined notations, this quality measure can be described as follows :

$$\frac{\sum_{P \in P_{in} \cap P_{out}} c(P)}{\sum_{P \in P_{in}} c(P)} \tag{3.13}$$

The fraction of dominant paths that had least cost

"A dominant path is the most commonly seen path between two nodes." [1] and "All dominant paths are weighted equally" [1].

We first define the dominant path between source S and destination D in a path matrix M as:

$$dominant(M, S, D) = \underset{P \in M(S,D)}{argmax}\, c(P) \tag{3.14}$$

And we can then describe the measure as:

$$\frac{\sum_{S \in V, D \in V} \mid dominant(M_{in}, S, D) \cap dominant(M_{out}, S, D) \mid}{\sum_{S \in V, D \in V} \mid dominant(M_{in}, S, D) \mid} \tag{3.15}$$

The fraction of node-pairs between which the routing was fully characterized

"To capture how well routing between node-pairs is characterized, we partition them into the following classes:

- *full*: each least cost path between the pair was seen;
- *partial*: some, but not all, least cost paths were seen;
- *none*: no least cost path was seen.

" [1]

We only describe the equations for the fully characterized routing. Similar equations can be easily obtained for *partial* and *none*.

Firstly we describe the "fully characterized" function for a source S and destination D by comparing the paths matrices M_{in} and M_{out}

$$full(S, D) = \begin{cases} 1 & \text{if } M_{in}(S, D) = M_{out}(S, D) \\ 0 & \text{if not} \end{cases} \tag{3.16}$$

We can now express the measurement as:

$$\frac{\sum_{S \in V, D \in V} full(S, D)}{\mid V \mid^2} \tag{3.17}$$

3.4.2 Review of measures from [1]

We have two key points to raise on the quality measures. Firstly, all three quality measures described above share a common bias. All of them use the input of the solver to compare their results. This means, from a general point of view, that if we can create a solver that over-fits the data given an input, then our model will be evaluated as very good while actually performing poorly. Secondly we would like to be able to understand the limits of our model, thus it becomes essential to be able to manage the inputs fed to the solver.

The need for manageable inputs is already answered by the *Simulator* described in subsection 3.3.2. We thus need to use new, non-biased quality measures. These are described in the following subsections.

		Assigned class	
		Positive	Negative
Actual class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Table 3.7: Outcome of classification into positive and negative classes

3.4.3 New quality measures

Inferred link weights, these are really difficult to evaluate because they will always differ from at least one order of magnitude compared to the real ones. Instead we chose to look at the shortest paths predicted.

If we assumed an input graph, we can create every shortest path between every pair of vertices and store them. Then, we can create manageable inputs (see section 3.3.2) and infer weights using the solver. To be consistent with previous notations, we created a new variable P_{real} as the shortest paths computed using the "real" weights. Remember that P_{out} are the shortest paths computed using the weights output by the solver.

At this point, our aim is to compare P_{real} and P_{out}. Since computing shortest paths can be seen as extracting, from every possible paths, the ones with the least weight, we use common measures in information retrieval : precision, recall and $F_\beta - score$. We first recapitulate what those measure are in information retrieval, then translate them into usable measures for our problem and finally give them meaning depending on their value.

"Precision and recall are the measures used in the information retrieval domain to measure how well an information retrieval system retrieves the relevant documents requested by a user. The measures are defined as follows:

- **Precision** : Total number of documents retrieved that are relevant/Total number of documents that are retrieved.
- **Recall** : Total number of documents retrieved that are relevant/Total number of relevant documents in the database

" [25]

The $F_\beta - score$ is simply a combination of precision and recall, as we will see in equation 3.20. "Let relevant documents be positive examples and irrelevant documents, negative examples." [25] Both the precision and the recall can be defined using the confusion matrix described in table 3.7 :

- **Recall** = True positives/Total number of actual positives = $\frac{TP}{TP+FN}$
- **Precision** = True positives/Total number of positives predicted = $\frac{TP}{TP+FP}$

Making a parallel between classic information retrieval and our problem we get that the true positives (TP) is the number of paths that are both in P_{real} and P_{out} while the total number of actual shortest paths ($TP + FN$) is given by the size of P_{real} and the total number of shortest paths predicted ($TP + FP$) is given by the size of P_{out}. We thus get the following equations for our measures :

$$Precision = \frac{|P_{out} \cap P_{real}|}{|P_{out}|} \tag{3.18}$$

$$Recall = \frac{|P_{out} \cap P_{real}|}{|P_{real}|} \tag{3.19}$$

$$F_\beta = (1 + \beta^2)\frac{precision * recall}{(\beta^2 precision) + recall} \tag{3.20}$$

AS	name	Routers	Links	Symmetry
	synth50	50	138	0.04
	synth100	100	286	0.99
1221	Telstra (au)	104	151	1.0
1755	Ebone (eu)	87	161	1.0
3967	Exodus (us)	79	147	1.0

Table 3.8: ISPs with the number of backbone routers and links and the percentage of symmetric links

Interpretations

The interpretations of these measurements are the following:

The precision (equation 3.18) gives a quality measure of the predictions. The precision can not be lower than 0 nor greater than 1 ($0 \leq precision \leq 1$) and is of course closer to 1 for a greater precision, meaning every shortest path predicted is an actual shortest path used.

The recall (equation 3.19) gives a measure of the completeness of the predictions, a similar analysis as the one for the prediction can be trivially made.

Finally the F_β (see equation 3.20) is a combination of the two where the parameter β can be tuned. A $\beta < 1$ emphasizes the precision while $\beta > 1$ favours the recall. We chose a commonly used measure called the F1-score where $\beta = 1$, in which neither precision nor recall is favoured.

What about the quality measures used in [1] ?

First, let us note that, the counter associated to each traceroute can not be used any more, or is equal to one in every case. From our point of view, this reflects the fact that we do not give more importance to one shortest path or another. This is one property that makes our measure non-biased.

Thus, the precision (equation 3.18) only differs from the fraction of observed paths (equation 3.13) in that it uses P_{real} and not P_{in} to compare the predictions.

The measurement using dominant paths (equation 3.15) has no sense any more since all counters are equal to one.

The last measurement looking at fully characterized routing (equation 3.17) shares attributes with both recall (equation 3.19) and precision. It firstly looks at completeness by considering every pair of nodes and then at the precision by looking at the predictions themselves. We kept this measure because of its duality and expressiveness compared to the F_β measure but use P_{real} instead of P_{in}.

3.5 Evaluation of the first model

In this section we present the results obtained using the first model. Each test was conducted on several ISPs whose key informations such as name, number of routers, number of links and percentage of symmetric links are shown in table 3.8. The percentage of symmetric links is the fraction of links for which the weight is the same in both directions.

Each of the following subsections contains a test to assess the performance of the solver and understand its limits. They are organized in the form of questions/answers.

3.5.1 Is the model correct?

Let us remember that non strict inequalities '$\leq$' were set under the assumption that not every shortest paths of the network was indeed observed (see subsection 3.2.3). If it could be possible

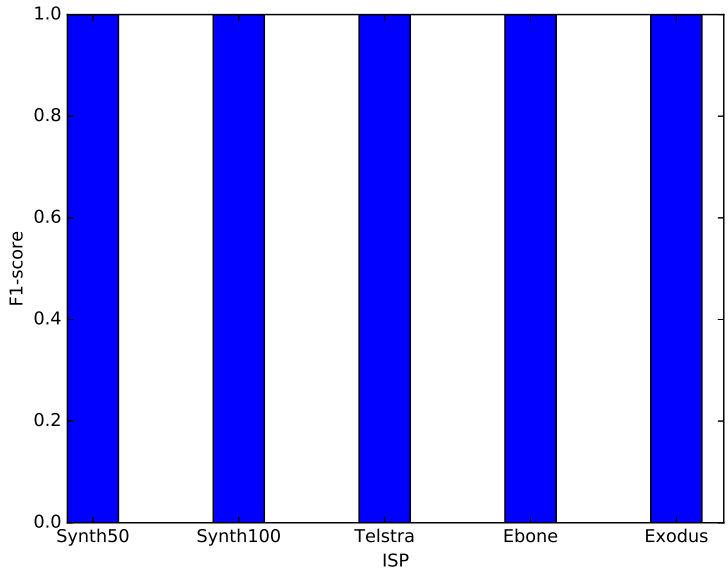


Figure 3.7: $F_1 - scores$ for the different ISPs when all shortest paths are given in input and strict inequalities are used

to get every shortest path (and it is, because we know the full topology a priori) then the use of '$<$' instead of '$\leq$' should produce weights characterizing perfectly the routing. If we could use these weights to compute every shortest paths, then we should observe a $F_1 - score = 1$ meaning that not only every shortest path predicted is indeed a shortest path ($precision = 1$) but also that we predicted every "real" shortest path ($recall = 1$). Note that no pre-process is applied to the data since we already have every shortest path.

We do this for each ISP presented in table 3.8. The associated $F_1 - scores$ are shown in figure 3.7. As the reader can see, they are all equal to 1 meaning that our model is both correct as well as properly implemented.

While this experiment might seem trivial, it was important to assess the correctness of both the model and our implementation. In addition, the weights obtained in this "perfect information" context will be used later in Subsection 3.5.4.

3.5.2 Are *Optimal substructure* and *Reverse path symmetry* correct/useful hypotheses?

We answer this question in six steps:

1. We first present the tests ran on each ISP.
2. We then present the quality measures obtained (as described in section 3.4) when no hypothesis is made. This data will be used as reference to quantify the impact of the hypotheses.
3. We conduct the same tests using *Optimal substructure*
4. We conduct the same tests using *Reverse path symmetry*
5. We conduct the same tests using both *Optimal substructure* and *Reverse path symmetry*.
6. We conclude by answering the question

	Optimal substructure	*Reverse path symmetry*
Step 2	False	False
Step 3	True	False
Step 4	False	True
Step 5	True	True

Table 3.9: Values taken for *Optimal substructure* and *Reverse path symmetry* for the firsts experiments

Values taken by *Optimal substructure* and *Reverse path symmetry* for steps 2. to 5. are shown in table 3.9.

Step 1: Experiment

In this experiment we progressively narrow the number of available traceroutes. Let us remember that both hypotheses described in Subsection 3.3.3 are made to overcome the fact that not all traceroutes might be observed. In this context we created a test where we limited the number of available source nodes that can be used for traceroutes as well as the number of destination nodes to which traceroutes can be done. We thus defined the following test by specifying its components as described in subsection 3.3.4: **The topology**, **Sub opt** and **Reverse opt** depends on the the topology studied as well as on the step (see table 3.9).

- **Sampling size**: 1
- **Events**:
  - One *Source nodes unreachable* event starting at time 0 and ending at time 1 (thus lasting during the whole sampling). This event is created by specifying which nodes can not be used as source nodes for traceroutes
  - One *Destination nodes unreachable* similar to the previous event but for destination node.

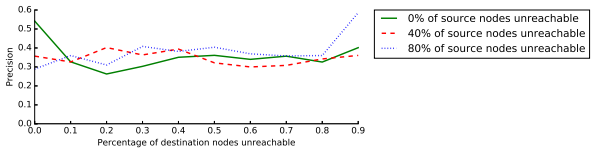
In order to measure the impact of the hypotheses we repeated this test for a greater percentage of source nodes unreachable (from 0.0 to 0.9 by step of 0.1) and destination nodes unreachable, thus making 100 tests. Each of the hundreds test is repeated 5 times for data significance. In the figures shown in this thesis, we chose to show the results for only 30 of the 100 tests done for a topology for obvious readability reasons. Moreover, we only show the results that are useful to describe the behaviour of the *Solver*, more specifically we only show results for ISPs Synth50 and the Telstra network. Comparable figures for other topologies can be found in Appendix A.1.

Step 2: no hypothesis

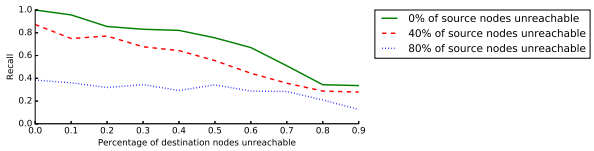
In figures 3.8a, 3.8b, 3.8c and 3.8d, we represented the precision, the recall, the $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP Synth50 when no hypothesis is made. Similar results for ISP Telstra are presented in 3.9a, 3.9b, 3.9c and 3.9d.

These figures will be re-used to compare the impact of hypotheses on the results. We can already make some observations:

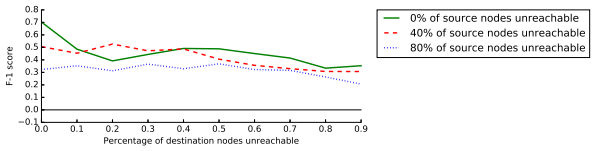
- For ISP *synth50*, we observe that the precision (answering "How good our predictions are ?", see figure 3.8a) stays quiet low for all tests (never higher than 0.6).



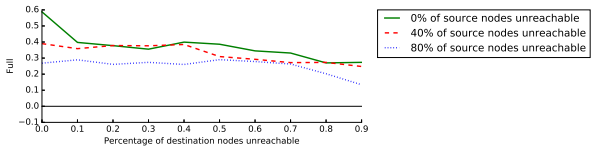
(a) Precision



(b) Recall

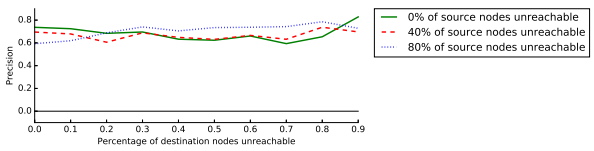


(c) $F_1 - score$

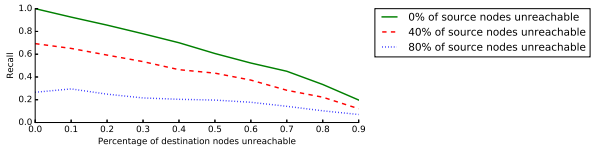


(d) The fraction of node-pairs between which the routing was fully characterized

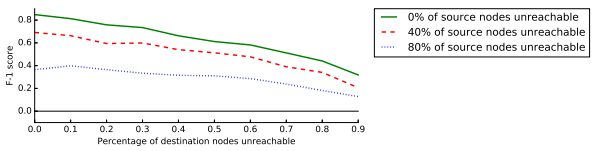
Figure 3.8: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP Synth50 when no hypothesis is made



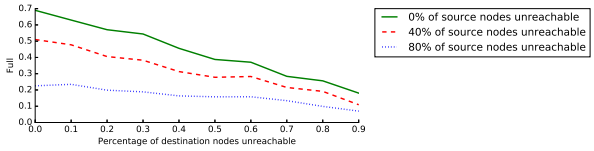
(a) Precision



(b) Recall



(c) $F_1 - score$



(d) The fraction of node-pairs between which the routing was fully characterized

Figure 3.9: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP rf1221 when no hypothesis is made

- For both ISPs, the precision (see figures 3.8a and 3.9a) increases when the number of available source nodes decreases (the precision is higher when only 0.8 of the source nodes are reachable), this is even more true for in the *Telstra* network where the precision also increases with the number of destination nodes unavailable. This observation led us to a second question that is presented and answered in Subsection 3.5.3. Indeed we investigate what causes the precision to decrease when the available information increases.

- The recall (answering "how complete my results are ?") decreases for both ISPs (figures 3.8b and 3.9b) when the number of source/destination nodes unreachable increases. This was expected, indeed the less information about the network, the less our results should be complete.

- As we explained the meaning of every measure in Subsection 3.4.3, we stated that *the fraction of node-pairs between which the routing was fully characterized* and the $F_1 score$ were somehow similar in that they both look at the precision and the completeness of the predictions. As we can see by comparing figures 3.8d and 3.9d respectively with figures 3.8c and 3.9c, they do behave similarly and take almost the same values, or at least in the same range. In the following subsections we thus do show these two measures since their value can be derived from the precision and the recall[1].

Step 3: when *Optimal substructure* is used

In order to quantify the impact of the *Optimal substructure* on the precision and the recall we measure the gain by computing the difference between the results when the hypothesis is made and the ones when no hypotheses is made (described in previous Subsection 3.5.2). Thus a positive (resp. negative) gain means that the hypothesis has a positive (resp. negative) effect on the routing characterization. For the ease of interpretation, we show the 0 value by a black horizontal line on each figure.

The gains in precision and recall for ISP *Synth50* are presented in figures 3.10a and 3.10b. Similar results for the *Telstra* network are shown in figures 3.11a and 3.11b.

We should expect a positive impact on both measures since this hypothesis brings more information on the network. As we will see this might not always be the case.

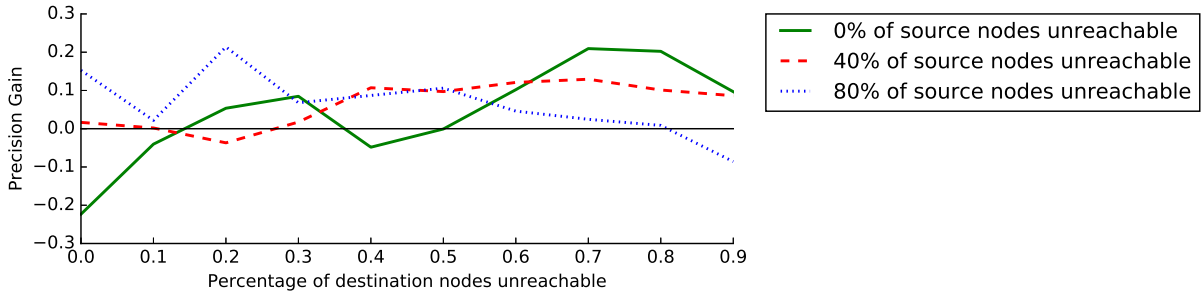
Before exploring the results, let us recall that we look at the gain in term of precision and recall. Since the recall measures the completeness of the results, it should increase with the number of available traceroutes. It is also very important to remark that we can only have a positive gain in recall if there is missing information. Therefore, in figures 3.10b and 3.11b, the recall gain begins at 0. We can make the following observations:

- The gain in precision for both ISPs (figures 3.10a and 3.11a) are almost always positive. Once again, as we already observed that in the previous step, sometimes when more information is given, it has a negative impact on the precision, meaning that our predicted shortest paths are less accurate. This issue will be studied in Subsection 3.5.3.

