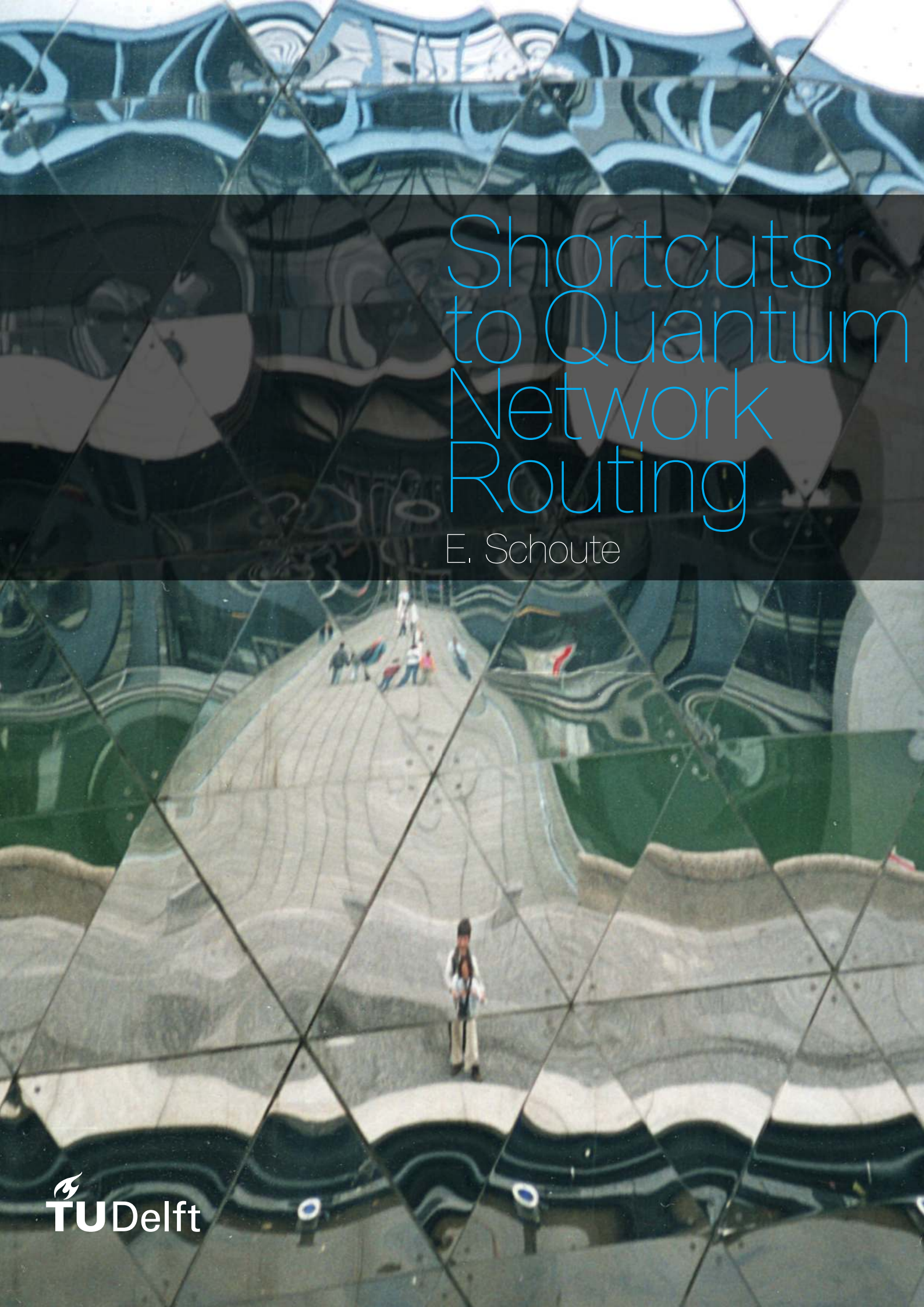


Shortcuts to Quantum Network Routing

E. Schoute

TU Delft

Shortcuts to Quantum Network Routing

by

E. Schoute

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday August 26, 2015 at 13:00.

Student number:	4101790	
Project duration:	December 16, 2014 – August 26, 2015	
Thesis committee:	Dr. S. Wehner,	TU Delft, supervisor
	Prof. dr. ir. L. M. K. Vandersypen,	TU Delft
	Dr. ir. M. M. de Weerdt,	TU Delft

TU Delft Delft University of Technology

QUTech

This thesis is based on the following paper:

- **Shortcuts to Quantum Network Routing**
  Eddie Schoute, Laura Mančinska, Tanvirul Islam, Iordanis Kerenidis, Stephanie Wehner.
  In preparation.

Abstract

Communication is a key part of society that we make huge investments in. Each message has to both arrive quickly and use minimal network resources. Moreover, communications must sometimes also remain private, so encryption is used to prevent third parties from listening in. However, standard classical encryption protocols are proven not to be secure against quantum adversaries. With quantum communication we have a method that does provably attains perfect secrecy, even against adversaries with unlimited (quantum) computational power. It is, however, infeasible to have a direct quantum connection between all users of quantum communications. We thus look into the possibilities of a *quantum network*, which is able to connect arbitrary users of the network as if they had direct quantum channel available.

Once such a quantum network is built, we will need to route quantum messages effectively to their destination. In this work, we take the first step in studying network structures for quantum networks on a high level, and their accompanying routing algorithms. Current ongoing research into quantum communication with satellites and the possibility of a global network made the case of a quantum network of satellites relevant. We will consider a simplified model where each satellite can perform quantum communication with its immediate neighbours, and can create quantum virtual links that form shortcuts through the network.

The first step towards a quantum network that we take is the structure of the network. Using virtual links as shortcuts we can reduce the maximum distance between nodes (the diameter) by an exponential factor. Let $|V|$ be the size of the set of all vertices in a graph, equal to the number of satellites. Then for the case of satellites we give a network structure with the following properties:

- The diameter of the graph is $4\log_2\left(\frac{|V|-2}{10}\right)+3 = O(\log|V|)$. This means that the distance between any two nodes in the graph will scale logarithmically in the number of nodes.
- The maximum degree of a vertex is $12\log_2\left(\frac{|V|-2}{10}\right)+6 = O(\log|V|)$. The degree of a vertex is directly related to the size of its quantum memory. Since quantum memories are currently limited in size, it is important for feasibility reasons that the degree stays low.

With a logarithmic scaling, the diameter and degree scale well for large graphs. This is especially relevant for networks, which can become large. A lower distance between vertices also allows quantum communications to fail less often, in effect decreasing the communication time. We can thus achieve a much smaller diameter by using virtual links, while the degree of each vertex remains low.

We furthermore give a routing algorithm for finding the shortest path on this network structure that only uses local information at each vertex to route, where local information is the nearby surroundings of a vertex. The routing algorithm has the following properties:

- The space complexity of the algorithm is $O(\log^5|V|)$ per vertex. A small space complexity implies little memory usage of the algorithm. Since network routers are typically small computers, the small memory requirement allows them to run this algorithm.
- The time complexity of the algorithm is $O(\log^2|V|)$ per vertex. A small time complexity allows network nodes to compute the next step in the path in minimal time. The time necessary to communicate decreases with decreasing time complexity.

For large enough $|V|$, our algorithm outperforms standard routing algorithms such as Dijkstra's algorithm by an exponential factor. Furthermore, we have proved that the local routing algorithm always gives a shortest path. Local routing transitions well into a real world implementation, so that every network router can make simple decisions for routing.

E. Schoute
Delft, August 2015

Acknowledgements

This thesis was written under supervision of Stephanie Wehner of the QuTech Research Centre and the Intelligent Systems Department, Cyber Security group, at the Delft University of Technology.

I want to thank Stephanie Wehner for her guidance and contributions, as well as Laura Mančinska for her contributions to this thesis. Thanks to Tanvirul Islam and Iordanis Kerenidis, who are co-authors on the paper for quantum networks routing. Additionally, I thank David Elkouss and Jędrzej Kaniewski for providing valuable feedback that helped improve this work. I would also like to thank the members of the examination committee, Stephanie Wehner, Lieven Vandersypen, Mathijs de Weerdt, David Elkouss and Jędzrej Kaniewski for making time available to provide comments and supervise the graduation. Finally, thanks to my friends who I have learned great deal from during our many great projects, and my parents for supporting me through all these years.

Contents

1 Introduction

Communication is a key part of society that we have built huge intercontinental computer networks for. The global network uses large undersea network cables, satellite communications and local fine grained connections to connect every business and home. Each message has to both arrive quickly and use minimal network resources. Moreover, communications must sometimes also be kept private, so we use authentication and encryption to prevent third parties from listening in. Quantum mechanics has showed us that perfect privacy is attainable, even when faced with adversaries of unlimited computation power. So far, quantum communication mostly uses only a direct connection from one point to another. Ideally, though, we would like to have a *quantum network* to connect any user to any other user on that network, while still being fast, efficient and also perfectly secure.

1.1. Why Quantum Networks

A driving force for quantum networks is the security of modern-day encryption protocols such as RSA and the Advanced Encryption Standard (AES) [DR99]. These protocols are not proven to be secure against adversaries with classical computers. It has not been shown to be impossible to find a solution in polynomial time to the factoring problem that these encryption standards rely on. Moreover, they are proven to be insecure against adversaries with a quantum computer, which can solve factoring in a polynomial number of steps. For some communications it is not only important they remain secret on the short term, but they should also remain secret on the long term. And since a large enough quantum computer can break the encryption, someone could store communications being sent today and decrypt them once such a quantum computer becomes available. The Dutch national intelligence agency has also published a white paper warning about this issue [Alg15], which estimates the arrival of such a quantum computer in 15 to 20 years. Even though other encryption techniques have been developed for post-quantum security, it is still unproved that these are secure against a quantum attacks. However, it is proved that quantum communication would be resistant even against quantum attacks [LC99; SP00]. A protocol named Quantum Key Distribution can distribute perfectly secure keys to the communicating parties. These keys may then be used to classically send perfectly secure communications.

A unique feature of quantum networks is the possibility to create virtual links, in addition to the physical connections the network is bound to. Quantum network nodes have some fixed physical connections they may use to establish an *entangled pair* with each other, which requires a little time [Ber+13]. An analogue for such an entangled pair are two mobile phones that can only communicate among themselves using what is called *teleportation*. Two users of a quantum network that each have one phone can then perform quantum communication. Furthermore, a user may send their quantum phone to someone else in what is called an *entanglement swap*, Entanglement swapping is illustrated in Fig. 1.1, where Bob swaps entanglement so that Alice and Charlie share an entangled pair. The new holders of the entangled pair may then communicate as if they were connected by direct connection. The communication does require some classical information to be sent over the network as well, thus quantum communication cannot be faster than classical communication. The connection between the entangled pairs does not have to be along one of the physical quantum connections, but instead can be seen as a virtual connection between network nodes. A drawback, though, is that the virtual links are used up on every communication. Once a virtual link has been established it will also only be available for use until it expires, which is related to the lifetime of entanglement (currently at most on the 10 millisecond scale [Ber+13]). The lifetime of entanglement is still significantly being improved on, which allows for better preparation of the network, as virtual links will not have to be refreshed so often.

