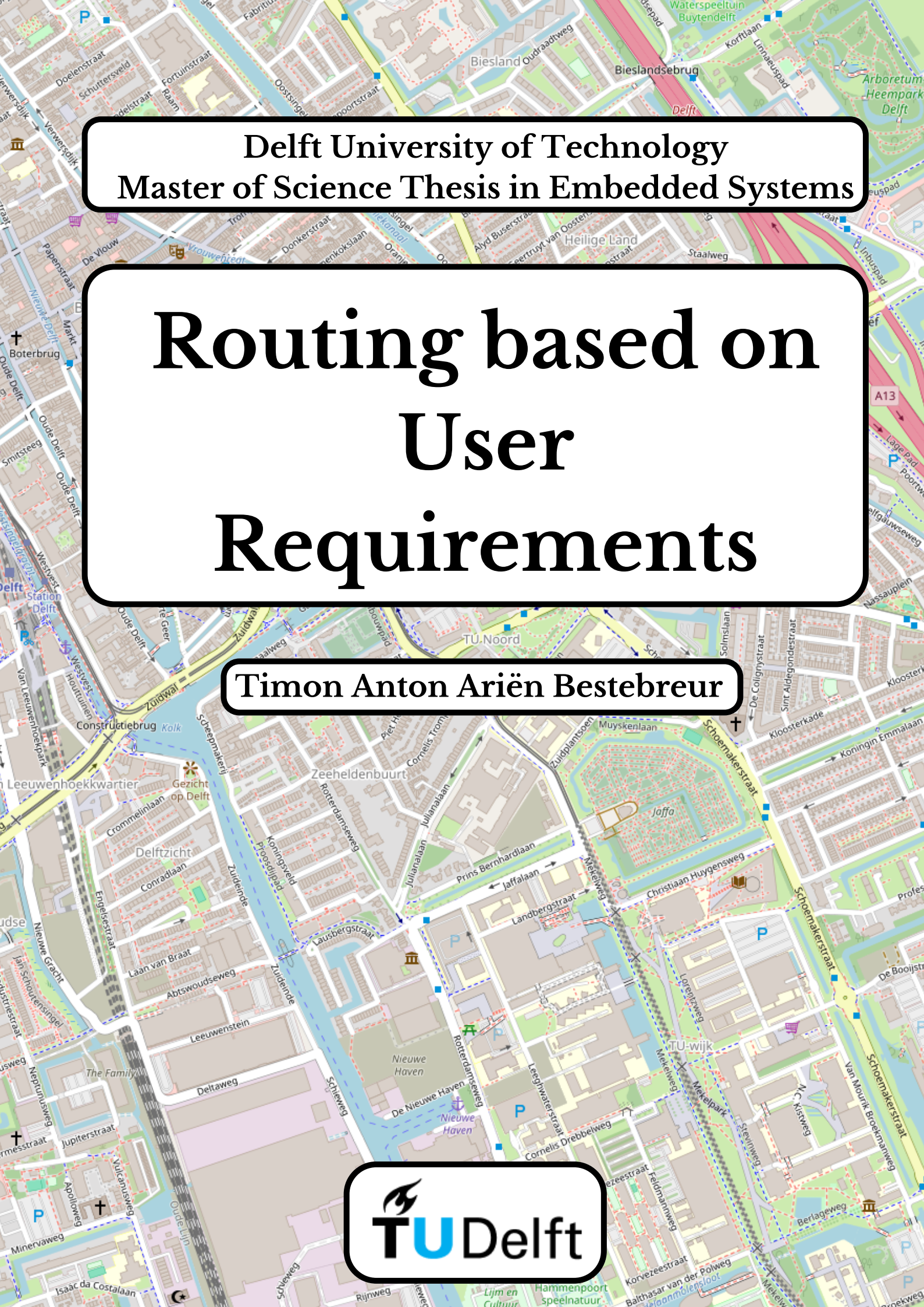
Delft University of Technology
Master of Science Thesis in Embedded Systems

Routing based on User Requirements

Timon Anton Ariën Bestebreur



Routing Based on User Requirements

Master of Science Thesis in Embedded Systems

Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

The map in the background of the cover page is provided by openstreetmap.com.

Timon Anton Ariën Bestebreur

26 January 2024

Author
Timon Anton Ariën Bestebreur

Title
Routing Based on User Requirements
MSc Presentation Date

8 February 2024

Graduation Committee

prof. dr. ir. Fernando Kuipers (chairman)	Delft University of Technology
dr. ir. Johan Pouwelse	Delft University of Technology
Adrian Zapletal	Delft University of Technology

Abstract

The versatility of the internet enables many applications that play an increasingly bigger role in our society. However, users have little control over the route that their internet traffic takes, which prevents them from controlling who sees their packets and how their traffic is handled. Researchers have proposed an extension to the internet, called the responsible internet, that aims to provide users with control over the route that their internet traffic takes.

Providing this control is the aim of this thesis. Users can control their route by specifying requirements that their route has to fulfill. This thesis defines the Maximum Path Requirement Intersection (MPRI) problem as the problem of finding the route that satisfies as many of the user's requirements as possible, and this thesis proves that MPRI is NP-hard. Subsequently, both a heuristic to solve the problem in a reasonable amount of time as well as an exact algorithm that guarantees to find the globally best path are introduced.

The performance of the heuristic is measured relative to the globally optimal solution given by the exact algorithm. Results show that less features allow the heuristic to have a larger search space, which improves the results; that the runtime of the heuristic scales polynomially in the number of hops between the start and end node; that the heuristic is most effective in graphs that have a power-law degree distribution and least effective in grid-like graphs; and that in a realistic setting the heuristic runs quickly while performing close to optimal.

"Show me your ways, Lord, teach me your paths." – Psalm 25:4

Preface

As this project comes to an end, it marks the end of a 20-year period of formal education. All the things I learned in pre-school, lower school, middle school, the bachelor of science, and the master of science, have built upon each other to get me where I am today; on the verge of graduation of the Master of Science in Embedded Systems of the prestigious university TU Delft. I am very excited about achieving this great feat, and I am very proud.

The last step of this journey was the Master's Thesis, which has been quite the project. It had its ups and downs, but in the end I am proud of the result. Now, let me say some words of gratitude.

Thank you Fernando, for your great feedback. Often it was great to hear, sometimes it was not fun to hear, but each time the thesis got better because of it. I learned a lot from your comments, which were almost always exactly on point.

Thank you Adrian, for being so available. Almost every Monday morning that I worked on the thesis, you were there to listen to my struggles and help me get to the next step. I appreciate the many hours of your time, and all the great feedback you gave on my writing.

Thank you Johan, for reading the report and being part of the thesis committee. I greatly appreciate the time and attention that you devoted to my graduation.

Thank you Ioana, for being such a great partner. You have encouraged me more times than I can count, and that has helped me tremendously. All those days together in the fish tank are memories I'll keep with me.

Thank you to mom and dad, for being proud of me and encouraging me along the way. Thank you to my brother, sisters, and friends, who have walked beside me as this project went along. I am grateful to have such a warm group of people around me.

Finally, thank you to God, for making us, and helping us create computers and all the wonderful things that they allow us to do. Every day I wake up, it's such a joy to know that God loves me, and I'm very grateful for that.

I have enjoyed being a student, and now I am ready for the next step! Ave Atque Vale!

Timon Anton Ariën Bestebreur

Delft, The Netherlands
26 January 2024

Acronyms & Terms

Notation	Description
AS	Autonomous System. 2, 5, 6, 7, 41, 42, 45, 46
BFS	Breadth-First Search. 3, 5, 10, 16, 41, 45
BGP	Border Gateway Protocol. 6
DAG	Directed Acyclic Graph. 45, 46
eBGP	External Border Gateway Protocol. 6
GUI	Graphical User Interface. 9
IANA	Internet Assigned Numbers Authority. 20
iBGP	Internal Border Gateway Protocol. 6
ID	identifier. 46, 47
ISD	Isolation Domain. 19, 45, 46
ISP	Internet Service Provider. 1, 2, 4, 5, 41, 42
IXP	Internet Exchange Point. 2, 5
MANRS	Mutually Agreed Norms for Routing Security. 9, 20
MPLS	Multiprotocol Label Switching. 6
MPRI	Maximum Path Requirement Intersection. iii, 3, 4, 9, 18, 26, 38, 47, 48, 49
MPRI	Maximum Path Requirement Intersection. 7, 8
MSI	Maximum k-Subset Intersection. 8
NCP	Network Control Plane. 2
Nile	Network Intent LanguagE. 44, 45
NIP	Network Information Plane. 2, 4, 9, 41, 42, 45, 49
NLP	natural language processing. 44
ONOS	Open Network Operating System. 45
P4	Programming Protocol-independent Packet Processors. 45
PAN	Path-Aware Networking. 46
PCB	Path Construction Beacons. 45
PCFS	Packet Carried Forwarding State. 45
PSR	Private Source Routing. 46, 47
QoS	Quality of Service. 6, 44
RPKI	Resource Public Key Infrastructure. 2
SCION	Scalability, Control, and Isolation On Next-Generation Networks. 19, 44, 45, 46
SDN	Software-Defined Networking. 45
SeR	Segment Routing. 6
SoR	Source Routing. 6, 46, 47
UCIP	User Controlled Internet Protocol. 46, 47

Contents

Chapter 1

Introduction

The whole world is interconnected via the Internet. This connectivity enables many applications, and the roles that these applications play in our society continuously grow in importance [30, 43]. Meanwhile, users have little control over and insight into how the Internet is operated. There are efforts to improve transparency and provide insights into various aspects of the Internet[1] [24, 45, 64, 82], but more can be done to improve control.

1.1 Motivation

One area where this lack of control is prevalent, is control over the path that traffic takes through the Internet. Internet traffic that passes through routers located in a certain country may be processed according to the laws of that country, allowing governments to keep track of the information that is flowing through their country. This permit results in the collection and analysis of Internet traffic that passes through, for example, the US[2] and collection and analysis of Internet traffic that passes through Russia [28]. If a citizen of the European Union does not like that the US and Russia (and other countries) collect and inspect Internet traffic that passes their borders, and if this citizen wants to ensure that their Internet traffic remains within the European Union to ensure that it remains under the General Data Protection Regulation, they currently have no way to do so.

Thus, users may want to have control over the countries through which their Internet is routed. However, control over the geographical areas that Internet traffic goes through is only one part. Some Internet Service Providers (ISPs) are more careful with their customer data than others[3]. If customers want their Internet paths to include only include ISPs that did not have a data breach in, say, the last 10 years, then customers currently have no way request that. Further, some ISPs gather more information than is strictly needed to provide their Internet services [17]. If users want to ensure that their path avoids these ISPs, they need a way to gain control over their Internet paths.

In contrast, if users *did* have control over the path of their Internet traffic, users could, among other things, control what geographical areas are avoided by their traffic, place demands on the security practices of the companies that handle their traffic, and decide how privacy-friendly they want the ISPs in their Internet path to be. In addition, companies would be able to specify security demands for their Internet traffic, allowing them to, for example, rest assured that traces of their communications with other companies are not intercepted or leaked. Giving control over Internet paths benefits ISPs as well, as ISPs could distinguish themselves by supporting features that users demand, which leads to more traffic due to people sending their Internet traffic via these innovative ISPs. All of the above is only possible if users can decide which path their Internet traffic takes. This requirement makes control over the path of Internet traffic a relevant problem to address.

Systems that allow users to define their own internet path already exist [19, 49, 72, 76, 81, 84]. However, these systems make two assumptions. First, they assume that the user has all the information about the network that is needed to determine which path they want. Second, they assume the user has sufficient knowledge about how the internet works to make an informed decision about what the best path is, since the work of actually choosing the best path is pushed to the user. The first assumption excludes users that do not have the required network

[1] For efforts that improve transparency, see `https://isbgpsafeyet.com`, `https://Internet.nl`, `https://www.peeringdb.com`, `https://www.waveform.com/tools/bufferbloat`, `https://www.email-security-scans.org/`, and `https://cands.cc.gatech.edu/blog/what-does-it-take-fcc-challenge/`.

[2] `https://www.propublica.org/article/nsa-data-collection-faq`

[3] Some ISPs have had data breaches, as described in the following news articles: `https://www.bitdefender.co.uk/blog/hotforsecurity/data-breach-canadas-fitness-depot-blames-isp-for-security-incident/`, `https://www.zdnet.com/article/data-breach-at-russian-isp-impacts-8-7-million-customers/`, `https://www.theverge.com/2023/9/20/23881825/t-mobile-account-security-breach-customer-information-leak`, and `https://www.zdnet.com/article/pocket-inet-isp-exposed-73gb-of-data-including-secret-keys-plain-text-passwords/`.

information to choose a path, and the second assumption excludes less technical users that do not understand the internet well enough to make an informed choice. Instead, this thesis assumes the user has no network information, and this thesis places no requirements on the level of knowledge of the user about the internet. The user only needs to provide the requirements for their path, and our system takes care of the rest. These assumptions make the system in this thesis more approachable and usable, and provides more users with control over their internet path.

1.2 Responsible Internet

The goals of this thesis align with the goals of the responsible Internet [39]. The responsible Internet is a proposed secure-by-design extension of the Internet that aims to improve transparency, control, and accountability in the Internet. The vision for the responsible Internet consists of three components: the Network Information Plane (NIP), the Network Control Plane (NCP), and policies to govern the way we interact with the Internet. The NIP provides information about the network to users, which increases transparency. The NCP provides users with control over how their data is handled, which increases control. Finally, the responsible Internet allows users to verify that the information they receive about the network is accurate, which increases accountability.

One aspect of the responsible Internet is to design and evaluate a method to program NCP requests into the underlying open infrastructure. The work of this thesis aligns with that part, and the work of this thesis can be used to improve control. While this thesis does not contribute to the NIP, the NIP *does* play an important role in this thesis because the NIP can provide information about what features are supported by each ISP, which allows users to control which of these ISPs to avoid or include in their path.

1.3 Overview

The goal of this thesis is calculating a path through the Internet on the level of Autonomous Systems (ASs) that best fits the preferences of the user that requests the path. ASs are groups of routers that are managed by one party, which can be an ISP, an Internet Exchange Point (IXP), or a large organization.

These user requirements correspond to features that can be supported or not supported by ASes. By including a certain requirement in the list of requirements, a user indicates that they wish that all ASs on the path via which their Internet traffic travels support the feature that corresponds to that requirement. This requirement can relate to a security feature (e.g., each AS on my path should support Resource Public Key Infrastructure (RPKI)), a privacy feature (e.g., each AS on my path should notify customers of data breaches), a geolocation-based feature (e.g., each AS on my path should be located within the European Union), or anything else. To give an idea of the type of features that a user might request to be fulfilled by their path, Appendix B contains a potential list of features that the user might want to request.

The big picture idea is to use the information from the NIP to gain insight into what features each AS supports, and make the list of available features transparent to the user. The user can then select the features they deem important, and provide the path request to the algorithm proposed in this thesis. Our algorithm then attempts to find a path that fulfills the given requirements, and returns it to the user if successful. The user can then use that path to route their traffic according to their wishes.

Requirements can be placed in two categories: strict and best effort. Strict requirements are requirements that are non-negotiable. If an AS does not support any of the strict requirements, it should under no circumstance be part of the resulting network path. For the requirements in the best effort category, our system tries to find a path that satisfies as many requirements as possible, but it does not guarantee that the selected path satisfies any best effort requirement.

1.4 Problem and Solution

The sub-problem of finding the path that satisfies as many best effort requirements as possible is defined by us, and we call it the the Maximum Path Requirement Intersection (MPRI) problem. The MPRI problem originates from the context of AS-level Internet routing, but MPRI is more widely applicable. For example, when one wants to find a car highway route that avoids toll roads, always has nature next to the road, and always uses silent asphalt, one can encode these elements as features of the road segments, encode the wishes for the roads as requirements, and find the route for which all nodes satisfy as many requirements as possible. The same can be done to find, for example, routes through a city that avoid bad neighbourhoods and use walkable areas, or to find paths for robots that always keep the robots' solar panels exposed to the sun. Thus, solving MPRI provides a building block towards solving multiple relevant problems, on top of enabling control over Internet paths.

After introducing the MPRI problem, we prove that MPRI is NP-hard. This NP-hardness creates the need for a heuristic algorithm that provides relatively good answers in little time. We introduce such a heuristic algorithm in Chapter 4. The main mechanic that allows the heuristic to return reasonably good answers in little time is a highly configurable set of limits that controls the search space. We prove that the heuristic always finds the best possible path out of all paths within the limits given to the heuristic, where the best path is the path that satisfies the maximum amount of best effort requirements.

The main idea of the heuristic is to initially find the shortest path from source to destination for which all nodes comply with the strict requirements, and then to identify "bottlenecks" in the path. In the context of this thesis, bottlenecks are nodes that reduce the number of requirements that are fulfilled by all nodes in the path, because the bottleneck node fulfills less or other requirements than the other nodes in the path. For these bottlenecks, the heuristic calculates detours around the bottleneck, and updates the path with a detour if replacing the bottleneck with the detour results in a path for which all nodes satisfy a larger shared set of features than the path with the bottleneck. The strength of the heuristic is in its configurability. The heuristic is built around being configurable in the size of its search space, which allows users to customize it for the type of graph and the time they are willing to wait. This configurability allows for a trade-off between speed and accuracy; if the user is willing to wait, the heuristic is able to produce results that are closer to the globally optimal result, while a less patient user may choose to be satisfied with results of lower quality and a faster runtime.

The performance of the heuristic is primarily measured in the number of best effort requirements it is able to fulfill for a given path request, which we refer to as the score of a path. The score of the path found by the heuristic is compared to the score of the globally best solution. To find this globally best solution, a global Breadth-First Search (BFS) algorithm is introduced. Next to that, we prove that the globalBFS algorithm is guaranteed to always return the globally best path.

1.5 Evaluation

The main goal of the evaluation is to answer the following questions.

1. What is the influence of the number of features and requirements on the performance of the heuristic?
2. What is the influence of the ratio between minimum and maximum number of features supported by each AS on the performance of the heuristic?
3. How well does the heuristic scale?
4. On what type of graph topology is the heuristic most effective?
5. How well does the heuristic perform in a realistic scenario?

These questions are evaluated by five experiments, each of which answers one of the questions. Our results inform five main findings. First, less features and requirements increase the speed of the heuristic. Second, the ratio between minimum and maximum features has little influence

on the performance of the heuristic. Third, the runtime of the heuristic scales polynomially in the length of the shortest path, which indicates that the heuristic is most effective in graph types where the average shortest path does not grow with the graph size. Four, the heuristic is most effective on graphs that have a power-law degree distribution and least effective on grid-like graphs, Five, in a realistic setting (which is a full size graph of the Internet and 25 requirements per request) the heuristic fulfills on average 98% of the best effort requirements of the globally optimal solution in less than 40 ms per request.

1.6 Main Contributions

The main contributions of this thesis are:

1. The introduction of the MPRI problem and its application in the domain of AS-level Internet routing.
2. The proof that the MPRI problem is NP-hard.
3. A heuristic algorithm for the MPRI problem that, on average, satisfies 98% of the best effort requirements of the globally optimal solution in less than 40 ms per request.

1.7 Structure

The rest of the thesis is structured as follows. Chapter 2 covers the relevant background knowledge. Chapter 3 defines the problem and proves its NP-hardness. Chapter 4 details the design of the heuristic algorithm and the global BFS algorithm, their related proofs, implementation details, and their deployment in the internet. Chapter 5 contains the experimental setup, the results, and the discussion of the results. Chapter 6 discusses how ISPs can be incentivized to provide accurate information to the NIP. Chapter 7 goes into the related work, and finally, Chapter 8 draws conclusions and points to potential directions for future work.

Chapter 2

Background

This chapter presents the relevant background knowledge. First, it discusses pathfinding, which is the theory of finding a path from source to destination through a network (Section 2.1). After that, the chapter explains Internet routing, which applies pathfinding to the context of the Internet (Section 2.2).

2.1 Path Finding

When determining the path from a source to a destination through a network, it is helpful to model the network as a graph, consisting of nodes and links connecting nodes to each other. See [29] for definitions of graphs, nodes and links. Note that nodes can also be called vertices, and links can also be called edges. Finding a path through a network essentially consists of starting at some source node and using the links to explore adjacent nodes until the destination is reached. Path finding techniques differ in the order in which these adjacent nodes are explored.

An algorithm to find paths in a network is Breadth-First Search (BFS). This technique is used in both the heuristic algorithm (the fast algorithm) and in the globalBFS algorithm (the algorithm that finds the globally optimal solution). BFS starts with the source node, explores all first-level neighbours (all nodes that can be reached from the source node using one link) and checks whether any of the newly explored nodes is the destination node. If not, BFS explores all second-level neighbours and checks whether any of these nodes is the destination. This exploration continues with the third level neighbours, fourth level neighbours, etcetera, until the destination is found. Nilsson showed that BFS always finds a path from source to destination with the minimum number of links [61], which makes it a suitable algorithm if the goal is to find the path with the minimum number of links. Other algorithms are better suited if the links have weights and one wants to find the path with the minimum weight, but these algorithms are not used in this thesis.

2.2 Internet Routing

When two systems want to communicate via the Internet, their data needs to be transmitted from the source to the destination[1]. Internet routing is the process of determining the route via which this communication happens. The devices that handle these packets are referred to as routers [4], and the Internet consists of a large network of interconnected routers. Packets flow from router to router until they reach their destination.

The transport of packets starts by one computer that outputs its packets on a certain port, which connects the computer to a router. The router that receives the packet then looks at the destination address of the packet, determines to which port it should forward the packet for it to reach the destination, and outputs the packet on that port. The packet is received by the next router and forwarded again. This forwarding process continues until the packet arrives at its destination.

2.2.1 Autonomous Systems

With 29.3 billion networked devices in 2023 [73], calculating routes from any device to any other device is challenging. To make this problem more tractable, the Internet is split up into large groups of routers called Autonomous Systems (ASs) [75]. These ASs can be Internet Service Providers (ISPs), Internet Exchange Points (IXPs), or large organizations.

Internet routing is divided into two levels: routing within one AS, and routing from one AS to another. If a computer in one AS wants to reach a computer in another AS, its packet goes

[1] Internet traffic is not restricted to one source and one destination. Data can also be sent from one source to multiple destinations, like in Multicast [51].

through multiple levels. First, the computer sends its packets to the edge of the source AS. Then, the packet is routed from the edge of the source AS to the edge of the destination AS via routing on the level of ASs. Finally, the packet is routed from the edge of the destination AS to the destination computer[2].

2.2.2 Border Gateway Protocol

Currently, the Internet uses Border Gateway Protocol (BGP) for routing between ASes. BGP enables ASs to reach each other by defining how reachability information is exchanged between ASs [69]. Specifically, BGP facilitates the exchange of information about the route via which an AS can reach its neighbours. Each AS combines this information in a large routing table. This table is propagated through the network, such that all ASs know how to reach the other ASs. When data needs to be sent to another AS, one of the available paths is selected and used.

Path selection happens based on policies. Some example policies are shortest path, earliest received path, or personal preference of the network operator [13]. BGP works for routing within one AS (Internal Border Gateway Protocol (iBGP)) as well as routing between multiple ASs (External Border Gateway Protocol (eBGP)). This flexibility makes BGP a versatile protocol.

2.2.3 Source Routing

In traditional Internet routing, the packets are at the mercy of the routing policy in terms of the route that the packet takes through the network. If one wants to explicitly define what route a packet should take, then that person is out of luck; the traditional routing approach does not provide a mechanism to do so.

A mechanism that does allow one to specify the path a packet should take is Source Routing (SoR) [65]. In SoR, packets carry a list of locations via which they should be routed, in addition to their destination address. Routers that support SoR heed this information, and thus the packets follows the route as defined by the sender.

2.2.4 Segment Routing

Segment Routing (SeR) is a mechanism that enables network packets to carry a list of instructions, called segments, which are executed by the routers that process the packets [32]. Segment routing is similar to source routing, as the segments can consist of the specific path along which the packet should travel towards the destination. However, segments are more versatile. A segment can also contain instructions that specify by which virtual machine that packet should be processed or which Quality of Service (QoS) treatment it should receive.

A segment can be encoded as a Multiprotocol Label Switching (MPLS) label. MPLS is a technique that enables routing by label, instead of by destination address, which enables overriding the default routing protocol [77]. This routing by label is possible because the labels are processed before the destination address. Because of this implementation, SeR can be used by all routers that support MPLS. On top of that, SeR can be used with IPv6 as well, which makes it even more versatile.

[2]The work in this thesis focuses on routing between multiple ASs.

Chapter 3

Problem

This chapter describes the problem statement for this thesis. The chapter starts by explaining concepts and definitions used in the problem statement. Then the chapter describes the problem and its complexity, including the proof that the problem is NP-hard.

3.1 Concepts and Definitions

Loosely worded, the problem consists of finding a path through a network that adheres to the user's requirements. In the context of this thesis, the nodes of the network are Autonomous Systems (ASs), the links that connect the nodes are Internet links between ASs, and the AS-level **path** that we are searching for consists of a sequence of ASs connecting the source AS to the destination AS. However, the nodes of the network can also represent other entities than ASes, and the network links can represent something else than Internet links.

Requirements are features that nodes on a path can support. These features can be security features, privacy features, or any other feature that the user deems important. If all nodes in a path support a set of features that the user requires, we say that this path satisfies these user requirements. Requirements are split up into strict requirements and best effort requirements.

Strict requirements are requirements that should always be satisfied by all nodes in the path. **Best effort requirements** are requirements that do not need to be satisfied. If a path satisfies all strict requirements and satisfies no best effort requirements, the path can still be returned to the user. However, the goal is to find a path that satisfies all strict requirements and as many best effort requirements as possible. In other words: If there are two paths that both satisfy the strict requirements, but the first path satisfies more best effort requirements than the second path, the first path should be returned to the user.

For this thesis, the requirements given by the user are restricted to binary requirements, where a requirement can either be completely fulfilled or not fulfilled at all. Thus, requirements cannot be partially fulfilled. This restriction has little influence on the applicability, as non-binary requirements can be modeled as binary requirements: If a user wants all ASes on their Internet path to support feature X for at least 60 %, this requirement can be modeled as a binary requirement, where all ASes that do not support feature X for at least 60 % are ignored when calculating paths. Thus, for this thesis, the models assume purely binary requirements, but with some small adjustments, this system can be extended to support non-binary requirements as well.

3.2 Problem

The problem that this thesis aims to solve is the following.

Problem 1 *Given a network, a source node v_s, a destination node v_t, information about what features are supported by which node, and user requirements consisting of a list of strict requirements and a list of best effort requirements, find a path from source to destination for which all nodes on the path support all strict requirements and the path satisfies as many best effort requirements as possible.*

3.3 Complexity

The best effort part of this problem is NP-hard. To clearly separate this best effort sub-problem from the main problem, it is renamed to the Maximum Path Requirement Intersection (MPRI) problem, defined as follows.

Problem 2 *Given a graph $G = (V, E)$ and a finite set of features $F = \{f_1, \ldots, f_g\}$, where each vertex $v_i \in V$ contains a subset $F_i \subseteq F$, and given a set of best effort requirements $R \subseteq F$,*

a start vertex $v_s \in V$ and an end vertex $v_t \in V$, find a path $P = (v_s, \ldots, v_t)$ such that the intersection of the feature subsets contained in the vertices of the path intersected with the set of best effort requirements R is maximal in size. Thus, the goal is to find a path such that $\mid (F_s \cap \cdots \cap F_t) \cap R \mid$ is of maximum size.

MPRI is NP-hard. The proof for this statement defines a polynomial-time Turing reduction from the NP-hard Maximum k-Subset Intersection (MSI) problem [80] to the MPRI problem. The MSI problem is defined as follows.