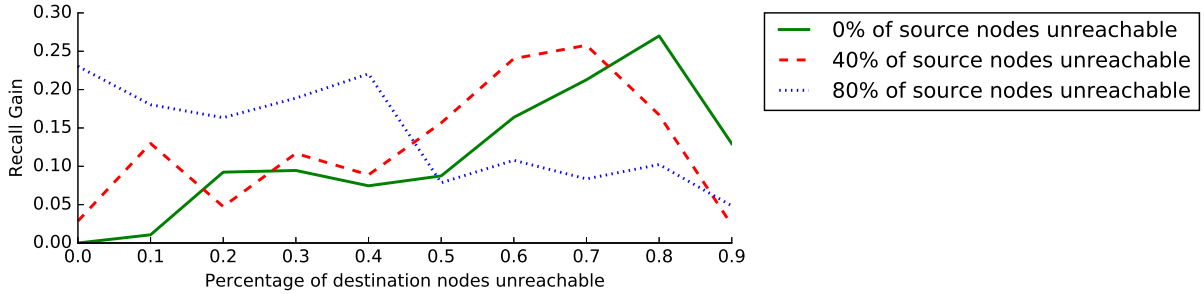
- The *Optimal substructure* hypothesis (almost) always have a positive impact on the recall. This is even more the case for the ISP Synth50 where the recall is improved by 25% in some cases.

- The fact that this hypothesis sometimes has a negative impact on the recall of the *Telstra* network (see figure 3.11b) is due to randomness in the experiment. Indeed we do not use the same sets of source/destination nodes unreachable from one experiment to another (i.e. when no hypothesis is made versus when *Optimal substructure* is used), we keep choosing them at random. Thus it can happen that the sets of source/destination nodes

[1] All the measure are still presented in Appendix A.1 for the unconvinced reader



(a) Gain in precision



(b) Gain in recall

Figure 3.10: Gain in precision and recall for ISP *Synth50* when *Optimal substructure* is used in comparison when no hypothesis is made

unreachable in this experiment led to lower performance even when *Optimal substructure* is used. Furthermore, we can clearly see that the negative effect never exceed 1%.

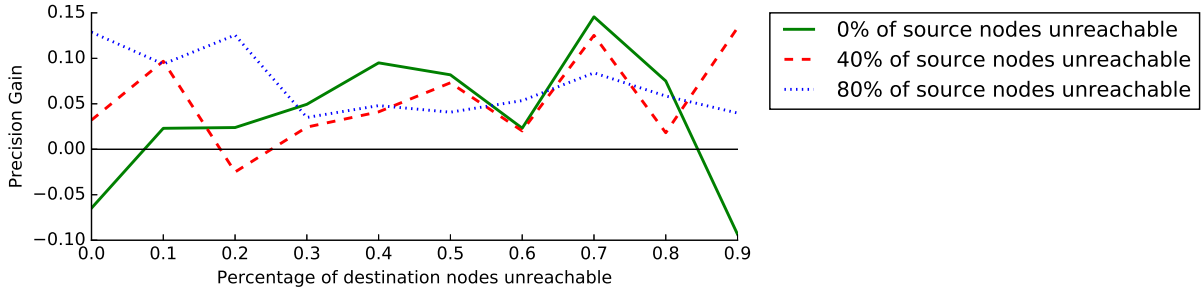
Step 4: when *Reverse path symmetry* is used

Table 3.8 shows the percentage of symmetric paths inside each ISP. Thus the *Reverse path symmetry* hypothesis should produce better results for every ISP except Synth50 (since this ISP only has 3% of symmetric links).

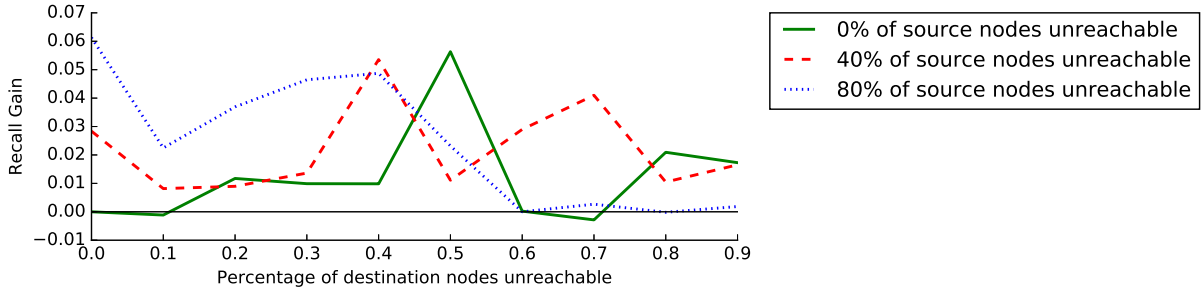
In figures 3.12a and 3.12b are shown the precision and recall gain for ISP Synth50. Similar results for the *Telstra* network are presented in figures 3.13a and 3.13b.

We can make the following observations:

- The gain in precision for the ISP Synth50 is always negative but this negative effect decreases with the number of source/nodes reachable. This is simply because as stated previously, the ISP Synth50 doesn't actually use *Reverse path symmetry*.
- The gain in recall for ISP Synth50 is nevertheless positive. We believe this is because the *Solver* predicts weights in a way that a lot a shortest path are predicted and in that enormous quantity of predicted shortest paths, some of them must actually be shortest paths thus yielding a good recall. This possibility is explored in Subsection 3.5.3.
- The gain in both precision and recall is positive for the *Telstra* network (figures 3.13a and 3.13b). We observed that it even goes up to 80% when 100% of the source nodes are available. The fact that this hypothesis has such a positive impact on the prediction for the *Telstra* network while only a gain of 40% at maximum for the Synth50 ISP can be observed comfort us in the idea that the gain for the Synth50 ISP actually hides a bigger problem.

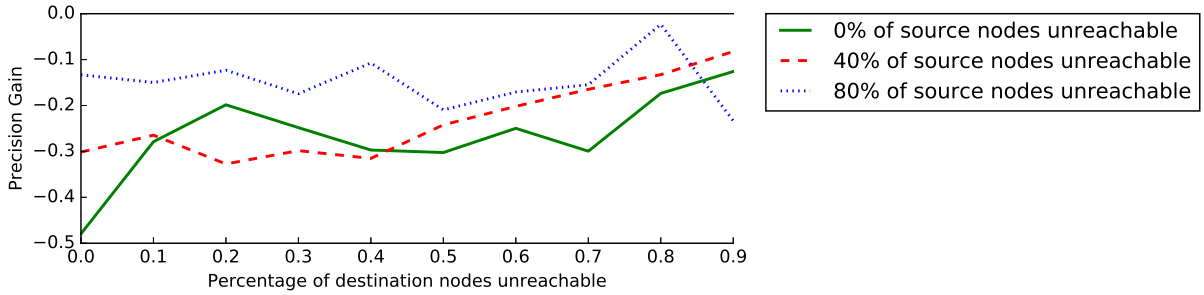


(a) Gain in precision

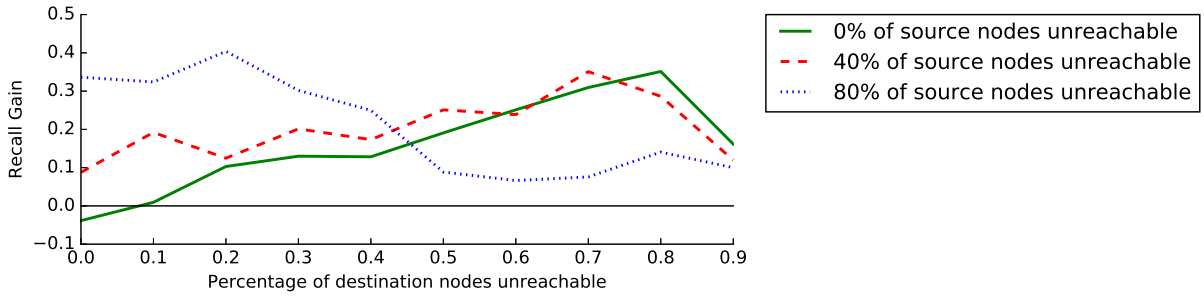


(b) Gain in recall

Figure 3.11: Gain in precision and recall for the *Telstra* network when *Optimal substructure* is used in comparison when no hypothesis is made

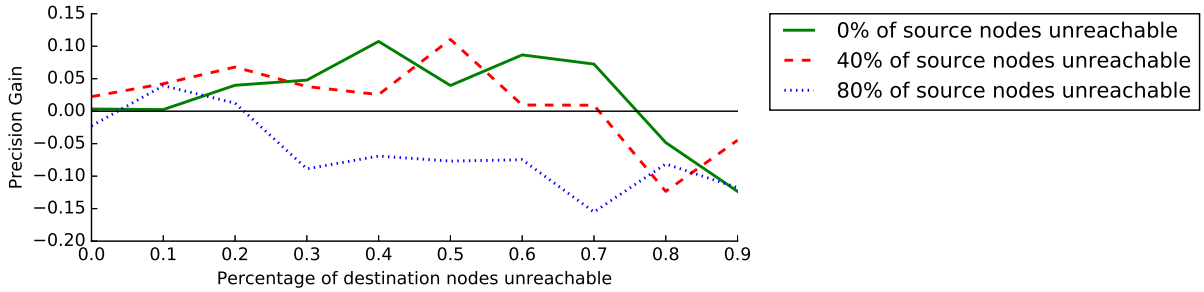


(a) Gain in precision

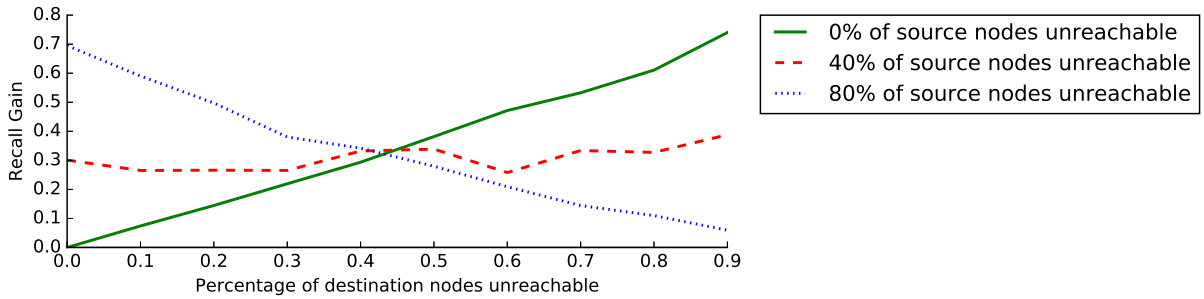


(b) Gain in recall

Figure 3.12: Gain in precision and recall ISP Synth50 when *Reverse path symmetry* is used in comparison when no hypothesis is made



(a) Precision



(b) Recall

Figure 3.13: Gain in precision and recall for the *Teslstra* network when *Reverse path symmetry* is used in comparison when no hypothesis is made

Step 5: when both hypotheses are used

In this step we show the gain in precision and recall for both ISPs: Synth50 (figures 3.14a and 3.14b) and the *Telstra* network (figures 3.15a and 3.15b). In these results we expect to see the impact of both *Optimal substructure* and *Reverse path symmetry* combined (described in the two previous steps). This is indeed the case, as we can still see, the gain in precision is still negative for the ISP Synth50 and only goes up to 20% for the *Telstra* network.

Step 6: answer to the question

The first hypothesis (*Optimal substructure*) is in any case useful. For every ISP presented in table 3.8, this hypothesis has improved every quality measure.

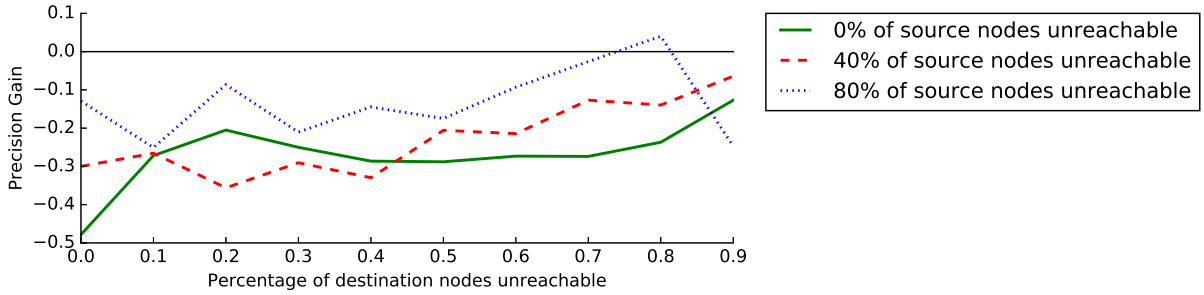
Concerning the second hypothesis (*Reverse path symmetry*), only results of the Synth50 topology did not improved as expected.

Since the only topologies for which there is a negative effect are the synthetic ones we conclude that both hypotheses can reasonably be used when facing real, unknown topologies. In the further subsections, every result is thus obtained using these two hypothesis.

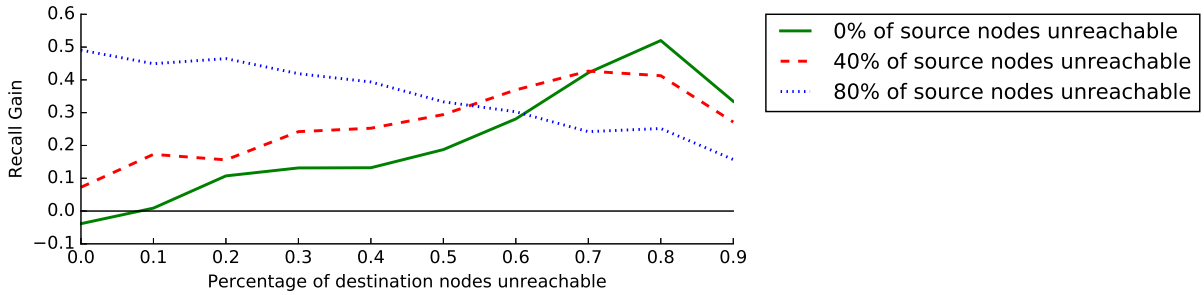
3.5.3 What causes bad precisions?

Remember that in the second experiment (Subsection 3.5.2), we diagnosed a quiet low precision even when all information (every shortest path) was available. This was even more true for the SYnth 50 ISP. In this subsection we therefore look at what causes such bad performances on the Synth50 ISP. Let us look at figure 3.16 where we aggregated the paths predicted in two categories :

- Correctly predicted: the shortest path predicted using the weights of the solver is an actual shortest path of the initial topology with its initial weights ($|P_{real} \cap P_{out}|$ using notations developed in Section 3.4).