To perform quantum communication the sender and receiver must share an entangled pair. If the sender and receiver are not directly connected with a physical connection, then intermediate nodes may perform

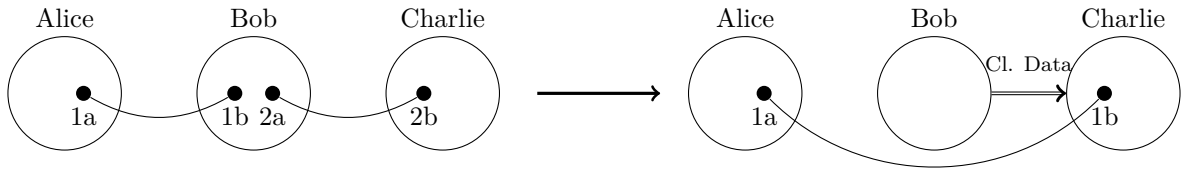


Figure 1.1: At first, Alice and Bob share entanglement, and Bob and Charlie share entanglement (connected dots). Bob performs an entanglement swap that requires the entanglement with both Alice and Charlie, which entangles Alice with Charlie [NC10]. This procedure moves the entanglement of Bob with Alice (1b), to Charlie, using the entanglement between Bob and Charlie (2a,2b). This also requires some classical data to be sent to Charlie, and the entanglement between Bob and Charlie is consumed by the teleportation. Bob may re-establish entanglement with a physical connection.

entanglement swaps to create a pair between sender and receiver. A virtual link is established between any sender and receiver in this way. However, each swap also has a certain failure probability which may force retries to establish a virtual link. By continuously creating virtual links, even when there is no communication going on, it is possible to have some set of virtual links already available once communication is requested. If these links are properly positioned in the network, then they reduce the number of entanglement swaps necessary to perform quantum communication between arbitrary senders and receivers. Note that the total number of entanglement swaps is not reduced by the use of virtual links, as the number of swaps required to establish a virtual link is the same as trying to perform quantum communication directly. This also implies that it is harder to establish a long virtual links than a short virtual link, since it requires more swaps to be established. Nonetheless there are less failures than when a virtual link has to be established from scratch, which in turn reduces the time necessary for a user of the network to communicate. All in all, with quantum networks it is possible to establish virtual connections in advance even when there is no communication going on, reducing the time a future user of the network needs to communicate. Once a user shares a virtual link with the intended receiver they may then perform perfectly secure quantum communications.

Some basic structures of quantum networks have advantages and disadvantages. If we take the network that has a virtual link from every node to every other node then communication is trivial. However, it requires every node to maintain as many connections as there are nodes in the network (except itself), where each connection requires one quantum phone to be stored. Each phone is one out of a pair of *quantum bits* (qubits). Currently, the largest quantum memories are of the size of ten qubits, and scaling these memories is not practically feasible because of the costs involved. Thus it is not feasible to build completely connected networks using virtual links.

Alternatively, there is the network that uses only physical connections and no virtual links. This network does not require storing more than two qubits, since each vertex may set up an entangled pair and perform entanglement swaps with them immediately. The disadvantage is that network has a higher risk of failure, since it requires more entanglement swaps to establish a connection.

Furthermore, it needs to be possible to route on the network, which requires a routing algorithm. The algorithm should run quickly, so that any communication is not delayed by the routing. Since entangled pairs have such a short lifetime, the algorithm must be quick enough to use the available entanglement. We also want the memory usage of the algorithm to be low, so that network nodes (which are typically small computers) are able to run the algorithm. A standard routing algorithm called Dijkstra's algorithm [KT06] requires every node to know the entire network structure, which is why it is infeasible for large scale networks as the size of memory in each device is limited to reduce costs. By limiting the available information of a node to only its nearby vertices, which is *local information* for that node, we can reduce the required memory significantly.

1.2. Contribution

In this thesis we propose a feasible and fast network structure for satellites together with a routing algorithm. We look at satellites because of current research into quantum communications with satellites, and the possibility of a global quantum network. There are only few satellites needed to enable communications from one side of the Earth to the other. We assume that these satellites can only perform direct quantum communication with other nearby satellites. The satellites are modelled as nodes in a network that are spread out over a sphere, where each node has a limited quantum memory size. Because of the limitations on the quantum memory, this problem may also be viewed as a resource distribution problem, where the resources to be distributed are the virtual links. When designing the quantum network we ask the following research questions:

1. What network structure using virtual links can we prepare ahead of time such that the number of swapping operations to establish a long distance link is minimized for every pair of nodes that may wish

to communicate?

2. Is it feasible to implement this network considering the current technological limitations, such as the size of quantum memory?

3. How can we efficiently route on the network? Ideally, we would like a routing algorithm which requires only local information at each node, and is computationally efficient.

We will now go into further detail for each of these questions.

1.2.1. Distributing Entanglement and Feasibility

The first step is designing a network structure that can be added on top of the existing physical network of satellites. We have seen the pros and cons of a fully connected network and a network with only physical links. Instead, we give a network that is somewhere between the two extremes. This network has a distance between any two nodes that is logarithmic in the number of nodes, but still requires only a logarithmic size memory. The limited memory size makes implementing such a network feasible, since the high cost prevents us from having large quantum memories.

Prior work on quantum networks only looks into point-to-point communication, or networks that lie on a line. As such we take a high-level approach to this novel problem, and not concern ourselves with noise, lifetime of qubits, or the procedure for establishing such long-distance links. However, we have kept the number of long distance virtual links to a minimum, since they require more entanglement swaps to create. We assume that there is some process which continuously establishes entangled pairs, and refreshes the virtual links. Instead, we look at the distribution of resources and the possibilities of quantum networks, keeping in mind the limited size of quantum storage in a quantum computer. The scientific contribution of our network model is thus a first step towards of the novel properties and possibilities of quantum networks, and an insight into future research of quantum networks.

1.2.2. Routing Algorithm

A routing algorithm on the network structure that can be performed by the network nodes is also desired. We have seen that Dijkstra's algorithm, and other algorithms which require global information, do not suffice for this purpose. We use the structure of the network to prove properties about shortest paths in that network. One of these properties shows that it is possible to route in a towards certain nodes if the destination is not nearby. This allows us to formulate an algorithm that calculates the shortest path, but that does not require the network nodes to know the global structure, since it only has to know whether the destination is nearby.

Another scientific contribution is thus a routing algorithm on the network structure. This algorithm requires only a logarithmic computation time and classical memory in the number of nodes. This implies the algorithm scales well with the size of the network and is suitable to run on network nodes, which have only limited computation power and memory.

In the upcoming Chapter 2 we will give a short introduction of the required knowledge for this thesis. For the following results, we first giving a high-level overview and explain our reasoning, which is later followed by technical details proving the results. We give a high-level over of our network model in Chapter 3, and later prove the correctness of our statements in Section 3.2. Then we proceed with a high level overview of the network structure analysis and the routing algorithm in Chapter 4, followed by the proofs of the results in Chapter 5. At the end we will conclude and discuss our results and possible future research in Chapter 6.

2

Background

In this chapter we give a quick introduction of relevant background knowledge to understand this thesis. This includes networks (Section 2.1), algorithmics (Section 2.2) and finally quantum mechanics (Section 2.3). We also give an overview of the current state of the art in Section 2.4 that gives some pointers for further reading.

2.1. Networks

A network may been seen as result of connecting computers such that they may communicate. The resulting structures may be very regular and easy to grasp, or they can be complex systems which are the subject of a line of research called Complex Networks. For this section we reference [Mie06] which gives a basic introduction of classical computer networks.

A network is usually represented in a graph, as can be seen in Fig. 2.1. All computers in a network are represented by a node which is also known as a vertex, while the connections are represented by edges. In this thesis we will restrict ourselves to *simple* graphs, where an edge may be used in both directions, edges are unweighted, there is at most one edge between two vertices, and there are no edges to a vertex itself (loops). Let the graph be defined as $G = (V, E)$, where V is the set vertices and E the set of edges. Because the graph is undirected it is assumed that $\{v, u\} \in E = \{u, v\} \in E$, as edges are sets of two vertices. Then the degree of a vertex $v \in V$ is

$$\deg(v) = \sum_{(v,u)\in E} 1\,. \tag{2.1}$$

And the set of neighbours of a vertex N is

$$N(v) = \{u | (v, u) \in E\}\,. \tag{2.2}$$

A path in a network is a sequence of nodes prescribing in which order nodes are visited to travel from a starting vertex to the end vertex. A path P_{α_1,α_i} from α_1 to α_i, is expressed as the following list

$$P_{\alpha_1,\alpha_i} = [\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, \alpha_i]\,.$$

This path can only be valid if all edges $\{\alpha_{j-1}, \alpha_j\} \in E$ for $j \in [1, 2, \ldots, i]$. Let us define the distance function

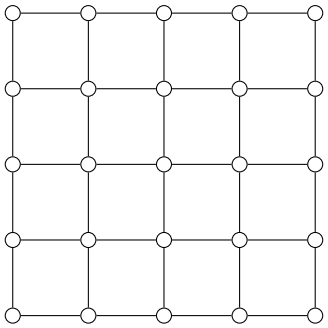


Figure 2.1: A 5×5 grid network, depicted as a graph. Every circle is a computer node in the network, and a line between nodes (an edge) represents a connection.

as the minimum length of a path required to reach v from u, and ∞

$$d(u,v) = \begin{cases} \min\{|P_{u,v}|\} & \text{if } P_{u,v} \text{ exists,} \\ \infty & \text{otherwise.} \end{cases} \tag{2.3}$$

Then a graph is called *connected* if for all $v, u \in V$

$$d(u,v) \neq \infty\,, \tag{2.4}$$

i.e. there is a path between any two vertices. A related quantity is the *connectivity*, which expresses the minimum number of edges or vertices that have to be removed, in order for a graph to become unconnected. We precisely define the edge connectivity $\lambda(G)$ as

$$\lambda(G) = \min_{E' \subseteq E} \{|E'| : G' = (V, E \setminus E') \text{ is not connected}\}\,. \tag{2.5}$$

Another interesting network quantity is the diameter $D(G)$

$$D(G) = \max_{x,y \in V} \{d(x,y)\}\,, \tag{2.6}$$

which expresses the maximum distance between any two vertices.

2.2. Algorithmics

A computer program is nothing more than a long list of basic instructions a computer must perform, this list of instructions is also called an algorithm. When given an algorithm for a certain problem, it is crucial to assess not only its correctness, but also how fast it is and how much memory it uses. We will introduce the reader to a way of thinking about algorithms which characterises the usability on a real computer, formalised in Algorithm Analysis.

But what is exactly meant by a small amount of time or when does an algorithm use little memory? Algorithm Analysis tries to address these issue by introducing constructs for classification of the complexity of algorithms, which in our case is the time complexity (how long the program runs) and the space complexity (how much space the program needs). For a more extensive introduction we refer to Sipser [Sip06] for an introduction to the theory of computation, and Kleinberg & Tardos [KT06] for algorithm design. We will mostly restrict ourselves to *worst-case analyses* where the goal is to upper bound the complexity of an algorithm.

In *run-time analysis* we concern ourselves with the number of instructions a program will perform depending on the input size. Say we have some algorithm in Qoogle Maps which finds the fastest way to travel between any two of the N cities on a map. Suppose that the algorithm requires at most $2N^2 \log N + 50N + 100$ steps to calculate this path. Here a step may be defined as one instruction of a normal computer processor. (In formal Algorithm analysis Turing Machines are used to define what an instruction is [Sip06], but that is not relevant to our purposes.) For small N we can see that $50N + 100$ will be a significant contributor to the run-time complexity, but the $2N^2 \log N$ will overtake its contribution at $N = 19$. For algorithms we are mostly interested in the behaviour for large N, which is why we use the O (big-O) notation. Given some runtime function $T(n) \in \mathbb{R}^+$ of an algorithm, then it is asymptotically upper-bounded by $O(f(n))$ for $f(n) \in \mathbb{R}^+$, some constant $c > 0$, and an integer constant $n_0 > 1$ if [GT10]

$$T(n) \leq c \cdot f(n)\,, \qquad \text{for } n \geq n_0\,. \tag{2.7}$$

The Qoogle Maps run-time function can be written as

$$T(N) = 2N^2 \log N + 50N + 100\,.$$

Then an $O(f(N))$ which upper bounds $T(N)$ is

$$f(N) = N^2 \log N\,.$$

However, $f(N) = N^3$ would also be a valid answer as $\log N$ is upper bounded by $O(N)$.