Problem 3 *Given a collection $C = \{S_1, \ldots, S_m\}$ of m subsets over a finite set of elements $Q = \{q_1, \ldots, q_n\}$, and a positive integer k, the objective is to select exactly k subsets $S_a, \ldots, S_x$ whose intersection size $|S_a \cap \cdots \cap S_x|$ is maximum.*

Theorem 1 *MPRI is NP-hard.*

Proof. The Turing reduction from the MSI problem to the MPRI problem works as follows. Given an instance (C, Q, k) of MSI, construct an instance for MPRI by creating one node v_s, one node v_t, and create for each subset $S_i \in C$ one node v_i. Then set $F = R = Q$, assign to node v_s the subset $F_s = Q$, and to node v_t the subset $F_t = Q$. Assign to each node v_i the subset $F_i = S_i$. For edges, connect v_s to each node v_i (for all nodes v_i that correspond to the subsets $S_i \in C$), connect each node v_i to each other node v_j such that the nodes v_i corresponding to the subsets $S_i \in C$ form a clique, and finally connect each node v_i to node v_t.

Considering the construction above, we claim that for any collection of k subsets $S_a, \ldots, S_x$ whose intersection size $|S_a \cap \cdots \cap S_x|$ is maximum, there exists a set $(F_s \cap \cdots \cap F_t)$ in the corresponding instance of MPRI that maximizes $\mid (F_s \cap \cdots \cap F_t) \cap R \mid$. Let $P_{max} = (F_1, \ldots, F_p)$ be the path that maximizes the intersection of the subsets $F_1, \ldots, F_p$. Since $R = F$, the optimization objective $\mid (F_s \cap \cdots \cap F_t) \cap R \mid$ can be reduced to $\mid (F_s \cap \cdots \cap F_t) \mid$. Next, each of the subsets $S_a, \ldots, S_x$ corresponds to a node in the graph. Since all nodes corresponding to these subsets are connected, any subset of these nodes can form a path. Since v_s and v_t are connected to all nodes corresponding to the subsets $S_a, \ldots, S_x$, the path through the nodes corresponding to $S_a, \ldots, S_x$ can on one end connect to v_s and on the other end connect to v_t, making it a path that connects v_s to v_t. Finally, since the subsets $S_a, \ldots, S_x$ are the subsets that result in an intersection of maximum size, the subsets that are assigned to the nodes of the path are also the subsets that result in an intersection of maximum size. Thus, for any collection of k subsets $S_a, \ldots, S_x$ whose intersection size $|S_a \cap \cdots \cap S_x|$ is maximum, there exists a set $(F_s \cap \cdots \cap F_t)$ in the corresponding instance of MPRI that maximizes $\mid (F_s \cap \cdots \cap F_t) \cap R \mid$.

On the other hand, we claim that for any path $P = (v_s, \ldots, v_t)$ that maximizes $\mid (F_s \cap \cdots \cap F_t) \cap R \mid$, there is a selection of k subsets $S_a, \ldots, S_x$ whose intersection size $|S_a \cap \cdots \cap S_x|$ is maximum. If we remove v_s and v_t from P, then the feature subsets of the vertices in the path result in an intersection of maximum size, which also corresponds to the subsets $S_i \in C$ that have an intersection of maximum size.

Suppose there is a polynomial algorithm $A(G = (V, E), F, R, v_s, v_t))$ that solves MPRI and returns $P = (v_s, \ldots, v_t)$, where each $v_i \in P$ contains the set of features $F_i \in F$ corresponding to the vertex in the graph. Then Algorithm 1 solves the MSI problem.

Algorithm 1 Solver for the MSI problem.

Input: C, Q, k
Output: $(S_a, \ldots, S_x)$
1: ▷ Given C, Q, and k, construct the graph and vertices for the MPRI problem.
2: Let $(G = (V, E), v_s, v_t) \leftarrow$ construct(C, Q, k)
3: Let $P = (v_s, \ldots, v_t) \leftarrow A((G = V, E), Q, Q, v_s, v_t)$
4: Let $P' = P \setminus \{v_s, v_t\}$
5: **return** P'

By the proof given by Xavier [80], MSI is NP-hard. We have shown that solving MPRI enables us to solve MSI. This means that MPRI is NP-hard. □

Chapter 4

Solution

This chapter presents a solution to the Maximum Path Requirement Intersection (MPRI) problem. The chapter first provides a general overview of the problem in Section 4.1, highlights the main idea behind the heuristic in Section 4.2, and then describes the workings of the heuristic in detail in Section 4.3. After that, Section 4.4 gives the proof of completeness within bounds for the heuristic algorithm, and Section 4.5 introduces the global BFS algorithm and proves its completeness. Section 4.6 explains how the algorithms are implemented. Finally, Section 4.7 discusses to what extent the given solution can be deployed in the current Internet.

4.1 General Overview

When a user wants to find a network path that satisfies their requirements, they can provide their path request via a browser extension or application[1]. A path request contains the source node, the destination node, the list of strict requirements, and the list of best effort requirements. Besides the path request, our system takes a model of the network where each node contains the list of features that is supported by that node[2]. Using that information, our system can get to work on finding a path. Once a path is found, this path is returned to the user, and the user can then use that path to route their Internet traffic. The main idea behind efficiently finding paths that satisfy as many user requirements as possible is introduced in the next section.

4.2 Main Idea: Avoid Bottlenecks

Our system attempts to find a good path for the user, that is, one that satisfies as many best-effort requirements as possible. To compare multiple paths and return the best one, paths are evaluated by their score, which is defined as the total number of best effort requirements that are supported by all nodes in the path. If we denote the set of best effort requirements that the user has provided in their path request as B and the features supported by node v_i as F_i, then the score S of a path $P = [v_1, v_2, \ldots, v_n]$ is calculated as $S = |B \cap F_1 \cap F_2 \cap \cdots \cap F_n|$, where $|X|$ denotes the cardinality of set X. Thus, this score depends on the overlap between the user's best effort requirements and the features supported by all nodes in the graph.

Now, for an example, assume that we have the graph given by the nodes and edges in Figure 4.1, where the numbers in each node signify the features that are supported by that node. Further, let $B = \{1, 2, 3\}$ be the set of best effort requirements given by the user. The selected path is marked with the bold lines. This path is naively chosen because it is the shortest path. In this situation, the bottleneck node in the middle decreases the score from a potential of 3 to 1, since the intersection of B and the features supported by the nodes in the path results in the set $\{1\}$, whereas the score could have been 3 if the longer path via the two nodes above the bottleneck was chosen instead. This reduction in score makes node B a *bottleneck*.

[1] Building the interface is outside the scope of this thesis, but a simple Graphical User Interface (GUI) where the user is guided through the different options would suffice. The GUI can also come pre-loaded with presets that users can select, like "Keep network traffic within the EU" or "Use the requirements from Mutually Agreed Norms for Routing Security (MANRS)".

[2] That model of the network can be provided by the Network Information Plane (NIP), or it comes from another source. This thesis assumes this information is readily available.

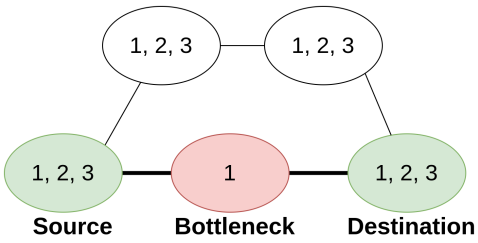


Figure 4.1: *Example graph where the numbers in the nodes indicate the features that are supported by that node. The shortest path uses the bottleneck node which decreases the total score of the path from a potential of 3 to 1. If the path can be adjusted to take a detour around this node via other nodes that do support all three features, the score for the whole path increases by two.*

The main idea of the heuristic algorithm is to find nodes that create a bottleneck, and calculate detours around these bottlenecks. The heuristic starts this process by first calculating the shortest path that satisfies the strict requirements, and then the heuristic improves the path by calculating detours around bottlenecks in that shortest path, after which the bottlenecks are replaced by the best available detour if that detour improves the overall score.

The strength of the heuristic algorithm lies in its configurability. The algorithm is configurable in two ways. First, the depth of the search for detours can be limited with the depthLimit, which controls how many hops the detours are allowed to take away from the original path. Second, the number of neighbours that are considered for a detour on each level can be limited using the neighbourLimit. Using these limits, the heuristic creates its own smaller search space around the shortest path. The size of this search space does not scale with the total size of the graph. This decoupling allows the heuristic to find solutions much faster than an approach that covers the whole graph. The idea is that we search until the limits are reached, and return the best solution that we found, similar to [52].

The downside of limiting the search space by using the depthLimit and neighbourLimit is that part of the graph is excluded from the search by the heuristic. This exclusion of part of the graph prevents the heuristic from always finding the globally best solution, since these globally best solutions might utilize one or more of the nodes that fall outside the local search area that the heuristic searches in.

4.3 Detailed Description

This section details how a path is found by describing how the heuristic algorithm works step by step. The pseudocode of the heuristic algorithm is detailed in Algorithm 2 in Appendix A[3].

The heuristic starts by checking whether start and end nodes comply with the strict requirements. If not, no valid path can ever be found, and the search is stopped. If the start and end nodes comply, the shortest path from the user's start node to the user's end node is calculated. This path calculation is done via bidirectional Breadth-First Search (BFS) by the bidirectionalBFSWithFilter method (Algorithm 5). If this path exists, the algorithm verifies that the user selected at least one best effort requirement. If the user selected no best effort requirements at all, there is no optimization to be done and the shortest path is returned as the path.

Assuming there are best effort requirements, the algorithm generates all possible start and end nodes for detours. The generation of detour start and end nodes is done for each detourDistance. The detourDistance is the number of hops between the start and the end of the path segment that is being replaced by a detour. Thus, if one node is denoted as bottleneck, the detour skips a path segment of two hops, resulting in a detourDistance of two. Similarly, a detour that skips two nodes skips a path segment of three hops resulting in a detourDistance of three, etcetera[4]. This relationship between bottleneck and detourDistance is visualized in Figure 4.2.

Using this detourDistance, the nodes where the detours begin and end, which we refer to as detourStart and detourEnd, are selected by enumerating all pairs of (detourStart, detourEnd)

[3] All other pseudocode is placed in Appendix A as well.

[4] We cannot skip the start and end nodes of the original path, thus the detourDistance can become $|path| - 1$ at most.

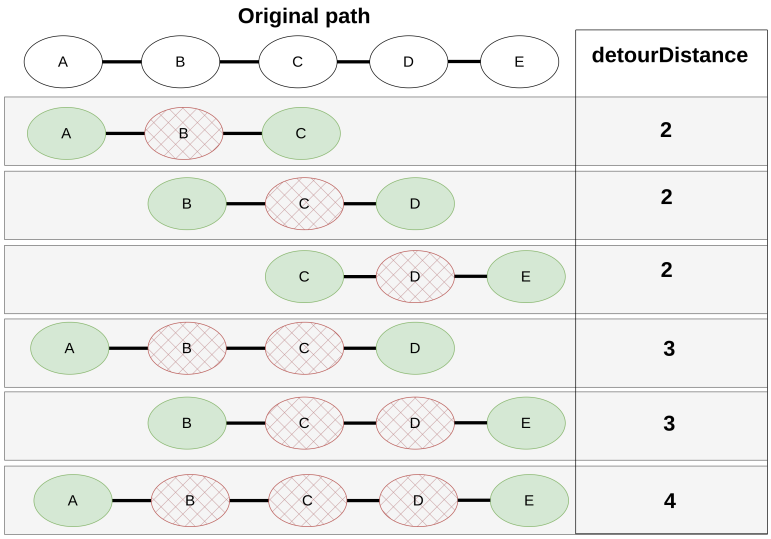


Figure 4.2: *Visualization of all possible detourDistance values for a 5-node path and their corresponding detours. White nodes indicate the original path, solid green nodes indicate the start and end of a detour, and nodes with the diagonal red squares pattern as background indicate the bottleneck nodes that are skipped by the detour. A path of 5 nodes allows for detourDistance values of 2, 3, and 4, resulting in a total of 6 (detourStart, detourEnd) pairs.*

nodes with that detourDistance. These detourStart and detourEnd nodes are selected only from the nodes of the *original* shortest path. If a node is selected as detourStart or detourEnd, but that node is not in the original shortest path, that pair of detourStart and detourEnd nodes is skipped for detour calculation. From the detourStart and detourEnd nodes the detours are calculated by the findDetours() subroutine (Algorithm 3). The selection of the start and end locations of the detours is visualized in Figure 4.2.

The findDetours() subroutine calculates the detours for a given start and end location. Calculating detours with detourDistance set to 1 is done by finding out which nodes are a neighbour of both detourStart and detourEnd and adjusting the path to go via this shared neighbour to bypass the bottleneck. If longer detours are desired, neighbours of detourStart and detourEnd can also be queried for shared neighbours, which allows us to move from the detourStart and detourEnd to one of their neighbours, and then connect the path via a shared neighbour of the neighbours. Even longer detours can be constructed by consulting the neighbours of the neighbours of the neighbours, etcetera. This path extension can go on for as many levels as desired. See Figure 4.3 for an illustration of the shared neighbours and the multiple levels of detours.

Next to detours of odd length, detours of even length are detected. If we denote a neighbour of the detourStart as startNeighbour and a neighbour of the detourEnd as an endNeighbour, then these even-length detours are detected by checking for each (startNeighbour, endNeighbour) pair whether they are connected via an edge. If the pair is connected, the subpath via these nodes is added as a detour. This search can be extended similarly to the odd-length detours by querying whether a neighbour of the startNeighbour is connected to a a neighbour of the endNeighbour, etcetera. This situation is visualized in Figure 4.4.

The process of calculating detours via neighbours starts by calculating the neighbours of the detourStart and detourEnd nodes and removing all nodes from those sets that are also present in the path. This node removal is done to prevent a situation where the bottleneck nodes are selected as a detour or a situation where one node is selected for a detour multiple times. Next, the set of neighbours is limited to the maximum size according to the neighbourLimit parameter by the limitNeighbours() subroutine (Algorithm 4).

Limiting neighbours consists of reducing the pool of selected neighbours to a smaller pool to reduce the search space[5]. This limiting is done in a smart way to ensure that the path can be improved as much as possible despite the limitations. First, each node in the set of

[5]If there are less neighbours than the limit, all neighbours are conserved of course.

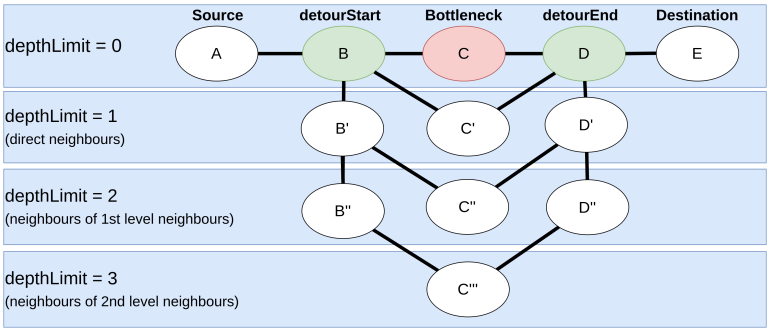


Figure 4.3: *An overview of how multiple levels of detours are generated. To avoid the bottleneck node C, nodes B and D can be queried for shared neighbours, resulting in a detour of B-C'-D. Further, neighbours of B and D (B' and D') can also be queried for shared neighbours, resulting in a longer detour of B-B'-C"-D'-D. This detour detection can continue to deeper levels, where neighbours of the neighbours are queried for shared neighbours, until the depthLimit is reached. Note that the heuristic also detects detours of even length. These even-length detours are illustrated in Figure 4.4.*

.

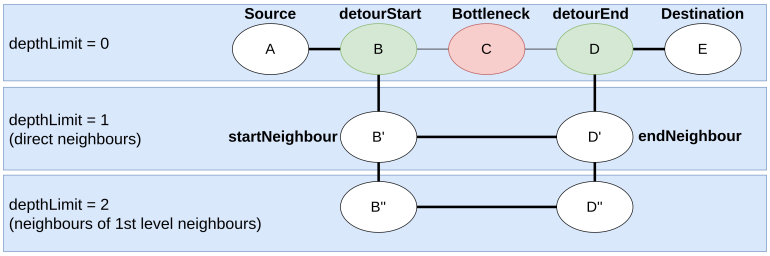


Figure 4.4: *An overview of how multiple levels of detours are generated in even-length detours, where neighbours directly connect to form a detour. Note that nodes B' and D' are not shared neighbours of B and D, but they can still be used in the four-hop detour B-B'-D'-D. Detours of higher length can be constructed by increasing the depth.*

neighbours is scored based on the size of the intersection of its features with the set of best effort requirements supported by all nodes in the path, excluding the bottleneck nodes. This number indicates which neighbour is most suitable to be used as a detour. Then the nodes are ordered on descending score, and finally the specified number of nodes are selected in order of descending score. This ordering ensures that the selection contains the nodes with the highest score, maximizing the chances of keeping the neighbours needed to form useful detours while filtering out the less relevant neighbours. This process is visualized in Figure 4.5.

After the set of neighbours has been limited, the findDetours() subroutine continues by determining which of the selected nodes are neighbours of both detourStart and detourEnd. This overlapping set of neighbours becomes the set of shared neighbours. All nodes in the set of shared neighbours are a neighbour of both the detour's start node and the detour's end node, which means that any of these nodes can be used as a detour around the bottleneck. Thus, each of these nodes is added as a detour, resulting in 3-node detours (detourStart, shared neighbour, detourEnd). After that, the neighbours are analysed to see whether there are pairs of (startNeighbour, endNeighbour) nodes that are connected with an edge. The pairs that are connected are added as a detour as well, resulting in 4-node detours (detourStart, startNeighbour, endNeighbour, detourEnd).

Through recursive calls detours longer than 4 nodes can be calculated. To ensure that the detours contain no loops, the start and end sections of the detour are stored in two variables called prefix and postfix. These prefix and postfix variables store the start and end of the detour until the point where new neighbours are considered to be added to the detour. As the detour is constructed, the prefix and postfix are updated and passed along to each recursive call to ensure that the complete detour can be constructed at each subsequent level. The amount of levels is limited using the depthLimit variable.

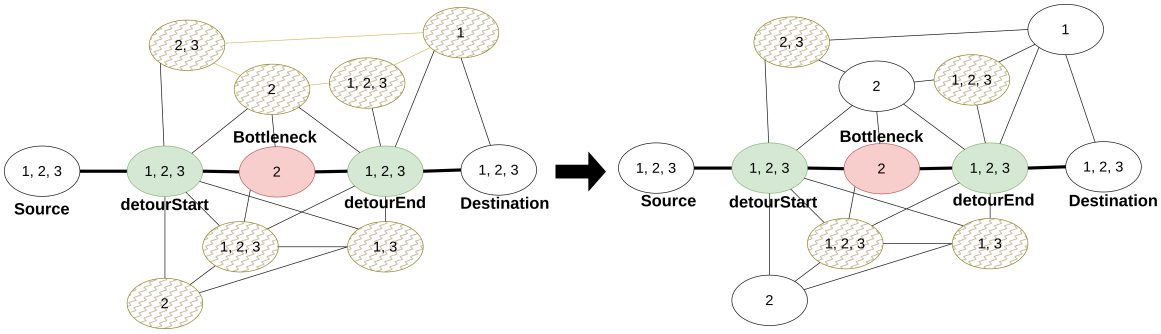


Figure 4.5: *Limiting of neighbours based on number of implemented best effort requirements. The bold edges denote the current path, the green nodes denote the detourStart and detourEnd nodes, the red node denotes the bottleneck, and the nodes with the yellow wavy lines as background denote the neighbours considered for a detour. The neighbours are limited based on the number of best effort requirements they satisfy. In this case the neighbours are limited to the top 3 for both the detourStart and the detourEnd, causing some of the previously selected neighbours to become deselected.*

After the detours are calculated, the heuristic continues by assessing the extent to which each detour improves the path. Path improvement of a detour is measured by the total number of best effort requirements the path satisfies when the bottleneck is replaced by a detour. To calculate the path improvement, first the number of best effort requirements that the current path satisfies is calculated, which we refer to as the overall score. Then the path is temporarily stripped of the bottleneck, and the set of best effort requirements satisfied by the path without the bottleneck is calculated, which we refer to as the bottleneck free score. If these numbers are equal, the selected bottleneck turns out to not be a bottleneck node after all, since the overall score does not decrease if this "bottleneck" node is included in the path. In that case, the comparison ends, and the next bottleneck is selected for optimization. However, if the overall score *is* lower than the bottleneck free score, the bottleneck does decrease the overall score, and we have something to improve. Once this is the case, the comparison can begin.

To do the comparison, the algorithm loops over all detours for the current detourStart and detourEnd, and calculates the score of the path in the case that the bottleneck is replaced by the detour, which we refer to as the detour score. While calculating the detour score for each detour, the algorithm keeps track of the detour that results in the largest detour score. Once all detours are assessed, the best detour is installed in the path. This process is visualized in Figure 4.6.

For each detourStart and detourEnd combination, detours are calculated using the complete search space given by the depthLimit and neighbourLimit. This way the heuristic can be sure that the detour that is installed in the end is the best possible detour that could be found within the given limits. Another approach would be to search detours first with depthLimit 1, see if any of the detours improve the path, and, if this is not the case, stop optimizing for this combination of detourStart and detourEnd. The heuristic does not follow this iterative approach, since longer detours (that are found on a higher depth) can provide improvements that are not available in shorter detours. Thus, terminating the optimization if shorter detours do not improve the path could yield a worse result compared to when the full search space was used to find detours. This is why the heuristic does not use the iterative approach, but instead calculates detours using the complete search space before starting the comparison.

The process of selecting a detourStart and detourEnd, selecting the best neighbours, finding all possible detours, and updating the path with the best detour continues for all skipAmounts. In the end, the resulting path is improved as much as possible within the given limitations, and the path is returned to the user. Pseudocode for each algorithm and routine is given in Appendix A.

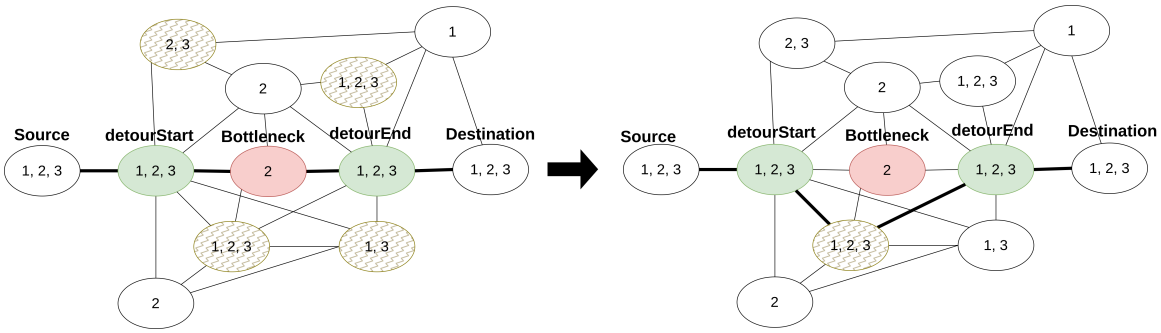


Figure 4.6: *The bold line denotes the current path, and the nodes with the marked with the yellow wavy lines are the neighbours that are analysed for potential detours. The bottom two marked nodes represent a potential detour, and the left node is chosen because it results in the path with the highest score. Note that in the right figure the bold edges that denote the path go via the marked detour node instead of the red marked bottleneck node.*

4.4 Proof of Completeness within Bounds

Proving the completeness of the heuristic algorithm consists of showing that it finds the best path out of the all paths within the given limits. The main idea of the proof is the following.

Proof summary: The search space of the heuristic algorithm is limited by the values for depthLimit and neighbourLimit. The heuristic algorithm finds the shortest path, and then replaces the bottlenecks of that path with detours if that results in a bigger set of satisfied best effort requirements for the whole path, where the detours are calculated using all nodes within the specified limits. While the path is being updated, the heuristic keeps track of the current best path that the heuristic found within its limits at any time. Any alternative path located within the given depthLimit and neighbourLimit that is better than the current path will eventually be found in the form of a detour, and because the heuristic replaces parts of the originally found shortest path if a better alternative can be found, the heuristic will end up with the best possible path out of all the paths within the given limits.

4.4.1 Proof

The full proof is structured by induction. The proof talks about paths being the 'best path out of all paths within the given limits' or paths being 'equally as good as the best path out of all paths within the given limits'. Formally, the best path out of all paths within the given limits in this context is the path that satisfies the biggest set of the user's best effort requirements out of all the paths that can be found in the search space of the heuristic limited by the given depthLimit and neighbourLimit, and a path that is equally as good as the best path out of all paths within the given limits is a path that satisfies the same number of best effort requirements as the best path out of all paths within the given limits.

To prove: *Given a graph G(V, E), a path request, and arbitrary values for depthLimit and neighbourLimit such that depthLimit* $\geq$ *0 and neighbourLimit* $\geq$ *0, the heuristic algorithm finds the path from the source to the destination located within the search space limited by depthLimit and neighbourLimit that satisfies the largest possible set of the user's best effort requirements of all paths located within the search space limited by the given values for depthLimit and neighbourLimit.* The source, the destination, and the user's set of best effort requirements are specified in the path request.

Base case: In the base case, the limits are set such that the search space is as small as possible. A minimal search space implies that depthLimit = 0 and neighbourLimit = 0. In this base case, no neighbours are examined, which causes the graph to be restricted to the graph that consists of just the nodes and edges in the path found by the `bidirectionalBFSWithFilter()` method. In this graph, the path that satisfies the most best effort requirements is the path found by the `bidirectionalBFSWithFilter()` method, and this path is also the path that the heuristic algorithm returns. In the case that no path can be found due to the graph not being

connected, the algorithm returns no path, which is in that case the correct answer[6]. Thus, the base case is true.

Inductive hypothesis: Let the limits be set to arbitrary integers such that depthLimit $\geq$ 0 and neighbourLimit $\geq$ 0. Assume that the heuristic returns the path that satisfies the most best effort requirements out of all paths within the given limits.

Inductive step (only depthLimit increases): Let depthLimit' = depthLimit + 1 and let neighbourLimit stay the same. We show that the heuristic returns the path that satisfies the most best effort requirements in the graph that is formed by the limits as specified. In this case, the graph is extended by an extra level of depth, which allows for the calculation of detours that visit the neighbours of the previously deepest level of neighbours. These new detours can be used to further update the path if the detours improve the path.

Then there are two cases: Either the best path out of all paths within the given limits resided in the graph that was formed using the previous value of depthLimit, or the best path out of all paths within the given limits goes via the nodes that were added by increasing depthLimit to depthLimit'. In the first case, according to the *inductive hypothesis*, the heuristic had already found the best path out of all paths within the given limits before extending the limits, and since the path is only updated if it can be improved, this best path out of all paths within the given limits (or an equally good path) will be returned. In the second case, detours that include the new nodes will be calculated and considered, ensuring that the detour needed to form the new current best path that is found by the heuristic (or an equally good path) will be found. Thus, in the second case the current path will be updated with the detour that makes the current path the best path out of all paths within the given limits, and the heuristic will return that path, which is the best path within the given values for depthLimit and neighbourLimit.

Inductive step (only neighbourLimit increases): Let depthLimit stay the same and let neighbourLimit' = neighbourLimit + 1. We show that the heuristic returns the path that satisfies the most best effort requirements in the graph that is formed by the limits as specified. In this case, more neighbours can be explored for each node, which allows for the calculation of detours that visit these extra neighbours. These new detours can be used to further update the path if the detours improve the path.

Then there are again two cases. In the first case, the new neighbours are not needed to find the best path out of all paths within the given limits, in which case, according to the *inductive hypothesis*, the heuristic already found the best path out of all paths within the given limits (or another equally good path), and it will return this path. In the second case, the new neighbours *are* needed to find the best path out of all paths within the given limits, in which case the newly calculated detours will ensure that the path is updated to get a score equal to the best possible path. Thus, in the second case the heuristic will find the best path out of all paths within the given limits, or another path that satisfies an equal amount of the user's best effort requirements, and the heuristic will return that path.

