Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Distributed route discovery in communication networks using neighbour information

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

Pieter Kleer

Delft, the Netherlands
June 2015

MSc thesis APPLIED MATHEMATICS

Distributed route discovery in communication networks using neighbour information

Pieter Kleer

Delft University of Technology

Daily supervisors

Dr. D. C. Gijswijt
Ir. B. M. M. Gijsen

Responsible professor

Prof. dr. ir. K. I. Aardal

Other committee members

Dr. R. J. Fokkink

June, 2015

Delft

Preface

This thesis has been written as graduation work for the master Applied Mathematics at the TU Delft, and has been carried out at the company TNO (Nederlandse Organisatie voor Toegepast-Natuurwetenschappelijk Onderzoek). From September 2014 until June 2015, I have studied routing protocols within communication networks. Routing protocols are used to specify how traffic should be forwarded through the network. For example, in source routing, the complete route between source and destination is stored in the data that is to be transferred, so that an intermediate node knows to which neighbour to forward the data. In this thesis, we will describe a distributed algorithm to discover routes in a communication network, that can be used for source routing. We will give a mathematical analysis of this algorithm in order to see how we can make optimal use of the principles described in it.

Structure of the thesis. This thesis consists of three parts. The first part provides the context in which this work is performed: routing in communication networks. Chapter 3 in this part gives an introduction to the problem that we are analysing, and might also be used as a starting point for the reader that is mostly interested in mathematics. The first two chapters serve as context for routing principles in communication networks. The second part of this thesis contains my primary contribution in the form of a thorough analysis of a distributed algorithm for discovering multiple disjoint paths in a (communication) network. This distributed algorithm is (partially) described in a patent of the company TNO, that was the reason for this work. The third part consists of a short high-level discussion on the applicability of the patent and will summarize some of the important results obtained in the second part. The first and third part are written in an informal fashion, whereas the second part is strongly mathematical.

Acknowledgements. I would like to thank all of my supervisors that have helped me this last nine months to create this thesis. First of all, I would like to thank Bart Gijsen (TNO) for his daily supervision at TNO. I have learned a lot from our many meetings and enjoyed having you as my supervisor. Secondly, I want to thank Karen Aardal and Dion Gijswijt (TU Delft) for their supervision at the university. I would especially like to thank Dion for taking the time to go through a lot of the mathematical descriptions in the second part of this thesis, and for the many suggestions to improve them. Lastly, I would like to thank the department of Performance of Networks and Systems (PoNS) at TNO for giving me the opportunity to be part of their team and for having been great colleagues the last nine months.

Pieter Kleer, June 2015

Contents

Part I

Introduction

Chapter 1

Communication networks

A *communication network* is a collection of hardware components connected by network links (data channels) that can be used to transmit data between the different components. In this context, data is a sequence of bits, which can be converted to a physical signal that is sent over a network link, for example a wireless connection, a glass fiber or USB cable. We say that the network link connecting component A and B is *bidirectional* if it is possible to send data in both directions of the link, which is most often not the case in communication networks. We then call the link *directed*. Communication networks can vary from small networks, such as a Local Area Network (LAN) connecting the computers and printers within a household, to worldwide networks such as the internet. The first data communication networks emerged from military applications. An example is the SAGE network, that was used to collect data from different radar sites in order to create a complete image of the airspace over a wide area.

In order to transmit data between two components of a network, we need to find a *route* through the network. In this context, a route is a sequence of components such that there exists a network link between two consecutive components in the sequence, over which data can be transmitted from a component to its successor in the route. *Routing* is the process of selecting routes between different components.

A *protocol* is a set of rules that describe data exchange between, or within, components. A *routing protocol* is a protocol concerned with routing traffic through the network and consists mainly of two parts: route discovery and route maintenance. In the route discovery phase, the *routing table*, containing information about the forwarding rules, is created. Route maintenance is concerned with the updating of the routing table in case of a topology change (e.g., if components are added/removed or if a network link fails).

A communication network often consists of different hardware components that need to be able to send each other data. To that end, there are *open systems*, which are sets of protocols allowing different hardware components to communicate (or in general, different systems). For example, it must be possible to get a phone call from someone with a phone from a different manufacturer. For this reason, certain *standards* have been developed in order to allow communication between different hardware components. These standards can either be hardware-related (e.g., standardized USB-ports) as well as software-related (e.g., IP-protocols describing the communication on the internet).

There exists a theoretical framework, the *Open Systems Interconnection model (OSI Model)* [1], that characterizes and standardizes the functions of communication networks, using a seven layer model. The goal of this model is showing how to establish communication between different systems without having to take into account the underlying hard- or software components of these systems. The model itself is not a protocol, but is used for understanding and designing communication networks.

The idea behind the seven layers of the OSI model is the division of different networking functions. The lower layers (Physical, Data Link, Network and Transport) are mostly concerned with moving data around the network, but do not care about the actual data. The three highest

levels (Session, Presentation and Application) are concerned with the interaction between users and applications, regardless of how the data is sent over the network. Going from bottom to top, one becomes less concerned with hardware functions and more with software (application) functions.

In the following sections, we will describe some communication networks, and the process of routing within them. We will also place the relevant components in the OSI-model, in order to get an idea of how this framework is designed.

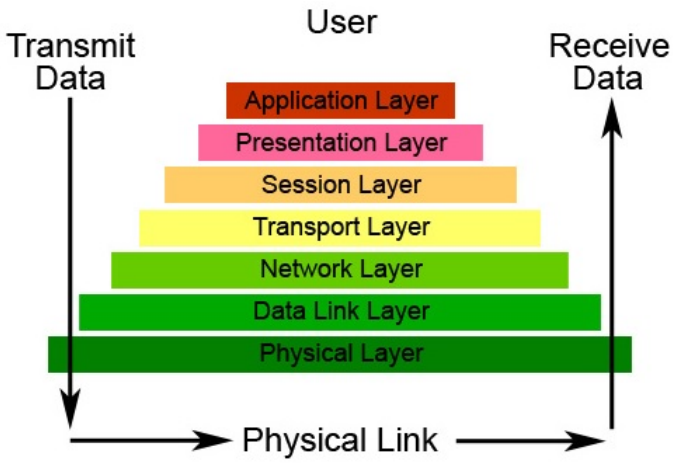

Figure 1.1: Schematic overview of the seven layers of the OSI Model.

1.1 Local Area Network

We will illustrate the concept of *routing* using the Ethernet network, a computer network first introduced in the 80's, that is still used nowadays. The Ethernet network is an example of a LAN. The hardware components of the Ethernet can be divided into two groups: data terminal equipment (DTE), e.g., personal computers and printers, and data communication equipment (DCE), e.g., routers and switches.

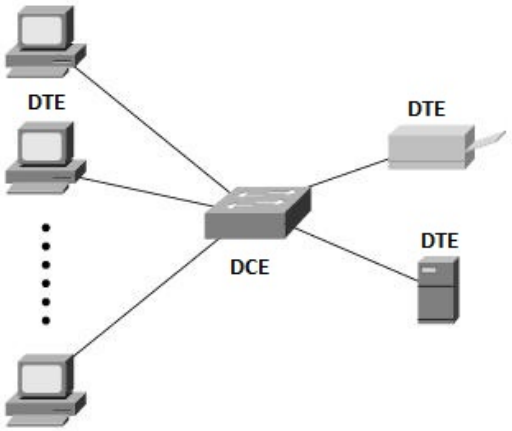

Figure 1.2: Schematic example of an Ethernet network.

Every hardware component within the LAN has a unique identifier, called the MAC-address (Media Access Control address). This address is assigned by the manufacturer of the component and is also called the physical address of the hardware component.

The network links are Ethernet cables, see Figure 1.3. These links are bidirectional so that data can be sent in both directions of the cable simultaneously (also known as full-duplex transmission). The first Ethernet cables did not have this property. A terminal component is connected to one communication component, whereas a communication component can be connected to multiple terminal components. Functionalities and properties of cables (or other network links) are described at the Physical layer of the OSI Model.

Within the Ethernet network, data is transmitted over network links using Ethernet *frames*, that are standardized by the Institute of Electrical and Electronics Engineers (IEEE). These frames contain, among other things, MAC-addresses of both the source and destination, a Frame Check Sequence (FCS), and the data that needs to be transmitted. The FCS is used to verify whether or not the frame was received correctly by the destination address.

The functionalities of an Ethernet network are described in the Data Link layer of the OSI Model. For example, there is a standardization of which bits in an Ethernet frame correspond to what information. The Physical layer is only concerned with the actual sending and receiving of physical signals, whereas the Data Link layer is, for example, concerned with checking whether or not the frame is correctly received, e.g., using the FCS.

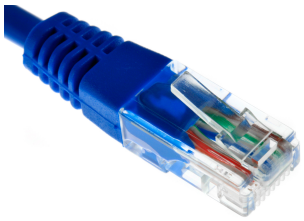

Figure 1.3: End part of Ethernet cable.

We can also describe an Ethernet network using an undirected graph $G = (V, E)$, where V is the set of MAC-addresses and E the set of pairs of MAC-addresses that share an Ethernet cable, see Figure 1.4. One important requirement for the Ethernet topology is that it cannot contain cycles, i.e., the graph G must be a connected tree. This means that there exists a unique route between all pairs of components.

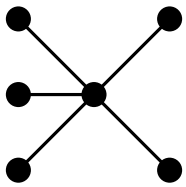

Figure 1.4: Sketch of graph G corresponding to network in Figure 1.2.

Routing is performed by data communication equipment, for example a network switch, see Figure 1.5. A switch has multiple ports in which Ethernet cables can be plugged in. These cables can either lead to terminal components or other communication equipment (e.g., other switches).

A switch has a *forwarding table*, that (ideally) assigns the MAC-address of every component to one specific port. This is the port that contains the network link lying on the unique route from the switch to the component. Note that multiple MAC-addresses can be assigned to the same port. If the switch receives a frame, it can look up the destination address in the forwarding table and forward the frame through the assigned port. These forwarding tables can either be manually configured or can be 'learned' by the switch itself. That is, whenever a switch receives a frame of which the source MAC-address is not yet in the forwarding table, it assigns the MAC-address of the source to the port through which the frame arrived. If the destination address is not in the table, it forwards the frame to all other ports containing an Ethernet cable. This shows the importance of the requirement that the Ethernet topology does not contain cycles, otherwise the same frame might arrive at the switch over and over again, until the destination address is in the forwarding table.

Figure 1.5: A network switch.

For large Ethernet networks, it might not be clear if the topology contains cycles, since there might not be a central 'view' of the network. To solve this problem, there exist protocols that establish a spanning tree within the Ethernet network, by blocking certain network links that might cause cycles. This also shows the concept of having different *layers* within a network. On the Physical layer, the network contains cycles, but these are not 'observed' on the Data Link layer.

It is also possible to divide end nodes, that are connected to different switches, in multiple groups (instead of physically dividing the network into multiple components). In this way, one physical LAN can be partitioned in multiple Virtual Local Area Networks (VLAN). This partitioning can be done by explicitly assigning certain ports to certain VLANs. Or, it can be done by *tagging* Ethernet frames with a VLAN-tag that contains the corresponding VLAN it is destined for. In this way, multiple VLANs can use the same physical cable.

1.2 IP network

An IP Network is a communication network in which the Internet Protocol (IP) is used by devices to communicate. The largest example of an IP network is The Internet: a worldwide system of interconnected computer networks. Just as in Ethernet networks, every component has a unique identifier: the IP address. In the fourth version of the IP protocol (IPv4), this is an address consisting of 32 bits, represented by four hexadecimal numbers each consisting of 8 bits, for example

192.168.1.256

which means there are $2^{32} \approx 4 \times 10^9$ unique addresses. One of the major developments at the moment is the transition to a new protocol, IPv6, where every IP address consists of 128 bits (the current protocol is referred to as IPv4 in the OSI Model). This new protocol will increase the number of unique addresses significantly. The Internet Assigned Numbers Authority (IANA) is responsible for the global coordination of IP-addresses. The addresses are distributed in a hierarchical way: fixed ranges of addresses are reserved for particular regions in the world (Africa, Asia/Pasific, North America, Latin America and Europe/Middle East/Central Asia). Furthermore, within these regions, ranges are reserved for large companies, governments and Internet Service Providers (ISP). Whenever an institution requires extra IP-addresses, IANA has to allocate IP-addresses not yet assigned in that region.

The functionalities of an IP network are described at the Network layer of the OSI model. However, there also exists a four layer model, called the TCP/IP Model (e.g., [2]), that serves as a description of the functionalities of IP networks. The Internet Protocol is mostly concerned with routing data through the network, whereas the Transfer Control Protocol (TCP) is concerned with the reliability of the data communication process.

When a computer (or router) is connected to an IP network, it needs to get assigned an IP address. This is done using the Dynamic Host Configuration Protocol (DHCP). For example, if you have an internet subscription, the Internet Service Provider (ISP) needs to assign an IP address to your computer. DHCP is used to automatically assign one of the free IP addresses that the ISP has in store (although hardware components such as e-mail servers often have static IPv4 and IPv6 addresses). Furthermore, the Domain Name System (DNS) is used to map web-addresses to actual IP addresses. This is done for the simple reason that it is easier to remember human readable web-addresses, rather than their IP address.

The routing in the internet is done by routers, as opposed to switches in Ethernet networks, and is described in the Network layer of the OSI Model. Routers can work with multiple types of network links such as optic fibre cables, Ethernet cables or wireless connections. Data is being sent using standard IP packets that contain, among other things, the source and destination IP address, the version of the IP protocol used (e.g., IPv4, IPv6), and a Time To Live (TTL). The Time To Live serves as a hop-count measure of how many routers may forward the packet. Whenever a router forwards a packet, the TTL is decremented by one. When a router receives a packet of which the TTL is zero, the packet is not forwarded, but dropped. Just as switches, routers have forwarding tables, that contain information regarding the forwarding process of incoming packets. These tables can be configured manually or the router can 'learn' these routes for certain (ranges of) IP addresses.

There are two major classes of routing protocols: *distance-vector* and *link-state* routing protocols. In a distance-vector routing protocol, every router stores the next router (hop) on a shortest path to the destination. The hops are determined using, for example, a distributed version of the Bellman-Ford algorithm, which will be explained in Section 2.3. A shortest path, in this context, is a path with an optimal *routing metric*, for example the number of hops on the path, or the time to get to the destination. In a link-state routing protocol, every router within the network has a complete overview of the network, i.e., it knows which routers are connected, and can calculate a shortest-path tree, containing a shortest path to every destination in the network. In order to build this complete view of the network, routers send Link-State Advertisements (LSA), containing their own identifier, the identifiers of their neighbours and a sequence number. In order to determine who their neighbours are, a *reachability protocol* can be used. When a router receives an LSA from another router, it checks the sequence number to see if it already received this advertisement (whenever a router identifies a change of neighbours, it sends out an LSA with a new sequence number). If it has not yet received the advertisement, it stores the connectivity information and floods the message to all of his neighbours. In this way, every router can construct a complete view of the network. If not all routers construct the same tree, loops can occur. For example, suppose we have a router A, that has router B as the next hop, in order to reach router C. Now, if router B has router A as next hop to reach router C, a packet will be send between A and B without ever getting to C. Nowadays, link-state protocols have mechanisms to prevent loops.

One of the issues with link-state routing is the stream of link-state advertisement packets, indicating an update in the network, for example an additional router or link failure. Suppose that a router indicates a link failure (which is observed by a reachability protocol), e.g., a broken cable. The router then sends a new LSA over the network, which will be flooded by every router, resulting in $\mathcal{O}(N^2)$ packets that have to be sent (in a network of N routers). This means that the overhead of the network increases significantly when the network grows, i.e., the protocol *scales* badly (solutions have been proposed to reduce the amount of LSA packets, e.g., [3]).

Note that the routing principles within distance-vector and link-state routing are not inherent to Internet networks, but can also be used in other communication networks. The protocols describing these rules are necessary for the implementation of these principles within IP networks.

Whether link-state or distance-vector routing is used, depends on the application. An advantage of link-state routing is that it is possible to directly calculate new routes, since the complete network topology is available at the router. However, distance-vector routing has the advantage of having to store less information in the router, since paths are stored implicitly at routers (in terms of next hops, instead of complete paths).

The routers that form the internet are connected in a hierarchical manner (which can also be seen in the IP-addresses). ISPs have their own (physical) networks, wherein it is possible to perform routing actions between all attached components. When components from another ISP have to be reached, so-called Network Access Points (NAPs) are used. These serve as points where traffic between distinct ISPs can be exchanged (for example within the same country). On top of that, networks are connected via The Internet Backbone, which is a collection of routes hosted by, governments, commercial companies and universities (among others). Here one might think of cable connections between different countries.

This hierarchy is also used when establishing forwarding tables. For example, routers within an *autonomous system* (such as the network of an ISP) will only contain routing information concerning routers within that same system. On the boundary different forwarding tables are used,

in order to connect to other systems. On this level, routing is done using IP-ranges, rather than specific IP addresses.

1.3 Wireless network

A *wireless* network is a communication network using wireless network links. The physical workings of these links are described in the Physical layer of the OSI Model. Communication can, for example, be established using *broadcasts*, where a radio signal is sent out that can be received by all components within a certain radius around a given component. The radius, called the *transmission range*, depends on the signal qualifications. Furthermore, it might be possible for components within the transmission range to not receive a broadcast, if there are physical obstacles such as buildings or walls. Furthermore, it is also possible for components to have directional antennae, that allow to broadcast only into a specific geographical area. This is, for example, possible in wireless cellular base stations.

An example of such a network is a Wireless Local Area Network (WLAN). A WLAN consists of *access points* and *clients*. Access points are able to receive and transmit both wireless and wired signals. The region in which an access point is accessible using a wireless connection is called a *hotspot*. Access points can be connected to, for example, a WLAN switch that is part of a LAN. Examples of clients are laptops, tablets and smart phones. The functionalities of WLANs are specified in the *IEEE 802.11 standard*, and are often referred to as *Wi-Fi* networks. There also exists hardware combining access points and WLAN switches.

If an access point is connected to a router, which is in turn connected to the internet, then it can also serve as a connection between a client and the internet. The quality of this internet connection mostly depends on the router and not the quality of the signal between the client and access point. For example, the bars on a smart phone screen do not indicate the quality of the connection to the internet, but to the access point.

When a client connects to the internet via an access point, it gets assigned an IP address by the router. This means that the client has a different identifier, depending on its place in the network. When a client connects to the WLAN, it will use its MAC-address as an identifier, independent of the router it is connected to.

In order to retrieve an IP address from an access point, a client communicates with the access point using its MAC address. Revelations of whistle blowers, such as Edward Snowden, suggest that spy agencies use this communication to track MAC addresses. In a reaction to this violation of privacy, Apple has developed a method that implements a random MAC address on its system (which makes it harder to determine the device corresponding to the MAC address).

Another example of a wireless network is a Mobile Ad Hoc Network (MANET). A MANET is a wireless network consisting of *mobile* nodes, that are free to move independently in every direction that is possible. The 'ad hoc' part refers to the fact that there is no central administration or predetermined infrastructure (which is for example the case in LANs or IP networks, in the form of switches and routers at fixed locations). An example is a platoon of soldiers and military vehicles distributed over a geographical area.

Just as in static networks it is possible to maintain a neighbour list, i.e., a list of nodes within the transmission range (which is done by periodically sending out so-called *hello* messages). If mobile nodes do this, we call them *stateful*, and otherwise *stateless* (these terms can also be used in static networks).

In MANETs, every node can serve as both an end node and router (as opposed to computer networks, where terminal equipment does not perform routing actions). This can become an issue, since mobile nodes often have a limited power supply and bandwidth. Furthermore, MANETs can consist of many nodes, which means that pro-active routing protocols will perform badly, since there will be a lot overhead in the network due to LSA packets, or distance vectors (caused by changing neighbour lists because of the mobility of the nodes).

1.4 Software Defined Networking

One of the upcoming trends in communication networking is Software Defined Networking (SDN). This technology enables the management of networks to be much more flexible than technologies used nowadays. One of the defining properties of SDN is the separation of data and control plane. The control plane is where forwarding decisions are made, whereas the forwarding plane does the actual forwarding. For example, in routing protocols, the routing table is part of the forwarding plane. The configuration of flows in the routing table is done in the control plane.

Using SDN technology, the control plane will become a central point from where all the forwarding decisions will be made. This yields a global overview of the whole network and also implies that only the control plane will have to be programmed (instead of every switch in the network). This makes the system much more flexible, e.g., with respect to software updates and the distribution of network traffic. It also enables programmers to configure the network on a more abstract level, which increases the manageability of the network. However, end-to-end routing between communication components in different domains would still require a coordinated, decentralized routing mechanism.

One of the main questions is whether or not SDN can address scalability issues of networks. In [4] it is shown that the scalability issues for traditional networks are no worse than those of SDN-driven networks. Note that SDN technology is not (directly) applicable in, for example, MANETs, due to the fact that those networks have no predetermined infrastructure.

1.5 Multipath routing

In the descriptions of various network types in the previous sections, some network routing concepts have been introduced. These concepts describe how a single shortest path between source and destination can be established. It is also possible to use *multipath routing*, where multiple paths between a source and destination are established. Multipath routing consists of three elements: route discovery, traffic distribution and route maintenance. The only difference with single path routing is the element of traffic distribution, which is now possible because data can be send over multiple routes.

We will explain some of the benefits of multipath routing, which have been taken from [5].

i) **Load balancing:** If a link becomes over-utilised, which leads to congestion, multipath routing can be used to select another path not using the congested link.

ii) **Bandwidth aggregation:** Data can be split and send over multiple routes in order to use the available bandwidth efficiently. This also means that the data will arrive faster at the destination node (if the paths use distinct network links or hardware components).

iii) **Reduced delay:** In particular in reactive on-demand routing, when there is a path failure, a new route discovery process has to be initiated, leading to a significant delay in the data delivery. If additional paths have already been identified (not using the failing components of the original path), then we do not experience this type of delay.

iv) **Reliability:** By sending the same message over multiple paths, there is a higher probability of the message arriving at the destination node.

v) **Security:** Splitting of data, as done in bandwidth aggregation, provides an extra layer of security. If someone is eavesdropping on a certain network link, he can never retrieve all the data that is being sent (although encryption of the data might already be sufficient).

In order to make use of these benefits, paths need to satisfy certain *disjointness* conditions. The two main types of disjoint paths are *node-disjoint* and *arc-disjoint* (link-disjoint) paths. Node-disjoint paths do not have any nodes (components) in common, whereas link-disjoint paths do not have any network links in common. Note that node-disjointness implies link-disjointness. In case node- and link-disjoint paths do not exist between a given source-destination pair, there is also the possibility of having *maximally disjoint* paths. These can, for example, be paths that have the

least amount of nodes (or links) in common.

The type of paths required, and also the number of paths, depends on the network and the purpose of multipath routing. For example, in wired networks, the most vulnerable element of the network is the collection of network links, meaning that it is sufficient to have link-disjoint paths in order to overcome a link failure. In MANETs, a link might fail due to the fact that a node moves behind a building.

There is one major difference between wired and wireless networks when it comes to multipath routing. In wired networks, when data is being sent over two distinct network links, this means it is being sent over two physically different cables. However two wireless connection links might have overlap in their transmission range, which can lead to interference. This is also known as *route coupling*. Paths for which there is no overlap between transmission ranges of nodes on different path are called *zone-disjoint* paths. In order to overcome the problem of route coupling, it is possible to use multiple channels, i.e., the usage of different frequencies for different paths, or directional antennae that make it possible to broadcast into a more specific geographical area (in order to create zone-disjoint paths).

In general, load balancing and bandwidth aggregation require link-disjoint paths, whereas reduced delay requires either link-disjoint or node-disjoint paths, depending on the network and its failing element. Furthermore, load balancing and reduced delay can be performed with one additional path. For bandwidth aggregation, we would like to have as many disjoint paths as possible.

We can also take into account that a set of disjoint paths has to satisfy a certain routing metric. For example, it is possible to use bandwidth aggregation to minimize the time it takes for all data to reach the destination. Suppose that we want to have k disjoint paths over which we would like to send an equal amount of data, and that every path has a certain end-to-end delay (as routing metric), i.e., the time it takes to get the data from the source to the destination. Then we would like to find k disjoint paths so that the maximum delay, among the k paths, is minimized. That is, we want to minimize the time at which the last data fragment arrives at the destination.

In case of reduced delay, depending on the probability of a failure in the network, we might want to have one short path and one back-up path, of which the routing metric of the latter is less important (or a back-up path minimizing the expected delay for the original and back-up path).

In conclusion, depending on the objective behind the application for multipath routing, different optimization problems arise for selecting the desired paths.

Chapter 2

Routing concepts

In this section we will discuss the most important routing concepts. To do this, we will look at communication networks as mathematical graphs. We will first introduce some graph theoretical notation and discuss both central and distributed algorithms for the routing concepts. Central algorithms are useful, e.g., in SDN or link-state routing protocols. Distributed algorithms can, for example, be used for distance vector routing in MANETs. We will not describe the algorithms in full detail.

2.1 Notation

Let $G = (V, A)$ be a *directed* graph, where V is a finite set and

$$A \subseteq \{(u, v) : (u, v) \in V \times V\}$$

a set of directed arcs in G (see also Example 2.1.1). Let $c : A \to \mathbb{R}$ be a cost function on the arcs of G, where we use the notation

$$c_{uv} = c(u, v)$$

for $(u, v) \in A$. For $u \in V$ we define

$$\mathcal{N}^+(u) = \{v : (u, v) \in A\} \quad \text{and} \quad \mathcal{N}^-(u) = \{v : (v, u) \in A\}$$

and say that v is a *neighbour* of u if $v \in \mathcal{N}^+(u)$. We say that $P = p_0 p_1 \dots p_l$ is a *simple path* in G if

$$V(P) = \{p_0, \dots, p_l\} \subseteq V \quad \text{and} \quad (p_{i-1}, p_i) \in A \text{ for } i = 1, \dots, l$$

with all the p_i distinct. We then define

$$A(P) = \{(p_{i-1}, p_i) : i = 1, \dots, l\}.$$

We say that P is an a, b-path if $p_0 = a$ and $p_l = b$. Furthermore, for $p_i, p_j \in V(P)$ we write

$$Pp_i = p_0 p_1 \dots p_i, \; p_i P = p_i \dots p_l, \; p_i P p_j = p_i \dots p_j$$

and for $u \in \mathcal{N}^+(p_l) \setminus V(P)$, we write $Pu = p_0 p_1 \dots p_l u$. We denote

$$\mathcal{P}_G = \{P : P \text{ is a simple path in } G\}, \quad \mathcal{P}_{G,s} = \{P : P \text{ is a simple path in } G \text{ with } p_0 = s\}$$

and

$$\mathcal{P}_{G,s,d} = \{P : P \text{ is a simple } s, d\text{-path in } G\} \subseteq \mathcal{P}_G.$$

Two simple paths $P = sp_1 \dots p_l d$ and $Q = sq_1 \dots q_{l'} d$ are *node-disjoint s, d-paths* if $V(P) \cap V(Q) = \{s, d\}$. We say that k simple paths are node-disjoint s, d-paths, if they are pairwise node-disjoint

s, d-paths. Two simple paths $P = sp_1 \dots p_l d$ and $Q = sq_1 \dots q_{l'} d$ are *arc-disjoint s, d-paths* if $A(P) \cap A(Q) = \emptyset$. We say that k simple paths are arc-disjoint s, d-paths, if they are pairwise arc-disjoint s, d-paths. Furthermore, we define

$$c : \mathcal{P}_G^k \to \mathbb{R}$$

as a cost function on all tuples of k simple paths in G. It will follow from the context whether we mean the cost function for the arcs or for the paths. The goal is to solve the problem

$$\min \left\{ c(P_1, \dots, P_k) \ : \ (P_1, \dots, P_k) \in \mathcal{P}_{G,s,d}^k \text{ and } P_1, \dots, P_k \text{ are node-disjoint } s, d\text{-paths} \right\}. \quad (2.1)$$

We will refer to the arc-disjoint version of this problem as the formulation where 'node-disjoint' is replaced by 'arc-disjoint'. For $k = 1$, we call (2.1) the single path problem, and for $k > 1$, the multipath problem.

Example 2.1.1. In Figure 2.1 we have given an example of a directed graph. This is also an example of a *bidirectional* graph, i.e., a graph for which $(u, v) \in A$ if and only if $(v, u) \in A$. Note that for a bidirectional graph, we have $\mathcal{N}^+(v) = \mathcal{N}^-(v)$ for every $v \in V$. The red arcs indicate the simple s, d-path $P = sv_1v_4d$. Furthermore, we have

$$\mathcal{P}_{G,s,d} = \{sv_1v_3d, sv_1v_4d, sv_2v_4d, sv_2v_4v_1v_3d\}$$

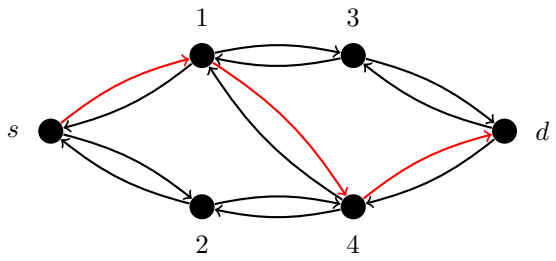

Figure 2.1: Example of a graph $G = (V, A)$.

For $k = 2$ and $c : A \to \mathbb{R}$ a constant non-negative function, there is a unique optimal solution for (2.1) given by the paths $P = sv_1v_3d$ and $Q = sv_2v_4d$. Note that this is also the optimal solution for the arc-disjoint version of (2.1). □

2.2 Central algorithms

In this section, we will give some algorithms that can solve (2.1) using as input a graph $G = (V, A)$ and cost function $c : A \to \mathbb{R}$. These are called *central* algorithms. Here, we take

$$c(P_1, \dots, P_k) = \sum_{i=1}^{k} c(P_i)$$

where $c(P) = \sum_{i=1}^{l} c_{p_{i-1}p_i}$ for a path $P = p_0 \dots p_l$. This problem will be referred to as the *Min-Sum* problem, or simply Min-Sum.

For a general $c : A \to \mathbb{R}$, solving Min-Sum is NP-complete for every k. For $k = 1$, this can be seen by a reduction to the Hamiltonian path problem: the problem of finding a directed path in G that visits every node exactly once. We can define $c_{uv} = -1$ for all $(u, v) \in A$ and solve (2.1) for all $s, d \in V$. If there exists an optimal solution with objective value $1 - |V|$, this means that there exists a Hamiltonian path. Furthermore, the problem of finding k paths can be reduced to the problem of finding $k + 1$ paths, so this means the problem is NP-complete for all k.

If $c : A \to \mathbb{R}$ satisfies the condition that there are no negative weight cycles, then it is solvable in time polynomial in $|V|$. For $k = 1$, this is simply the shortest path problem and can be solved

using, for example, the Bellman-Ford algorithm (e.g. [6], [7]), that runs in time $\mathcal{O}(|V||A|)$. A high-level description of this algorithm is given in Algorithm 2.2.1.

Algorithm 2.2.1. *(Bellman-Ford algorithm)*
The algorithm constructs functions $f_0, \ldots, f_n : V \to \mathbb{R} \cup \infty$, so that $f_b(v)$ is the objective value of a shortest s, v-path, using at most b arcs. This means that $f_n(v)$ is the objective value of a shortest s, v-path.

i) Set $f_0(s) = 0$ and $f_0(v) = \infty$ for all $v \in V \setminus \{s\}$.

ii) For $b = 0, \ldots, n-1$ and $v \in V$:

$$f_{b+1}(v) = \min_{(u,v)\in A}\{f_b(v), f_b(u) + c(u,v)\}.$$

Furthermore, we can define $g(v)$ as the node for which $f_{b+1}(v) = f_b(g(v)) + c(g(v), v)$ (and update it whenever $f_{b+1}(v) < f_b(v)$). The nodes $g(v)$ can then be used to trace back the shortest path s, d-path, i.e., the path formed by the nodes $d, g(d), g(g(d)), \ldots, s$. □

Example 2.2.1. Let G be the graph from Figure 2.1, and $c : A \to \mathbb{R}$ defined by

$$c_{uv} = \begin{cases} -1 & \text{If } (u,v) = (1,4) \\ 2 & \text{If } (u,v) = (s,2) \\ 1 & \text{Otherwise} \end{cases}$$

Note that G then does not contain negative weight cycles. In Table 2.2.1, we have given an overview of the values $f_b(v)$ for $b = 0, \ldots, 4$.

b \ v	s	1	2	3	4	d
0	0	∞	∞	∞	∞	∞
1	0	1	2	∞	∞	∞
2	0	1	2	2	0	∞
3	0	1	1	2	0	1
4	0	1	1	2	0	1

Table 2.1: Overview of the values $f_b(v)$ of the Bellman-Ford algorithm for the graph in Figure 2.1.

For every $v \in V$, the shortest s, v-path consists of at most three arcs, which means that the algorithm terminates after three steps ($b = 3$). This also follows from the fact that the $f_3(v) = f_4(v)$ for all $v \in V$. □

For $k = 2$, we can use the Edge Disjoint Shortest Pair (EDSP) algorithm ([8]) to solve the arc-disjoint version of the problem. A high-level description is given in Algorithm 2.2.2.

Algorithm 2.2.2. *(Edge Disjoint Shortest Pair algorithm)*
The algorithm consists of the following steps.

i) Determine a shortest s, d-path P_0.

ii) Let $G' = (V, A')$ where

$$A' = (A \setminus A(P_0)) \cup A(P_0^{-1})$$

with $(v,u) \in A(P_0^{-1})$ if and only if $(u,v) \in A(P_0)$ and $(v,u) \notin A$. Furthermore, we define

$$c'_{uv} = \begin{cases} -c(v,u) & \text{If } (v,u) \in A(P_0) \\ c(u,v) & \text{Otherwise} \end{cases}$$

iii) Determine a shortest (augmenting) s, d-path P_1 in G', for the cost function c'.

iv) Construct two edge-disjoint paths from the symmetric difference of P_0 and P_1. □

Note that the graph G' does not contain negative cost cycles, since P_0 is a shortest s,d-path. Furthermore, the shortest path algorithm used in the first and third step must be able to work with arbitrary cost functions that do not give rise to negative cost directed cycles. The *symmetric difference* of the paths P_0 and P_1 is defined as the graph $G(P_0, P_1) = (V, A)$ where

$$V = V(P_0) \cup V(P_1) \quad \text{and} \quad A = (A(P_0) \cup A(P_1)) \setminus A^*$$

Here, A^* is the set of arcs forming directed cycles of length two, i.e., $(u,v),(v,u) \in A^*$ if $(u,v) \in A(P_0)$ and $(v,u) \in A(P_1)$.

The node-disjoint version of the problem can be solved by using vertex-splitting methods [8]. Intuitively, we split up every node $v \in V$ into two nodes v^+ and v^-. We connect all the outgoing arcs with v^+ and all the incoming arcs with v^-. Furthermore, we add the arc (v^-, v^+) with cost zero.

Example 2.2.2. Let G and c be as in Example 2.2.1. Then the shortest s,d-path in G is given by $P_0 = sv_1v_4d$. The resulting graph G' is given in Figure 2.2.

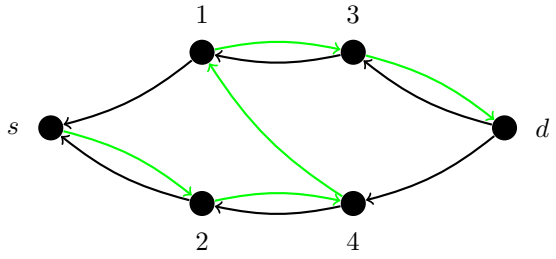

Figure 2.2: The graph $G' = (V, A')$, for G and c as in Example 2.2.1.

The green arcs indicate the shortest s,d-path P_2 in the graph G' (this is the only s,d-path). The symmetric difference of the path P_1 and P_2 is then given by the union of the paths $Q_1 = sv_1v_3d$ and $Q_2 = sv_2v_4d$ (note that $A^* = \{(1,4),(4,1)\}$). □

If $c : A \to \mathbb{R}_{\geq 0}$, then we can use Dijkstra's algorithm ([9]) for $k = 1$, which runs in time $\mathcal{O}(|V|^2)$. Note that this is (worst-case) better than the Bellman-Ford algorithm. If $k = 2$, there is also an improvement of the EDSP algorithm, namely Suurballe's algorithm ([10],[11]). A high-level description is given in Algorithm 2.2.3.

Algorithm 2.2.3. *(Suurballe's algorithm)*
The algorithm consists of the following steps.

i) Determine a shortest path tree T (with root s), i.e., for $v \in V$ the s,v-path in T is a shortest s,v-path. Let P_0 be the s,d-path in T.

ii) Let $G' = (V, A')$ where

$$A' = (A \setminus A(P_0)) \cup A(P_0^{-1})$$

with $(v,u) \in A(P_0^{-1})$ if and only if $(u,v) \in A(P_0)$ and $(v,u) \notin A$. Furthermore, for all $(u,v) \in A'$, we define

$$c'(u,v) = c(u,v) - d(s,v) + d(s,u)$$

where $d(s,v)$ is the cost of the s,v-path in T.

iii) Determine a shortest (augmenting) s,d-path P_1 in G'

iv) Construct two edge-disjoint paths from the symmetric difference of P_0 and P_1.