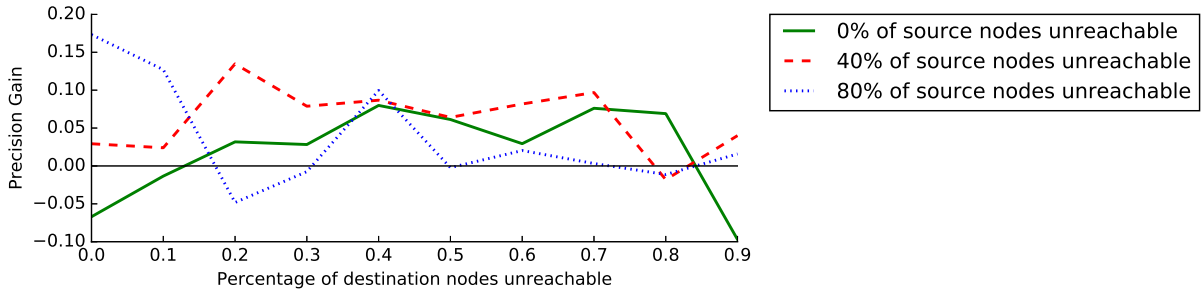


(a) Gain in precision

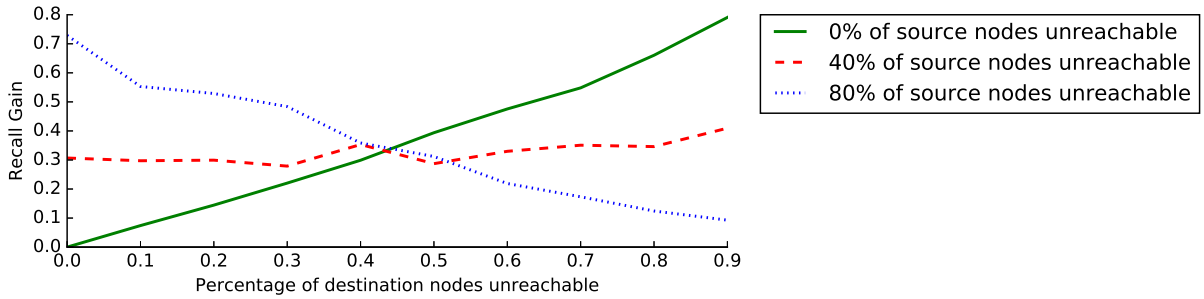


(b) Gain in recall

Figure 3.14: Gain in precision and recall for ISP Synth50 when *Optimal substructure* and *Reverse path symmetry* are used in comparison when no hypothesis is made



(a) Gain in precision



(b) Gain in recall

Figure 3.15: Gain in precision and recall for the *Teslstra* network when *Optimal substructure* and *Reverse path symmetry* are used in comparison when no hypothesis is made

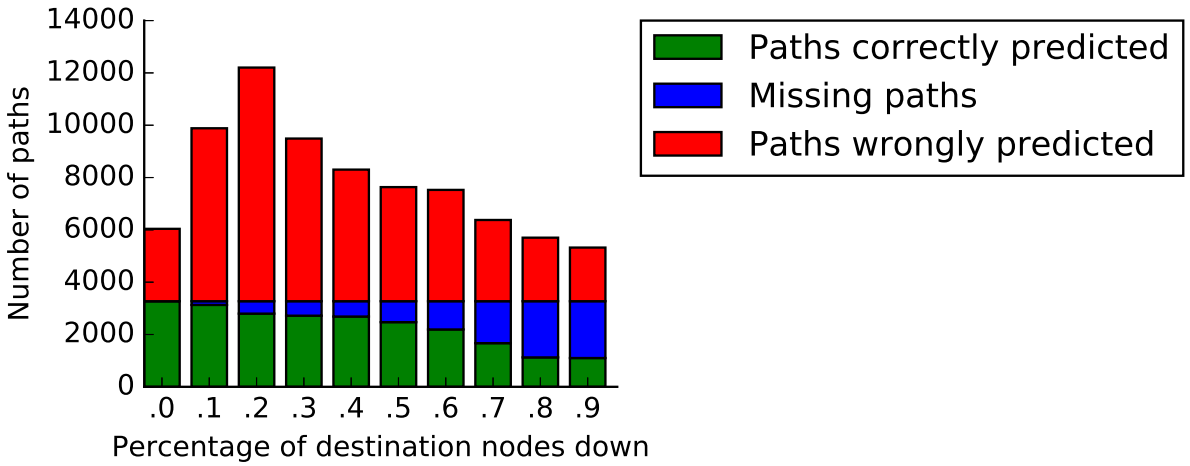


Figure 3.16: Number of paths predicted on ISP Synth50 when all source nodes are available and the number of available destination nodes decreases (with no hypothesis)

- Wrongly predicted: the shortest path predicted using the weights of the solver is not an actual shortest path of the initial topology with its initial weights ($|P_{out}| - |P_{real} \cap P_{out}|$ using notations developed in Section 3.4)

The last category ("Missing") are all the path present in the initial routing but that do not appear in the predicted path ($|P_{real}| - |P_{real} \cap P_{out}|$ using notations developed in Section 3.4).

As we can see on figure 3.16 the solver might predict more paths than it should even when every shortest path is given in input. We observe this in the first bar of figure 3.16 where there is no missing paths but the solver produced weights leading to 6040 paths when only 3273 were given in input explaining the low precision.

When testing the second hypothesis (step 4 of Subsection 3.5.2), we found out that the recall increased while the precision decreased and suspected a enormous number of predicted paths. We thus show figure 3.17. As we can see the number of shortest path is way to high, reaching 70000 in the worst case.

While we do not give more information on the underlying reasons that let the model chose weights that led to bad precisions, we use this observation as the starting point in the following chapter (Chapter 4) where we tried to develop a new model capable of handling the surplus of predicted paths.

3.5.4 Are inferred weights comparable to the real ones?

We would like to measure if the weights produced by the solver and the real ones are similar. As stated in Section 3.1, we can not compare weights directly. Instead of comparing the weights produced by the solver directly with the real ones, we slightly change our aim: we would like to compare the weights produced by the solver with weights characterizing perfectly the network.

We already have computed those weights when answering the first question in Subsection 3.5.1. We can thus use these "perfect" weights and compare them against the one produced by the solver when every shortest path is fed in input. Indeed since both weights are produced by the solver, they are comparable.

In figures 3.18a, 3.18b and 3.18c are shown the Cumulative Distribution Function (CDF) of the weights' values in the *Telstra* network. As it can be observed on those figures, the perfect weights (figure 3.18b) only take six different values to perfectly characterize the routing while the real ones (figure 3.18a) take height different values. While these two values are similar, the actual values taken by the weights are in completely different ranges: $(1, 2, 3, 4, 5, 6)$ for the

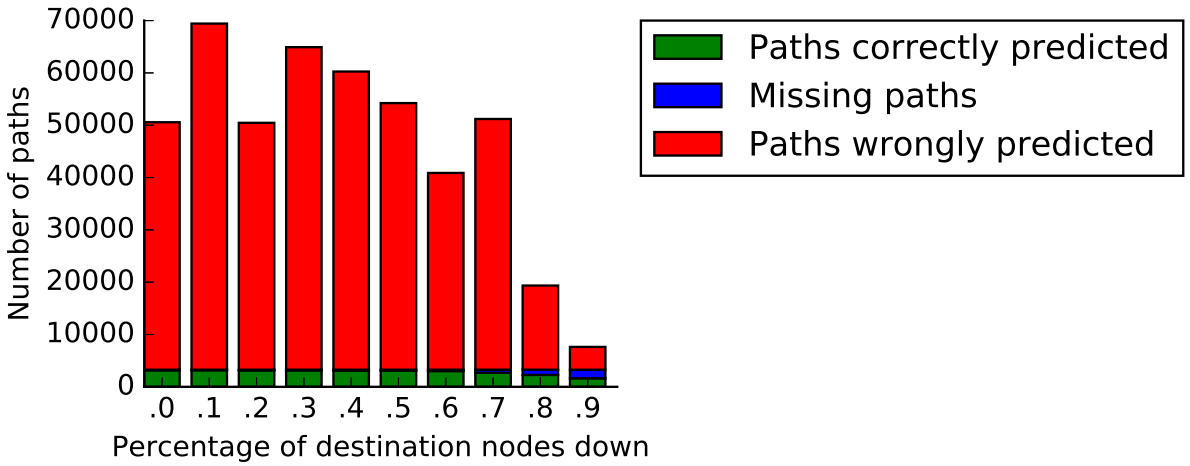


Figure 3.17: Number of paths predicted on ISP Synth50 when all source nodes are available and the number of available destination nodes decreases (with *Reverse path symmetry* hypothesis)

perfect ones,$(100, 200, 250, 300, 400, 450, 500, 700)$ for the real ones. This was expected, as stated they might always differ from a factor, in this case 100.

Let us now compare the perfect weights (figure 3.18b) with the predicted ones (figure 3.18c). The number of different values drops to three for the predicted weights, thus taking the following values: 1, 2 or 3. This last observation let us understand how Gurobi predicts more shortest paths even when all shortest paths are fed in input: a great portion of weights (80% in figure 3.18c) take a singular value (1 in our case) an thus every path from a source to a destination that crosses an equivalent number of edges has a high probability to have the same weight.

Finally, to answer the question: yes, we can compare weights but only if we use weights produced by the *Solver* perfectly characterizing the routing[2].

3.6 Further work

In this section, we describe the possible improvements that could be applied to both the first model (see Section 3.2) and the testing environment (see Section 3.3).

3.6.1 Testing environment

The testing environment described in Section 3.3 needs a graph and its weights as input. We could even go further and create our own topologies. This could be done using IGEN [26] through design heuristics.

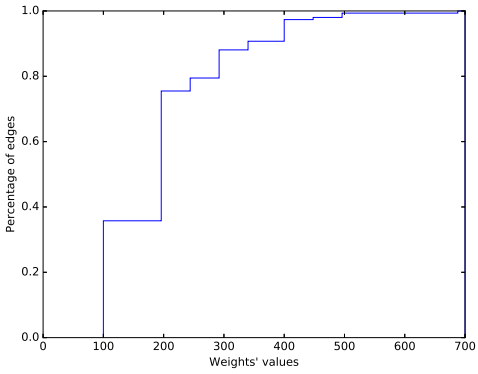
We tested IGEN[3] but did not actually use it for two reasons:

1. It does not provide the possibility to have integer weights for the links between the routers.
2. We already had enough available topologies to test our implementation upon.

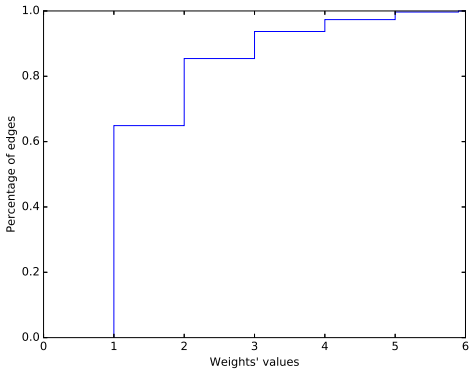
But still we think that it would be very interesting to generate topologies through network design heuristics and use them in our implementation. This would allow to further test the limits of the solver in term of network design.

[2]Similar results are obtained for other topologies and are presented in appendix A.1

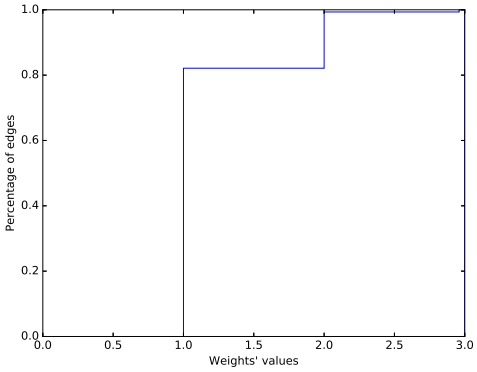
[3]For the installation be careful to: exactly use the CPAN module Graph (version 0.20105) and change line 76 of the *igen-gui.pl* file from "use Statistics::Basic::CoVariance;" to "use Statistics::Basic::Covariance;"



(a) CDF of the real weights



(b) CDF of the "perfect" weights



(c) CDF of the predicted weights

Figure 3.18: CDFs of the weights for the *Telstra* network

3.6.2 First model

While the first model is still tractable, it does take time to get results (several hours for the biggest ISPs). As stated in the design choices section (section 3.1), we chose to use integers only having the consequence that it hardens a lot Gurobi's task to find an optimal solution. A possible improvement would be to use 'continuous' variables instead of integers. As mentioned in section 3.1, a side effect of using integers was the possibility to overcome round-off errors on the weights yielded by Gurobi. If we use continuous variables, this is not the case any more and we have to find another way to mitigate round-off errors as they might lead to wrong shortest paths computations. What we think as a simple (and possibly effective) solution would be to round errors to x significant figures, thus lowering the precision of the returned weights (only considering a precision of 10^{-X}). If x is too low, then the computation of shortest paths might find a lot of additional shortest paths, and inversely. Since the results of our implementation are freely available, we could measure the impact of rounding weights on the shortest paths predicted and thus find the optimal value for x.

A second reason for us to only use integers was that weights in networks are indeed integers (see section 3.1). This allowed us to provide weights characterizing the observed routing in the form of integers as well. If we use continuous weights, this is not the case any more. We can use the same rounding technique as previously described and simply multiply every weight by 10^x such that they all become integers. As for the previous observation, we should measure the impact of the weights obtained by such techniques using our weights as a baseline.

Chapter 4

To an improved model

As presented in Subsection 3.5.3, the model described in [1] might lead to poor performances in term of precision and thus even when the solver is given all the information about the topology.

This motivated us to try and develop a second model. In a first time we identify what causes low precisions and consequently define the problem to be solved. In a second time we present a second model. Lastly, we explain the limits of this model and unfortunately show that it does not exactly answer our needs.

4.1 Problem definition

Let us be more precise about the problem to be solved. Traceroutes are observed paths between two endpoints of the topology. Using non-strict inequalities (Subsection 3.2.3), we let the first solver (Chapter 3) determine whether a non-observed path is a shortest path or not. Since we use '$\leq$' inequalities, the solver has the choice to set it to '=' or '<', if it chooses to set the two paths as equal, then a new shortest path would be "added". The aim of the second solver is to be able to define a new quantity Q_{SD} that represents the number of shortest paths added between the source S and destination D in addition to the one already observed in the traceroutes.

If we were able to express such a quantity Q_{SD} then we could set an upper bound. Since we observed that the solver might predict too much shortest paths between two nodes, this would solve the problem.

4.2 Second model

In this section we first introduce some MIP modelling techniques used throughout the solver, then we describe how to obtain Q_{SD} into a sequence of problem/solution subsections.

Because this chapter describes additions made to the first model, we can safely assume that every constraint described previously still holds. We recapitulate those constraints for the ease of the reader.

1. Equality constraints :

$$W(P_{SiD}) - E(P_{SiD}) = W(P_{Si'D}) - E(P_{Si'D}) \tag{4.1}$$

2. Non-strict inequality constraints:

$$W(P_{SiD}) - E(P_{SiD}) \leq W(P_{Si'I}) - E(P_{Si'I}) + W(P_{Ii''D}) - E(P_{Ii''D}) \tag{4.2}$$

We made a slight change to the implementation on how inequality constraints are set up. In subsection 3.2.5 we pointed out some facts in order to avoid setting up redundant constraints. From now on, we will ignore about these facts because they are not (yet) used in the second model.

We invite the reader to take a look at the chapter 3 in order to understand how these constraints were obtained. Let us recall that:

- Every observed path P has an associated error $E(P)$ such that if $E(P) = 0$ then P is a shortest path, otherwise not.
- Both errors and weights are integers for the reasons described in section 3.1.

4.2.1 Modelling techniques

He we describe the two modelling techniques we use in the following subsections:

Strict inequality modelling

This approach allows to control non-strict inequalities ('$\leq$') constraints: we would like to decompose such constraints into '$<$' and '$=$' and have a boolean to enforce one or another. Let A and B respectively be what is on the left and right side of the non strict inequality ($A \leq B$). Let us define R as:

$$R = \begin{cases} 1 & \text{if} \quad A = B \quad \text{is satisfied} \\ 0 & \text{if} \quad A < B \quad \text{is satisfied} \end{cases} \tag{4.3}$$

We assume A and B to be integers. This approach has been designed for integers only. We can model the previously stated behaviour by replacing $A \leq B$ with two other constraints as described in the following equation:

$$A \leq B \Leftrightarrow \begin{cases} (B - A)R = 0 & (1) \\ (B - A) + R \geq 1 & (2) \end{cases} \tag{4.4}$$

Where R can only take one of two values: 0 or 1. This way if $R = 1$ then the first constraint forces B to be equal to A while the second constraint is in any case satisfied ($1 \leq 1$). If $R = 0$ the first constraint is always satisfied ($0 = 0$) and the difference between B and A must be greater than 1 ($B - A \geq 1 \leftrightarrow B > A$). A limitation imposed by Gurobi [23] means that we can only use equality or non-strict inequality constraints. Thus, the fact that A and B are integers and stating that their difference must be greater than one is equivalent to say that one is greater than the other ($B > A$ in our case). Equation 4.4 yields quadratic terms. Using this kind of approach in our implementation means that our model will no longer be a linear program. Fortunately, Gurobi [23] can handle such quadratic terms.

Non-linear term modelling

While Gurobi [23] allows for quadratic terms, we might be interested in creating terms of higher power or simply to use a linear term instead of a quadratic one. Let Q be a quadratic term and L a linear term. We would like to be able to create a cubic term $C = QL$. This is done by introducing a new linear term T along with the following constraints:

$$C = QL \Leftrightarrow \begin{cases} T = Q & (1) \\ C = TL & (2) \end{cases} \tag{4.5}$$

If used multiple times this equation would allow us to create terms of the desired power.

4.2.2 Adding a shortest path

Let us assume that we have a boolean variable $R_{Si'Ii''D}$ that is equal to 1 when the alternate path $P_{Si'Ii''D}$ (see equation 4.2) is considered as a shortest path and takes value 0 when not. Then Q_{SD} (the number of paths added between the source S and destination D) could be described as :

$$Q_{SD} = \sum_{i',I,i''} R_{Si'Ii''D} \tag{4.6}$$

This first problem focuses on how to describe and implement such $R_{Si'Ii''D}$ values.

Description

The first question to be answered is what are the conditions under which path $P_{Si'Ii''D}$ is considered as a shortest path? We identified two of them:

1. At least one observed path is a shortest path.
2. The weight of the alternate path $P_{Si'Ii''D}$ is equal to the weight of an observed path considered as a shortest path.

We first model these two conditions and then describe how to obtain $R_{Si'Ii''D}$.

Solution for the first condition

Let us denote the first condition as a boolean variable R_{SD} whose behaviour is the following:

$$R_{SD} = \begin{cases} 1 & \text{if one of the observed path is shortest} \\ 0 & \text{if not} \end{cases} \tag{4.7}$$

A path is considered as on the shortest if its error is equal to 0. Thus we define a new variable S_{SiD} for each observed path P_{SiD}:

$$S_{SiD} = \begin{cases} 1 & \text{if} \quad E(P_{SiD}) = 0 \\ 0 & \text{if} \quad E(P_{SiD}) > 0 \end{cases} \tag{4.8}$$

Using the first modelling technique (see subsection 4.2.1) we can express S_{SiD} for our model. We can now define R_{SD} as follows:

$$R_{SD} = \begin{cases} 1 & \text{if} \quad 0 < \sum S_{SiD} \\ 0 & \text{if} \quad 0 = \sum S_{SiD} \end{cases} \tag{4.9}$$

This expression does not exactly match the first modelling technique. We thus introduce another variable T_{SD} having the opposite behaviour of R_{SD}:

$$T_{SD} = \begin{cases} 1 & \text{if} \quad 0 = \sum S_{SiD} \\ 0 & \text{if} \quad 0 < \sum S_{SiD} \end{cases} \tag{4.10}$$

Then $S_{SD} = (1 - T_{SD})$ and T_{SD} can easily be modelled. Let us show that S_{SD} has the desired behaviour. If at least one observed path is considered as a shortest path then there is at least one $S_{SiD} = 1$ resulting in $\sum S_{SiD} > 0$, $T_{SD} = 0$ and finally $S_{SD} = 1$. If none of the observed path is considered as a shortest path, then $\sum S_{SiD} = 0$, $T_{SD} = 1$ and $S_{SD} = 0$.