A *space analysis* uses many of the same properties as run-time analysis, but then applied to the memory used by the algorithm depending on the input size. Again we here keep the definition of unit of memory unspecified, but it may be considered within a constant factor of bits (this is also defined using Turing Machines [Sip06]). We look again at Qoogle Maps, which requires N for storing the cities and at most $N(N-1)$ for storing the roads between cities, thus requiring $N + N(N-1)$ total space. We can upper bound the worst-case space complexity of the Qoogle Maps by $O(N^2)$.

2.2.1. Routing Algorithms

Routing must be performed f some vertex in the network wants to communicate with another vertex that is not a neighbour, such that they are able to send data to each other. When performing routing on a network, we usually want to calculate All Pairs of Shortest Paths (APSP) in the graph. A common algorithm which calculates the distance between all vertices is the Floyd-Warshal algorithm [Flo62]. A more hierarchical structure is used for the Internet, called the Border Gateway Protocol (BGP), which clusters large areas of the Internet and routes between them [RLH06]. But within each cluster still what is essentially an APSP problem must be performed However, storing all shortest paths that a node may ever use, costs a significant amount of memory. An amount that is linear in the number of nodes in the network, thus an APSP solution is insufficient.

Instead, it is possible to calculate the shortest path on demand. The standard algorithm for finding a single shortest path in a graph is called Dijkstra's algorithm [KT06, Ch. 4.4]. In a nutshell, it increases its search radius in steps, looking at the vertices that are closest to the starting vertex first. The runtime of Dijkstra' algorithm is $O(|E| \log |V|)$, but also requires every node to store the entire network topology resulting in a space complexity of $O(|E| + |V|)$ per node, since each edge and each vertex in the graph must be stored in every node. Thus Dijkstra's algorithm is also insufficient. We instead give a routing algorithm which has a logarithmic run-time, and also a logarithmic memory size. This is only possible by limiting the information every node needs to its direct surroundings, which is also called a *local* algorithm.

2.3. Basic Quantum Mechanics

In this section we introduce the parts of quantum mechanics that are relevant to understanding this thesis. For a more thorough understanding we refer to the self-contained books by Nielsen and Chuang [NC10] or by Mark Wilde [Wil13], which are suitable for people new to the field of quantum mechanics.

Quantum mechanics is a new field, but still older than that of modern Computer Science. Discrete energy states in physical systems were first hypothesised by Ludwig Boltzman in 1877 and this was later followed up by the work of Henrich Hertz, Max Planck and Albert Einstein. Since then, quantum mechanics come to full speed for experimental implementations of a quantum computer. Providing a mathematical framework for physical theories, quantum mechanics allows us to explain the behaviour of matter and energy on the scale of atoms [NC10].

Originally rejected by Einstein, Podolsky and Rosen (EPR) in their famous EPR paper [EPR35], quantum mechanics has survived the rigorous testing performed so far. Later, Bell published his paper disproving the possibility of a local hidden variable model that could reproduce the predictions of quantum mechanics in what is now known as Bell's Theorem [Bel64]. A simple game was later proposed in [Cla+69] where the optimal classical strategy achieves a lower success rate than a quantum strategy. The optimal bound on a classical strategy has been called the CHSH inequality [Cla+69], which was later experimentally exceeded using quantum strategies [AGR81; ADR82]. Another question was whether a quantum state may be cloned, which was originally disproved in [WZ82] This has far-reaching implications in for example quantum cryptography, where it is not possible to perfectly intercept communication between two parties.

Quantum cryptography studies the use of quantum mechanics for securing communication between parties from unauthorized access. One important application of quantum mechanics is Quantum Key Distribution (QKD), where a private key is transmitted in such a way that it may only be intercepted with vanishingly low probability. In QKD the goal is to have *information-theoretic* security, where even an adversary with unlimited computational power cannot decrypt the message [Weh08], assuming only that the quantum mechanics framework is correct. One of the most famous publications in this field is the BB84 protocol [BB84], which provably achieves secure QKD [LC99; SP00]. It can even be shown that if the eavesdropped is only bounded by the speed of light (no-signaling) that secure key distribution is still possible [BHK05].

An important application of quantum computing is the quantum Fourier transform (QFT), which achieves an exponential speed-up compared to the classical discrete Fourier transform [NC10]. Algorithms that use the QFT are the Deutsch-Jozsa algorithm for deciding if a function is *constant* (always outputs '0' or '1') or *balanced* (output '0' half the time and '1' the other half) in $O(1)$, as opposed to $O(2^{n-1})$ for a classical algorithm [DJ92]. Shor's algorithm for finding the prime factors p and q of an integer $r = pq$ in $O(n^2 \log n \log \log n)$ as opposed to exponential run-time for a classical algorithm [Sho97]. Unrelated to the QFT, but nonetheless a major discovery is Grover's algorithm for finding entries in a database in $O(\sqrt{n})$ as opposed to an expected $\frac{n}{2}$ for any classical algorithm [Gro96]. Thus giving a quadratic speed-up for any exhaustive search.

Physically, quantum bits (qubits) may be implemented with various techniques that exhibit quantum behaviour. One such technique uses nitrogen-vacancy centres in diamond where an electron is stored, of which the spin may be used to represent a qubit [Ber+13]. Another technique is called trapped ions, where an ion is stabilised such that its energy state represents the quantum state [DM10]. Both techniques use photons to perform quantum communications, and have the possibility of being used in a quantum network. There are

more possible physical implementations of qubits, but it is yet unclear how these can be used in a quantum network and thus not relevant for this thesis.

2.3.1. Mathematical Framework

We begin by introducing the two quantum states $|0\rangle$ and $|1\rangle$ (pronounced "ket 0" and "ket 1") using bra-ket notation, which are similar to the 0 bit and 1 bit [NC10]

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.8}$$

These states are represented as 2-dimensional vectors. The conjugate transpose of these states are also denoted as $\langle 0|$ and $\langle 1|$ (pronounced "bra 0" and "bra 1")

$$\langle 0| = |0\rangle^\dagger = \begin{pmatrix} 1 & 0 \end{pmatrix}, \qquad \langle 1| = |1\rangle^\dagger = \begin{pmatrix} 0 & 1 \end{pmatrix}. \tag{2.9}$$

Now it is easier to compactly denote the products between these states, e.g.

$$\langle 0|0\rangle = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1, \qquad |0\rangle\langle 1| = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}. \tag{2.10}$$

However, quantum states do not only consist of a '0' and '1' such as in classical computers, but also all linear combinations between $|0\rangle$ and $|1\rangle$. Such a superposition of '0' and '1' can be described in a general one-qubit state $|\psi\rangle$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \tag{2.11}$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. Thus $|\psi\rangle$ is a normalised vector in the 2D complex vector space $\mathbb{C}^2$.

We can define a two-qubit state $|\psi\rangle$ as follows

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} \tag{2.12}$$

Or for any number of qubits

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle, \tag{2.13}$$

where

$$\sum_x |\alpha_x|^2 = 1. \tag{2.14}$$

A multiple qubit state is a system where multiple qubits are allowed to interact. This leads to behaviour that is very unlike classical qubits, such as entanglement (Section 2.3.4).

A useful operator for multi-qubit states is the *tensor-product* $\otimes$ which extends an n dimensional state $|\psi\rangle$ and a p dimensional state $|\phi\rangle$ to one np dimensional state. The tensor product can be applied to the vector representation, where it is known as the Kronecker product, which is defined as [NC10]

$$|\psi\rangle \otimes |\phi\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} \otimes |\phi\rangle = \begin{pmatrix} \alpha_0|\phi\rangle \\ \alpha_1|\phi\rangle \\ \vdots \\ \alpha_n|\phi\rangle \end{pmatrix}. \tag{2.15}$$

It is common to leave out the tensor product similarly to multiplication

$$|\psi\rangle|\phi\rangle \equiv |\psi\rangle \otimes |\phi\rangle. \tag{2.16}$$

The tensor product has the following properties:

1. For a scalar z and states $|\psi\rangle$ and $|\phi\rangle$

$$z(|\psi\rangle \otimes |\phi\rangle) = z|\phi\rangle \otimes |\psi\rangle = |\phi\rangle \otimes z|\psi\rangle. \tag{2.17}$$

2. It also has the right-distributive property among states, given some other state $|\zeta\rangle$

$$(|\psi\rangle + |\phi\rangle)\,|\zeta\rangle = |\psi\rangle\,|\zeta\rangle + |\phi\rangle\,|\zeta\rangle\ . \tag{2.18}$$

3. And also left distributivity

$$|\zeta\rangle\,(|\psi\rangle + |\phi\rangle) = |\zeta\rangle\,|\psi\rangle + |\zeta\rangle\,|\phi\rangle\ . \tag{2.19}$$

An example for the tensor product with matrices in

$$|0\rangle \otimes |1\rangle = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\1\\0\\1\end{pmatrix} \tag{2.20}$$

An example of an entangled state is

$$\frac{|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle}{\sqrt{2}} = \begin{pmatrix}\frac{1}{\sqrt{2}}\\0\\0\\\frac{1}{\sqrt{2}}\end{pmatrix}\ . \tag{2.21}$$

You can see that if the first qubit 0, then the second qubit is that as well, and vice versa.

2.3.2. Quantum Operations

Of course it is possible to perform operations on the quantum states so that some computation is performed, or a different requested state is reached. The first type of operations that we look at are one-qubit gates. The Pauli-X operation or bit-flip gate is specified as

$$X = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}\ . \tag{2.22}$$

We can see that if we apply this gate to an arbitrary state $|\psi\rangle$ we get

$$X\,|\psi\rangle = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}\begin{pmatrix}\alpha\\\beta\end{pmatrix} = \begin{pmatrix}\beta\\\alpha\end{pmatrix}\ ,$$

which has flipped the $\alpha\,|0\rangle$ to $\alpha\,|1\rangle$ and $\beta\,|1\rangle$ to $\beta\,|0\rangle$. Some more common operations are Pauli-Z, Pauli-Y and the Hadamard gate

$$Z = \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}\ , \tag{2.23}$$

$$Y = \begin{pmatrix}0 & -i\\i & 0\end{pmatrix}\ , \tag{2.24}$$

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}\ . \tag{2.25}$$

You may have noticed that these gates meet a certain property: They are unitary. This means that for any gate U must hold that

$$U^\dagger U = UU^\dagger = \mathbb{I}\,, \tag{2.26}$$

where $U^\dagger$ is the adjoint or conjugate transpose of U: $U^\dagger = \overline{U^T}$ and $\mathbb{I}$ is the identity matrix. Indeed, all quantum gates have this property, so that the requirement $|\alpha|^2 + |\beta|^2 = 1$ for a state in Eq. (2.11) is not broken after a gate is applied, and so that any operator is invertible.

For two-qubit gates there is one important operator, the CNOT, which is specified as

$$\text{CNOT} = \begin{pmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{pmatrix}\ . \tag{2.27}$$

This gate will invert the second qubit it operates on, depending on the value of the first bit and can be expressed as $|A, B\rangle \to |A, B \oplus A\rangle$, where $\oplus$ is modulo 2 addition. Of course the CNOT gate also meets the requirement that it is unitary.