□

By construction we have that c' is a non-negative cost function, meaning that we can use Dijkstra's algorithm in the first and third step. This implies that the running time of the algorithm is $\mathcal{O}(|V|^2)$, since the second and fourth step can also be done in $\mathcal{O}(|V|^2)$. A difference with the EDSP algorithm is that we need a shortest path tree in Suurballe's algorithm to determine the cost function c', as opposed to just a shortest path in the case of EDSP. Note that the simple path P is a shortest s,d-path under c if and only if it is a shortest s,d-path under c'.

The correctness of both the EDSP and Suurballe's algorithm follows from the fact that the problem of finding k arc-disjoint paths, between a given source and destination, can be formulated as a *minimum-cost flow* problem (see e.g., [12]). This problem can be solved using the concept of augmenting paths (as used in these two algorithms).

Example 2.2.3. Let G and c be as in Example 2.2.1. Then the shortest path tree T is given by the red arcs in Figure 2.3. The values $d(s,v)$ can be found in Table 2.2.1, on the row for $b = 3$.

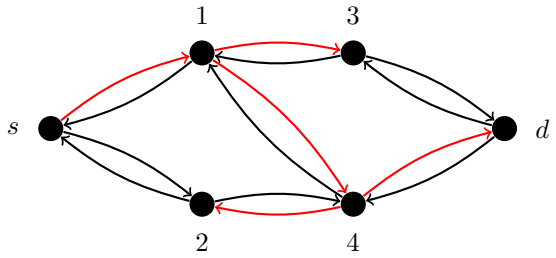

Figure 2.3: Example of a graph $G = (V, A)$.

□

Some other well-known choices for the cost function $c : \mathcal{P}_G^k \to \mathbb{R}$ are the following. The Min-Max problem is c defined by

$$c(P_1, \ldots, P_k) = \max\{c(P_1), \ldots, c(P_k)\}$$

which is NP-complete, already for $c : A \to \mathbb{R}_{\geq 0}$ (e.g. [13]). Solving this problem can be useful for minimizing the latency of k disjoint paths, used for transferring data between source and destination.

The Min-Min problem is (2.1) with c defined by

$$c(P_1, \ldots, P_k) = \min\{c(P_1), \ldots, c(P_k)\}$$

and is also NP-complete (see [14]). Solving this problem can be useful when a disjoint back-up path (having no additional requirements) is needed.

2.3 Distributed algorithms (pro-active)

In this section, we will describe and comment on some *distributed* versions of the algorithms from the previous section. Instead of having one central processor calculating s,d-paths, using the whole graph as input, nodes all have a processor that can perform calculations, based on information from neighbouring nodes. Nodes can exchange (network) information, that can be used to establish routing tables. Note that it is possible for both wired and wireless networks to be defined as directed graphs.

We will explain how we can translate the Bellman-Ford algorithm to a distributed version that can be used in distance-vector routing. Remember that in distance-vector routing, every node on the chosen shortest s,d-path stores the next node on the path, so that whenever a message destined for the destination arrives, it can forward it to the next node on the shortest path. We will assume here that the graph G is bi-directional, i.e., $(u,v) \in A \Leftrightarrow (v,u) \in A$, and $c_{uv} = c_{vu}$, and that c is a non-negative cost function.

Every node u maintains an (incomplete) vector of distances (costs) to other nodes d in the network, together with the next node *next* on the path to reach d. We will denote the distance with C. For every node d, this information can be stored in the triplet

$$\big[d, \mathit{next}(u,d), C(u,d)\big]$$

where the distance vector then consists of all these triplets. Initially, this vector contains the neighbours, i.e., $d = v \in \mathcal{N}^+(u)$.

The node u sends this vector to all neighbours v, that can use it to update their own vector. That is, if some v receives a distance vector of u such that there is a destination d for which

$$C(u,d) + c(u,v) < C(v,d),$$

then node v updates its triplet $\big[d, \mathit{next}(v,d), C(v,d)\big]$ to

$$\big[d, u, C(u,d) + c(u,v)\big]$$

Here we use the assumption that G is bi-directional, since the path will use the arc (v,u) although the vector arrives over the arc (u,v). Note that there exists a clear relation with the central Bellman-Ford algorithm in this step. Furthermore, the distance vector will give next hops for all other nodes in the network (instead of for a fixed destination).

A high-level description of the algorithm is given in Algorithm 2.3.1. Note that this description is a protocol performed by an individual node v. The moment that the initial vector is created does not have to be the same for all nodes. This can happen when the first distance vector is received from a neighbour. This means that, for example, a network administrator must tell at least one node to create its distance vector, but apart from that, the process can be completely automated.

Algorithm 2.3.1. *(Distributed Bellman-Ford algorithm)*
A node v performs the following steps.

(1) Create initial distance vector, containing the distances to all direct neighbours.

(2) Send distance vector to all neighbours.

(3) Upon receiving a distance vector from a neighbour: update distance vector where possible. That is, if it contains a destination not yet known, add that triplet to the distance vector, or, if

$$C(u,d) + c(u,v) < C(v,d)$$

then update the known triplet. If an update has been performed, go to (2).

□

Example 2.3.1. Consider the graph and cost function from Example 2.2.1. The initial distance vector of node s is given by

s	-	0
1	1	1
2	2	2

Table 2.2: Initial distance-vector of node s.

This distance-vector is sent to both node v_1 and v_2, for which the distance-vectors after updating become:

s	s	1
1	-	0
2	s	2
3	3	2
4	4	0

s	s	1
2	-	0
1	s	2
4	4	2

Table 2.3: Distance-vector of node v_1 (left) and v_2 (right).

This process can be repeated to find the complete routing tables for all nodes of the graph. □

There are some well-know issues for this algorithm, such as *bouncing effect, counting to infinity* and *looping*, that mostly occur when a link fails and a new route has to be found. There also exist adjustments of this basic algorithm that resolve these issues (e.g., [15]).

A distributed implementation of Suurballe's algorithm is described in [16]. In general it is not difficult to come up with a distributed version of a central algorithm, but a lot of technical details are involved, for example in order to handle failed transmissions or link failures.

In this project we are more interested in the concept of *source routing*, which is the reason we do not discuss link-state and distance-vector routing in detail. However, in order to give a complete overview of routing concepts, we have discussed some of these methods.

2.4 Distributed algorithms (reactive on-demand)

Distance-vector and link-state routing are two examples of *pro-active* routing, i.e., paths between routers are determined regardless whether or not they will be used. This is not a problem when the network is *static*, which means that there are not much topology changes.

In dynamically changing network topologies, pro-active routing creates a lot of overhead (in terms of LSA packets or distance vectors). In order to solve this problem, *reactive on-demand* routing can be used. Here, a route between a source and destination is only established when data has to be sent. The source node initiates the process of discovering a route.

There exist various adjustments for distance-vector and link-state routing protocols in order to have them work in a reactive on-demand way, e.g., Ad hoc On-Demand Distance Vector routing (AODV), see [17]. Another routing principle, used a lot within dynamically changing topologies, is *source routing*. Here, the next node for a path to the destination is not stored in a routing table, but the path to the destination is explicitly stored in the packets that are being sent. In order to do this, we first need to determine a path between the source and destination.

One of the major advantages of source routing is that a message will not be routed in a loop, because the path is contained in the message itself.

2.4.1 Dynamic Source Routing (DSR)

We will give a short description of the Dynamic Source Routing protocol (DSR) - a reactive on-demand routing protocol that uses source routing - in order to describe a route discovery process (see [18]). Suppose we have a source node s that wants to send data to a destination node d, but that no path to the destination is known. Initially, the route discovery process works as follows. The source broadcasts a Route Request message (RREQ) containing its address and the address of the destination node. If a neighbouring node receives the RREQ, it appends itself to the *route record*, that will serve as a list of nodes this specific RREQ has traversed (which in the end gives an s, d-path). Afterwards, it broadcasts the updated RREQ to the neighbours within its transmission range. In general, if a node h receives a RREQ, it appends itself to the route record and rebroadcasts the RREQ, if it is not the destination node d. The source node also adds a sequence number to the initial RREQ and every intermediate node processes at most one RREQ of a sequence number. This means that in a network of N mobile nodes, there will be at most N broadcasts of RREQ messages (for a fixed sequence number). Eventually, the destination node will

receive a number of RREQ messages corresponding to paths between the source and destination (it receives precisely one path from all of the nodes within its transmission range). It can select an optimal path, with respect to some routing metric, and initiate a Route Reply message (RREP) that contains this path. This message will be broadcast by the nodes of the optimal path in the reversed direction, so that it will eventually reach the source node, that then has a path to the destination. Furthermore, instead of just forwarding an RREQ, an intermediate node can also store the path from the route record. These stored routes can be used to construct paths between other sources and destinations. In this way a node constructs a partial view of the network.

We will given an example of the route discovery process, using the graph given in Figure 2.4. Here we have used an undirected graph (which is equivalent to a bidirectional directed graph).

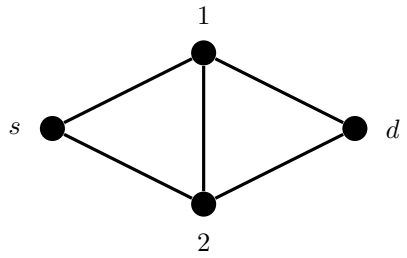

Figure 2.4: Example of a graph $G = (V, E)$ with $V = \{s, 1, 2, d\}$ and E the set of edges.

At first s sends out an RREQ, that will be received by nodes v_1 and v_2. These nodes append themselves to the route record and rebroadcast the RREQs, corresponding to respectively the routes sv_1 and sv_2. Both v_1 and v_2 will receive each other's broadcast, but do not rebroadcast these, because they already have rebroadcast the RREQ of node s. Node d will receive the RREQs of both v_1 and v_2 and will recognize itself as the destination. It will then select the route sv_id for $v_i \in \{v_1, v_2\}$, depending on which broadcast arrives first, and broadcast an RREP of the selected route. That will be broadcast by v_i, leading to the arrival of an s, d-path at node s.

In a network of N mobile hosts, there will be N RREQ messages (corresponding to a fixed sequence number) that are broadcast. Note that this procedure only gives a feasible s, d-path, and not necessarily an optimal route with respect to some routing metric. In order to solve this problem, every host can store the routing metric of an RREQ messages that it has broadcast. Whenever another RREQ message, containing a path with a better routing metric, arrives, it can rebroadcast that RREQ message. For example, for the graph in Figure 2.4, if the path sv_1v_2 has a better routing metric than the path sv_2, host v_2 can rebroadcast the RREQ that it received from host v_1.

From a mathematical point of view, this is equivalent to a distributed implementation of the Bellman-Ford algorithm, which is for example used in distance-vector routing protocols.

2.4.2 Broadcast storm problem

One of the main problems with DSR is the number of broadcasts that can occur. If multiple hosts are close together and all start rebroadcasting an RREQ at approximately the same time, collisions can occur, leading to a failure in correctly receiving the RREQs. For example, in Figure 2.4, if both host v_1 and v_2 rebroadcast the RREQ of s almost at the same time, a collision can occur. The problem of having many broadcasts is known as the *broadcast storm* problem ([19]). The problem is not inherent to DSR, but is defined in a more general context, when a host wants to send one particular message to all other hosts. That is, the content of the message is not changed throughout the broadcast process, as opposed to DSR where the route record is updated at intermediate hosts. Nevertheless, this problem, and the proposed solutions, also apply to the DSR algorithm.

In [19], solutions have been proposed for reducing the number of broadcasts (and the probability of collisions). We will give a short description of the solutions. It is assumed that the hosts are distributed in a geographical area, meaning that the location of host A can be expressed by

the coordinates (x_A, y_A). Some of the solutions depend on the analysis of the (additional) area that a broadcast can cover, depending on the number of broadcasts it has already received.

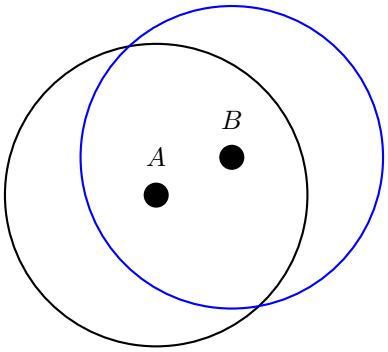

Figure 2.5: Sketch of additional coverage of host B, given the coverage of host A.

For example, if B is within the transmission range of host A and receives a message from A, then the additional area that node B can cover is somewhere between 0%-60% of the area covered by A (assuming equal transmission ranges). In general, the average additional area that can be covered by a broadcast of node B, decreases as the number of received broadcasts (from different hosts) increases.

It is clear that the probability of a collision will decrease when the number of broadcasts decreases. Another technique used to avoid collisions is by adding a random delay to the rebroadcasting of a message. For example, in the graph of Figure 2.4, if both v_1 and v_2 wait some small random time before rebroadcasting the message, the probability that they will start broadcasting at the same time decreases significantly. Techniques such as these can, for example, be found in PANDA, see [20]. Here, all neighbours get a delay related to the distance of the neighbour to the original host. The farther away from the host, the shorter the given delay.

We will now describe four schemes that can be used to reduce the number of broadcasts. We will explain schemes that a host h can use to determine whether or not to rebroadcast the message.

i) **Probabilistic Scheme:** Whenever h receives a message, it will be rebroadcast with probability $p \in [0, 1]$. Note that $p = 1$ corresponds to the original situation.

ii) **Counter-Based Scheme:** A counter threshold C is used to keep a record of the number of times a broadcast has been received. When a broadcast is received by h for the first time, a random number of time slots is chosen. If within those time slots, $C - 1$ more broadcasts are received, then the message is not rebroadcast. Otherwise, if after the random number of time slots the threshold has not been reached, the message is rebroadcast.

iii) **Distance-Based Scheme:** This scheme uses a similar approach as the previous one. A counter threshold D is used to keep a record of the host closest to h, of which h has received the broadcast. After receiving the first broadcast, h waits a random number of time slots. If within those time slots a broadcast is received by a host h', for which the distance to h is smaller than the threshold, the messages is not rebroadcast.

iv) **Location-Based Scheme:** This scheme assumes that host h knows the positions of the hosts from which it receives the broadcast. Again, after receiving the first broadcast, a random number of time slots is chosen. For all the broadcasts arriving within that time, the locations of the corresponding hosts can be used to give a precise calculation of the additional area that can be covered (on top of the union of the areas of all the received broadcasts). If this value is lower than some threshold value A, the message is not rebroadcast.

In order to compare these methods, there exist various performance indicators, e.g., the number of nodes that actually receive the broadcast or the average latency: the time between the initiation of the broadcast and the last rebroadcast. The location-based scheme has the best performance, but it has the disadvantage of having to perform a lot of calculations to determine

the additional coverage area. The first method is the simplest, but does not take into account any advantages that can be obtained from knowing the locations of hosts that have sent broadcasts already.

There also exist methods that establish a dominating set, i.e., a set of hosts so that the union of the areas they cover, contains all hosts. This method is called *multipoint relaying* ([21]). The hosts in the dominating set are called multipoint relays (MPRs). Finding the minimal dominating set is NP-hard, so therefore often heuristic methods are used. In [21], a heuristic method is presented that finds a dominating set within a factor $\log(n)$ of the optimal solution, where n is the number of nodes in the network.

MPRs are also used in, for example, the Optimized Link State Routing Protocol (OLSR), see [22]. The MPRs serve as intermediate nodes on paths between sources and destinations.

In Location-Aided Routing (LAR), see [23], the source node knows the geographical location (and speed) of the destination node. This information is then used to determine a *request zone*, i.e., a region containing both the source and destination node. An intermediate node responds to an RREQ if and only if it is in the request zone. Intuitively, the network is restricted to a subset of nodes for which it is likely that a good s, d-path exists among them. If the speed of a node is also known, then this information can be incorporated in the determination of the request zone.

2.4.3 Multipath protocols

We will now describe some popular multipath routing protocols based on DSR. We will focus on the route discovery process, since this is the most relevant part for this thesis.

The first protocol we discuss is a direct extension of the DSR protocol by Napsipuri and Das ([24]). Instead of only replying to the RREQ containing a shortest path (primary path), the destination can also reply to other RREQs, containing paths that are link-disjoint with the primary path. The number of RREQs the destination responds to, is an input parameter of the protocol. A disadvantage of this method is that only the source node has knowledge of multiple paths. For that reason, a second method is proposed, in which the destination node replies to every intermediate node of the primary path with a link-disjoint path to that node. In this way, whenever a failure occurs, instead of having to inform the source node that an alternative path is needed, an intermediate node itself can continue the message via another path.

The second protocol we will discuss is Split Multipath Routing (SMR), as introduced in [25]. Just as the DSR protocol, the first RREQ that is received by a node is broadcasted (and the route from the route record stored). Whenever another RREQ is received, it is rebroadcast only if the routing metric of the corresponding path is better and the neighbouring node, from which the RREQ has been received, is different than the neighbouring node of which the first RREQ has been received. Although this method requires more RREQs to be send in the network, the resulted paths are more disjoint than those of the direct extension of the DSR protocol.

The third protocol ([26]) divides the network into two disjoint sets of nodes. This division can, for example, be made based on the identifiers of the nodes. In an IP-based network, it can be possible to take the sets of even and odd IP-addresses. The source node broadcasts two RREQs: one for the even set of addresses and one for the odd set. An intermediate node only reacts to the RREQ corresponding to the set it is in. This means that the RREQ must contain information about the set it is meant for (which can be added by the source node). In this way, paths arriving at the destination, corresponding to different sets, are by construction node-disjoint paths.
The last protocol we mention constructs two node-disjoint paths, using the idea of an augmenting path, see [27]. This method is also based on dynamic source routing.

An overview of other routing protocols for multipath discovery, also based on AODV, can be found in [5].

2.5 Overview

In this chapter we have introduced various routing concepts. An important factor that influences the choice of routing protocol, is how often the network topology changes. In static communication networks, this does not happen a lot, whereas in dynamic networks, the topology changes very often. In routing protocols, we have the transition

$$\text{Pro-active routing} \quad \rightarrow \quad \text{Reactive on-demand routing}$$

when the network topology goes from

$$\text{Static} \quad \rightarrow \quad \text{Dynamic.}$$

The reason for this is that it is no longer useful to establish routes before they are used, as in pro-active routing, since the lifespan of routes decreases significantly when the network topology becomes more dynamic. The same holds for the usage of central algorithms, where we first need to establish a central view of the network.

The advantage of pro-active routing is that data can be sent the moment it is requested, which is not the case for reactive on-demand routing, since there we first need to establish a route. This can be seen as a trade-off between additional overhead due to actively updating routes, and an additional delay for the time it takes to get the data from source to destination.

For reactive on-demand routing, the two most popular protocols are Dynamic Source Routing (DSR) and Ad hoc On-demand Distance Vector routing (AODV). The choice of protocol mostly depends on the type of communication network. For example, AODV performs better in networks with high mobility (e.g., in MANETs), whereas DSR creates less routing overhead.

An overview of the routing concepts described in this chapter can be found in Figure 2.6.

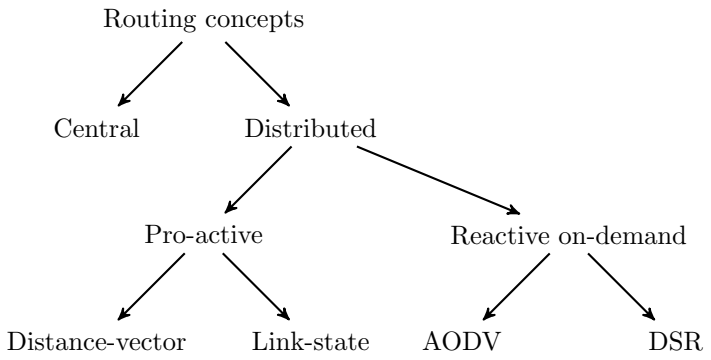

Figure 2.6: Overview of routing concepts described in this chapter.

Chapter 3

Goal of the project

In this project we analyse a distributed algorithm for finding an optimal set of disjoint routes in a communication network, between a given source and destination. In this chapter, we will give an informal introduction to the problem we are solving, and the techniques that are used.

A communication network can be formulated in terms of an undirected graph $G = (V, E)$, where V is the set of hardware components, and E the set of network links between the components in V. Routes between a source s and destination d are called s, d-*paths*. Furthermore, paths are *node-disjoint* if they do not have any nodes in common, except for the source and destination. We also define a cost function $c : E \to \mathbb{R}$, that assigns a cost to every edge of the network. The exchange of network information, in order to establish s, d-paths, is called the *route discovery process*.

The distributed algorithm, formulated in a patent of the company TNO, has originally been proposed in order to provide an algorithm that:

i) finds an optimal set of k node-disjoint paths for a given source-destination pair, i.e., solve the problem
$$\min \{c(P_1, \ldots, P_k) \ : \ P_1, \ldots, P_k \text{ are node-disjoint } s, d\text{-paths}\},$$

ii) can be executed by cooperative nodes in a network, i.e., a distributed algorithm,

iii) tries to use a minimum number of route discovery messages,

iv) does not allow intermediate nodes to store any information from route discovery messages (messages are processed independent of each other).

It will follow from the context whether we mean the cost function c for the edges or the paths. Furthermore, it is assumed that nodes in the network know who their direct neighbours are, i.e., every node $v \in V$ has a list of the nodes $u \in V$ for which $\{u, v\} \in E$, and their cost c_{uv}. As already stated in the previous two sections, there exist various algorithms for finding paths that satisfy the first three conditions (in the context of directed graphs). To the best of our knowledge, algorithms that also satisfy the fourth condition have not been studied. The incentive for the fourth condition is that nodes might be alleviated if they do not have to store information regarding the route discovery process. It might also be beneficial for reasons of safety and security.

The algorithm described in the patent uses the same *path extension* principle as in DSR (see Section 2.4.1). However, when a node v receives an RREQ, it does not only append itself to the route record, but also includes its list of neighbours u, and the costs c_{uv} to reach them. Roughly speaking, the edge $\{a, b\}$ is stored in the RREQ if and only if the path, corresponding to the RREQ, uses at least one of the nodes a or b. This set of information will be referred to as *neighbour information*. Whenever a node v receives an RREQ, it can append its own neighbour list and costs to the RREQ. The resulting message will be referred to as the *updated* RREQ at node v.

The idea is now that we want to use this neighbour information to determine whether or not to forward the RREQ to a certain neighbour (as opposed to DSR, where it is forwarded to all

neigbhours, based on a sequence number). Note that this means that the forwarding is done solely based on information already contained in the (updated) RREQ, and that a node might process more than one RREQ.

The goal is to find one or more optimal disjoint s, d-paths. This means that we would like to have multiple RREQs arriving at the destination d, of which the corresponding paths form an optimal solution. In order to minimize the number of RREQs, we do not want to forward an RREQ to a neighbour w, if the resulting path cannot be part of some s, d-path P' that is contained in an optimal solution. Therefore, we will investigate how we can use the neighbour information to determine whether or not forwarding an RREQ to a neighbour is useful. For example, consider the graph in Figure 3.1,

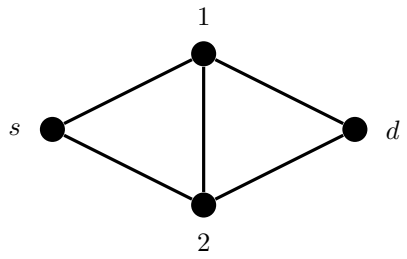

Figure 3.1: Example of a graph $G = (V, E)$, where $c(u, v) = 1$ for all $\{u, v\} \in E$.

and the updated RREQ corresponding to the path $P = sv_1$ given in Figure 3.2. We take $c_{uv} = 1$ for all $\{u, v\} \in E$, and would like to minimize the aggregated cost of a path, which in this case reduces to minimizing the number of nodes on an s, d-path. Note that the edge $\{s, 2\}$ is appended to the initial RREQ that is sent to nodes v_1 and v_2.

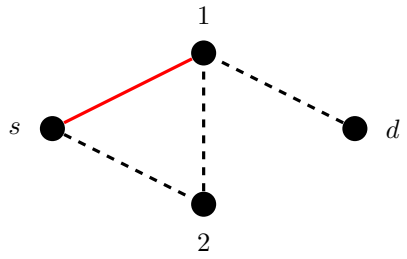

Figure 3.2: Example of updated RREQ at node v_1, corresponding to the path $P = sv_1$.

We will now show that it is useless to send the updated RREQ to node v_2, because we can argue that sv_1v_2 can never be part of an optimal s, d-path, based on the information from the updated RREQ in Figure 3.2. This follows directly from the fact that if sv_1v_2 would be extended to some optimal s, d-path P', then we can replace the part sv_1v_2 with sv_2. Since $c_{sv_2} < c_{sv_1} + c_{sv_2}$, this means that the path P' is not optimal, leading to a contradiction. We say that the node v_2 can be *excluded* by node v_1, for the given updated RREQ.

The goal of the next part of this thesis is to give a mathematical framework to define so-called *exclusion rules*, that can be used to determine to which neighbour we do not want to send an updated RREQ. Furthermore, this framework also allows us to prove results with respect to the optimal usage of the neighbour information, i.e., to find as many exclusion rules as possible, while still guaranteeing that a set of optimal paths always arrives at the destination. We will also give upper bounds on the number of RREQs, that will be sent through the network, and analyse different heuristics for finding an optimal set of multiple disjoint paths.

In the third part of the thesis, we will give a (short) high-level discussion on the structural differences between the proposed method and existing methods (DSR-based), and comment on the applicability of the patent.

Part II

Mathematical analysis

Chapter 4

Problem description

In this section we will introduce graph theoretical notation and explain the problem we want to solve. We will give a general mathematical framework to use so-called *exclusion rules*. The purpose of these exclusion rules is to determine how local information around a path can help decide whether or not it can be extended to a path contained in an optimal solution of one or more disjoint paths. Furthermore, we will use the mathematical framework to prove optimality results, i.e., for a given set of characteristics of the graph G and cost function c, we want to find as much exclusion rules as possible, while still guaranteeing that we always find an optimal solution for every instance satisfying the given characteristics.

4.1 Notation

In this section we will introduce the graph theoretical notation (see also Example 4.2.1). Let $G = (V, E)$ be an *undirected graph*, where $V = \{v_i : i \in I \subset \mathbb{N}\}$ is a finite set, and

$$E \subseteq \{U \subseteq V : |U| = 2\}.$$

Let $c : E \to \mathbb{R}$ be a cost function on the edges of G, where we use the notation

$$c_{uv} = c(\{u, v\})$$

for $\{u, v\} \in E$. For $u \in V$ we define

$$\mathcal{N}(u) = \{v : \{u, v\} \in E\}$$

and say that v is a *neighbour* of u if $v \in \mathcal{N}(u)$. We say that $P = p_0 p_1 \dots p_l$ is a *simple path* in G if

$$V(P) = \{p_0, \dots, p_l\} \subseteq V \quad \text{and} \quad \{p_{i-1}, p_i\} \in E \text{ for } i = 1, \dots, l$$

with all the p_i distinct. Also, we then define $E(P) = \{\{p_{i-1}, p_i\} : i = 1, \dots, l\}$. We say that P is an a, b-path if $p_0 = a$ and $p_l = b$. Furthermore, for $p_i, p_j \in V(P)$ we write

$$Pp_i = p_0 p_1 \dots p_i,\ p_i P = p_i \dots p_l,\ p_i P p_j = p_i \dots p_j$$

and for $u \in \mathcal{N}(p_l) \setminus V(P)$, we write $Pu = p_0 p_1 \dots p_l u$. Note that Pv is the *restriction* to v if $v \in V(P)$ and the *extension* of the path P to v otherwise. We denote

$$\mathcal{P}_G = \{P : P \text{ is a simple path in } G\},\ \ \mathcal{P}_{G,s} = \{P : P \text{ is a simple path in } G \text{ starting at } s\} \subseteq \mathcal{P}_G$$

and

$$\mathcal{P}_{G,s,d} = \{P : P \text{ is a simple } s, d\text{-path in } G\} \subseteq \mathcal{P}_G.$$

We say that two simple paths $P = sp_1 \dots p_l d$ and $Q = sq_1 \dots q_{l'} d$ are *node-disjoint s, d-paths* if $V(P) \cap V(Q) = \{s, d\}$. We say that k simple paths are node-disjoint s, d-paths, if they are pairwise node-disjoint s, d-paths.

We say that two simple paths $P = sp_1 \dots p_l d$ and $Q = sq_1 \dots q_{l'} d$ are *edge-disjoint s, d-paths* if $E(P) \cap E(Q) = \emptyset$. We say that k simple paths are edge-disjoint s, d-paths, if they are pairwise edge-disjoint s, d-paths.

Definition 4.1.1. *A branching tree is a directed graph (see Section 2.1) $T = (W, A)$ with a root $r \in W$, so that for every $w \in W$ there exists a unique path from r to w in T. We call $\{r\}$ the node of level zero and for every other $w \in W$, we say that w lies on level k if the (unique) path from r to w has length k. We denote all the nodes on level k with the set $L_{T,k}$. Furthermore, we say that a node $w \in L_{T,k}$ branches into b nodes of level $k+1$, if w lies on the unique path from r to x for precisely b nodes $x \in L_{T,k+1}$.*

We say that two branching trees $T = (W, A)$ and $T' = (W', A')$ are isomorphic if there exists a bijection $f : W \to W'$ with the property that $(x, y) \in A \Leftrightarrow (f(x), f(y)) \in A'$. In particular, this means that the root of T is mapped to the root of T'.

4.2 Mathematical framework

We define

$$c : \mathcal{P}_G^k \to \mathbb{R}$$

as a cost function on all tuples of k simple paths in G. It will follow from the context whether we mean the cost function for the edges or the paths. The goal here is to solve the problem

$$\min \left\{ c(P_1, \dots, P_k) \ : \ (P_1, \dots, P_k) \in \mathcal{P}_{G,s,d}^k \text{ and } P_1, \dots, P_k \text{ are node-disjoint } s, d\text{-paths} \right\}. \quad (4.1)$$

We can solve (4.1) by enumerating over all simple s, d-paths and then search for k paths that minimize the objective function. In order to enumerate over all s, d-paths, we construct the following branching tree $T = (W, A)$. We take

$$\mathcal{P}_{G,s,d} \subseteq W = \{P : P \text{ a simple path } s, v\text{-path in } G \text{ and } d \in V(P) \Leftrightarrow v = d\} \subseteq \mathcal{P}_{G,s}$$

and

$$A = \{(P, Pv) : P, Pv \in W \text{ with } v \notin V(P)\}.$$

Note that $P \in W$ if and only if it is an s, d-path or a path starting at s that does not contain d. The tree T can then be constructed recursively by letting a path P *branch* into all the paths Pv for $v \in \mathcal{N}(p_l) \setminus V(P)$, where we start with $P = s$ (the root of the tree).

Example 4.2.1. Consider the graph and cost function in Figure 4.1 and let $k = 2$.

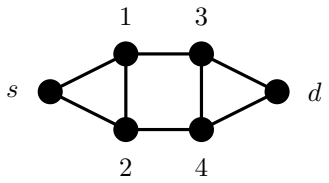

Figure 4.1: Example of a graph with $|V| = 6$, $|E| = 8$ and $c_{ij} = 1$ for all $\{i, j\} \in E$.

Furthermore, we take

$$c(P, Q) = \sum_{i=1}^{r} c_{p_{i-1}p_i} + \sum_{j=1}^{t} c_{q_{i-1}q_i}$$

for paths $P = p_0 \dots p_r$ and $Q = q_1 \dots q_t$. The branching tree is given in Figure 4.2. We can see that there are eight s, d-paths, and by inspection it follows that the optimal solution is given by sv_1v_3d and sv_2v_4d.

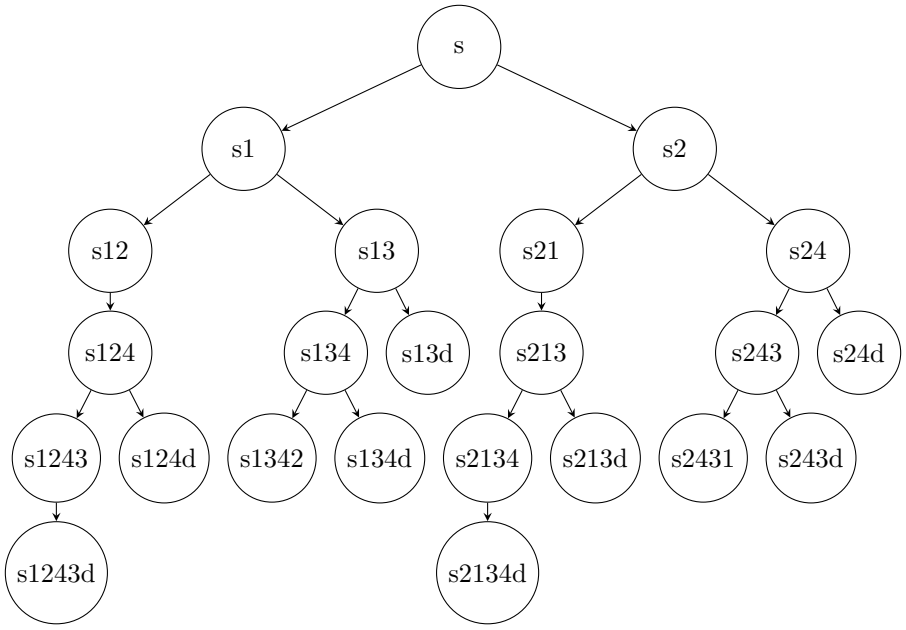

Figure 4.2: Branching tree for example in Figure 4.1.

□

We now want to construct a subtree $T' = (W', A')$ of T with the same root s and the property that at least one optimal solution can be formed from the nodes of W'. Because the tree T' is smaller, it is easier to determine the optimal solution. In order to get a smaller tree, we want to reduce the set

$$B(P) \subseteq \{v : Pv \in W\},$$

i.e., the number of paths we branch into for a path P. This is equivalent to saying that the subtree with root Pv will be removed for the v that are excluded from the set $B(P)$. In deciding whether or not to exclude a certain neighbour, we only allow partial information of the graph G and cost function c to be used. To this end, we will make the following definition.

Definition 4.2.1. *An information mapping ϕ is a function that maps every tuple (s, d, P, G, c) to a subgraph of $G = (V, E)$, i.e.,*

$$\phi((s, d, P, G, c)) = G_{P,\phi}$$

where $P = s \dots p_l \in \mathcal{P}_{G,s}$, c a cost function on E and $d \in V$. Furthermore $G_{P,\phi} = G_P = (V_P, E_P)$ must satisfy the following conditions:

$$V(P) \cup \mathcal{N}(p_l) \cup \{d\} \subseteq V_P \quad \textit{and} \quad E(P) \cup \{\{p_l, q\} : q \in \mathcal{N}(p_l)\} \subseteq E_P.$$

We denote the trivial mapping as the mapping defined by

$$V_P = V(P) \cup \mathcal{N}(p_l) \cup \{d\} \textit{ and } E_P = E(P) \cup \{\{p_l, q\} : q \in \mathcal{N}(p_l)\}.$$

Furthermore, we denote c_P as the restriction of c to G_P.

Informally speaking, G_P is a subgraph of G that contains the path P, neighbours of p_l, and some additional information that may be used to determine whether or not to exclude a neighbour. Note

that G and c are arbitrary in this definition. The idea of only allowing partial information of G to be used, follows from existing routing techniques, where components in a communication network only know who their direct neighbours are (see first part of this thesis). A path (starting at a fixed source s) arrives at an intermediate node, that then has to decide whether or not to extend this path to some of its neighbours. Along the way, we can include information about neighbouring nodes in the message containing the path, that can be used to make this decision. This is the reason why we are mostly interested in the information mapping explained in Example 4.2.2.

Example 4.2.2. An example of the mapping ϕ, that we are interested in, is $G_P = (V_P, E_P)$ defined by

$$V_P = \{d\} \cup \bigcup_{y \in V(P)} \mathcal{N}(y) \quad \text{and} \quad E_P = \bigcup_{y \in V(P)} \{\{x, y\} : x \in \mathcal{N}(y)\}.$$

For the graph in Example 4.1, the graph G_P for $P = sv_2v_4$ is given in Figure 4.3.

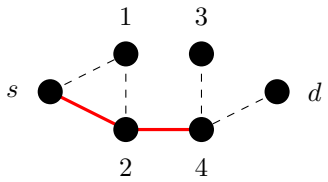

Figure 4.3: The graph G_P for $P = sv_2v_4$ and G as in Example 4.1.