Solution for the second condition

The second condition is a bit more complex. Let us remember that if there are several observed paths between S and D then equality constraints have been set up as described in equation 4.1. If one of the observed path between S and D is considered as a shortest, then the adjusted weight (weight W diminished by its excess E) of the others observed paths will take the same value (due to equality constraints). Thus, the adjusted weight ($W - E$) of every path between S and D is equal to the weight of the shortest path. Therefore we can rephrase the second condition as: the weight of the alternate path is equal to the adjusted weight of one of the observed paths (because they are all equal). Let us thus take one observed path P_{SiD} and define $S_{Si'Ii''D}$ with the following behaviour:

$$S_{Si'Ii''D} = \begin{cases} 1 & \text{if} \quad W(P_{SiD}) - E(P_{SiD}) = W(P_{Si'Ii''D}) \\ 0 & \text{if} \quad W(P_{SiD}) - E(P_{SiD}) < W(P_{Si'Ii''D}) \end{cases} \tag{4.11}$$

As we can see, there is no error terms for the alternate path $P_{Si'Ii''D}$, this is because we would like to be able to know whether this alternate path is a shortest path or not, or equally, if its weight is equal to the weight of any shortest path between S and D. This variable $S_{Si'Ii''D}$ is simply translated in our model using the first modelling technique (Subsection 4.2.1).

Modelling $R_{Si'Ii''D}$

Modelling $S_{Si'Ii''D}$ is now an easy task: $R_{Si'Ii''D} = R_{SD}S_{Si'Ii''D}$. Indeed if the two conditions are respected then both R_{SD} and $S_{Si'Ii''D}$ are equal to 1 and so is $R_{Si'Ii''D}$. If one of them is not respected (and thus equal to 0) then $R_{Si'Ii''D}$ is equal to 0 thus describing the desired behaviour.

It should be noted that $R_{Si'Ii''D}$ describes the fact that "the weight of the alternate path is equal to the adjusted weight of one of the observed path", and therefore, it is only necessary to set up the inequality constraint for one observed path and not all of them (thus retaking the way to set up constraints described in subsection 3.2.5).

4.2.3 Adding all shortest paths

The attentive reader will have spotted a problem that might arise when describing Q_{SD} as set up in equation 4.6. Indeed since using the constraint reduction criterion (subsection 3.2.2) allows us to not set up inequality constraints for every alternate paths, there might be missing paths when computing Q_{SD}.

We expose this problem on our sandbox graph, then describe the solution and show that this solves this issue on our graph.

Description using the sandbox

Let us assume that we have seen the traceroutes described in table 4.1 and set up the constraints accordingly as described in table 4.2.

Putting every error to 0, figure 4.1 presents a possible weights' assignment. By using the solution already described above for the first problem (see subsection 4.2.2), we set up two variables, namely R_{AEFC} and R_{ADF}.

Since $AEFC$ and ADF are shortest paths (figure 4.1), both variables R_{AEFC} and R_{ADF} are equal to 1. Table 4.3 compares the value taken for Q_{AC} and Q_{AF} using equation 4.6 and the values it should have taken by looking at the graph. As it can be seen, Q_{AC} does not have the correct value. Indeed, the combination of both $AEFC$ and ADF has created a second new shortest path between A and C: $ADFC$. The information of path ADF being chosen as shortest should have been "propagated" to Q_{AC}. Based on this observation, we refine the computation of Q_{AC} in the next subsection.

(1)	A B C
(2)	A E F
(3)	F C
(4)	A D
(5)	D F

Table 4.1: Traces observed

(1)	$W(P_{ABC}) - E(P_{ABC})$	$\leq$	$W(P_{AEF}) - E(P_{AEF}) + W(P_{FC}) - E(P_{FC})$
(2)	$W(P_{AEF}) - E(P_{AEF})$	$\leq$	$W(P_{AD}) - E(P_{AD}) + W(P_{DF}) - E(P_{DF})$

Table 4.2: Constraints for the naive approach

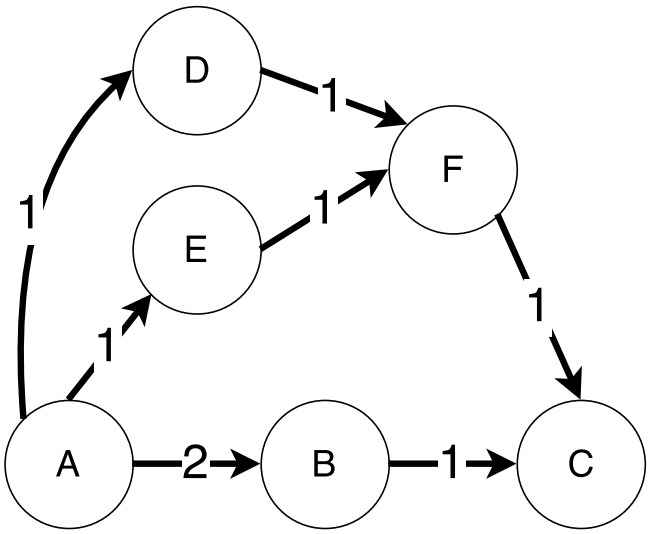


Figure 4.1: Example graph with assignment

Solution

We can see equation 4.2 as a dependency between Q_{SD}, Q_{SI} and Q_{ID}. The value of the left value of the inequality depends on the two of the right. If $P_{Si'Ii''D}$ is considered as a shortest path ($R_{Si'Ii''D} = 1$) then Q_{SI} and Q_{ID} must be added to Q_{SD} (if they are defined by the instance). Thus we define Q_{SD} as:

$$Q_{SD} = \sum_{i',I,i''} R_{Si'Ii''D} + R_{Si'Ii''D}(Q_{SI} + Q_{ID}) \tag{4.12}$$

In equation 4.12, if $R_{Si'Ii''D} = 1$ then both Q_{SI} and Q_{ID} are added to Q_{SD}.

From an implementation point of view, this can be simply done by adding the constraints described in equation 4.12.

	Using equation 4.6	Expected value
Q_{AC}	1	2
Q_{AF}	1	1

Table 4.3: Comparison of equation 4.6 and reality

On sandbox example

By taking the exact same example as the one described in the first part of subsection 4.2.3, we can compute Q_{AC} and Q_{AF} using equation 4.12. So have:

$$Q_{AC} = R_{AEFC} + R_{AEFC}(Q_{AF})$$

$$Q_{ADF} = R_{ADF}$$

Since $Q_{ADF} = 1$, we conclude that $Q_{AC} = 2$ which is the expected value (see table 4.3).

4.3 Limitations

Our implementation of this second model was not tractable enough to handle topologies above 10 nodes and 20 links. Indeed when we tried to it on a bigger topology (50 nodes, 138 links) Gurobi's solver warned us that our model might take 800000 seconds per iteration, summing up to almost one year and a half since 50 iterations of the solving algorithm are usually necessary to find the optimal solution [1]. This limitation proved to be a problem when trying to assess the performances of our implementation.

Finally after creating several topologies manually, we found out that, in some cases, our implementation of the Q_{SD} quantity would not always take the right value. We present such a case in Subsection 4.3.1, we then give a new meaning to Q_{SD} in Subsection 4.3.2.

4.3.1 When is Q_{SD} wrongly defined?

Let us consider the topology described in figure 4.2 where the initial weights of the topology are assigned to each edge. We are interested in Q_{AE} (the number of paths predicted in addition to the one observed). Note that in the graph described by figure 4.2, only a single shortest path between A and E exists, namely $ABCDE$.

Let us now create some traceroutes on this graph, more particularly, the ones described in table 4.4 and consider them as being observed. Accordingly to these observed traces, we set up the constraints described in table 4.5. Let us remember that for every inequality constraint, a variable $R_{Si'Ii''D}$ is as well set up (as described in Subsection 4.2.2).

This system is solvable with an optimal objective equal to 0, meaning that every error term is equal to 0. With such a system, Gurobi might chose to set every weight equal to 1 as presented in figure 4.3. In this context, every inequality constraints are satisfied by equating both sides of the inequality hence all $R_{Si'Ii''D}$ are equal to 1

Let us now look at the value taken by Q_{AE} using equation 4.12.

$$Q_{AE} = R_{AFCDE} + R_{AFCDE}(Q_{AF} + Q_{FE}) + R_{ABCGE} + R_{ABCGE}(Q_{AG} + Q_{GE})$$

Let us as well define Q_{FE} and Q_{AG}.

$$Q_{FE} = R_{FCGE} + R_{FCGE}(Q_{FG} + Q_{GE})$$

$$Q_{AG} = R_{AFCG} + R_{AFCG}(Q_{AF} + Q_{FG})$$

Since they do not appear on the left of any constraint, Q_{AF}, Q_{GE} and Q_{FG} are equal to 0. As previously stated, $R_{AFCDE} = R_{ABCGE} = R_{FCGE} = R_{AFCG} = 1$ and thus Q_{AE} is equal to 4. The attentive reader will now have spotted the problem: at the beginning only one path between

[1] According to Gurobi [23]

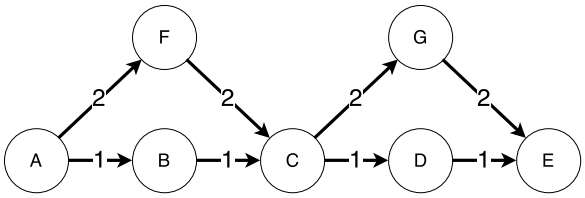


Figure 4.2: Topology with initial weights

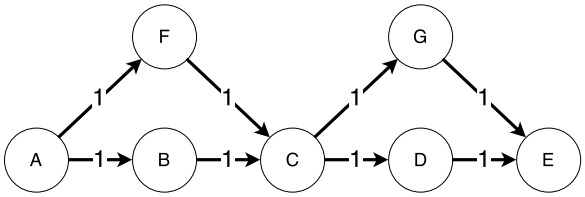


Figure 4.3: Topology with predicted weights

A and E was observed (see figure 4.2) and now there are 4 of them (see figure 4.3), hence Q_{AE} should have been equal to $4 - 1 = 3$.

To analyse what happens, let us go through the constraints described in table 4.5 one by one (remember that error terms can be set to 0). Equating both sides of constraint (1) means that the path $AFCDE$ will be added a shortest path between A and E ($W(P_{ABCDE}) = W(P_{AFCDE})$). Doing the same for constraint (3) then yields that $AFCGE$ is also a shortest path ($W(P_{FCDE}) = W(P_{FCGE})$. A similar observation for constraint (2) adding the path $ABCGE$ ($W(P_{ABCDE}) = W(P_{ABCGE})$). And this is where things go wrong, when constraint (4) sides are also equal: the path $AFCGE$ is added a second time ($W(P_{ABCG}) = W(P_{AFCG})$).

(1)	A B C D E
(2)	A F
(3)	F C D E
(4)	A B C G
(5)	G E
(6)	F C G

Table 4.4: Traceroutes made on topology described in figure 4.2

(1)	$W(P_{ABCDE}) - E(P_{ABCDE})$	$\leq$	$W(P_{AF}) - E(P_{AF}) + W(P_{FCDE}) - E(P_{FCDE})$
(1)	$W(P_{ABCDE}) - E(P_{ABCDE})$	$\leq$	$W(P_{ABCG}) - E(P_{ABCG}) + W(P_{GE}) - E(P_{GE})$
(1)	$W(P_{FCDE}) - E(P_{FCDE})$	$\leq$	$W(P_{FCG}) - E(P_{FCG}) + W(P_{GE}) - E(P_{GE})$
(1)	$W(P_{ABCG}) - E(P_{ABCG})$	$\leq$	$W(P_{AF}) - E(P_{AF}) + W(P_{FCG}) - E(P_{FCG})$

Table 4.5: Constraints

4.3.2 A new meaning for Q_{SD}

The example described in the previous Subsection(4.3.1) means that our definition of Q_{SD} sometimes counts an added path several times. Thus our definition is only an upper bound of the quantity it should describe. This is a problem since initial aim was to be able to constraint such a quantity in order to control the number of path added between a source S and a destination D. We unfortunately did not manage to correctly define Q_{SD} in our model.

4.4 Further work

Concerning the second model described in this chapter, an upper bound on the number of paths added between two nodes is not enough and the model as it is now, clearly is not usable. Not only the model does not provide the quantities it is supposed to, but the use of quadratic constraints hardens even more the task for Gurobi to find an optimal solution (predicted at least one and a half year to solve an instance of the problem for ISP Synth50). In order to soften this task, we could use the same techniques as described for the first model (use continuous variable and round errors), but the aim obviously remains to first express a model correctly quantifying the number of shortest path predicted in addition to the one observed.

Chapter 5

Inferred topologies

In this chapter are presented and evaluated the topologies inferred by Hydrogen. Note that we only show results for the topologies shown in table 5.1 since these are the only ones for which we could find public maps that were possible to translate manually to graph files.

We begin by explaining how the two stages are linked. Then we present our maps (only *Internet2* and *Hivane* could be drawn) with their inferred weights. Finally we evaluate both the topology and the link inferences.

Let us remember that stage one infers a topology while stage two infers link weights.

5.1 Link between hydrogen stages

In this section we explain how stage 1 outputs can be fed to the second stage.

Once the graph is completed, the next step is to make it compatible with the second Stage's inputs. Let us remember that two inputs are needed as described at the beginning of Chapter 3.

- A topology (or Graph).
- Traceroutes collected for this topology.

Those inputs can be fed in the form of files: the Graph File and the Traceroute File.

The Graph File, whose role is to define the topology, contains all the information about the nodes and edges. Details on the file's format are left in the code, but note that this file contains the information inferred from the first stage. Let us also note that the first stage considered links as undirected while the second stage used directed graphs. This is not a problem: the first stage just has to produce edges in both directions.

Concerning the Traceroute File, it should contain every traceroute passing through the AS and each traceroute should appear as much time as it appeared in the initial set of traces. Consequently, we couldn't use the reduced set of traceroutes, when generating the traceroute, but all of them. To achieve this, we had to go back to the intermediary traceroute file, described in section 2.5. Looping over all traceroutes saved in this files, producing all paths that corresponds

AS	name	Routers	Links
11537	Internet2	10	14
34019	Hivane	7	11
559	Switch	59	99

Table 5.1: ISPs with the number of backbone routers and links

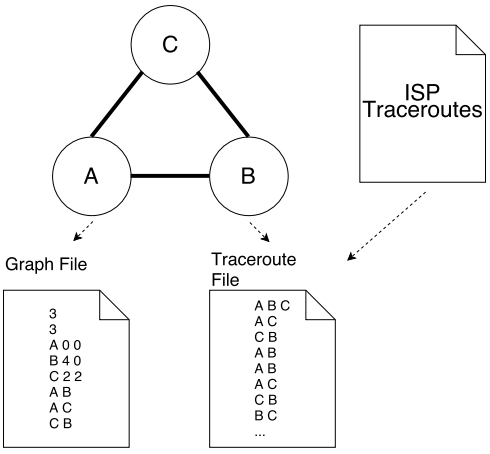
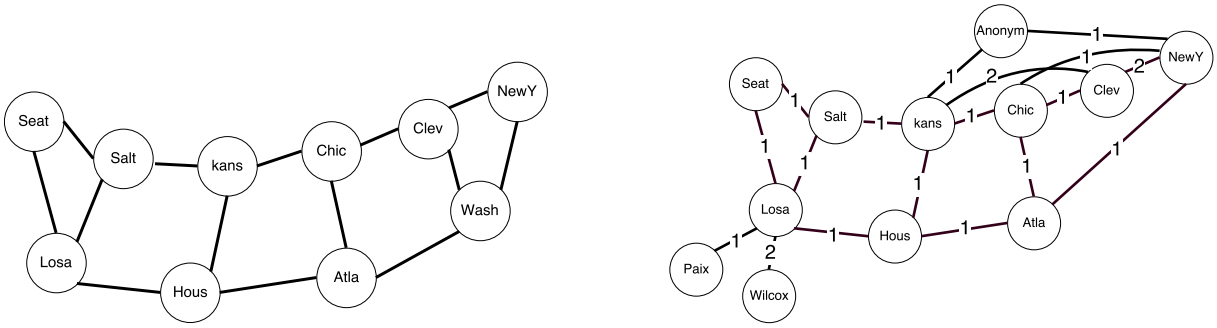


Figure 5.1: Graph is converted into 2 files before Stage 2



(a) Topology redrawn from weather map[1]

(b) Inferred topology from traceroutes

Figure 5.2: Weather map topology compared to our inferred topology for the *Internet2* network

to the graph and changing hop's names to correspond with nodes names allowed us to create the second input of the *Solver*.

5.2 Topologies

The *Internet2* and *Hivane* topologies are respectively shown in figures 5.2 and 5.3. For these two topologies, weather maps describing them were available such that we could redraw them in figures 5.2a and 5.3a. Our inferred topologies are shown aside from the weather maps to allow for easy comparison (see figures 5.2b and 5.3b).