Inductive step (both depthLimit and neighbourLimit increase): Let depthLimit' = depthLimit + 1 and let neighbourLimit' = neighbourLimit + 1. We show that the heuristic returns the path that satisfies the most best effort requirements in the graph that is formed by the limits as specified.

In this case, as in the previous two inductive steps, new nodes are available for the path, and either they are not needed for the best path out of all paths within the given limits, or they are needed. In the first case, according to the *inductive hypothesis*, the heuristic already found the best path out of all paths within the given limits (or an equally good one) and will return it. In the second case the heuristic will find the detours needed to construct the best path out of all paths within the given limits (or an equally good path), update the current path with these new detours, and return the updated path.

In conclusion, **by the induction rule**, the heuristic finds the best possible path out of all paths within the given limits (or an equally good path) for any depthLimit $\geq$ 0 and any neighbourLimit $\geq$ 0. Thus, the heuristic finds the path from the source to the destination that satisfies the biggest set of the user's best effort requirements out of all paths within the given limits. □

[6]The next steps of the proof assume that there is a path from source to destination. Accordingly, in the next steps this edge case is omitted.

4.4.2 Reasoning about Optimality

The heuristic algorithm finds the best paths out of all paths *within* its limits, as proven before. However, once these limits cause the search space of the heuristic to be smaller than the full graph, the guarantee of finding the globally best path is removed, as there are nodes in the graph that the heuristic does not consider, while these could be needed to form the globally best path. Thus, no proofs of global optimality can be given.

Despite the global non-optimality of the paths produced by the heuristic, an effort was made to improve the results. This improvement is done by ordering the neighbours in descending order of the size of the intersection of their features and the set of features supported by all nodes in the path except the bottleneck nodes. Then, when the neighbours are limited according to the neighbourLimit, the neighbours that are selected are the ones with the biggest potential for an improvement of the overall score. While this selection does not guarantee that the best detours are found, it does help.

The second limit that is imposed on the heuristic is the depthLimit. This limit reduces the search space, but what does it do to the optimality of the heuristic? The graph of our target domain (the AS-level internet) follows a power-law degree distribution, which causes the average path length to remain relatively short at 3.9 hops [54]. This short average path length reduces the impact on path optimality of the depthLimit, since the probability of the optimal path (or one that is equally as good) being available in the search space is higher if the length of that path is lower. Thus, the low average path length in the AS-level internet causes the impact of the depthLimit to be limited. This statement is supported by the evaluation, that empirically shows that the heuristic performs close to optimal despite the limitations imposed by the depthLimit and neighbourLimit (see Section 5.7).

4.5 GlobalBFS: Algorithm for Globally Optimal Paths

The heuristic described in the previous section can provide paths that are globally optimal if there are no limitations. However, the heuristic is specifically designed to work with a restricted set of neighbours. Once the set of neighbours is expanded to include the whole graph by setting the depthLimit equal to the total number of nodes and the neighbourLimit equal to the total number of nodes, the heuristic is significantly slower than an algorithm that is designed from the ground up to find the globally optimal path, since the main speedup in the heuristic comes from the fact that its search space can be limited very effectively.

Thus, for this task, globalBFS is introduced. GlobalBFS is a BFS search with smart stopping. Smart stopping means that globalBFS keeps track of the current best path and its score and stops exploring a branch if the score of that branch is less than or equal to the score of the current best path, where the score is the number of best effort requirements that the path fulfills. This smart stopping does not prevent globalBFS from finding the globally optimal path since the skipped branch can never result in a path that is better than the current best path.

GlobalBFS has as a FIFO queue of internal states, where each state keeps track of the next node to be explored, the current path at that point in the search, and the set of best effort requirements that this intermediate path fulfills. This extra bookkeeping is necessary to ensure that all possible paths are explored, specifically paths that might result in more best effort requirements while taking a longer route. If a longer path from the source to some intermediate node would be overwritten by a shorter path from the source to that same intermediate node, we lose a path that is longer in length but potentially results in a larger score. Allowing all paths to be explored is essential, as the goal is not to find the shortest path from source to destination, but to find the path that satisfies the most best effort requirements, regardless of the length of the path.

4.5.1 Functionality Description

The globalBFS algorithm works as follows. First the algorithm checks whether the start and end nodes satisfy the strict requirements. If one of them or both do not satisfy the strict requirement, an empty path is returned. Then, a FIFO queue that keeps track of the state of each search branch is initialized and filled with the first entry that contains the start node, an empty path, and the full set of best effort requirements given by the user. Then the BFS loop

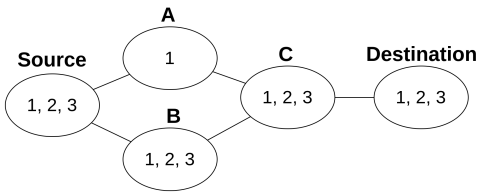


Figure 4.7: *Illustration to motivate that globalBFS does not filter nodes that are to be explored based on whether they have been visited already. This non-marking of nodes as being visited is to ensure that after globalBFS finds the path Source-A-C-Destination, it also explores the path Source-B-C-Destination, which results in a higher score. If previously visited nodes were not re-explored, node C would not be considered for the second path, and the second path would never be found. GlobalBFS does skip nodes that are already in the current intermediate path to avoid loops.*

begins by dequeueing the first item from the queue and updating the variables for the current path and current set of best effort requirements. If the size of the current set of best effort requirements is less than or equal to the current globally best score, the search is terminated (this is the smart stopping). If the current node is the final node, we have arrived. When we have arrived, we check whether our current score is better than the globally best score, and if that is the case, we update the globally best score as well as the path that resulted in that score. If the current node is not the final node, we add all neighbours to the queue, together with our current path and our current set of best effort requirements. Neighbours are only added to the queue if they satisfy the user's strict requirements and if they are not already part the path. Those are the only two requirements. Note that we do not keep track of visited nodes, since that prevents paths from being explored, while these paths might result in a better score. This situation is visualized in Figure 4.7. When the queue is empty, the search is completed, and the best path together with the score of that path are returned. The globalBFS algorithm is outlined in Algorithm 7 in Appendix A.

Note that the runtime of globalBFS is extremely long, due to the need to check every possible path. The average runtime of globalBFS is 2183 times slower than the average runtime of the heuristic. Because of these long runtimes, globalBFS cannot be used in practice. Its main purpose is to find the globally optimal path so the performance of the heuristic can be measured relative to that optimal path.

4.5.2 Proof of Correctness and Completeness

We claim that globalBFS is *correct* and *complete*. GlobalBFS being *correct* means that globalBFS always returns a path for which all nodes satisfy the strict requirements. GlobalBFS being *complete* means that globalBFS always returns the path that satisfies the maximum amount of best effort requirements.

Claim: globalBFS is correct.

Proof: There are two situations where a node that does not satisfy strict requirements can be added to the path. The first is during the initial addition of the start node to the queue. The second is when some node other than the start node is being considered as the next node to be explored. GlobalBFS takes care of the first situation by initially verifying that the start and end node satisfy the strict requirements. If one of these nodes (or both) do not satisfy the strict requirements, no path is returned, as no path from start to end node for which all nodes satisfy the strict requirements can be found. The second situation is dealt with by only exploring nodes that satisfy the strict requirements. Thus, both situations are covered, and we can conclude that globalBFS is correct. □

Claim: globalBFS is complete.

Proof: The proof examines the three states the algorithm can be in.

The first state is the state where the current branch contains a set of best effort requirements that is larger in size than the current best, and the current node is the destination node. In this case, a path that satisfies a larger set of best effort requirements than the current best has been found, and the global best scores are updated with the values of the current branch. In the case that this path ends up being the globally best path, it will be returned by globalBFS.

If another path ends up satisfying more best effort requirements, that other path will overwrite the global best scores with its values, and that path is returned.

The second state is the state where the current branch contains a set of best effort requirements that is larger in size than the current best, and the current node is *not* the destination node. In this case, the algorithm keeps exploring, ensuring that any path that flows from the current incomplete path that ends up being the globally best path will be found.

The third state is the state where the current branch contains a set of best effort requirements that is smaller than or equal to the global best. In this case, this branch can never result in a path that is better than the current globally best path. Thus, the search in this branch can be stopped without breaking the claim that globalBFS always returns the globally best path.

Finally, in the case that no path can be found due to too strict strict requirements, or due to a disconnected graph, the globally best path is no path because we should never return a path that does not fulfill the strict requirements. Thus, when no path can be found because of too strict strict requirements, the best path is no path. In that case, globalBFS indeed returns no path, which makes it complete in this final case as well.

Thus, in each state, globalBFS ensures that the path that is returned is the globally best path, and we can conclude that globalBFS is complete. □

4.6 Implementation

Our algorithms are implemented in Python. The graph model is created using the NetworkX Python package. Analysis and visualization of the results is done using Numpy and MatPlotLib. The source code of our algorithms and experiments can be found on this github repository: `https://github.com/rocketman1243/routing-based-on-user-requirements`. The implementation closely follows the pseudocode as displayed in Appendix A.

The code does not run in parallel but sequentially. This choice was made because the MPRI problem requires a path for which *all* nodes in the path support the largest possible set of requirements. If one were to split up the path into segments and optimize the segments individually, the segments may barely overlap in terms of requirements that all segments support. The path score, which is the total number of best effort requirements that all nodes in the path support, is a metric that can be highly influenced by any of the nodes, which motivates centralized and sequential optimization. Parallelization could be a promising avenue for future work, but not for this thesis.

4.7 Deployment

This section explores in what ways the heuristic can be integrated in the current Internet. The system proposed in this thesis can only be used in practice if it can work within the protocol that is used for AS-level routing, which is currently BGP. The only thing that is needed from BGP is allowing users to specify which route they want their packet to take. If users can specify this route for their packets, the route can be calculated using the system proposed in this thesis, and BGP ensures that this route is taken. The technology that enables this is source routing. Thus, the question becomes: Does BGP support source routing?

In theory: yes. In practice: no. BGP supports source routing via segment routing [32, 59, 65, 66]. Thus, in theory, BGP supports our system. However, in practice, this feature is disabled by default since source routing allows attackers to bypass firewalls [2]. Thus, in practice, many ASes do not support source routing, which makes it not feasible to use BGP source routing.

If source routing via BGP is not possible, what *can* we do? This section describes three approaches that in varying degrees implement the routing based on user requirements proposed in this thesis. The first is based on xBGP, the second on SCION, and the third on regular BGP. None of them fully implement the needed functionality, but they do get us in the right direction.

4.7.1 xBGP

The first approach uses an extension to BGP called xBGP [79]. xBGP is a new implementation model for BGP that enables ISPs to innovate by easily deploying BGP extensions in their

multivendor network. The multivendor nature of the Internet causes a difference between the set of features that is supported by the different ASes, which do not all use hardware from the same vendor. This vendor asymmetry reduces the pool of features supported by all ASes to the intersection of the features supported by all vendors. xBGP solves the vendor asymmetry problem by introducing a virtual machine that allows the execution of a shared library of BGP functions, which can then be ran by each ISP and allows for execution of BGP extensions.

One of the use cases of xBGP is the support for a more fine-grained control over path selection, which is implemented by having all edge routers submit *all* their routes to a shared virtual routing and forwarding table, instead of only propagating the best path. This shared table is then filtered for all paths to a certain AS, and that AS gets access to those paths. Now an ISP can choose from all paths instead of only the best paths, and choose which path it installs from the complete pool of paths.

This system can be combined with our system by analyzing the paths in the virtual routing and forwarding table, and scoring and filtering them based on the user's strict and best effort requirements. This score can then be used to decide which route to install. This approach does not fully allow the user to send packets along the exact route according to their strict and best effort requirements, but it does allow ISPs to install routes that are more in line with the demands of their customers.

4.7.2 SCION

The second approach uses Scalability, Control, and Isolation On Next-Generation Networks (SCION), which is a clean-slate Internet architecture [84]. It has security by design and it aims to provide availability even with active malicious actors present on the network. Routing is done in an hierarchical way. All ASes are grouped into Isolation Domains (ISDs), and in each ISD a trusted group of ASes is appointed as core nodes. These core nodes then calculate multiple paths from themselves to each non-core node in the ISD and install them in that non-core node[7]. Paths from a non-core node to a core node are referred to as up-segments, and paths from a core node to a non-core node are referred to as down-segments, and these segments can consist of one or multiple ASes.

SCION allows the end user to choose which of the up-segments and down-segments is used for the route. Thus, our system could be extended to perform pathfinding in the SCION way, and inform the user about the up-segments and down-segments that satisfy the user's strict requirements and the combination of up-segment and down-segment that in total satisfies the most of their best effort requirements. The user can then select the corresponding up-segment and down-segment and let their network traffic take the selected route.

This approach does not fully allow the user to exactly determine what route the packet should take, as the user can only choose from the pre-computed up-segments and down-segments. Full control over the path would allow users to exactly and precisely determine the route on the level of individual ASes. However, in the case of SCION, the smallest path unit that the user has control over is an up-segment or down-segment, which can consist of multiple ASes, and there are no guarantees that all possible paths from a node to a core node and back are available for the user to pick from. Thus, in that way, the choice of the user is restricted. However, there is still some level of control over the route, which is better than nothing.

4.7.3 BGP

This final approach uses purely BGP. Unfortunately, it does not fully get us where we want to be, but it can help to move the Internet in the right direction. The idea works as follows.

The path selection process of BGP consists of a comparison based on a number of attributes of the path, and the attribute that allows us to implement our system is the local preference. This attribute can be used to prefer a certain path over another. We can use this local preference parameter in conjunction with BGP communities to perform community-based traffic engineering. This technique consists of setting specific communities in our BGP route announcements that trigger our neighbours to update their local preference for us to a higher or lower value.

[7] Routing between ISDs is done by going to a core node, and from the core node of one ISD to the core node of another ISD, and then to the non-core node of the destination ISD.

Specifically, ASes can encode the features they support in the form of communities that they add to their BGP announcements. Communities can be seen as a tag that can be attached to a BGP path announcement. A BGP announcement can have multiple communities attached to it. Recipients of the announcement can match on the tags in a way that is similar to regular expressions, and take certain actions based on what communities they find attached to the announcement. One of the actions that recipients can take is increasing or decreasing their local preference for a certain path based on what communities they encounter in the announcement.

In our situation, we have a sender AS that announces routes to its neighbours, and attaches to these routes a community that indicates how many features it supports. Neighbours of the sender AS receive the announcement and can then see the feature-indicating communities in the announcement. Based on the number of features that the sender AS supports, the receiver AS can then increase their local preference for the path just announced by the sender AS. Once the new path is installed and the higher local preference is updated, packets targeted at the AS that sent the new route will take the path it just announced, since that path has the highest local preference.

This approach implements part of the system of this thesis, but it does not get there all the way. The downside of this approach is that we lose the granularity of sending traffic over a path that satisfies the exact set of requirements a user requested. We can only make sure that the likelihood of sending packets over a path that implements more features is increased, causing these ASes to receive more traffic. BGP only allows one path to be selected per destination, thus with the only modifier we have, the local preference, we can only choose one path for each destination. Thus, within the constraints that the BGP protocol enforces combined with the situation where ASes always retain the last word on where they do or do not send their traffic, the above-mentioned approach is the best we can do. This system works best when a group of ASes agrees on what communities map to which features, and what actions are taken when a BGP announcement with a certain set of communities is received. A potential group of ASes for which this system could work is the group of participants of the MANRS project.

Mutually Agreed Norms for Routing Security (MANRS) is an organization that encourages ASes to support four main features: filtering BGP announcements, implementing anti-spoofing measures, maintaining globally accessible and up-to-date contact information, and publishing your routing information. At the time of writing, there are 1185 ASes registered as a member of the MANRS organization. If the MANRS organization defines communities for each of the features they advocate for, and define what the local preference should be set to for each combination of supported features, the member ASes can implement that. Then the MANRS members can communicate their supported features to each other via BGP updates and send their packets to preferred peers based on these features using the adjusted value for local preference. Motivation for why ASes would want to participate in this system is described in Section 6.3.

To ensure a standardized format for the feature communities, we can use BGP extended communities [74]. These communities are registered by the Internet Assigned Numbers Authority (IANA) [42]. The MANRS organization can define their format and request an officially registered extended community to represent the MANRS features. The rest of this section details parts of the design of this extended community to ensure it reaches its goal.

There are two types of extended communities: transitive and non-transitive communities. Transitive communities are communities that may be forwarded to neighbours that receive it as part of a BGP announcement, whereas non-transitive communities should not be forwarded. The feature communities should be 'conditionally transitive', meaning that a feature community should only be forwarded by an AS if that AS supports that feature. Only when an AS announces itself to its neighbours, it is allowed to add communities that were not there before. If an AS only forwards an announcement and adds itself to the AS path, it can only remove the communities from the announcement that correspond to features it does not support. This mechanism is useful, because in this way ASes will receive AS paths with a certain set of communities and only forward the announcement with the (sub)set of features it supports as well. This ensures that all ASes on the AS path of the announcement support the set of features that correspond to the communities in the BGP announcement. The ASes receiving that announcement can then update their local preferences correspondingly, and thus route their traffic along the paths that support the most features.

To be on the safe side, we suggest to set the transitiveness of the extended community to non-transitive when registering the features community with IANA, such that ASes do not forward features they do not support. ASes themselves can then decide whether to add or remove the feature community based on their supported features and 'override' the non-transitive property of the community by forwarding it after updating. An obvious risk here is that ASes announce features they do not support. To combat this risk, ASes modifying their local preference based on the received communities should cross-compare whether the AS they received the community from actually supports all features it claims to support using the openly available MANRS data on what operator supports what feature [57], and only act on the information if it matches up. Each AS that receives the community should do this cross-comparison, and then the path is verified for each hop.

One might suggest that we skip the supported-features community and have each AS calculate for themselves for each AS path they receive which common set of features is supported using the MANRS data. However, unfortunately currently the majority of ASes are not a member of the MANRS organization, which would cause ASes to do large amounts of computation with the result that the set of commonly supported features is empty. Thus, for scalability the supported-features community makes more sense, as it only appears once an AS actually supports these features, and it does not appear when an AS supports no features.

The extended communities registered by IANA come in various types, and the type that suits our use case best is the Opaque Extended Community [74]. This community type has 8 bits to define a sub-type that identifies this community as our feature community, and 48 bits to define a value. This value can be used to represent what features are supported. To allow for flexible matching and execution, we propose a design where we both encode the total number of features supported, as well as the specific features that are supported. This double encoding allows matching either on specific features or on the number of supported features and allows for extensions to the feature-based routing system later on.

We propose the following design of the 48 value bits: the first 42 bits encode one bit per feature, where 0 means the feature is not supported and 1 means the feature is supported. The last 6 bits represent the total number of supported features in binary notation. This system thus supports up to 42 individual features, but it also works for less than 42 features. For example, let's take the 4 MANRS features (Filtering, Anti-Spoofing, Coordination, and Routing Information), and assign them an index in alphabetical order (Anti-Spoofing is 0, Coordination is 1, Filtering is 2, and Routing Information is 3). Applying this feature set to an AS that supports only the first two features, this AS would send a BGP announcement with the following extended BGP community attached to it:

```
[IANA_SUB_TYPE] 11000000 00000000 00000000 00000000 00000000 00000010
```

This community indicates that the AS supports Anti-Spoofing and Coordination by setting the first two bits to 1, and it indicates that the AS supports 2 features in total by having the last 6 bits set to 000010, which is 2 in binary.

Once an AS receives this community, this receiver AS can use the last 6 bits to update its local preferences according to the score, or update the local preference based on a specific subset of features if desired[8]. Next, the receiver AS can quickly calculate the features that are supported by both the sender AS and itself by applying a bit mask of the features that this AS supports onto the community. The bitmask can be similarly structured to the community, and with an AND comparison the new bitstring is quickly calculated. The only thing left to do for the receiver AS is to calculate the new total of supported features, append itself to the AS path and forward the announcement to its peers[9].

The design of this system facilitates gradual implementation, because a BGP announcement will only be decorated with a supported-features community if all ASes in the AS path support these features. Thus, an AS that does not participate in the feature-based routing system will not add the supported-features community, which will leave BGP working as usual. However,

[8] We do not propose actual values for the local preference, as each ISP has another policy. Thus, their default values and policies that govern the value for the local preference are different, and each ISP has to design their own values such that they are compatible with their policies.

[9] As a sidenote, for Cisco systems the forwarding of BGP communities is disabled by default. Thus, all ASes participating in this system should enable it with the command `neighbor [ip-address] send-community extended` or `neighbor [ip-address] send-community both`.

for the ASes that do participate, the special community that indicates the supported features allows them to modify their local preference for the destination and thus route based on the supported features.

Finally, note that some MANRS featues are not binary but gradual, where a feature can be supported for, say, 58 %. One can of course alter the design to use 7 bits per feature and thus encode percentage between 0 % and 100 %, but this design makes matching more complex, and it reduces the number of supported features from 42 to 6. Although 6 features would be enough for the 4 features that MANRS currently has, there is little room for future expansions down the road. Another option is to encode the percentages in steps of 10% or 12.5%, which reduces the number of bits needed per feature. If needed, this design can be redesigned such that each feature is represented by its own community, but that makes matching and updating the local preference value more complicated than the current design.

Chapter 5

Evaluation

This chapter analyzes the effectiveness of the heuristic in relation to the globally optimal path. The evaluation answers the following research questions.

1. What is the influence of the number of features and requirements on the performance of the heuristic?
2. What is the influence of the ratio between minimum and maximum number of features supported by each AS on the performance of the heuristic?
3. How well does the heuristic scale?
4. On what type of graph topology is the heuristic most effective?
5. How well does the heuristic perform in a realistic scenario?

These questions are answered in five experiments. The variable score range experiment (Section 5.3) evaluates the impact of different numbers of features and requirements on the performance of the heuristic. The ratios experiment (Section 5.4) examines how different ratios between minimum and maximum number of supported features per AS influence the performance of the heuristic. The heuristic scalability experiment (Section 5.5) evaluates the scalability of the heuristic in the length of the shortest path on a grid-based graph. The infrastructure experiment (Section 5.6) evaluates the effectiveness of the heuristic on four different graph topologies that are based on real world infrastructure networks. Finally, the Realistic Scenario experiment (Secion 5.7) evaluates the effectiveness of the heuristic on a graph of the AS-level Internet with realistic experiment parameters.

The evaluation chapter is structured as follows. Section 5.1 discusses the setup that was built around the algorithms to run the experiments. Section 5.2 describes and motivates the metrics that were used. Sections 5.3 – 5.7 describe the setup and the results of the experiments that were performed. Finally, Section 5.8 summarizes the results and extracts the key insights from the results.

5.1 Setup

The programming language version and package versions that were used to implement the experiments and algorithms are listed in Table 5.1, and the hardware that was used to run the experiments is displayed in Table 5.2. The source code can be found in this github repository: `https://github.com/rocketman1243/routing-based-on-user-requirements`.

To answer the aforementioned research questions, a comprehensive evaluation system has been created. The setup consists of a system to generate graph instances and path requests, an algorithm and graph selection system, and a data analysis system. The graph and path request generation systems create custom graphs and custom path requests according to given experiment parameters. The algorithm and graph selection system select the correct algorithm (heuristic or globalBFS) and the correct input graph for a given experiment. The data analysis system selects the correct results and processes them according to the selected experiment. A visual overview is given in Figure 5.1, and the following sections describe each part of the evaluation system in more detail.

5.1.1 Graph and Path Request Generation System

The graph instances are built from a list of files, each of which encodes the information for one node. Such a file contains the node id, the features the node supports, and the nodes it is connected to. These files act as the intermediary between the generation of graph instances and the construction of a graph model based on the files. This decoupling allowed the design of the graph generation system to develop independently of the experiment execution system.

Item	Value
Python version	3.10.12
NetworkX version	3.2.1
Numpy version	1.26.2
Matplotlib version	3.8.2

Table 5.1: *Code versions of the packages used in the implementation.*

Item	Value
Laptop model	Lenovo ThinkPad P16v Gen 1
CPU	Intel i7-13700H @ 4.8GHz
RAM	32 GB
Operating System	Pop!_OS 22.04 LTS x86_64

Table 5.2: *Specifications of the laptop that was used to run the experiment.*

The node files are generated by a script. This script contains many parameters that can be tweaked to change the parameters of the node files, which in turn changes the structure and characteristics of the graph model. Using the switches and different graph generator programs, the wide range of graph instances that has been created for the evaluation could be generated quickly and efficiently.

The path requests are generated using a similar script, with many switches and parameters to customize the contents of the path requests. Using these parameters, the path requests can reflect the specific scenarios and experiments that were tested during the evaluation. Each path request is encapsulated in its own file, which contains the start node, the end node, the strict requirements, and the best effort requirements. Using the node files and path request files as an intermediary allows easy modifications in terms of extra information that needs to be communicated. If one wants to add, for example, the latency of all the edges, or the geolocation of a node, or any other information, it can be easily added.

Once the node files and path request files are created, they are read by the main program. The main program constructs a NetworkX graph model based on the node files, and feeds this model, along with one of the path requests, to either the heuristic algorithm or the globalBFS algorithm.

5.1.2 Algorithm and Graph Selection System

The evaluation uses two algorithms: the heuristic algorithm (introduced in Section 4.1) and the globalBFS algorithm (introduced in Section 4.5). Each algorithm can satisfy a path request on multiple types of graphs, and the experiments used multiple different graphs as input. To deal with this diversity, an algorithm and graph selection system was developed to select the correct combination of algorithm and input graph for each experiment. This system is a collection of switches that allows one to indicate which algorithm and which graph is being used, and based on that the correct files are read, the correct environment variables are set, and the correct algorithm is run with the correct input. Tying the setup to the chosen switches allows someone to quickly set up and run multiple experiments while having to do very little setup for each experiment. While the chosen algorithm is satisfying the chosen set of path request on the chosen graph, the runtime and other statistics are gathered. Once the experiment is completed, the results are written to a file.

5.1.3 Data Analysis System

Data analysis starts by reading the data from the files created by the previous system. This data is then processed, and visualized. Similar to the previous systems, the data analysis system is also highly configurable. Using switches, the desired experiment and the corresponding results files can be selected. These switches also determine what graphs should be generated. Once the desired graphs are generated, they are saved to figures to be used as illustrations in the report.