Note that G_P might be disconnected (for example for $P = sv_2$, since d has no neighbours in that graph). Roughly speaking, the graph G_P contains all the edges for which at least one node lies on the path P. We will refer to this mapping as the *neighbour mapping* $\phi_{\mathcal{N}}$. □

The decision of which neighbours are to be excluded, will be formulated as a set of so-called exclusion rules. Note that ϕ is not used to define these rules and, again, that G and c are arbitrary in this definition.

Definition 4.2.2. *An exclusion rule R is a function that maps every tuple (s, d, P, G, c) to a set*

$$X = X_{(s,d,P,G,c,R)} \subseteq \mathcal{N}(p_l).$$

Here we have, $P = s \dots p_l \in \mathcal{P}_{G,s}$, c a cost function on E and $d \in V$. A set of exclusion rules $\mathcal{R} = \mathcal{R}(R_1, \dots, R_r)$ is defined as the function that maps (s, d, P, G, c) to the union of the subsets $X_1, \dots, X_r$. We call a set of exclusion rules good if

$$\mathcal{N}(p_l) \cap V(P) \subseteq \bigcup_{i=1,\dots,r} X_{(s,d,P,G,c,R_i)}.$$

We say that two sets of exclusion rules $\mathcal{R}(R_1, \dots, R_r)$ and $\mathcal{R}'(R'_1, \dots, R'_{r'})$ are equivalent if

$$\bigcup_{i=1,\dots,r} X_i = \bigcup_{i=1,\dots,r'} X'_i$$

for every (s, d, P, G, c).

Unless stated otherwise, we always assume that a set of exclusion rules is good. Note that a good set of exclusion rules guarantees that the extensions we *do* make, will lead to simple paths (since all the nodes that are already on the path will be excluded). For a fixed graph G, we want to apply exclusion rules to the the tuples (s, d, P, G_P, c_P) for a fixed mapping ϕ. Here, c_P is the restriction of c to G_P. We will now give some more definitions.

Definition 4.2.3. *Let ϕ, G, c, s, d and $\mathcal{R}$ be fixed. We say that a path $P = p_0 p_1 \dots p_l \in \mathcal{P}_{G,s}$ is being extended to v if*

$$v \in \mathcal{N}(p_l) \setminus X = X_{(s,d,P,G_P,c_P,\mathcal{R})}.$$

Furthermore, we say that a path P arrives at p_l if Pp_i is being extended to p_{i+1} for $i = 0, \dots, l-1$. If we say that the path P arrives, we mean that P arrives at p_l.

Definition 4.2.3 states that a path P arrives, if it can be found in the subtree T' (as defined earlier in this section).

Definition 4.2.4. *Let $\phi, \mathcal{R}$ be fixed and let $\mathcal{G}$ be a collection of tuples (G, c, s, d) where $G = (V, E)$, c a cost function on E and $s, d \in V$. We assume that, for all tuples, there exists a solution to (4.1). We say that the set of exclusion rules $\mathcal{R}$ is feasible for $\mathcal{G}$ if for every $(G, c, s, d) \in \mathcal{G}$, there arrive k simple s, d-paths at d that form a solution to (4.1). We say that the set of exclusion rules $\mathcal{R}$ is feasible for a collection $\mathcal{G}'$ of pairs (G, c) if it is feasible for the collection*

$$\mathcal{G} = \bigcup_{(G,c)\in\mathcal{G}'} \{(G, c, s, d) : s, d \in V(G) \text{ and there exists a solution to (4.1)}\}.$$

It will follow from the context if we mean a collection of tuples or pairs. Example 4.2.3 illustrates the concept of feasibility.

Example 4.2.3. Suppose $\mathcal{G}$ is a collection of pairs (G, c) where $G = (V, E)$ has a spanning tree T with the property

$$\begin{cases} c_{uv} = 0 & \{u, v\} \in E(T) \\ c_{uv} > 0 & \{u, v\} \notin E(T) \end{cases}$$

and that we use the trivial mapping ϕ. Consider the following set of exclusion rules:

$$X = \{v \in \mathcal{N}(u) \setminus \{p_l\} : c_{uv} > 0\} \cup (\mathcal{N}(u) \cap V(P))$$

For this collection of $\mathcal{G}$, a shortest s, d-path will arrive at d for every (G, c, s, d) and hence, for $k = 1$ in (4.1), the set is feasible for $\mathcal{G}$. However, it is easy to think of collections where this exclusion rule is not feasible. □

Definition 4.2.5. *Let ϕ, G, c, s, d and $\mathcal{R}$ be fixed. We define $T = T(G, c, s, d, \mathcal{R}, \phi) = (W, A)$ as the branching tree where*

$$W = \{P : P \in \mathcal{P}_{G,s} \text{ arrives}\}.$$

For a path $P = s \dots p_l \in \mathcal{P}_{G,s}$ and $v \notin V(P)$, we have

$$\big(P, Pv\big) \in A \quad \Leftrightarrow \quad v \in \mathcal{N}(p_l) \setminus X_{(s,d,P,G_P,c_P,\mathcal{R})},$$

i.e., if P is extended to v, which is equivalent to saying that the path Pv arrives at v (see Definition 4.2.3). Note that the $L_{T,k}$ (see Definition 4.1.1) contains all arriving paths of length k and that s (as a path of length zero) is the root of the tree. Furthermore, if $\big(P, Pv\big) \notin A$, we say that P excludes the path Pv (for $v \in \mathcal{N}(p_l)$).

For example, the exclusion rule that takes $X = V(P) \cap \mathcal{N}(p_l)$ gives the branching tree in Figure 4.2 for (G, c, s, d) as in Figure 4.1. We will now give the definition of optimality for a set of exclusion rules.

Definition 4.2.6. *Let ϕ be a fixed mapping, $\mathcal{G}$ a collection of tuples (G, c, s, d) and let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two sets of exclusion rules that are feasible for $\mathcal{G}$. We say that*

$$\mathcal{R}_1 \preceq \mathcal{R}_2 \quad \Leftrightarrow \quad |A(T(G, c, s, d, \mathcal{R}_1, \phi))| \leq |A(T(G, c, s, d, \mathcal{R}_2, \phi))|$$

for every $(G, c, s, d) \in \mathcal{G}$. A set of exclusion rules $\mathcal{R}$ is optimal for the collection $\mathcal{G}$, if $\mathcal{R} \preceq \mathcal{R}'$ for every set of exclusion rules $\mathcal{R}'$ that is feasible for $\mathcal{G}$.

Note that this gives a partial ordering on the sets of exclusion rules. It is not guaranteed that a minimal element exists for every collection $\mathcal{G}$. For example, it is in general not true that the union of two sets of exclusion rules is again a feasible set. Also note that we can use the cardinality of the set W instead of A, since we have $|W| = |A| + 1$ for a directed tree $T = (W, A)$. In some situations (in the next section) we use the following assumption on a set of exclusion rules.

Assumptions 4.2.1. *Let $\mathcal{R}$ be a set of exclusion rules. If we have two tuples (s, d, P, G, c) and (s', d', P', G', c') - satisfying Definition 4.2.2 - that are isomorphic, then we must have*

$$f(X_{(s,d,P,G,c,\mathcal{R})}) = X_{(s',d',P',G',c',\mathcal{R})}.$$

Here, an isomorphism between two tuples is defined as a bijection $f : V \to V'$ so that $\{x, y\} \in E \Leftrightarrow \{f(x), f(y)\} \in E'$, $f(p_i) = p'_i$ for $i = 0, \ldots, k$ (where $P = sp_1, \ldots, p_k$ and $P' = sp'_1, \ldots, p'_k$) and $f(d) = d'$. Furthermore, the cost functions c, c' must satisfy $c(\{x, y\}) = c'(\{f(x), f(y)\})$.

For example, the tuples for the paths $P = sv_2v_4$ and $P = sv_1v_3$ are isomorphic, in Example 4.2.2. Assumption 4.2.1 is made so that the exclusion rules can only use the structure of the graph and not the names of the nodes (we assume that V is always given by $v_1, v_2, \ldots, v_n$). We do not want to get a different branching tree if we would interchange names of nodes. An example of an exclusion rule, that does not satisfy Assumption 4.2.1, is given below.

Example 4.2.4. An example of an exclusion rule that does not satisfy Assumption 4.2.1 is the following: path $P = s \ldots v_i$ excludes a path Pv_j for $v_j \in \mathcal{N}(v_i) \setminus V(P)$ if $j > i$.

Chapter 5

Single path problem

In this section we will introduce exclusion rules that we can use for the single path problem, i.e., the minimization problem for $k = 1$ in (4.1). We will give optimal sets of exclusion rules for some non-trivial collections $\mathcal{G}$. We will also explain the reason for Assumption 4.2.1, and analyse the mathematical framework with and without this assumption. Furthermore, we will explain the concept of equivalent information mappings, i.e., given a set of exclusion rules $\mathcal{R}$ what information of G_P, defined by the information mapping ϕ, do we really need to apply the exclusion rules. Although we are more interested in the multipath problem, the single path problem serve as a good introduction to get a grip on the definitions introduced in the previous chapter. Furthermore, in Chapter 9, we will give some heuristic approaches for the multipath problem, based on the single path problem. Here, (4.1) reduces to

$$\min\{c(P) \ : \ P \in \mathcal{P}_{G,s,d}\}, \tag{5.1}$$

where

$$c(P) = \sum_{i=1}^{l} c(\{p_{i-1}, p_i\}) = \sum_{i=1}^{l} c_{p_{i-1}p_i}.$$

The goal is to find optimal sets of exclusion rules for some non-trivial collections $\mathcal{G}$. We will describe a number of exclusion rules and illustrate these using the mapping ϕ as defined in Example 4.2.2: the neighbour mapping $\phi_{\mathcal{N}}$. In the first place, we are interested in non-negative cost functions $c : E \to \mathbb{R}_{\geq 0}$. Therefore, the exclusion rules will be defined in such a way that they are feasible for such a cost function (although they can be applied for arbitrary cost functions, but might not be feasible in that case). We will come back to more general cost functions in a later section, where we analyse the problem for cost functions c that have no negative cost cycles. For the non-negative case, the feasibility of most of the exclusion rules is based on Lemma 5.0.2.

Lemma 5.0.2. *Let $G = (V, E)$ be a graph and $s, d \in V$. Let $P = sv_1 \dots v_{k-1}d$ be a shortest s, d-path. Then $sv_1 \dots v_l$ is a shortest s, v_l-path for $1 \leq l \leq k-1$.* □

5.1 Exclusion rules

The first exclusion rule, the loop free rule, is the trivial exclusion rule. It is named this way because the path Pv contains a loop for $v \in \mathcal{N}(p_l) \cap V(P)$.

Exclusion Rule 5.1.1 (Loop free rule)**.** This is the exclusion rule that maps (s, d, P, G, c) to

$$X = \mathcal{N}(p_l) \cap V(P).$$

□

The second exclusion rule excludes a neighbour $w \in \mathcal{N}(p_l) \setminus V(P)$ if there exists a path $P' = s \dots w$ in G with the property that $c(P') \leq c(Pw)$. One way to find such a path is by looking for a path $P' = Pxw$, for some $x \in V(P)$. In general, there are more ways to find P' (one extreme case would be to run a central shortest path algorithm in the G). The reason for this formulation is that it can also be used in the multipath case (which is not the case for all formulations).

Exclusion Rule 5.1.2 (Unnecessary Node rule)**.** This is the exclusion rule that maps (s, d, P, G, c) to

$$X = \{w : w \in \mathcal{N}(p_l) \setminus V(P) \text{ and there is } x \in V(P) \setminus \{p_l\} \text{ so that } \{x, w\} \in E \text{ and } c(Pxw) \leq c(Pw)\}.$$

Figure 5.1 illustrates the situation. Note that $c(Pxw) \leq c(Pw)$ is equivalent to $c_{xw} \leq c(xP) + c_{p_l w}$.

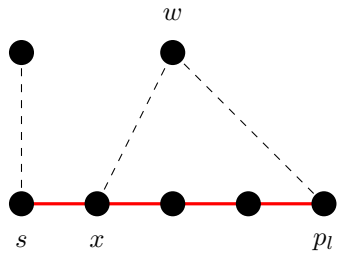

Figure 5.1: Sketch of G where there might be more nodes on the red edges (i.e., the path P).

□

The third exclusion rule excludes a neighbour $w \in \mathcal{N}(p_l) \setminus V(P)$ if there exists a path $P' = s \dots d$ in G with the property that $c(P') \leq c(Pw)$. Again, one way to find such a path is by looking for an $x \in V(P)$ such that $c(Pxd) \leq c(P')$.

Exclusion Rule 5.1.3 (Destination rule)**.** This is the exclusion rule that maps (s, d, P, G, c) to

$$X = \{w : w \in \mathcal{N}(p_l) \setminus V(P) \text{ and there is } x \in V(P) \text{ so that } \{x, d\} \in E \text{ and } c(Pxd) \leq c(Pw)\}.$$

Figure 5.2 illustrates the situation. Note that this rule is equivalent to the unnecessary node rule in the case that $w = d$. The definition then has to be adjusted by replacing '$x \in V(P)$' with '$x \in V(P) \setminus \{p_l\}$'.

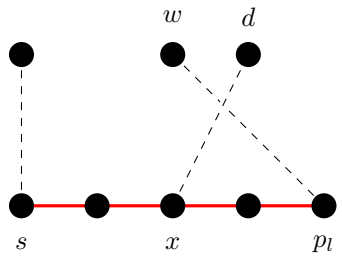

Figure 5.2: Sketch of G where the red edges might contain more nodes.

□

Example 5.1.1. In Figure 5.3, we have given the branching tree for the set of exclusion rules consisting of the loop free, unnecessary node and destination rule. For example, the path sv_1v_2 is excluded based on the unnecessary node rule and the path $sv_1v_3v_4$ is excluded based on the destination rule.

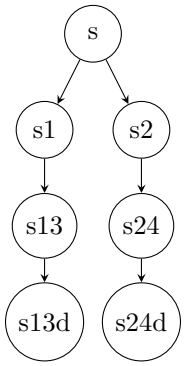

Figure 5.3: Branching tree for example in Figure 4.1 with Exclusion Rules 5.1.1, 5.1.2 and 5.1.3.

□

The fourth exclusion rule is based on the fact that there is a path $P' = s \dots p_l$ in G with the property that $c(P') < c(P)$. The difference with the formulation of the unnecessary node and destination rule is that there is already a shorter path to the end node of P, instead of a shorter path to a neighbour.

Exclusion Rule 5.1.4 (Shortcut rule (inequality))**.** This is the exclusion rule that maps (s, d, P, G, c) to

$$X = \begin{cases} \mathcal{N}(p_l) & \text{If there is a strict shortcut for } P \\ \emptyset & \text{Otherwise} \end{cases}$$

We say that $w \notin V(P)$ forms a strict shortcut for the path P if there exists an $w_1 \in V(P)$ such that $c_{w_1 w} + c_{w p_l} < c(w_1 P)$. Figure 5.4 illustrates the situation.

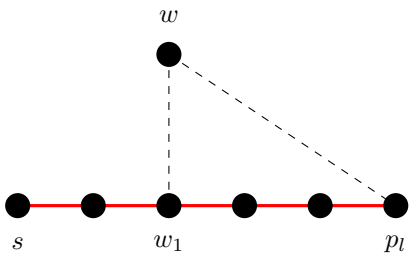

Figure 5.4: Sketch of G where the red edges might contain more nodes

□

For $x \in \mathcal{N}(p_l) \setminus \{w\}$, the feasibility follows from Lemma 5.0.2. Regarding w, using the non-negativity of $c_{w w_2}$, we have

$$c_{w_1 w} \leq c_{w_1 w} + c_{w p_l} < c(w_1 P) \leq c(w_1 P) + c_{w p_l}$$

which means that we can also exclude w, based on the unnecessary node rule. Also note that if we cannot exclude w based on the unnecessary node rule, then we can never have a shortcut via w. For arbitrary c, the above reasoning does not have to hold, but we will come back to this later. The benefit of being able to define this exclusion rule as above will follow from the implementation of the distributed algorithm in Chapter 9.

The fifth exclusion rule also treats the case of a shortcut, as described above, but with $c_{w p_l} + c_{w_1 w} = c(w_1 P)$. This rule excludes the whole neighbourhood if there exists another path P' to p_l in G that has $c(P) = c(P')$, but with less nodes, i.e. $|V(P')| < |V(P)|$.

Exclusion Rule 5.1.5 (Shortcut rule (equality))**.** This is the exclusion rule that maps (s, d, P, G, c) to

$$X = \begin{cases} \mathcal{N}(p_l) & \text{If there is an equal shortcut for } P \\ \emptyset & \text{Otherwise.} \end{cases}$$

We say that $w \notin V(P)$ forms an equal shortcut for the path P if there exists an $w_1 \in V(P)$ such that $c_{w_1 w} + c_{w p_l} = c(w_1 P)$ and $|V(w_1 P)| > 3$, i.e., there lie at least two nodes between w_1 and p_l on P). □

Intuitively, this means that we prefer a shortest path that also use a minimum number of nodes. Note that it could also be possible to give a preference to paths using a maximum number of nodes.

Example 5.1.2. The reason we cannot choose $|V(w_1 P) > 2|$, i.e., allow one node between w_1 and p_l, is the following. Consider the graph in Figure 5.5, where we take $c_{uv} = 1$ for all $\{u, v\} \in E$, and the neighbour mapping $\phi_{\mathcal{N}}$ as information mapping.

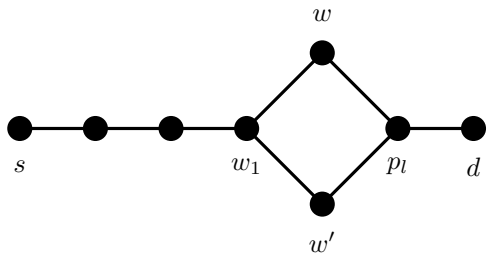

Figure 5.5: Graph G with c a constant cost function.

Note that G_P and $G_{P'}$ are isomorphic (in fact, they both equal the whole graph G) by using the bijection that interchanges w and w' (with $P = s \dots w_1 w p_l$ and $P' = s \dots w_1 w' p_l$). If we would allow $|V(w_1 P) = 3|$ in Exclusion Rule 5.1.5, then both paths P and P' would be excluded which means that no (shortest) path arrives at d. □

Intuitively, we do not want to exclude a path based on the fact that we see another path with the same cost and length. Otherwise, we could get a sort of deadlock effect where there is a series of paths $P_1, \dots, P_q$ so that path P_i is excluded based on path P_{i+1}, for $i = 1, \dots, q-1$, and path P_q based on P_1. However, it is possible to define a partial ordering on the set $\mathcal{P}_{G,s}$, using the names of the nodes (see Section 5.3), but these kind of rules do not satisfy Assumption 4.2.1. From now on, we will refer to the shortcut rule as the union of Exclusion Rules 5.1.4 and 5.1.5.

5.2 Optimal set of exclusion rules

In this section we will show that there exist collections $\mathcal{G}$ so that Exclusion Rules 5.1.1-5.1.5 form an optimal set for the neighbour mapping $\phi_{\mathcal{N}}$. Note that optimality is always concerned with finding the best set of exclusion rules $\mathcal{R}$ for a fixed collection $\mathcal{G}$ and information mapping ϕ.

Definition 5.2.1. *Let ϕ be a fixed mapping, $\mathcal{G}$ a collection of tuples (G, c, s, d) and let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two sets of exclusion rules that are feasible for $\mathcal{G}$. We say that*

$$\mathcal{R}_1 \preceq \mathcal{R}_2 \quad \Leftrightarrow \quad |A(T(G, c, s, d, \mathcal{R}_1, \phi))| \leq |A(T(G, c, s, d, \mathcal{R}_2, \phi))|$$

for every $(G, c, s, d) \in \mathcal{G}$. A set of exclusion rules $\mathcal{R}$ is optimal for the collection $\mathcal{G}$, if $\mathcal{R} \preceq \mathcal{R}'$ for every set of exclusion rules $\mathcal{R}'$ that is feasible for $\mathcal{G}$.

We will now show that the Exclusion Rules 5.1.1-5.1.5 form an optimal set for the collection of pairs (G, c) for which G only has cycles of length five or larger, and where c is a non-negative cost

function. These exclusion rules can be summarized by the statement that we exclude a neighbour w if we already see a shorter path to w than Pw, or to the destination d, in G_P. Here a path is shorter if its cost if lower, or in case of equal costs, if it uses less nodes.

Theorem 5.2.1. *Let ϕ be the neighbour mapping and let $\mathcal{G}$ be the collection of pairs where every pair (G, c) satisfies the following conditions. The graph $G = (V, E)$ does not contain cycles of length smaller than five, i.e. $\varrho(G) \geq 5$ (girth of G), and the cost function c is non-negative. Then the set $\mathcal{R}$, consisting of the loop free, unnecessary node, destination and shortcut rule, is optimal for $\mathcal{G}$.*

Proof. The feasibility of $\mathcal{R}$ follows from the the fact that every shortest s, d-path, that also uses the least number of nodes for a shortest path, arrives at d. Suppose that there is some other set $\mathcal{R}'$ and a pair $(G, c) \in \mathcal{G}$ such that $|A(T(G, c, s, d, \mathcal{R}', \phi))| < |A(T(G, c, s, d, \mathcal{R}, \phi))|$ for some $s, d \in V(G)$. This means that there exists a node v, a path $P = s \dots v$ arriving at v and a neighbour $w \in \mathcal{N}(v)$ so that $w \in X_{(s,d,P,G_P,c_P,\mathcal{R}')}$ but $w \notin X_{(s,d,P,G_P,c_P,\mathcal{R})}$. The fact that $w \notin X_{(s,d,P,G_P,c_P,\mathcal{R})}$ tells us that Pw is the unique shortest path from s to w in G_P (by definition of the exclusion rules in $\mathcal{R}$). This follows from the fact we do not have any cycles of length three or four, which means we have no shortcuts with inequality nor equality. If we would have cycles of length three or four, then Pw would not have to be unique. For the same reason, P is the unique shortest path from s to v in G_P.

Case 1: $w = d$**.** Note that the pair $(G_P, c_P) \in \mathcal{G}$, i.e., G_P as the whole graph. Since $Pw = Pd$ is the unique shortest path to d, this means that $\mathcal{R}'$ is not feasible, since for the tuple (G_P, c_P, s, d) the optimal solution now does not arrive at d. To be more precise, for a path $P = sp_1p_2...v$ we have that G_{Pp_i} is a subgraph of G_{Pp_j} whenever $i \leq j$, which means that P will arrive at v, but then w will be excluded.

Case 2: $w \neq d$**, but** $d \in G_P$**.** This means that

$$c(Pw) < c(P')$$

for every path s, d-path P' in G_P due to the destination rule. Let H be the graph that consists of G_P and the edge $\{d, w\}$ (with cost zero). Then the path Pwd is the unique shortest path from s to d in H, but using the same reasoning as in the first case, this means that $\mathcal{R}'$ is not feasible. Note that H might contain a cycle of length smaller than five. This problem can be solved by adding extra nodes on the edge $\{d, w\}$ and giving the resulting edges all cost zero. Then we have $(H, c_H, s, d) \in \mathcal{G}$.

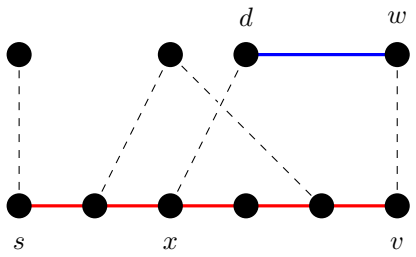

Figure 5.6: The graph H in Case 2.

Case 3: $w \neq d$ **and** $d \notin G_P$**.** Then we can make the graph H consisting of G_P together with $\{d, w\}$ where we can choose $c_{dw} = 0$ (see Figure 5.7).

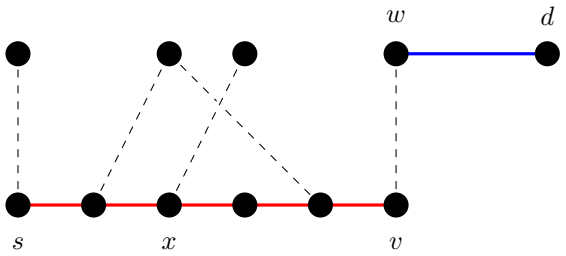

Figure 5.7: The graph H in Case 3.

Since d is only connected to w, a path P' is a shortest s,d-path if and only if $P'w$ is a shortest s,w-path. Furthermore, Pw is the unique shortest path from s to w, so if path P excludes Pw using $\mathcal{R}'$, then $\mathcal{R}'$ is not feasible for this tuple (H, c_H, s, d). □

Note that the techniques of the proof consist of finding a pair $(H, c, s, d) \in \mathcal{G}$ that leads to the infeasibility of $\mathcal{R}'$. Intuitively (in this case), we can only exclude a path if we know for certain that it can never be extended to a shortest s,d-path (based on the information from G_P and c_P). If we assume that c is a constant non-negative cost function, we can make the following statement.

Theorem 5.2.2. *Let ϕ be the neighbour mapping and let $\mathcal{G}$ be the collection of pairs (G, c) satisfying the following conditions. The graph $G = (V, E)$ does not contain cycles of length smaller than four, i.e. $\varrho(G) \geq 4$ (girth of G), and the cost function c is non-negative and constant. Then the set $\mathcal{R}$, consisting of the loop free, unnecessary node, destination and shortcut rule, is the unique optimal set of exclusion rules (under Assumption 4.2.1).*

Proof. Without loss of generality, we can assume that $c = 1$. The feasibility of $\mathcal{R}$ again follows from the fact that every shortest s,d-path, that also uses the least number of nodes for a shortest path, arrives at d. Suppose that there is some other set $\mathcal{R}'$ and a pair $(G, c) \in \mathcal{G}$ such that $|A(T(G, c, s, d, \mathcal{R}', \phi))| < |A(T(G, c, s, d, \mathcal{R}, \phi))|$ for some $s, d \in V(G)$. This means that there exists a node v, a path $P = s \ldots v$ arriving at v and a neighbour $w \in \mathcal{N}(v)$ so that $w \in X_{(s,d,P,G_P,c_P,\mathcal{R}')}$ but $w \notin X_{(s,d,P,G_P,c_P,\mathcal{R})}$.

We will now first introduce some notation. If $x \in V_P \setminus V(P)$, such that $|\mathcal{N}(x)| = 1$ in G_P, then we define x' as the node in $\mathcal{N}(x)$. If $x \in V_P \setminus V(P)$ such that $|\mathcal{N}(x)| = |\{x_1, x_2\}| = 2$ in G_P, then we define x' as the node that lies between x_1 and x_2 on P. This node is well-defined, because if there would lie more than one node between x_1 and x_2 then we would have a strict shortcut which means that P can never have arrived at v, or, in the case that $x_2 = v$, we would not extend P to w (using the shortcut rule from $\mathcal{R}$). Furthermore, if x_1 and x_2 would be successors on P, then we would have a cycle of length three, but we do not have those by assumption.

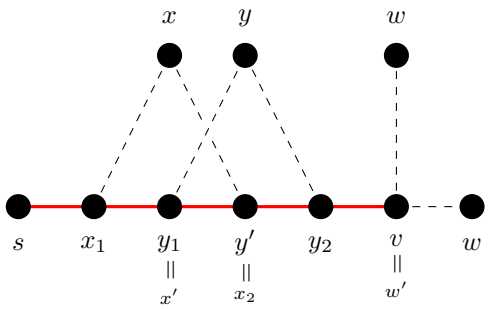

Figure 5.8: Example for notation of shortcuts.

We say that two shortcuts via x and y are similar if ($x_1 = y_1$ and $x_2 = y_2$), overlapping if ($x_1 = y'$ or $x_2 = y'$), and disjoint otherwise. Note that $x_1 = y'$ implies $y_2 = x'$. For $z \in V(P)$ we

define

$$\mathcal{N}_1(z) = \{w \in V_P \setminus V(P) \ : \ \mathcal{N}(w) = \{z\}, \ \text{ i.e., } z = w'\}$$

and $\mathcal{N}_1^*(z)$ as the set that contains a copy w^* for every $w \in \mathcal{N}_1(z)$. Furthermore, we define

$$\mathcal{S}(P) = \{x \in V_P \setminus V(P) : |\mathcal{N}(x)| = 2 \text{ in } G_P\}$$

and

$$E(\mathcal{S}(P)) = \{\{x, y\} : x, y \in \mathcal{S}(P) \text{ form an overlapping shortcut}\}.$$

Case 1: $w \neq d$**.** Because every edge has the same cost, d cannot have positive degree in G_P. Otherwise P can never have arrived at v due to the destination rule or, in the case that d would be connected to v, we would exclude w, also based on the destination rule in $\mathcal{R}$. We will embed G_P in the graph H defined by

$$V(H) = V(G_P) \bigcup_{x \in \mathcal{S}(P)} \mathcal{N}_1(x, x')$$

where $\mathcal{N}_1(x, x')$ is a unique copy of $\mathcal{N}_1^*(x')$ (since it might happen that $x' = y'$ for distinct $x, y \in \mathcal{S}(P)$). Furthermore, we have

$$E(H) = \{w, d\} \cup E(G_P) \cup E(\mathcal{S}(P)) \bigcup_{x \in \mathcal{S}(P)} \{\{x, w\} : w \in \mathcal{N}_1(x, x')\}.$$

For example, G_P as in Figure 5.9 would be embedded in the graph H in Figure 5.10.

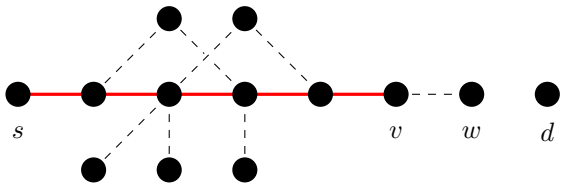

Figure 5.9: Example of G_P.

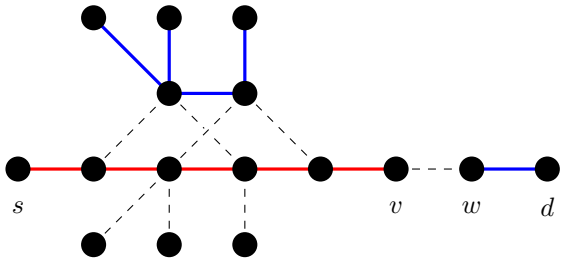

Figure 5.10: Graph H for Case 1.

Note that this embedding has no cycles of length three by construction, so together with the same constant cost function, we have $(H, c, s, d) \in \mathcal{G}$. The essence of the proof is that for every path P' that arrives at v in this embedding, and possibly could be extended to a shortest s, d-path, $(s, d, P', G_{P'}, c_{P'})$ is isomorph with (s, d, P, G_P, c_P). But then based on Assumption 4.2.1, the set $\mathcal{R}'$ will not send these paths P' to w either and so a shortest s, d-path will never arrive at d, meaning that $\mathcal{R}'$ is not feasible for $(H, c, s, d) \in \mathcal{G}$. The claim that $(s, d, P', G_{P'}, c_{P'})$ is isomorph with (s, d, P, G_P, c_P) is true by construction, since every path P' either uses a node $x \in V(P)$ or a node $y \in V_P \setminus V(P)$ for which $y' = x$ (if P' can lead to a shortest s, d-path).

Case 2: $w = d$**.** The same proof as in Case 1 holds, but now we do not have to add the edge $\{w, d\}$.

The uniqueness (modulo equivalent sets of exclusion rules) follows using the same strategy. If there would be another (non-equivalent) optimal set of exclusion rules, then there must exist a tuple (G, c, s, d) for which the branching trees differ at some point. Then we can use the same arguments as above to show that this other set must be infeasible. □

Theorem 5.2.2 cannot be generalized to arbitrary non-negative cost functions. This can be seen using the Exclusion Rule 5.2.1 (which still satisfies Assumption 4.2.1). Furthermore, we can also not (directly) take G arbitrary, i.e., allowing cycles of length three, since the isomorphisms, as described in the proof, are then no longer guaranteed.

Exclusion Rule 5.2.1 (Shortcut rule (equality-breaking))**.** This is the exclusion rule that maps (s, d, P, G, c) to

$$X = \begin{cases} \mathcal{N}(p_l) & \text{If there is an equal-breaking shortcut for } P \\ \emptyset & \text{Otherwise.} \end{cases}$$

We say that $w \notin V(P)$ forms an equality-breaking shortcut for the path P, if there exists an $w_1 \in V(P)$ such that $c_{w_1 w} + c_{w p_l} = c(w_1 P)$, $|V(w_1 P)| = 2$, and $c_{w_1 w'} < c_{w_1 w}$. Here, w' is the node that lies between w_1 and p_l (which is well-defined because of $|V(w_1 P)| = 2$). See Figure 5.5 for a sketch of the situation. □

The reason for the feasibility of Exclusion Rule 5.2.1, together with Exclusion Rules 5.1.1-5.1.5, can be explained using a partial ordering on the set $\mathcal{P}_{G,s,d}$ of all simple s, d-paths (for a fixed cost function c). We say that $P \preceq P'$ if either

i) $c(P) < c(P')$,

ii) $c(P) = c(P')$ and $|V(P)| < |V(P')|$,

iii) $c(P) = c(P')$, $|V(P) = |V(P')|$, and

$$c(p_j p_{j+1}) < c(p'_j p'_{j+1}) \quad \text{where} \quad j = \text{argmin}_{i \in 1, \ldots |V(P)|} \{i : c(p_i p_{i+1}) \neq c(p'_i p'_{i+1})\},$$

with $P = p_0 p_1 \ldots p_l$ and $P' = p'_0 p'_1 \ldots p'_{l'}$. Then a shortest path $P \in \mathcal{P}_{G,s,d}$, that is also minimal under this partial ordering, will always arrive at d. Note that there are other choices for this partial ordering, e.g., by taking

$$c(p_j p_{j+1}) > c(p'_j p'_{j+1})$$

in condition iii). The fact that we can choose multiple partial orderings, makes it more difficult to prove optimality results (since there probably will be multiple optimal sets of exclusion rules, depending on the choice of partial ordering). This could be solved by just redefining the original problem, i.e., by adding the condition that the optimal path must be a minimal element of a partial ordering of the form above (since the solution to this new problem is also a solution to the original problem). The Exclusion Rules 5.1.1-5.2.1 can then be redefined as one rule that excludes a neighbour $w \in \mathcal{N}(p_l)$ if the graph G_P contains a s, w-path P' such that $P' \prec Pw$. If we do not redefine the original problem, we will need techniques other than the ones used in this section, to prove optimality results.

5.3 Tie breaking rule

We will introduce a new exclusion rule that deals with ties in case of an equal shortcut (see Exclusion Rule 5.1.5) for a constant cost function c. This shows that the optimality result of Theorem 5.2.2 does no longer hold if we drop Assumption 4.2.1. This rule can also be related to the discussion at the end of the previous section, since this rule leads to an ordering on the paths in $\mathcal{P}_{G,s,d}$. For this rule, we will define an ordering '$\preceq$' on the set $\{v_i : i \in \mathbb{N}\}$. Remember from Section 4.1 that we 'encode' V as a finite subset of the set $\{v_i : i \in \mathbb{N}\}$.

Exclusion Rule 5.3.1 (Tie breaking rule)**.** This is the exclusion rule that maps (s, d, P, G, c) to

$$X = \begin{cases} \mathcal{N}(p_l) & \text{If there is a tie breaking shortcut for } P \\ \emptyset & \text{Otherwise} \end{cases}$$

We say that $w \notin V(P)$ forms a tie-breaking shortcut for the path P, if there exists an $w_1 \in V(P)$ such that $c_{w_1 w} + c_{w p_l} = c(w_1 P)$, $|V(w_1 P)| = 2$, and $c_{w_1 w'} = c_{w_1 w}$. Here, w' is the node that lies between w_1 and p_l (which is well-defined because of $|V(w_1 P)| = 2$). See Figure 5.5 for a sketch of the situation. □

It is clear that Exclusion Rule 5.3.1 does not satisfy Assumption 4.2.1. We will now give an example that shows that a different numbering of the same graph can lead to big differences in the sizes of the branching trees, when applying the tie breaking rule.

Example 5.3.1. Consider the graph $G = (V, E)$ below with the two given numberings

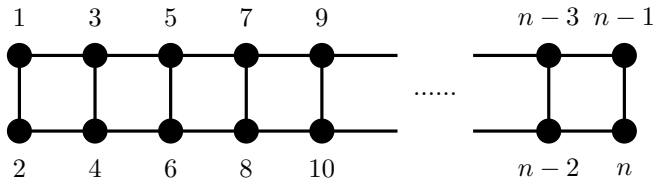

Figure 5.11: First numbering of G.

and

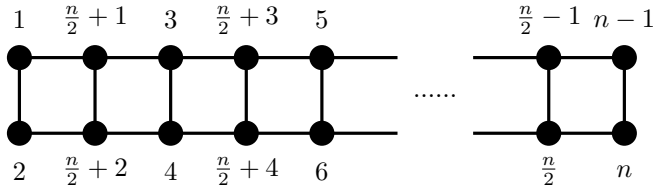

Figure 5.12: Second numbering of G.

with $n = 4k$ for $k \in \mathbb{N}$. Furthermore, we take the ordering defined by $v_i \preceq v_j$ if and only if $i \leq j$. Also we take c a non-negative constant cost function and the neighbour mapping $\phi_{\mathcal{N}}$. If we then apply the loop free, unnecessary node and tie breaking rule for these two graphs, the first numbering will give a branching tree of size linear in n, whereas the second numbering will give a branching tree of size exponential in n.