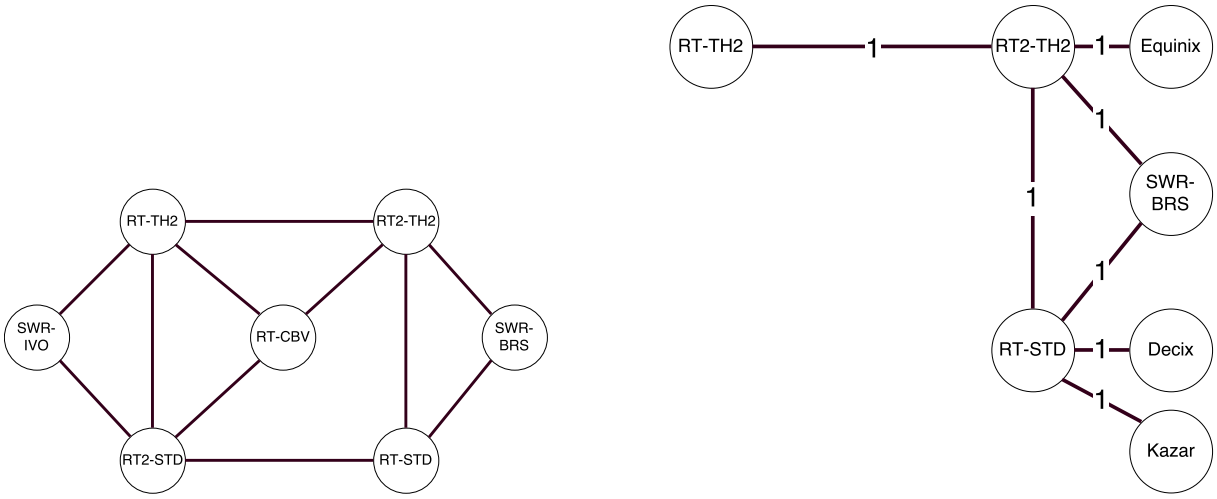
5.3 Evaluation of the inferences

5.3.1 Topology inference

Evaluating a topology proved to be a hard task. We could not compare them node by node, edge by edge, because nodes in inferred maps had other names than nodes from the weathermaps. We decided to use the CDF of node's degree as a proper benchmark between two graphs. Unfortunately results of this comparison were not concluding. For example, figure 5.4 compares both graphs for *Internet2*. As we could see on figure 5.2, the inferred topology is not very different from the real topology. But the CDFs are very disparate, mostly due to the two additional routers linking *Losa* with other ASes, and the disappearance of *wash* which perturbate the topology.

[1]From `http://atlas.grnoc.iu.edu/atlas.cgi?map_name=I2%20IPv6`

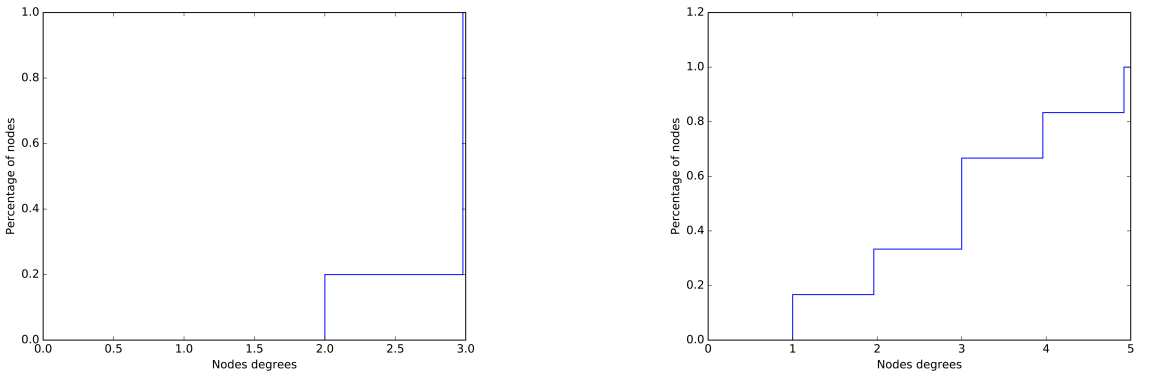
[2]From`http://www.hivane.net/info/weathermap.html`



(a) Topology redrawn from weather map [2]

(b) Inferred topology from traceroutes

Figure 5.3: Weather map topology compared to our inferred topology for the *Hivane* network



(a) CDF of the node degree in Internet2's weather map

(b) CDF of the node degree in Internet2's inferred map

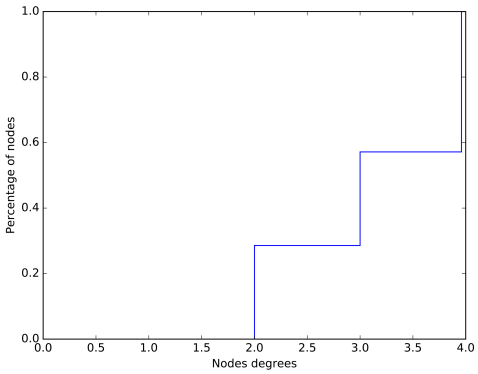
Figure 5.4: Node degree CDF comparison for Internet2

When examining figures 5.6. We retrieve more similarities between the two CDFs. Even if the weathermap CDF present a "higher resolution" in the high-degree nodes, the proportion for the low-degree nodes are very similar between the two figures. Lastly, the inferred map of the ISP *Hivane* revealed itself as only a fraction of the real network. This means that a large part of this ISP was undiscovered by the traceroutes, and thus not visible on the inferred topology. The two CDFs from figure 5.5 do not show any similarities, which is normal for such different maps.

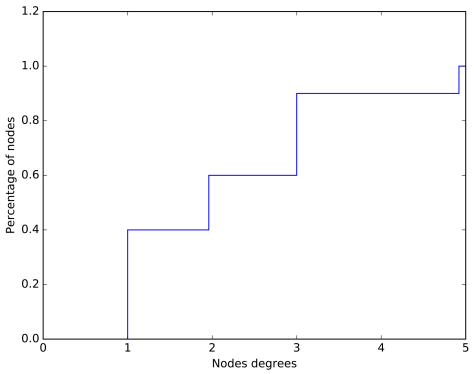
5.3.2 Link weights inference

The evaluation of the first model (described in Section 3.5) provided us with useful informations about when the link weights inference should be trusted. Indeed, since Stage 1 produces a topology along with traceroutes, we can determine a percentage of source/destination nodes reachable and thus estimate the performance of the second stage. This estimation has to be done using an a priori known topology (see table 3.8), so we select a known topology having a similar number of nodes and edges.

For each topology presented in table 5.2 we thus provide the percentage of source/destination nodes unreachable, the similar topology chosen and the expected precision and recall of the

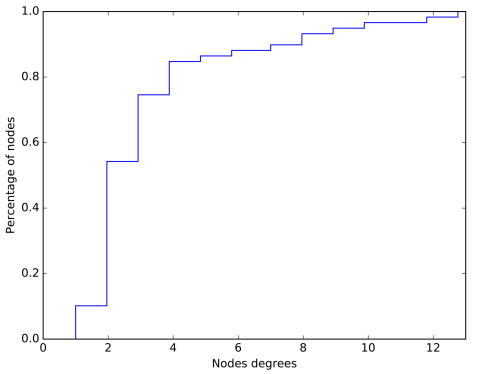


(a) CDF of the node degree in Hivane's inferred map

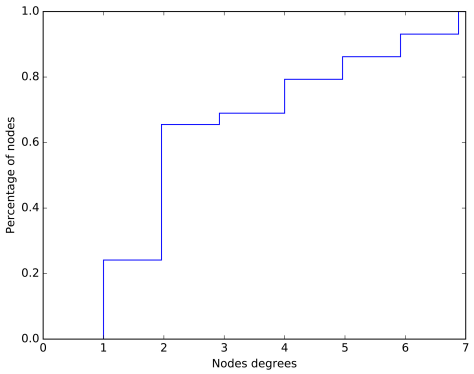


(b) CDF of the node degree in Hivane's inferred map

Figure 5.5: Node degree CDF comparison for Hivane



(a) CDF of the node degree in Switch's weather map



(b) CDF of the node degree in Switch's weather map

Figure 5.6: Node degree CDF comparison for Switch

Topology	% Source nodes	% Destination nodes	reference	precision	recall
Internet2	0.29	0.43	Telstra	0.7	0.85
Hivane	0.33	0.0	Telstra	0.7	1
Switch	0.45	0.66	Telstra	0.7	0.75

Table 5.2: Source nodes unreachable, destination nodes unreachable, reference topology, precision and recall (expected) for each topology

weights' predictions (see Section 3.4 for more details). Note that the percentage of source/destination nodes reachable is computed using the number of nodes found by the first stage and not the ones of the real topology: we want to characterize if the routing is good and do not assess the topology itself.

While these results are reflecting a good characterization of the network, they have to be put in perspective of the evaluation of the first stage. For example, the characterization of the routing for the *Hivane* network has a recall equal to 1 but only for 4 out of 7 nodes that are actually in the topology were discovered.

5.4 Further work

As explained before in section 1.5, we encountered quite a lot of issues in our topologies inference due to the infamous MPLS routing protocol. Section 1.1 explained why we should fear this protocol. Now we see the influence of MPLS on the results.
With MPLS, network operators can hide their traffic, and thus topologies, from the "eyes" of Traceroute. This is very inconvenient for us, because we can only infer from data we have. And when a part of the topology is obfuscated by MPLS, it simply do not appears in the generated map. Another issue is the "resolution" of Caida's data. And by resolution we mean the coverage of a network by traceroutes. Some ISP are very well covered, like *Internet2*. Others, like *Switch* benefit a lesser coverage. This is mainly due to a higher Ark probe density in America than in Europe.
To fix those two issues, Caida could deploy more probes to increase their worldwide coverage, and implement anti-MPLS techniques in their Traceroute to fight opaque tunnels [14].

Another improvement, this time in the test environment, would be the implementation of a better metric for comparing two graphs. Comparing router's degree CDFs is a good beginning, but a better benchmark could be used. Tale [27] is a good candidate for this task.

Chapter 6

Conclusion

Realistic network topologies are a key element for the research community in the field of networking as they might guide protocol designs and help understand the Internet's structure. In this thesis we revealed Hydrogen, an inference tool inspired by Rocketfuel [3] which measured and published public maps for 10 ISPs in 2002. Hydrogen is composed of two inference stages: the router-level topology inference, similar to RocketFuel, generates maps, and the link weights inference, fully inspired by [1] computes realistic link weights for the topology. These stages are consecutive and feature a modular design. We share Hydrogen with the community on an open-source basis[1].

Hydrogen's first stage (the topology inference engine) is a modernized version of the 14 years old RocketFuel. The aim of this topology inference engine is to build realistic internet networks topologies by using traceroutes.
Hydrogen fulfils the same task as RocketFuel, but in a way that makes sense in the current age of telecommunications. Instead of collecting data by itself, Hydrogen taps into Caida's huge databanks of traceroutes collected from probes around the world, just waiting to be exploited (section 1.2.4). This makes Hydrogen faster and more reliable. This first stage is also now compatible with Internet Protocol version 6, which should make it future-proof.

But the first stage can produce accurate topologies only as long as the traceroutes are complete. Hydrogen cannot infer any information about the network if there is no information about it in the traceroutes (section 5.4). In addition, if there are anomalies in the data, they will be anomalies in the inferred topology. Unfortunately, this is often the case. One of the main culprits is MPLS (section 1.1), which enables network operators to completely hide their topologies from the Traceroute tool. By routing packets into opaque MPLS tunnels, no information can be collected about the MPLS hops. Furthermore, Caida's traceroute coverage is not complete. Probe density varies for different locations, and so does the "resolution" of the collected information (section 5.4).

As indicated previously, Hydrogen's second stage's aim is to enhance router-level topologies with link weights characterizing observed routing in the best possible way. This was possible using a constraint-based approach.

To evaluate the quality of the inferred link weights for unknown topologies as well as to understand the limits of this second stage's piece called the *Solver*, we developed a testing environment (described in 3.3) . This environment allowed to experiment the solver using a priori known topologies by simulating the lifespan of a network (including failures) and the collection

[1] Available at `https://bitbucket.org/gfeuillen/hydrogen`

of data such as traceroutes. A possible improvement would be to produce our own topologies through design heuristics as it could be done with IGEN [26]. This possibility is examined in Section 3.6.1.

The simulation of a network lifespan led us to identify a second essential component that was necessary to the performance evaluation: quality measures. All of the quality measures described in [1] (except 1) were not suited for our new, non-biased, testing environment, we thus described new quality measures for our inference tool. These quality measures were borrowed from Information Retrieval and customised to our needs in Section 3.4.

The evaluation of this first stage (described in Section 3.5) helped us to prove its operation as well as understanding its limits. More precisely, we identified two of them:

- The increasing computation time for the inference for large networks. This issue was addressed in Section 3.6.2 where we identified a possibility to facilitate Gurobi's [23] task to find an optimal solution.
- The model had a quiet overall low precision. This issue was explained in the evaluation of the first model and led us to develop a second model (see Chapter 4). This second model proved itself to be both harder to solve as well as inaccurate but we think of it as a base for a new model.

Finally, the collaboration of phase 1 and 2 was achieved in Chapter 5. In this Chapter, we provided inferred maps along with their weather-maps to allow for comparison. The router topologies inferred showed inaccuracies (see Subsection 5.3.1), mostly due to incomplete traceroute coverage and some routing protocols such as MPLS. The link weights inference appeared to be accurate (see Subsection 5.3.2) but those impressions had to be put in perspective of the first stage's results: predicting accurate link weights on partially inferred maps only has a little meaning.

To our knowledge, we are the first ones to freely provide such an implementation and we hope that through its design, researchers will be able to improve Hydrogen.

Bibliography

[1] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 231–236.

[2] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[3] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring isp topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, Feb. 2004. [Online]. Available: http://research.cs.washington.edu/networking/rocketfuel/papers/sigcomm2002.pdf

[4] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker, "In search of path diversity in isp networks," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. ACM, 2003, pp. 313–318.

[5] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding traceroute anomalies with paris traceroute," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 153–158. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177100

[6] "Center for applied internet data analysis," http://www.caida.org/home/, accessed: 2016-05-24.

[7] Y. Rekhter and T. Li, "A border gateway protocol 4 (bgp-4)," United States, 1995.

[8] E. REILENT, "Whiteboard architecture for the multi-agent sensor systems," Ph.D. dissertation, dissertation. Tallinn: Tallinn University of Technology Press, 2012.

[9] A. Basu and J. Riecke, "Stability issues in ospf routing," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 225–236, 2001.

[10] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets," in *ACM SIGCOMM computer communication review*, vol. 31, no. 4. ACM, 2001, pp. 15–26.

[11] "The ipv6 topology dataset," http://www.caida.org/data/active/ipv6_allpref_topology_dataset.xml, accessed: 2016-05-24.

[12] E. Rosen, A. Viswanathan, R. Callon *et al.*, "Multiprotocol label switching architecture," 2001.

[13] D. O. Awduche and J. Agogbua, "Requirements for traffic engineering over mpls," pp. 7–8, 1999.

[14] B. Donnet, M. Luckie, P. Mérindol, and J.-J. Pansiot, "Revealing mpls tunnels obscured from traceroute," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 2, pp. 87–93, 2012.

[15] "Minimum allocation," https://www.ripe.net/publications/docs/ripe-481#minimum_allocation, accessed: 2016-06-03.

[16] L. Daigle, "WHOIS Protocol Specification," RFC 3912, Mar. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc3912.txt

[17] K. Keys, "Internet-Scale IP Alias Resolution Techniques," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 40, no. 1, pp. 50–55, Jan 2010.

[18] M. Luckie, R. Beverly, W. Brinkmeyer, and k. claffy, "Speedtrap: Internet-Scale IPv6 Alias Resolution," in *Internet Measurement Conference (IMC)*, Oct 2013, pp. 119–126.

[19] "Explore proximus connectivity," https://www.proximuswholesale.be/en/id_explore/public/connectivity/explore.html, accessed: 2016-06-03.

[20] Y. Hyun, "The archipelago measurement infrastructure," *7th CAIDA-WIDE workshop*, Nov. 2006. [Online]. Available: http://www.caida.org/publications/presentations/2006/young_wide0611_ark/young_wide0611_ark.pdf

[21] M. J. Luckie, "Scamper: a scalable and extensible packet prober for active measurement of the internet." in *Internet Measurement Conference*, M. Allman, Ed. ACM, 2010, pp. 239–245. [Online]. Available: http://dblp.uni-trier.de/db/conf/imc/imc2010.html#Luckie10

[22] "Graphstream a dynamic graph library," http://graphstream-project.org/, accessed: 2016-05-02.