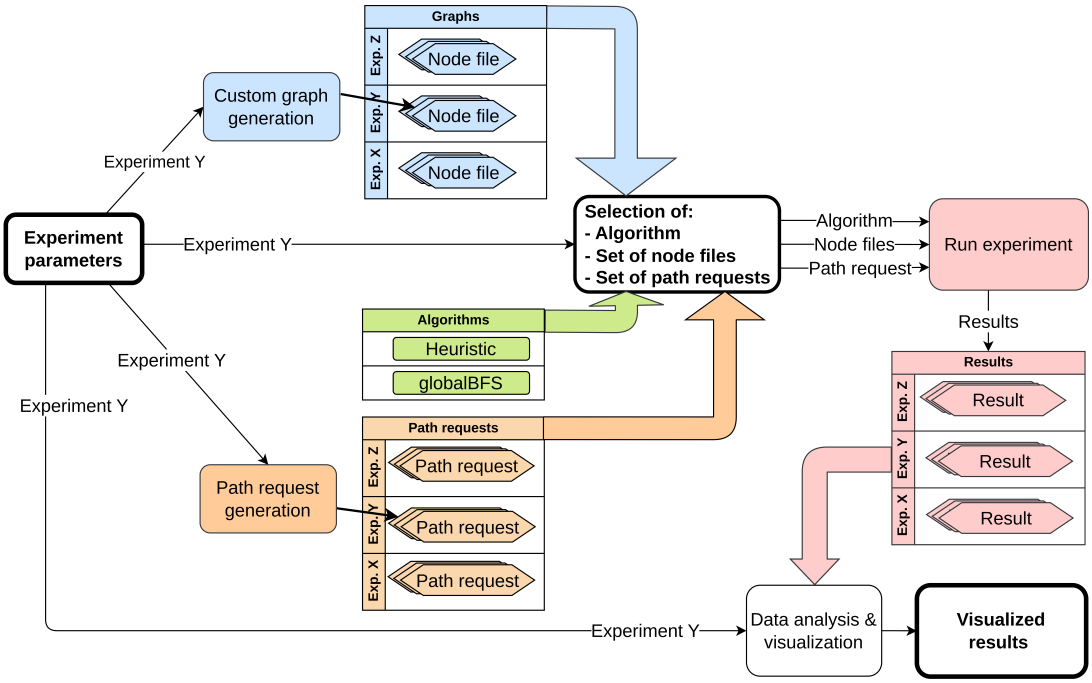


Figure 5.1: *Visual overview of the evaluation system. Graphs are represented by sets of node files that encode all information for one node, including its edges. One path request results in one path. The algorithms handle one path request at a time. Results are stored separately for each experiment, and the analysis and visualization is customized for each experiment. Each part can be executed independently of each other, allowing for efficient preparation and batch running of experiments.*

5.2 Metrics

The evaluation uses three metrics: average path improvement, average runtime, and score difference[1]. The following sections explain how these metrics are calculated and motivate why they were chosen as the metrics for this evaluation.

5.2.1 Average Score Improvement

Average score improvement is the amount of extra best effort requirements that the heuristic is able to satisfy by updating the path with detours compared to the original path. This metric is relevant because it shows the extent to which the heuristic can find detours to eliminate bottlenecks. If the limits are too strict, the search space for detours is too limited, causing a situation where the path can hardly be improved. If the results of multiple sets of limits are compared, and at some point the improvement plateaus, the limits are too wide and can be tightened down to reduce the heuristic runtime with only a minuscule impact on the score. Thus, the average score improvement gives clear insights into the effectiveness of the heuristic.

5.2.2 Average Runtime

The aim of this thesis is to create a solution to the problem that is usable, where usability is expressed as the extra delay that the heuristic adds to the process of web browsing or other routing applications. This goal of usability makes measuring the runtime relevant. The runtime is averaged over multiple path requests, since one path request can have positively or negatively skewed results. Using the average runtime we can analyze the relationship between different values for the limits and the corresponding average runtime, and determine what the best trade off is.

[1] Which metric is used in what experiment is explained when the experiments are introduced.

Min number of features	Max number of features	Requested requirements
4	5	1, 2, ..., 4, 5
20	25	1, 2, ..., 24, 25
80	100	1, 2, ..., 99, 100
400	500	1, 2, ..., 499, 500

Table 5.3: *Values for minimum and maximum number of features for the variable score range experiment. To illustrate the way the range is used: 20 – 25 features means that the minimum number of features for a node was 20, the maximum number of features for a node was 25, and the exact number was randomly chosen via a uniform distribution between the minimum and the maximum. Further, it means that the path requests were assigned the set of 1 through 25 as best effort requirements. The same applies for the other ranges.*

5.2.3 Score Difference

The performance of the heuristic on its own is hard to evaluate. When does it perform well? When does it lag behind? To get accurate answers to these questions, it makes sense to compare its performance to some standard. In these experiments, the globally optimal results from the globalBFS algorithm provide that standard.

When looking at the performance of the heuristic relative to globalBFS, comparing runtime makes no sense. The heuristic is designed to reduce the search space in order to significantly reduce the runtime, while globalBFS has to search the whole graph for the globally best path[2]. Thus, the heuristic will be much faster than globalBFS. Instead, the difference in score between the heuristic path and the globally optimal path is compared. This comparison gives an insight into how well the heuristic solves the NP-hard Maximum Path Requirement Intersection (MPRI) problem despite being limited by the depthLimit and neighbourLimit. With the setup and the metrics explained, we can now look at the experiments and their results.

5.3 Variable Score Range Experiment

To express how 'good' a path is, we measure the score of a path. The score of a path is defined as the number of requirements that this path fulfills. The number of requirements that a user can request in their path requests and the number of features that can be assigned to ASes determine what range this score value can take on. If an AS can support between 0 and 5 features, and users can request between 0 and 5 requirements, the score ranges between 0 and 5. However, if the features and requirements range between 0 and 100, the score ranges between 0 and 100.

This experiment aims to answer the question: What is the influence of different score ranges on the performance of the heuristic? To that end, four different feature ranges and four corresponding requirement sets are introduced, and the performance with these score ranges is compared to each other.

5.3.1 Experiment Setup

The feature ranges examined in this experiment are: between 4 and 5 features, between 20 and 25 features, between 80 and 100 features, and between 400 and 500 features. The requirements in each path request are set to the full set of requirements from 1 up to and including the maximum of the feature range. These ranges are summarized in Table 5.3. For each of these ranges, the number of features assigned to each node in the graph is randomly chosen from that range according to a uniform distribution. Reasoning behind the choice of distribution can be found in Section 6.4.

Each set of values aims to maintain the 4:5 ratio of minimum to maximum. The reason that the minimum number of features is not set to 0 is that using the full range of features reduces the clarity of the results. The main aim of the experiments is to examine the effectiveness of the heuristic relative to the globally optimal path. However, if nodes in the graph have very

[2]Note that globalBFS stops expanding a branch when it cannot result in a better score than the current best path with the current best score, so it can happen that globalBFS does not search all nodes.

little features, both the globally optimal solution and the heuristic will return paths with very low scores. The difference between these scores is then low, which makes it hard to see what influence certain parameters have. If the solution space is, instead, larger, there is more room for the score of the heuristic to be nearer or further away from the globally optimal solution, which allows us to draw more informed conclusions about the effectiveness of the heuristic. In a realistic scenario the minimum number of features and requirements would be 0, of course. Such a scenario is examined in the realistic scenario experiment (Section 5.7).

The ratio of 4:5 is chosen to ensure that the solution space is of desired size for each number of features, while having some consistency in the ratio between upper and lower limit across the different feature ranges. The effect of using different ratios is examined in the ratios experiment (Section 5.4), which shows that different ratios produce nearly the same result. Thus, the exact choice of value for the ratio does not matter much.

The heuristic supports the exclusion of autonomous systems based on strict requirements. However, eliminating autonomous systems from the graph due to not satisfying a set of strict requirements is not a hard task, and it reduces the search space, making the problem easier and the runtime lower, since there are less nodes to consider. Thus, to challenge the heuristic and keep the complexity of the problem intact, the path requests have no strict requirements such that no nodes are filtered from the graph.

In this experiment, the path requests are configured to request all best effort requirements between 1 and the upper limit of the feature range (e.g., for the feature range 80 – 100, the path requests request the best effort features 1, 2, 3, ..., 99, and 100.). Setting the best effort requirements in the path requests to all requirements ensures that only the intersection between neighbours causes reduction in the shared pool of features, instead of having the set of best effort requirements be a limiting factor as well.

The experiment is run on a graph based on the Internet on the level of ASes. This graph is built from the AS Relationships dataset from CAIDA[3]. This dataset is chosen because its scale is representative of the number of autonomous systems on the Internet[4], which allows us to determine whether the heuristic is fast enough to handle graphs of realistic size.

For this experiment, the main two questions that are answered for each combination of number of features and requirements are:

1. What is the optimal trade-off between runtime and quality of results?
2. How close are the results of the heuristic to the globally optimal results?

These questions are answered in two stages, one stage per question. The first stage is aimed at finding the best set of limits to give as parameters to the heuristic algorithm. The second stage is aimed at examining the difference between the score of the globally best path and the path found by the heuristic algorithm.

5.3.1.1 Limit Stage

The heuristic algorithm can be limited in two ways. The depth of the detour generation can be limited with the depthLimit, and the number of neighbours of the detourStart and detourEnd nodes that is examined for potential detours on each level can be limited with the neighbourLimit.

For each of these limits, there exists a trade-off between runtime and results. If the value of the limit is increased, the search space grows. On the one hand, the increased search space allows the heuristic algorithm to find more detours that could potentially result in a path with a higher score. On the other hand, the increased search space causes the heuristic to have a longer runtime, as the larger search space results in more and/or longer detours for which the score needs to be calculated.

The limit stage examines this trade-off. The experiment defines a number of limit values for the heuristic, runs the algorithm on a graph for each combination of limits, and then shows both the runtime and the improvement for each set of limits. Here, the improvement is number of extra best effort requirements that are satisfied on top of the set satisfied by the shortest path.

[3]The CAIDA AS Relationships Dataset 2023-06-01, from `https://publicdata.caida.org/datasets/as-relationships/serial-1/`, which contains 75,388 nodes and 501,550 edges, with an average degree of 13.31.

[4]As of 5 January 2024, CIDR reports 75,350 active ASes [40], while the AS Relationship dataset contains 75,388 nodes.

Parameter	Value
Number of strict requirements in each path request	0
Number of path requests	100
Runtime threshold	500 ms

Table 5.4: *Parameters for the limit stage. Note that the number of features and requirements are displayed in Table 5.3.*

The idea is that the set of best effort requirements satisfied by the shortest path is received 'for free', since the limits do not influence the search space for finding the shortest path. The part that the limits do influence is the calculation of detours and the subsequent *improvement* of the path, over the baseline set by the shortest path.

For this stage, a set of 100 path requests is randomly generated, meaning that each path request has a random start and end node. The algorithm will use this same set of path requests as input with each set of limit values to facilitate comparison between the limit values. All experiment parameters for the limit stage are summarized in Table 5.4.

Using the different limits, one can choose the trade-off between runtime and path improvement that one likes best. This choice can be made on the basis of average runtime, median runtime, maximum path improvement, or other metrics. For the next stage of this experiment, we pick one set of limits and compare the performance of the heuristic, configured with that set of limits, to the globally best solution. For this thesis, we will pick the example of selecting a set of limits based on average runtime. As a threshold for the runtime, again, any value can be chosen. We set the threshold such that the runtime should be below 500 ms[5]. However, other values can also be used. The result of the limit stage is the set of limits that provides the highest average score improvement while having an average runtime below the threshold of 500 ms.

5.3.1.2 Comparison Stage

The comparison stage takes the limits that provided the best results of the heuristic while keeping the average runtime under 500 ms in the limit stage, and compares the quality of the results of the heuristic with those limits to the globally best results. This comparison is performed by running both the heuristic and globalBFS on the same graph instance, and having both algorithms satisfy the same 100 randomly generated path requests.

Since we are dealing with an NP-hard problem, finding the globally best solution can take a tremendous amount of time. This phenomenon is visible in the runtime of globalBFS. GlobalBFS has to examine the full graph to guarantee that the path that it returns is indeed the globally best path, which results in a significant runtime[6]. This significant runtime makes gathering enough results to smooth out the influence of individual path requests that result in a significantly higher runtime than the rest too time intensive. To deal with that, we make two changes to the setup.

First, we run the heuristic and globalBFS on a smaller graph with 500 nodes. To create a graph with a similar structure, the graph is generated using the `random_internet_as_graph` graph generator[7] from NetworkX, which is based on the AS graph model by Elmokashfi, Kvalbein, and Dovrolis [26][8]. Since the small graph has the same structure and characteristics as the large graph, and the limits that the heuristic is configured with are determined on the large

[5]This time limit is chosen since it represents the amount of time that would not too significantly impact the experience of an end user for typical web traffic; waiting 500 ms extra is reasonable, while waiting 5 seconds extra can be too much. People tend to leave a website when the loading takes too long. The average limit is to be around 5 seconds[71, 33, 12], which make 500 ms a reasonable amount of time to add to the delay.

[6]Note that due to the smart stopping globalBFS might end up ignoring some nodes, but still its search space is nearly the whole graph.

[7]The source code for the generator can be found here: `https://networkx.org/documentation/stable/reference/generated/networkx.generators.internet_as_graphs.random_internet_as_graph.html#networkx.generators.internet_as_graphs.random_internet_as_graph`

[8]This model is from 2010, which makes it slightly outdated. However, the provided graph generator makes the model very user-friendly, which is why it was chosen. Further, a more recent study from 2021 [62] showed similar degree distributions in AS-level graphs to the work of Elmokashfi et al., which justifies the use of the generator in 2024.

Parameter	Value
Number of strict requirements in each path request	0
Total number of path requests	200
Number of path requests used in comparison	first 100 that finished within limit
Time limit per path request for globalBFS	5 minutes

Table 5.5: *Parameters of the comparison stage. Note that the number of features and requirements are displayed in Table 5.3.*

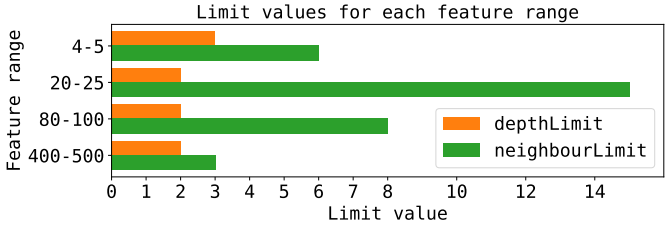


Figure 5.2: *Best limits for the feature range experiment. These limits are the best limits for each range of features applied to the nodes, given that the average runtime should stay below 500 ms. A feature range of 4 – 5 means that the minimum number of features for a node is 4, the maximum number of features for a node is 5, and the exact number is randomly chosen via a uniform distribution between the minimum and the maximum. Note that in general an increasing number of features causes the limits to decrease in order to keep the average runtime below the threshold of 500 ms (note that 4-5 has a lower neighbourLimit but a higher depthLimit).*

graph, we can expect that the results related to score differences between the heuristic and the globally best solution from the small graph are applicable the large graph as well.

Second, we stop the execution of globalBFS if it exceeds the time limit for one path request, which we set to 5 minutes. To deal with the fact that we might end up with less than 100 samples due to globalBFS not completing within the timeout, we run in total 200 path requests, and take the first 100 samples for which globalBFS completed within the time limit, counted by their filename index (so not sorted on lowest runtime). The parameters for this stage are summarized in Table 5.5.

5.3.2 Results

To compare the best effort requirements with the features supported by all nodes, many set intersections need to be calculated. These intersections, in the worst case, take $O(n^2)$ time. Consequently, an increase in the size of the sets on which the intersections need to be performed cause these intersections to take more time. This higher runtime requirement is clearly reflected in the results of the limit stage, shown in Figure 5.2. These limits are based on the runtime and path improvement results of the limit stage for each feature range, which are displayed in Appendix C.

The resulting limits show that a smaller score range allows the heuristic to explore a larger area, which is reflected in the higher limits of the smaller feature ranges. Conversely, as the score range grows, the limits decrease, since the amount of work that the heuristic can do within the 500 ms threshold is less, causing the space that the heuristic can explore to shrink. Thus, the results clearly show the relation between score range and search space.

However, when we zoom in on the limits of the 4–5 range and the 20–25 range, an interesting result emerges. The 4–5 range has a depthLimit of 3 and a neighbourLimit of 6, resulting in a total search space of $6 \cdot 6 \cdot 6 = 216$ nodes. The 20–25 feature range has a depthLimit of 2 and a neighbourLimit of 15, resulting in a search space of $15 \cdot 15 = 225$. Thus, surprisingly, the 20–25 feature range has a *larger* search space than the 4–5 feature range. An explanation for this phenomenon is that the larger 20–25 feature range gave more opportunities for the heuristic to find detours that improve the path than the smaller 4–5 feature range, which allowed the heuristic to produce better paths with a higher path score improvement with the 20–25 feature range than with the 4–5 feature range. This flexibility and this higher score could have led the heuristic to the higher search space, even though there were more features to process.

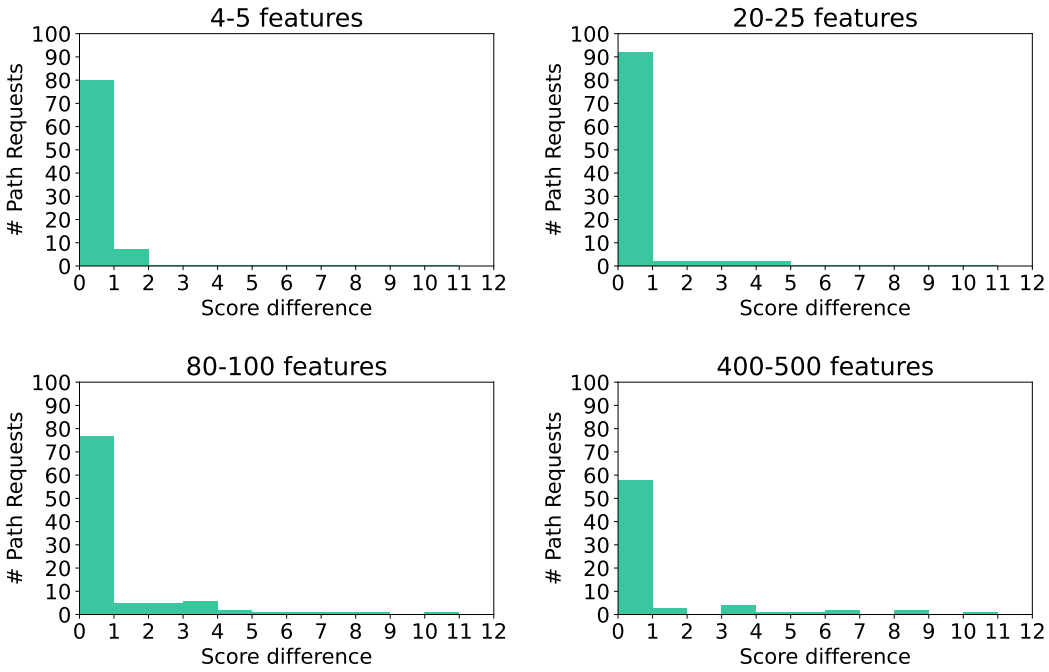


Figure 5.3: *Histogram of difference between number of best effort requirements satisfied by the globally best path and the number of best effort requirements satisfied by the heuristic path for each of the feature ranges for 100 randomized path requests. Note that the heuristic performs similar across feature ranges. The performance of the 400-500 range seems worse than the other ranges, but when looking at the relative performance (Figure 5.4) the performance for this range is similar to the other ranges. Thus, this seemingly worse performance could be because small relative differences are bigger in absolute numbers with the larger score range (note that the x-axis is equal for each score range).*

Next, in what amount does the reduced search space influence the effectiveness of the heuristic? While the larger feature ranges do negatively impact the heuristic performance somewhat in the score difference histograms in Figure 5.3, the difference is not significant, as the relative performance in Figure 5.4 shows. When examining the results more closely, the best relative performance is surprisingly achieved by the $20 - 25$ feature range, instead of by the $4 - 5$ feature range, while the theory of lower score range causing higher search space causing a more effective heuristic would suggest the opposite. An explanation could be that the $4 - 5$ features range does allow for a large search space, but the low number of features restricts the search space in terms of features more than the $20 - 25$ features range does. The larger "wiggle room" provided by the $20 - 25$ features range could be the cause of the higher relative performance and better results in the histogram.

5.4 Ratios Experiment

In the variable score range experiment, a ratio of 4:5 between minimum number of features and maximum number of features supported by each node was maintained across the different feature ranges. This number was chosen to have sufficient score range to have meaningful differences between the heuristic and globalBFS. However, would different ratios have influenced the results significantly? This experiment answers that question by comparing the performance of the heuristic across four different ratios between the minimum and maximum number of features that an AS supports.

5.4.1 Experiment Setup

The experiment is structured similar to the limit stage of the variable score range experiment. The goal is to find the best set of limits for which the average path improvement is maximal

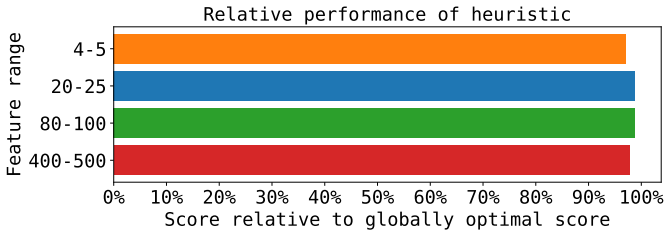


Figure 5.4: *Average relative score of heuristic compared to the globally best score for each feature range. 4 – 5 features means that the minimum number of features for a node was 4 and the maximum number of features for a node was 5. Note how all feature ranges have scores that are above 97% of the globally optimal score, with little differences among the different score ranges.*

Ratio	Min number of features	Max number of features
2:3	66	99
3:4	75	100
4:5	80	100
5:6	85	102

Table 5.6: *Ratios and their corresponding minimum and maximum number of features for the ratios experiment, given that the maximum number of features is approximately 100.*

while staying below the example runtime limit of 500 ms for each feature ratio. The nodes are configured to use approximately the same number of features, only with a different ratio between the minimum and the maximum. The maximum number of features is always approximately 100, and the minimum number of features is determined according to the ratios in Table 5.6. The number of requested best effort requirements is set to the maximum number of features for each ratio. The other parameters for the experiment are outlined in Table 5.7. Like in the the limit stage of the variable score range experiment, the experiment uses the full size graph of the AS-level internet based on the CAIDA dataset. Further, similar to the variable score range experiment, the path requests have no strict requirements.

5.4.2 Results

The full runtime graphs are available in Appendix D, but the summary of the results is that the values for the best limits are very similar for each ratio, having the same value for depthLimit and a neighbourLimit that differs by at most 2 among the ratios. This shows that the exact values that are chosen for the ratio between the minimum number of supported features and the maximum number of supported features does not significantly affect the performance of the heuristic. The limit values are visualized in Figure 5.5.

Note that a natural follow-up question would be: What is the effect of the minimum number of supported features on the results of the heuristic? This could be answered by looking at feature ranges like 10–100, 20–100, 30–100, up to 100–100. However, while these results are theoretically interesting, they have no real-world meaning. The only reason that the experiments use a minimum number of supported features for each AS in the graph is to ensure that the score range is high enough to show clear differences between the heuristic and globalBFS. In reality there will be no minimum on the number of supported features, as each AS has different priorities regarding the implementation of features, and each AS has their own goals. For that reason the above-mentioned ratios of 10–100, 20–100, etcetera, are not examined.

5.5 Heuristic Scalability Experiment

The internet is growing every day [41, 73], and the system in this thesis is aimed at routing internet paths. A relevant question then becomes: How well does the heuristic scale? That question is answered by this experiment, in which the runtime of the heuristic is measured relative to the length of the shortest path between start and end node.

Parameter	Value
Number of total features	Approximately 100
Number of strict requirements in each path request	0
Number of path requests	100
Runtime threshold	500 ms

Table 5.7: *Parameters for the limit stage of the ratios experiment. The number of total features is approximately 100 because the ratios did not always line up to the round number 100, since the number of features needs to be integer.*

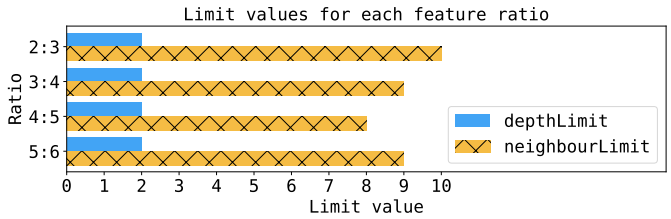


Figure 5.5: *Limit values for the ratio experiment. Note that the limits are very similar across different feature ratios, indicating that the exact choice of ratio does not significantly affect the results.*

To do that, a square grid graph of 500 by 500 nodes was set up with a feature range of 80–100 features per node, and path requests that request all 100 best effort requirements for each path request. These path requests were created with an increasing distance between start and end node, starting at 10 hops, and increasing with 10 hops until a maximum length of 200 hops. We generated 10 path requests with a random start and end location for each distance, resulting in 200 path requests in total. The task for the heuristic is simply to solve all path requests on the 500x500 grid graph. For each path request, we kept track of the runtime. The result is an overview of the runtimes for each distance between start and end location.

We chose a grid graph for this experiment because the regular layout of this graph ensures that choosing certain start and end coordinates results in a path request for which the shortest path is at least the intended number of hops. Further, the regular layout of the grid graph creates a setup where the amount of work that the heuristic needs to do is deterministic, regardless of what start and end location are chosen. This determinism creates a stable environment to test the scalability of the runtime. We tried to repeat the scalability experiment on the full size AS graph, but the power law degree distribution of that graph causes shortest paths between two nodes to remain short, even if the size of the graph grows. This characteristic made it difficult to find start and end positions for which the shortest path has a length equal to 10, and finding paths of higher length proved even more difficult. This absence of a large number of long shortest paths makes the AS graph not suitable for the scalability experiment.

5.5.1 Results

The results are visualized in Figure 5.6. The results show that the runtime of the heuristic grows in a polynomial fashion with the distance between start and end, both when looking at the median and maximum values of the runtimes for each distance. This polynomial scaling means that the heuristic is mostly useful for small instances of graphs where the length of the shortest path grows with the graph size, as the growing runtime makes the heuristic less useful as the graph size becomes larger.

A note on these results; keep in mind that the scalability of the heuristic is less of an issue if it is used to calculate routes on internet graphs, as the degree distribution of a graph of the internet follows a power law, which keeps shortest paths short even if the graph grows in size. This fact makes this result interesting, but less relevant.

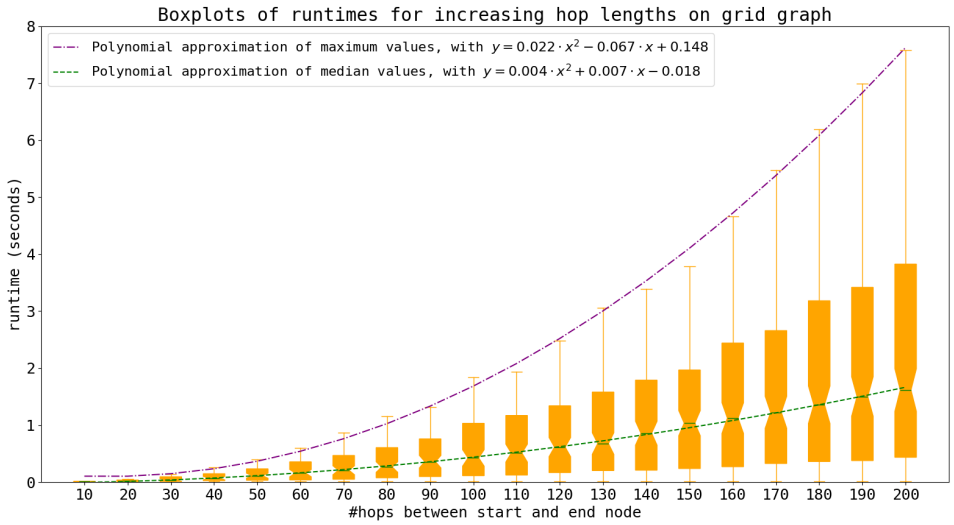


Figure 5.6: *Boxplots of the runtimes of path requests with increasing distance between start and end nodes on the grid graph. The result for each distance is a boxplot that visualizes the runtime of 10 path requests that requested a path with that distance between start and end. For distance, the Manhattan distance was used. Note that the runtime increases in a polynomial fashion, both for the median and maximum runtime values.*

5.6 Infrastructure Experiment

The infrastructure experiment answers the question: In what type of graph topology is the heuristic most effective? Answering this question is done by evaluating the performance of the heuristic on four different graph types. The first graph type is the Internet on the level of ASes based on the CAIDA AS Relationships dataset that was used in previous experiments as well. The other three graphs are based on real-world infrastructure networks.