The fact that gates have to be unitary has a certain effect on the possible operations, they have to be *reversible*. It must always be possible to apply $U^\dagger$ on the state $U|\psi\rangle$ to get back $|\psi\rangle$. Thus non-reversible operations such as a XOR or NAND are not possible in the quantum world. For example, while there is unitary that transforms $|A, B\rangle \to |A, B \oplus A\rangle$, there is not unitary which does some transformation $|A, B\rangle \to |A \oplus B\rangle$. Since from the value $|A \oplus B\rangle$ it is not revert to the values of A and B. Operations may be extended to any number of qubits, as long as the meet the unitary requirement.

Using the tools so far, we can prove that a quantum state $|\psi\rangle$ cannot be cloned. This was first shown in [WZ82], and we give a proof from [NC10]. Suppose the initial state is $|\psi\rangle \otimes |s\rangle$, where $|s\rangle$ is the state we want to clone $|\psi\rangle$ to. Then there must be some unitary U so that

$$|\psi\rangle \otimes |s\rangle \xrightarrow{U} U(|\psi\rangle \otimes |s\rangle) = |\psi\rangle \otimes |\psi\rangle \ . \tag{2.28}$$

If we then want also want to clone a state $|\phi\rangle$ it must also hold for the same 'cloning' unitary U that

$$U(|\phi\rangle \otimes |s\rangle) = |\phi\rangle \otimes |\phi\rangle \ . \tag{2.29}$$

We then take the inner product of Eqs. (2.28) and (2.29)

$$(\langle\phi| \otimes \langle s|)U^\dagger U(|\psi\rangle \otimes |s\rangle) = (\langle\phi| \otimes \langle\phi|)(|\psi\rangle \otimes |\psi\rangle)\,, \tag{2.30}$$

$$(\langle\phi|\psi\rangle \otimes \langle s|s\rangle) = \langle\phi|\psi\rangle \otimes \langle\phi|\psi\rangle \ , \tag{2.31}$$

$$\langle\phi|\psi\rangle = (\langle\phi|\psi\rangle)^2 \ . \tag{2.32}$$

The equation $x = x^2$ has only two solutions, $x = 0$ or $x = 1$. Thus either $|\phi\rangle$ and $|\psi\rangle$ are orthogonal so that $\langle\phi|\psi\rangle = 0$ or they are equal $|\psi\rangle = |\phi\rangle$ so that $\langle\phi|\psi\rangle = 1$. Thus there does not exist a cloning unitary for general $|\psi\rangle$ and $|\phi\rangle$.

2.3.3. Measurement

Now that we are able to create some basic quantum states and perform operations on them, how can the classical world get results from those quantum states? Quantum measurements allow us to gain information about the state the system was in. However, the measurements are different from what we are used to, they actually modify the state itself, something that may be surprising.

A set of measurements operators is defined as $\{M_m\}$, where m indicates the measurement outcome. The probability for measuring outcome m in a state $|\psi\rangle$ is given by

$$\Pr[m] = \langle\psi|M_m^\dagger M_m|\psi\rangle \ , \tag{2.33}$$

and the state after measurement is

$$\frac{M_m|\psi\rangle}{\Pr[m]} \,. \tag{2.34}$$

It is required that

$$\sum_m M_m^\dagger M_m = \mathbb{I}\,, \tag{2.35}$$

so that the probabilities of measurement sum up to one

$$\sum_m \Pr[m] = \sum_m \langle\psi|M_m^\dagger M_m|\psi\rangle = \langle\psi| \left(\sum_m M_m^\dagger M_m\right) |\psi\rangle = \langle\psi|\,\mathbb{I}\,|\psi\rangle = 1\,. \tag{2.36}$$

As an example, we can perform measurements in the *computational basis* $M_{\text{comp}} = \{|0\rangle_0\,, |1\rangle_1\}$ of the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. The probabilities of measuring an outcome M_x is given by $\Pr[x]$

$$\Pr[0] = \langle\psi|0\rangle\,\langle 0|\psi\rangle = |\alpha|^2\,,$$
$$\Pr[1] = \langle\psi|1\rangle\,\langle 1|\psi\rangle = |\beta|^2\,.$$

And we know that $|\alpha|^2 + |\beta|^2 = 1$, thus all probabilities indeed sum up to one. After the measurement with M_{comp} the state after measurement $|\psi'\rangle$ will be

$$|\psi'\rangle = \begin{cases} |0\rangle & \text{if measured } 0\,, \\ |1\rangle & \text{otherwise}\,. \end{cases} \tag{2.37}$$

From these results we can see that the measurement itself has determined the post-measurement state of $|\psi\rangle$.

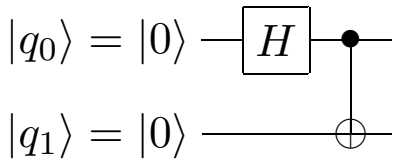


Figure 2.2: Entangling two qubits to form an EPR pair. First a Hadamard (the H in a box) is applied on q_0, then a CNOT is applied to form the Bell state $|\beta_{00}\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$ as show in Eq. (2.46). A CNOT is represented as the control gate with a black dot, and the changed qubit with an $\oplus$. Thus, if $|q_0\rangle = |1\rangle$ then $|q_1\rangle \to |q_1 \oplus 1\rangle$.

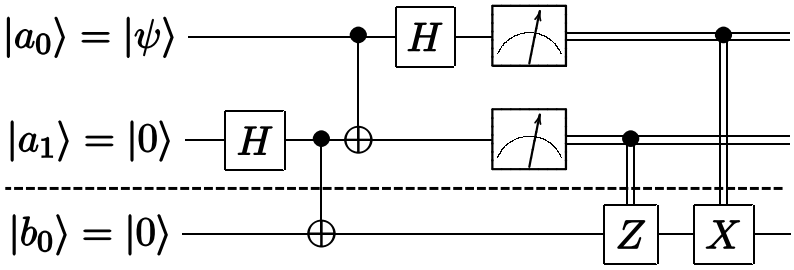


Figure 2.3: The teleportation of a qubit $|\psi\rangle$ from Alice to Bob, where Alice has the qubits a_0, a_1 and Bob b_0. First a Bell state is prepared using a_1 and b_0 according to Fig. 2.2. Next, the arbitrary state a_0 is entangled with part of the Bell state a_1 using a CNOT. Finally, a_0 and a_1 are measured after a Hadamard on a_0 (the box with a needle readout), to produce two classical correction bits (double lines) such that a conditional Z and conditional X may be applied on b_0. When Bob receives these classical bits from Alice, he may apply a correction on b_0 such that he ends up with $|b_0\rangle = |\psi\rangle$. So, as long as Bob receives the correction bits from Alice, he may move b_0 anywhere after the initial Bell state is created.

2.3.4. Entanglement

Another interesting property of quantum mechanics, is that two qubits can be *entangled*. This roughly means that there is a correlation of information between the two qubits. In contrast to entanglement, a *separable* state has the form [Weh15]

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \ . \tag{2.38}$$

When the state $|\psi\rangle$ is *not* separable, it is called entangled. It must then hold that

$$|\psi\rangle \neq |\psi_1\rangle \otimes |\psi_2\rangle \ . \tag{2.39}$$

We can find examples of entangled states in the Bell states or EPR states (after Einstein, Podolsky and Rosen) [NC10]

$$|\beta_{00}\rangle = |\Phi_+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \,, \tag{2.40}$$

$$|\beta_{10}\rangle = |\Phi_-\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}} \,, \tag{2.41}$$

$$|\beta_{01}\rangle = |\Psi_+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \,, \tag{2.42}$$

$$|\beta_{11}\rangle = |\Psi_-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \,. \tag{2.43}$$

It is not possible to break these states into two separate qubits, i.e. $|\beta_{ij}\rangle \neq |\psi_1\rangle \otimes |\psi_2\rangle$.

We can produce the $|\beta_{00}\rangle$ by performing a Hadamard and a CNOT operation

$$\begin{aligned} \mathrm{CNOT}_{AB}(H_A \otimes I_B)(|0\rangle_A \otimes |0\rangle_B) &= \mathrm{CNOT}_{AB}(H_A |0\rangle_A \otimes I_B |0\rangle_B) && (2.44)\\ &= \mathrm{CNOT}_{AB}\left(\frac{|0\rangle_A + |1\rangle_A}{\sqrt{2}} \otimes |0\rangle_B\right) && (2.45)\\ &= \frac{|00\rangle_{AB} + |11\rangle_{AB}}{\sqrt{2}} \,, && (2.46) \end{aligned}$$

where we have indicated the qubit being operated on with a subscript A and B. This same process can also be graphically shown, using the circuit diagram in Fig. 2.2.

Using entanglement it is possible to move a quantum state from one party, Alice, to another called Bob without the presence of a quantum channel. This is called quantum teleportation. In the beginning Alice and

Bob prepare a shared Bell state $|a_1 b_0\rangle = |\beta_{00}\rangle$ and may then move far apart. Note that $|\beta_{00}\rangle \neq |a_0\rangle \otimes |b_0\rangle$ since it is entangled, so we cannot separate it into two qubits. At some point in time Alice can then send an arbitrary one-qubit state $|\psi\rangle$ to Bob by following the circuit diagram in Fig. 2.3.

If we extend the picture a little bit and assume that Alice and Bob share an EPR pair $|\beta\rangle_{AB}$, and there is a third party called Charlie with whom Bob also shares an EPR pair $|\beta\rangle_{BC}$, then we can perform what is called entanglement swapping. By teleporting Bob's qubit in $|\beta\rangle_{AB}$ to Charlie it is possible to create an EPR pair between Alice and Charlie $|\beta\rangle_{AC}$. This process may be repeated multiple times, creating an entangled pair between nodes which are far apart [BVK98]. Once an entangled pair has been created between Alice and the far-away party, Alice may then send her quantum state using teleportation.

So far we have only talked about states that are entangled between two parties, but it is possible to extend this to any number of parties. The GHZ state (after Greenberger, Horne and Zeilinger) is an entangled state shared between 3 parties, it is defined as [Gre+90; Wil13]

$$|\Psi_{\text{GHZ}}\rangle = \frac{|000\rangle + |111\rangle}{\sqrt{2}}\,. \tag{2.47}$$

Indeed, we can define a similar state for N parties

$$|\Psi\rangle = \frac{|0\rangle^{\otimes N} + |1\rangle^{\otimes N}}{\sqrt{2}}\,, \tag{2.48}$$

where $|\psi\rangle^{\otimes N} = |\psi\rangle_1 \otimes |\psi\rangle_2 \ldots \otimes |\psi\rangle_N$. An interesting property of the GHZ state is that if the qubits in a GHZ state are distributed over multiple parties, then if any one party loses their qubit, the entire state becomes disentangled. Thus if anyone decides not to cooperate, all quantum advantage is lost.

2.3.5. Mixed States

We can generalise the quantum states we have seen so far, which are vectors, to matrices. The state $|\psi\rangle$ is equivalent to the *density operator* $\rho = |\psi\rangle\langle\psi|$ [Weh15]. Note that the rank of ρ is one, as it has one eigenvalue with eigenstate $|\psi\rangle$. Density operators with rank one, are also called pure states.