[23] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2015. [Online]. Available: http://www.gurobi.com

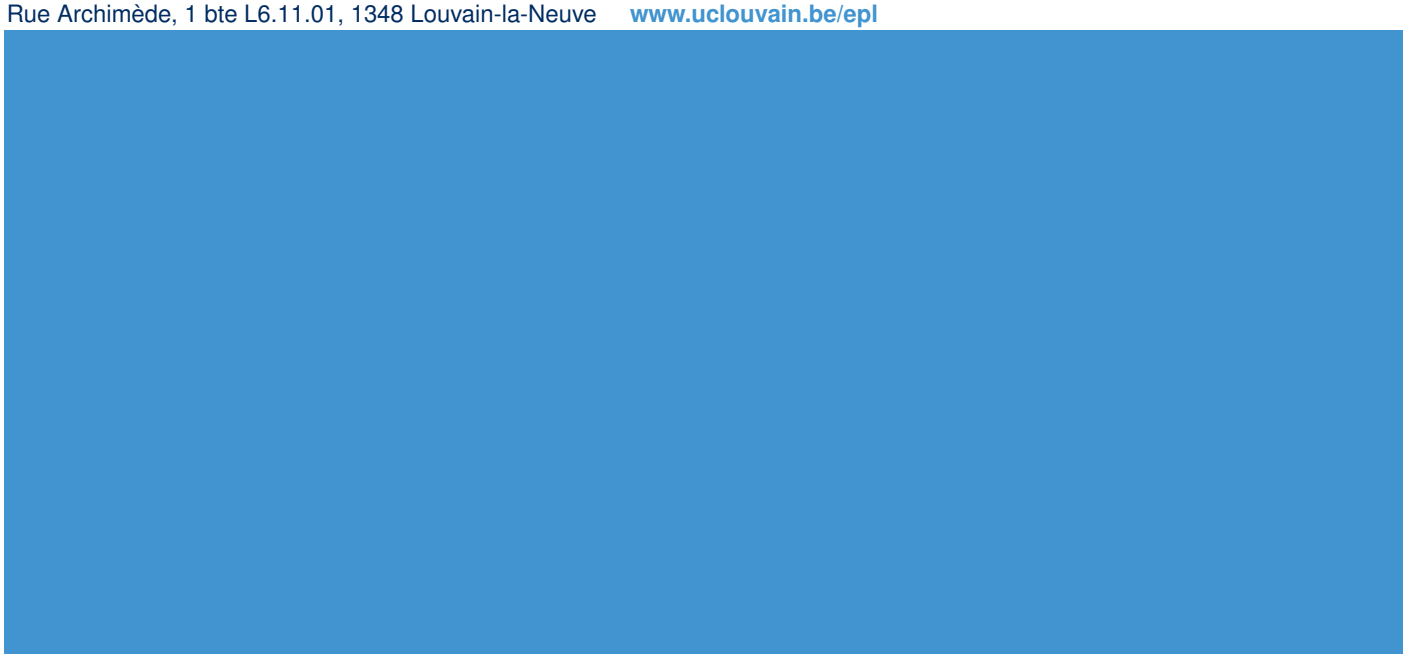
[24] J. Moy, "Ospf version 2," 1997.

[25] C. Sammut and G. I. Webb, *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[26] B. Quoitin, "Igen: Topology generation through network design heuristics," 2010. [Online]. Available: http://informatique.umons.ac.be/networks/igen/

[27] Y. Tian and J. M. Patel, "Tale: A tool for approximate large graph matching," in *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*. IEEE, 2008, pp. 963–972.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**



Appendices

Appendix A

Additional figures for the first model

This appendix regroups all the figures obtained for the evaluation of the first model (see Section 3.5). It follows the exact same structure as the associated section: a flow of questions. In addition to this flow, figures are grouped by ISP, we repeat the ISPs key information in table A.1 for the ease of the reader. For more information about this appendix, we invite the reader to go through Section 3.5 first.

AS	name	Routers	Links	Symmetry
	synth50	50	138	0.04
	synth100	100	286	0.99
1221	Telstra (au)	104	151	1.0
1755	Ebone (eu)	87	161	1.0
3967	Exodus (us)	79	147	1.0

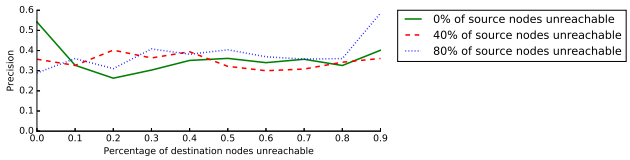
Table A.1: ISPs with the number of backbone routers and links and the percentage of symmetric links

A.1 Are *Optimal substructure* and *Reverse path symmetry* correct/useful hypotheses ?

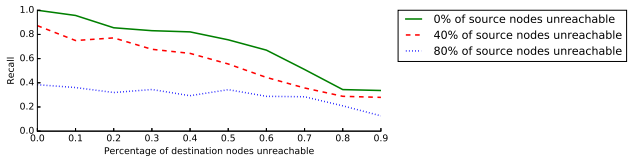
In this section are contained all the figures associated to the question answered in Subsection 3.5.1.

A.1.1 When no hypothesis is used

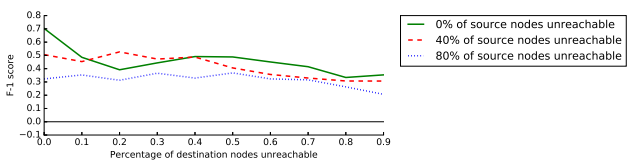
Synth50



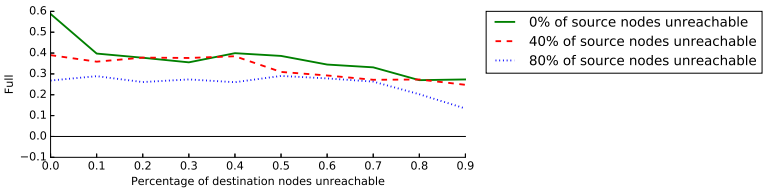
(a) Precision



(b) Recall



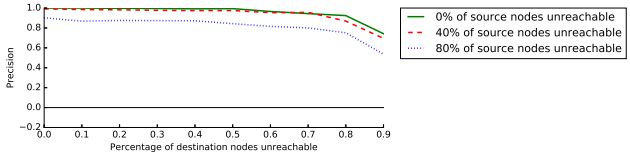
(c) $F_1 - score$



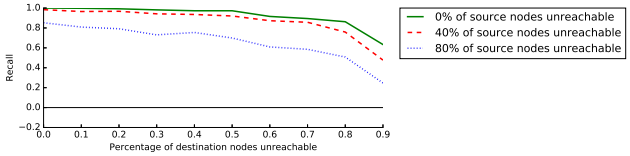
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.1: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP Synth50 when no hypothesis is made

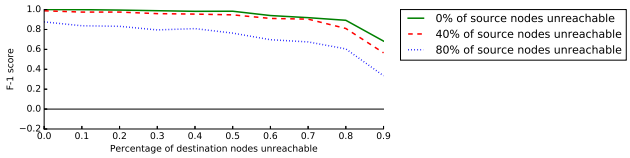
Synth100



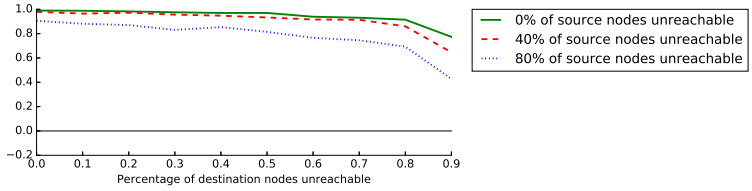
(a) Precision



(b) Recall



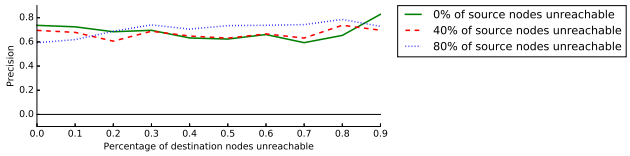
(c) $F_1 - score$



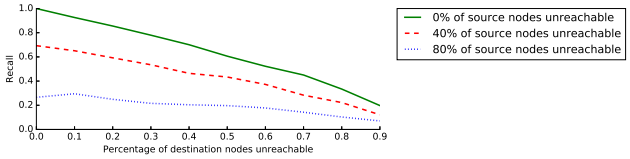
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.2: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP Synth100 when no hypothesis is made

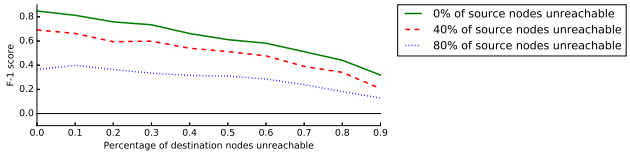
Telstra network



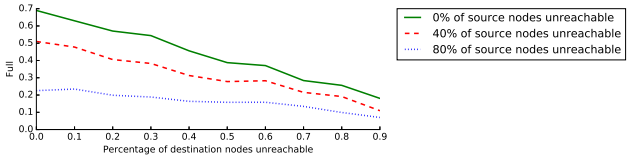
(a) Precision



(b) Recall



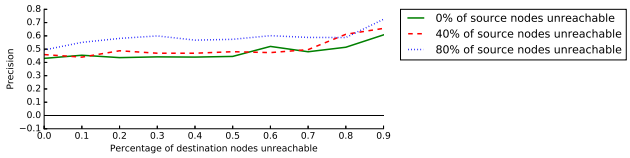
(c) $F_1 - score$



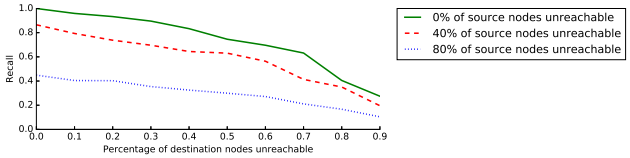
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.3: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for *Telstra* network when no hypothesis is made

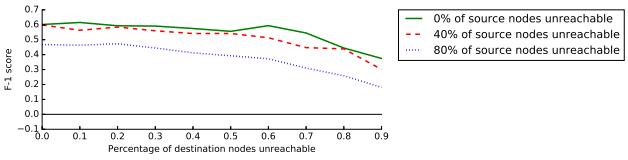
Ebone network



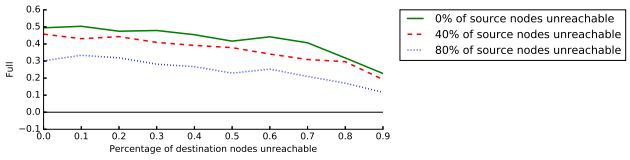
(a) Precision



(b) Recall



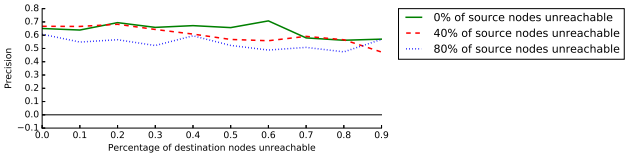
(c) $F_1 - score$



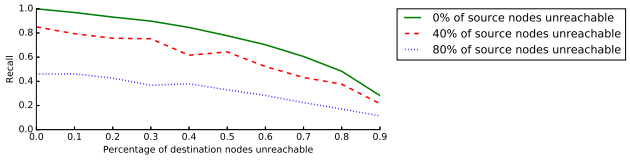
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.4: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Ebone* network when no hypothesis is made

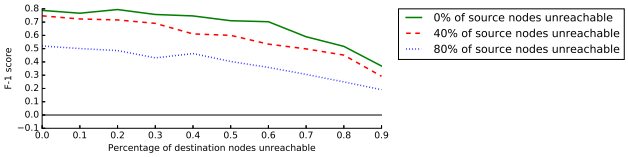
Exodus network



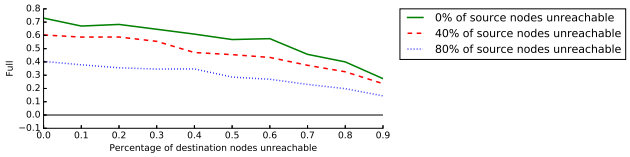
(a) Precision



(b) Recall



(c) $F_1 - score$

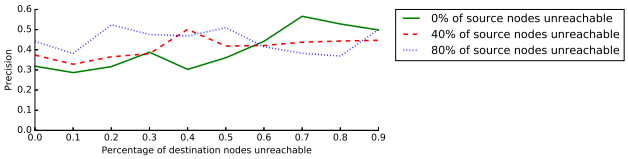


(d) The fraction of node-pairs between which the routing was fully characterized

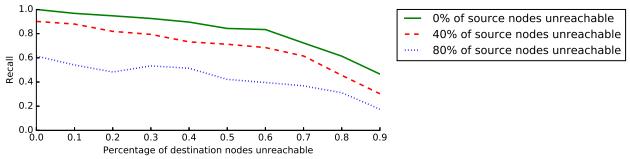
Figure A.5: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Exodus* network when no hypothesis is made

A.1.2 When *Optimal substructure* hypothesis is made

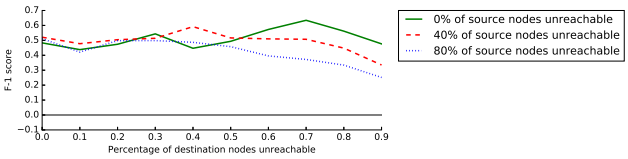
Synth50



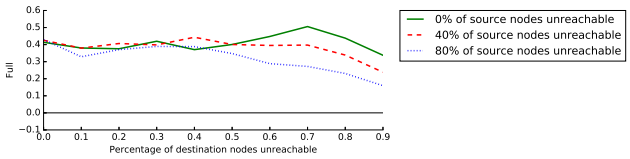
(a) Precision



(b) Recall



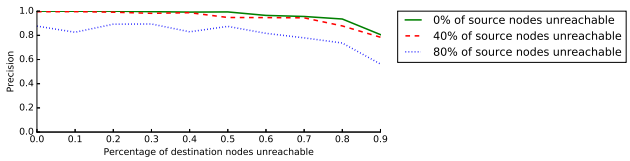
(c) $F_1 - score$



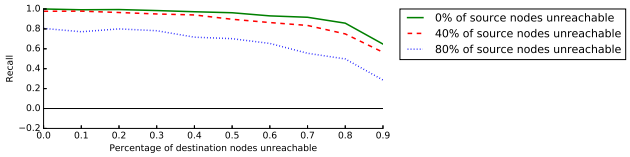
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.6: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP Synth50 when *Optimal substructure* is used

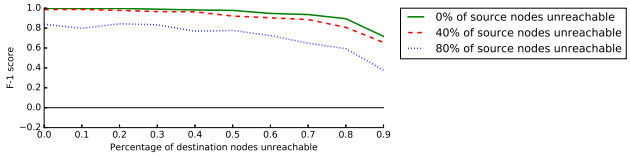
Synth100



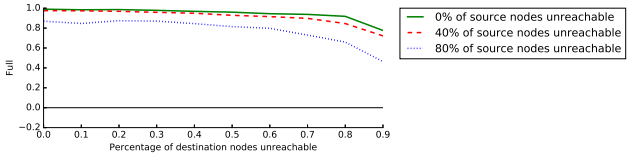
(a) Precision



(b) Recall



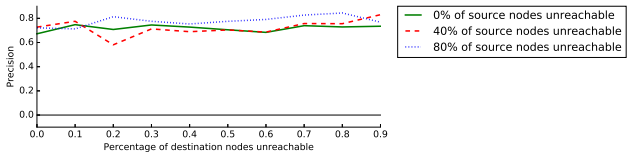
(c) $F_1 - score$



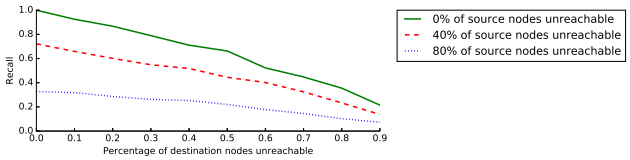
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.7: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP Synth100 when *Optimal substructure* is used

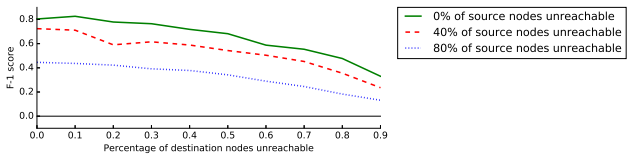
Telstra network



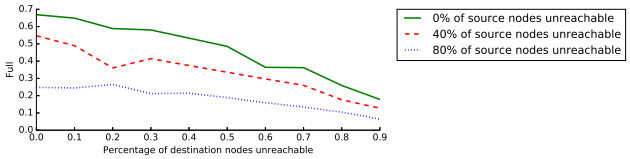
(a) Precision



(b) Recall



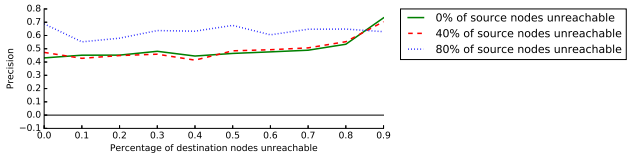
(c) $F_1 - score$



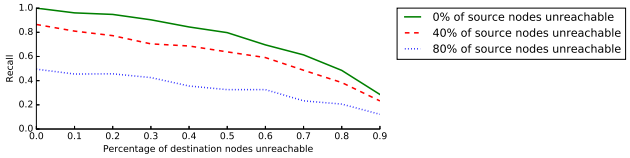
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.8: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Telstra* network when *Optimal substructure* is used

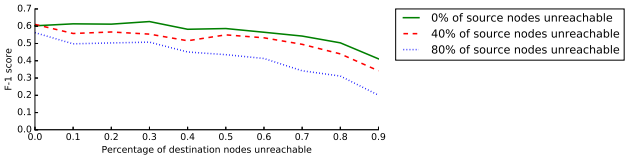
Ebone network



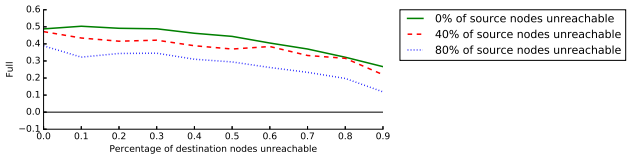
(a) Precision



(b) Recall



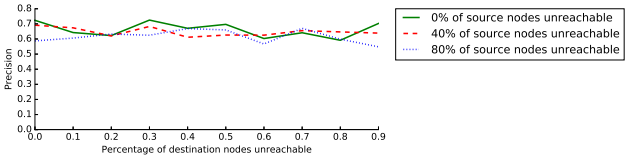
(c) $F_1 - score$



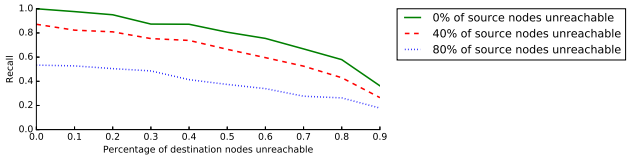
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.9: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Ebone* network when *Optimal substructure* is used