When evaluating the performance of the heuristic, its performance on the internet graph is of primary interest, as the autonomous systems network is the domain that this thesis is aimed at. However, the heuristic might be useful in other domains as well. To evaluate the usefulness of the heuristic in other domains, three other graph types are selected for the evaluation of the heuristic alongside the graph of the AS-level Internet. When choosing additional graphs, many options are available. We chose to focus on real-world infrastructures because they represent a domain where custom routes are relevant and useful, much more so than other domains like social networks, citation networks, or collaboration networks[9]. Thus, this experiment puts the heuristic to the test on multiple graphs that are inspired by real-world infrastructures.

Like in the variable score range experiment, the main two questions that this experiment aims to answer are:

1. What is the optimal trade-off between runtime and quality of results?
2. How close are the results of the heuristic to the globally optimal results?

However, instead of answering these questions for one graph topology and four different amounts of features and requirements, this experiment uses one amount of features and requirements, and answers the question for four different graph topologies. The experiment is set up similar to the variable score range experiment, with the limit stage determining the limit values, and the comparison stage comparing the performance of the heuristic with the results of globalBFS. Further, similar to the variable score range experiment, each infrastructure graph type has a large instance and a small instance, where the large instance is used in the limit stage, and the small instance is used in the comparison stage.

[9]Note that networks of website connectivity (websites modeled as nodes and a link from website to another modeled as an edge) are not relevant as well, as calculating a route between two websites that only traverses websites that adhere to user's requirements is not a relevant use case, since people don't use the Internet like that.

Parameter	Value
Number of total features	100
Minimum number of supported features per node	80
Maximum number of supported features per node	100
Best effort requirements in each path request	1, 2, ..., 100
Number of strict requirements in each path request	0
Number of path requests	100
Runtime threshold	500 ms

Table 5.8: *Parameters for the limit stage of the infrastructure experiment.*

Parameter	Value
Number of total features	100
Minimum number of supported features per AS	80
Maximum number of supported features per AS	100
Best effort requirements in each path request	1, 2, ..., 100
Number of strict requirements in each path request	0
Total number of path requests	200
Number of path requests used in comparison	first 100 that finished within timeout
Time per path request for globalBFS	5 minutes

Table 5.9: *Parameters of the comparison stage of the infrastructure experiment.*

Further, similar to the variable score range experiment, the path requests have no strict requirements. For the range of features that is distributed over the ASes, the amount is chosen uniformly between 80 – 100 features, and the path requests request all 100 requirements. The parameters for the limit stage are summarized in Table 5.8, and the comparison stage parameters are summarized in Table 5.9.

5.6.1 Graph setup

This experiment uses graphs inspired by the following infrastructures: an autonomous systems network, a network of city streets, a flight connectivity network, and a village street network. These infrastructures represent different characteristics, which challenge the heuristic in multiple ways. The graphs are created using graph generators. These graph generators aim to create a graph that is structurally similar to a real-world type of infrastructure. The inputs for the generators are based on the statistics of the dataset of the graph representation of the real-world infrastructure that the generator replicates. The datasets and the statistics of the graph types are displayed in Table 5.10. Note that the large AS graph is not made with a generator but instead it is built using the dataset, as the AS graph dataset results in a connected graph, which makes it feasible to use in our experiments. The other graph datasets do not result in a connected graph, making it difficult to calculate paths as some path requests are denied due to having a start and end in different connected components[10].

5.6.1.1 Autonomous Systems Network

The network of autonomous systems is based on the topology of the Internet on the level of autonomous systems. Each autonomous system is represented by a node, and a peering relationships between two ASes is represented by an edge. The large network is built from the CAIDA AS relationships dataset from 2023-06-01[11]. The small network has 500 nodes and is created using the `random_internet_as_graph` graph generator[12] from NetworkX, which is based on the AS graph model by Elmokashfi, Kvalbein, and Dovrolis [26].

[10] While searching for datasets, alternatives that did result in a connected graph could not be found, except for the AS graph dataset.

[11] `https://publicdata.caida.org/datasets/as-relationships/serial-1/`

[12] Source: `https://networkx.org/documentation/stable/reference/generated/networkx.generators.internet_as_graphs.random_internet_as_graph.html#networkx.generators.internet_as_graphs.random_internet_as_graph`

Real-world infrastructure	AS Network	City Streets	Flights	Village Streets
Number of nodes (both)	75,388	4,426[a]	6,827	2,776[b]
Number of edges (dataset)	501,550	9,626	37,595	3,731
Number of edges (graph)	501,550[c]	8,699	41,196	2,927
Average degree (dataset)	13.31	4.35	11.01	2.69
Average degree (graph)	13.31	3.94	11.98	2.14
Source of dataset	CAIDA[d]	Kaggle[e], [9]	Kaggle[f]	Harvard[g], [10]
Average degree difference	13.28	0.82	7.34	1.02

Table 5.10: *Statistics of the datasets that provided inputs for the generators. Average degree difference represents the average absolute difference in degree between the nodes in the dataset and the generated graph based on that dataset after these degrees are sorted in descending order.*

[a]The exact amount of nodes of the graph is 4,416.

[b]The exact amount of nodes in the graph is 2,732.

[c]The nodes and edges of the graph are the nodes and edges of the dataset. That is why the number of edges and the average degree is exactly the same.

[d]The CAIDA AS Relationships Dataset 2023-06-01, from `https://publicdata.caida.org/datasets/as-relationships/serial-1/`

[e]Source: Manhattan, from `https://www.kaggle.com/datasets/crailtap/street-network-of-new-york-in-graphml`

[f]Source: `https://openflights.org`, via `https://www.kaggle.com/datasets/open-flights/flight-route-database`

[g]Source: The Apple Valley edgelist from `https://dataverse.harvard.edu/file.xhtml?persistentId=doi:10.7910/DVN/CUWWYJ/HDWT5G&version=2.0`

5.6.1.2 City Streets

The city streets network is rougly approximated by using a 2D grid. A 2D grid is not the same as the network formed by the streets of a city, but for the purposes of this thesis it is close enough, especially since the characteristics of the generated 2D grid network are based on the streets of Manhattan, which is laid out in a way that comes close to a 2D grid[13]. The 2D grid graph consists of a 2D mesh of nodes where the nodes are connected in a grid-like fashion. The large and small graph of this type are created using the `grid_2d_graph` generator from NetworkX[14]. The large graph is a grid graph of 64 by 69 nodes, resulting in a grid graph of 4,422 nodes, which approximates the 4,426 nodes in the original dataset. The small graph is a grid graph of 22 by 23 nodes, resulting in a total of 506 nodes, approximating the target size of the small graph of 500 nodes.

5.6.1.3 Flights Network

The flights network is based on airline connectivity between airports. Airports are represented by nodes, and flights between airports are represented by edges. The network is created using the `powerlaw_cluster_graph` graph generator from NetworkX[15] with as inputs: 6,827 nodes, and 6 new edges per added node, informed by the statistics of the openflights.com airline connectivity dataset the network is based on[16]. The probability of forming a triangle after adding an edge is set to 0.001. The power law generator is applicable to model airline connectivity as the degree distribution of flight connectivity graphs tends to follow a power law distribution [35]. The small graph is created with 500 nodes, the same 6 new edges per added node and the same value of 0.001 as the probability of forming a triangle after adding an edge.

[13]We tried to find datasets of city streets networks to use as a graph, but we were unable to find a city streets network that was connected. The disconnectedness of the networks we did find made them unsuitable for pathfinding, as some paths are not possible due to the start and end node being located in a different connected component.

[14]Source: `https://networkx.org/documentation/stable/reference/generated/networkx.generators.lattice.grid_2d_graph.html`

[15]Source: `https://networkx.org/documentation/stable/reference/generated/networkx.generators.random_graphs.powerlaw_cluster_graph.html#networkx.generators.random_graphs.powerlaw_cluster_graph`

[16]Source: `https://openflights.org/`, via `https://www.kaggle.com/datasets/open-flights/flight-route-database`.

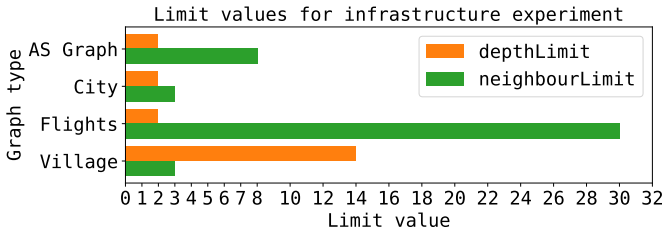


Figure 5.7: *For each graph type the limits that maximized improvement of supported best effort requirements while having an average runtime below 500 ms. Two noteworthy results; First, note that the flights neighbourLimit is significantly higher than the other neighbourLimits. This result could be due to the efficient power-law degree distribution and the relatively small graph size of the flights graph. Second, note that the depthLimit of the village streets network is significantly higher than the others. This result could be due to the corridor-like features of the graph which force the heuristic to find detours of higher length to go around the corridors.*

5.6.1.4 Village Streets Network

The village streets network is based on the network of streets in a small town, where streets are less interconnected. In this case, the degree is lower, as streets are modeled using multiple edges, instead of one edge. An intersection is one node, then a street consists of a path graph of 2 or more nodes, and the street ends in another intersection, which is modeled as a node. This structure decreases the average degree of the village streets network when compared to the city streets network.

The network is created using a custom generator made by us, which works as follows. Initially a 2D grid is created. Then, between each pair of nodes of the original grid network, the edge that connects these nodes is replaced by a path graph of a random length between 2 and 10 hops. The addition of the path graphs between the intersections creates a network with 'tunnels' that make the heuristic less effective, since there are less opportunities for detours. The large graph is constructed using an original 2D grid network of 15 by 15 nodes, resulting in 225 nodes in the grid. The path graph inserts are set to have a length between 2 and 10 hops, which increases the total number of nodes to 2,732. The small graph is created with an original 2D grid graph of 10 by 10 nodes, with path graph inserts ranging in length between 1 and 4 hops, which brings the total number of nodes to 539.

5.6.2 Results

First, the limit stage resulted in runtime and path improvement values displayed in Appendix E. The boiled down version of these results is the depthLimit and neighbourLimit for which the path improvement was largest while having an average runtime below the example runtime limit of 500 ms. These limit values are visualized for each graph type in Figure 5.7.

A surprising result is the high value for the neighbourLimit for the Flights graph type, which at 30 is significantly higher than the neighbourLimit values for the other graph types. An explanation for this result can be that the flights network has a relatively small number of nodes compared to the AS Network, while having a similar degree distribution. This smaller number of nodes reduces the length that paths can be, and, like we saw in the scalability experiment, shorter paths mean that the heuristic can do more work in the same amount of time. In this case it means that the heuristic can consider more neighbours within the time limit than it could in the AS graph network, which explains the difference between the neighbourLimit values.

Another surprising result is the depthLimit for the village streets network, as this value is significantly higher than the depthLimit values for the other graph types. This result can be explained by looking at the structure of the village network. The custom generator created path graphs between the street intersections of between 2 and 10 hops, creating "meta-streets" of at most 12 hops. A high depthLimit is needed to find a detour that goes around such a meta-street.

Second, the results of the comparison stage, which are the differences in score between the globally optimal path and the heuristic path, are aggregated and displayed in Figure 5.8. The results show that the heuristic is most effective on the AS graph network, and least effective on

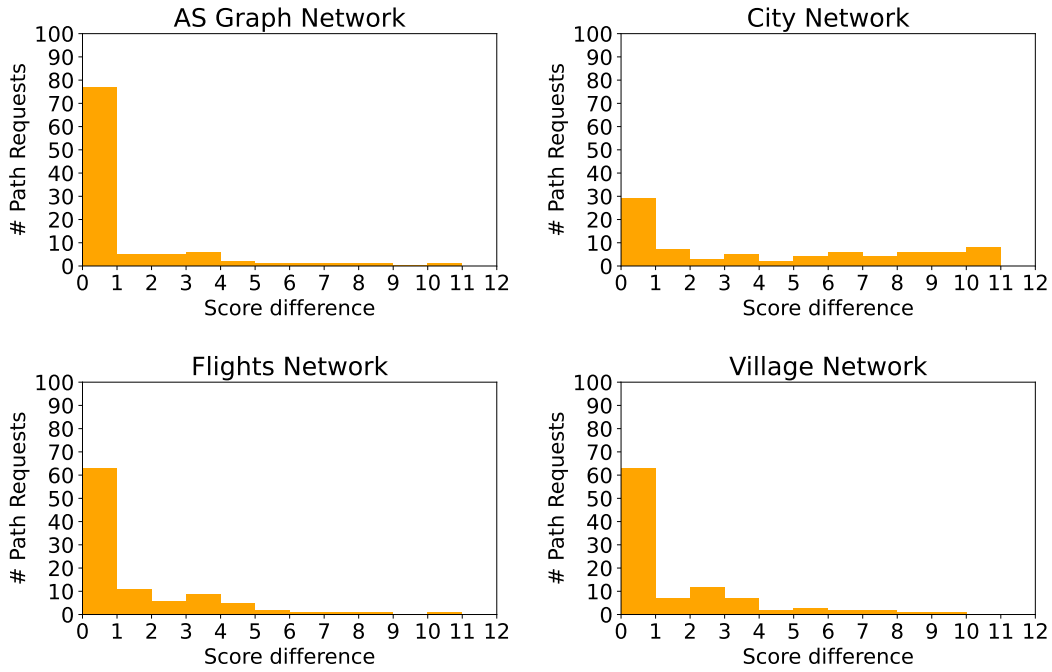


Figure 5.8: *Histogram of difference between the score of the globally best path and the score of the heuristic path for each graph type for 100 randomized path requests. Note that the performance of the heuristic is least effective on the city network graph, and most effective on the AS network graph.*

the city network. This difference can be explained by looking at the structure of the network. The AS-level Internet network follows a power-law degree distribution, which keeps the paths short [21, 54]. The grid graph, however, has a much more constant degree distribution, with nearly all nodes having a degree of 4. This lower degree impacts the heuristic performance in multiple ways. First, the low degree reduces the number of detours that can be found, which reduces the heuristic's ability to improve the score of the path over the score of the initial shortest path. Second, a grid layout causes paths between two randomly chosen nodes to become longer, since these nodes can be far apart and there are no high-degree nodes that allow quick traversal of large parts of the graph. This absence of high degree nodes increases the average length of the shortest path, which increases the amount of work that the heuristic has to do. Consequently, the runtime of the heuristic goes up, which decreases the values of the limits for which the heuristic still completes on average below 500 ms. Third, due to the low degree, and the low values for the limits, the search space of the heuristic is small, which increases the chances that the globally optimal path uses nodes that fall outside of the search space of the heuristic. These factors influence the results for the city network, causing its scores to be significantly worse than the scores of the other networks.

Finally, the relative performance of the heuristic compared to the globally optimal path is displayed in Figure 5.9. The results in percentages confirm the picture painted in the histograms: the heuristic is least effective on the city network, and most effective on the AS graph network.

5.7 Realistic Scenario Experiment

This final experiment answers the question: How well does the heuristic perform in a realistic scenario? Answering this question is done by assessing the performance of the heuristic on the full size AS graph in a more realistic scenario, where the main goal is to show what runtimes can be expected from the heuristic configured with different sets of limits, along with an overview of their score compared to the globally optimal score.

The realism of this scenario lies in two elements. First, the experiment takes the example list of 25 requirements in Appendix B as the source of requirements. This list contains the four requirements from MANRS, as well as requirements from a related RFC (RFC 2013). The

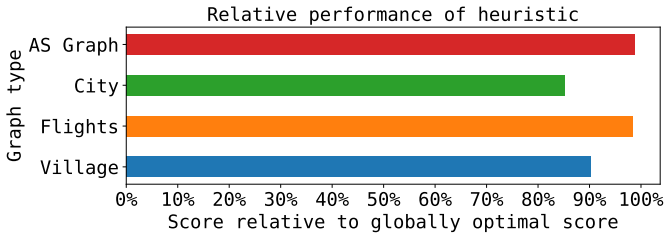


Figure 5.9: *Average score of the heuristic relative to the globally optimal score. Note that the city and village fall behind, while the AS graph and Flights graph perform well, with both having a score that is over 98% of the globally best score.*

Parameter	**Value**
Best effort requirements in each path request	1, 2, ..., 25
Number of strict requirements in each path request	0
Number of path requests in limit stage	100
Number of path requests in comparison stage	200
Minimum number of features supported by an AS	0
Maximum number of features supported by an AS	25
Heuristic average runtime upper limit	None
GlobalBFS timeout	5 minutes

Table 5.11: *Parameters for the Realistic Scenario experiment.*

system in this thesis is not yet adopted in the Internet, but if it were, the set of 25 requirements in Appendix B would be a reasonable set of requirements to request.

Second, in this experiment there is no lower bound on the number of features that an AS supports. The absence of a lower bound means that the number of supported features for an AS is randomly taken from between 0 and 25 according to a uniform distribution instead of having a minimum of 20 supported features like in previous experiments. The full range of features makes this experiment more realistic, because in reality ASes will not always support a certain minimum number of features. Like previous experiments, each path request requests all the 25 requirements as best effort requirements, while requesting no strict requirements.

The experiment is divided into the limit stage and the comparison stage. In the limit stage the heuristic satisfies 100 randomized path requests on the full size Internet graph based on the CAIDA AS Relationships dataset, resulting in boxplots of the runtime of the heuristic for a number of limits. In this experiment, the example runtime limit of 500 ms will not be applied. Instead, the comparison stage compares *all* configurations of the limit stage with the globally best solution. This comparison is set up similar to previous experiment; both the heuristic and globalBFS satisfy 200 path requests on a 500-node Internet graph generated by NetworkX's `random_internet_as_graph` graph generator[17], and globalBFS has a timeout of 5 minutes. The results of this comparison stage show the score of the heuristic path compared to the score of the globally optimal path for the first 100 path requests that globalBFS completed within the timeout. The parameters of this experiment are summarized in Table 5.11.

5.7.1 Results

The limit stage results, visualized in Figure 5.10, show that many heuristic configurations on average provide a solution to the NP-hard MPRI problem in *less* than 20 ms on a *full size* Internet graph, and most heuristic configurations have an average runtime below or around 40 ms. These low average runtimes show that the heuristic, in a realistic setup with a reasonable number of requirements and the full range of supported features, is able to satisfy path requests swiftly.

The results of the comparison stage, visualized in Figure 5.11, shows that all selected heuristic

[17]Source: `https://networkx.org/documentation/stable/reference/generated/networkx.generators.internet_as_graphs.random_internet_as_graph.html#networkx.generators.internet_as_graphs.random_internet_as_graph`

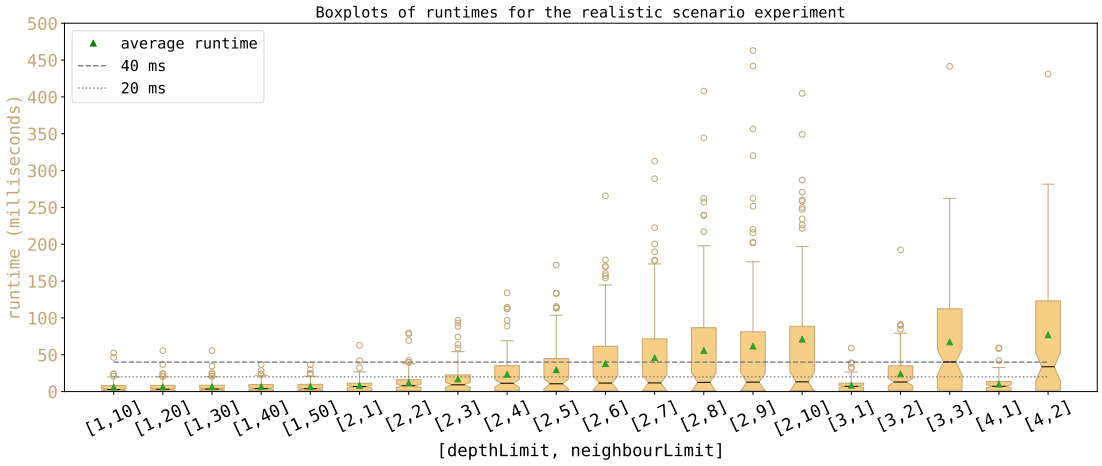


Figure 5.10: *Heuristic runtime boxplots for the Realistic Scenario experiment, as a result of the limit stage. The dashed gray line denotes a runtime of 40 ms, while the dotted gray line indicates a runtime of 20 ms. Note that most configurations have an average runtime below or around 40 ms.*

configurations produce paths that on average differ from the globally best solution by only 2 requirements, with the majority of the average differences being less than 1 requirement away from the globally optimal solution. In terms of percentages, all configurations achieve at least 75% of the globally optimal results, with the majority of the configurations achieving scores that are more than 90% of the globally optimal results. These high relative scores indicate that the heuristic performs *close to optimal* while often having average runtimes *below 40 ms*. Thus, in a realistic scenario, the heuristic satisfies path requests quickly while performing close to optimal in terms of score.

5.8 Summary of Results

The results of the five experiments provide us with the insights needed to answer the five main questions. These answers summarize the main results.

5.8.1 What is the influence of the number of features and requirements on the performance of the heuristic?

The results indicate that a lower number of features and requirements makes the heuristic faster, which allows it to search a larger search space within a certain amount of time. Thus, more features and requirements increases the heuristic search space, and less features and requirements reduces the search space.

However, restricting the search space too much could restrict the heuristic in finding detours around a bottleneck, causing the effectiveness to slightly degrade because of that. This score degradation suggests that a number of features and requirements that is too low might negatively impact the performance of the heuristic.

In practice, this result means that graphs where many nodes support little features, or path requests that request little best effort requirements, can result in slightly worse results produced by the heuristic. However, in general, if the number of features and requirements is not too small, then a smaller number of features and requirements tends to improve the effectiveness of the heuristic.

5.8.2 What is the influence of the ratio between minimum and maximum number of features on the performance of the heuristic?

The results show that the exact numbers chosen for the ratio barely affect the performance of the heuristic, since the limit values are nearly the same. Thus, the ratio of 4:5 can be used without the risk of significantly influencing the results due to choosing exactly that ratio.

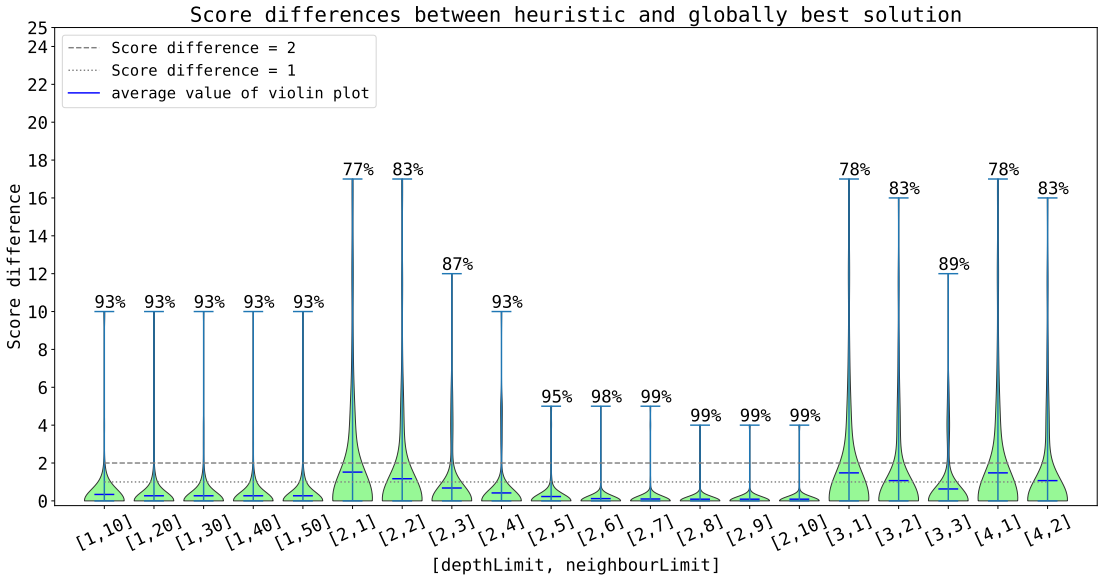


Figure 5.11: *Heuristic runtime violin plots for the Realistic Scenario experiment, as a result of the comparison stage, with the average heuristic score relative to the globally optimal score denoted in percentages above each violin plot. The dashed gray line denotes a score difference of 2, while the dotted gray line indicates a score difference of 1. Average values are denoted with the blue horizontal marker in the body of each violin plot. Note that most configurations have an average score difference below 1. Note as well that the widest part of each violin plot is placed on 0, which indicates that the majority of the score differences is 0. This shows that the heuristic performs very close to optimal.*

5.8.3 How well does the heuristic scale?

The results show that the heuristic scales polynomially in the length of the shortest path on a grid-based graph. This polynomial scaling could mean that, in the case of a grid-based graph, the heuristic is most effective in smaller graphs rather than larger graphs. In general, this result could mean that the heuristic is more effective on graphs where the average shortest path does not scale much with the size of the network.

5.8.4 On what type of graph topologies is the heuristic most effective?

The results show that the heuristic is most effective on the AS graph network. This result is good news, as the main focus of the thesis is on routing on the AS-level Internet. Since the degree distribution of the graph of autonomous systems follows a power law [60], this result suggests that the heuristic is effective on other graphs where the degree distribution follows a power law as well. The heuristic performed worse in the city streets network, which was modeled as a grid graph. This result suggests that the heuristic is less effective on grid-based graphs.

5.8.5 How well does the heuristic perform in a realistic scenario?

The results show that the heuristic satisfies the majority of the path requests within 40 ms for many configurations, while satisfying a number of best effort requirements that is close to optimal. These results indicate that, under realistic circumstances, the heuristic performs well.

Chapter 6

Discussion

This chapter discusses a number of topics that did not fit in the other chapters. Section 6.1 describes potential alternative pathfinding algorithms to calculate the shortest path. Section 6.2 discusses an idea on how to represent latency within an AS. Section 6.3 discusses how to incentivize ISPs to provide their information to the Network Information Plane (NIP), and Section 6.4 explains why a uniform distribution of features was used in the experiments.

6.1 Alternative Pathfinding Algorithms and Why They Were not Used

The heuristic algorithm starts by finding the shortest path from source to destination. This first step solves the generic shortest path problem, and there are many algorithms that solve it in different ways. We chose bidirectional Breadth-First Search (BFS). This section explains why bidirectional BFS was chosen over alternatives.

The domain for which the heuristic is designed is the Internet on the level of Autonomous Systems (ASs). That network changes multiple times per day [63], which makes search techniques that require extensive precomputation, like Hub-Labelling [1] which needs precomputed contraction hierarchies [34], less relevant, as the precomputation needs to be updated multiple times per day because the graph structure changes. Because of the frequent graph updates, we preferred uninformed approaches without precomputation over informed approaches that do require precomputation.

Another approach is heuristic-based search, where a heuristic guides the search in the general direction of where it should go. Heuristic based search works best in the domain of grid-based networks, where the nodes and edges are laid out in a regular pattern with every node that is not on the edge of the networks having the same degree. Euclidean distance, Manhattan distance or octile distance can then be used as the heuristic that guides the search. However, the graph of ASes is not a grid-based graph, which makes heuristic-based approaches like A* [37], HPA* [11], and Jump Point Search [36] less relevant.

For uninformed pathfinding algorithms, we considered BFS, DFS, and Dijkstra. The main distinction between Dijkstra and the other two algorithms is that Dijkstra uses edge weights to determine which node to expand next. In theory, our graph can have edge weights in the form of latency over an edge between two ASes. However, this data is hard to gather as Internet Service Providers (ISPs) are not keen of sharing latency information as it is quite business-sensitive information, and latency changes frequently due to congestion. This aversion to sharing data, along with the frequent changes, makes latency not a useful metric to use as link weight. Instead, we chose to use hopcount to determine what path is the shortest. In that case, Dijkstra functions similar to BFS. Since BFS does not have the overhead of sorting its list of nodes that needs to be explored, we prefer BFS over Dijkstra. Between BFS and DFS, the choice goes to BFS, as DFS can spend much time exploring a path while it took the 'wrong turn' and missed the destination, while BFS incrementally grows the search space, ensuring that it does not take more steps than necessary. Finally, to reduce the search space of BFS, we implemented BFS in a bidirectional fashion.