Opposed to pure states, are mixed states. One example of a mixed states is one where either $|\psi_1\rangle$ is prepared with $1/2$ probability, or $|\psi_2\rangle$ is prepared with $1/2$ probability. It is not the case that these two states are in a superposition, because only one state occurs at any time. The real state is a mixture of the two

$$|\psi\rangle = \frac{1}{2}|\psi_1\rangle\langle\psi_1| + \frac{1}{2}|\psi_2\rangle\langle\psi_2|\,.$$

If a state $|\psi_x\rangle$ is prepared with some probability $\Pr[x]$, then the mixed state will be

$$\rho = \sum_x \Pr[x]\,|\psi_x\rangle\langle\psi_x|\,. \tag{2.49}$$

Applying a unitary U to a mixed state performed as

$$\rho \xrightarrow{U} U\rho U^\dagger \tag{2.50}$$

Measuring mixed states is also slightly different than measuring pure states. The measurement probability of x given a set of measurement operators $\{M_x\}$ is given by

$$\Pr[x] = \operatorname{tr}(M_x^\dagger M_x \rho)\,, \tag{2.51}$$

where the trace operator is defined as

$$\operatorname{tr}(M) = \operatorname{tr}\begin{pmatrix} a_{00} & \ldots & a_{n0} \\ \vdots & \ddots & \vdots \\ a_{0n} & \ldots & a_{nn} \end{pmatrix} = \sum_{i=0}^{n} a_{ii}\,. \tag{2.52}$$

A property that we use often is that the trace is cyclic, i.e.

$$\operatorname{tr}(AB) = \operatorname{tr}(BA)\,. \tag{2.53}$$

The post-measurement state is

$$\frac{M_x \rho M_x^\dagger}{\Pr[x]}\,. \tag{2.54}$$

One useful application of mixed states is the representation of subsystem of a larger multipartite quantum state. If we have the shared state $|\psi\rangle_{AB}$ how do write out the state that only Alice has or that Bob has? To this end we use the *partial trace* operator [NC10]

$$\rho_A \equiv \mathrm{tr}_B(\rho_{AB})\,, \tag{2.55}$$

and it is defined as [Weh15]

$$\mathrm{tr}_B(\rho_{AB}) = \sum_{ijkl} \gamma_{ij}^{kl} \left|i\right\rangle \left\langle j\right| \otimes \mathrm{tr}(\left|k\right\rangle \left\langle l\right|) = \sum_{ij} \left(\sum_k \gamma_{ij}^{kk} \right) \left|i\right\rangle \left\langle j\right| .$$

If we then apply the partial trace on a separable state, we get

$$\mathrm{tr}_B(\rho_{AB}) = \mathrm{tr}_B(\rho_A \otimes \sigma_B) = \rho_A \,\mathrm{tr}(\sigma_B) = \rho_A\,.$$

Thus the global state ρ_{AB} does not give more information about the separate state ρ_A. However, if we apply the partial trace to one of the Bell states we get

$$\begin{aligned}
\mathrm{tr}_B(\left|\beta_{00}\right\rangle \left\langle\beta_{00}\right|) &= \frac{\mathrm{tr}_B(\left|00\right\rangle \left\langle 00\right| + \left|00\right\rangle \left\langle 11\right| + \left|11\right\rangle \left\langle 00\right| + \left|11\right\rangle \left\langle 11\right|)}{2} \\
&= \frac{\left|0\right\rangle \left\langle 0\right| \left\langle 0|0\right\rangle + \left|0\right\rangle \left\langle 1\right| \left\langle 1|0\right\rangle + \left|1\right\rangle \left\langle 0\right| \left\langle 0|1\right\rangle + \left|1\right\rangle \left\langle 1\right| \left\langle 1|1\right\rangle}{2} \\
&= \frac{\left|0\right\rangle \left\langle 0\right| + \left|1\right\rangle \left\langle 1\right|}{2}\,,
\end{aligned}$$

which is a mixed state. Thus Alice does not have maximal knowledge about her state, but if we include knowledge about Bob's state, then we would have a pure state $|\beta_{00}\rangle$.

We have seen that the state Alice is by itself a mixed state, which does not contain any information about the entanglement with Bob. The state $\mathbb{I}/2$ is also known as the *maximally mixed state* π, which is a classical uniform distribution over orthogonal states [Wil13]

$$\pi = \frac{1}{d} \sum_{x \in \mathcal{X}} \left|x\right\rangle \left\langle x\right| = \frac{\mathbb{I}}{d}\,,$$

where d is the dimension of the state. Measuring the maximally mixed in any basis will result in a uniform distribution over the outcomes.

2.3.6. Noise

Thus far we have assumed that we have closed quantum systems that do not interact with the outside world. Unfortunately, quantum computers are delicate instruments that are influenced by the environment. For example, if a qubit is represented by position of an electron then the charged particles around it influence the quantum system.

So far we have been using $|0\rangle$ and $|1\rangle$ as our main tools for computation. Because of its simplicity it is also called the *computational basis*. Nevertheless, there are many more bases which are of use in algorithms, such as the *Hadamard basis* defined by

$$M_{\text{Hadamard}} = \left\{ \left|+\right\rangle = \frac{\left|0\right\rangle + \left|1\right\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \left|-\right\rangle = \frac{\left|0\right\rangle - \left|1\right\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \right\}$$

or any other combination of two vectors that are orthogonal, which means that any vector $|v\rangle \in \mathbb{C}^2$ can be written as a linear combination of the basis $\{|v_1\rangle, |v_2\rangle\}$.

In the previous example of noise, we have seen that all off-diagonal elements of ρ disappear. The components along the diagonal are precisely align with the computational basis that we used. It is, however, possible that the noise is applied along a different basis if we used some different unitary U, such that different components of the state are lost.

There are many different noise models that affect the state in various way. First there is a type of noise, called *bit flip channel*, that applies an X to the state with some probability $1 - p$

$$E(\rho) = p\rho + (1-p) X \rho X^\dagger\,.$$

The *phase flip channel* is similar but applies a Z

$$E(\rho) = p\rho + (1-p) Z \rho Z^\dagger\,.$$

A different kind of noise the depolarizing noise, which depolarizes ρ to the maximally mixed state $\mathbb{I}/2$ with probability $1-p$

$$E(\rho) = p\rho + (1-p)\frac{\mathbb{I}}{2}\,.$$

There are of course many more types of noise models, but a basic understanding of how noise acts on a quantum system is sufficient for understanding the next topic, entanglement purification.

2.3.7. Entanglement Purification

Let us assume that we have n entangled states that are not perfectly entangled. Quantum algorithms frequently require perfectly entangled states (singlets) to operate, is it possible to convert these n states to m near-perfect singlets? This situation occurs often in quantum communication, where channels are noisy but the protocols require some high-quality states to function. We will discuss a technique that is often used to combat noise called *entanglement purification* [Ben+96], which is also know as *entanglement distillation* [NC10]. For this thesis we require entanglement distillation to create perfect entanglement between network nodes over long distances.

We will illustrate how purification works according to a simplified example [Wil13, Ch. 18]. Assume there is some pure state shared between Alice and Bob that is partially entangled for $\theta \in [0, \pi/4]$

$$|\psi\rangle = \cos(\theta)\,|00\rangle_{AB} + \sin(\theta)\,|11\rangle_{AB}\,. \tag{2.56}$$

Alice and Bob do not have just one, but two of these states and they want a maximally entangled state to use in a quantum algorithm. The complete state $|\Psi\rangle$ of the two partially entangled systems is

$$\begin{aligned}|\Psi\rangle = |\psi\rangle_1 \otimes |\psi\rangle_2 &= \cos^2(\theta)\,|00\rangle_{AB}\,|00\rangle_{AB} + \sin^2(\theta)\,|11\rangle_{AB}\,|11\rangle_{AB} \\ &\quad + \cos(\theta)\sin(\theta)\,|00\rangle_{AB}\,|11\rangle_{AB} + \sin(\theta)\cos(\theta)\,|11\rangle_{AB}\,|00\rangle_{AB} \\ &= \cos^2(\theta)\,|00\rangle_A\,|00\rangle_B + \sin^2(\theta)\,|11\rangle_A\,|11\rangle_B \\ &\quad + \cos(\theta)\sin(\theta)\,|01\rangle_A\,|01\rangle_B + \sin(\theta)\cos(\theta)\,|10\rangle_A\,|10\rangle_B\,.\end{aligned} \tag{2.57}$$

From this we observe that states with zero or two "ones" are not entangled, but the states with one "one" are maximally entangled. Thus we define a measurement M_i on Alice's system that measures the number of "ones" in the system i

$$\begin{aligned}M_H = \big\{ M_0 &= |00\rangle_A\,\langle 00|_A\,, \\ M_1 &= |01\rangle_A\,\langle 01|_A + |10\rangle_A\,\langle 10|_A\,, \\ M_2 &= |11\rangle_A\,\langle 11|_A \big\}\end{aligned} \tag{2.58}$$

Bob can perform the same measurement on his own system. There are three possible outcomes if Alice and Bob then measures $|\Psi\rangle$ with M_H on their systems. With probability $\cos^4(\theta) + \sin^4(\theta)$ we get M_0 or M_2 and the algorithm fails, because the resulting state is not entangled. Even so, we get the maximally entangled state with probability $2\cos^2(\theta)\sin^2(\theta)$

$$|\Psi'\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} = |\beta_{01}\rangle\,, \tag{2.59}$$

where β_{01} is one of the Bell states (2.42). It is possible to generalise this procedure to an arbitrary number of imperfectly entangled pure states [Wil13]. In the limit of the number of states the probability of failure becomes negligible and the dimension of the purified state grows exponentially.

For pure states the optimal algorithm for purification is known [Wil13], but for mixed states it is not. Mixed states are also of interest in this thesis, because noise creates mixed states in general. There are some algorithms known that perform purification on generalised mixed states [Ben+96; Hor+09], but the information so far is sufficient for understanding the thesis.

2.4. Related Work

We give an overview of the current state of the art for quantum networks, and implementations of quantum systems for networks. These systems have an impact on the physical limitations of networks, which influence the parameters in our model. Additionally, we give an overview of the state of the art in graph theory that has the closest relation to the networks we will construct.

2.4.1. Quantum Networks

The process of creating entanglement over a network is noisy, which is why it is non-trivial to create perfect EPR-pairs (singlets) between connected nodes using entanglement swapping. This problem can be alleviated by performing entanglement distillation, which uses multiple sets of noisy entangled pairs and distills one less noisy pair from them [Ben+96; Dür+99; Hor+09]. Because it is impossible to clone a quantum state, it is also not possible to build simple signal amplifiers. Closely related to distillation and error correction, there has recently been interest in creating a quantum repeater, which uses the ideas of distillation to be able to create entanglement over large distances [San+11; Mun+15]. These methods aim to create repeaters that do not require an exponential amount of resources in the distance between endpoints [Bri+98]. Normally the noise on the state grows exponentially with the distance, but with these new type of repeaters it may be possible to create distilled pairs at longer distances, that use only a logarithmic amount of noisy entangled pairs and a polynomial time overhead. We will specify quantum networks that have connections over large distances, which is why these techniques are also essential to this thesis.

There have also been developments on a higher level of abstraction, on the network layer, where this thesis also contributes. One field is percolation theory, where randomised networks spontaneously become fully connected once the probability of establishing a connection exceeds a percolation threshold [SA94]. An analogue also exists for quantum networks called entanglement percolation, where the question is how to most effectively establish entanglement between nodes in a quantum network. It turns out that transforming the quantum network using simple local operations and classical communication (LOCC) to a network that has a significantly lower threshold [ACL07; CC09], so that establishing a connection has a lower failure probability. Some unique applications of quantum networks are also being investigated, for example the synchronization of clocks around the world [Kom+14].