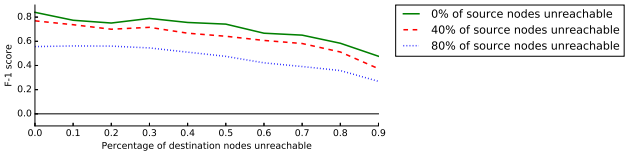
Exodus network



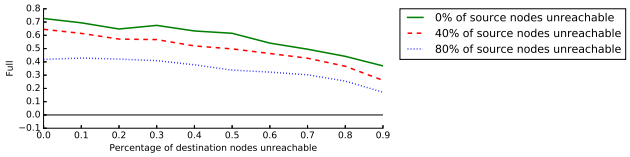
(a) Precision



(b) Recall



(c) $F_1 - score$

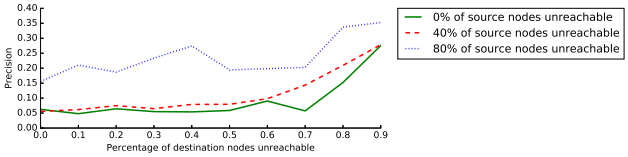


(d) The fraction of node-pairs between which the routing was fully characterized

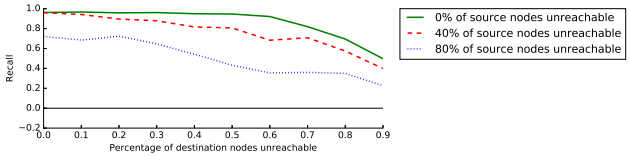
Figure A.10: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Exodus* network when *Optimal substructure* is used

A.1.3 When *Reverse path symmetry* hypothesis is made

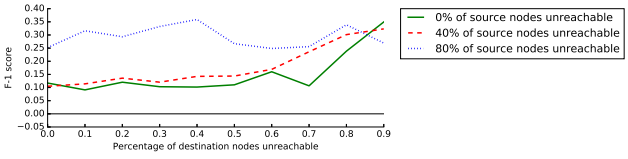
Synth50



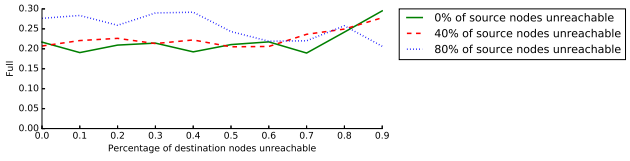
(a) Precision



(b) Recall



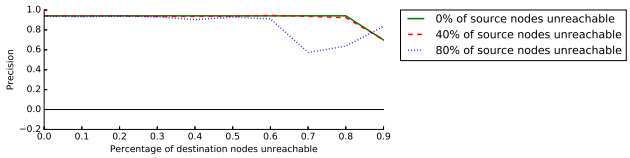
(c) $F_1 - score$



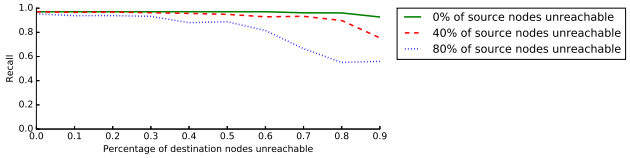
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.11: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP Synth50 when *Reverse path symmetry* is used

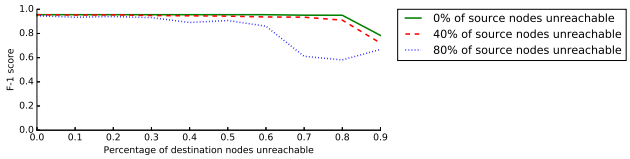
Synth100



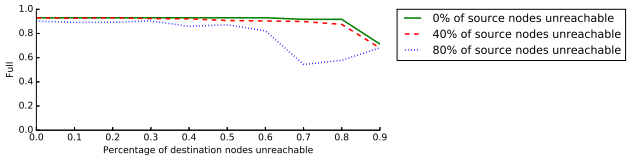
(a) Precision



(b) Recall



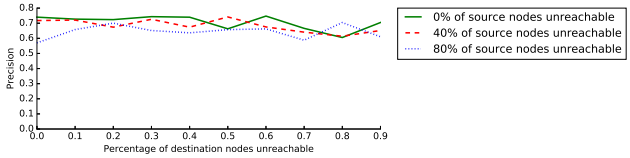
(c) $F_1 - score$



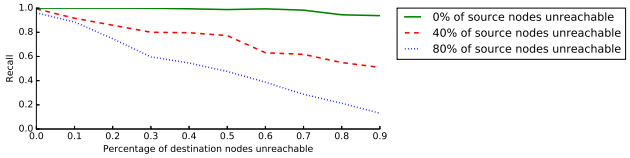
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.12: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP Synth100 when *Reverse path symmetry* is used

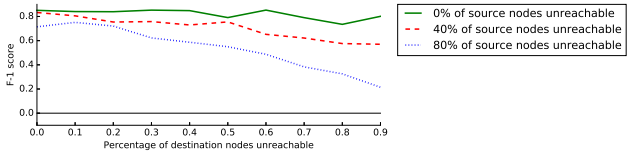
Telstra network



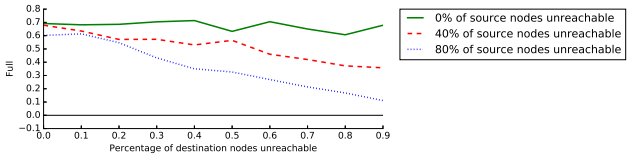
(a) Precision



(b) Recall



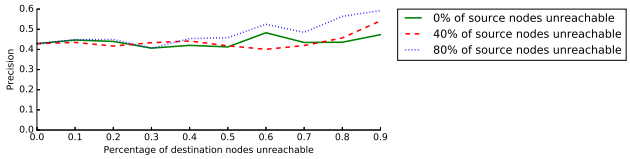
(c) $F_1 - score$



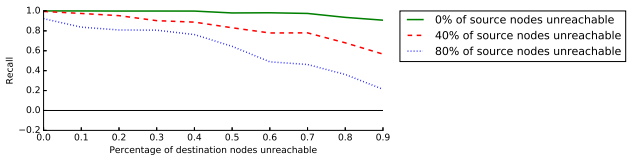
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.13: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Telstra* network when *Reverse path symmetry* is used

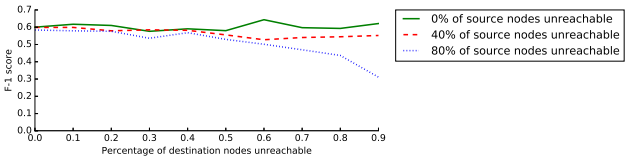
Ebone network



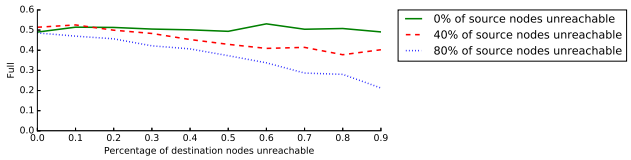
(a) Precision



(b) Recall



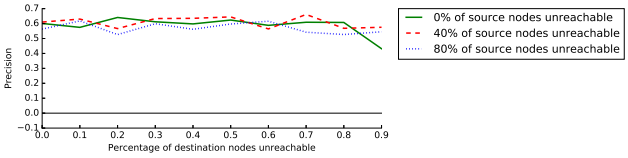
(c) $F_1 - score$



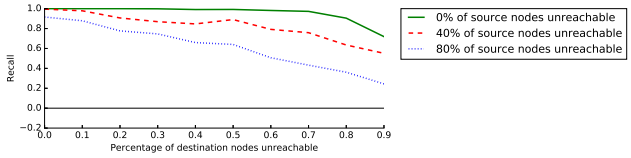
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.14: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Ebone* network when *Reverse path symmetry* is used

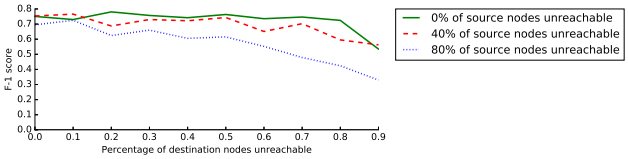
Exodus network



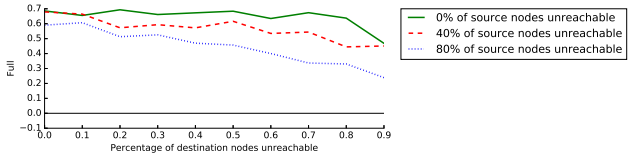
(a) Precision



(b) Recall



(c) $F_1 - score$

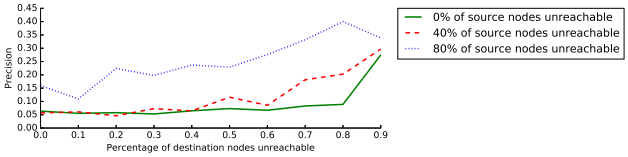


(d) The fraction of node-pairs between which the routing was fully characterized

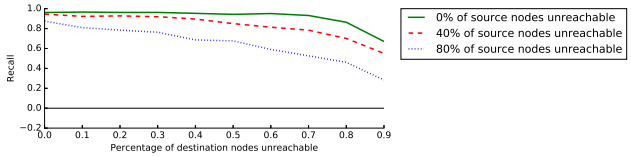
Figure A.15: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Exodus* network when *Reverse path symmetry* is used

A.1.4 When both *Optimal substructure* and *Reverse path symmetry* hypotheses are made

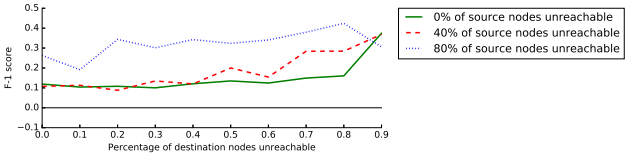
Synth50



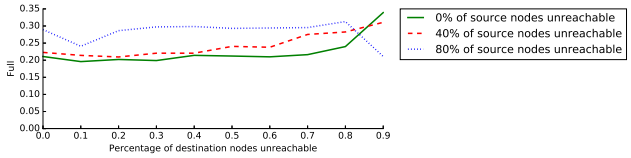
(a) Precision



(b) Recall



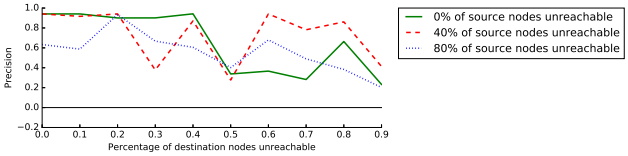
(c) $F_1 - score$



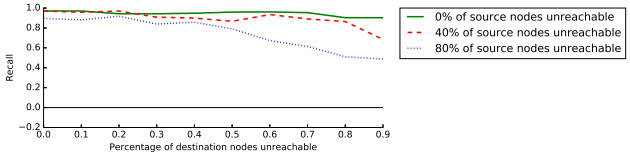
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.16: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP Synth50 when both *Optimal substructure* and *Reverse path symmetry* hypotheses are used

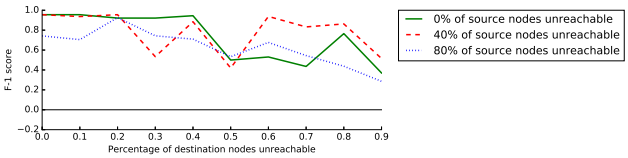
Synth100



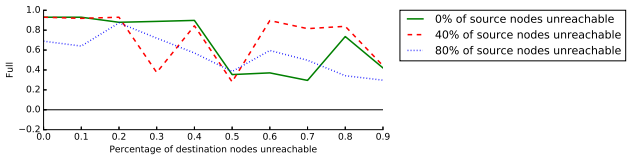
(a) Precision



(b) Recall



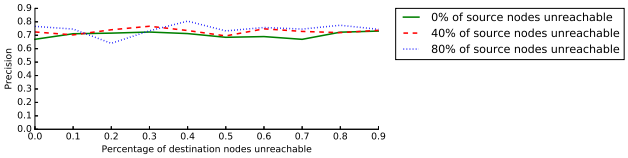
(c) $F_1 - score$



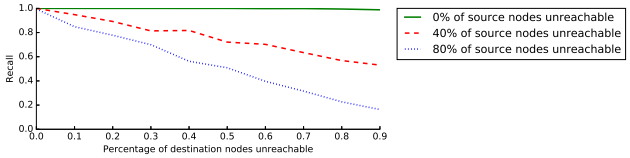
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.17: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for ISP Synth100 when both *Optimal substructure* and *Reverse path symmetry* hypotheses are used

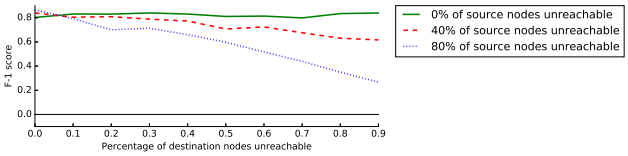
Telstra network



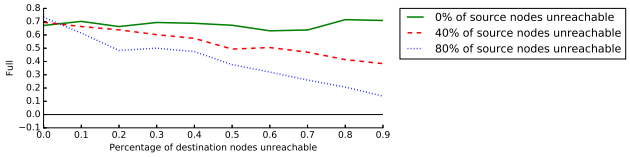
(a) Precision



(b) Recall



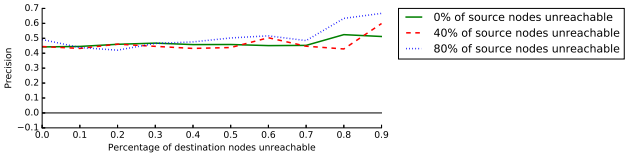
(c) $F_1 - score$



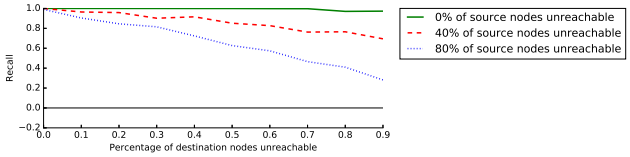
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.18: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Telstra* network when both *Optimal substructure* and *Reverse path symmetry* hypotheses are used

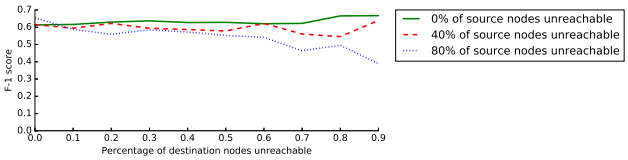
Ebone network



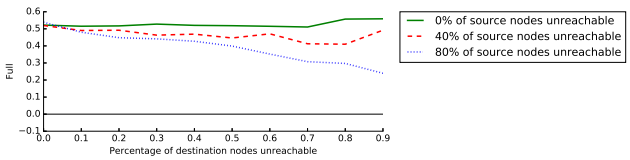
(a) Precision



(b) Recall



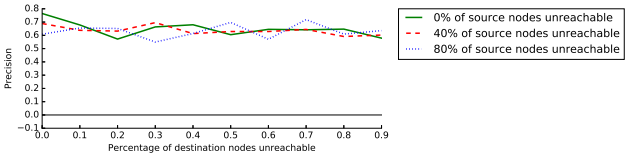
(c) $F_1 - score$



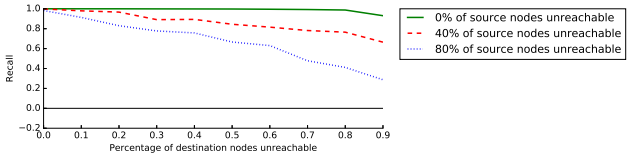
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.19: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Ebone* network when both *Optimal substructure* and *Reverse path symmetry* hypotheses are used

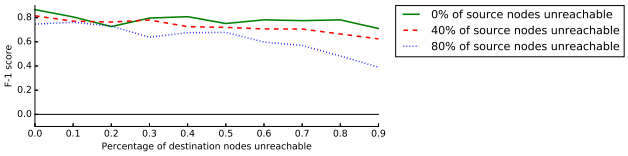
Exodus network



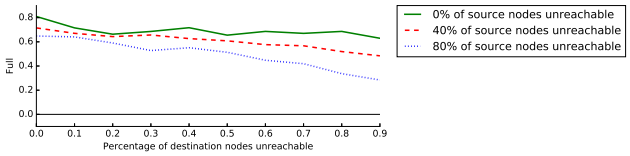
(a) Precision



(b) Recall



(c) $F_1 - score$



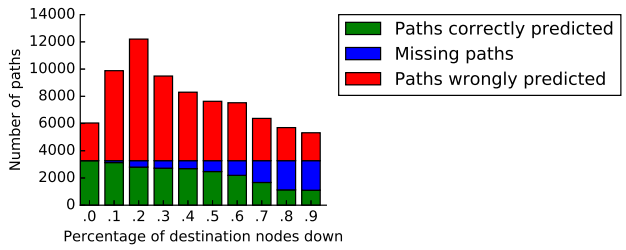
(d) The fraction of node-pairs between which the routing was fully characterized

Figure A.20: Precision, recall, $F_1 - score$ and the fraction of node-pairs between which the routing was fully characterized for the *Exodus* network when both *Optimal substructure* and *Reverse path symmetry* hypotheses are used

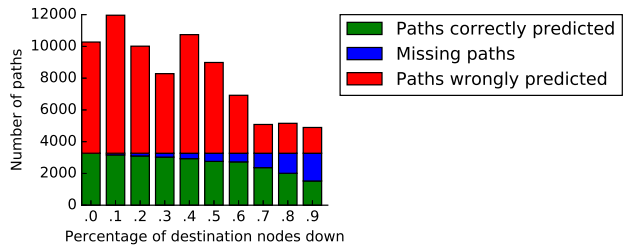
A.2 What causes bad precisions ?

In this section are contained all the figures associated to the question answered in Subsection 3.5.3.