The linearity for the first numbering follows from the fact that a path will be excluded at node v_{i+1} if it uses a vertical edge $\{v_i, v_{i+1}\}$ for some $i \in \{3, 5, 7, \ldots, n-3\}$. Consider the example below, where the path $P = v_1 v_3 v_5 v_6$ will be excluded based on a tie breaking shortcut via v_4. The same would hold for the path $P = v_1 v_2 v_4 v_6 v_5$, since we would then have a tie breaking shortcut via v_3. This argument actually shows that for every $\{u, v\} \in E$, there will be at most

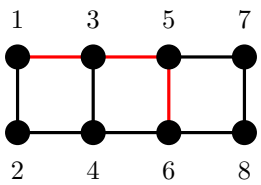

Figure 5.13: Example for $n = 8$ with path $P = v_1 v_3 v_5 v_6$.

two paths in the branching tree using this edge. Since for every n, the degree of the nodes is

bounded by three, this implies that the branching tree will have a size linear in n (this estimate can be improved). The exponential size of the second numbering follows from the fact that for every $U \subseteq \{\{i, i+1\} : i = 1, 3, 5, \ldots, n-3\}$, there exists a path $P \in W(T)$ that uses the edges of U. Intuitively, we get paths with a lot of crenellations, going up and down between the even and odd numbered nodes. □

Example 5.3.1 shows that the tie breaking rule is not stable, in the sense that a different ordering on the nodes (or a different numbering of the graph) can lead to completely different results. Therefore, when analysing the size of the branching tree of a graph, it is not wise to include the tie breaking rule (unless we would enumerate over all numberings). However, note that the size of the branching tree will never increase when including the tie breaking rule (by definition of an exclusion rule). In practice it is therefore wise to break ties, but not during the analysis of a graph (unless the ordering of the nodes has some explicit meaning).

Note that a similar argument can be given with respect to the cost function c. For a given c, we can consider $c' = c + \varepsilon$ where $\varepsilon : E \to \mathbb{R}_{\geq 0}$ is a small perturbation, so that $c'(P_1) \neq c'(P_2)$ for all $P_1, P_2 \in \mathcal{G}_{G,s,d}$, with the property that the optimal path under c' is also an optimal path under c. For Example 5.3.1, we can find both perturbations that will make the size of the branching tree linear, as well as exponential in n. No perturbations on the edges will yield a branching tree of exponential size. For a tree of linear size, we can add perturbations $\varepsilon_{12} < \varepsilon_{34} < \varepsilon_{56} < \cdots < \varepsilon_{(n-1)n}$ on the vertical edges of the graph in Figure 5.11. These perturbations are useful for the shortcut rule, since this rule can then be applied more often. The fact that different perturbations lead to different sizes of branching trees, can be considered as an unstable property of the shortcut rule with respect to the cost function. The unnecessary node and destination rule do not have this property, for example, in the case of a constant cost function (if the perturbation is much smaller than the cost of the edges).

5.4 Equivalent information mapping

For the exclusion rules defined in the previous section, and the neighbour mapping $\phi_{\mathcal{N}}$, we do not use all the information of $G_{P,\phi}$. Mathematically speaking, we can find another mapping ϕ' so that $G'_{P,\phi'}$ is a subgraph of $G'_{P,\phi}$ for all tuples (s', d', P', G', c'). This mapping has the property that, for G, c, s, d and $\mathcal{R}$ fixed, a path P arrives under ϕ if and only if it arrives under ϕ'. To that end, we make the following definition.

Definition 5.4.1. *We say that two mappings ϕ, ϕ' are equivalent for a collection $\mathcal{G}$ using the set of exclusion rules $\mathcal{R}$, if for every $(G, c, s, d) \in \mathcal{G}$ the branching tree $T(s, d, G, c, \mathcal{R}, \phi)$ is equal to the tree $T(s, d, G, c, \mathcal{R}, \phi')$. Furthermore, we define*

$$\mathcal{C}(\mathcal{G}, \mathcal{R}, \phi) = \{\phi' : \phi' \text{ is equivalent to } \phi \text{ when using the set } \mathcal{R} \text{ for the collection } \mathcal{G}\}$$

and for $\phi' \in \mathcal{C}(\mathcal{G}, \mathcal{R}, \phi)$, we say that

$$\phi' \preceq \phi \quad \Leftrightarrow \quad G_{P,\phi'} \text{ is a subgraph of } G_{P,\phi}$$

for all tuples (s, d, P, G, c).

Note that the sets $\mathcal{C}(\mathcal{G}, \mathcal{R}, \phi)$ form equivalence classes for all the information mappings for a fixed collection $\mathcal{G}$ and set $\mathcal{R}$.

Example 5.4.1. Consider the collection $\mathcal{G}$ of all possible pairs (G, c). Let $\mathcal{R}$ be the set consisting of the loop free rule. Then we have one equivalence class $\mathcal{C}(\mathcal{G}, \mathcal{R}, \phi)$ where the trivial mapping ϕ is minimal under the partial ordering defined in Definition 5.4.1. □

We will now give an equivalent mapping ϕ' so that $\phi' \preceq \phi_{\mathcal{N}}$, and that only uses information of one specific edge to a neighbour reachable from a path. To be more precise, we define ϕ' as the

mapping that sends a tuple (s, d, P, G, c) to $G_P = (V_P, E_P)$ defined by

$$V_P = \{d\} \cup \bigcup_{y \in V(P)} \mathcal{N}(y) \quad \text{and} \quad E_P = E(P) \cup \{\{q, p_l\} : q \in \mathcal{N}(p_l)\} \bigcup_{w \in V_P \setminus (V(P) \cup \{d\})} \{w, w*\}$$

where $w^* \in V(P) \setminus \{p_l\}$ is the first node on $P = s \dots p_l$ that minimizes

$$\min\{c(s \dots x) + c_{xw} : x \in V(P) \cap \mathcal{N}(w)\}.$$

It is clear from the definition of the unnecessary node rule (Exclusion Rule 5.1.2) that if there exists an x with the desired properties for some w, then we can take $x = w^*$ for that w. This means that in order to apply the unnecessary node rule (and also the destination rule), it is sufficient to only store this one edge from w to the path P. The question that arises is, if this edge is also enough to apply the shortcut rule, i.e., if there exists a shortcut in G_P defined by $\phi_{\mathcal{N}}$, does there also exist a shortcut in G_P as defined above, for a path P arriving at some node v. This question is answered in Lemma 5.4.1. This lemma can also be proven if we have a shortcut with equality, but we will not do this here. The proof is similar to the proof given below.

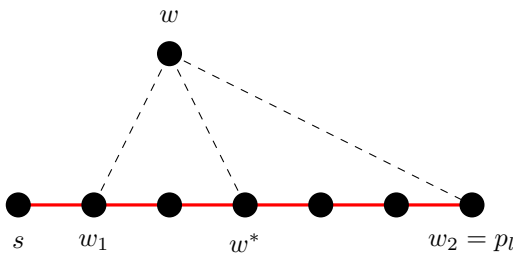

Figure 5.14: Sketch of the arguments for the equivalent mapping ϕ'.

Lemma 5.4.1. *Suppose that the edges $\{y, w\}$ and $\{w, v\}$ form a strict shortcut and that the path from x to w is shorter than the path from y to w, i.e.,*

$$c_{yw} + c_{wv} < c_{yx} + c_{xv} \quad \textit{and} \quad c_{yx} + c_{xw} < c_{yw},$$

then

$$c_{xw} + c_{wv} < c_{xv}$$

i.e., the edges $\{x, w\}$ and $\{w, v\}$ also form a shortcut. Furthermore, also if $\{x, w\}$ and $\{w, v\}$ form a shortcut and the path from y to w is shorter than that of x to w, then $\{y, w\}$ and $\{w, v\}$ form a shortcut.

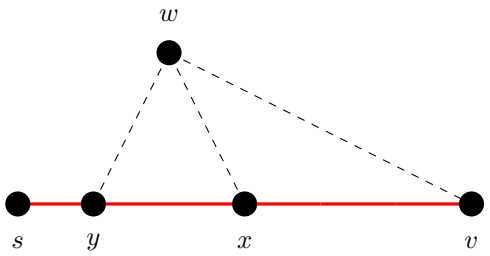

Figure 5.15: Schematic overview where the red edges might contain more nodes.

Proof. Note that the two given inequalities imply

$$c_{wv} < c_{yx} + c_{xv} - c_{yw} \quad \text{and} \quad c_{xw} < c_{yw} - c_{yx}$$

Adding these together, we get

$$c_{wv} + c_{xw} < c_{yx} + c_{xv} - c_{yw} + c_{yw} - c_{yx} = c_{xv}$$

In the second case, we have the inequalities

$$c_{xw} + c_{wv} < c_{xv} \quad \text{and} \quad c_{yw} < c_{yx} + c_{xw}$$

and combining these, we find

$$c_{yw} + c_{wv} < c_{yx} + c_{xw} + c_{xv} - c_{xw} = c_{yx} + c_{xv}$$

which shows that $\{y, w\}$ and $\{w, v\}$ also form a shortcut. □

From this lemma, it follows that the two information mappings are equivalent when using an arbitrary collection $\mathcal{G}$ and the set of exclusion rules consisting of the loop free, unnecessary node, destination and shortcut rule. We will denote this mapping as the *equivalent information mapping* $\phi_{\mathcal{N}^*}$.

5.5 Cost function c without negative cost cycles

For non-negative cost functions c, we know that Exclusion Rules 5.1.1-5.1.5 form a feasible set for every (G, c, s, d). This follows from the fact that every shortest path, that also uses the least number of nodes, arrives at d. We will now discuss what happens if we choose a more general cost function, namely $c : E \to \mathbb{R}$ that satisfies the condition of having no negative cost cycles. The loop free and unnecessary node rule still hold (these even hold for an arbitrary cost function c, which follows for example from Lemma 6.0.1). The destination rule as defined in Exlusion Rule 5.1.3 does no longer hold.

Example 5.5.1. Consider the following graph $G = (V, E)$ in Figure 5.16.

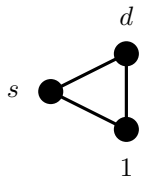

Figure 5.16: Graph $G = (V, E)$ with $c_{sv_1} = c_{sd} = 1$ and $c_{v_1 d} = -1$.

The optimal path is $P = sv_1d$, but the destination rule, as defined in Exclusion rule 5.1.3, excludes the path $P = sv_1$, which means the optimal solution will never arrive at d. □

The shortcut rule as defined in Exclusion Rules 5.1.4-5.1.5 also no longer holds, see Example 5.5.2.

Example 5.5.2. Consider the graph $G = (V, E)$ in Figure 5.17.

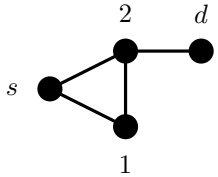

Figure 5.17: Graph $G = (V, E)$ with $c_{sv_1} = c_{sv_2} = c_{v_2 d} = 1$ and $c_{v_1 v_2} = -1$.

The optimal path is $P = sv_1v_2d$, but, using the original shortcut rule, the path $P = sv_1$ would exclude v_2, based on a shortcut via $w = v_1$. □

The difference with the original shortcut rule is that we can no longer exclude the node w, if see a shortcut via that node. That is,

$$c_{ww_1} + c_{wp_l} < c(w_1 P p_l)$$

does no longer imply that

$$c_{ww_1} < c(w_1 P p_l) + c_{wp_l},$$

which can only be concluded if $c_{wp_l} \geq 0$.

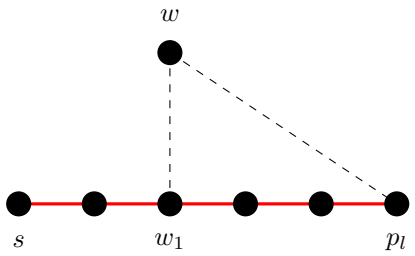

Figure 5.18: Sketch of G where the red edges might contain more nodes

This bring us to the following adjusted definition of the shortcut rule.

Exclusion Rule 5.5.1 (Shortcut rule)**.** This is the exclusion rule that maps (s, d, P, G, c) to

$$X = \begin{cases} \mathcal{N}(p_l) & \text{If there exist two strict shortcuts for } P \\ \mathcal{N}(p_l) \setminus \{w\} & \text{If there exists precisely one shortcut (via } w\text{) for } P \\ \emptyset & \text{Otherwise} \end{cases}$$

We say that $w \notin V(P)$ forms a strict shortcut for the path P if there exists an $w_1 \in V(P)$ such that $c_{w_1 w} + c_{wp_l} \leq c(w_1 P p_l)$, where we must have strict inequality in the case that $|V(w_1 P p_l)| = 3$.

The feasibility of this rule is based on Lemma 5.5.1.

Lemma 5.5.1. *Let $w \in \mathcal{N}(p_l) \setminus V(P)$ form a strict shortcut for the path P, and let $x \in \mathcal{N}(p_l) \setminus \{w\}$ arbitrary. Then Px can never be a subpath of an optimal s, d-path, that also uses the least number of nodes.*

Proof. Suppose that Px is part of some optimal s, d-path P' that also uses the least number of nodes.

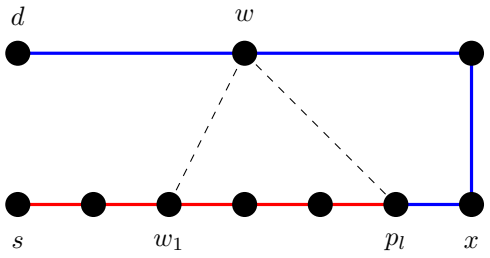

Figure 5.19: Sketch of G where the red/blue edges might contain more nodes

In particular we must have $w \in V(P')$ because of the fact that we have a shortcut via w, otherwise $(Pw_1)w(p_l P')$ would be a simple path with strictly lower cost than P'. This means that the path $(Pw_1)w(p_l P')$ is not simple but contains a cycle, which in turn contains the edge $\{w, p_l\}$. Removing this cycle leads to the simple s, d-path $(Pw_1)(wP')$, with the property that

$$c(P') \geq c(Pw_1)w(p_l P')) \geq c((Pw_1)(wP'))$$

where the last inequality holds, because we do not have any cycles of negative cost. Clearly $(Pw_1)(wP')$ uses less nodes than P', which gives a contradiction. □

The feasibility of Exclusion Rule 5.5.1 now follows from the fact that, if $w \neq w'$,

$$(\mathcal{N}(p_l) \setminus \{w\}) \cup (\mathcal{N}(p_l) \setminus \{w'\}) = \mathcal{N}(p_l).$$

We can also exploit the fact that we do not have any negative cost cycles. Let $x, y \in V(P)$ so that $\{x, w\}, \{y, d\} \in E$, and define

$$\gamma = c_{yd} + c(Q) + c_{xw} \quad \text{and} \quad \delta = c(Pyd) - c(Pw),$$

where $Q \in \{xPy, yPx\}$, depending on whether x lies before or after y on the path P (see Figure 5.20). Pyd is the restriction of P to Py, extended with d.

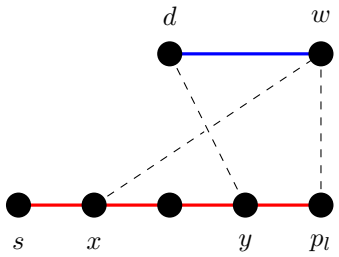

Figure 5.20: Sketch of G where $Q = xPy$ (blue edge represents wP').

If Pw could be extended to some path P' for which $c(P') = c(Pw) + c(wP') < c(Pyd)$, meaning that it might be a shortest s, d-path, then we find

$$c(wP') < c(Pyd) - c(Pw) = \delta.$$

However, the subpath wP' forms a cycle with $\{y, d\}, Q$ and $\{x, w\}$, with cost

$$c(wP') + c_{yd} + c(Q) + c_{xw} = c(wP') + \gamma < \delta + \gamma$$

This means that if $\delta + \gamma \leq 0$, then we would have a cycle of negative cost, which gives a contradiction. A simple example would be where all the edges in Figure 5.20 have cost zero. Then $c(Pyd) = c(Pw) = \gamma = 0$, so that $\gamma + \delta = 0$. More general, there might be multiple edges $\{t, d\} \in E$. We must have $c(P') < c(Ptd)$ for all those $t \in V(P)$, if P' would be a shortest s, d-path, meaning that we can generalize the definition of δ to

$$\delta = \min\{c(Ptd) : \{t, d\} \in E\} - c(Pw).$$

We can summarize these arguments in the following exclusion rule (note that the *cycle constraint* is defined in the exclusion rule).

Exclusion Rule 5.5.2 (Cycle rule)**.** This is the exclusion rule that maps (s, d, P, G, c) to

$$X = \{w : w \in \mathcal{N}(p_l) \setminus V(P) \text{ and there are } x, y \in V(P) \text{ that satisfy the } \textit{cycle constraint} \text{ for } w\}.$$

We say that $x, y \in V(P)$ satisfy the *cycle constraint* for w, if $\{y, d\}, \{x, w\} \in E$ and

$$[c_{yd} + c(Q) + c_{xw}] + [\min\{c(Ptd) : t \in V(P) \text{ and } \{t, d\} \in E\} - c(Pw)] \ \leq \ 0.$$

Here, $Q \in \{xPy, yPx\}$, depending on whether x lies before or after y on the path P (see also Figure 5.20). □

If c would be a non-negative cost function, then

$$[c_{yd} + c(Q) + c_{xw}] + [\min\{c(Ptd) : t \in V(P) \text{ and } \{t, d\} \in E\} - c(Pw)] \ \leq \ 0$$

implies

$$\min\{c(Ptd) : t \in V(P) \text{ and } \{t, d\} \in E\} \leq c(Pw),$$

using the non-negativity of the left term (in brackets). This last inequality is equivalent to the destination rule, i.e., if we already see a shorter path to d via some node on P. We also have the following theorem, of which the proof can be found in Appendix A.

Theorem 5.5.2. *Let ϕ be the neighbour mapping and let $\mathcal{G}$ be the collection of pairs where every pair (G, c) satisfies the following conditions. The graph $G = (V, E)$ does not contain cycles of length smaller than five, i.e., $\varrho(G) \geq 5$ (girth of G), and the cost function c has no cycles of negative cost. Then the set $\mathcal{R}$, consisting of Exclusion Rules 5.1.1, 5.1.2, 5.5.1 and 5.5.2, is optimal for $\mathcal{G}$.*

Proof. See Appendix A. □

Here, it is not directly clear whether or not the mapping $\phi_{\mathcal{N}^*}$ is an equivalent mapping for $\phi_{\mathcal{N}}$. If this is not the case, it means that this generalized cost function has some drawbacks for the distributed algorithm that will be introduced in Chapter 9. Furthermore, the rules introduced here also show that there exist less trivial exclusion rules than in the case for the non-negative cost function.

5.6 Concluding remarks

In this chapter we have shown how the mathematical framework, for using neighbour information, can be applied to the single path problem. We have given some collections $\mathcal{G}$ for which it was able to formulate an optimal set of exclusion rules. We have only given an analysis with respect to the information mapping $\phi_{\mathcal{N}}$, since this one follows in a natural way from the real life situation where nodes know who their neighbours are.

We have shown that some exclusion rules are unstable with respect to the input information. For example, the shortcut rule can give completely different results (in size of branching tree) if we add a small perturbation to the cost function on the edges. Furthermore, tie breaking based on the indices of the nodes can also lead to different results, depending on the choice of how to break these ties. Therefore, we should be careful on the conclusions drawn from these rules regarding the size of the branching tree we find.

We also have shown that Assumption 4.2.1 makes it possible to prove the uniqueness of some sets of exclusion rules, although one might argue that *a priori* defining an equivalent single path problem (in terms of a partial ordering on all simple paths) makes this assumption redundant. Nevertheless, the assumption does give insight into the possibilities when restricting the exclusion rules to the structure of G_P, independent of the names of the nodes. From a practical point of view, this means that the applied method is independent of how nodes are identified in the network (e.g., MAC or IP addresses). Also, it might not be trivial to implement a mechanism that can compare two nodes using a given partial ordering.

A minimal equivalent mapping tells us what the minimum amount of neighbour information that is needed to obtain the same branching tree as for the original neighbour information (for a fixed set of exclusion rules). This is important in the context of a distributed algorithm to compute the branching tree (see Chapter 9), since this will reduce the size of the messages that have to be sent. Examples in Chapter 9 show that G_P, as defined in the information mapping $\phi_{\mathcal{N}}$, can contain almost the whole graph G, which makes it harder to implement the distributed algorithm for constructing the branching tree. The fact that we can find the equivalent mapping $\phi_{\mathcal{N}^*}$ implies that the size of the messages used in the distributed algorithm will be at worst linear in the number of nodes $|V|$.

We have also seen that there exist somewhat less trivial exclusion rules in the case where c has no negative cost cycles. This also shows that the rules, that were feasible in the non-negative

case, cannot directly be extended to more general cost functions (although there exist similar formulations). Furthermore, it is also not directly clear whether or not we can find an equivalent mapping here, as in the case for non-negative cost functions.

Chapter 6

Multipath problem

In this section we will solve the problem in (4.1) for $k > 1$, with the cost function c for the paths defined by

$$c(P_1, \dots, P_k) = \sum_{j=1}^{k} c(P_j) = \sum_{j=1}^{k} \sum_{i=1}^{l_j} c_{p_{i-1}^j p_i^j},$$

where $P_j = p_0^j \dots p_{l_j}^j$ for $j = 1, \dots k$. Some of the exclusion rules from the previous section are still applicable in the case $k > 1$, which follows from the following lemma.

Lemma 6.0.1. *If P and Q are two node-disjoint paths and there exist $p_i, p_j \in P$ such that $\{p_i, p_j\} \in E(G)$ and $c(\{p_i, p_j\}) \leq c(p_i P p_j)$, then $P' = P p_i p_j P$ and Q are also two node-disjoint s, d-paths with*

$$c(P') + c(Q) \leq c(P) + c(Q).$$

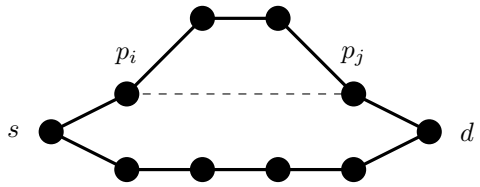

Figure 6.1: Sketch of statement. Q is the path using the lower nodes, P the path using the upper nodes.

Proof. This is clear. □

6.1 Exclusion rules

From Lemma 6.0.1, it follows that the loop free and unnecessary node can still be applied when $k > 1$. The destination rule can also still be applied as long as $x \neq s$ in the definition of Exclusion Rule 5.1.3. In Example 6.1.1, we will show that the shortcut rule is no longer feasible in general.

Example 6.1.1. Consider the graph G in Figure 6.2 with cost function c defined by

$$c_{uv} = \begin{cases} 0 & \text{if } \{u, v\} = \{v_1, v_4\} \\ 1 & \text{otherwise} \end{cases}$$

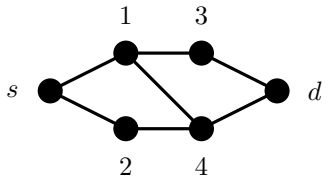

Figure 6.2: Example where shortcut rule does not work.

Furthermore, we take $k = 2$ in (4.1). Then based on the shortcut rule, the path sv_2v_4d will not arrive at d because d will be excluded by the path sv_2v_4 (using the neighbour mapping $\phi_{\mathcal{N}}$). However, it is clear that there is only one optimal solution and that one contains the path sv_2v_4d. □

For $k = 2$ we will show that is suffices to have two shortcuts, satisfying some additional requirements. Generalizations for arbitrary k will be discussed at the end. The exclusion rule itself will be formulated after we have explained the arguments. Throughout this section, we will use the neighbour mapping $\phi_{\mathcal{N}}$, and assume that c is a non-negative cost function (in order to guarantee the feasibility of the rule).

Suppose that x and y form shortcuts for a path P, i.e.,

$$c_{y_1y} + c_{yy_2} < c(y_1Py_2) \quad \text{and} \quad c_{x_1x} + c_{xv} < c(x_1P),$$

so that $E(x_1Px_2) \cap E(y_1Py_2) \neq \emptyset$. If this property holds for two shortcuts, we say that they are overlapping, see Figure 6.3. Without loss of generality, we can assume that $x_2 = p_l$, however, this does not have to hold for y_2 (see also Exclusion Rule 5.1.4).

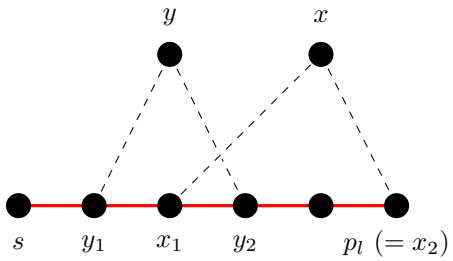

Figure 6.3: Sketch of two overlapping shortcuts.

We will show that P cannot be contained in an optimal solution, i.e., there exists no path $P' = Pp_l \dots d$ that is contained in an optimal solution of two node-disjoint s, d-paths. Suppose that P is part of some optimal solution of paths $Q_1 = P \dots d$ and Q_2, then both x and y must lie on Q_2. They cannot lie on Q_1 because of the fact that they form shortcuts for P. If either of the two would not lie on Q_2 as well, we would find a contradiction with the optimality of the pair Q_1 and Q_2, because we could replace y_1Py_2 with y_1yy_2 (or x_1Px_2 with x_1xx_2). This means that the optimal solution consists of $Q_1 = Pv \dots d$, and $Q_2 = s \dots y \dots x \dots d$ or $Q_2 = s \dots x \dots y \dots d$. Also note that P cannot be extended to x, because of the unnecessary node rule (if we assume that the cost function is non-negative).

Case 1: $Q_2 = s \dots y \dots x \dots d$**.** The optimal solution is sketched in Figure 6.4.

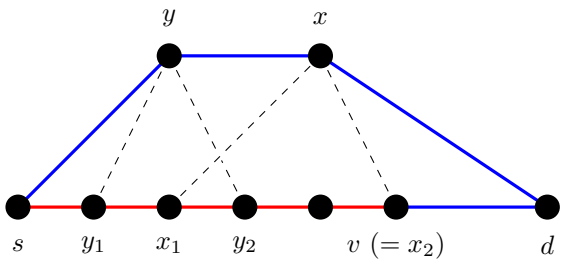

Figure 6.4: Schematic overview of the optimal solution in Case 1. The blue/red edges might contain more nodes.

We can construct a new solution by removing yQ_2x and $x_1Q_1y_2(= x_1Py_2)$, and adding the edges $\{y, y_2\}$ and $\{x_1, x\}$, in order to end up with the feasible solution (see Figure 6.5)

$$Q_1' = sQ_1x_1xQ_2d \quad \text{and} \quad Q_2' = sQ_2yy_2Q_1d.$$

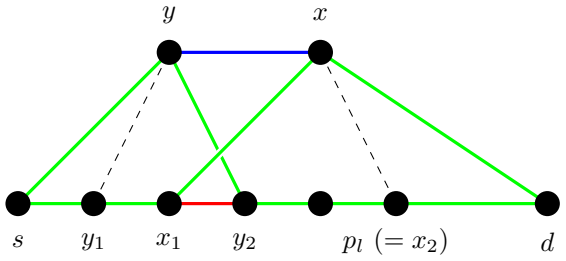

Figure 6.5: Alternative feasible solution represented using the green edges.

Due to the optimality of Q_1 and Q_2, we must have

$$c(yQ_2x) + c(x_1Q_1y_2) \leq c_{yy_2} + c_{x_1x}.$$

But this means that if

$$c_{yy_2} + c_{x_1x} < c(yQ_2x) + c(x_1Q_1y_2),$$

then P can never be part of an optimal solution. Since $c(yQ_2x)$ cannot be determined from (s, d, P, G_P, c_P), we can only conclude that P cannot be part of an optimal solution if

$$c_{yy_2} + c_{x_1x} < c(x_1Q_1y_2), \tag{6.1}$$

because we might have $c(yQ_2x) = 0$ (using the non-negativity of c).

Case 2: $Q_2 = s \ldots x \ldots y \ldots d$**.** The optimal solution is sketched in Figure 6.6.

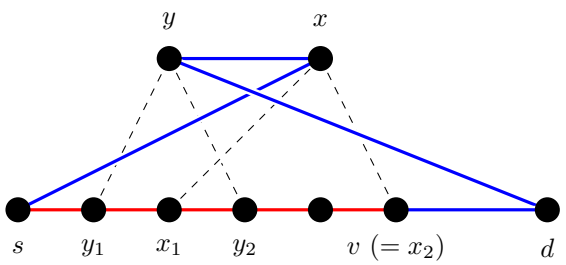

Figure 6.6: Schematic overview of optimal solution in Case 2. The blue/red edges might contain more nodes.

We can construct a new solution by removing yQ_2x and $x_1Q_1y_2$ and adding the edges $\{y, y_2\}$ and $\{x_1, x\}$ in order to end up with the feasible solution (see Figure 6.7)

$$Q_1' = sQ_1y_1yQ_2d \quad \text{and} \quad Q_2' = sQ_2xx_2Q_1d$$

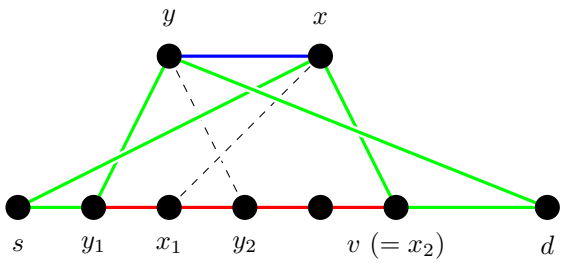

Figure 6.7: Alternative feasible solution represented using the green edges.

Due to the optimality of Q_1 and Q_2, we must have

$$c(yQ_2x) + c(y_1Q_1x_2) \leq c_{y_1y} + c_{xx_2}$$

But this means that if

$$c_{y_1y} + c_{xx_2} < c(yQ_2x) + c(y_1Q_1x_2),$$

then P can never be part of an optimal solution. Since we cannot determine $c(yQ_2x)$ from (s, d, P, G_P, c_P), we can only conclude that P cannot be part of an optimal solution if

$$c_{y_1y} + c_{xx_2} < c(y_1Q_1x_2), \tag{6.2}$$

because we might have $c(yQ_2x) = 0$.

In summary, we can only conclude that P cannot be part of an optimal solution if both (6.1) and (6.2) hold, for the simple reason that we do not know whether x or y will appear first on Q_2. Also note that these equations depend on which shortcut is defined as x and y. For some collections, it is easier to check whether or not P can be contained in an optimal solution, using the arguments in this section.

Lemma 6.1.1. *Let $\mathcal{G}$ be a collections of pairs (G, c) so that $G = (V, E)$ has no cycles of length smaller than five, i.e. $\varrho(G) \geq 5$ (girth of G), and c a constant positive cost function. Then the exclusion rule that excludes all neighbours of p_l if G_P contains two overlapping shortcuts, is feasible for $\mathcal{G}$.*

Proof. We assume, without loss of generality, that $c = 1$. Given the properties of G, the shortcuts form cycles of at least length five with the path P. Again, suppose that P would be part of an optimal solution, with Q_1 and Q_2 as in this section.

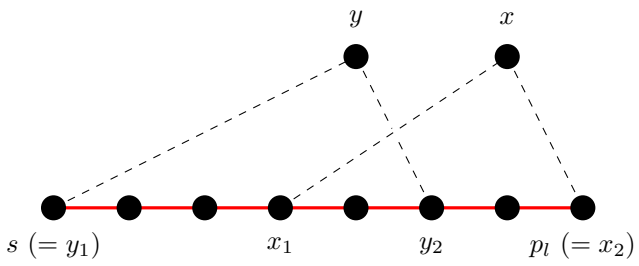

Figure 6.8: Schematic overview of two overlapping shortcuts

Because of the optimality of Q_1 and Q_2, we must have

$$c_{yy_2} + c_{x_1x} \geq c(yQ_2x) + c(x_1Q_1y_2)$$

and

$$c_{y_1y} + c_{xx_2} \geq c(yQ_2x) + c(y_1Q_1x_2)$$

as described earlier. Note that $c_{yy_2} = c_{x_1x} = 1$ and $c(yQ_2x) \geq 1$ (since there is at least one edge in yQ_2x). Then the first inequality implies that

$$1 + 1 \geq c(yQ_2x) + c(x_1Q_1y_2) \geq 1 + c(x_1Q_1y_2),$$

which gives $c(x_1Q_1y_2) \leq 1$. Since the shortcuts are overlapping, we also have $c(x_1Q_1y_2) \geq 1$, so we can conclude that $c(x_1Q_1y_2) = 1$ (note that the figure is only a sketch). This means that x_1 and y_2 are adjacent on P. From the fact that we do not have any cycles of length three or four, this means that there must be at least one node (equivalent to two edges) between y and x on Q_2, i.e., $c(yQ2x) \geq 2$. Otherwise $x_1y_2yxx_1$ would be a cycle of length four. Inserting all this information in the second inequality (using that $c_{y_1y} = c_{xx_2} = 1$), we find

$$1 + 1 \geq c(yQ2x) + 1 \geq 2 + 1,$$

which is a contradiction. This means that both inequalities cannot hold at the same time, implying that P can never be part of an optimal solution. Therefore, the exclusion rule is feasible. □

We will refer to this rule as the *double shortcut rule*. The arguments in this section can also be extended to the case where we want to find $k > 2$ node-disjoint paths. Then we will need k strict shortcuts, that are pairwise overlapping. Using the same reasoning as for the case $k = 2$, it can be argued that all the nodes, forming the shortcuts, must lie on the $k - 1$ other paths. Since we have k shortcuts, this means that there is at least one path containing two of the nodes that form the shortcuts. The proof is then reduced to the case $k = 2$.

Up till now, we assumed to have two shortcuts with strict inequality. The question that arises (as in the single path case) is whether this rule also works when we have one, or two, shortcuts with equality. The next two examples show that this is not true.

Example 6.1.2. Consider the graph $G = (V, E)$ given in Figure 6.9 and let c be a positive constant cost function.

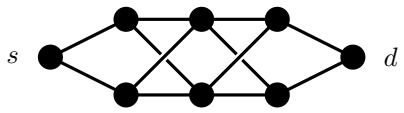

Figure 6.9: Example that shows we cannot exclude based on two equalities.

For every path P that arrives at one of the neighbours of d, we have that (s, d, P, G_P, c_P) is isomorph with the graph in Figure 6.10. This means that every message arriving at one of the neighbours of d, excludes d based on two overlapping shortcuts, for which equality holds.

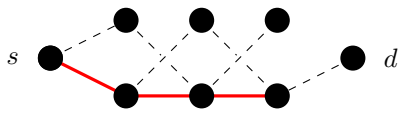

Figure 6.10: Sketch of G_P that every path, arriving at a neighbour of d, is isomorph with.

In Figure 6.11, we have given an example where we have one shortcut with inequality and one with equality (again taking $c_{uv} = 1$ for all $\{u, v\} \in E(G)$). Then for every optimal pair of paths, at least one of the two paths will be excluded by a neighbour of d, so from the paths that arrive at d, we can never construct an optimal solution.

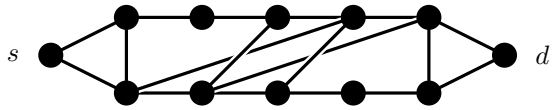

Figure 6.11: Example that shows we cannot exclude based on one inequality and one equality.

□

6.2 Optimal set of exclusion rules

We have the following result for the collection $\mathcal{G}$ of pairs (G, c) with $\varrho(G) \geq 5$, and c a positive constant cost function.

Theorem 6.2.1. *Let $\mathcal{G}$ be the collection where a pair (G, c) satisfies the following conditions. The graph $G = (V, E)$ only contains cycles of at least size five, i.e. $\varrho(G) \geq 5$ (girth of G), and the cost function c is positive and constant. Then the set $\mathcal{R}$, consisting of the loop free, unnecessary node, destination and double shortcut rule (as described in Lemma 6.1.1), is optimal.*

Proof. The feasibility of $\mathcal{R}$ follows directly from the feasibility of the exclusion rules in $\mathcal{R}$, as explained in this chapter. Suppose that there is some other set $\mathcal{R}'$, and a pair $(G, c) \in \mathcal{G}$ such that $|A(T(G, c, s, d, \mathcal{R}', \phi))| < |A(T(G, c, s, d, \mathcal{R}, \phi))|$ for some $s, d \in V(G)$. This means that there exists a node v, a path $P = s \dots v$ arriving at v and a neighbour $w \in \mathcal{N}(v)$ so that $w \in X_{(s,d,P,G_P,c_P,\mathcal{R}')}$ but $w \notin X_{(s,d,P,G_P,c_P,\mathcal{R})}$.