The specific implementations have an impact on the properties of the network. Properties such as the decay rate of entanglement in time, the coherence time of a qubit, the communication noise, the time needed to create entanglement, and the quantum memory size. All quantum technologies use photons to communicate over long distances, although the quantum memory itself is implemented in various ways. There has been a study of the feasibility of a quantum internet in [Kim08], which compared some of the technologies in that time. Currently, one possible implementation are spins in nitrogen-vacancy (NV) centres in diamond [Ber+13; Pfa+14]. The coherence time in NV centres is relatively long, but generating entanglement also takes a long time. Another possibility is the use of trapped ions [DM10], where they have an entanglement event approximately every 8.5 minutes [Moe+07]. Compared to the nanosecond scale coherence time, this is extremely long. A generalized overview for photon communications is given in [Mun+15].

2.4.2. Graph Theory

The properties we are interested in for a quantum network have much in common with graph theory, where a lot of research has already been performed. In this thesis we will use a recursive definition of graphs and that is why we looked into their definitions and properties. There is some work in relation to in graph coloring, where the goal is to assign a color to each vertex so that none of its neighbours have the same color while minimizing the total amount of different colors assigned. As it turns out, recursive graphs have properties that allow efficient colorization [Bea76; GL98] and many other linear-time algorithms [BPT92]. Another line of research within graph theory is graph homeomorphism, where by subdividing and smoothing edges the goal is to generate a subgraph which is equivalent to a secondary graph [FHW80]. For the analytical analysis of the graph, it may be desirable to have a mathematical definition. Spectral graph theory is a field that specialises in the properties of graphs defined in adjacency matrices [GYZ13]. Furthermore, there has been research into forming hierarchies in graphs for routing, which is relevant to our problem since we also define a certain hierarchy in our graph [Gei+08].

For our graphs we want to minimize the diameter while also limiting the degree of each node. The base structure of the graph is already fixed, but we can add edges to this graph to lower the diameter using entanglement. This problem, if turned into a decision problem with a YES and NO answer, is known as *Minimum Bounded Diameter Augmentation* [Cre+05] (MBDA). Unfortunately, this problem is NP-complete, so there do not exist polynomial-time algorithms for finding an optimal solution (under the assumption that P $\neq$ NP). It is, however, tractable by approximation for a constant node degree [Fra+13] (specifically an FPT 4-approximation). Because a spherical graph can also be drawn on the plane, it is also a planar graph [GT87]. Algorithms restricted to planar graphs are more tractable than general graphs. The MBDA problem is equivalent to solving the dominating set problem [LMS92], to which an Polynomial-Time Approximation Scheme (PTAS) is given in [Bak94]. Thus it may also be possible to approach this problem using a polynomial-time approximation algorithm and generate a set of edges E' so that for the graph $G = (V, E \cup E')$ the diameter $D(G)$ is minimized. An added difficulty will be limiting the degree of each node, so that a small quantum storage is sufficient.

3

Network Model

In this chapter we give a high-level overview of our network model and resolve our first two research questions detailed in Section 1.2.1. We will explain the intuition and the motivation behind the choices we have made and the algorithms that have been formulated. Furthermore, we will also give some important theorems that are later proved. The Section 3.2 will go into the technical details and the proofs themselves. In this work we will mostly disregard classical communication and routing, as those problems have already been solved.

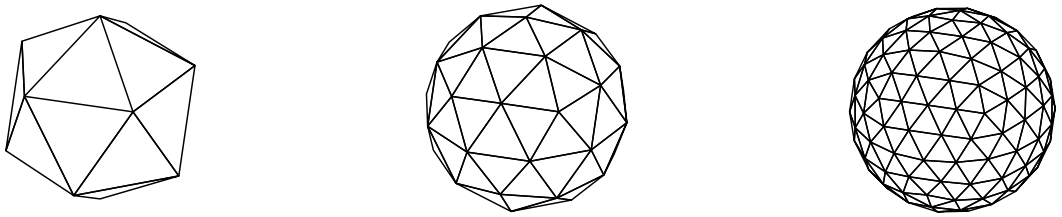
The starting point of our problem are satellites in space that can perform quantum communication with nearby satellites. Furthermore, we assume that the satellites are evenly spread out over the surface of a sphere, called the earth. We model this problem using a network graph. The satellites can be mapped to network nodes in the graph, while the satellites that can communicate are connected with edges. Nodes can perform entanglement swapping and entanglement purification to create entangled pairs between nodes that are not adjacent. We will add virtual links to this graph by approximating a sphere.

3.1. Approximating a Sphere

To the given graph we want to add edges that both decrease the diameter of the graph, and limit the degree of all nodes. We do this by starting with a base graph, the icosahedron, and performing a procedure on that graph until every vertex in it equals a vertex in the given graph. To approximate a sphere and have a uniform distribution we look at platonic solids, the only 5 three-dimensional polyhedra that are regular. We will add vertices to a platonic solid, in such a way that the resulting graph better approximates a sphere. For prior work we look at 3D modelling, where it is a common problem to approximate smooth surfaces from few polygons. A standard algorithm in 3D modelling for subdividing polyhedra is Loop subdivision [Loo87]. This algorithm requires triangular polygons, which leave only the tetrahedron, octahedron and icosahedron to choose from. Loop subdivision performs approximation iterations in what are called *subdivisions*. In each subdivision, a vertex is placed on the middle of edges and new nodes are connected to their nearby neighbours. The platonic solid which when subdivided resembles a sphere the most, is the icosahedron. We thus start with a graph representation of the icosahedron, which is scaled so that all 12 vertices are placed on the sphere.

We will use Loop subdivision to approximate the sphere, but save all edges that are generated during the algorithm. These edges will later function as the long-distance virtual links. In every subdivision iteration a vertex is placed on every edge, which is then connected to the endpoints of that edge, and also to nearby new vertices. An example of subdivision is given in Fig. 3.1, which illustrates how we can subdivide an icosahedron to approximate a sphere. The subdivision for one face is shown in detail in Fig. 3.1, which also includes the long-distance links. Each vertex in the subdivided graph can be placed on the sphere, so that they are distributed uniformly over the sphere. We assume that the number of vertices in the given graph can be generated by the subdivision algorithm, so once the graph has been subdivided enough times each vertex maps to one of the satellites.

After subdivision, we assume there is some procedure that will distribute entanglement over the graph, such that each edge is an entangled pair. The procedure that generates these long-distance entanglements runs continuously. Old entanglements may decohere, which means that they become useless after some time (on the scale of microseconds). However, once long-distance entangled links are established and purified, we need fewer entanglement swaps to communicate. Which in turn results in a lower error rate, because each entanglement swap introduces some noise.



(a) The base icosahedron (b) One subdivision (c) Two subdivisions

Figure 3.1: Approximating the sphere by subdividing edges of the icosahedron. By iteratively performing this procedure, the resulting graph closer and closer approximates a sphere.

Entanglement Distribution

So how is entanglement distributed so that the resulting network is equal to the subdivided graph? Let us assume that we have subdivided once, which results in 42 nodes (3.18) that are spread on every face of the icosahedron as given in Fig. 3.2b. The algorithm places every vertex in the graph on the sphere, such that the vertices are uniformly distributed. The physical connections of every vertex are contained in the edges that were generated last, the edges that are the shortest length. Thus physical network nodes only have to ability to perform direct quantum communications through black solid edges, they are the physical connections. Entanglement can be established between two adjacent nodes using the physical connection. Then, to create for example the edge $\{\alpha_1, \alpha_3\} \in E$, an entanglement swap is performed by $\beta_1 \in V$ on the entanglement that can be created using physical connections between $\{\alpha_1, \beta_1\}$ and $\{\beta_1, \alpha_3\}$. If purification of the entanglement is successful, perfect entanglement is established between α_1 and α_3. If it is not, the process is restarted to try again. Once successful, the nodes α_1 and α_3 can communicate directly using this entangled pair, as if they were adjacent. The same procedure is also followed by β_2 and β_3 to create $\{\alpha_1, \alpha_2\}$ and $\{\alpha_2, \alpha_3\}$ respectively.

For more subdivisions, such as in Fig. 3.2c, we perform the above procedure 3 times to create the edge $\{\alpha_1, \alpha_3\}$. First by γ_2 and γ_6 to create $\{\alpha_1, \beta_1\}$ and $\{\beta_1, \alpha_3\}$, then by β_1 to create $\{\alpha_1, \alpha_3\}$. This uses up the entanglement created between $\{\alpha_1, \beta_1\}$ and $\{\beta_1, \alpha_3\}$ though, so these edges will have to be created once again.

Graph Notation

Let us define some of the notation we use to describe subdivided graphs. The set of vertices in the subdivided graph G is V, and let E be the set of edges. The following notation is used for vertices

$$\alpha, \beta, \gamma, \eta, \pi \in V \qquad \text{all vertices,} \tag{3.1}$$
$$V_i \subseteq V \qquad \text{the vertices after } i \text{ iterations,} \tag{3.2}$$
$$L_i \subseteq V_i \qquad \text{the vertices generated in the } i\text{-th iteration.} \tag{3.3}$$

For references to edges we use

$$\{\alpha, \beta\} \in E \qquad \text{all undirected edges,} \tag{3.4}$$
$$E_i \subseteq E \qquad \text{the edges generated in the } i\text{-th iteration.} \tag{3.5}$$

And for the graph

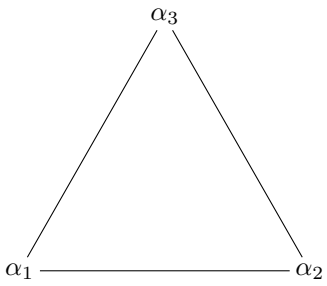
$$G = (V, E) \qquad \text{the simple graph,} \tag{3.6}$$
$$G_i = (V_i, \bigcup_{\ell=0}^{i} E_\ell) \qquad \text{the graph after } i \text{ iterations.} \tag{3.7}$$

There is a difference between the way V_i and E_i are defined, which will later be useful in the proofs. The intuition is that in subdivision iteration $i+1$ all vertices in V_i and all edges in E_i are used. Another useful term is that of a *layer*, which may be compared to the layers in an onion. For each subdivision another layer is added to the graph. So layer i would refer to all vertices and edges in L_i and E_i. When we refer to going up layers, that means that we are increase i for L_i and E_i. Conversely, going down the layers means decrease i. From now on, if we refer to V, E or the graph G then they refer to the subdivided graph, not general graphs.

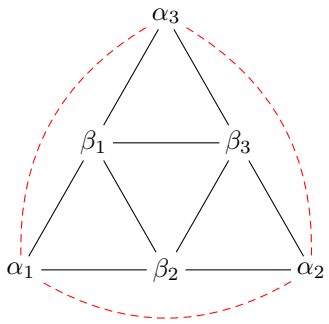
With the graph notation we can give the pseudocode for subdivision algorithm in Algorithm 3.1.

3.1.1. Graph Properties

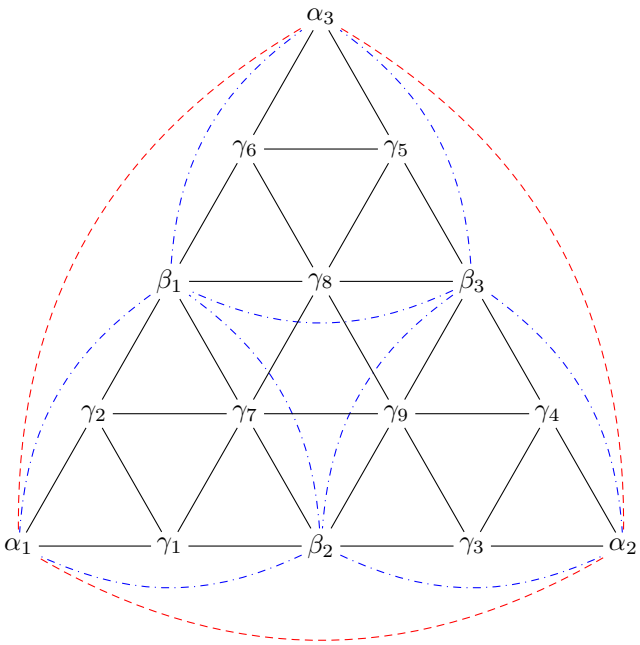
The subdivided icosahedron possesses several interesting properties relevant to our problem statement.