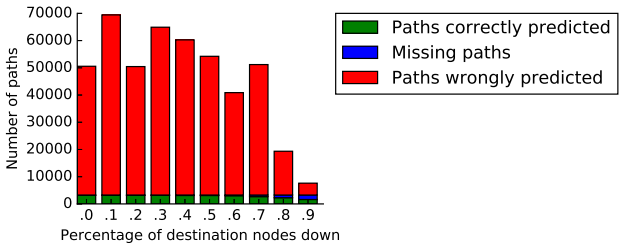
A.2.1 Synth50



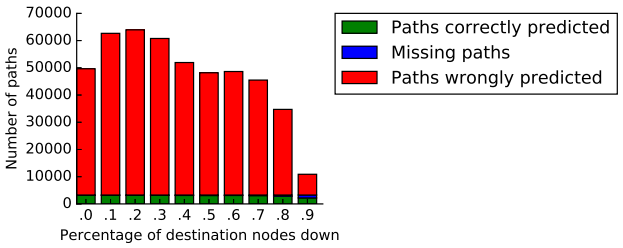
(a) When no hypothesis is made



(b) When *Optimal substructure* hypothesis is made



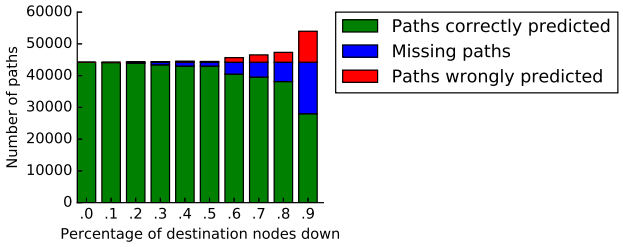
(c) When *Reverse path symmetry* hypothesis is made



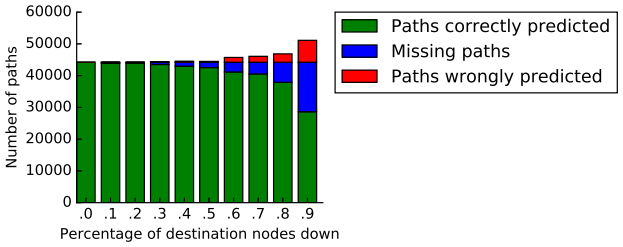
(d) When both *Optimal substructure* and *Reverse path symmetry* hypotheses are made

Figure A.21: Number of paths predicted on ISP Synth50 when all source nodes are available and the number of available destination nodes decreases

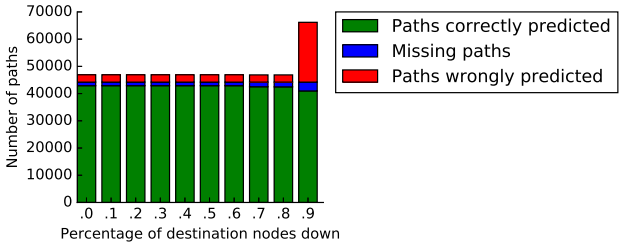
A.2.2 Synth100



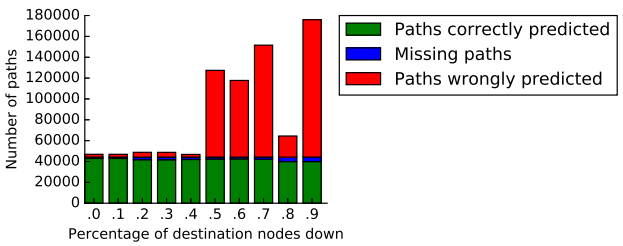
(a) When no hypothesis is made



(b) When *Optimal substructure* hypothesis is made



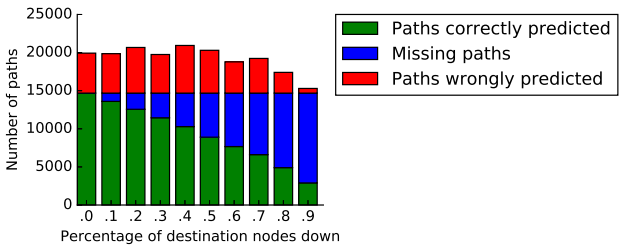
(c) When *Reverse path symmetry* hypothesis is made



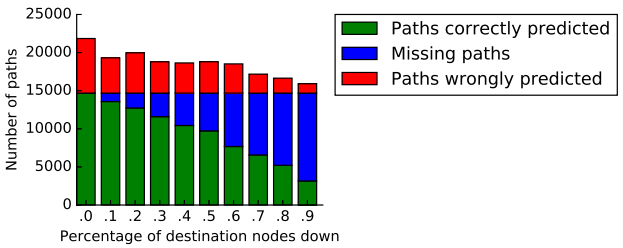
(d) When both *Optimal substructure* and *Reverse path symmetry* hypotheses are made

Figure A.22: Number of paths predicted on ISP Synth100 when all source nodes are available and the number of available destination nodes decreases

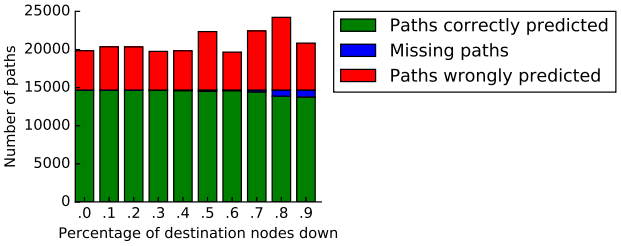
A.2.3 *Telstra* network



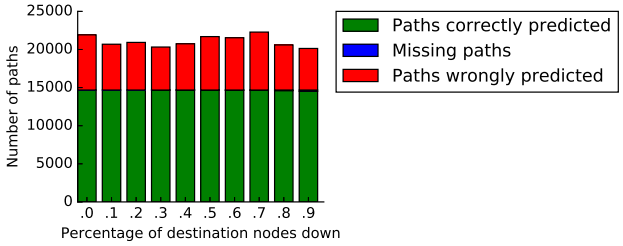
(a) When no hypothesis is made



(b) When *Optimal substructure* hypothesis is made



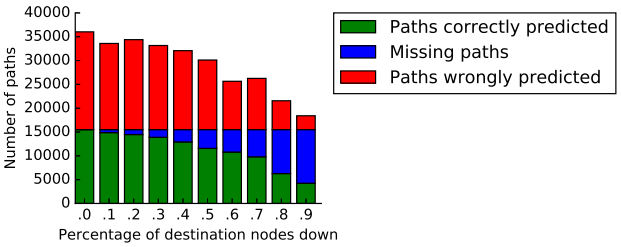
(c) When *Reverse path symmetry* hypothesis is made



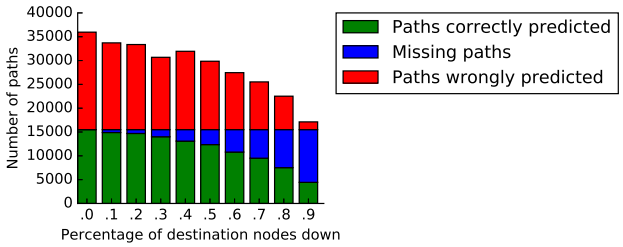
(d) When both *Optimal substructure* and *Reverse path symmetry* hypotheses are made

Figure A.23: Number of paths predicted on the *Telstra* network when all source nodes are available and the number of available destination nodes decreases

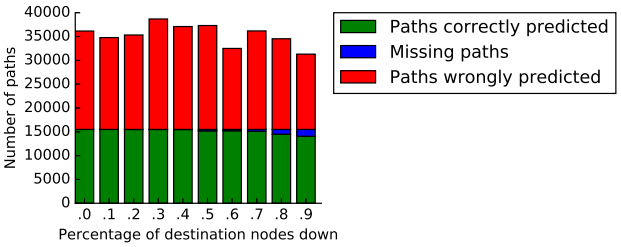
A.2.4 *Ebone* network



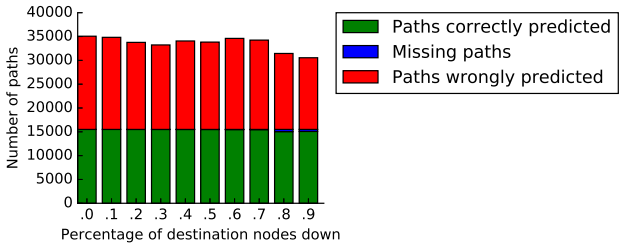
(a) When no hypothesis is made



(b) When *Optimal substructure* hypothesis is made



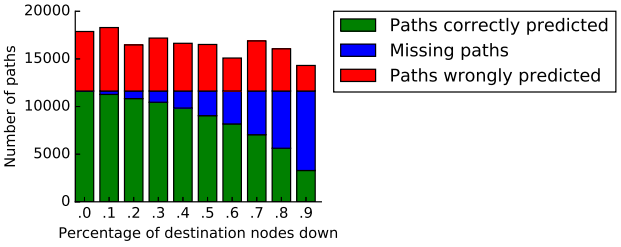
(c) When *Reverse path symmetry* hypothesis is made



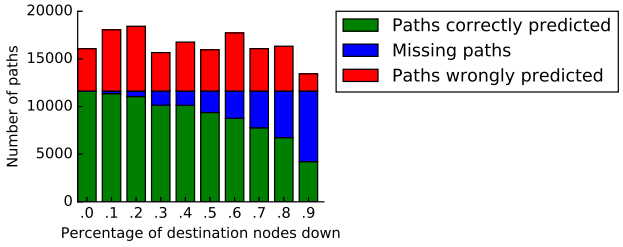
(d) When both *Optimal substructure* and *Reverse path symmetry* hypotheses are made

Figure A.24: Number of paths predicted on the *Ebone* network when all source nodes are available and the number of available destination nodes decreases

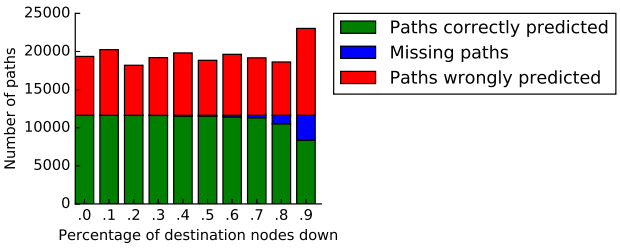
A.2.5 *Exodus* network



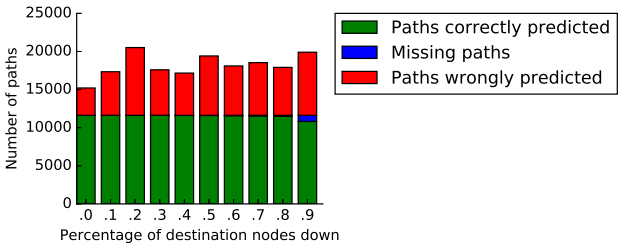
(a) When no hypothesis is made



(b) When *Optimal substructure* hypothesis is made



(c) When *Reverse path symmetry* hypothesis is made



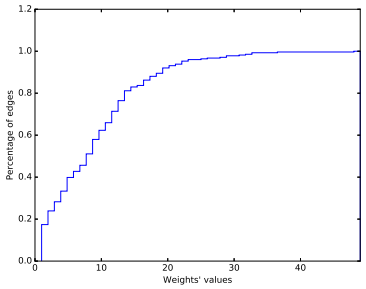
(d) When both *Optimal substructure* and *Reverse path symmetry* hypotheses are made

Figure A.25: Number of paths predicted on the *Exodus* network when all source nodes are available and the number of available destination nodes decreases

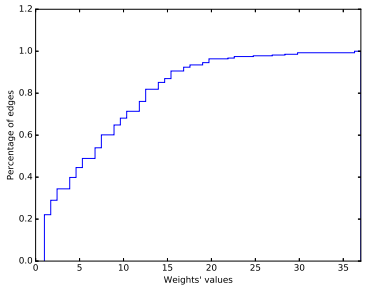
A.3 Are inferred weights comparable to the real ones ?

In this section are contained all the figures associated to the question answered in Subsection 3.5.4.

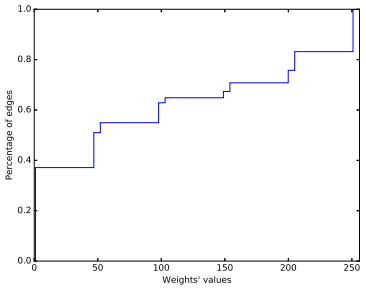
A.3.1 Synth50



(a) CDF of the real weights



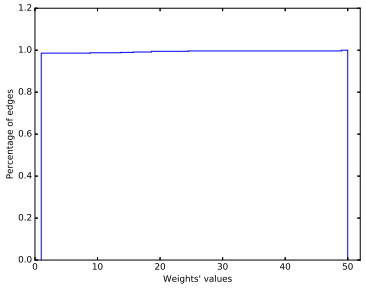
(b) CDF of the "perfect" weights



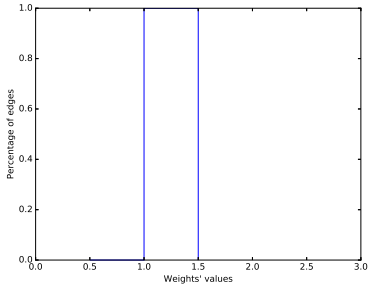
(c) CDF of the predicted weights

Figure A.26: CDFs of the weights for the ISP Synth50

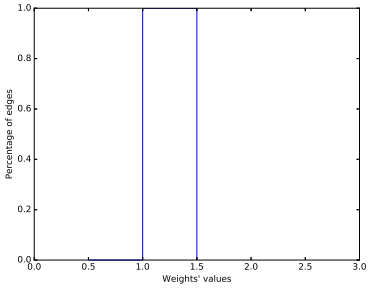
A.3.2 Synth100



(a) CDF of the real weights



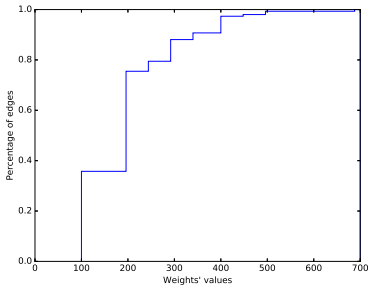
(b) CDF of the "perfect" weights



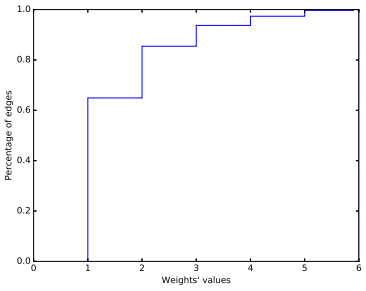
(c) CDF of the predicted weights

Figure A.27: CDFs of the weights for the ISP Synth100

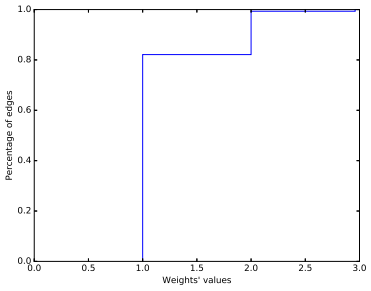
A.3.3 *Telstra* network



(a) CDF of the real weights



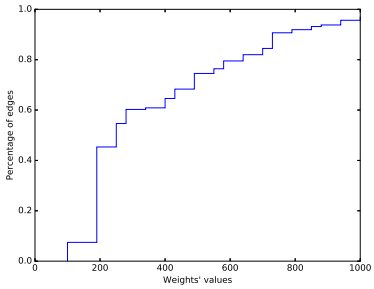
(b) CDF of the "perfect" weights



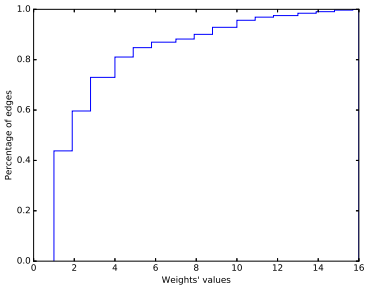
(c) CDF of the predicted weights

Figure A.28: CDFs of the weights for the *Telstra* network

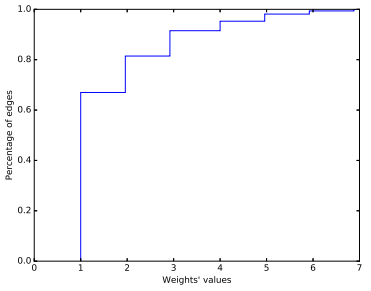
A.3.4 *Ebone* network



(a) CDF of the real weights



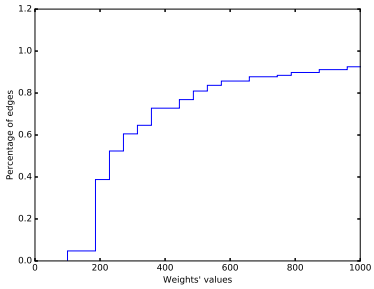
(b) CDF of the "perfect" weights



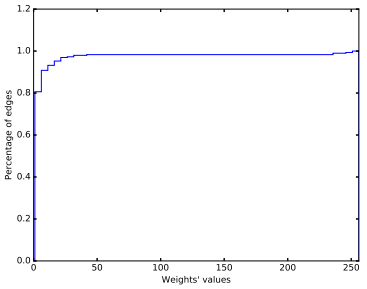
(c) CDF of the predicted weights

Figure A.29: CDFs of the weights for the *Ebone* network

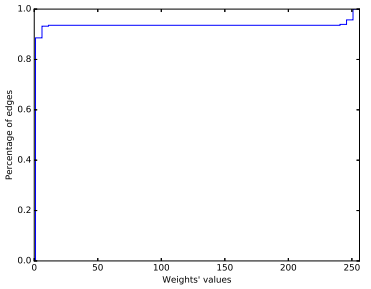
A.3.5 *Exodus* network



(a) CDF of the real weights



(b) CDF of the "perfect" weights



(c) CDF of the predicted weights

Figure A.30: CDFs of the weights for the *Exodus* network