6.2 Latency Triplets

What is a good way to calculate latency between ASes? For the sake of this discussion, let our path be AS1–AS2–AS3. Then, an intuitive calculation of latency where only the "edge latency" is taken into account would be: latency between AS1 and AS3 = latency between AS1 and AS2 + latency between AS2 and AS3. However, this treatment does not take into account the fact that the transferring of a packet from one edge of AS2 to the other edge of AS2 takes time as well.

A way to account for the latency *within* AS2 in the total latency is to define latency between AS1 and AS3 as: the latency between AS1 and AS2 + the latency between AS2 and AS3 + the latency *within* AS2. This triplet of values can account for the total latency in a more complete way than the intuitive approach does.

The latency triplets approach has its own problems, as not all packets take the same path through an intermediate AS, but using latency triplets could make latency estimation a bit more accurate. The latency triplets ended up not being relevant for this thesis, but the idea was still worth sharing for other people to incorporate in their work, which is why it is included here.

6.3 Information Completeness

The algorithms described in this thesis can only calculate paths if the model of the network contains information from the NIP about what features are supported by which AS. The NIP can only provide that information if the ISPs that manage those ASs provide that information to the NIP. Thus, the main question is: *what incentive do ISPs have to provide network information to the NIP?* We argue that there are three main benefits for an ISP to providing information to the NIP, which we explain in the following sections.

6.3.1 New Marketing Opportunities

Publicly displaying the features an ISP supports provides a visible way for that ISP to distinguish itself from other ISPs that do not support as many features, allowing clear marketing for new customer groups. The clear marketing consists of communicating to potential users about which features are supported. Users can then compare the offering of one ISP with the offering of another ISP and choose the ISP that suits them best. Thus, ISPs that go out of their way to support certain features now have a meaningful way to communicate their supported features to potential new users.

6.3.2 New Basis for Peering Relationships

Availability of supported features provides clear metrics that can be used as a new basis to determine with what networks an ISP wants to enter a peering relationship. An ISP can now set up a peering relationship with another ISP if they both support a certain set of features, allowing their users to enjoy network paths that support features they care about.

Further, an anonymized database of path requests from users allows ISPs to analyze what demands users have from their networks, which can lead to new insights into the types of services that ISPs can offer. These insights can spark new collaborations between ISPs to form a constellation of ISPs that together offer routing that supports a shared set of features over a part of the Internet where many companies and users demand these features. This cooperation can lead to more customers for the ISPs that join such a constellation, as a high availability of routes that adhere to the jointly supported features is a great selling point. This collaboration is enabled by the clear definition of requirements and the availability of information about who does and does not support these features.

6.3.3 New Basis for Extra Revenue

By providing their information to the NIP, ISPs allow customers to see what features an ISP supports. The ISP can use this information to justify asking for a higher compensation. This higher compensation can be clearly linked to the well-defined features that an ISP supports, resulting in customers understanding what the higher price is based on. On top of that, the extra peering opportunities and extra customers explained before provide extra revenue as well. Thus, providing information to the NIP allows ISPs to generate more revenue.

6.4 Distribution of Features

The distribution of features determines how the features are distributed over the nodes in the graph. In the experiments, a uniform distribution was used, and this section explains why.

A uniform distribution means that the number of features for each node is a randomly selected number between the given minimum and maximum, and that number of features is randomly selected from the total pool of features according to a uniform distribution. This distribution was chosen to ensure that the distribution of features has the least possible influence on the results of the heuristic. Other distributions are possible as well. For example, one could map the number of supported features of each node to the node degree, where a node with a higher degree supports more features, and a node with low degree supports little features. Or one could increase the likelihood that a node supports a feature depending on how many of its neighbours support that feature as well, simulating peering relationships based on supported features. However, the main disadvantage of these distributions is the way they reduce the solution space.

If a distribution causes many nodes to support little features, the range that the score of a path that uses these nodes can take on becomes less. This lower score range means that the average score of the paths created by the heuristic goes down. However, the average score of the globally optimal path goes down as well[1]. Because of this simultaneous decrease, the difference in score between the globally optimal path and the heuristic path becomes smaller, which makes it harder to get clear insight into the performance of the heuristic relative to the globally optimal path. When the solution space is small, differences are less pronounced, and spotting a trend or surprising differences between graph types or different configurations becomes harder.

Instead, using a uniform distribution ensures a large solution space, even for low-degree nodes. This large solution space ensures that the potential difference in score between the heuristic path and the globally optimal path is large, which facilitates a more clear and accurate evaluation of the performance of the heuristic.

[1] This simultaneous decrease is assuming that there is no path around these low-feature nodes, which is, for example, the case when the start or end node is one of these low-feature nodes.

Chapter 7

Related Work

Previous work has been done on the subject of converting user requirements into network paths. This chapter covers that work by introducing several domains, showing what relevant work has been done in each domain, and examining whether these works solve the problem that this thesis aims to solve. Section 7.1 discusses Constraint-Based Routing, Section 7.2 covers Intent-Based Networking, Section 7.3 looks at Scalability, Control, and Isolation On Next-Generation Networks (SCION), Section 7.4 discusses Path Aware Networking, Section 7.5 gives an overview of the User Controlled Internet Protocol, and finally Section 7.6 covers Policy Based Routing.

7.1 Constraint-Based Routing

Related work on routing algorithms in the context of networking and user requirements has mainly focused on routing with constraints, where the goal is to find a route that satisfies Quality of Service (QoS) constraints[1] [15, 47, 56, 78] or to find a route that stays within some constraint while being optimal [20, 27, 38, 48, 53, 67, 68].

These algorithms do not solve the problem of this thesis, because the routing in this thesis is constrained by qualitative, binary constraints. Such metrics cannot be optimized: one either complies with the criterion, or one does not. Only after these qualitative metrics are fulfilled, one can apply the techniques in these papers.

7.2 Intent-Based Networking

The problem of this thesis focuses on finding network paths based on requirements. One way to find these paths is through intent-based networking. Intent-Based Networking is a framework that allows users to specify operational goals that the network should fulfill, without having to understand how the network works [16]. A recent development in this field is the use of natural language processing (NLP) as a form of input [83]. This form of input can be used to allow network operators and users to define their goals for the network using natural language. This information is fed to a system that can recognize the commands, convert the commands into technical instructions and execute them. An example is Merlin, which converts high-level instructions in natural language to network management instructions [72]. Merlin does not solve the problem that this thesis is aimed at, as Merlin is explicitly aimed at network operators, and requires considerable networking knowledge to use. Additionally, Merlin provides path selection based on minimum bandwidth, while this thesis aims to support other types of requirements.

Another example of a conversion from natural language to network policies is Lumi, which is a system that takes natural language commands and converts them into operational instructions to inspect or change a network [44]. After being confirmed by a human operator, these network operations are deployed in the network. One of the intermediate steps between the input of the natural language commands and the output of network operations is converting the natural language to Network Intent LanguagE (Nile), which is an abstraction layer between natural language intents and network configuration commands [44]. Nile is similar enough to natural language to make it human-readable, but structured enough to facilitate automated conversion to network operations. This combination makes Nile very versatile.

Meijer et al. use Nile as an encoding for network path requirements [58]. They created a syntax and interpreter to define requirements for a network path. The system by Meijer et al. works as follows: first a user inserts their requirements in natural language, which are interpreted by the Rasa chatbot[2]. These requirements are then converted to a custom path

[1] Quality of Service (QoS) refers to how network packets are handled (in terms of being prioritized or being dropped) when too many packets arrive at a router at the same time [5]. The most popular QoS mechanism is Differentiated Services [8].

[2] `https://rasa.com/docs/rasa/`

requirement syntax designed by Meijer et al., or to Nile[3]. The path requirements are used to create a list of network nodes that comply with the requirements, after which a path is found using Breadth-First Search (BFS) over the subset of selected nodes.

The work by Meijer et al. has some similarities with the work of this thesis. However, the path requirement selection mechanism in the work of Meijer et al. only includes strict-level requirements. The work in this thesis improves on that by allowing a user to differentiate between strict and best effort requirements, and this thesis goes beyond the work by Meijer et al. by finding the path that satisfies the most possible best effort requirements, while the work by Meijer et al. does not. This more fine-grained level of control and the maximization of best effort requirements gives the work in this thesis more flexibility than the approach by Meijer et al.. Further, Meijer et al. assume the user knows all the information about the network needed to calculate the path, while this thesis does not require this knowledge. Instead, this thesis assumes the network information is provided by the Network Information Plane (NIP), which does not places the burden on the user to provide all the network information.

Bezahaf et al. [7] use Nile as an intermediate step as well. They use Dialogflow[4] (a natural language processing service by Google) to convert natural language to Nile, which is subsequently converted to API calls to the Open Network Operating System (ONOS) [6], a distributed Software-Defined Networking (SDN) control platform. The system by Bezahaf et al. is mainly focused on connecting these services into a working system. While it is an interesting work, their system requires extensive networking knowledge to operate, just like Merlin and Lumi, whereas the work in this thesis is aimed at end users who lack the domain-specific knowledge.

Another use of Nile as an intermediate step is P4I/O [70], an SDN approach to Intent-Based Networking where natural language that expresses network intents is converted to an extended version of Nile, which is then converted into Programming Protocol-independent Packet Processors (P4) code. This approach allows network operators to directly program the data-plane using natural language instructions. P4I/O does not allow users to generate paths based on their constraints, thus it does not solve the problem that this thesis focuses on.

7.3 SCION

Scalability, Control, and Isolation On Next-Generation Networks (SCION) is a clean-slate Internet architecture [84] by Zhang et al.. It has security by design and it aims to provide availability even with active malicious actors present on the network. This summary focuses on SCION's approach to routing.

Routing in SCION is done by storing the route that packets take inside the packet. This packet-stored route is referred to as the Packet Carried Forwarding State (PCFS). The PCFS allows senders to specify the route via which a packet should travel through the Internet. The routing in SCION is done on the level of an Autonomous System (AS). ASs are grouped into logical units, referred to as Isolation Domains (ISDs). This grouping allows routing to be simplified, as the best path and some backup paths from one ISD to another only have to be calculated once for all traffic between these two ISDs.

Within each ISD there is a group of ASs referred to as the core. The core is a small group of trusted ASs. These trusted ASs form the connection between all other ASs. When one AS wants to route to another AS that is in the same ISD, the path goes from the source AS first to the core, and from the core the path goes to the destination AS. Thus, the core knows how to reach all ASs in the ISD, and all ASs in the ISD know how to reach the core.

Routes from and to the core within one ISD are calculated using Path Construction Beacons (PCB). These PCBs are sent from the core to all ASs in the ISD. Once an AS receives the PCB, it does three things; first, it selects a number of paths from the core to itself and publishes those to the path server, a server that is maintained by the ISD core. These paths are the up-segments (path segment from the AS to the core) and down-segments (path segment from the core to the AS) for that AS. Second, the AS adds itself to the PCB as the next hop. Third, the AS forwards the updated PCB to its neighbours. The ASs in an ISD are organized as a Directed Acyclic Graph (DAG) with the ISD core as the root. This DAG structure ensures that the PCB packets will eventually reach a leaf ISD, causing the process to terminate.

[3]The conversion to Nile was added to make the system more generalizable, as Nile is used more often than the custom syntax by Meijer et al..

[4]`https://dialogflow.cloud.google.com`

Routes between multiple ISDs are calculated by the cores of these ISDs. The cores exchange routing information and use that to construct routing tables. This process works similar to Border Gateway Protocol. The scalability is less of an issue as the set of core ASs will be small (i.e. a few hundred) and remain stable.

When an AS wants to reach another AS, it first selects one of its own up paths. Next, the source AS queries the path server for down-segments from the core to the destination AS. The ASs in the core form a clique, so the combination of an up-segment from source to core and a down-segment from core to destination forms a complete end-to-end path. If the up-segment and the down-segment have overlap, the packet does not go "up-stream" and "down-stream" over the overlapping segment (which would make the packet end up on the same node) but instead the packet skips that overlapping sub-segment and directly takes the down segment towards the destination. A possible improvement of this scheme could be to use contraction hierarchies [34] to prevent the situation where two ASs that are close end up in a separate branch of the DAG, resulting in a relatively long path (from source to the core to the destination) compared to the (short) path from the source directly to the destination that would be used in a non-SCION network.

Finally, the authors include the possibility of the end user to choose which of the up-segments and down-segments is used for the route; next to letting the gateway in an endpoint AS decide which paths to use, a host can also negotiate with its provider AS about supporting customized path selection policies. This negotiation could allow some level of controllable routing in the SCION routing scheme. However, SCION's routing does not find paths based on user requirements, and thus it does not solve the problem that this thesis focuses on. The approach used in this thesis could, however, be extended to be used in SCION, as SCION still uses ASes as logical units. The main adjustment that would be needed is path finding, as SCION has its own approach to that.

7.4 Path Aware Networking

Path-Aware Networking (PAN) is a framework that enables endpoints to receive information about the paths their data is flowing through, and endpoints can act on that information as well [76]. This framework gives applications some control over the path that their data uses. PAN has been implemented in SCION by Davidson et al..

Davidson et al. have used SCION to create a browser extension that allows users to define via which countries they want their data to flow [19]. Their browser extension allows users to determine through which geographical areas their traffic is supposed to flow, which also allows users to define what geographical areas to avoid. This work intersects with the work in this thesis. However, the path creation is a minor focus in the work by Davidson et al.. They allow the user the manually block or allow ISDs (groups of ASs), and then the authors leverage SCION's built-in path-awareness to find a path. The problem is that the user needs to decide which ISDs to block, which requires network information that the end user rarely has. The focus of the authors is the connection of their browser extension to SCION, and in doing so they do not solve the problem of this thesis. However, a browser extension could be used as a user interface for the result of this thesis.

Further, Xiang et al. propose a system where ASs expose information about themselves, which can be used by users to find a policy-compliant route [81]. However, as with the previous system, the user has to do all the work. Thus, this work does not solve the problem that this thesis focuses on as well.

7.5 User Controlled Internet Protocol

On the procotol level, User Controlled Internet Protocol (UCIP) describes a protocol that allows users to exactly define via which route their data flows [49]. The routing in UCIP is done by Private Source Routing (PSR), which works similar to Source Routing (SoR). The route that is stored in the packet consists of a list of ASs. Once a packet containing such a list is sent on its way, each router that receives the packet forwards the packet in the direction of the next domain on the list. To ensure that this route is not visible to anyone who gets access to the packet, the identifier (ID) of each location is encrypted with the public key of the location one hop earlier

in the path. This encryption ensures that each hop can only see the next destination, but no other destinations, because each hop can only decrypt the identifier of the next hop using its own private key, but it cannot decrypt the other locations. This system does not require a key-exchange, as the public keys are available for the source to encrypt the hops with, and the hops that need to see the address of the next hop can decrypt that hop with their private key.

A problem with PSR is its scalability: The first PSR packet that contains the route also contains a flow ID, which is used to route subsequent packets of the same flow. Routing subsequent packets in the flow is done by installing the flow ID into the routing table. By storing the flow ID in the routing table, the path does not have to be stored in subsequent packets of that flow, reducing the packet size of these subsequent packets. However, the downside of this route-by-ID approach is that it requires per-flow state, which does not scale well, especially in the core network of the Internet.

One could see PSR as SoR with some additional layers of privacy and security. This thesis focuses on selecting a path, while the actual routing is done via SoR. Thus, UCIP does not solve the problem that this thesis focuses on.

7.6 Policy Based Routing

Several previous works have focused on routing based on policies. One approach attempts to find an Internet route for an AS that is most in line with the preferences of the source AS. This is done by giving each edge a utility score and finding the set of paths that maximizes this score [31]. This approach does not look into finding the path for which all nodes satisfy the largest common set of requirements. Instead, this work reduces these preferences to one number that is assigned to each edge (the utility score). That approach does not solve the Maximum Path Requirement Intersection (MPRI) problem. Another work focused on assigning preferences to nodes and finding routes that maximized the total preference [14]. However, here the focus is not on the maximization of features satisfied by a path, but on visiting the nodes that provide the highest cumulative score. Thus, that work does not solve the problem of this thesis as well.

Chapter 8

Conclusions

This thesis addresses the problem of calculating routes based on user requirements. First, the general problem was defined as finding a path from source to destination that adheres to the given strict requirements and satisfies as many of the given best effort requirements as possible. From that general problem, the sub-problem of finding the path that satisfies the most best effort requirements was labelled the Maximum Path Requirement Intersection (MPRI) problem, and a proof that MPRI is NP-hard was given.

Two algorithms to solve the general problem were proposed. A heuristic algorithm with finetuned limits that control the search space, which allows it to find paths in little time, and an exact algorithm that is guaranteed to find the globally optimal path. The performance of the heuristic algorithm is evaluated in five experiments, which answer the following five questions.

1. What is the influence of the number of features and requirements on the performance of the heuristic?
2. What is the influence of the ratio between the minimum and maximum number of features supported by each AS on the performance of the heuristic?
3. How well does the heuristic scale?
4. On what type of graph topologies is the heuristic most effective?
5. How well does the heuristic perform in a realistic scenario?

The results inform the following main findings. First, a lower number of features and requirements makes the heuristic faster, while a higher number of features and requirements makes it slower. Second, the limit values for the heuristic are almost exactly the same for different ratios, which tells us that the exact choice for the ratio does not significantly impact the results. Third, the heuristic scales polynomially in the length of the shortest path, which indicates that the heuristic is most effective in graph types where the average shortest path does not grow with the graph size. Fourth, the heuristic is most effective on graphs that have a power-law degree distribution and least effective on grid-like graphs. Finally, in a realistic scenario the heuristic runs quickly while performing close to optimal.

8.1 Future work

Future work could look into the following things. First, the format for the path requirements currently uses a custom JSON format. To make this work connect with other work in the area of converting user requirements in routing requests, the system could be modified to use the Nile [44] format. Using Nile enables the embedding of this system into other systems, which can enable broader applications of the work in this thesis.

Second, future work can look into extending this system to also support non-binary requirements. Supporting non-binary requirements allows the system to satisfy more refined path requests, and allows users to express requirements even when they are not just true or false. A first step in this direction is the insight that non-binary requirements can be modeled as binary requirements by introducing a requirement that requires a certain value to be above or below a threshold.

Third, future work could look into dynamically determining the best limits within a certain runtime threshold while running. Currently the heuristic utilizes the full search space as determined by the limit values which are given as input parameters. Within that search space the heuristic then satisfies the path requests, and based on the runtime (or any other metric) the best limit values are chosen. If instead the heuristic can dynamically determine the best set of limit values based on a given metric while running, the need to perform the limit stage operations is removed, which makes the heuristic more versatile and user friendly.

Fourth, this work can be extended to a system where the best effort requirements have weights. Currently all best effort requirements are equally important, and the score is equal to the number of best effort requirements that is satisfied by the path. However, if the best effort requirements have weights, the score could be the sum of the weights of the satisfied best effort requirements, and the goal becomes maximizing the total weight (which leads to a optimization problem similar to the MPRI problem) or to minimize the total weight, which allows the user to express a 'penalty' for using a certain requirement. This extension can lead to new problems and, hopefully, new solutions.

Fifth, future work could look into precomputing globally optimal paths based on often-used sets of requirements and storing them in an efficient data structure. These precomputed paths can be used to satisfy requests even faster. Research has to be done into whether trading runtime for storage is worth the trade-off, especially since these precomputed paths would need to be installed on each user's machine to properly benefit from the faster time, and precomputation might not be worth it due to frequent changes of the network.

Sixth, future work could look into converting the code of the heuristic to other programming languages that are more efficient than Python in terms of runtime, such as C++, Rust, or Go. When doing this conversion, the often-used set intersection operation could possibly also be improved by using techniques from [3] and [23].

Seventh, future work could focus on making the heuristic work in a parallelized fashion. Parallelization could be used to improve the performance of the bidirectional BFS algorithm used to find the initial shortest path. Next to that, parallelization could be used to speed up the optimization of the path using detours. If the initial shortest path is broken up into segments, then each of these segments can be optimized independently from each other by calculating detours for that segment and replacing the segment with the best detour. Once each segment is optimized, the results can be combined into the best overall path. This does not work directly, as the set of supported requirements of one segment might not be compatible with the set of supported requirements of another segment. Thus, some communication between threads is needed, but this could prove a promising direction for future improvements.

Eighth, future work could look into creating a way to verify whether ASes really support the features that they claim they do. This is more related to the Network Information Plane (NIP), but it would be an addition that makes this work more relevant and usable. This verification could be similar to this MANRS validator (`https://github.com/manrs-tools/manrs-validator`) or the verification system from the HSTS preload list on `https://hstspreload.org`, but then adapted for the features and requirements of autonomous systems in a routing context.

References

[1] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Hierarchical Hub Labelings for Shortest Paths. In Leah Epstein and Paolo Ferragina, editors, *Algorithms – ESA 2012*, Lecture Notes in Computer Science, pages 24–35, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-33090-2_4.

[2] Firkhan Ali Bin Hamid Ali. A study of technology in firewall system. In *2011 IEEE Symposium on Business, Engineering and Industrial Applications (ISBEIA)*, pages 232–236, September 2011. URL: https://ieeexplore.ieee.org/abstract/document/6088813, doi:10.1109/ISBEIA.2011.6088813.

[3] Ricardo Baeza-Yates. A Fast Set Intersection Algorithm for Sorted Sequences. In Suleyman Cenk Sahinalp, S. Muthukrishnan, and Ugur Dogrusoz, editors, *Combinatorial Pattern Matching*, Lecture Notes in Computer Science, pages 400–408, Berlin, Heidelberg, 2004. Springer. doi:10.1007/978-3-540-27801-6_30.

[4] Fred Baker. Requirements for IP Version 4 Routers. Request for Comments RFC 1812, Internet Engineering Task Force, June 1995. Num Pages: 175. URL: https://datatracker.ietf.org/doc/rfc1812, doi:10.17487/RFC1812.

[5] Fred Baker, David L. Black, Kathleen Nichols, and Steven L. Blake. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. Request for Comments RFC 2474, Internet Engineering Task Force, December 1998. Num Pages: 20. URL: https://datatracker.ietf.org/doc/rfc2474, doi:10.17487/RFC2474.

[6] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, and Guru Parulkar. ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking (HotSDN)*, pages 1–6, Chicago Illinois USA, August 2014. ACM. URL: https://dl.acm.org/doi/10.1145/2620728.2620744, doi:10.1145/2620728.2620744.

[7] Mehdi Bezahaf, Eleanor Davies, Charalampos Rotsos, and Nicholas Race. To All Intents and Purposes: Towards Flexible Intent Expression. *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pages 31–37, June 2021. . Place: Tokyo, Japan. Publisher: IEEE. URL: https://ieeexplore.ieee.org/document/9492554/, doi:10.1109/NetSoft51509.2021.9492554.

[8] David L. Black, Zheng Wang, Mark A. Carlson, Walter Weiss, Elwyn B. Davies, and Steven L. Blake. An Architecture for Differentiated Services. Request for Comments RFC 2475, Internet Engineering Task Force, December 1998. Num Pages: 36. URL: https://datatracker.ietf.org/doc/rfc2475, doi:10.17487/RFC2475.

[9] Geoff Boeing. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems. Publisher: Elsevier.*, 65:126–139, September 2017. URL: https://www.sciencedirect.com/science/article/pii/S0198971516303970, doi:10.1016/j.compenvurbsys.2017.05.004.

[10] Geoff Boeing. Street Network Models and Measures for Every U.S. City, County, Urbanized Area, Census Tract, and Zillow-Defined Neighborhood. *Urban Science*, 3(1):28, March 2019. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute. URL: https://www.mdpi.com/2413-8851/3/1/28, doi:10.3390/urbansci3010028.

[11] Adi Botea, Martin Müller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *J. Game Dev.*, 1(1):1–30, 2004. Publisher: Citeseer.

[12] Anna Bouch, Allan Kuchinsky, and Nina Bhatti. Quality is in the eye of the beholder: meeting users' requirements for Internet quality of service. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 297–304, The Hague The Netherlands, 2000. ACM. URL: https://dl.acm.org/doi/10.1145/332040.332447, doi:10.1145/332040.332447.

[13] M. Caesar and J. Rexford. BGP routing policies in ISP networks. *IEEE Network*, 19(6):5–11, November 2005. URL: https://ieeexplore.ieee.org/abstract/document/1541715, doi:10.1109/MNET.2005.1541715.

[14] Chi-kin Chau. Policy-based routing with non-strict preferences. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, pages 387–398, New York, NY, USA, August 2006. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/1159913.1159957, doi:10.1145/1159913.1159957.

[15] J. Noel Chiappa, Martha E. Steenstrup, and Isio M. Castineyra. The Nimrod Routing Architecture. Request for Comments RFC 1992, Internet Engineering Task Force, August 1996. Num Pages: 27. URL: https://datatracker.ietf.org/doc/rfc1992, doi:10.17487/RFC1992.

[16] Alexander Clemm, Laurent Ciavaglia, Lisano Zambenedetti Granville, and Jeff Tantsura. Intent-Based Networking - Concepts and Definitions. Request for Comments RFC 9315, Internet Engineering Task Force, October 2022. Num Pages: 23. URL: https://datatracker.ietf.org/doc/rfc9315, doi:10.17487/RFC9315.

[17] Federal Trade Commission. A Look At What ISPs Know About You: Examining the Privacy Practices of Six Major Internet Service Providers. Technical report, 2021. URL: https://www.ftc.gov/system/files/documents/reports/look-what-isps-know-about-you-examining-privacy-practices-six-major-internet-service-providers/p195402_isp_6b_staff_report.pdf.

[18] Alissa Cooper, Hannes Tschofenig, Bernard D. Aboba, Jon Peterson, John Morris, Marit Hansen, and Rhys Smith. Privacy Considerations for Internet Protocols. Request for Comments RFC 6973, Internet Engineering Task Force, July 2013. Num Pages: 36. URL: https://datatracker.ietf.org/doc/rfc6973, doi:10.17487/RFC6973.

[19] A. Davidson, M. Frei, M. Gartner, H. Haddadi, A. Perrig, J. Subirà Nieto, P. Winter, and F. Wirz. Tango or square dance? how tightly should we integrate network functionality in browsers? In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, HotNets '22, pages 205–212, New York, NY, USA, November 2022. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3563766.3564111, doi:10.1145/3563766.3564111.

[20] H. De Neve and P. Van Mieghem. TAMCRA: a tunable accuracy multiple constraints routing algorithm. *Computer Communications*, 23(7):667–679, March 2000. URL: https://www.sciencedirect.com/science/article/pii/S014036649900225X, doi:10.1016/S0140-3664(99)00225-X.

[21] Amogh Dhamdhere and Constantine Dovrolis. The Internet is flat: modeling the transition from a transit hierarchy to a peering mesh. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 1–12, New York, NY, USA, November 2010. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/1921168.1921196, doi:10.1145/1921168.1921196.

[22] Sara Dickinson, Benno Overeinder, Roland van Rijswijk-Deij, and Allison Mankin. Recommendations for DNS Privacy Service Operators. Request for Comments RFC 8932, Internet Engineering Task Force, October 2020. Num Pages: 34. URL: https://datatracker.ietf.org/doc/rfc8932, doi:10.17487/RFC8932.

[23] Bolin Ding and Arnd Christian König. Fast Set Intersection in Memory, March 2011. arXiv:1103.2409 [cs]. URL: http://arxiv.org/abs/1103.2409, doi:10.48550/arXiv.1103.2409.

[24] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, Krishna Gummadi, Ratul Mahajan, and Stefan Saroiu. Glasnost: Enabling End Users to Detect Traffic Differentiation. In *Proceedings of 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, April 2010. URL: https://www.microsoft.com/en-us/research/publication/glasnost-enabling-end-users-to-detect-traffic-differentiation/.