We can assume that $w \neq d$, the case where $w = d$ goes similar (see proof of Lemma 5.2.2). Because we do not exclude w using the set $\mathcal{R}$, this means that $w \notin V(G_P)$ and since $w \neq d$, we must have $d \notin V(G_P)$ as well. Furthermore, all shortcuts are pairwise disjoint. In Figure 6.12, we have given a sketch of the graph G_P.

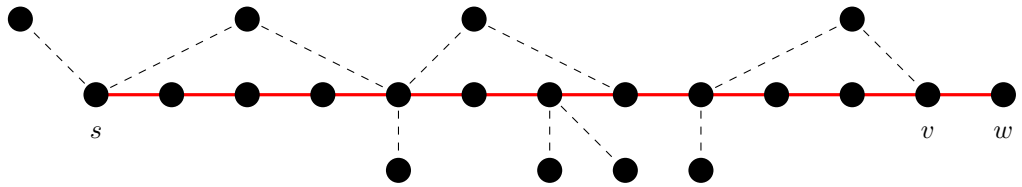

Figure 6.12: Sketch of the incoming message at v

A node that forms a shortcut might be connected to more than two nodes on the paths. Also note that s can be part of at most one shortcut (otherwise these shortcuts would automatically be overlapping). We now embed G_P in the graph H that is formed as described below. A sketch for G_P as in Figure 6.12 is given in Figure 6.13.

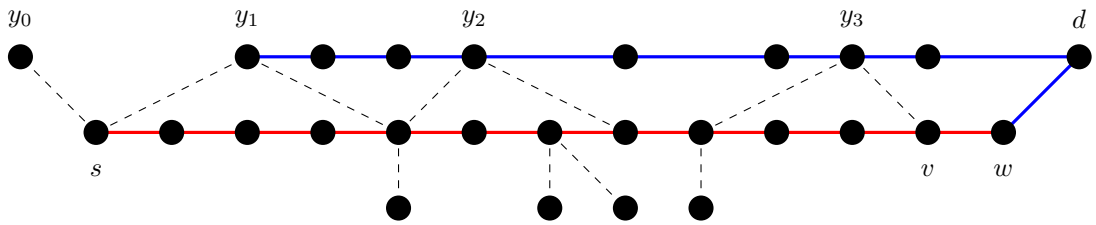

Figure 6.13: The graph H in which we embed G_{Pv}. We do not use y_0, since y_1 is connected to s.

We order of all the nodes $y_1, \dots, y_q$ that form at least one shortcut for P and add the edges $\{y_i, y_{i+1}\}$ for $i = 1, \dots, q-1$. If y_1 is not connected to s, then we choose some neighbour y_0 of s (which exists since we assume that there exists an optimal solution) and add the edge $\{y_0, y_1\}$.

If this leads to cycles of length three or four, we can add extra nodes on the edges $\{y_i, y_{i+1}\}$ that create these cycles. Furthermore, we connect w and y_q with the destination and add extra nodes if necessary, to guarantee that there are no cycles of length smaller than five. By inspection, it is clear that there is an unique pair of optimal node-disjoint paths, of which Pw is a subpath, so this means that the set $\mathcal{R}'$ is infeasible, since $(H, c_H, s, d) \in \mathcal{G}$. □

6.3 Equivalent information mapping

For an arbitrary cost function, it is not directly clear which information from G_P is sufficient to determine whether or not the double shortcut rule holds. For a constant cost function this is easier.

Lemma 6.3.1. *Suppose we have nodes x and y that form overlapping shortcuts via respectively x_1, x_2 and y_1, y_2. Let $x_3 \in P$ be such that $x_1Px_2 \subset x_1Px_3$, i.e. x_3 comes after x_2 on P. Then x and y also form shortcuts via x_1, x_3 and y_1, y_2.*

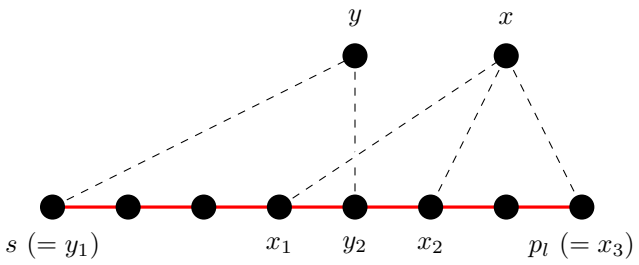

Figure 6.14: Sketch of the statement in the lemma.

Proof. This is clear. □

From Lemma 6.3.1, we can conclude that it is enough for a shortcut to store the two edges $\{x', x\}$ and $\{x, x''\}$ so that $|V(x'Px'')|$ is maximal, i.e., the first and last edge to x. Since the first edge is sufficient for the unnecessary node and destination rule, we have found an equivalent mapping for the neighbour mapping $\phi_{\mathcal{N}}$ as defined in Example 4.2.2. Note that this only holds for the collection $\mathcal{G}$ as described in Lemma 6.2.1.

Furthermore, if we only use the loop free, unnecessary node and destination rule, then the mapping $\phi_{\mathcal{N}^*}$ (see the end of Section 5.4) is an equivalent mapping for $\phi_{\mathcal{N}}$ (for all pairs (G, c) with c a non-negative cost function).

6.4 Alternative objective function

The double shortcut rule is feasible because of the definition of $c : \mathcal{P}_G^k \to \mathbb{R}$ as

$$c(P_1, \ldots, P_k) = \sum_{j=1}^{k} c(P_j) = \sum_{j=1}^{k} \sum_{i=1}^{l_j} c_{p_{i-1}^j p_i^j}.$$

Another possible objective function could be

$$c(P_1, \ldots, P_k) = \max\{c(P_1), \ldots, c(P_k)\},$$

where $c(P_j) = \sum_{i=1}^{l_j} c_{p_{i-1}^j p_i^j}$ with $P_j = p_0^j \ldots p_{l_j}^j$ for $j = 1, \ldots k$. This problem, known as the Min-Max problem, is NP-hard. The double shortcut rule is no longer applicable, because for the

paths Q'_1 and Q'_2 (see Section 6.1), it might not be the case that

$$\max\{c(Q'_1), c(Q'_2)\} \leq \max\{c(Q_1), c(Q_2)\}.$$

We mention this case to emphasize that the feasibility of an exclusion rule does also depend on the *choice* of objective function, and not only on the properties of the function $c : E \to \mathbb{R}$.

6.5 Concluding remarks

In this section we have analysed how the neighbour mapping $\phi_{\mathcal{N}}$ can be used to determine whether or not a path can be part of an optimal solution of multiple node-disjoint paths. We have shown that there exists a generalization of the shortcut rule, as in the single path case, and used this rule to provide an optimality result.

It might be interesting to study exclusion rules for more general collections, although those are probably only interesting from a mathematical point of view. For example, for an arbitrary non-negative cost function, it is not directly clear which edges suffice to apply the double shortcut rule (as opposed to the single path case, where it was enough to store just one edge). This is also the reason why the concept of an equivalent mapping is important. Otherwise, the distributed algorithm (see Chapter 9) following from these concepts, will not perform well. Also, if we want to find more than two disjoint paths, the computation time needed to determine if there are sufficient overlapping shortcuts in G_P will be too high.

It is also interesting to note that the loop free, unnecessary node and destination rule can be used to solve the Min-Max problem (which follows directly from Lemma 6.0.1). This also tells us, a priori, that the size of the branching tree will probably not be polynomial in $|V|$ (unless $P = NP$). We will investigate the size of the branching tree in the next chapter.

Chapter 7

Bounds on the branching tree

In this section we will give upper bounds, in terms of $|V|$, on the size of the branching tree $T(G, c, s, d, \mathcal{R}, \phi)$, based on the neighbour mapping $\phi = \phi_{\mathcal{N}}$ (or $\phi_{\mathcal{N}^*}$), and the set of exclusion rules $\mathcal{R}$ consisting of Exclusion Rules 5.1.1 and 5.1.2. As explained before, these two rules can solve (4.1) for every k. From some worst-case examples it will follow that in general, the destination rule will not lead to structural improvements for the upper bounds. We first give an upper bound for arbitrary pairs (G, c) and later an improved upper bound for some specific non-negative cost functions c (and G arbitrary). We will introduce some additional notation and definitions regarding branching trees. In this section, we use k to indicate the number of levels of a branching tree, not the number of paths we are looking for.

Remember from Definition 4.1.1 that $L_{T,k}$ denotes the set of nodes on level k of the branching tree $T = (W, A)$, i.e., the unique path from the root r to the node $v \in L_{T,k}$ has length k. We say that a branching tree has k levels if

$$V = \bigcup_{i=0,\ldots,k} L_{T,i},$$

where $L_k \neq \emptyset$, and we define $L_T = L_{T,k}$. Furthermore, the *subtree with root* $r' \in V$ is the induced subgraph that contains r' and all the nodes that can be reached from r'.

Example 7.0.1. In Figure 7.1, an example of a branching tree is given to illustrate the notation.

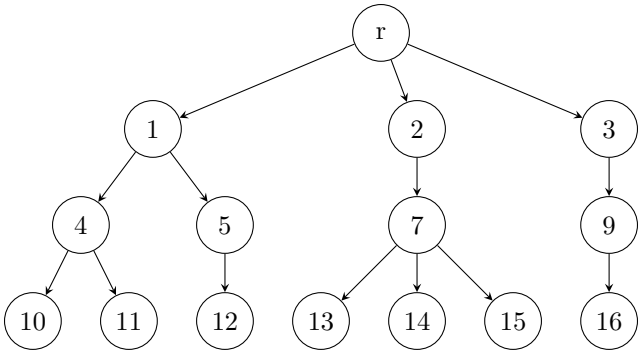

Figure 7.1: Example of a branching tree with $L_T = L_3 = 7$

Here we have $L_1 = \{1, 2, 3\}$, $L_2 = \{4, 5, 6, 7, 8, 9\}$ and $L_3 = \{10, 11, 12, 13, 14, 15\}$. We can say, for example, that node 7 branches into three nodes of level three. The subtree with root 1 is the induced subgraph that contains the nodes $1, 4, 5, 10, 11$ and 12. □

7.1 Upper bound for arbitrary (G, c)

We will now give an upper bound on the size of the tree $T(s, d, G, c, \phi, \mathcal{R})$ for arbitrary pairs (G, c) where c is a non-negative cost function. In this section we use the notation $|V| = n$.

Theorem 7.1.1. *Let $G = (V, E)$ be a graph, c a non-negative cost function on the edges of G and $s, d \in V$ arbitrary. Then we have*

$$|A(T(s, d, G, c, \phi, \mathcal{R}))| \leq \sum_{i=0,\ldots,k} \binom{n}{i} \leq 2^{n-1}$$

where $\mathcal{R}$ consists of the loop free and unnecessary node rule, and $\phi = \phi_{\mathcal{N}}$. Furthermore, k is the number of levels of the branching tree $T = (W, A)$.

Proof. We will show that for a fixed set $U = \{s, u_1, \ldots, u_k\} \subseteq V$, with $|U| = k+1$, we can have at most one path $P \in L_{T,k}$. Suppose we have a paths $P \in L_{T,k}$ where, without loss of generality, we assume that $P = su_1 \ldots u_k$. Furthermore, let $P' = sw_1 \ldots w_k$ be a path that uses the u_j in some different order. Let i be the smallest index so that $u_i \neq w_i$ (which implies that $Pu_{i-1} = P'w_{i-1}$, see Figure 7.2).

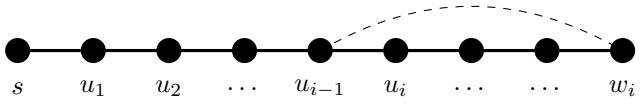

Figure 7.2: Sketch of the situation. P is the path using the solid lines.

The fact that Pw_i arrives at w_i implies that

$$c(\{u_{i-1}, u_i\}) \leq c(u_{i-1}Pw_i) < c(\{u_{i-1}, w_i\})$$

because of the unnecessary node rule. But this means that (note that u_i occurs on P' after w_i by construction)

$$\begin{aligned} c(P'u_i) &\geq c(P'w_i) \\ &= c(Pu_{i-1}) + c(\{u_{i-1}, w_i\}) \\ &> c(Pu_{i-1}) + c(\{u_{i-1}, u_i\}) \\ &= c(P'u_{i-1}) + c(\{u_{i-1}, u_i\}) \end{aligned}$$

using the non-negativity of c. This implies that $P'u_i$ will never arrive at u_i, because the node v (so that $P'u_i = s \ldots w_i \ldots vu_i$) will not extend $P'v$ to u_i based on the unnecessary node rule. This means that $P' \notin L_{T,k}$. We have shown that for every $U \subseteq V$ of size $k+1$, there can be at most one path in $L_{T,k}$ using these nodes. This shows that the number of subsets of V, containing s, forms an upper bound for $|A(T)|$, because we have an injective mapping between the $P \in W(T)$ and the subsets of V containing s (by what we have just shown). This yields an upper bound of 2^{n-1}, because every subset must contain s. It is actually possible to given an upper bound of

$$\binom{n}{k}$$

for every level k, leading to a better upper bound, but since *a priori* we do not know the number of levels of the branching tree, a bound in terms of n makes more sense. □

Example 7.1.1 shows that the upper bound 2^{n-1} is tight if we only use the loop free and unnecessary node rule. Whether or not this bound also holds if the shortcut rule is included, will be discussed at the end of this chapter. The example given in 7.1.1 cannot be used.

Example 7.1.1. Consider the complete graph $K_n = (V, E)$, where $V = \{v_1, \ldots, v_n\}$, and define c by

$$c_{ij} = |i - j| - \varepsilon_n \quad \text{for all} \quad \{i, j\} \in E$$

for some $\varepsilon_n > 0$ sufficiently small. Without loss of generality, we can take $s = v_1$ and $d = v_n$. We will show that a path $P = v_{i_1} v_{i_2} \ldots v_{i_k}$ arrives at v_{i_k} if and only if $i_1 < i_2 < \cdots < i_k$, where $i_1 = 1$.

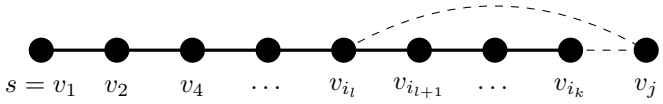

Figure 7.3: Example of path with increasing sequence of nodes.

Suppose that a path with increasing sequence arrives at node i_k (for some k). We will show that Pv_j arrives at v_j for all $j > i_k$. If P excludes one of these v_j, then this happens because of the unnecessary node rule (we cannot have $v_j \in V(P)$ since P uses an increasing sequence of nodes). For $v_{i_l} \in V(P) \setminus \{v_{i_k}\}$ we have, by construction of P,

$$c(Pv_{i_l}) + c(\{v_{i_l}, v_j\}) = (i_l - 1) - |E(Pv_{i_l})|\varepsilon_n + (j - i_l) - \varepsilon_n = (j - 1) - (|E(Pv_{i_l})| + 1)\varepsilon_n$$

and

$$c(Pv_j) = (j - 1) - (|E(P)| + 1)\varepsilon_n$$

Since there lies at least one node between v_{i_l} and v_j on Pv_j, we have $|E(Pv_{i_l})| + 1 < |E(P)| + 1$, implying that

$$c(Pv_j) < c(Pv_{i_l}) + c(\{v_{i_l}, v_j\}).$$

Because $v_{i_l} \in V(P) \setminus \{v_{i_k}\}$ was arbitrary, we will not exclude v_j based on the unnecessary node rule. Now, for every fixed $U \subseteq V$ with $|U| = k + 1$, we know that at most one path, using the nodes in U, can be in $L_{T,k}$. But the path using the nodes of U in increasing order will arrive, by what has just been proven. This means that we have a bijection between the subsets U and the nodes in $W(T)$ which shows that the bound 2^{n-1} is tight. □

7.2 Improved upper bound for (G, c) with cycle-inequality

For certain cost functions c, we can give a better upper bound. We will first prove results for arbitrary branching trees and apply these results to the branching trees $T(G, c, s, d, \phi_{\mathcal{N}}, \mathcal{R})$. We will introduce some additional definitions regarding arbitrary branching trees.

Definition 7.2.1. *For $v \in L_{T,i}$, we define $b(v)$ as the number of nodes that v branches into, on level $i + 1$. We denote the set of nodes that v branches into by $B(v)$. Then we define*

$$M_v = 1 + \sum_{j \in V(Q) \setminus \{v\}} b(j)$$

where Q is the (unique) path from r to v. Note that M_v is the number of nodes on the path from r to v together with the total number of direct neighbours of the nodes on the path (except for v)

Assumptions 7.2.1. *We assume $b(v) \geq 1$ for all $v \in V \setminus L_T$. This implies that $L_{T,i} \leq L_{T,j}$ for $i \leq j$.*

Definition 7.2.2. *We say that a tree T is balanced if, for every k, $b(v) = b(w)$ whenever $v, w \in L_{T,k}$. We then write $b(v) = b(w) = b_k$ and $T = T(b_0, \ldots, b_{k-1})$.*

We will now give a result for balanced trees (satisfying Assumption 7.2.1) on the number $|L_T|$ and later generalize this result to arbitrary trees (satisfying Assumption 7.2.1), in order to give an improved upper bound.

Lemma 7.2.2. *Let T be a balanced branching tree of k levels and*

$$M_T = \max\{M_v : v \in L_T\}.$$

Then there exists a unique balanced tree $T^* = T^*(b_0^*, \ldots, b_{k-1}^*)$ such that

$$M_T = M_{T^*} \quad \textit{and} \quad |L_T| \leq |L_{T^*}|$$

where

$$b_i^* \in \left\{ \left\lfloor \frac{b_0 + b_1 + \cdots + b_{k-1}}{k} \right\rfloor, \left\lfloor \frac{b_0 + b_1 + \cdots + b_{k-1}}{k} \right\rfloor + 1 \right\}$$

and $b_i^* \leq b_j^*$ if and only if $j \leq i$.

Proof. First consider the case where we have a branching tree of two levels, with $b(r) = b_0$ and $b(v) = b_1$ for every $v \in L_{T,1}$, so that $M_v = 1 + b_0 + b_1$ for all $v \in L_{T,2}$ (see Figure 7.4 for an example).

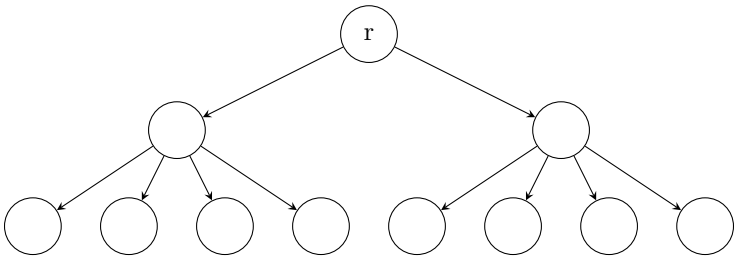

Figure 7.4: Branching tree T, where $M_v = 1 + b_0 + b_1 = 1 + 2 + 4 = 7$ for all nodes on level two.

If $b_1 \geq b_0 + 1$, then we can define the branching tree T' by letting r branch into $b_0 + 1$ nodes of $L_{T',1}$ and let every $v \in L_{T',1}$ branch into $b_1 - 1$ nodes (see Figure 7.5, where T is taken from Figure 7.4).

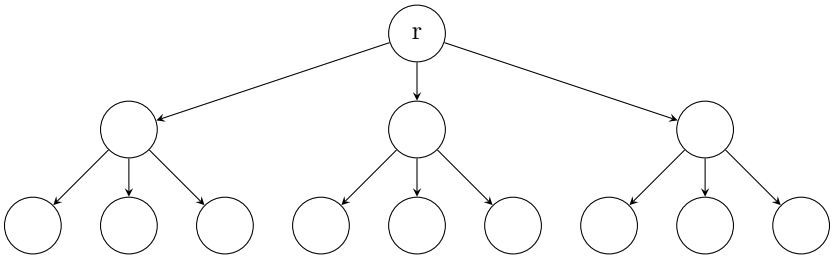

Figure 7.5: Branching tree T', where $M_v = 1 + 3 + 3 = 7$ for all nodes on level two.

Then we have that $M_v = 1 + (b_0 + 1) + (b_1 - 1) = 1 + b_0 + b_1$, implying that $M_T = M_{T'}$. However, we now have

$$|L_{T'}| = (b_0 + 1)(b_1 - 1) \geq b_0 b_1 = |L_T|$$

since $b_1 \geq b_0 + 1$. Furthermore, it is clear that if we interchange b_0 and b_1 then the resulting branching tree T'' has the properties that

$$L_T = b_0 b_1 = b_1 b_0 = L_{T''} \quad \text{and} \quad M_T = 1 + b_0 + b_1 = 1 + b_1 + b_0 = M_{T''}$$

For the branching tree T from Figure 7.5, we find the branching tree of Figure 7.6.

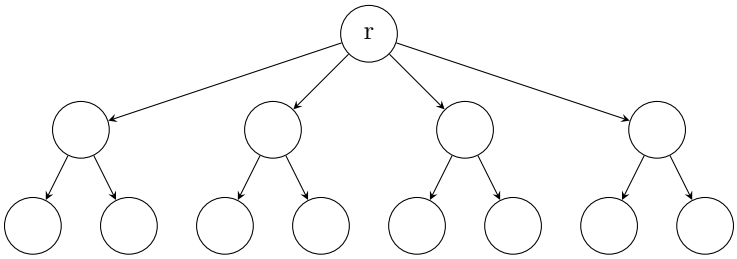

Figure 7.6: Branching tree T'', where $M_v = 1 + b_1 + b_0 = 1 + 4 + 2 = 7$ for all nodes on level two.

If we now take a balanced tree T of k levels, then we also can interchange b_i, b_{i+1} for an $i \in \{0, \dots, k-2\}$ in order to end up with T'', that satisfies $L_T = L_{T''}$ and $M_T = M_{T''}$. This follows from the fact that

$$L_T = \prod_{i=0}^{k-1} b_i \quad \text{and} \quad M_T = 1 + \sum_{i=0}^{k-1} b_i$$

and the fact that multiplication and addition are commutative (generalization of the argument above).

We can use the two techniques explained above (to obtain T' and T'') to show that for every balanced tree T of k levels, there exists a tree $S = S(T)$ such that $|L_S| \geq |L_T|$, $M_S = M_T$ and (where the b'_i are the numbers in S)

$$|b'_i - b'_j| \leq 1 \text{ for all } 0 \leq i, j \leq k-1,$$

i.e., there exists an s such that $b'_i = s$ or $b'_i = s+1$ for every $i = 1, \dots, k-1$. This can be done as follows. If $|b_i - b_j| \geq 2$ for some i and j, then we can interchange so that b_i, b_j occur on the last two levels. Then we apply the first technique for all the subtrees with roots on level $k-2$. Furthermore, we can even take S such that $b'_j \leq b'_i$ if $j \geq i$, i.e., S first has some levels where every node branches into $s+1$ nodes and then some levels where every node branches into s nodes. This s is unique, namely

$$s = \left\lfloor \frac{b_0 + b_1 + \cdots + b_{k-1}}{k} \right\rfloor.$$

Note that if we have two of these trees $T_1 = T_1(a_1, \dots, a_{k-1})$ and $T_2 = T_2(b_1, \dots, b_{k-1})$ satisfying

$$\sum_{i=0}^{k-1} a_i = \sum_{i=0}^{k-1} b_i,$$

then $S(T_1) = S(T_2)$, which follows from the uniqueness of s (explained above) and the fact that T_1 and T_2 have the same number of levels. □

Lemma 7.2.3. *For every branching tree T of k levels, satisfying Assumption 7.2.1, there exists a balanced tree T' of k levels such that*

$$|L_T| \leq |L_{T'}| \quad \textit{and} \quad M_T = M_{T'}$$

Proof. First, note that w.l.o.g. we can assume that $M_v = M_T$ for all $v \in L_T$. If this is not the case, then we can replace $b(w)$ by $b'(w) = b(w) + M_T - M_v$, where w is such that $v \in B(w)$. Then we have $M_{v'} = M_T$ for every $v' \in B(w)$. Doing this for all $w \in L_{T,k-1}$ where $M_w + b(w) \neq M_T$, we end up with a tree T'' where $|L_{T''}| \geq |L_T|$ and $M_T = M_{T''}$, and it is then sufficient to show the result for T''.

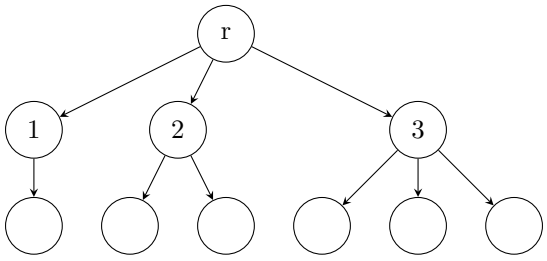

Figure 7.7: We let $b'(1) = b(1) + 2$ and $b'(2) = b(2) + 1$ to get Figure 7.5 (where the claim holds).

We will use induction on the number of levels k of the tree. For a tree of one level, the result clearly holds (since such a tree is itself a balanced tree). Now let T be a tree with $k+1$ levels, and suppose that the result holds for all trees of k levels. In particular, this means that the subtrees with root $t \in L_{T,1}$ (denoted by T_t) all have a balanced tree $S_t = S(T_t)$, as described in the proof of Lemma 7.2.3, so that $M_{S_t} = M_{T_t}$ and $|L_{S_t}| \geq |L_{T_t}|$. Note that

$$M_{S_t} = M_T - |L_{T,1}| = M_T - b(r)$$

holds for all $t \in L_{T,1}$, since we assumed $M_v = M_T$ for all $v \in L_T$. But this means that $S_{t_1} = S_{t_2}$ for all $t_1, t_2 \in L_{T,1}$ by the uniqueness proven in Lemma 7.2.3. If we then replace all the subtrees T_t (for $t \in L_{T,1}$) by this $S = S_{t_1} = S_{t_2} = ..$, we have proven the induction hypothesis for $k+1$, since the resulting tree is a balanced tree (by the uniqueness of S). □

Theorem 7.2.4. *Let T be a branching tree of k levels, satisfying Assumption 7.2.1, and let*

$$M_T = \max\{M_v : v \in L_T\}.$$

Then we have

$$|L_T| \leq \left(\frac{M_T - 1}{k}\right)^k \leq \left(e^{\frac{1}{e}}\right)^{M_T - 1} \approx 1.4447^{M_T - 1} \tag{7.1}$$

Using Lemma 7.2.3, it is enough to show the result for a balanced tree T. This balanced tree satisfies

$$|L_T| = b_0 b_1 b_2 \cdots b_{k-2} b_{k-1} \quad \text{and} \quad M_T = 1 + \sum_{i=0}^{k-1} b_i$$

This means that the (continuous) program

$$\begin{array}{rll} p^* = & \max & b_0 b_1 b_2 \cdots b_{k-2} b_{k-1} \\ & \text{s.t.} & \sum_{i=0}^{k-1} b_i = M_T - 1 \\ & & b_i \geq 0 \end{array}$$

forms an upper bound for $|L_T|$. Using convex optimization methods or the arithmetic-geometric inequality, we find the optimal solution

$$b_i^* = \frac{M_T - 1}{k}$$

for $i = 0, \ldots, k-1$. This gives us an objective value of

$$p^*(k) = \left(\frac{M_T - 1}{k}\right)^k$$

Optimizing this over $1 \leq k \leq M_T$ gives the value

$$k^* = \frac{M_T - 1}{e}$$

so that

$$p^*(k^*) = \left(e^{\frac{1}{e}}\right)^{M_T - 1}$$

□

It is possible to give a slightly better exponential upper bound using a more direct approach. The added value of the previous theorem is that we have a better upper bound in terms of the number of levels of the branching tree. However, in the branching trees of our interest, we will in general not know the number of levels of the tree, so that the theorem given below might be more interesting.

Theorem 7.2.5. *Let T be a branching tree of k levels and*

$$M_T = \max\{M_v : v \in L_T\}$$

Then we have

$$|L_T| \leq \left(3^{1/3}\right)^{M_T - 1} \tag{7.2}$$

Proof. We will use induction on the level of the tree. Assume that (7.2) holds for all trees of i levels. Now suppose that T is a tree with root r of $i+1$ levels and let $b(r) = b$. By the induction hypothesis, we have

$$|L_{T'_v}| \leq \left(3^{1/3}\right)^{M_T - 1 - b}$$

for all subtrees T'_v with root $v \in L_{T,1}$. We then have

$$|L_T| = \sum_{v \in L_{T,1}} |L_{T'_v}| \ \leq \ b\left(3^{1/3}\right)^{M_T - 1 - b} \ \leq \ \left(3^{1/3}\right)^{M_T - 1}$$

where the last inequality holds if and only if

$$b^{1/b} \leq 3^{1/3}.$$

This last inequality is true for every integer $1 \leq b \leq M$, since the maximum of the function $f(b) = b^{1/b}$ is attained for $b = 3$. □

We will apply these theorems to pairs (G, c), where c satisfies the following definition.

Definition 7.2.3. *We say that a pair (G, c) satisfies the cycle-inequality if for every cycle $C = a_1 \dots a_l a_1$ in G, we have*

$$2 \times c(\{a_i, a_{i+1}\}) \leq c(C)$$

for $i = 1, \dots, l$ (where $a_1 = a_{l+1}$). This is equivalent to saying that the path $P = a_i a_{i+1}$ is a shortest a_i, a_{i+1}-path.

Example 7.2.1. Assume that c' is a metric on $V \times V$, i.e., the ordered pair (V, c') is a metric space. We take the cost function c as the restriction of c' to E (which is well-defined because of the symmetry property of a metric). The triangle-inequality for the metric c' implies the cycle-inequality for c. An example of a metric could be the Euclidean distance between nodes in a plane. □

We will give an improved upper bound for pairs (G, c) satisfying the cycle-inequality.

Lemma 7.2.6. *Let (G, c) satisfy the cycle-inequality, then for all $s, d \in V$, we have*

$$|A(T(s, d, G, c, \phi, \mathcal{R}))| \ \leq \ n\left(3^{1/3}\right)^n$$

Here, ϕ and $\mathcal{R}$ are as in Theorem 7.1.1.

Proof. Let $P = s \dots v$ be a path that arrives at v, i.e., $P \in W$. We claim that for every $w \notin V(P)$ there can be at most one $x \in V(P)$ such that Pxw arrives at w, i.e., $Pxw \in W$. In particular, if such an x exists it will be the first $x \in V(P)$ connected to w for which Pxw arrives at w. This is true because if there would be another $x' \in V(P)$ also connected to w, then we would have

$$c_{xw} \leq c(xPx') + c_{x'w}$$

using the cycle-inequality that (G, c) satisfies. This means that w will be excluded by Px' based on the unnecessary node rule.

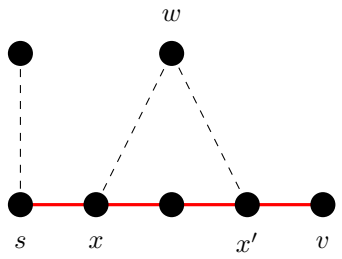

Figure 7.8: Sketch of G_P where there might be more nodes on the red edges (i.e. the path P).

This implies that

$$M_P = 1 + \sum_{j \in V(Q_P) \setminus \{v\}} b(j) \;\leq\; n$$

since a node $w \in V$ either lies on P, or there is at most one $x \in V(P)$ so that $Pxw \in W$. We now define $i_{\max}$ as the level of T that maximizes

$$\max\{|L_{T,i}| : i = 0, \dots, k\}$$

and $T_{\max}$ as the tree with

$$W_{\max} = \{w \in W : w \text{ lies on a path from the root } s \text{ to a node } P \in L_{T,i_{\max}}\}$$

and $A_{\max}$ as the induced subgraph on $W_{\max}$. Note that $T_{\max}$ satisfies Assumption 7.2.1. Using Theorem 7.2.5 we find

$$|L_{T,i_{\max}}| \leq \left(3^{1/3}\right)^n$$

which implies that $|A(T)| \leq n \left(3^{1/3}\right)^n$. □

In Example 7.2.2, we will show that the upper bound of (7.1) is tight for every integer $1 \leq k \leq M$.

Example 7.2.2. Let k be fixed and take $M = tk + 1$ for $t \in \mathbb{N}$. We define $G = (V, E)$ by

$$V = \{s\} \bigcup_{i=1,\dots k} V_i$$

where the $V_i = \{v_i^1, v_i^2, \dots, v_i^t\}$ are pairwise disjoint sets, not containing s, and $d = v_k^1$. Furthermore, E is defined by

$$\{u_i^a, v_j^b\} \in E \quad \Leftrightarrow \quad |i - j| = 1$$

for all i, j, a, b. For c, we take a constant non-negative cost function on E.

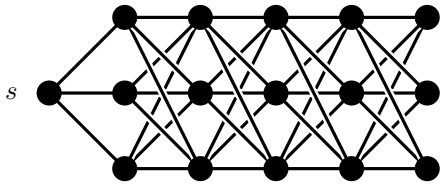

Figure 7.9: Example where $t = 3$ and $M = 16$.

Then we have

$$|L_T| = |L_{T,k}| = \left(\frac{M-1}{k}\right)^k$$

for every $t \in \mathbb{N}$, showing that the bound in (7.1) is tight with respect to M. Note that we can use the case $t = 3$ to show that the bound in (7.2) is tight. □

7.3 Comments and other bounds

For the proofs in the previous two sections, we only used the loop free and unnecessary node rule. We will now explain why the destination and shortcut rule were not taken into account.

In the worst-case Example 7.2.2, it can be seen that the shortcut rule will never be applied (both for single path and multipath), meaning that this rule will not lead to a structural improvement in the case that a pair (G, c) satisfies the cycle-inequality. For general pairs (G, c) it might be possible to improve the bound 2^{n-1} when the shortcut rule is included.

The destination rule can also not lead to a structural improvement. Suppose we have a graph $G = (V, E)$ and $s, d \in V$ fixed. We can then make the graph $G' = (V', E')$ by adding a node on all the edges that contain the destination node d. In this way, the destination rule will never be applied. We will illustrate this using the graph from Figure 4.1.

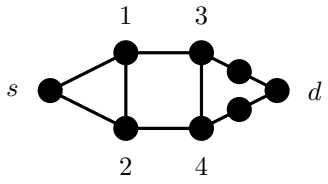

Figure 7.10: The graph G' for the graph G in Figure 4.1.

For example, the path $P = sv_2v_4v_3$ will no longer be excluded, since v_4 is not connected to the destination d anymore. Note that in general, the degree of d in G does not depend on the size of G, showing that for arbitary pairs (G, c) this rule will not lead to structural improvements.

We will also give an additional upper bound based on the minimum degree

$$d_{min} = \min\{d(v) : v \in V\}$$

where $d(v)$ is the degree of a node v in the graph G, where the pair (G, c) satisfies the cycle-inequality.

Lemma 7.3.1. *Fix some $M \in \mathbb{N}$ and let T be a branching tree so that*

$$M_T = \max\{M_v : v \in L_T\} \leq M$$

Furthermore, let $f(M)$ be an upper bound on $|A(T)|$. Then we have

$$|A(T)| \leq bf(M - b),$$

where $b = b(r)$ with r the root of T,

Proof. This is clear. □

We want to apply this lemma as follows (see Section 9.2.4). If we have some pair (G, c), satisfying the cycle-inequality, and

$$d_{\min} = \alpha n \quad \text{for} \quad \frac{1}{n} \leq \alpha \leq \frac{n-1}{n},$$

then we have

$$|A(T)| \leq \mathcal{O}\left(\alpha n \left(3^{1/3}\right)^{(1-\alpha)n}\right)$$

using $f(n) = \left(3^{1/3}\right)^n$. From this it can be seen that as $\alpha \to 1$, we get a more linear upper bound. This means that the size of the branching tree will decrease when the minimum degree of the graph increases, or, roughly speaking, when the graph becomes more dense. We will use this result in Section 9.2.4.

A last comment we want to make here, is about the relation between the original approach, in which we enumerate over all possible paths, and the approach using exclusion rules, for the information mapping $\phi_{\mathcal{N}}$. Suppose that $G = (V, E)$ is an arbitrary graph. We can make the subdivision G' by dividing every edge in E into two edges. In this way we create a graph $G' = (V', E')$ with $|V'| = |V| + |E|$. We give both edges the same cost as the original edge. This new cost function will be denoted by c'. Then we have

$$2|A(T(s, d, G, c, \phi_{\mathcal{N}}, \mathcal{R}_{loop})| = |A(T(s, d, G', c', \phi_{\mathcal{N}}, \mathcal{R})|$$

where $\mathcal{R}_{loop}$ is the loop free rule, $\mathcal{R}$ the set of Exclusion Rules 5.1.1-5.1.5 and $s, d \in V$. This tells us *a priori* that there cannot exist a polynomial upper bound on the size of the branching trees.