(a) A face of the icosahedron not subdivided, $k = 0$.



(b) A face of the icosahedron subdivided once, $k = 1$.



(c) A face of the icosahedron subdivided two times, $k = 2$.

Figure 3.2: Example steps in the subdivision algorithm on one face of the icosahedron, for given subdivide(k). In the first subdivision, β_i are placed on edges $\{\alpha_i, \alpha_j\} \in E_0$, and connected to other nearby β_i. The edges between α_i are kept in the graph (red, dashed). The second iteration better shows which edges are added on a subdivision. For example, γ_8 is connected to all 4 nearby γ_i, and the vertices β_1 and β_3 on the edge that was subdivided into γ_8 (black, solid). Again the edges of the previous graph are kept (red, dash & blue, dash dot). The colored and dashed (dotted) edges are long-distance connections that reduce the graph diameter. The physical connections are those edges that are on the last layer. So if $k = 2$ is the final subdivision, then the black solid edges in Fig. 3.2c are the physical connections.

First and foremost, the graph must have a small diameter, while still having a small degree for each vertex. A small diameter is desired to reduce the number of entanglement swap necessary to establish a communication. If two vertices are not adjacent, then entanglement swaps have to performed until they are, only then can they communicate securely. We will show that the graph diameter scales logarithmically in the number of nodes, i.e. an exponential increase in the number of nodes results in only a linear increase in diameter.

Proposition 3.1.1 (Graph Diameter)**.** *The diameter of the subdivided graph $D(G_k)$, where $k \in \mathbb{N}_0$, is*

$$D(G_k) \leq 2k + 3 = 4 \log_2 \left(\frac{|V| - 2}{10} \right) + 3 \,.$$

Furthermore, we upper bound the degree of each node. For every edge connected to a node, some quantum bit must be stored in a quantum memory. These quantum memories are currently limited in size, and for practical applications must be kept small. We show that the degree of every node scales logarithmically in the total number of nodes.

Algorithm 3.1: Subdivision algorithm of the icosahedron. A vertex is placed on each edge in E_i and then connected to nearby vertices. The new vertices are grouped in L_{i+1}, and the new edges in E_{i+1}.

Data : $G_0 = (V_0, E_0)$ the icosahedron.
Input : $k \in \mathbb{N}$, the number of subdivisions.
Output : $G = (V, E)$, the subdivided graph.

Function subdivide(k) **is**
  for $i \leftarrow 0$ **to** $k-1$ **do**
    $L_{i+1} \leftarrow \emptyset$
    $p \leftarrow$ empty Map : $V \to e$ `// A mapping of a vertex to its edge`
    for $e \in E_i$ **do**
      $\alpha \leftarrow$ midpoint of e on sphere.
      $L_{i+1} \leftarrow L_{i+1} \cup \{\alpha\}$
      $p(\alpha) \leftarrow e$ `// Remember which edge created` α
    end
    `// Generate the edges`
    $E_{i+1} \leftarrow \emptyset$
    for $\alpha \in L_{i+1}$ **do**
      $(e = (\beta_1, \beta_2)) \leftarrow p(\alpha)$
      $E_{i+1} \leftarrow E_{i+1} \cup \{\{\alpha, \beta_1\}, \{\alpha, \beta_2\}\}$ `// Create edges to vertices in` V_i
      `// The vertices` β_3, β_4 `that form triangles with` β_1 `and` β_2
      $\beta_3, \beta_4 \in V_i$: adjacent to β_1 and β_2
      triangles $\leftarrow$ edges in $\{\beta_1, \beta_2, \beta_3\}, \{\beta_1, \beta_2, \beta_4\}$
      $\mathcal{W} \leftarrow \{\gamma : p(\gamma) \in \text{triangles}\} \setminus \{\alpha\}$
      $E_{i+1} \leftarrow E_{i+1} \cup \{\{\alpha, \gamma\} : \gamma \in \mathcal{W}\}$ `// Create edges to vertices in` L_{i+1}
    end
    $V_{i+1} \leftarrow V_i \cup L_{i+1}$
  end
  return $G_k = (V_k, \bigcup_{i=0}^{k} E_i)$
end

Proposition 3.1.2 (Vertex Degree)**.** *The degree of every vertex $v \in V$ in the subdivided graph as a function of $|V|$ is upper bounded by*

$$\deg(v) \leq 12 \log_2 \left(\frac{|V| - 2}{10} \right) + 6.$$

3.2. Technical Details

In this section we will prove properties of the subdivided graph that are important for the physical implementation, as well as the usefulness of the network model. We prove how the number of subdivisions k relates to the number of nodes $|V|$ and edges $|E|$. Additionally, what the maximum degree is of any node in the graph, which is relevant for the amount of quantum memory required to implement such a node. Furthermore, we look at the diameter of the graph. With a small diameter we know that few entanglement swaps are necessary to connect sender and receiver.

First we look at the size of $|V|$ and $|E|$ depending on the number of subdivisions k. The subdivision algorithm (Algorithm 3.1) consists of multiple iterations, let i be the current iteration. Per subdivision, a vertex is placed on each edge $e \in E_i$. A new vertex is connected to 2 to vertices in V_i, and 4 nearby and new vertices in L_i. The edges to the new vertices should not be counted double, thus every new vertex adds $(2 + 4/2)|E_{k-1}| = 4|E_{k-1}|$ edges. The icosahedron starts with 30 edges, so that

$$|E_i| = \begin{cases} 30 & \text{if } i = 0\,, \\ 4|E_{i-1}| & \text{otherwise} \end{cases} \tag{3.8}$$

$$= 4^i \cdot 30\,. \tag{3.9}$$

The total number of edges is

$$|E| = |\bigcup_{i=0}^{k} E_i| = \sum_{i=0}^{k} 4^i \cdot 30 \tag{3.10}$$

$$= \frac{30(1-4^{k+1})}{1-4} \tag{3.11}$$

$$= 10 \cdot 4^{k+1} - 10\,, \tag{3.12}$$

where we have used the direct formula for a geometric series. A vertex is placed on the midpoint of each edge in E_i, so that the number of vertices V_i (3.2) is

$$|V_i| = \begin{cases} 12 & \text{if } i = 0 \\ |V_{i-1}| + |E_{i-1}| & \text{otherwise} \end{cases} \tag{3.13}$$

$$= |V_{i-2}| + |E_{i-2}| + |E_{i-1}| \tag{3.14}$$

$$= |V_0| + |E_0| + \ldots + |E_{i-1}| \tag{3.15}$$

$$= 12 + \sum_{l=0}^{i-1} 4^l 30 \tag{3.16}$$

$$= 12 + \frac{30(1-4^i)}{1-4} \tag{3.17}$$

$$= 10 \cdot 4^i + 2\,. \tag{3.18}$$

Then the total number of nodes $|V| = |V_k|$.

Next we analyse the diameter and degree of the graph. A small degree will allow the algorithm to run with smaller quantum memories, while a smaller diameter is desired to have less entanglement swaps per communication. Less swaps results in a lower error rate, since each swap introduces noise into the state.

Proposition 3.2.1 (Graph Diameter)**.** *The diameter of the graph $D(G_k)$, where $k \in \mathbb{N}_0$ is the number of subdivision iterations, is*

$$D(G_k) \leq 2k + 3 = 4\log_2\left(\frac{|V|-2}{10}\right) + 3\,.$$

Proof. We give a proof using a recurrence relation over k. Let $D(G)$ be the diameter function on a graph G. Let $G_k = (V_k, \bigcup_{l=0}^{k} E_k)$ be the graph at iteration k.
Basis: $D(G_0) \leq 2 \cdot 0 + 3 = 3$, which holds because the icosahedron has a diameter of 3.
Induction hypothesis: Assume that $D(G_{k-1}) = 2(k-1) + 3$.
Induction: For any vertex $\alpha \in L_k$ it is possible to reach a vertex in layer L_{k-1} in one step. Thus

$$D(G_k) \leq D(G_{k-1}) + 2 \tag{3.19}$$

$$\overset{\text{IH}}{=} 2(k-1) + 3 + 2 \tag{3.20}$$

$$= 2k + 3 \tag{3.21}$$

$$\overset{(3.23)}{=} 4\log_2\left(\frac{|V|-2}{10}\right) + 3\,. \tag{3.22}$$

As such the diameter upper bound is proven. □

Proposition 3.2.2 (Vertex Degree)**.** *The degree of every vertex $v \in V$ in the subdivided graph as a function of $|V|$ is upper bounded by*

$$\deg(v) \leq 12\log_2\left(\frac{|V|-2}{10}\right) + 6.$$

Proof. We show that the degree is upper bounded by calculating the number of vertices any vertex is connected to, when there have been k subdivisions. The exact formula for $|V_k|$ is found in Eq. (3.18). For

some number of subdivisions k

$$|V_k| = 10 \cdot 4^k + 2 \iff k = \log_4\left(\frac{|V_k| - 2}{10}\right) = 2\log_2\left(\frac{|V_k| - 2}{10}\right). \tag{3.23}$$

A vertex on the base icosahedron $\alpha \in V_0$ is connected to 5 vertices $\beta \in V_0$. However, any other vertex $\alpha \in L_i$, for $i \neq 0$, is connected to 6 vertices $\beta \in V_i$, because of how the algorithm connects new vertices to their parents and their neighbours. In every subdivision iteration $j : j > i$ a new midpoint vertex γ is added to L_{j+1}, and an edge $\{\alpha, \gamma\} \in E_{j+1}$ as well. This edge is in turn again subdivided on the next iteration $(j+1)$. Because the degree of the vertex α is the same as its number of edges, each vertex will add as many nodes as its degree in each subdivision iteration. Thus degree of a vertex $\alpha \in V$ is upper bounded by

$$\deg(\alpha) \leq \begin{cases} 5(k+1) & \text{if } \alpha \in V_0 \\ 6k & \text{otherwise,} \end{cases} \tag{3.24}$$

$$< 6(k+1) \tag{3.25}$$

$$= 12\log_2\left(\frac{|V| - 2}{10}\right) + 6\,. \tag{3.26}$$

Thus proving the upper bound on the degree of every vertex. □

Thus both the degree and the diameter are logarithmic in the number of vertices $|V|$. Which means that these parameters scale well for large $|V|$.

4

Routing Algorithm

It must also be possible to route on the subdivided icosahedron, preferably in a manner that is both fast and requires little (classical) memory, as posed by our last research question in Section 1.2.2. To that end, we started looking into defining structures of the graph. With knowledge of these structure it is possible to define such a routing algorithm. We will give a high-level overview of our approach and reasoning for this algorithm. In Chapter 5 we will go into the technical details of the results and the proofs.

4.1. Graph Structure

An important property of every vertex, except those on the lowest layer, is that it has been generated from an edge. This edge has two endpoints, which in turn are also generated from edges, etc. The relation structure that this assumes is tree-like, but may contain splitting and joining branches. Because two vertices generate one vertex on subdivision, we call them *parents* of the *child* vertex. The parents of the parents are then, of course, grandparents. The parent relation is illustrated in Fig. 4.1.