[25] Ben Du, Cecilia Testart, Romain Fontugne, Gautam Akiwate, Alex C. Snoeren, and Kimberly C. Claffy. Mind your MANRS: measuring the MANRS ecosystem. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, pages 716–729, New York, NY, USA, October 2022. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3517745.3561419, doi:10.1145/3517745.3561419.

[26] Ahmed Elmokashfi, Amund Kvalbein, and Constantine Dovrolis. On the Scalability of BGP: The Role of Topology Growth. *IEEE Journal on Selected Areas in Communications*, 28(8):1250–1261, October 2010. URL: https://ieeexplore.ieee.org/abstract/document/5586438, doi:10.1109/JSAC.2010.101003.

[27] Funda Ergun, Rakesh Sinha, and Lisa Zhang. An improved FPTAS for Restricted Shortest Path. *Information Processing Letters*, 83(5):287–291, September 2002. URL: https://www.sciencedirect.com/science/article/pii/S0020019002002053, doi:10.1016/S0020-0190(02)00205-3.

[28] Ksenia Ermoshina, Benjamin Loveluck, and Francesca Musiani. A market of black boxes: The political economy of Internet surveillance and censorship in Russia. *Journal of Information Technology & Politics*, 19(1):18–33, January 2022. Publisher: Routledge _eprint: https://doi.org/10.1080/19331681.2021.1905972. doi:10.1080/19331681.2021.1905972.

[29] John W. Essam and Michael E. Fisher. Some Basic Definitions in Graph Theory. *Reviews of Modern Physics*, 42(2):271–288, 1970. URL: https://link.aps.org/doi/10.1103/RevModPhys.42.271, doi:10.1103/RevModPhys.42.271.

[30] Deborah Fallows. The Internet and Daily Life. *PEW INTERNET & AMERICAN LIFE PROJECT*, August 2004. URL: https://www.pewresearch.org/internet/wp-content/uploads/sites/9/media/Files/Reports/2004/PIP_Internet_and_Daily_Life.pdf.

[31] Joan Feigenbaum, Rahul Sami, and Scott Shenker. Mechanism design for policy routing. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, PODC '04, pages 11–20, New York, NY, USA, July 2004. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/1011767.1011770, doi:10.1145/1011767.1011770.

[32] Clarence Filsfils, Stefano Previdi, Les Ginsberg, Bruno Decraene, Stephane Litkowski, and Rob Shakir. Segment Routing Architecture. Request for Comments RFC 8402, Internet Engineering Task Force, 2018. Num Pages: 32. URL: https://datatracker.ietf.org/doc/rfc8402, doi:10.17487/RFC8402.

[33] Dennis F. Galletta, Raymond M. Henry, Scott McCoy, and Peter Polak. When the Wait Isnt So Bad: The Interacting Effects of Website Delay, Familiarity, and Breadth. *Information Systems Research*, 17(1):20–37, March 2006. doi:10.1287/isre.1050.0073.

[34] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In Catherine C. McGeoch, editor, *Experimental Algorithms*, Lecture Notes in Computer Science, pages 319–333, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-68552-4_24.

[35] Ding-Ding Han, Jiang-Hai Qian, and Jin-Gao Liu. Network topology and correlation features affiliated with European airline companies. *Physica A: Statistical Mechanics and its Applications*, 388(1):71–81, January 2009. URL: https://www.sciencedirect.com/science/article/pii/S0378437108007942, doi:10.1016/j.physa.2008.09.021.

[36] Daniel Harabor and Alban Grastien. Improving Jump Point Search. *Proceedings of the International Conference on Automated Planning and Scheduling*, 24:128–135, May 2014. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13633, doi:10.1609/icaps.v24i1.13633.

[37] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. URL: https://ieeexplore.ieee.org/abstract/document/4082128?casa_token=8h8bUKrpYEYAAAAA%3AJMIi5XySYA2a-KCYIE7HECb4LxUcaMIqoELdV4G0A6oHxMLa3G6Ch9CNhNbKj_gY7o3paAg8Og, doi:10.1109/TSSC.1968.300136.

[38] Refael Hassin. Approximation Schemes for the Restricted Shortest Path Problem. *Mathematics of Operations Research*, 17(1):36–42, February 1992. Publisher: INFORMS. URL: https://pubsonline.informs.org/doi/abs/10.1287/moor.17.1.36, doi:10.1287/moor.17.1.36.

[39] Cristian Hesselman, Paola Grosso, Ralph Holz, Fernando Kuipers, Janet Hui Xue, Mattijs Jonker, Joeri de Ruiter, Anna Sperotto, Roland van Rijswijk-Deij, Giovane C. M. Moura, Aiko Pras, and Cees de Laat. A Responsible Internet to Increase Trust in the Digital World. *Journal of network and systems management*, 28(4):882–922, October 2020. Publisher: Springer. URL: https://research.utwente.nl/en/publications/a-responsible-internet-to-increase-trust-in-the-digital-world, doi:10.1007/s10922-020-09564-7.

[40] Geoff Houston, Philip Smith, and Tony Bates. CIDR Report, 2024. Publication Title: CIDR report. URL: https://www.cidr-report.org/as2.0/.

[41] Geoff Huston. BGP in 2022 – the routing table, January 2023. URL: https://blog.apnic.net/2023/01/06/bgp-in-2022-the-routing-table/.

[42] Internet Assigned Numbers Authority (IANA). Border Gateway Protocol (BGP) Extended Communities, 2023. URL: https://www.iana.org/assignments/bgp-extended-communities/bgp-extended-communities.xhtml.

[43] Md Anwarul Islam and Kazi Mostak Gausul Hoq. Community Internet Access in Rural Areas: A Study on Community Information Centres in Bangladesh. *Malaysian Journal of Library and Information Science*, 15(2):109–124, August 2010. Number: 2. URL: https://samudera.um.edu.my/index.php/MJLIS/article/view/6936.

[44] A. Jacobs, R. Pfitscher, R. H. Ribeiro, R. Ferreira, L. Granville, W. Willinger, and Sanjay G. Rao. Hey, Lumi! Using Natural Language for Intent-Based Network Management. In *USENIX Annual Technical Conference*, 2021. URL: https://www.semanticscholar.org/paper/Hey%2C-Lumi!-Using-Natural-Language-for-Intent-Based-Jacobs-Pfitscher/9e1db4bd7c01aad02849966068e0d74d36e603c9.

[45] Milena Janic, Jan Pieter Wijbenga, and Thijs Veugen. Transparency Enhancing Tools (TETs): An Overview. In *2013 Third Workshop on Socio-Technical Aspects in Security and Trust*, pages 18–25, June 2013. ISSN: 2325-1697. doi:10.1109/STAST.2013.11.

[46] George M. Jones. Operational Security Requirements for Large Internet Service Provider (ISP) IP Network Infrastructure. Request for Comments RFC 3871, Internet Engineering Task Force, September 2004. Num Pages: 81. URL: https://datatracker.ietf.org/doc/rfc3871, doi:10.17487/RFC3871.

[47] A. Juttner, B. Szviatovski, I. Mecs, and Z. Rajko. Lagrange relaxation based method for the QoS routing problem. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 2, pages 859–868 vol.2, April 2001. ISSN: 0743-166X. doi:10.1109/INFCOM.2001.916277.

[48] A. Karaman. Constraint-Based Routing in Traffic Engineering. In *2006 International Symposium on Computer Networks*, pages 1–6, June 2006. doi:10.1109/ISCN.2006.1662507.

[49] Morteza Kheirkhah, Truong Khoa Phan, XinPeng Wei, David Griffin, and Miguel Rio. UCIP: User Controlled Internet Protocol. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 279–284, 2020. doi:10.1109/INFOCOMWKSHPS50562.2020.9162833.

[50] Tom Killalea. Recommended Internet Service Provider Security Services and Procedures. Request for Comments RFC 3013, Internet Engineering Task Force, November 2000. Num Pages: 13. URL: https://datatracker.ietf.org/doc/rfc3013, doi:10.17487/RFC3013.

[51] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, June 1993. URL: https://ieeexplore.ieee.org/abstract/document/234851, doi:10.1109/90.234851.

[52] Richard E. Korf. Depth-limited search for real-time problem solving. *Real-Time Systems*, 2(1-2):7–24, May 1990. URL: http://link.springer.com/10.1007/BF01840464, doi:10.1007/BF01840464.

[53] FA Kuipers, T Korkmaz, M Krunz, and P Van Mieghem. A Review of Constraint-Based Routing Algorithms. *IEEE Communications Magazine*, 40(12):50, June 2002. Publisher: IEEE Communications Society.

[54] Mirjam Kühne and Vasco Asturiano. Update on AS Path Lengths Over Time, 2012. URL: https://labs.ripe.net/author/mirjam/update-on-as-path-lengths-over-time/.

[55] Matthew Luckie, Robert Beverly, Ryan Koga, Ken Keys, Joshua A. Kroll, and Kimberly C. Claffy. Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 465–480, New York, NY, USA, 2019. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3319535.3354232, doi:10.1145/3319535.3354232.

[56] Qingming Ma and Peter Steenkiste. Quality-of-Service Routing for Traffic with Performance Guarantees. In Andrew T. Campbell and Klara Nahrstedt, editors, *Building QoS into Distributed Systems: IFIP TC6 WG6.1 Fifth International Workshop on Quality of Service (IWQOS '97), 21–23 May 1997, New York, USA*, IFIP — The International Federation for Information Processing, pages 115–126. Springer US, Boston, MA, 1997. doi:10.1007/978-0-387-35170-4_12.

[57] MANRS. MANRS Public API | MANRS Public API. URL: https://manrs.stoplight.io/docs/manrs-public-api/38c368e1d6b43-manrs-public-api.

[58] Anne-Ruth Meijer, Leonardo Boldrini, Ralph Koning, and Paola Grosso. User-driven Path Control through Intent-Based Networking. In *2022 IEEE/ACM International Workshop on Innovating the Network for Data-Intensive Science (INDIS)*, pages 9–19, November 2022. ISSN: 2831-3879. doi:10.1109/INDIS56561.2022.00007.

[59] Juniper Networks. segment-list | Junos OS | Juniper Networks, 2021. URL: https://www.juniper.net/documentation/us/en/software/junos/bgp/topics/ref/statement/segment-list-protocols-source-packet-routing.html.

[60] Juan-ivan Nieto-Hipolito, Jose-maria Barcelo-Ordinas, Oscar-ivan Lepe-Aldama, Jose-antonio Michel-Macarty, Juan-de-dios Sanchez-Lopez, and Haydee Melendez-Guillen. AS-top: A New Topology Generator at the Autonomous Systems Level. In *Electronics, Robotics and Automotive Mechanics Conference (CERMA'06)*, volume 2, pages 355–360, 2006. URL: https://ieeexplore.ieee.org/abstract/document/4019818?casa_token=q01buXLYSzEAAAAA%3AsKERaBexSWAciQ-EgjDEvzI-2_SJ4T8uab5u-HhQtmRaInIFFjym5Pm1C0J_ZsaPYlhgo7Sdiw, doi:10.1109/CERMA.2006.105.

[61] Nils J. Nilsson. *Principles of Artificial Intelligence*. Springer Science & Business Media, 1982.

[62] Abdullah Yasin Nur. Analysis of Autonomous System Level Internet Topology Graphs and Multigraphs. In *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–7, Dubai, United Arab Emirates, 2021. IEEE. URL: https://ieeexplore.ieee.org/document/9615677/, doi:10.1109/ISNCC52172.2021.9615677.

[63] Ricardo V. Oliveira, Beichuan Zhang, and Lixia Zhang. Observing the evolution of internet as topology. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 313–324, Kyoto Japan, 2007. ACM. URL: https://dl.acm.org/doi/10.1145/1282380.1282416, doi:10.1145/1282380.1282416.

[64] Christos Pappas, Raphael M. Reischuk, and Adrian Perrig. FAIR: Forwarding Accountability for Internet Reputability. In *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, pages 189–200, November 2015. ISSN: 1092-1648. URL: https://ieeexplore.ieee.org/abstract/document/7437128, doi:10.1109/ICNP.2015.22.

[65] Stefano Previdi, Clarence Filsfils, Bruno Decraene, Stephane Litkowski, Martin Horneffer, and Rob Shakir. Source Packet Routing in Networking (SPRING) Problem Statement and Requirements. Request for Comments RFC 7855, Internet Engineering Task Force, May 2016. Num Pages: 19. URL: https://datatracker.ietf.org/doc/rfc7855, doi:10.17487/RFC7855.

[66] Stefano Previdi, Clarence Filsfils, Acee Lindem, Arjun Sreekantiah, and Hannes Gredler. Segment Routing Prefix Segment Identifier Extensions for BGP. Request for Comments RFC 8669, Internet Engineering Task Force, December 2019. Num Pages: 15. URL: https://datatracker.ietf.org/doc/draft-ietf-idr-bgp-prefix-sid-27, doi:10.17487/RFC8669.

[67] A. Puri and S. Tripakis. Algorithms for Routing with Multiple Constraints. *Proc. of AIPS'02, Toulouse, France*, April 2002.

[68] D.S. Reeves and H.F. Salama. A distributed algorithm for delay-constrained unicast routing. *IEEE/ACM Transactions on Networking*, 8(2):239–250, April 2000. doi:10.1109/90.842145.

[69] Yakov Rekhter, Susan Hares, and Tony Li. A Border Gateway Protocol 4 (BGP-4). Request for Comments RFC 4271, Internet Engineering Task Force, January 2006. Num Pages: 104. URL: https://datatracker.ietf.org/doc/rfc4271, doi:10.17487/RFC4271.

[70] Mohammad Riftadi and Fernando Kuipers. P4I/O: Intent-Based Networking with P4. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 438–443, June 2019. doi:10.1109/NETSOFT.2019.8806662.

[71] Andrew Sears, Julie A. Jacko, and Michael S. Borella. Internet delay effects: how users perceive quality, organization, and ease of use of information. In *CHI '97 extended abstracts on Human factors in computing systems looking to the future - CHI '97*, page 353, Atlanta, Georgia, 1997. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=1120212.1120430, doi:10.1145/1120212.1120430.

[72] Robert Soulé, Shrutarshi Basu, Parisa Jalili Marandi, Fernando Pedone, Robert Kleinberg, Emin Gun Sirer, and Nate Foster. Merlin: A Language for Provisioning Network Resources. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 213–226, Sydney Australia, December 2014. ACM. URL: https://dl.acm.org/doi/10.1145/2674005.2674989, doi:10.1145/2674005.2674989.

[73] Cisco Systems. Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper, 2018. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[74] Dan Tappan, Srihari R. Sangli, and Yakov Rekhter. BGP Extended Communities Attribute. Request for Comments RFC 4360, Internet Engineering Task Force, 2006. Num Pages: 12. URL: https://datatracker.ietf.org/doc/rfc4360, doi:10.17487/RFC4360.

[75] Paul S. Traina, John Scudder, and Danny R. McPherson. Autonomous System Confederations for BGP. Request for Comments RFC 5065, Internet Engineering Task Force, August 2007. Num Pages: 14. URL: https://datatracker.ietf.org/doc/rfc5065, doi:10.17487/RFC5065.

[76] Brian Trammell, Jean-Pierre Smith, and Adrian Perrig. Adding Path Awareness to the Internet Architecture. *IEEE Internet Computing*, 22(2):96–102, 2018. doi:10.1109/MIC.2018.022021673.

[77] Arun Viswanathan, Eric C. Rosen, and Ross Callon. Multiprotocol Label Switching Architecture. Request for Comments RFC 3031, Internet Engineering Task Force, January 2001. Num Pages: 61. URL: https://datatracker.ietf.org/doc/rfc3031, doi:10.17487/RFC3031.

[78] Zheng Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, September 1996. doi:10.1109/49.536364.

[79] Thomas Wirtgen, Tom Rousseaux, Quentin De Coninck, Nicolas Rybowski, Randy Bush, Laurent Vanbever, Axel Legay, and Olivier Bonaventure. xBGP: Faster Innovation in Routing Protocols. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 575–592, 2023. URL: https://www.usenix.org/conference/nsdi23/presentation/wirtgen.

[80] Eduardo C. Xavier. A note on a Maximum k-Subset Intersection problem. *Information Processing Letters*, 112(12):471–472, June 2012. URL: https://linkinghub.elsevier.com/retrieve/pii/S0020019012000750, doi:10.1016/j.ipl.2012.03.007.

[81] Qiao Xiang, Jingxuan Zhang, Kai Gao, Yeon-sup Lim, Franck Le, Geng Li, and Y. Richard Yang. Toward Optimal Software-Defined Interdomain Routing. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 1529–1538, July 2020. ISSN: 2641-9874. doi:10.1109/INFOCOM41043.2020.9155486.

[82] Da Yu, Yibo Zhu, Behnaz Arzani, Rodrigo Fonseca, Tianrong Zhang, Karl Deng, and Lihua Yuan. dShark: A General, Easy to Program and Scalable Framework for Analyzing In-network Packet Traces. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 207–220, 2019. URL: https://www.usenix.org/conference/nsdi19/presentation/yu.

[83] Engin Zeydan and Yekta Turk. Recent Advances in Intent-Based Networking: A Survey. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5, May 2020. ISSN: 2577-2465. doi:10.1109/VTC2020-Spring48590.2020.9128422.

[84] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G. Andersen. SCION: Scalability, Control, and Isolation on Next-Generation Networks. In *2011 IEEE Symposium on Security and Privacy*, pages 212–227, 2011. ISSN: 2375-1207. doi:10.1109/SP.2011.45.

Appendix A

Pseudocode for all Algorithms introduced in this Thesis

This appendix contains the pseudocode for all the algorithms introduced in this thesis.

For the globalBFS algorithm the following notation is used: v_s and v_t denote the start node and end nodes, respectively, S denotes the set of the user's strict requirements, B denotes the set of the user's best effort requirements, F denotes the set of features for each node, P_b denotes the globally best path, and BglobalBestScore denotes the *number* of best effort requirements that the globally best path satisfies. The current values are denoted by v_c, P_c, and B_c, where B_c refers to the *set* of best effort requirements currently supported by all nodes in the path.

Algorithm 2 localSearchHeuristic()

Input: G(V, E), pathRequest, limits
Output: path

```
1: if start or end node does not meet strict requirements then
2:     return ∅
3: ▷ See Algorithm 5 for the definition of bidirectionalBFSWithFilter()
4: path ← bidirectionalBFSWithFilter(G, pathRequest.from, pathRequest.to)
5: if |path|= 0 then
6:     ▷ No path could be found
7:     return ∅
8: ▷ If the user has not specified any best effort requirements, there is nothing to optimize, so
   we return the shortest path.
9: if |pathRequest.BER| = 0 then
10:    return path
11: originalPath ← path
12: ▷ Define number of edges that is skipped with the detour.
13: for detourDistance ∈ {2, . . . , |originalPath| − 1} do
14:    for i ∈ {0, . . . , |originalPath| − detourDistance − 1} do
15:        if originalPath[i] ∈ path ∧ originalPath[i + detourDistance] ∈ path then
16:            startIndex ← path.indexOf(originalPath[i])
17:            endIndex ← path.indexOf(originalPath[i + detourDistance])
18:            detourStart ← path[startIndex]
19:            detourEnd ← path[endIndex]
20:            bottleneck ← path[{startIndex+1, . . . , endIndex - 1}]
21:            ▷ Calculate the set of BER all nodes in the current path satisfy.
22:            currentPathBER ← pathRequest.BER
23:            for node ∈ path do
24:                currentPathBER ← currentPathBER ∩ node.features
25:            ▷ Calculate the set of BER that all nodes in the current path excluding the
               bottleneck support
26:            bottleneckFreeBER ← pathRequest.BER
27:            for node ∈ path \ bottleneck do
28:                bottleneckFreeBER ← bottleneckFreeBER ∩ node.features
29:            if |currentPathBER| = |bottleneckFreeBER| then
30:                ▷ There is no potential for improvement, since the current bottleneck turns
                   out to not be a bottleneck
31:                Continue
32:            ▷ See the definition of findDetours() in Algorithm 3.
33:            ▷ [ ] denotes an empty ordered list
34:            detours ← findDetours(G, detourStart, detourEnd, pathRequest, path, lim-
               its.neighbourLimit, limits.depthLimit, [ ], [ ], bottleneckFreeBER)
35:            bestBER ← currentPathBER
36:            bestDetour ← ∅
37:            updatePath ← False
38:            for detour ∈ detours do
39:                potentialBER ← bottleneckFreeBER
40:                for node ∈ detour do
41:                    potentialBER ← potentialBER ∩ node.features
42:                if |potentialBER| > |bestBER| then
43:                    bestBER ← potentialBER
44:                    bestDetour ← detour
45:                    updatePath ← True
46:            if updatePath then
47:                path ← path[{0, . . . , startIndex}] ∪ bestDetour ∪ path[{endIndex, . . . ,
                   |path| − 1}]
48: return path
```

Algorithm 3 findDetours()

Input: G(V, E), detourStart, detourEnd, pathRequest, path, neighbourLimit, depthLimit, prefix, postfix, bottleneckFreeBER
Output: detours

```
1: if depthLimit = 0 then
2:     return ∅
3: detours ← ∅
4: startNeighbours ← neighbours(detourStart) \ path \ prefix \ postfix
5: endNeighbours ← neighbours(detourEnd) \ path \ prefix \ postfix
6: ▷ See Algorithm 4 for the definition of limitNeighbours().
7: startNeighbours ← limitNeighbours(G, startNeighbours, neighbourLimit, bottleneckFree-
   BER)
8: endNeighbours ← limitNeighbours(G, endNeighbours, neighbourLimit, bottleneckFree-
   BER)
9: sharedNeighbours ← startNeighbours ∩ endNeighbours
10: for n ∈ sharedNeighbours do
11:     if n fulfills strict requirements then
12:         detours ← detours ∪ { [prefix ∪ n ∪ postfix] }
13: for s ∈ startNeighbours do
14:     for e ∈ endNeighbours do
15:         if s and e fulfill the strict requirements then
16:             if s ≠ e ∧ |{s, e} ∩ (path ∪ prefix ∪ postfix)| = 0 then
17:                 ▷ Add the case where the two neighbours are connected with a link. These
                    two connected neighbours also form a detour.
18:                 if (s, e) ∈ E then
19:                     detours ← detours ∪ { [prefix ∪ s ∪ e ∪ postfix] }
20:                 prefix ← [prefix ∪ s]
21:                 postfix ← [e ∪ postfix]
22:                 detours ← detours ∪ findDetours(G, s, e, pathRequest, path, neighbourLimit,
                    depthLimit − 1, prefix, postfix)
23: return detours
```

Algorithm 4 limitNeighbours()

Input: G(V, E), neigbours, neighbourLimit, bottleneckFreeBER
Output: neighbours

```
1: sortedNeighbours ← list of neighbours sorted on descending score, where score is the size of
   the intersection of the neighbours' features and the set of best effort requirements supported
   by the path thus far
2: if |sortedNeighbours| ≤ neighbourLimit then
3:     return sortedNeighbours
4: else
5:     ▷ Select the [neighbourLimit] neighbours with the highest score
6:     sortedNeighbours ← sortedNeighbours[{0, . . . , neighbourLimit − 1}]
7:     return sortedNeighbours
```

Algorithm 5 bidirectionalBFSWithFilter()

Input: G(V, E), pathRequest
Output: path

```
1: ▷ See Algorithm 6 for the definition of findPredAndSucc()
2: pred, succ, currentNode = findPredAndSucc(G, pathRequest)
3: if pred = ∅ ∧ succ = ∅ ∧ currentNode = ∅ then
4:     return ∅
5: path ← [ ]
6: while currentNode ≠ ∅ do
7:     path ← path ∪ {currentNode}
8:     currentNode ← pred[currentNode]
9: path ← path.reverse()
10: ▷ path[−1] denotes the last element of the path
11: currentNode ← succ[path[−1]]
12: while currentNode ≠ ∅ do:
13:     path ← path ∪ {currentNode}
14:     currentNode ← succ[currentNode]
15: return path
```

Algorithm 6 findPredAndSucc()

Input: G(V, E), pathRequest
Output: pred, succ, meetupNode

```
source ← pathRequest.as_source
target ← pathRequest.as_destination
if target == source then
    return ({target: ∅}, {source: ∅}, source)
▷ The pred and succ data structures are dictionaries that map nodes to their predecessor or successor
pred ← {source: ∅}
succ ← {target: ∅}
forwardVisited ← ∅ ∪ { source }
reverseVisited ← ∅ ∪ { target }
▷ While loop cancels when both the forward and the reverse search have no unexplored nodes left. If no explored nodes are left, there is no path.
while |forwardVisited|> 0 and |reverseVistited|> 0 do
    if |forwardVisited|≤| reverseVisited| then
        thisLevel ← forwardVisited
        forwardVisited ← [ ]
        for currentNode ∈ thisLevel do
            if currentNode fulfills strict requirements then
                for neighbour ∈ neighbours[currentNode] do
                    if neighbour fulfills strict requirements then
                        if neighbour ∉ pred then
                            forwardVisited ← forwardVisited ∪ { neighbour }
                            pred[neighbour] ← currentNode
                        if neighbour ∈ succ then
                            ▷ We found a path, return
                            return pred, succ, neighbour
    else
        thisLevel ← reverseVisited
        reverseVisited ← [ ]
        for currentNode ∈ thisLevel do
            if currentNode fulfills strict requirements then
                for neighbour ∈ neighbours[currentNode] do
                    if neighbour fulfills strict requirements then
                        if neighbour ∉ succ then
                            reverseVisited ← reverseVisited ∪ { neighbour }
                            succ[neighbour] ← currentNode
                        if neighbour ∈ pred then
                            ▷ Again: We found a path, return
                            return pred, succ, neighbour
▷ If we reach this line, no path exists in the graph
return ∅, ∅, ∅
```

Algorithm 7 globalBFS()

Input: G(V, E), v_s, v_t, S, B, F
Output: P_b, BglobalBestScore

if start or end node does not satisfy strict requirements **then**
    return $\varnothing, -1$
BglobalBestScore $\leftarrow -1$
$P_b \leftarrow [\,]$ ▷ *[ ] denotes an empty list*
▷ Store intermediate results in FIFO queue
$Q \leftarrow$ FIFO Queue
Q$\leftarrow Q \cup \{v_s, [\,], \text{B}\}$)
while Q is not empty **do**
    $\{v_c, P_p, B_p\} \leftarrow$ Q.dequeue()
    ▷ Update path variables
    $P_c \leftarrow P_p \cup \{v_c\}$
    $B_c \leftarrow B_p \cap F(v_c)$
    if $\mid B_c \mid \leq$ BglobalBestScore **then**
        ▷ We cannot become better than the current best score, so we stop the search
        continue
    ▷ If we arrived, store result and exit
    if $v_c == v_t$ **then**
        if $\mid Bc \mid >$ BglobalBestScore **then**
            BglobalBestSCore $\leftarrow \mid B_c \mid$
            $P_b \leftarrow P_c$
        ▷ Stop exploring after finding the end node
        continue
    for $v_i \in$ neighbours(v_c **do**
        if $v_i \notin P_c \wedge v_i$ satisfies the strict requirements **then**
            Q $\leftarrow$ Q $\cup\{v_i, P_c, B_c\}$
return P_b, BglobalBestSCore

Appendix B

Example Table of Requirements

Table B.1 enumerates a number of security requirements, taken from [25] and [50]. This table is not complete. It serves as a showcase of the format that can be used to enumerate the requirements such that they may be referred to with only an integer to identify them. Further security requirements can be seen in [46], and privacy requirements can be seen in [18] and [22]. Future work can focus on incorporating these requirements into a publicly accessible database with a robust identification system that allows referencing these requirements via an integer, string, or other form of identification.