7.4 Lower bound using induced subgraphs

If we only use the loop free, unnecessary node and destination rule, we can give a lower bound for the size of the branching tree. We will show that every induced subgraph forms a lower bound on $|A(T(s, d, G, c, \phi_{\mathcal{N}}, \mathcal{R})|$.

Lemma 7.4.1. *Let (G, c) be a pair where $G = (V, E)$ a graph and c a non-negative cost function. Let H be an induced subgraph of G. Then*

$$|A(T(s, d, H, c_H, \phi_{\mathcal{N}}, \mathcal{R})| \leq |A(T(s, d, G, c, \phi_{\mathcal{N}}, \mathcal{R})|$$

where c_H is the restriction of c tot $E(H)$.

Proof. Let $P = s \ldots v$ be a path that arrives at v and is extended to some $w \in \mathcal{N}(v)$ for the pair (H, c_H), i.e., we have $(P, Pw) \in A(T(s, d, H, c_H, \phi_{\mathcal{N}}, \mathcal{R}))$. If we would exclude w for the pair (G, c), this would mean that there exists some $x \in V(P)$ such that $\{x, w\} \in E(G)$ and $c(Px) + c_{xw} < c(P) + c_{vw}$. Since H is an induced subgraph, we also have $\{x, w\} \in E(H)$, which means we also exclude w for the path P in the pair (H, c_H), but this gives a contradiction. The argument for the destination rule is similar. □

Note that this argument does not hold if we also use the shortcut rule, since we might have a shortcut for P using a node $y \notin V(H)$. Nevertheless, this lower bound is relevant for others versions of the problem, e.g., for solving the Min-Max problem.

7.5 Concluding remarks

In this chapter we have given upper bounds on the size of the branching tree, when using the loop free and unnecessary node rule. These rules have been chosen since they are applicable in both the single and the multipath problem. Furthermore, it can be shown that the destination rule will never lead to a structural improvement for reducing the size of the branching tree. We have also shown that the size of the tree decreases as the graph G gets more dense (for cost functions satisfying the cycle-inequality).

For the two above-mentioned rules, it also follows that every induced subgraph gives a lower bound on the size of the tree (when only using those two rules). Furthermore, the fact that these two rules can also be used to solve some NP-complete problems already tells us, a priori, that the size of the branching tree will probably not be polynomial in the number of nodes of the graph.

An open question is whether or not we can give a structural improvement when including the shortcut rule (in the single path problem). For cost functions satisfying the cycle-inequality, this is not true, as can be seen from the worst-case example in Example 7.2.2. However, it might be possible to give an improvement for pairs (G, c) where c is an arbitrary non-negative cost function, i.e., to improve the bound $2^{|V|-1}$. A lower bound (as suggested by Dion Gijswijt) can be given by taking the instance from Example 7.1.1, where we only consider edges between even and odd numbered nodes, i.e.,

$$\{v_i, v_j\} \in E \quad \Leftrightarrow \quad |i - j| \bmod 2 = 1.$$

This example then gives a lower bound of $\phi^{|V|-1}$, where $\phi = 1.61...$ is the golden ratio. The conjecture is that this bound is tight. Intuitively, this follows from the relation between the unnecessary node and shortcut rule, see Figure 7.11.

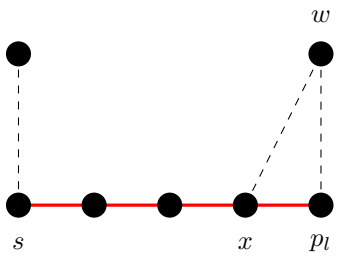

Figure 7.11: Sketch of G where $\{x, p_l\} \in E$.

Suppose that the path $Pw = s \dots x p_l w$ is not excluded based on the unnecessary node rule. This implies

$$c_{xp_l} + c_{p_l w} < c_{xw},$$

meaning that we have a shortcut, via p_l, for the path Pxw (restriction of P to x, extended with w). This means that if the path Pw is not excluded, then we immediately know that the path Pxw will never be extended after its arrival at w, which restricts the size of the branching tree.

Chapter 8

Augmenting paths

In this chapter we will explain how we can use exclusion rules to find two optimal edge-disjoint paths. We will start by introducing some notation regarding directed graphs, leading to the concept of augmenting paths. The reason for this analysis is that a lot of central algorithms use augmenting paths to find optimal solutions, so it is interesting to see how we can incorporate augmenting paths in the existing approach. We will give a translation of these concepts to undirected graphs (since up till now, we have defined the framework in terms of undirected graphs). This is a tedious procedure, but necessary to guarantee that everything is well-defined from a mathematical point of view. The notation for directed graphs can also be found in Section 2.1, but for the completeness of the second part of this thesis, we will repeat it here.

8.1 Notation

Let $G = (V, A)$ be a *directed* graph, where V is a finite set and

$$A \subseteq \{(u, v) : (u, v) \in V \times V\}$$

a set of directed arcs in G. Let $c : A \to \mathbb{R}$ be a cost function on the arcs of G, where we use the notation

$$c_{uv} = c((u, v))$$

for $(u, v) \in A$. For $u \in V$ we define

$$\mathcal{N}^+(u) = \{v : (u, v) \in A\} \quad \text{and} \quad \mathcal{N}^-(u) = \{v : (v, u) \in A\},$$

and say that v is a *neighbour* of u if $v \in \mathcal{N}^+(u)$. We say that $P = p_0 p_1 \dots p_l$ is a *simple path* in G if

$$V(P) = \{p_0, \dots, p_l\} \subseteq V \quad \text{and} \quad (p_{i-1}, p_i) \in A \text{ for } i = 1, \dots, l$$

with all the p_i distinct, where we write $A(P)$ for the set of these arcs. We say that P is an a, b-path if $p_0 = a$ and $p_l = b$. Furthermore, for $p_i, p_j \in V(P)$ we write

$$Pp_i = p_0 p_1 \dots p_i, \ p_i P = p_i \dots p_l, \ p_i P p_j = p_i \dots p_j$$

and for $u \in \mathcal{N}^+(p_l) \setminus V(P)$, we write $Pu = p_0 p_1 \dots p_l u$. We denote

$$\mathcal{P}_G = \{P : P \text{ is a simple path in } G\}, \quad \mathcal{P}_{G,s} = \{P : P \text{ is a simple path in } G \text{ starting at } s\} \subseteq \mathcal{P}_G$$

and

$$\mathcal{P}_{G,s,d} = \{P : P \text{ is a simple } s, d\text{-path in } G\} \subseteq \mathcal{P}_G$$

We say that two simple paths $P = sp_1 \dots p_l d$ and $Q = sq_1 \dots q_{l'} d$ are *node-disjoint s, d-paths* if $V(P) \cap V(Q) = \{s, d\}$. We say that k simple paths are node-disjoint s, d-paths, if they are pairwise node-disjoint s, d-paths.

We say that two simple paths $P = sp_1 \dots p_l d$ and $Q = sq_1 \dots q_{l'} d$ are *arc-disjoint s, d-paths* if $A(P) \cap A(Q) = \emptyset$. We say that k simple paths are arc-disjoint s, d-paths, if they are pairwise arc-disjoint s, d-paths.

Note that node-disjointness implies arc-disjointness, but that the reverse does not hold. There exists a natural relation between the undirected graph $G = (V, E)$ and the directed graph $G = (V, A)$, if we take

$$A = \bigcup_{\{u,v\} \in E} \{(u, v), (v, u)\}.$$

We call $G = (V, A)$ the directed version of G. Note that P is a simple path in $G = (V, E)$ if and only if P a simple path in $G = (V, A)$ (for their respective definitions). These will resp. be called the undirected and directed version of P. We will now explain what an augmenting path is.

Definition 8.1.1. *Let $G = (V, E)$ be an undirected graph and let $G = (V, A)$ be its directed version. Let $P_0 = s \dots d$ be a simple s, d-path in $G = (V, A)$. We define $G' = (V, A')$ as the directed graph with*

$$A' = A(G) \setminus A(P_0).$$

Then we call a simple path $P' = sp'_1 \dots d$ in G' an augmenting s, d-path for the path P_0. Note that P' is also a path in $G = (V, A)$ and its undirected version a path in $G = (V, E)$. We say that P' is an augmenting path in the undirected graph $G = (V, E)$ if it is an augmenting path in $G = (V, A)$.

Intuitively, if two paths P_0 and P' in $G = (V, E)$ use the same edge $\{u, v\}$ and u occurs before v on P_0, then v must occur before u on P'. We then say that P_0 uses the edge $\{u, v\}$ from u to v and P' uses the edge $\{u, v\}$ from v to u. It is possible to construct two new s, d-paths using a path P_0 and an augmenting path P'.

Example 8.1.1. Consider the graph in Figure 8.1, i.e., the graph $G = (V, A)$ for $G = (V, E)$ as in Figure 6.2, and let $P_0 = sv_1v_4d$.

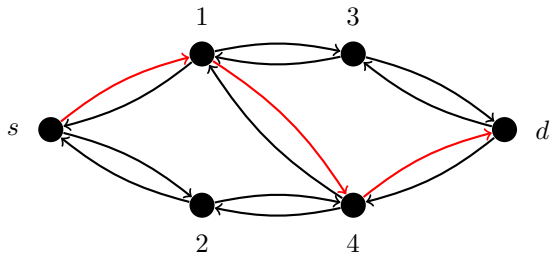

Figure 8.1: The graph $G = (V, A)$ for the graph $G = (V, E)$ in Figure 6.2.

Then we have $G' = (V, A')$ as given in Figure 8.2. Note that, for example, $P' = sv_2v_4v_1v_4d$ is an augmenting path in G'. Furthermore, the symmetric difference (see Definition 8.1.2) is given by the paths $Q = sv_1v_2d$ and $Q' = sv_2v_4d$.

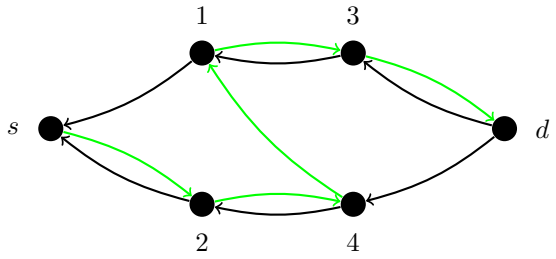

Figure 8.2: The graph G' for the graph $G = (V, A)$ in Figure 8.1, where $P_0 = sv_1v_4d$.

□

Definition 8.1.2. *Let $G = (V, E)$ be an undirected graph and let P_0 be simple s, d-path in G. Let P' be an augmenting s, d-path for P_0. Then the symmetric difference of P_0 and P' is defined as the graph $S = S(P_0, P')$ where*

$$V(S) = V(P_0) \cup V(P')$$

and

$$E(S) = E(P_0) \ \Delta \ E(P').$$

It can be shown that the component of S, containing s and d, is the union of two edge-disjoint simple s, d-paths Q and Q'. We refer to paths the Q and Q' as the symmetric difference of P_0 and P', i.e., for some choice of Q and Q'. The union of these paths is unique, but the paths themselves do not have to be, if $|V(Q) \cap V(Q')| \geq 3$ (if they have more nodes in common than just s and d).

8.2 Problem description

Let $G = (V, E)$ be an undirected graph, $c : E \to \mathbb{R}$ a cost function on the edges of G and let P_0 be a simple s, d-path. We want to solve the problem

$$\min\{c(P) : P \text{ is a simple augmenting } s, d\text{-path for } P_0\}. \tag{8.1}$$

In order to solve this problem, we can use the same approach as in the first section. We can enumerate over all simple paths P and select an augmenting path that minimizes the objective. In order to reduce the size of the branching tree, we can again use exclusion rules. For most of the exclusion rules given so far, there exist quite simple extensions so that they can also serve to find an augmenting path. We also need to add an exclusion rule that guarantees that we do not use an edge $\{u, v\}$ from u to v, if P_0 already uses the edge in that direction. For sake of mathematical completeness, we will adjust some of the basic definitions to the problem of finding an augmenting path. First, we will present the adjusted definition of an information mapping.

Definition 8.2.1. *An information mapping ϕ is a function that maps every tuple (s, d, P, G, c, P_0) to a subgraph of $G = (V, E)$, i.e.*

$$\phi((s, d, P, G, c, P_0)) = G_{P,\phi}$$

where $P = s \dots p_l \in \mathcal{P}_{G,s}$, c a cost function on E, $d \in V$, and P_0 a simple s, d-path in G. Furthermore $G_{P,\phi} = G_P = (V_P, E_P)$ must satisfy the following conditions:

$$V(P) \cup \mathcal{N}(p_l) \cup \{d\} \cup V(P_0) \subseteq V_P \quad \text{and} \quad E(P) \cup \{\{p_l, q\} : q \in \mathcal{N}(p_l)\} \cup E(P_0) \subseteq E_P.$$

We denote the trivial mapping as the mapping defined by

$$V_P = V(P) \cup \mathcal{N}(p_l) \cup \{d\} \cup V(P_0) \text{ and } E_P = E(P) \cup \{\{p_l, q\} : q \in \mathcal{N}(p_l)\} \cup E(P_0).$$

This definition is similar to the original one, with the extension that we also include the path P_0 in G_P. Roughly speaking, G_P now contains the information that may be used to determine whether or not P can be an augmenting path for the path P_0.

Example 8.2.1. The information mapping ϕ we are interested in is $G_P = (V_P, E_P)$ defined by

$$V_P = \{d\} \cup \left(\bigcup_{y \in V(P)} \mathcal{N}(y) \right) \cup V(P_0) \quad \text{and} \quad E_P = \left(\bigcup_{y \in V(P)} \{\{x, y\} : x \in \mathcal{N}(y)\} \right) \cup E(P_0).$$

We will refer to this mapping as the *neighbour mapping* $\phi_{\mathcal{N}}$. Consider the graph G from Figure 4.2.1 and let $P_0 = sv_1v_3d$. Then for the path $P = sv_2$, we have G_P as in Figure 8.3.

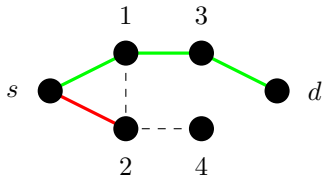

Figure 8.3: Example of graph G_P for $P_0 = sv_1v_3d$, $P = sv_2$ and $G = (V, E)$ as in Example 4.1.

□

We will also give the adjusted definition of an exclusion rule in this context.

Definition 8.2.2. *An exclusion rule R is a function that maps every tuple (s, d, P, G, c, P_0) to a set*

$$X = X_{(s,d,P,G,c,P_0,R)} \subseteq \mathcal{N}(p_l).$$

Here we have, $P \in \mathcal{P}_{G,s}$, c a cost function on E, $d \in V$ and P_0 a simple s,d-path in G. A set of exclusion rules $\mathcal{R} = \mathcal{R}(R_1, \ldots, R_r)$ is defined as the function that maps (s, d, P, G, c, P_0) to the union of the subsets $X_1, \ldots, X_r$. If $v \in V(P_0)$, we denote v^0 as the (unique) node that successes v on P_0, i.e., the node v^0 such that P_0 uses the edge $\{v, v^0\}$ from v to v^0. We call a set of exclusion rules good if

$$\mathcal{N}^+(P_0p_l) \cup (\mathcal{N}(p) \cap V(P)) \subseteq \bigcup_{i=1,\ldots,r} X_{(s,d,P,G,c,P_0,R_i)},$$

where

$$\mathcal{N}^+(P_0p_l) = \begin{cases} \{p_l^0\} & \text{If } p_l \in V(P_0) \\ \emptyset & \text{Otherwise} \end{cases}$$

We say that two sets of exclusion rules $\mathcal{R}(R_1, \ldots, R_r)$ and $\mathcal{R}'(R'_1, \ldots, R'_{r'})$ are equivalent if

$$\bigcup_{i=1,\ldots,r} X_i = \bigcup_{i=1,\ldots,r'} X'_i$$

for every (s, d, P, G, c, P_0).

For a fixed graph $G = (V, E)$ and s,d-path P_0, we want to apply the exclusion rules to the tuples (s, d, P, G_P, c_P, P_0). In order to guarantee that a set of exclusion rules is good, we can use the loop free rule together Exclusion Rule 8.2.1. Note that these rules make sure that all the paths in the branching tree will be simple augmenting paths for P_0.

Exclusion Rule 8.2.1 (Augmenting rule)**.** This is the exclusion rule that maps (s, d, P, G, c, P_0) to

$$X = \mathcal{N}^+(P_0p_l).$$

Here P_0 is a simple s,d-path in G and $P = s \ldots p_l$ a simple path in G. □

Furthermore, there are also natural extensions to the augmented case for the unnecessary node, destination and shortcut rule by adding the extra conditions that the paths Pxw, Pxd and Pw_1ww_2, in their respective definitions, must be augmenting paths for P_0 (see Definition 8.1.1). In the context of an augmenting path, if we refer to one of these exclusion rules, then we mean the exclusion rule with its *augmentation* condition. We will now give two examples to clarify these conditions.

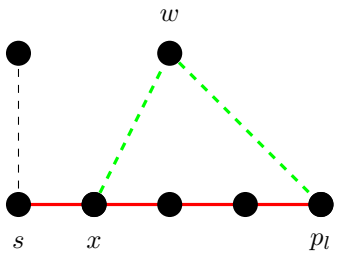

Figure 8.4: Example of augmentation condition for the shortcut rule.

Regarding the shortcut rule, we are not allowed to use a shorcut via w, if P_0 uses the edge $\{x, w\}$ from x to w, or the edge $\{w, p_l\}$ from w to p_l. This holds for the reason that if P_0 would use one of those edges in the mentioned direction, then $Pxwp_l$ is not an augmenting path.

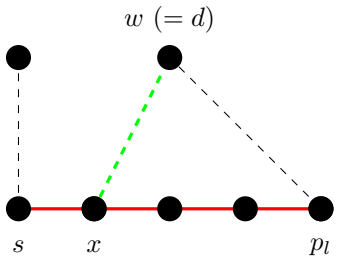

Figure 8.5: Example of augmentation condition for the unnecessary node and destination rule.

Similarly, we are not allowed to use the unnecessary node rule (or destination rule) if P_0 uses the edge $\{x, w\}$ from x to w, because then the path Pxw is not an augmenting path.

8.3 Optimal pair of edge-disjoint paths

As described in Section 2.2, an optimal pair of edge-disjoint s, d-paths is the symmetric difference of a shortest path P_0, for the tuple (G, c, s, d), and a shortest augmenting path P', for the tuple (G, c', s, d) where $c' : E \to \mathbb{R}$ is defined by

$$c'_{uv} = \begin{cases} -c_{uv} & \text{If } \{v, u\} \in E(P_0) \\ c_{uv} & \text{Otherwise} \end{cases}.$$

The fact that P_0 is a shortest path in G implies that G' does not contain cycles of negative cost. This means that we can use the loop free, unnecessary node, destination, shortcut, and augmenting rule (as in Exclusion Rules 5.1.1, 5.1.2, 5.5.1, 5.5.2 and 8.2.1) in order to find a shortest augmenting path. However, it turns out that in this case, we can also use the Exclusion Rules 5.1.1-5.1.5 and 8.2.1 (with augmentation conditions).

As explained in Section 5.5, if we have a shortcut via w, then we can exclude $\mathcal{N}(p_l) \setminus \{w\}$. Furthermore, if $c_{wp_l} \geq 0$, then the shortcut via w implies that we can exclude w based on the unnecessary node rule. However, here we might have $c_{wp_l} < 0$, which is the case if P_0 uses $\{w, p_l\}$. Since the path Pw_1wp_l must be an augmenting path, this means that w will be excluded based on the augmenting rule. It then follows that Exclusion Rule 5.1.4 is still feasible in this case. The benefit of being able to define the shortcut rule in this way will follow from Chapter 9, where we will give a distributed implementation to construct the branching tree.

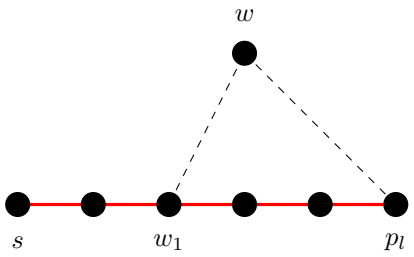

Figure 8.6: Sketch of G where the red edges might contain more nodes

We can also still use the destination rule (with augmentation condition). A schematic proof of this statement is given in Appendix A. This means that we can use Exclusion Rules 5.1.1-5.1.5 and 8.2.1 (with augmentation conditions), in order to solve the augmenting path problem. The benefit of this is that we only have to make a few adjustments to the distributed implementation (for constructing the branching tree) of the shortest path problem. Furthermore, the exclusion rules for the single path problem perform better (in terms of size of the branching tree) than the multipath rules, meaning that it is beneficial to use an augmenting path approach to find multiple disjoint paths.

8.4 Concluding remarks

In this chapter we have formulated the appropriate framework in order to use augmenting paths for finding multiple (edge-)disjoint paths. The benefit of this is that the we can use the single path approach for finding these paths. In Chapter 9 we will see that the single path problem can be solved more efficient than the multipath problem.

We have also shown that we can use the same exclusion rules for the augmenting path problem, as for the original single path problem (when looking for a shortest augmenting path), which means we only need to make small adjustments to the distributed implementation for constructing the branching tree in the single path case.

There are actually two approaches for finding two optimal edge-disjoint paths. Since G is undirected, any shortest s, d-path is also a shortest d, s-path. Instead of looking for an augmenting s, d-path, we can also look for an augmenting d, s-path. This is more natural from a practical point of view.

It is also possible to use augmenting paths in order to directly find a node-disjoint path, but this is much harder to implement, and will probably not be beneficial. Moreover, in this case we might need augmenting paths that are not simple, but those do not fit into our framework.

Chapter 9

Constructing the branching tree

In this chapter we will describe a method for constructing the branching tree in order to solve (4.1) for a given tuple (G, c, s, d). We will do this in such a way that it can be implemented as a distributed algorithm. Nodes receive messages that correspond with tuples

$$(s, d, P, G_p, c_P).$$

Based on a set of exclusion rules, a node v determines to which neighbours it wants to forward the updated message

$$(s, d, Pv, G_{Pv}, c_{Pv}),$$

i.e., to which neighbours it wants to extend the path P. We will use the *equivalent information mapping* $\phi_{\mathcal{N}^*}$ as definition for G_P (see end of Section 5.4). Messages corresponding to s, d-paths will arrive at the destination node d, that can then select k disjoint paths minimizing (4.1). For a node $v \neq d$, every message is processed independently of the other messages arriving at that node. Furthermore, a node does not store information from an incoming message, i.e., the nodes are *stateless* with respect to incoming messages. As soon as a message has been processed and forwarded, it forgets that it was ever there. Throughout this chapter, the terms 'message' and 'packet' are used interchangeably.

Example 9.0.1. Consider the following graph with $c_{uv} = 1$ for all $\{u, v\} \in E$.

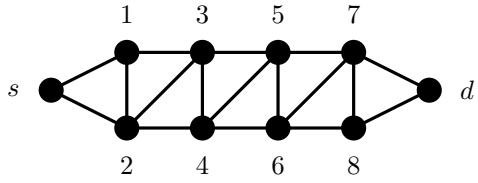

Figure 9.1: Example of graph G.

Suppose that node v_4 receives the message corresponding to the path $P = sv_2$. A schematic overview of the graph G_P, as defined by $\phi_{\mathcal{N}^*}$, is given in Figure 9.2.

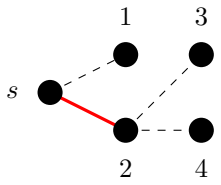

Figure 9.2: Incoming message at node v_4.

The updated message can be found in Figure 9.3.

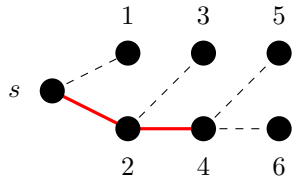

Figure 9.3: Updated message at node v_4.

□

In general, for other choices of $\mathcal{R}$ and ϕ, we have to satisfy the following assumptions.

Assumptions 9.0.1.

i) All nodes use the same (fixed) set of exclusion rules $\mathcal{R}$.

ii) All nodes update an incoming messages using the same (fixed) information mapping ϕ.

iii) The only information about G available at v, is the subgraph $H = (V', E')$ defined by

$$V'(H) = \{v\} \cup \mathcal{N}(v) \quad \text{and} \quad E'(H) = \{\{v, w\} : w \in \mathcal{N}(v)\}$$

Note that the third condition restricts the choice of ϕ in the second condition. In relation to the branching tree T, an incoming message (s, d, P, G_P, c_P) at v corresponds to the arc

$$(P, Pv) \in A(T(s, d, G, c, \phi, \mathcal{R})),$$

which means that the total number of messages that will be sent through the network, is equal to $|A(T(s, d, G, c, \phi, \mathcal{R}))|$. In the next sections, we will give an implementation of the protocol that a node v performs when receiving a message. We have analysed the algorithm for different types of graphs, by determining the number of messages that has to be sent. We made a distinction between the single path and multipath case.

9.1 Single path problem

9.1.1 Implementation

We will first explain the implementation in Matlab. We implemented the loop free, unnecessary node, destination and shortcut rule for arbitrary graphs G and non-negative cost functions c. Remember that for the information mapping $\phi_{\mathcal{N}^*}$, the graph G_P is defined by

$$V_P = \{d\} \cup \bigcup_{y \in V(P)} \mathcal{N}(y) \quad \text{and} \quad E_P = E(P) \cup \{\{q, p_l\} : q \in \mathcal{N}(p_l)\} \bigcup_{w \in V_P \setminus (V(P) \cup \{d\})} \{w, w*\},$$

with $w^* \in V(P) \setminus \{p_l\}$ the first node on P that minimizes

$$\min\{c(s \dots x) + c_{xw} : x \in V(P) \cap \mathcal{N}(w)\}.$$

This means that if there is an incoming message (s, d, P, G_P, c_P) at node v and

$$w \in (V_P \setminus (V(P) \cup \{d\})) \cap \mathcal{N}(v)$$

is such that

$$c(Pv) + c_{vw} < \min\{c(s \dots x) + c_{xw} : x \in V(P) \cap \mathcal{N}(w)\},$$

then we will delete the edge corresponding to the minimum and add the edge $\{v, w\}$ with its cost (if w is not yet reachable from P, we also add the edge $\{v, w\}$). We will now first describe how an incoming message is modelled, and then how it is updated.

Every incoming message at a node v is represented as a tree. This tree consists of the path P and for every reachable neighbour, the minimizing (as described above) edge to the path (which is added along the way). For example, for the incoming message from node v_4 to v_6 at Figure 9.3, we only have stored the following information up till v_4.

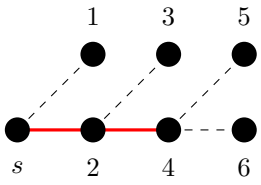

Figure 9.4: Incoming message at node v_6 (or v_5). All costs are equal to one.

We say that a node $x \in P$ lies on the i-th level, if it is the i-th node on the path (so s is the first node). We then say that a reachable neighbour w is on level i of the tree if the node x, such that $\{x, w\}$ is stored, is the $(i-1)$th node on the path. In Figure 9.4, node s lies on level one, v_1 and v_2 on level two, nodes v_3 and v_4 on level three, and nodes v_5 and v_6 on level four. We represent every node $z \in V(G_P)$ with a tuple

$$\big(\text{level},\ z, c(s \dots z)\big)$$

For example, $(1, s, 0)$ for node s, and $(3, v_3, 2)$ for node v_3. Then we can represent the tree as the sequence of tuples, ordered according to the levels of the nodes. Furthermore, the first tuple of a certain level in the sequence corresponds to the node on the path of that level. For Figure 9.4, the sequence becomes

$$(1, s, 0)(2, v_2, 1)(2, v_1, 1)(3, v_4, 2)(3, v_3, 2)(4, v_5, 3)(4, v_6, 3).$$

The nodes of the last level are not in the right order for all the receiving nodes. This is due to the fact that the node v ($v = v_4$ in this case) sends the same updated message to all its neighbours. So the first step when receiving a message is to change the order of the tuples of the last level so that the receiving node places itself as the first tuple of the last level. In our example, v_6 interchanges the tuples for v_5 and v_6, resulting in

$$(1, s, 0)(2, v_2, 1)(2, v_1, 1)(3, v_4, 2)(3, v_3, 2)(4, v_6, 3)(4, v_5, 3).$$

This can also be considered as saying that the node 'adds itself to the path', a phrase that is used throughout this report. We also have some additional information that is stored in every message (that will be updated along the way), namely

$$\big(d, r_d, c_d\big)$$

where

d	:	destination
r_d	:	binary variable that is 1 if destination is reachable from P and, 0 otherwise
c_d	:	cost of reaching the destination, if $r_d = 1$ (and otherwise $c_d = 0$)

This tuple will be placed at the beginning of the message, so the incoming message in Figure 9.4 is modelled as

$$\big(d, 0, 0\big)(1, s, 0)(2, v_2, 1)(2, v_1, 1)(3, v_4, 2)(3, v_3, 2)(4, v_6, 3)(4, v_5, 3).$$

A high-level description of the updating procedure is given in Algorithm 9.1.1. In Step 3, we use the variable r_d to combine the unnecessary node rule with the destination rule. Note that not

all the stored information is necessary. For the nodes on the path, we never use the aggregated cost, only for the last node v. Since we want to find quantitative results regarding the number of messages we have to send, minimizing the amount of information per message is not the priority here. For the sake of implementation in Matlab, we decided to add this information as well.

Algorithm 9.1.1. *Protocol executed by node v when receiving message $T = (G_P, c_P)$.*

Data:

$N(v)$	:	$\{w : \{v, w\} \in E(G)\}$
$C(v)$	:	$\{c_{vw} : w \in N(v)\}$
$N'(v)$	:	*List of neighbours we are going to send an updated message ($N(v)$ at start)*
c_i	:	*$c(s \dots i)$ i.e., the aggregated cost of the path to node i in G_{Pv} for all $i \in V(G_{Pv})$*

Step 1: Add v to the path

Order the tuples on the last level so that the tuple of v is the first tuple of that level

Step 2: Check if destination is reachable and make the necessary adjustments.

if $v = d$ ***then***
  STOP, do not forward message to any neighbour.
else if $d \in N(v)$ ***then***
  if *d is not yet reachable (i.e. $r_d = 0$)* ***then***
    Set $r_d = 1$, $c_d = c_v + c_{vd}$
  else if *d is reachable (i.e. $r_d = 1$) and $c_v + c_{vd} < c_d$* ***then***
    Set $c_d = c_v + c_{vd}$.
  else
    Set $N'(v) = N'(v) \setminus \{d\}$.
  end if
  $N(v) = N(v) \setminus \{d\}$.
end if

Step 3: Iterate over all neighbours and perform the necessary actions.

while $N(v) \neq \emptyset$ ***do***
  Take $w \in N(v)$.
  if *There is a tuple (l_w, w, c_w) in T* ***then***
    if *(l_w, w, c_w) is the first tuple of level l_w* ***then***
      Apply loop free rule: $N'(v) = N'(v) \setminus \{w\}$.
    else
      if $c_w + c_{wv} < c_v$ ***then***
        Apply shortcut rule (ineq.): STOP, do not forward message to any neighbour.
      else if $c_w + c_{wv} = c_v$ *and* $l_v - l_w \geq 2$ ***then***
        Apply shortcut rule (eq.): STOP, do not forward message to any neighbour.
      else if $c_v + c_{vw} \geq \min\{c_w, (1 - r_d)c_w + r_d c_d\}$ ***then***
        Apply unnecessary node/dest. rule: Set $N'(v) = N'(v) \setminus \{w\}$.
      else
        Delete tuple (l_w, w, c_w) and add tuple $(l_v + 1, w, c_v + c_{vw})$ at the end of T.
      end if
    end if
  else
    if $c_v + c_{vw} \geq (1 - r_d)(c_v + c_{vw} - 1) + r_d c_d$ ***then***
      Apply destination rule: Set $N'(v) = N'(v) \setminus \{w\}$.
    else
      Add tuple $(l_v + 1, w, c_v + c_{vw})$ at the end of T.
    end if
  end if
  $N(v) = N(v) \setminus \{w\}$
end while

Step 4: Send updated message to all neighbours in $N'(v)$.

9.1.2 Surfnet topology

In this section we will analyse the distributed algorithm using the graph in Figure 9.5, the topology of the Dutch company Surfnet. We have left out any redundant edges between adjacent nodes and only look at locations in The Netherlands. The resulting graph then consists of $|V| = 59$ nodes and $|E| = 83$ edges.

Figure 9.5: Abstract topology of Surfnet network.

In Figure 9.6 we have given a histogram of the number of messages, i.e., the size of the branching tree, for the s, d-pairs in the graph of Figure 9.5, using a constant positive cost function for the edges. On average, we need to send $\mu = 192$ messages through the network per s, d-pair (with a standard deviation of $\sigma = 110$) . Note that there is a high peak at the left, due to the fact that for adjacent s, d-pairs, we only send one message. For fixed source s and destination d, we can define the number M_v that represents the number of times a path is being extended to a given node v, i.e., the number of messages that v receives. We then define

$$M_{s,d} = \max_{v \in V} M_v$$

as the maximum number of messages an individual node receives when searching for an s, d-path. In Figure 9.7 we have given a histogram of the $M_{s,d}$ for all s, d-pairs.

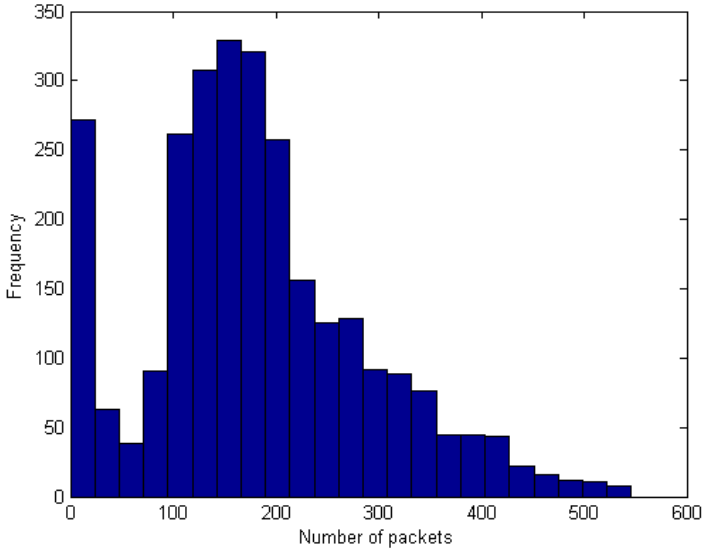

Figure 9.6: Histogram of number of messages for all s, d-pairs.

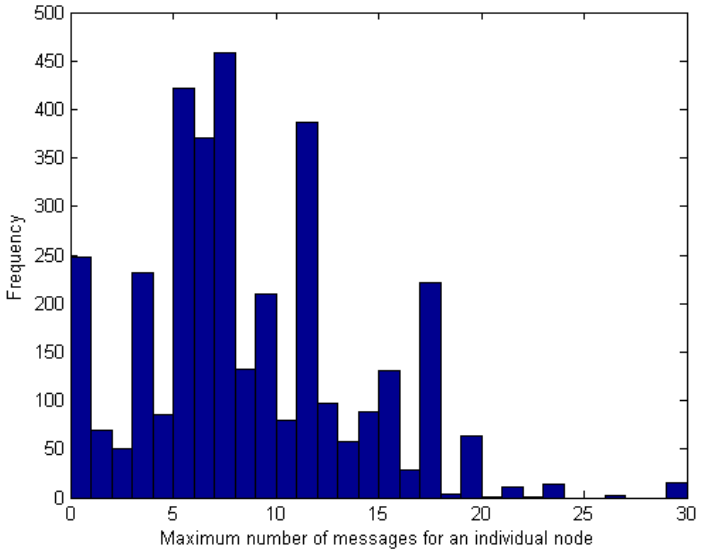

Figure 9.7: Histogram of $M_{s,d}$ values for all s, d-pairs.

From this, it can be seen that there are s, d-pairs so that an individual node might receive up to 30 messages, which is quite a lot. In order to reduce the amount of message, we can break ties, as explained in Section 5.3. For every $\{u, v\} \in E$, we introduce the stochastic variable $\varepsilon_{uv} \sim U(0, 1/100)$, and define

$$c_{uv} = 1 + \varepsilon_{uv}.$$

In Figure 9.8, we have given a histogram of the number of messages for all s, d-pairs for a realization of the variables ε_{uv}. On average, we have $\mu = 162$ messages per s, d-pair, which is a reduction of 16% with respect to the average without perturbation. We have a standard deviation of $\sigma = 85$, which is also a reduction with respect to the standard deviation without perturbation (both absolute and relative to the average).