ID	Name	Source
1	ISPs should prevent propagation of incorrect routing information by checking the validity of their customers' BGP announcements.	MANRS
2	ISPs should filter outbound traffic with spoofed source IP and run the CAIDA Spoofer software [55] to prevent DDoS attack traffic from being originated from the participant's network.	MANRS
3	ISPs should maintain up-to-date network contact information in IRR databases or PeeringDB	MANRS
4	Register intended BGP announcements in IRR or (preferrably) RPKI.	MANRS
5	ISPs have a duty to make sure that their contact information, in Whois, in routing registries [RFC1786] or in any other repository, is complete, accurate and reachable.	RFC3013
6	ISPs SHOULD adhere to [RFC2142], which defines the mailbox SECURITY for network security issues, ABUSE for issues relating to inappropriate public behaviour and NOC for issues relating to network infrastructure.	RFC3013
7	ISPs SHOULD have clear policies and procedures on the sharing of information about a security incident with their customers, with other ISPs, with Incident Response Teams, with law enforcement or with the press and general public.	RFC3013
8	ISPs should have processes in place to deal with security incidents that traverse the boundaries between them and other ISPs.	RFC3013
9	ISPs SHOULD be proactive in notifying customers of security vulnerabilities in the services they provide.	RFC3013
10	As new vulnerabilities in systems and software are discovered, ISPs should indicate whether their services are threatened by these risks.	RFC3013
11	When security incidents occur that affect components of an ISP's infrastructure the ISP should promptly report to their customers who is coordinating response to the incident, the vulnerability, how service was affected, what is being done to respond to the incident, whether customer data may have been compromised, what is being done to eliminate the vulnerability, the expected schedule for response, assuming it can be predicted	RFC3013
12	ISPs should have a well-advertised way to receive and handle reported incidents from their customers.	RFC3013
13	Every ISP SHOULD have an Appropriate Use Policy (AUP). An AUP should clearly identify what customers shall and shall not do on the various components of a system or network, including the type of traffic allowed on the networks. The AUP should be as explicit as possible to avoid ambiguity or misunderstanding.	RFC3013
14	In addition to communicating their AUP to their customers ISPs should publish their policy in a public place such as their web site so that the community can be aware of what the ISP considers appropriate and can know what action to expect in the event of inappropriate behaviour.	RFC3013
15	An AUP should be clear in stating what sanctions will be enforced in the event of inappropriate behaviour.	RFC3013

ID	Name	Source
16	ISPs should publicly register the address space that they assign to their customers so that there is more specific contact information for the delegated space.	RFC3013
17	ISPs should ensure that the registry information that they maintain can only be updated using strong authentication	RFC3013
18	BGP authentication [RFC2385] SHOULD be used with routing peers.	RFC3013
19	At the boundary router with each of their customers ISPs should proactively filter all traffic coming from the customer that has a source address of something other than the addresses that have been assigned to that customer.	RFC3013
20	At the boundary router with each of their customers ISPs should proactively filter all traffic going to the customer that has a source address of any of the addresses that have been assigned to that customer.	RFC3013
21	ISPs should filter the routing announcements they hear, for example to ignore routes to addresses allocated for private Internets, to avoid bogus routes and to implement "BGP Route Flap Dampening" [RFC2439] and aggregation policy.	RFC3013
22	Mail, news, webhosting, and other services, should be kept on separate systems to facilitate better security	RFC3013
23	ISPs should prevent their mail infrastructure from being used by 'spammers' to inject Unsolicited Bulk E-mail (UBE) while hiding the sender's identity [RFC2505]. While not all preventive steps are appropriate for every site, the most effective site-appropriate methods should be used.	RFC3013
24	ISPs should prevent their mail infrastructure from being used by 'spammers' to inject Unsolicited Bulk E-mail (UBE) while hiding the sender's identity [RFC2505]. While not all preventive steps are appropriate for every site, the most effective site-appropriate methods should be used.	RFC3013
25	Message submissions should be authenticated using the AUTH SMTP service extension as described in the "SMTP Service Extension for Authentication" [RFC2554].	RFC3013

Appendix C

Variable Score Range Experiment Filter Stage Figures

This appendix contains the limit stage runtime boxplots that show the range of runtimes and their average relative to the example runtime limit of 500 ms, along with the average score improvement for each set of limits. The best set of limits is the set of limits that maximizes the average score improvement while having a runtime strictly below 500 ms.

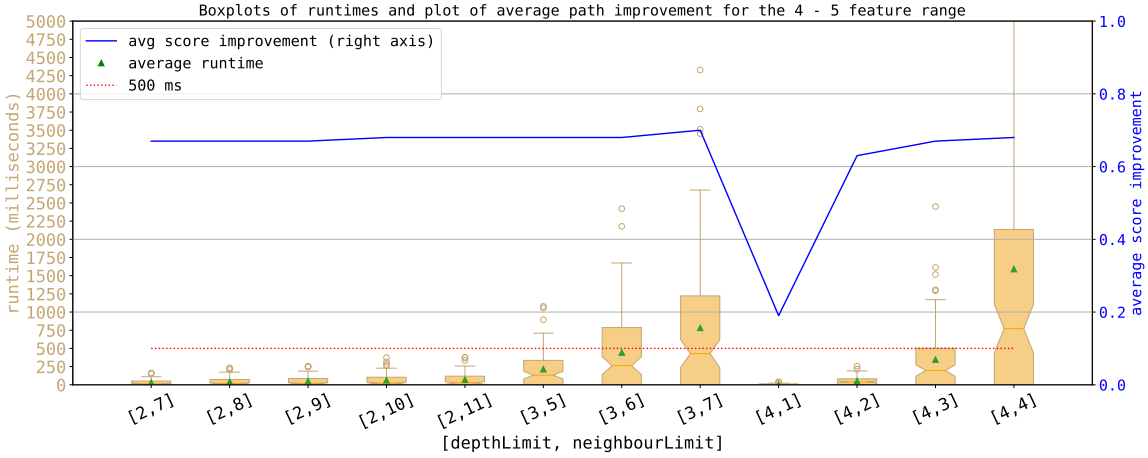


Figure C.1: *Plot with runtime boxplots and average score improvement data. The rightmost boxplot is cut off to keep the other boxplots readable, but the boxplot is included to show that [4,4] is not a viable option given our selection criterion. Note that [3,6] is the best set of limits.*

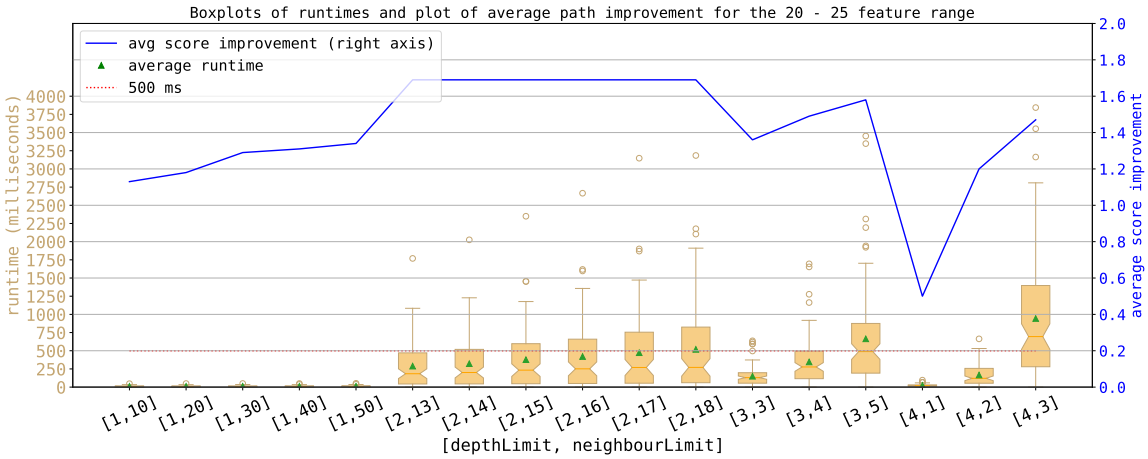


Figure C.2: *Plot with runtime boxplots and average score improvement data. Note that [2,13] until [2,17] are equally good sets of limits. We chose [2,15] here since it is in the middle, but any of these other limit sets would work as well.*

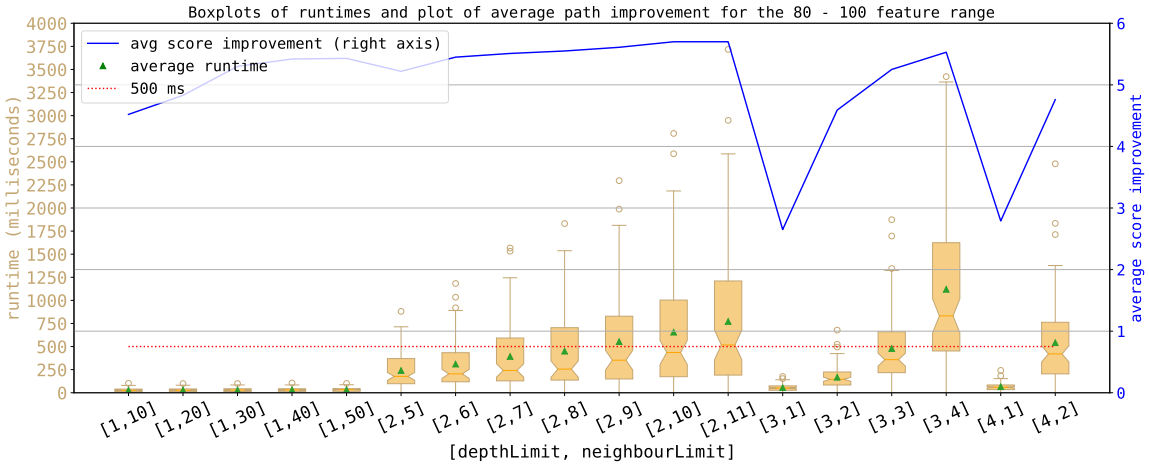


Figure C.3: *Plot with runtime boxplots and average score improvement data. Although it is hard to see, the average score improvement of [2, 8] is 5.55, which is slightly higher than the average score improvement of [3, 3] which is 5.25. Thus, [2, 8] is chosen here.*

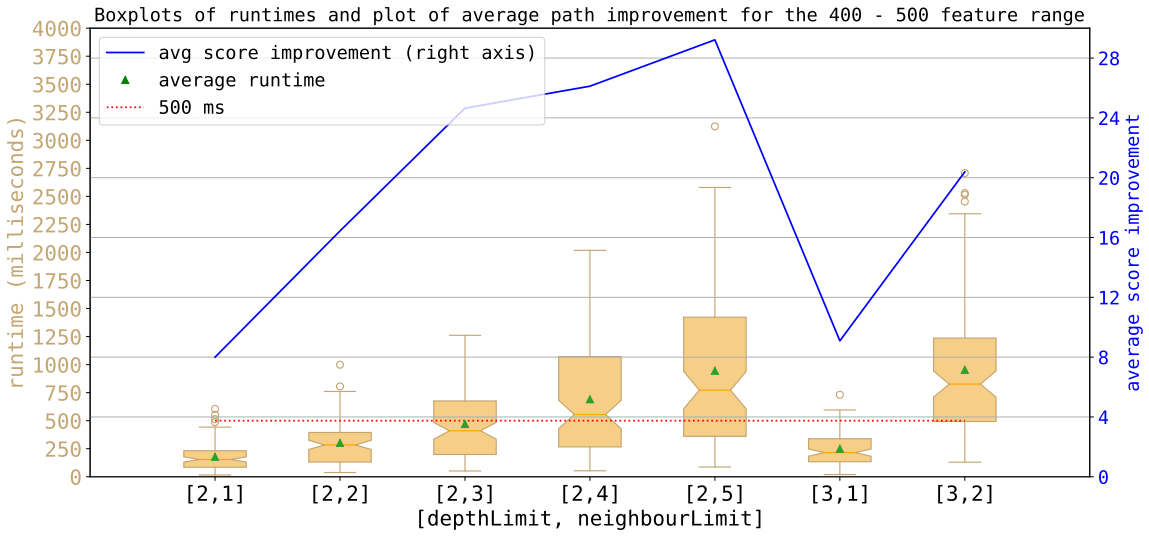


Figure C.4: *Plot with runtime boxplots and average score improvement data. Note that [2,3] is the best set of limits here.*

Appendix D

Ratio Experiment Filter Stage Figures

This appendix contains the tables with the runtime and average score improvement data that was used to evaluate the impact of different ratios between minimum number of features and maximum number of features. As the figures show, the set of limits for which the average path improvement is maximal while having an average runtime below 500 ms is very similar for each ratio.

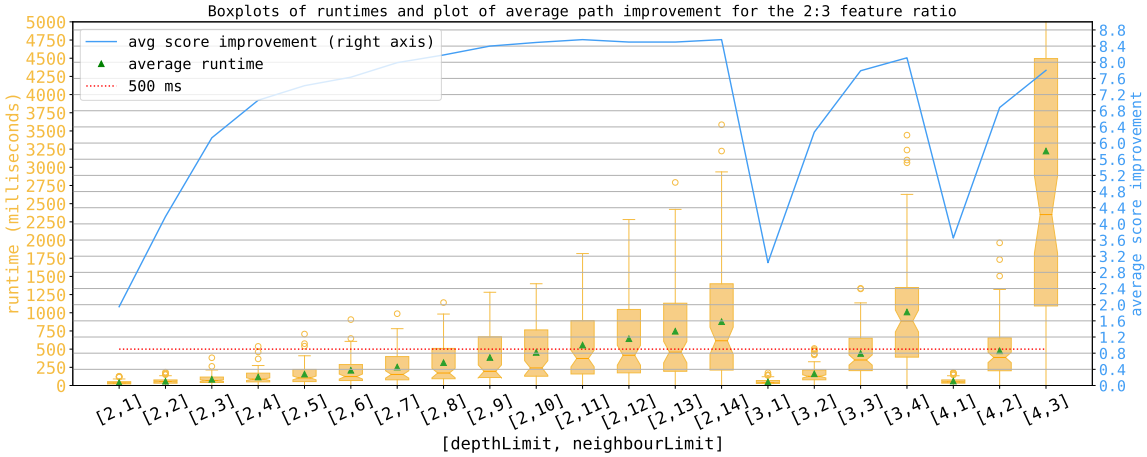


Figure D.1: *Plot with runtime boxplots and average score improvement data. Note that [2,10] is the best set of limits.*

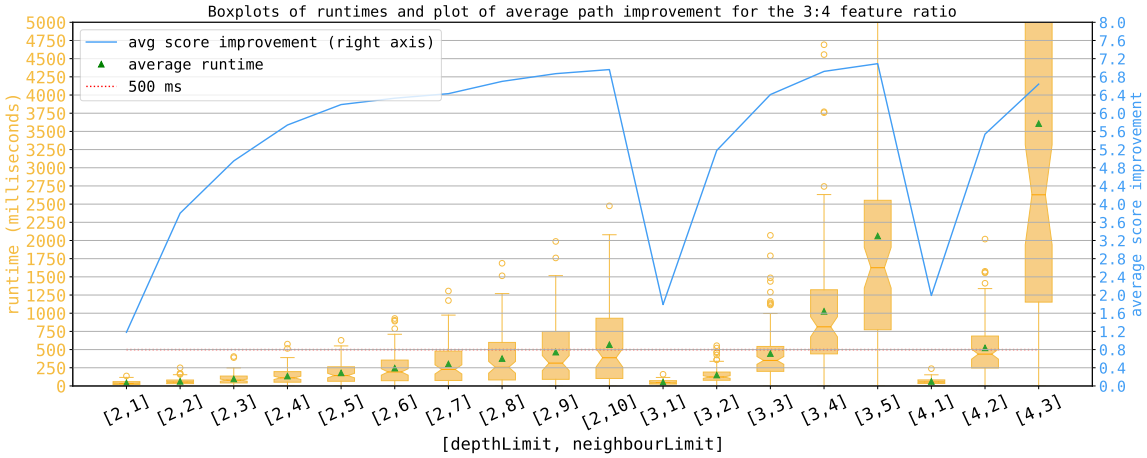


Figure D.2: *Plot with runtime boxplots and average score improvement data. Note that [2,9] is the best set of limits.*

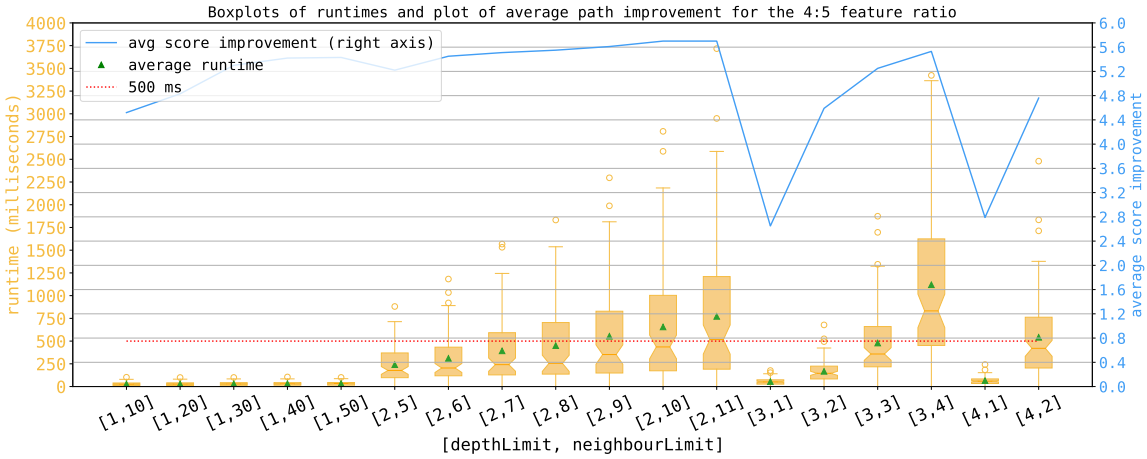


Figure D.3: *Plot with runtime boxplots and average score improvement data. Note that [2,8] is the best set of limits.*

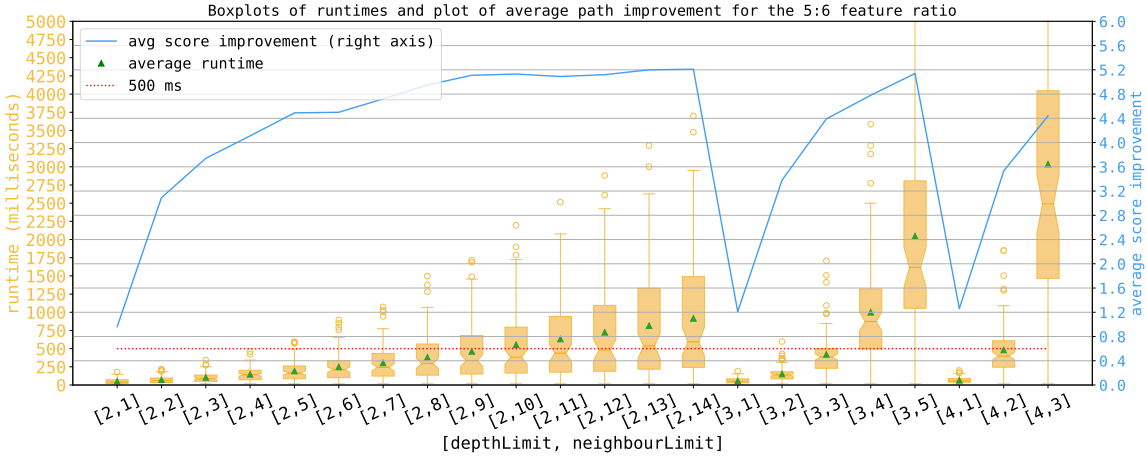


Figure D.4: *Plot with runtime boxplots and average score improvement data. Note that [2,9] is the best set of limits.*

Appendix E

Infrastructure Experiment Filter Stage Figures

This appendix contains the limit stage runtime boxplots that show the range of runtimes and their average relative to the example runtime limit of 500 ms, along with the average score improvement for each set of limits. The best set of limits is the set of limits that maximizes the average score improvement while having a runtime strictly below 500 ms.

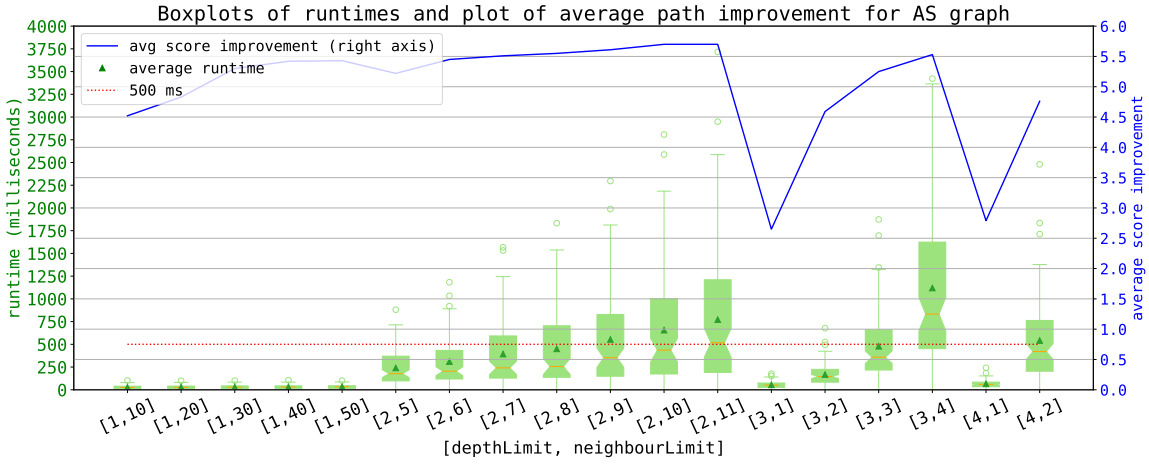


Figure E.1: *Plot with runtime boxplots and average score improvement data. Note that the average runtime of [3,3] was exactly 500 ms, while the criteria was that the average runtime was below 500 ms. That is why [2,8] was chosen instead.*

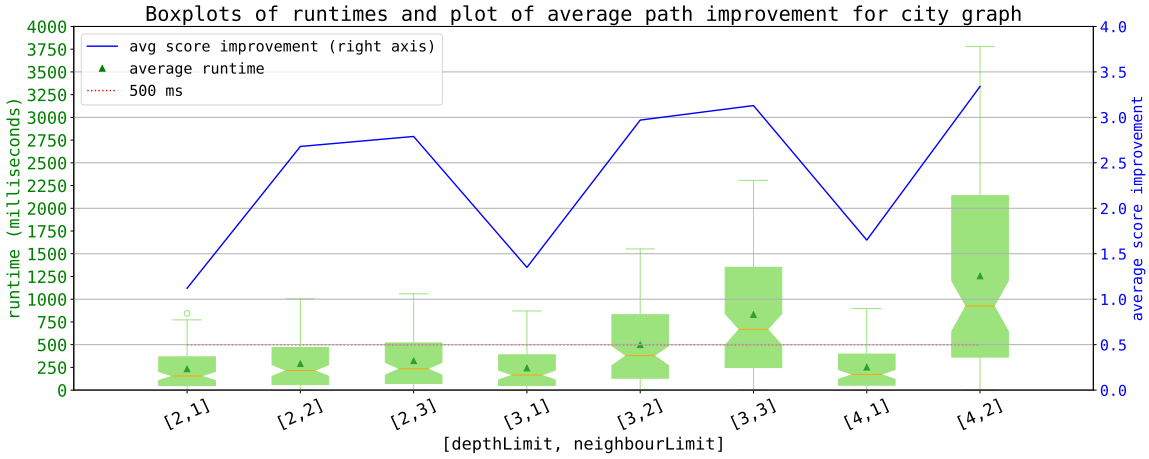


Figure E.2: *Plot with runtime boxplots and average score improvement data. Here the average runtime of [3,2] is just above the 500 ms runtime threshold, which is why [2,3] is chosen as the best set of limits.*

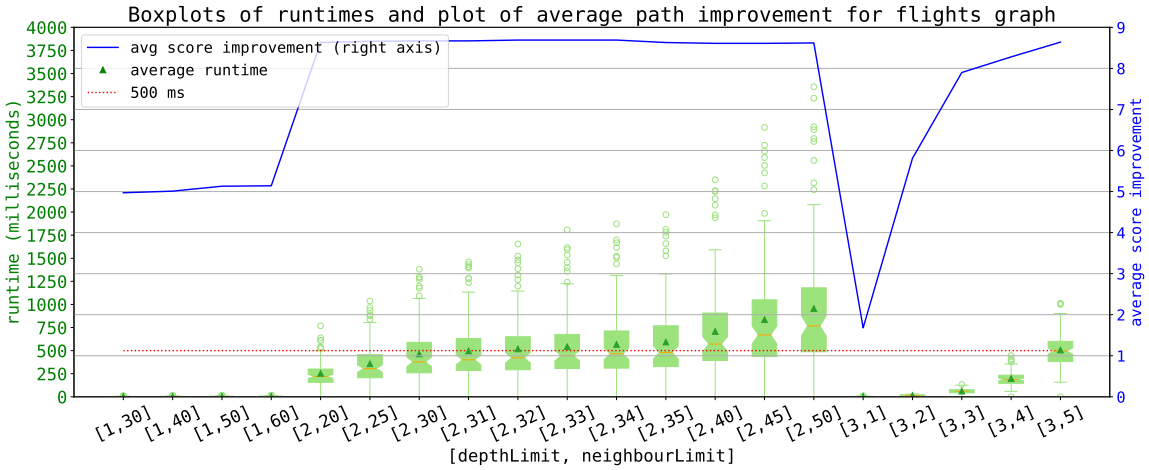


Figure E.3: *Plot with runtime boxplots and average score improvement data. Note that [2,30] is the best set of limits.*

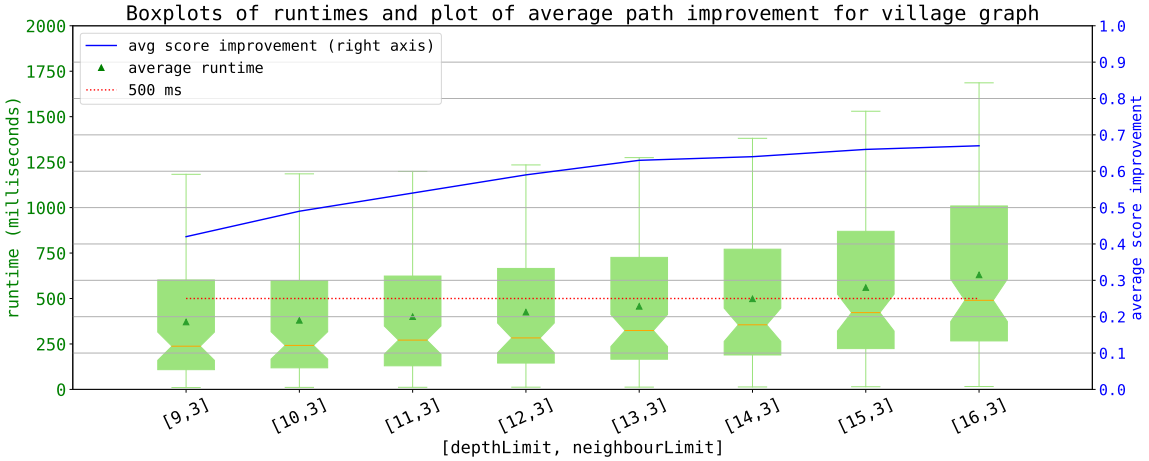


Figure E.4: *Plot with runtime boxplots and average score improvement data. The neighbourlimit is set to 3 since that is the most neighbours (that are not in the path already) that a node can consider in the village graph, since the maximum degree is 4. Instead, the real improvement here comes from the depthLimit, which, in consequence, is significantly higher than on the other maps. In the end [14,3] is chosen as the best set of limits.*