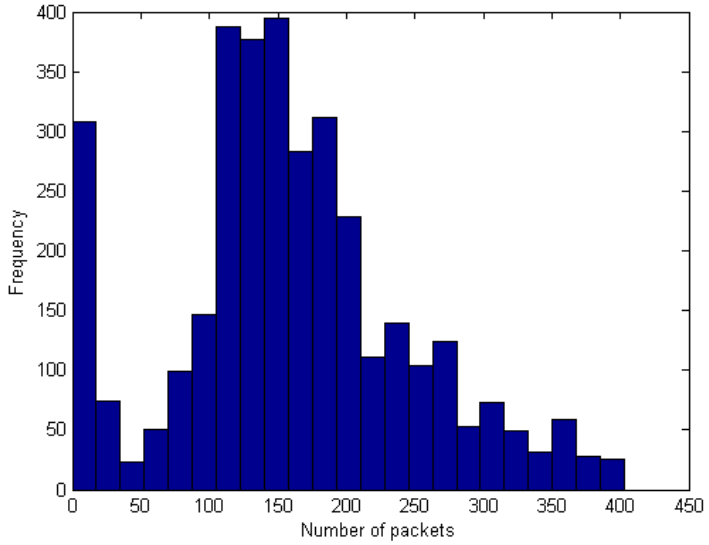

Figure 9.8: Histogram of number of messages for all s, d-pairs (with perturbation).

In Figure 9.9 we have given a histogram of the $M_{s,d}$ values. It can be seen that the maximum value of $M_{s,d}$ is 17 (as opposed to 30 for the case without perturbation). Although this is still a high number, it is a reduction of 43%. Not all the perturbations give results as good as this one, but from it, we can conclude that breaking ties can lead to a significant reduction in both the number of messages that has to be sent, as well as the maximum number of messages that an individual node receives.

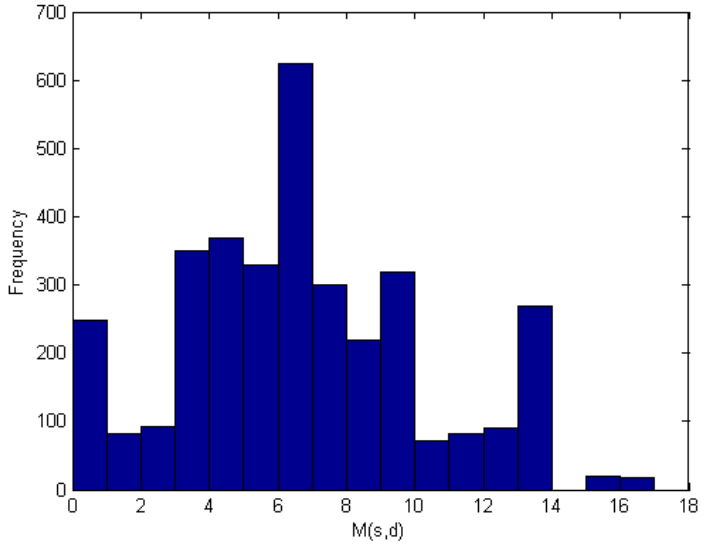

Figure 9.9: Histogram of $M_{s,d}$ values for all s, d-pairs (with perturbation).

9.1.3 Randomly distributed nodes with Euclidean cost function

In this section we will consider graphs of the following type. We let $V \subseteq [0,a] \times [0,b]$ be a finite set, and for nodes $x, y \in V$ define $\{x,y\} \in E$ if and only if

$$c_{xy} = \|x - y\|_2 \leq R$$

for some fixed $R > 0$, the transmission range. Here, $\|x\|_2 = \sqrt{x_1^2 + x_2^2}$ for $x = (x_1, x_2) \in \mathbb{R}^2$, i.e., the Euclidean distance norm. An example is given in Figure 9.10.

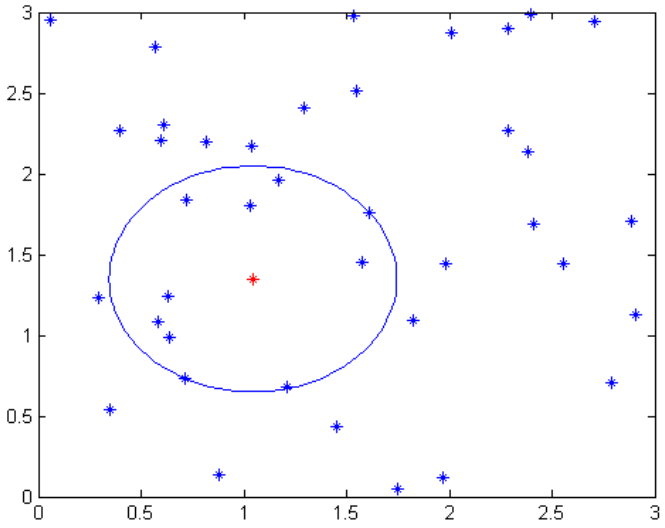

Figure 9.10: Example of $G = (V, E)$ where $n = 40$, $a = b = 3$ and $R = 0.7$. The red node gives an example of the transmission range.

Such a graph can, for example, represent a mobile ad-hoc network. In these kind of communication networks, information is being *broadcast* using a physical signal, that can be received by all nodes within the transmission range. Here we have the problem that all nodes within the transmission range will receive an updated message, although it might not be meant for every neighbour. This can be solved as follows. When we update a message, the neighbours to which we want to extend have a tuple on the last level of the updated message (by construction, see Section 9.1.1). This means that, whenever a node receives a message, it first has to check whether or not it is contained in a tuple on the last level. If this is not the case, a node can drop (not process) the message. We will only count the broadcasts in the branching tree, i.e., whenever a node extends a path to one or more neighbours, this will be counted as one message. There is a simple relation between the $|A(T(G, c, s, d)|$ and the number of broadcasts $B(G, c, s, d)$, namely

$$B(G, c, s, d) = |A(T)| - |\mathcal{E}(T)|$$

where $\mathcal{E}(T)$ is the set of end nodes of the branching tree T, i.e., the nodes with out-degree zero. In order to create graphs of this type, we randomly select n points from $[0,a] \times [0,b]$ and determine the number of broadcasts for all s,d-pairs in the graph. We will then analyse the performance of the graph using the measure

$$B(G, c) = \max_{s,d \in V} B(G, c, s, d)$$

Note that $B(G, c)$ can be interpreted as the number of broadcasts for a worst-case s,d-pair. In Figure 9.11, we have given a histogram of the values $B(G, c)$ for 500 realizations of the graph explained above, by randomly selecting n points from $[0,a] \times [0,b]$. We only consider realizations for which the resulting graph $G = (V, E)$ is connected. We do not want to run the algorithm on

a disconnected graph, because that would mean we are just sending messages within components, which will probably give better results in terms of messages, although no paths are actually found. In Figure 9.12, we have given a histogram of the values

$$\bar{B}(G,c) = \frac{1}{n(n-1)} \sum_{s,d \in V} B(G,s,c,d),$$

i.e., the average number of broadcasts for all pairs within one realization. Although this is not a good measure to judge the performance for a certain realization, it does show that the results in Figure 9.11 hold for rather extreme cases, instead of all s,d-pairs. In order to give a complete analysis of these types of graphs, it is necessary to do simulations for other values of a, b, n and R, in order to determine how the algorithm scales with respect to these variables.

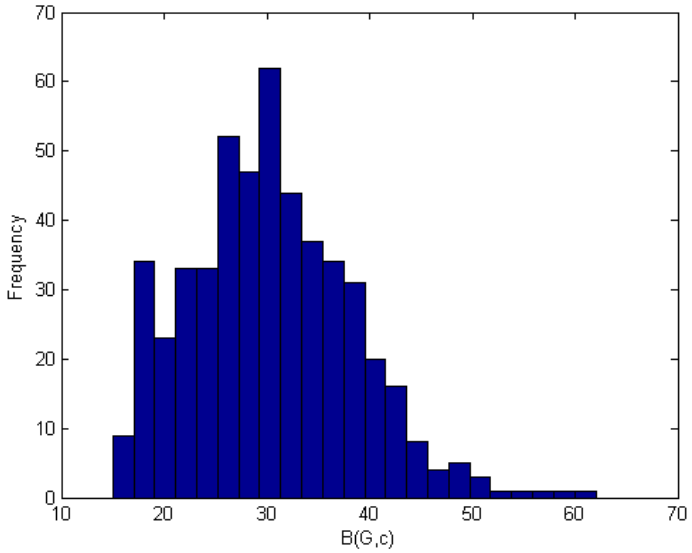

Figure 9.11: Histogram of $B(G,c)$ for 500 realizations with $n = 25,\ a = b = 2$ and $R = 0.6$.

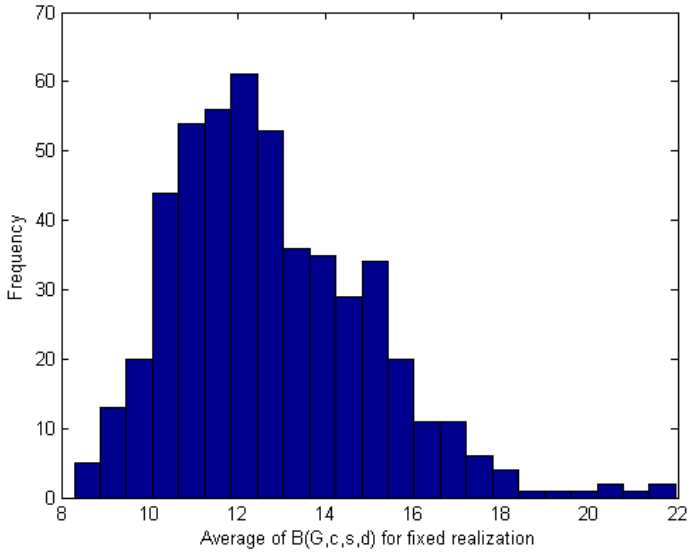

Figure 9.12: Histogram of $\bar{B}(G,c)$ for 500 realizations with $n = 25,\ a = b = 2$ and $R = 0.6$.

9.2 Multipath problem

We have used the same implementation as for the single path case. Furthermore, we have used a constant cost function defined by $c_{uv} = 1$ for all $\{u, v\} \in E$. For the first two examples, we have left out the shortcut rule, for the simple reason that it cannot be applied there (which is also in line with the worst-case scenarios concerning cost functions that satisfy the cycle-inequality). The third example illustrates the double shortcut rule (see Lemma 6.1.1).

9.2.1 Complete graph

The first example that we consider is that of a complete graph of n nodes, together with a source and destination attached to some nodes of the complete graph (which gives a total of $n+2$ nodes). We can denote this graph by $G = (V, E)$ where

$$V = \{1, 2, \ldots, n, s, d\}$$

is the set of nodes, and

$$E = \{s, s_1\} \cup \{s, s_2\} \cup \{d, d_1\} \cup \{d, d_2\} \cup \{\{i, j\} : 1 \leq i < j \leq n\}$$

the set of edges, where $s_1, s_2, d_1, d_2 \in \{1, \ldots, n\}$ are pairwise distinct. Here, we are interested in solving (4.1) for $k = 2$.

Example 9.2.1. Below we have given an example of G with $n = 6$.

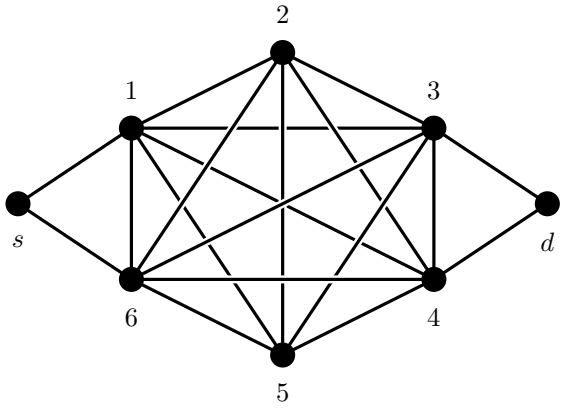

Figure 9.13: Example for $n = 6$ with $s_1 = 1, s_2 = 6, d_1 = 3$ and $d_2 = 4$.

□

Lemma 9.2.1. *Let $\mathcal{G}$ the collection of tuples (G, c, s, d) where G is of the form described above and c a constant non-negative cost function. Furthermore, let $\phi = \phi_{\mathcal{N}}$ and $\mathcal{R}$ consist of the loop free and unnecessary node rule. Then we have*

$$|A(T(s, d, G, c, \phi_{\mathcal{N}}, \mathcal{R})| = 2(n + 1)$$

*and, for the neighbour mapping $\phi_{\mathcal{N}}$, that $\mathcal{R}$ is optimal (under Assumption **??**).*

Proof. At first, the path $P = s$ will be extended to both s_1 and s_2 (two arcs in T). Then the path $P = ss_i$ will be extended to the neighbours $\{v_1, \ldots, v_n\} \setminus \{s_1, s_2\}$, based on the unnecessary node rule ($n - 2$ arcs in T) for $i = 1, 2$. After that, both d_i will extend the paths $P = ss_1d_i$ and $P = ss_2d_i$ to node d (two arcs in T) for $i = 1, 2$. This holds because of the unnecessary node rule

(note that we could also use the destination rule for this argument). This means we have a total of

$$2(1 + (n-2) + 2) = 2(n+1)$$

arcs in T. The optimality of $\mathcal{R}$ follows from the fact that the set containing the loop free and unnecessary node rule satisfies Assumption 4.2.1, and the fact that

$$(s, d, P, G_P, c_P) \quad \text{and} \quad (s, d, P', G_{P'}, c_{P'})$$

are isomorphic if two paths $P, P' \in W(T)$ are on the same level of the branching tree. □

From the example in Section 9.2.4, it will follow that the algorithm performs better for dense graphs (i.e., with $|E|$ very high). A complete graph is the extreme situation of this observation, since E is maximal there. This is also an example that shows we can gain a lot with respect to just the loop free rule. For the graph G as described above, applying just the loop free rule would result in a branching tree with

$$|A(T(s, d, G, c, \phi_{\mathcal{N}}, \mathcal{R})| = 2\left[1 + \sum_{i=2}^{n} \frac{(n-1)!}{(n-i)!} + 2\sum_{i=2}^{n} \frac{(n-2)!}{(n-i)!}\right]$$

where $\mathcal{R}$ is then the set of just the loop free rule (we will not prove this formula here). In the figure below, we have given a comparison between different sets of rules. From this it can also be seen that the set of the loop free, unnecessary node and destination rule is equivalent to the set of the loop free and unnecessary node rule.

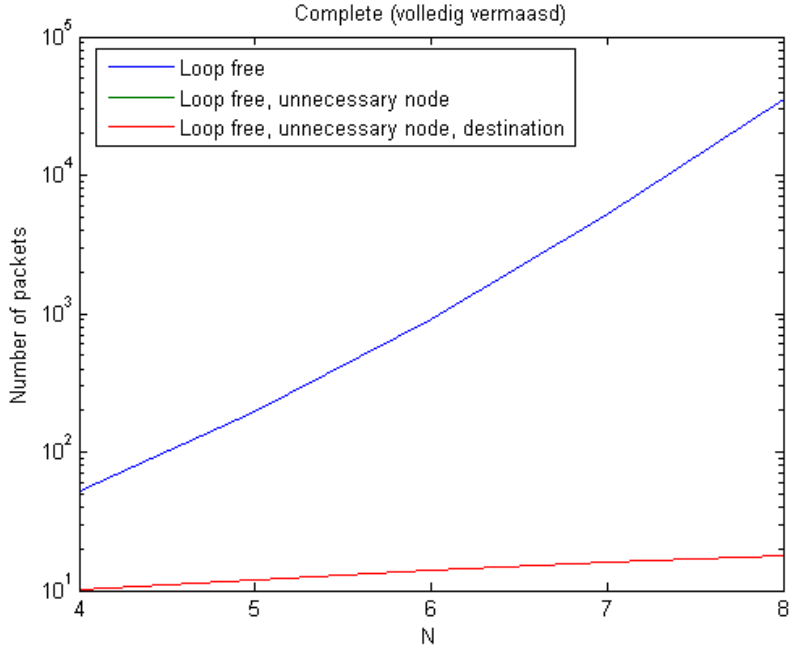

Figure 9.14: Logarithmic plot of different sets of exclusion rules for G as defined in this section.

9.2.2 Chordal graph

The second type of graph is $G = (V, E)$ defined by

$$V = \{1, 2, \ldots, n, s, d\},$$

where $|V| = n + 2 = 2k + 2$, and

$$E = E' \cup \{\{i, j\} : i < j \leq i + 2 \text{ for } i = 1, \ldots, n-2\} \cup \{n-1, n\},$$

with

$$E' = \{s, 1\} \cup \{s, 2\} \cup \{d, n-1\} \cup \{d, n\}.$$

Note that E' contains the edges needed to add the soure and destination node to the graph. For these type of graph, we will present both numerical and analytical results. It turns out that we can give an interesting lower bound on the size of the branching tree in terms of Fibonacci numbers.

Example 9.2.2. Below we have given an example for a graph G with $n = 10$.

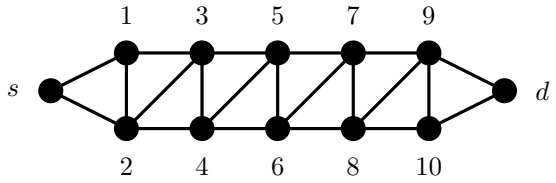

Figure 9.15: Example of G for $n = 10$, which means $k = 5$.

□

Numerical results are presented in the figure below. These results also show that the destination rule does not achieve a significant difference.

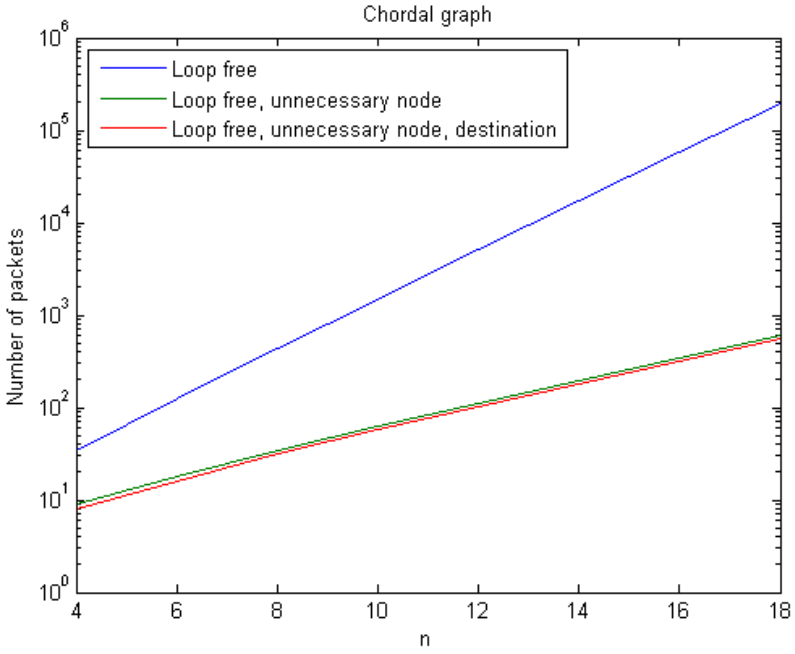

Figure 9.16: Logarithmic plot of different sets of exclusion rules for G as defined in this section.

From Table 9.1, it can be seen that the size of the tree, for the loop free and unnecessary node rule, seems to converge. Note that we only consider n even, which means that the observed complexity can be expressed as

$$\mathcal{O}(1.7549^k) = \mathcal{O}(1.7549^{n/2}) \approx \mathcal{O}(1.32^n).$$

n	22	24	26	28	30	32	34	36	38	40
$\|A(T)\|$	1894	3326	5839	10249	17988	31569	55402	97226	170622	299423
Ratio	-	1.7561	1.7556	1.7553	1.7551	1.7550	1.7549	1.7549	1.7549	1.7549

Table 9.1: Number of messages needed for G.

In Theorem 9.2.2, we give a lower bound on this value.

Theorem 9.2.2. *For G as defined in this section, $\mathcal{R}$ the set consisting of the loop free and unnecessary node rule and $\phi = \phi_{\mathcal{N}}$, we have*

$$|A(T(s, d, G, c, \phi_{\mathcal{N}}, \mathcal{R})| \geq F_{k+5} - 3 \quad \rightarrow \quad \varphi^k \qquad (\approx 1.26^n)$$

as $k \to \infty$. Here, F_{k+5} is the $(k+5)$th Fibonacci number (remember that $n = 2k$) and φ the golden ratio.

Proof (sketch). We will first prove that on level a, we have F_{a+2} nodes of which F_{a+1} nodes branch into two new nodes and F_a nodes branch into one node using induction on a for $a \leq k-1$. Suppose that it holds for some level a, then we have

$$2F_{a+1} + F_a = F_{a+1} + (F_{a+1} + F_a) = F_{a+1} + F_{a+2} = F_{a+3}$$

nodes on level $a+1$. Furthermore, if a node i on level a branches into two new nodes on level $a+1$, then these nodes are $i+1, i+2$. This means that on level $a+1$, the node $i+1$ will only branch into $i+3$ and the node $i+2$ will branch into $i+3, i+4$. If a node i on level a branches into one node, then this node will be $i+2$ and on level $a+1$, this node will then branch into $i+3, i+4$. This means that the amount of nodes on level $a+1$ that branches into two new nodes is

$$F_{a+1} + F_a = F_{a+2}$$

and the amount of nodes that branches into one node is F_{a+1} (we leave out the details of the statements above). This works well for $a \leq k-1$, but for $a = k$ we have one node less, namely n will only branch into d (and not into $n+2$, since that node does not exist). This means that the total amount of messages sent up to level $k+1$ equals

$$-1 + \sum_{i=2}^{k+3} F_i = -2 + \sum_{i=1}^{k+3} F_i = -2 + F_{k+5} - 1 = F_{k+5} - 3$$

using

$$\sum_{i=1}^{n} F_n = F_{n+2} - 1$$

which can easily be proven, using induction on n. We then have

$$|A(T(s, d, G, c, \phi_{\mathcal{N}}, \mathcal{R})| \ \geq \ \sum_{i=0}^{k+1} |L_{T,k}| = F_{k+5} - 3 \quad \rightarrow \quad \varphi^k \approx 1.26^n$$

using $n = 2k$. Here, $\varphi = 1.61..$ is the golden ratio. □

9.2.3 Graph with $\varrho(G) \geq 5$

In this section, we will give an example where the double shortcut rule can be used (see Lemma 6.1.1). For $k \in \mathbb{N}$, we define $G = (V, E)$ by

$$V = \{s, d, 1, \ldots, 4k\}$$

and

$$E = E' \cup \{\{i, j\} : j = i + 2 \text{ for } i = 1, \ldots n - 2\} \cup \{\{i, j\} : j = i + 1 \text{ for } i = 3, 5, 7, 9, \ldots n - 3\},$$

where

$$E' = \{s, 1\} \cup \{s, 2\} \cup \{d, n - 1\} \cup \{d, n\}$$

and $n = 4k$.

Example 9.2.3. A sketch of $G = (V, E)$ is given in the Figure 9.17.

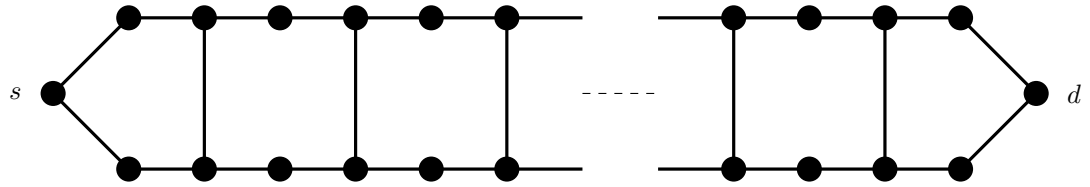

Figure 9.17: Sketch of G as defined in this section.

□

Paths that arrive at the destination node d can have at most one consecutive crenellation, because a path with two consecutive crenellations will be excluded based on the double shortcut rule (using overlapping shortcuts via x and y, see Figure 9.18).

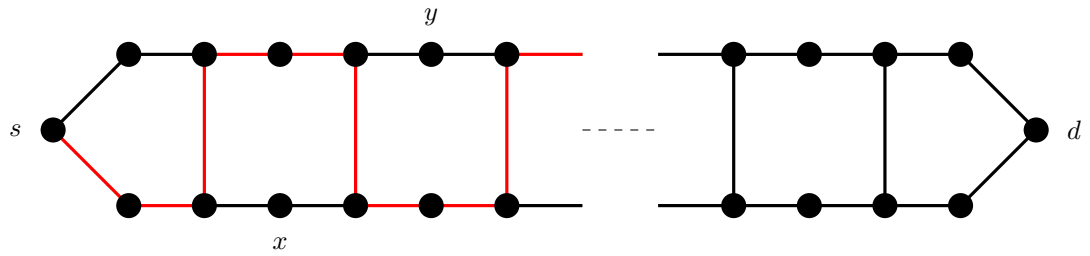

Figure 9.18: Sketch of two consecutive crenellations.

However, if we would only have used the loop free and unnecessary node rule, then a path such as that in Figure 9.18 would not be excluded. Note that this is also an example where the destination rule will never be applied.

9.2.4 Semi-random graphs

In this section we will analyse how the density of a graph G influences the size of the branching tree. We have generated graphs of the following type. We start out with a cycle $G = C_n$ of length n, together with source s and destination d, i.e.,

$$V(G) = \{s, d, 1, 2, \ldots, n\}$$

and

$$E(G) = \{\{i, i+1\} : i = 1, \ldots, n\} \cup \{s, 1\} \cup \{s, n\} \cup \{d, n/2\} \cup \{d, n/2+1\},$$

where $n + 1 \equiv 1$ (i.e., the numbering is modulo n).

Example 9.2.4. An example of the basic graph G for $n = 6$.

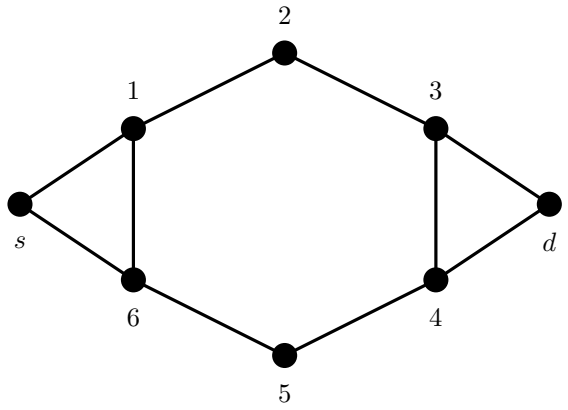

Figure 9.19: Example for $N = 6$ with $s_1 = 1, s_2 = 6, d_1 = 3$ and $d_2 = 4$

□

We then add an edge $\{u, v\}$ for $1 \leq u \neq v \leq n$ with probability $0 \leq p \leq 1$. Note that if $p = 0$, we get the graph described above and if $p = 1$, we get the complete graph described in Section 9.2.1. For a fixed p, we then determine the size of the branching tree for q realizations, using the loop free, unnecessary node and destination rule. Note that the expected number of edges in the graph is

$$\left[\binom{n}{2} - n\right] p + n + 4.$$

In Figure 9.20 we have given the average number of messages for 500 realizations, for every $n \in \{20, 22, 24, 26, 30\}$, with $\Delta p = 0.05$. It can be seen that the algorithm performs better as the number of edges increases (which is equivalent to saying that the average degree of the nodes increases). At first the number of messages that has to be sent increases rapidly as p increases, but after $p \approx 0.2$ the it decreases. This behaviour can be seen as a trade-off between the fact that more edges lead to more messages, but also to more information in the messages that are being sent. If we add an extra edge $\{u, v\}$ for $u, v \in V(G)$, then this means that u and v will also (possibly) start sending messages to each other, which increases the total number of messages, but every message that is being sent from u now also has information about the cost of reaching node v (and vice versa), which might prevent paths (that visited u) are later sent to v.

We also see that the algorithm is more stable for dense graphs than for sparse graphs, in the sense that small changes in the size of $|E|$ have more effect on sparse graphs than on dense graphs. In Figure 9.21 we have given a logarithmic plot of the results in Figure 9.20, that clarifies this behaviour.

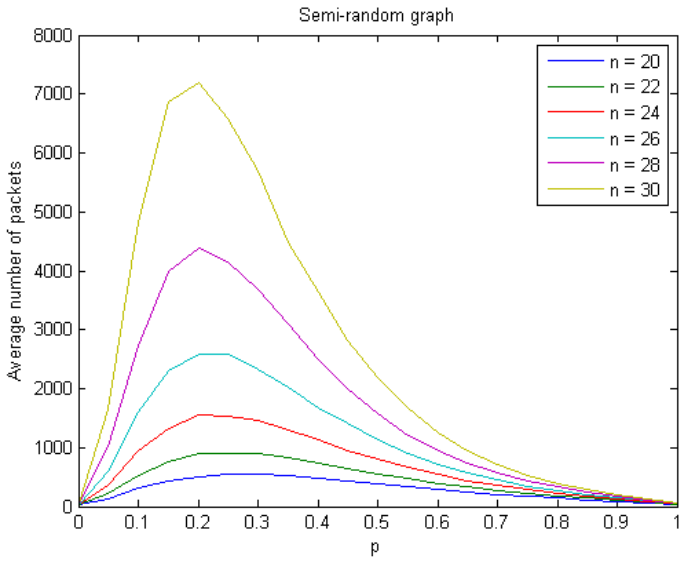

Figure 9.20: Average number of messages per n with $\Delta p = 0.05$, for 500 realizations per n.

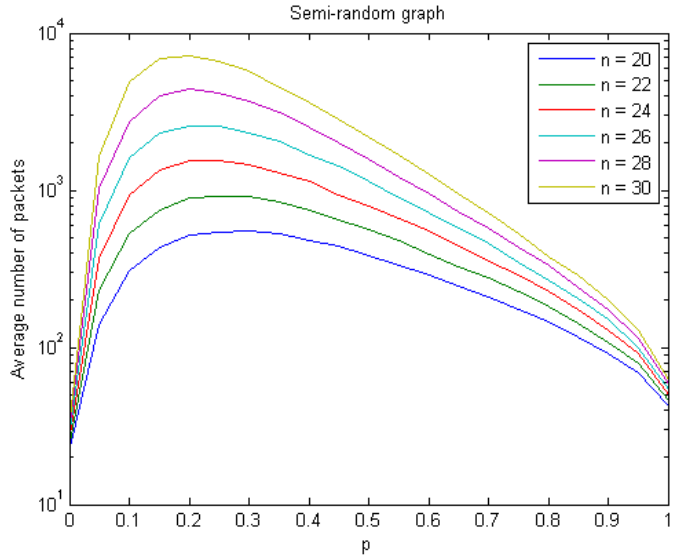

Figure 9.21: Logarithmic plot of the results in Figure 9.20.

9.3 Heuristic approaches

In this section we will discuss and compare some heuristic approaches for finding optimal node-disjoint paths. In the previous sections, we have mostly used the loop free, unnecessary node and destination rule and analysed the size of the branching trees. In general, these rules do not perform very well. To that end, we discuss and compare some heuristic approaches to see how the number of messages can be reduced (while also analysing the loss of optimality and feasibility for these methods). All the methods are based on the single path algorithm. For a fixed graph $G = (V, E)$ and cost function c, we define

$$F_i = \{(s, d) \in V : \text{Method } i \text{ finds two node-disjoint } s, d\text{-paths}\}$$

and then

$$\gamma_i = \frac{|F_i|}{|F_{total}|}$$

where F_{total} is the set of s, d-pairs for which two disjoint paths exist in G. Furthermore, we define

$$\rho_i = \max\{\rho_{s,d} : (s,d) \in F_i \text{ and } OPT_i(s,d) = \rho_{s,d} OPT(s,d)\}$$

where $OPT_i(s,d)$ is the objective value of the best solution found by method i, and $OPT(s,d)$ the objective value of the optimal solution. We also define

$$\mu_i = \frac{1}{|F_{total}|} \sum_{(s,d) \in F_{total}} |A(T_i(G,c,s,d))|$$

which is the average number of messages that has to be sent per feasible s, d-pair. Here, $T_i(G,c,s,d)$ is the branching tree we find when applying method i for a given s, d-pair. Furthermore, we define σ_i as the standard deviation of the numbers $T_i(G,c,s,d)$ and τ_i as the maximum value. We will use the Surfnet topology from Figure 9.5 for this analysis, with a constant non-negative cost function c. We have $F_{total} = 2570$ (found in Method 1). Note that the Surfnet graph is very sparse, which means, intuitively, that it has a low p-value in Figure 9.20.

9.3.1 Method 1: Exact approach (for comparison)

In Figure 9.22 we have given a histogram of the number of messages per s, d-pair, using the loop free, unnecessary node and destination rule. From this it can be seen that this set of rules does not perform well. We have $\mu_1 = 1287$, $\sigma_1 = 863$, and $\tau_1 = 5681$. By definition we have $\gamma_1 = \rho_1 = 1$, since this approach is exact.

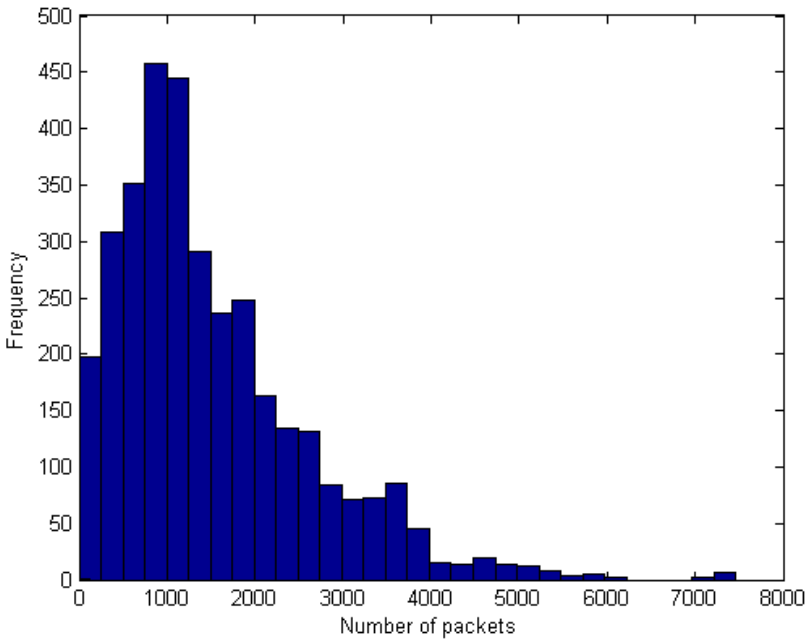

Figure 9.22: Histogram of the number of messages per s, d-pair for the Surfnet topology.

9.3.2 Method 2: Using single path solutions

This method searches for two disjoint paths in the branching tree obtained from the single path algorithm, i.e., we apply the single path algorithm (see Section 9.1.1) and see if two disjoint paths arrive at the destination d. If this is the case, we compare the value of this approximation with that of the optimal solution. We use the destination rule as in the multipath case, i.e., we never apply the destination rule with the edge $\{s, d\}$ (since in that case, the problem of finding two disjoint paths reduces to finding a shortest s, d-path in the graph with $\{s, d\}$ removed). We have $\mu_2 = 191$, $\sigma_2 = 99$, and $\tau_2 = 546$. Furthermore, we find $|F_2| = 2070$, which implies that $\gamma_2 = 2070/2570 = 0.81$. Also, for 1877 of the 2070 pairs, we find an optimal solution. For the other $2070 - 1877 = 193$ pairs, we have given a histogram of the values $\rho_{s,d}$ in Figure 9.23. From this figure it follows that $\rho_2 = 2.5$, although this value is not representative for all pairs.

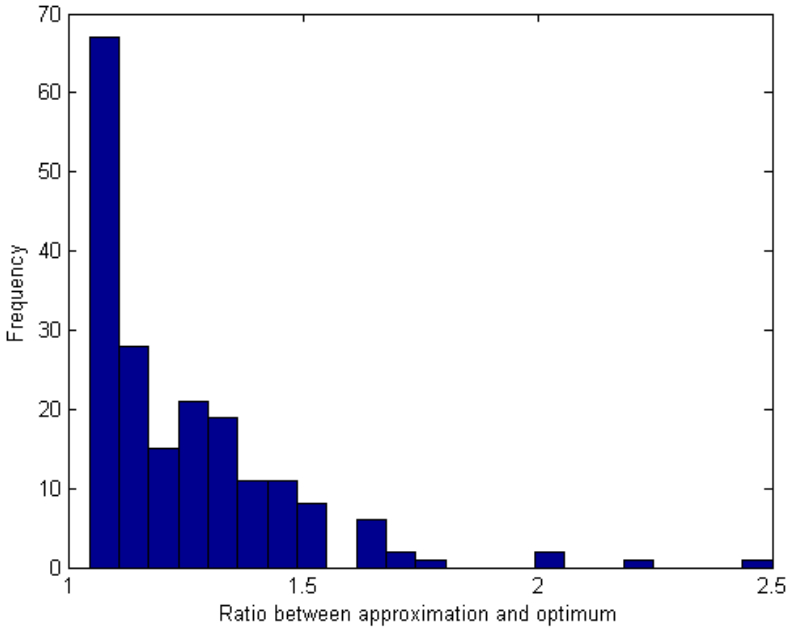

Figure 9.23: Histogram of the values $\rho_{s,d}$ greater than 1.

Just as in the single path case, we can also add a perturbation to the cost function. Although the average number of messages per s, d-pair decreases, the number of pairs for which we find two disjoint paths also decreases. In Figure 9.24 we have given a scatter plot of the average number of messages per s, d-pair, and corresponding values γ_2, for 60 perturbations. It can be seen that there exists a linear relation between these two parameters, as can be expected. In conclusion, although the (average) number of messages per s, d-pair decreases, using a perturbation to break ties, this also reduces the value of γ_2.

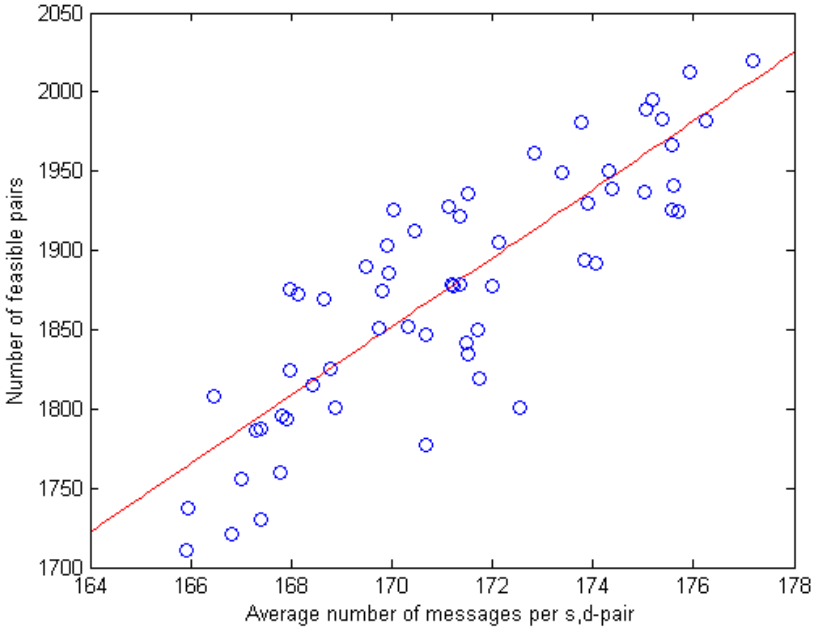

Figure 9.24: Scatter plot (and linear fit) for average number of messages per s, d-pair vs. $\gamma_2 \times |F_{total}|$.

9.3.3 Method 3: Using augmenting paths

In this method, we first determine a shortest s, d-path P_0 for the graph G with cost function c. In the case that multiple shortest paths arrive at d, we select one at random. Then we look for a (shortest) augmenting d, s-path, i.e., from all the augmenting paths that arrive at s, we determine the ones for which the symmetric difference with P_0 yields a node-disjoint pair of paths. From these augmenting paths, we choose the one so that the aggregated cost of the paths in the symmetric difference is minimal. Note that this means we consider more paths than just the shortest augmenting paths, when searching for a feasible solution (although a shortest augmenting path is needed for an optimal solution).

Instead of determining an augmenting d, s-path, we can also just determine an augmenting s, d-path. However, since the destination receives all the shortest paths (and G is undirected so that these are also shortest d, s-paths), it is more natural to look for an augmenting d, s-path (otherwise we would first have to notify the source about the shortest path). Also note that we choose a shortest s, d-path at random. In order to do a complete worst-case analysis, we have to check if we can find a good augmenting path for all shortest paths that arrive at d. If so, we have to choose the least optimal solution as input for ρ_3.

From the simulations, it follows that $\mu_3 = 448$, $\sigma_3 = 218$, and $\tau_i = 1315$ (for every s, d-pair, we have aggregated the number of messages for finding the shortest path and the augmenting path). We find two node-disjoint s, d-paths for 2264 of the 2570 feasible pairs, which implies that $\gamma_2 = 2264/2570 = 0.88$. Furthermore, we have $\rho_3 = 1.69$.

We have also performed a simulation with perturbation, where we found $\mu_3 = 390$, $\sigma_3 = 175$ and $\tau_3 = 1379$. Although this is just one perturbation, it shows that it is useful to break ties.

Compared with Method 2, this method has the advantage that the destination node only has to store one path at all times. As soon as a shorter path arrives, it can delete the previous path. In Method 2, we need to store all paths at the destination in order to determine if there exist two disjoint paths.

9.3.4 Method 4: Removing shortest path

In order to see whether or not we actually need an augmenting path to find an optimal solution, we will also analyse the following method. We first determine a shortest s, d-path P_0 and then look for a shortest d, s-path in the graph $G \setminus P_0$, i.e., the graph with the nodes $V(P_0) \setminus \{s, d\}$, and all edges adjacent to one of these nodes, removed. In Method 3, the average numbers of messages for resp. the shortest and augmenting path are 181 and 267. Clearly, when removing the path P_0 and applying the shortest path algorithm, the resulting number of messages will be lower.

From the simulations it follows that $\mu_4 = 305$, $\sigma_4 = 130$ and $\tau_4 = 933$. Again, we have aggregated the number of messages for both steps of the procedure. We find two node-disjoint paths for 2298 of the 2570 pairs, which implies that $\gamma_4 = 2298/2570 = 0.89$, and we have $\rho_4 = 2.56$.

9.3.5 Overview

In this section we will give an overview of the results for the Methods 1-4 and discuss them. The overview is given in Table 9.2.

Method	μ_i	σ_i	τ_i	γ_i	ρ_i
Exact approach	1287	863	5681	1	1
Using single path solutions	191	99	546	0.81	2.5
Using augmenting paths	448	218	1315	0.88	1.69
Removing shortest path	305	130	933	0.89	2.56

Table 9.2: Overview of performance indicators for Methods 1-4.

It can bee seen that the exact approach works best with respect to feasibility and optimality guarantees (γ_i and ρ_i), however, this method uses the highest (average) number of messages μ_i. Method 2 is the best method with respect to the number of messages μ_i, although the feasibility and optimality guarantees are bad. The third method is the best heuristic with respect to the cost of the approximation ρ_i. The fourth method gives the highest feasibility rate γ_i (slightly better than the third method), but the worst-case approximation of the optimal value is better for Method 3 (the s, d-pair for which the value ρ_4 is attained, is solved to optimality in Method 3).

Note that this also shows the importance of the shortcut rule for the single path algorithm, which can be seen from the difference between in μ_1 and μ_2 (the essential difference between Method 1 and 2 is the shortcut rule).

The performance indicators will be better if we would also allow edge-disjoint solutions. The third method will then be optimal. For the fourth method, the relaxation to edge-disjoint paths does not yield different values (since by construction we can never find edge-disjoint paths that are not node-disjoint).

An essential difference between the first/second, and third/fourth method, is that for the first two methods, the destination must be able to store multiple paths in his memory (in order to compare multiple incoming paths). For the third and fourth method, this is not needed. Whenever a better path arrives, the destination can just delete the old, and store the better path. The same reasoning applies for the augmenting case, although, in the third method, the source node must be able to calculate the symmetric difference between a shortest path and shortest augmenting path.

Part III

Discussion and conclusion

Chapter 10

Applicability of the patent

In this chapter, we will comment on the applicability of the concepts introduced in the previous part. We will give a high-level discussion with respect to DSR-based protocols and comment on the structural differences.

One of the main differences with traditional routing methods is that we do not store any information regarding the route discovery process at intermediate nodes. This means that, at this point, we are not interested in route maintenance. To the best of our knowledge, this has not been studied before. This also limits the number of ways in which we can compare this patent to other source routing methods, since route maintenance is an important part of most performance studies.

It should be noted that, from a mathematical point of view, the problem of finding disjoint paths is still solvable using a polynomial number of messages, when we are not allowed to store information at intermediate nodes. This can, for example, be done using a message that collects information about all edges in the network. That is, the source s sends a message to one of its neighbours v containing the edge $\{s, v\}$. Then v sends the message to some neighbour w and adds the edge $\{v, w\}$. We can continue in this fashion in order to traverse all edges in the network. Whenever the message arrives at a node of which all the edges are already in the message, we mark that node as 'searched'. From the information already gathered in the message, we can find a path to an unmarked node (that has an edge which has not yet been traversed). Once all the nodes are marked, the message is sent to the destination (over a path that can be found in the gathered information), if we are not already in the destination. Since the destination then has full topology information, it can use a central algorithm to solve the problem of finding disjoint s, d-paths. The number of messages involved is polynomial in the number of nodes n of the network (and we have not stored any information from the message at intermediate nodes).

One of the nice properties of this algorithm is that it is independent of time, in the sense that we can explicitly determine the messages that have to be sent. Furthermore, it is a rather straightforward approach, that has some similarities with DSR-based protocols, which means that it might be easy to implement the algorithm using the existing DSR standards.

The method used in the patent is probably more relevant in the context where source and destination fulfil another function than the intermediate nodes, for example, when soldiers in the field are used to establish a connection between two military vehicles. Otherwise, if we would also want to discover routes between intermediate nodes (as source and destination), we will need memory to store paths at intermediate nodes, but then we might as well use this memory to improve the route discovery process.

An advantage of not storing any information at intermediate nodes, is that the discovery process becomes more robust. Since every message is processed independent of any other messages arriving at a node, transmission failures might not have much influence on the complete discovery process, since a node will process other messages as well. We should be careful with this statement, since the transmission failures might occur due to the large amount of overhead, created by our own route discovery messages (which means we are trying to solve a problem that we created ourselves). In DSR, if the retransmission of the first RREQ fails, then any subsequent messages

are not processed (since the sequence number is already stored). However, it should be mentioned that there exist mechanisms to guarantee the correct transmission of a RREQ, so this will not be an immediate problem. The proposed method might take away the need for these fail-safe mechanisms.

10.1 Optimal vs. feasible paths

In dynamically changing network topologies, the problem of finding feasible paths is sometimes more important than finding optimal paths. This is also one of the major differences between routing in static and dynamic networks. In static networks, routes will have a longer lifespan, implying that it is useful to search for shortest paths, whereas in dynamically changing topologies this might take too long (relative to the lifespan of the route). Also, finding optimal paths will lead to more overhead in the network which might not outweigh the benefit of having a shorter path. That is, the additional overhead due to not using an optimal path does not have to be worse than the additional overhead created when searching for an optimal path. In some cases, looking for optimal paths is done implicitly, depending on the objective function. For example, in DSR, the first RREQ arriving at the destination node probably has a relatively small end-to-end delay (time to get from source to destination), although there are no explicit techniques involved in minimizing this end-to-end delay.

In order to use DSR for finding a single shortest s, d-path, there is an easy extension of the original DSR protocol, by having a node v store the routing metric of the first incoming RREQ. When a subsequent RREQ arrives, v only rebroadcasts it if the routing metric of the corresponding path is better than that of the first path. We then update the stored routing metric and repeat this process whenever a new RREQ arrives. From a mathematical point of view, this is precisely a distributed version of the Bellman-Ford algorithm.

Note that this method can be applied when the nodes are stateless, i.e., when they do not know who their neighbours are. If the nodes are stateful, we can use neighbour information, in the form of exclusion rules, to reduce the number of broadcasts. Note that this leads to more overhead in the network, since nodes have to periodically send *hello*-messages to determine who their neighbours are (but these are very small). We will illustrate the concept of using neighbour information in Example 10.1.1.

Example 10.1.1. Consider the following (partial) graph $G = (V, E)$, and let c be non-negative cost function satisfying $c(sv_1v_3) < c(sv_2v_3)$.

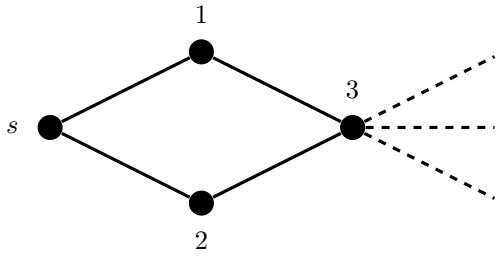

Figure 10.1: Example where we assume that $c(sv_1v_3) < c(sv_2v_3)$.

First assume that we do not use any neighbour information, and that the path $P = sv_2v_3$ arrives before $P' = sv_1v_3$. Then node v_3 stores the routing metric $c(P)$ and forwards the path to all its neighbours (in particular to neighbour v_1, but this is not relevant if we assume that $c(sv_1) < c(P'v_1)$). When path P' arrives, node v_3 will update its routing metric to that of $c(P')$ and send the path to all its neighbours.

If we would have used neighbour information, then the message corresponding to P could already have been dropped, by using the shortcut rule via v_1. Note that if the path P' would have

arrived before P, then we would have rejected P based on the fact that we already had stored the (better) routing metric of the path P'. In that case, the neighbour information is redundant. □

10.2 Increased message size and processing time

A question that arises, is whether or not it is useful to include neighbour information, as in Example 10.1.1. Being able to exclude neighbours, based on a previously processed path with lower routing metric, is a very strong tool, that can make neighbour information redundant. Although the number of neighbours to which the updated message is sent, is reduced, the messages themselves become larger. This means that the total overhead increases, i.e., the total amount of data sent through the network for the route discovery process increases. If this increase is not compensated by the messages that we not send, by using neighbour information, this method might not be beneficial.
The increase in message size might also lead to problems in the transmission of data. For example, an issue with MANETs is the limited bandwidth that can be used to communicate with neighbours, so increasing the size of the messages might therefore not be wise for those networks, since this can increase the transmission failure rate.

In order to give a comparison with the adjusted DSR-method, as explained in the previous section (see also Example 10.1.1), it might therefore be useful to analyse the total amount of data that has to be sent with, or without, neighbour information. We can express the size of a message by $|G_P|$, depending on the implementation of the distributed algorithm (see for example Section 9.1.1).

Note that the branching trees are stochastic if we allow intermediate nodes to store the best routing metric. This holds for the simple reason that we do not know in which order paths arrive at a given node. Therefore, we will speak of the expected size of the branching tree. We define

$$\sum_{P \in W(T)} |G_P| \tag{10.1}$$

and take the expectation, over all (theoretically) possible branching trees, as the performance indicator of the algorithm. This performance indicator then describes the expected total amount of overhead created by the route discovery process. In the next section we explain a possible choice for the framework in which the stochasticity can be captured.

The algorithm can be analysed using different information mappings ϕ. For example, the trivial mapping corresponds to the situation of including no neighbour information, whereas the mapping $\phi_{\mathcal{N}^*}$ (equivalent mapping for the neighbour mapping $\phi_{\mathcal{N}}$) might be used to include neighbour information. We might also choose mappings in between these two extreme situations, for example by only including neighbour information of the last r nodes on the path P. It might be interesting to see what mapping yields a minimization of the expected value in (10.1), for given types of graphs.

Not including all neighbour information also addresses the issue that information about neighbours, added at the beginning of the path, might be useless once the path becomes longer. In Example 10.1.1, the graph $G_P = (V_P, E_P)$ for $P = sv_1v_3$, contains the edge $\{s, 2\}$, but once this path passes node v_3, the edge will never be used to exclude v_2.

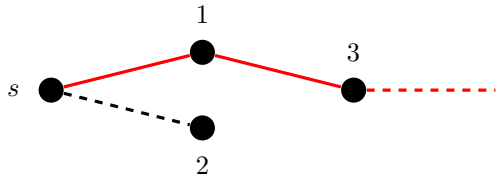

Figure 10.2: Example where the edge $\{s, 2\}$ becomes useless.

This problem can also occur in the multipath case. For example, in order to apply the double shortcut rule (see Lemma 6.1.1), we have to store at least two edges to every neighbour.

However, the number of times that we can actually apply this exclusion rule might not outweigh the storage of all the edges, i.e., the measure in (10.1) might be less when only storing one edge to every neighbour (and not using the double shortcut rule).

Another note we have to make is about the processing time of a message. Although storing additional information might increase the number of neighbours that can be excluded, it will also take more time to process a message. For example, in the DSR-based approach, a node v only has to append itself to the route record and see whether or not the resulting path has a lower routing metric than the best path seen so far at node v. This can be done in constant time.

In order to apply exclusion rules, we will need a non-constant amount of time, depending on the number of neighbours of node v, and the amount of neighbour information already included. However, if the main goal is to reduce the overhead, it might not matter that the processing of a message takes more time.

A drawback for MANETs is that mobile nodes often have a limited power supply. Therefore, a longer processing time might not be preferable.

10.3 Time stamps and priority rules

In the second part of this thesis, we have not considered the element of time. One might, for example, ask the question how long the destination has to wait before all messages have arrived (which makes it possible to know what the optimal solution is). Furthermore, from the simulations, it followed that some sets of exclusion rules do not perform very well (especially in the multipath case), in terms of total number of messages that have to be sent.

In order to introduce some methods that might solve this problem, we make the following assumptions.

i) Every node in the network has a clock, that is synchronized with the clock of every other node in the network.

ii) A node is able to make a *time stamp*, that can be included in a message to a neighbour.

The idea is to have the source node make a time stamp at time t, when it creates the initial message, and add a number of time units t_0 to it. The source includes the time stamp $t+t_0$ to the initial message, and intermediate nodes only process messages as long as they receive them before $t + t_0$. This also resolves the problem of the destination node not knowing how long to wait for incoming messages. However, we might lose the optimality guarantee, depending on the choice of t_0. In order to analyse the concept of a time stamp, we introduce the following discrete simulation model.

At every $t \in \{0, 1, 2, 3, \ldots, t_0\}$, a node can process at most one message, and send it to a subset of its neighbours (that can then process the updated message at time $t + 1$ or later). The initial message is created at time $t = 0$. Furthermore, since nodes might receive multiple messages at a given time t (all from different neighbours), we assume that every node has a buffer in which it can store incoming messages that cannot directly be processed. The messages are processed using the 'first-in-first-out' principle, i.e., they are processed in order of arrival. When multiple messages arrive at the same time t, we assume they arrive in some random order (this is the stochastic element of the model). This assumption might not be justified if, for example, the cost of an edge is related to the time it takes to get from a node to its neighbour.

Note that we assume the processing of a message can be done in constant time, which is not in line with what was explained in the previous section. However, in order to keep the model simple, we make this assumption.

A second concept we introduce, is that of priority rules for the order in which buffered messages are being processed, i.e., if there are multiple messages in the buffer waiting to be processed, we can select one based on some priority rule rather than using the 'first-in-first-out' principle.

For example, for a node v, we can select a message for which the cost of the path is minimal among the messages in the buffer. Note that the path of a message in the buffer corresponds to a path up till the neighbour w that has sent the message to v (since we extend the path to v once

the message is being processed, but this done after it is selected from the buffer). For a constant cost function, the shortest path in the buffer is also the shortest path up till v. In case of a tie, we select a shortest path with the earliest arrival time.

The most important parameter in this model is t_0. If we choose $t_0 = n$, and use the priority rule that selects a shortest path from the buffer, we can show that a shortest s, d-path will always arrive at d, within n time units (for c a constant non-negative cost function). However, a priori, the source will not know the size of the network n, which means that this choice of t_0 is not likely.

One way of resolving this problem is as follows. If the destination has received the desired number of disjoint paths after t_0 time units, it can send back acknowledgements of these paths, using them in their reversed direction. This means that after at most $2t_0$ time units, the source will have received all the acknowledgements of the disjoint paths (assuming that the processing of an acknowledgement will take one time unit at an intermediate node). If the source does not have the desired number of responses after $2t_0$ time units, it can choose a higher value of t_0 (for example, by doubling the initial value). Note that all the messages corresponding to the first value of t_0 then already have vanished from the network.

It is important to note that we assume here that there is no additional network traffic. If this is the case, the results in this section only serve as heuristic approaches. Furthermore, this simulation model is actually independent of the exclusion rules and information mappings. It only serves as a framework for improvements for graphs where the algorithm performs bad, so that the number of messages will not explode.

Example 10.3.1. We have applied the proposed simulation model to graphs of the type given in Figure 10.3: a grid of size $n = k^2$, where we have chosen $t_0 = n$. This implies that there will be at most n^2 messages involved in the route discovery process. Furthermore, we have used the loop free, unnecessary node and destination rule (this set can theoretically solve both the single and multipath problem).

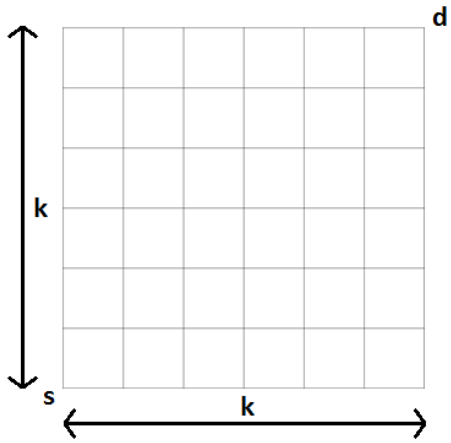

Figure 10.3: Grid of size $N = k^2$, with c a constant non-negative cost function.

In Table 10.1, we have given an overview of some of the results, based on 600 iterations per k. We have determined the percentage of simulations where the destination received two node-disjoint paths within $t_0 = n$ time units (success ratio). Furthermore, we have also given the average time $t^* \in \{0, 1, 2, \ldots, t_0\}$ after which an optimal solution could be formed from the paths that had arrived at d, in the interval $[0, t^*]$. The average is based on the simulations for which an optimal solution was found.

k	6	7	8	9	10	11
Success ratio	0.987	0.97	0.965	0.947	0.943	0.942
t^*	15	18	22	27	29	33

Table 10.1: Overview results for the simulation model on a grid.

It can be seen that the method performs worse when the size of the graph increases, which indicates that the proposed model might not scale well (with respect to finding an optimal solution). However, when the optimal solution is found, it is found within $3k$ time units (on average). □

10.4 Concluding remarks

In this chapter we have discussed the applicability of the algorithm proposed in the patent, and discussed some of the differences with the DSR-based approach. Furthermore, we have also introduced time stamps and priority rules in order to reduce the number of messages that is needed to find disjoint s, d-paths.

It is also possible to include certain techniques discussed in Section 2.4.2, in order to determine whether or not to forward a route discovery message. However, one might argue that messages are then no longer processed independent of each other, so this might not satisfy our initial condition on the algorithm. It is also not directly clear whether or not neighbour information can reduce the overall overhead, so this is an interesting topic for further research.

In order to give a complete analysis of the algorithm, it would be better to perform simulations in a technical simulator that actually mimics, for example, a mobile ad-hoc network. These also take into account factors such as transmission failure rate, which will play an important role, since we increase the size of the route discovery messages with respect to traditional DSR-based methods. Furthermore, on average, the processing time of a message will also increase, and does not have to be constant, which is one of the weaknesses of the simulation model introduced.

We should also be careful with the assumptions we make on the nodes. For example, in the case of priority rules, it is not likely that all types of hardware components are able to handle these. The same holds for the assumption of synchronized clocks in the nodes.

Chapter 11

Conclusion and recommendations

In this thesis, we have analysed a distributed algorithm for the discovery of multiple disjoint paths within a communication network, between a given source and destination. These paths can be used for source routing, a routing technique where the path from source to destination is explicitly stored in the data that has to be transferred.

The main difference with existing routing methods is the assumption that intermediate nodes are not allowed to store information regarding the route discovery process. To the best of our knowledge, this has not been studied before.

The idea of the algorithm is to use neighbour information, i.e., local information around a path, to determine whether or not a path can be a subpath of an optimal solution of multiple disjoint paths. We have presented a mathematical framework in order to determine how we can make optimal use this information. Furthermore, we have also provided bounds on the number of messages that will be sent through the network. From the results in Chapters 7 and 9, it follows that the algorithm performs well in dense graphs. This might be a drawback with respect to the scalability of the algorithm, since the degree will not scale with the size of the network, but will more likely be constant. Also, because we want to use neighbour information, nodes have to periodically exchange messages in order to check the reachability of their neighbours, which increases the network overhead.

The single path algorithm performs well in the Surfnet graph (especially if we break ties). The heuristic approaches, presented in Section 9.3, might therefore be very useful in order to establish multiple disjoint paths, although we lose the feasibility and optimality guarantees. In general there is a lot of variation in the size of the branching tree, depending on the choice of source and destination (for a fixed graph). This can, for example, be seen from the results in Section 9.1. It might be interesting to investigate what causes this large variation. However, we can probably not benefit from that analysis, since we cannot always influence the structure of the network in practice, e.g., when nodes are randomly distributed over a geographical area.

Also, the multipath algorithm, consisting of the loop free, unnecessary node and destination rule, can be used to find a maximum number of disjoint paths, which might compensate the fact that it does not perform very well when looking for two node-disjoint paths. Furthermore, it might also be used to solve problems such as Min-Min and Min-Max.

One of the main topics for further research is the comparison with existing methods. It might be possible to include some of the principles, described in the second part of this thesis, in other source routing protocols as well. This can, for example, be done as described in Section 10.1. Furthermore, there are also lots of possibilities for further research in the mathematical framework, for example in Chapter 7, by searching for better bounds for special pairs (G, c).

It is also possible to study more general information mappings, although there might not be a natural interpretation for those. In the multipath case, it might be interesting to investigate equivalent mappings, in particular in relation to the double shortcut rule for arbitrary cost functions c. That is, to see what neighbour information we actually need (from the information mapping $\phi_{\mathcal{N}}$), to apply the double shortcut rule.

It should also be noted that some of the exclusion rules, especially for the multipath problem, will only have a theoretical relevance. In practice it is probably not wise to use these rules, since we need to store a lot of neighbour information in order to apply them. The increased message size will probably not be compensated by the number of times we are able to apply the rules.

In conclusion, the mathematical framework introduced in the second part of this thesis, allows us to show how we can make optimal use of neighbour information. Furthermore, it also serves as a framework in which future research can be performed.

Appendix A - Proofs

Proof of Theorem 5.5.2:

The proof is similar to that of Theorem 5.2.1. Case 1 and 3 can be done in exactly the same way. For Case 2, we want $Pw..d$ to be the shortest path in H (which leads to the infeasibility of the other set $\mathcal{R}'$), but we still have to guarantee that we do not create negative cost cycles when choosing the cost for the edge(s) between w and d, otherwise $(H, c, s, d) \notin \mathcal{G}$.

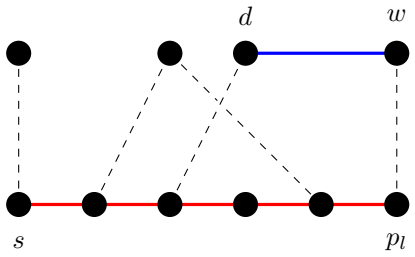

Figure 11.1: The graph H in Case 2 (see proof of Lemma 6.1.1).

The fact that path P arrives at p_l, using $\mathcal{R}$, implies that

$$c_{yd} + c(Q) + c_{xw} > c(Pw) - \min\{c(Pyd) : \{y, d\} \in E\}$$

for all combinations of $\{y, d\}$ and $\{x, w\}$ (using Exclusion Rule 5.5.2, because we do not exclude w using $\mathcal{R}$). Since there are only a finite number of these combinations, we can find some $\varepsilon > 0$ such that

$$c_{yd} + c(Q) + c_{xw} > c(Pw) - \min\{c(Pyd) : \{y, d\} \in E\} + \varepsilon.$$

We then define $c(w..d) = -\left[c(Pw) - \min\{c(Pyd) : \{y, d\} \in E\} + \varepsilon\right]$ (note that we might need more than one node on the blue edge in Figure 11.1 to satisfy the girth condition). This then implies that

$$c(Pw..d) = c(Pw) - \left[c(Pw) - \min\{c(Pyd) : \{y, d\} \in E\} + \varepsilon\right] = \min\{c(Pyd) : \{y, d\} \in E\} - \varepsilon$$

which means that $Pw..d$ is indeed the shortest s, d-path in H. This last claim holds because the shortest path to d in G_P was of the form Pyd for some $y \in V(P)$ because of Exclusion Rule 5.5.1. Furthermore, the shortest path to w is Pw, because of Exclusion Rule 5.1.2. By construction we have not created negative cost cycles, meaning that $(H, c, s, d) \in \mathcal{G}$. □

Proof that destination rule is still applicable (Section 8.3):

Let P_0 be a shortest s, d-path, $P = s \dots p_l$ a simple augmenting path arriving at p_l, $w \in \mathcal{N}(p_l) \setminus V(P)$ and let $x \in V_P$ such that $c(Pxd) \leq c(Pw)$. We claim that Pw can never be a subpath of some optimal augmenting path P' that also uses the least amount of nodes. If P' would be such a path, then the symmetric difference of P_0 and P' forms an optimal pair of edge-disjoint paths (for the cost function c). Since P' clearly uses more nodes that Pxd, we must have

$$c_{xd} > c(xPw) - \alpha + \beta, \tag{11.1}$$

i.e., P' must have cost strictly smaller than Pxd. Here we have

$$\alpha = \sum_{\{u,v\} \in E(wP') \cap E(P_0)} c_{uv} \quad \text{and} \quad \beta = \sum_{\{u,v\} \in E(wP') \setminus E(P_0)} c_{uv}.$$

Note that α is the sum of the solid orange edges and β the sum of the blue edges on wP' (see Figure 11.2). Since $c(Pxd) \leq c(Pw)$ implies $c_{xd} \leq c(xPw)$, we find $\beta < \alpha$, using (11.1).

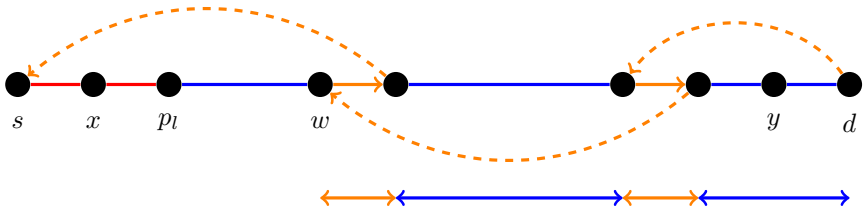

Figure 11.2: Sketch of P' (solid lines) where the red/blue/orange edges might contain more nodes. The orange edges (dashed and solid) form P_0. We denote y as the last node on P' that is connected to an edge of $E(P_0) \cap E(P')$.

The edge $\{p_l, w\}$ might also lie on P_0, but that is not relevant. Moreover, the path P itself (red edges) might use some edges of P_0. Also note that $\{d, y\} \notin E(P_0)$ by definition of an augmenting path. We claim that there exists a strictly shorter path than P_0, under the original cost function c, which gives a contradiction (since P_0 was the shortest path). This path can be found in the symmetric difference of P_0 and wP'. Note that wP' consists, alternately, of edges from $E(P_0) \cap E(wP')$ and $E(wP') \setminus E(P_0)$.

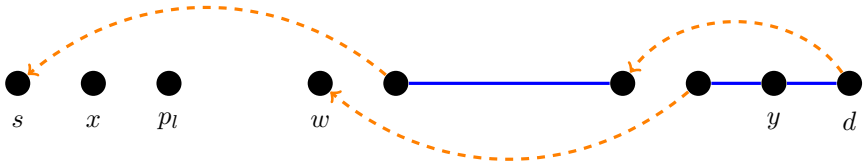

Figure 11.3: Symmetric difference of P_0 and wP', as in Figure 11.2.

Intuitively, we can follow the path P_0 starting from s. Whenever we reach an edge $\{u, v\} \in E(P_0) \cap E(wP')$, we continue the path with the edges from $E(wP') \setminus E(P_0)$, i.e., the (blue) edges between v to v'. At v' we then continue the path with the incoming (dashed) arc.

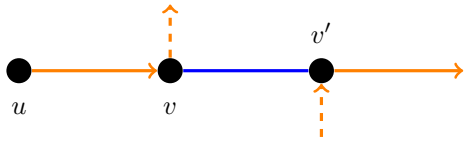

Figure 11.4: Sketch of the above mentioned argument.

All these $\{u, v\} \in E(P_0) \cap E(wP')$ are followed by edges in $E(wP') \setminus E(P_0)$ which makes sure that we can always continue our path in this way. Note that the path Q we find in this way might contain cycles, but these all have non-negative cost (since the path P_0 might use nodes lying on the blue part).

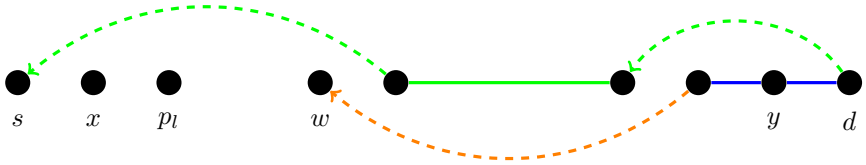

Figure 11.5: The path Q in the symmetric difference of P_0 and wP', as in Figure 11.2. Note that Q might contain non-negative cost cycles, which can be removed.

For this path Q, under the original cost function c, we have (note that the edges we sum over in the definition of α are not contained in $E(P_0 \cap Q)$)

$$c(Q) = c(P_0 \cap Q) + c(E(wP' \setminus P_0) \cap E(Q)) \leq c(P_0 \cap Q) + \beta < c(P_0 \cap Q) + \alpha \leq c(P_0)$$

□

Bibliography

[1] Wikipedia, "Osi model — Wikipedia, the free encyclopedia," 2015. [Online]. Available: http://en.wikipedia.org/wiki/OSI_model

[2] C. Kozierok, *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference.* San Francisco, CA, USA: No Starch Press, 2005.

[3] A. V. Aho and D. Lee, "Hierarchical networks and the lsa n-squared problem in ospf routing." in *GLOBECOM.* IEEE, 2000, pp. 397–404. [Online]. Available: http://dblp.uni-trier.de/db/conf/globecom/globecom2000.html#AhoL00

[4] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking." *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/cm/cm51.html#YeganehTG13

[5] J. Tsai and T. Moors, "A review of multipath routing protocols: From wireless ad hoc to mesh networks."

[6] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, pp. 87–90, 1958.

[7] L. R. Ford Jr, "Network flow theory," *Paper P-923.*, 1956.

[8] R. Bhandari, *Survivable networks: algorithms for diverse routing.* Springer Science & Business Media, 1999.

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

[10] J. Suurballe, "Disjoint paths in a network," *Networks*, vol. 4, no. 2, pp. 125–145, 1974.

[11] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[12] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[13] C.-L. Li, S. T. McCormick, and D. Simchi-Levi, "The complexity of finding two disjoint paths with min-max objective function," *Discrete Applied Mathematics*, vol. 26, no. 1, pp. 105–115, 1990.

[14] D. Xu, Y. Chen, Y. Xiong, C. Qiao, and X. He, "On the complexity of and algorithms for finding the shortest path with a disjoint counterpart," *IEEE/ACM Trans. Netw.*, vol. 14, no. 1, pp. 147–158, 2006. [Online]. Available: http://doi.acm.org/10.1145/1133553.1133565

[15] J. J. Garcia-Luna-Aceves, "A unified approach to loop-free routing using distance vectors or link states," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 4, pp. 212–223, Aug. 1989. [Online]. Available: http://doi.acm.org/10.1145/75247.75268

[16] R. G. Ogier, V. Rutenburg, and N. Shacham, "Distributed algorithms for computing shortest pairs of disjoint paths." *IEEE Transactions on Information Theory*, vol. 39, no. 2, pp. 443–455, 1993. [Online]. Available: http://dblp.uni-trier.de/db/journals/tit/tit39.html#OgierRS93

[17] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," United States, 2003.

[18] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*. Kluwer Academic Publishers, 1996, pp. 153–181.

[19] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," in *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, ser. MobiCom '99. New York, NY, USA: ACM, 1999, pp. 151–162. [Online]. Available: http://doi.acm.org/10.1145/313451.313525

[20] J. Li and P. Mohapatra, "Panda: An approach to improve flooding based route discovery in mobile ad hoc networks," 2002.

[21] A. Qayyum, L. Viennot, and A. Laouiti, "Multipoint relaying for flooding broadcast messages in mobile wireless networks," 2002, p. 298.

[22] P. Jacquet, P. Mhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," 2001, pp. 62–68.

[23] Y.-B. Ko and N. H. Vaidya, "Location-aided routing (lar) in mobile ad hoc networks," *Wirel. Netw.*, vol. 6, no. 4, pp. 307–321, Jul. 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1019106118419

[24] K. Wu and J. Harms, "On-demand multipath routing for mobile ad hoc networks," in *Networks EPMCC 2001, Vienna, 20th 22nd February 2001*, 2001, pp. 1–7.

[25] S. ju Lee and M. Gerla, "Split multipath routing with maximally disjoint paths in ad hoc networks," 2001.

[26] P. Thubert, J. Vasseur, and V. Ribiere, "Computing disjoint paths for reactive routing mesh networks," Nov. 5 2013, uS Patent 8,578,054. [Online]. Available: http://www.google.tl/patents/US8578054

[27] C. Liu, "Determining two node-disjoint paths using on-demand flooding," Mar. 25 2008, uS Patent 7,349,350. [Online]. Available: http://www.google.tl/patents/US7349350

Index