Besides some of the definitions given in Section 2.1 we precisely define the language used that is specific to the subdivided graph. We denote the set of all finite lists (i.e. tuples) of elements from a set $\mathcal{A}$ using $[\mathcal{A}]$, that is,

$$[\mathcal{A}] = \{[a_1, a_2, \ldots, a_n] : n \in \mathbb{N} \text{ and } a_1, \ldots, a_n \in \mathcal{A}\} \,. \tag{4.1}$$

We use the following notation for the family of all finite sets of elements from $\mathcal{A}$:

$$\mathcal{A}^* = \{\{\, a_1, a_2, \ldots, a_n \,\} : n \in \mathbb{N} \text{ and } a_1, \ldots, a_n \in \mathcal{A}\} \,. \tag{4.2}$$

We define the layer function which maps from vertices to their layer number

$$\begin{aligned} l &: V \to \mathbb{N} \,, \\ l(\alpha) &= k : \alpha \in L_k \,. \end{aligned} \tag{4.3}$$

And the direct parent function which gives the vertices that are on the edge that generated α

$$\begin{aligned} p &: V \to V^* \,, \\ p(\alpha) &= \{\beta : \beta \in (N(\alpha) \cap V_{l(\alpha)-1})\} \,. \end{aligned} \tag{4.4}$$

For simplicity we also define the parents of a set of vertices such that we can use them interchangeably

$$\begin{aligned} p &: V^* \to V^* \,, \\ p(\mathcal{A}) &= \{\pi : \alpha \in \mathcal{A} \,, \pi \in p(\alpha)\} \,. \end{aligned} \tag{4.5}$$

A path between α and β is usually represented by $P_{\alpha,\beta}$, which is a list of vertices including the endpoints. Another useful function is the list concatenation operator $+\!\!+$ which is defined as

$$[a_1, \ldots, a_n] +\!\!+ [b_1, \ldots, b_m] \triangleq [a_1, \ldots, a_n, b_1, \ldots, b_m] \,. \tag{4.6}$$

List concatenation is used mostly to construct paths.

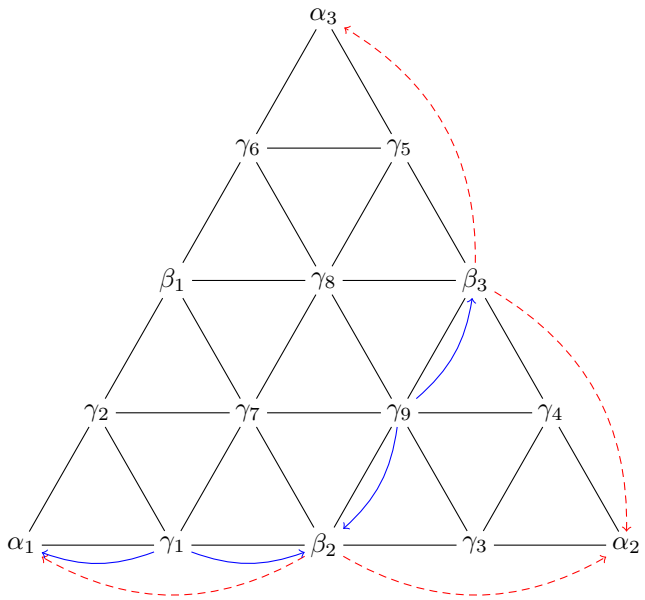


Figure 4.1: The parents of γ_1 and γ_9 visualised as a path on the graph. For clarity edges not in E_2 have been left out. The edge between α_1 and β_2 created γ_1 so they are the direct parents (blue, solid arrow). In turn β_2 was created by the edge (α_1, α_2) which are the second order parents of γ_1 (red, dashed arrow).

Example

To illustrate what the parents are, we give an example using Fig. 4.1. The direct parents of γ_1 are connected through one hop to γ_1

$$p(\gamma_1) = \{\beta_2, \alpha_1\},$$

and of γ_9 they are

$$p(\gamma_9) = \{\beta_2, \beta_3\}.$$

The parents on layer zero ($p_0(\alpha)$) are all nodes connected by a path of length 2, stopping at the first node on a layer less than 1, i.e. any α_i. Thus

$$p(p(\gamma_1)) = \{\alpha_1, \alpha_2\},$$
$$p(p(\gamma_9)) = \{\alpha_1, \alpha_2, \alpha_3\},$$

where we first used (4.4) and then (4.5).

Furthermore, it turns out that vertices that are connected must share one parent, which we call the *common parent*. We use the notation $\pi_{\alpha,\beta}$ to signify the unique common parent of any $\alpha, \beta \in V$ that are connected. Here π is intended as a mnemonic for **p**arent.

Proposition 4.1.1 (Common Parent)**.** *If any two vertices of $\alpha_1, \alpha_2 \in V$ are connected by an edge, they have some unique common parent π_{α_1,α_2}.*

The sphere graph also has the property that, when routing between two vertices, it is never shorter to route through edges on a higher layer than either vertex. This means that when routing between two vertices on some layer larger than k, then we may ignore any edges E_h and vertices L_h with $h > k$. This greatly reduces the number of options when routing between vertices, if the layer of the destination is known.

Theorem 4.1.2 (No Higher Edge)**.** *Let $\alpha_0 \in L_i$ and $\alpha_m \in L_j$ have a distance of m. Then all paths between α_0 and α_m of length m do not use an edge in E_h, where $h > \max(i, j)$.*

Now that we know higher layers are uninteresting for routing, we can start analysing what happens if two vertices are far apart. We show that when two vertices on the same layer $\alpha, \beta \in L_k$ and are at least 3 distance apart $d(\alpha, \beta) \geq 3$, then there is a shortest path between them that uses only lower layer vertices. So when we can guarantee that two vertices are some distance apart, we may assume that there are some parents of α and β, π_α and π_β, that have $d(\pi_\alpha, \pi_\beta) = d(\alpha, \beta) - 2$. Thus there are some parents that are on a shortest path between α and β.

Theorem 4.1.3 (Three Hops). *Consider a shortest path $P_{\alpha_1,\alpha_{m+1}}$ of length $m \geq 3$ between two vertices on the same layer $\alpha_1, \alpha_{m+1} \in L_k, k > 0$. Then there exists a path also of length m that contains only vertices in $V_h, h < k$ between α_1 and α_{m+1}, except for α_1 and α_{m+1}.*

This hints to the presence of a routing algorithm which routes through the parents when it knows that α and β are some distance apart. However, vertices always have two parents, so it remains to see which parent should be chosen. This will be addressed in the next section, on the labelling of nodes.

Example

To illustrate how the common parent works, we give a small example using Fig. 4.1 once more. From the figure it is possible to see that the common parent $\pi_{\gamma_1,\gamma_9} = \beta_2$, because both share that same direct parent. Some other examples showing the existence of common parents

$$\pi_{\gamma_1,\beta_2} = \beta_2\,,$$
$$\pi_{\beta_2,\beta_3} = \alpha_2\,.$$

However, α_1 and α_2 do not share a common parent, even though they are connected, since they are on the base layer. And γ_1 and β_3 also do not share a common parent, because they are not connected. While γ_1 and β_3 do share a common ancestor (α_2), the common parent is only applicable to direct parents.

4.2. Labelling

Every vertex gets a label which should uniquely identify where the vertex is located. As seen in the previous section, we also want to know which layer the vertex is on, and how far away we are from it. The label must also be small in size, as it functions as a data header included in every transmission over the network, indicating where the data must go. If the labelling is able to do so, we will be able to construct an efficient routing algorithm.

Every vertex is labelled similarly to a hierarchical routing scheme such as Internet Protocol (IP) which is widely used for routing in the current computer networks. Every node is assigned a unique ID, which is simply a unique integer. The label contains this unique ID, and the node's ancestry tree which includes the parents, then the grandparents, etc. It is constructed as a list of sets, i.e. $[\{a\}, \{\ldots\}, \{\ldots\}, \ldots]$, where $a \in [|V_k|]$ is the unique ID. In addition, let $[\ldots]_k$ denote the k-th element of a list. First we define an unfiltered label for a vertex α with parents $\{\beta_1, \beta_2\}$, which simply copies the labels of its parents into its own label, but places itself at index 1

$$\text{label}^{\text{u}} : V \to [V^*]\,,$$
$$\text{label}^{\text{u}}(\alpha) = \begin{cases} [\{\alpha\}] & \text{if } v \in V_0\,, \\ [\{\alpha\}] + [\text{label}^{\text{u}}(\beta_1)_1 \cup \text{label}^{\text{u}}(\beta_2)_1, & \\ \qquad \ldots, \text{label}^{\text{u}}(\beta_1)_\ell \cup \text{label}^{\text{u}}(\beta_2)_\ell] & \text{otherwise.} \end{cases} \tag{4.7}$$

where ℓ is the minimum size of either parent label, since we want to stay within their bounds

$$\ell = \min\left(|\text{label}^{\text{u}}(\beta_1)|\,, |\text{label}^{\text{u}}(\beta_2)|\right)\,. \tag{4.8}$$

The unfiltered label stops at the first occurrence of an element in the base layer ℓ, because the label^{u} is undefined for any higher indices. Intuitively, $\text{label}^{\text{u}}(\alpha)_2$ are the first-order parents of α, the parents to which v has a direct connection. Followed by $\text{label}^{\text{u}}(\alpha)_3$, which are the second-order parents of α, i.e. the grandparents. All the way to $\text{label}^{\text{u}}(\alpha)_k$ which must contain vertices in the base icosahedron V_0.

Currently, the unfiltered label may have duplicate elements. Duplicate elements take up more space, so they are removed. We construct the filtered label as follows

$$\text{label}^{\text{f}} : V \to [V^*]\,,$$
$$\text{label}^{\text{f}}(\alpha)_k = \begin{cases} \text{label}^{\text{u}}(\alpha)_k & \text{if } k \leq 2\,, \\ \text{label}^{\text{u}}(\alpha)_k \setminus \bigcup_{i=1}^{k-1} \text{label}^{\text{u}}(\alpha)_i & \text{otherwise.} \end{cases} \tag{4.9}$$

Then the filtered label can be defined as

$$\text{label}^{\text{f}}(\alpha) = [\text{label}^{\text{f}}(\alpha)_1, \text{label}^{\text{f}}(\alpha)_2, \ldots, \text{label}^{\text{f}}(\alpha)_\ell]\,. \tag{4.10}$$

We also define the term *adding to the label*, because it simplifies talking about the labeling.

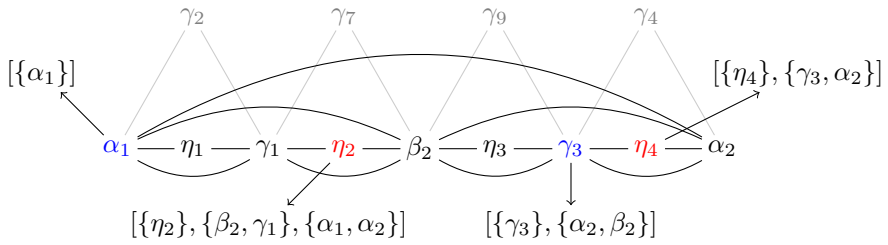


Figure 4.2: An example for the filtered labelling scheme in Eq. (4.10), with only vertices on the edge (α_1, α_2) shown. The filtered labels of the nodes $\alpha_1, \eta_2, \gamma_3$ and η_4 are shown.

Definition 4.2.1 (Adding to a Label)**.** *Consider some vertex γ that has a label* $\text{label}(\gamma)$*. Then we define that vertex α* adds *vertex $\beta \in p(\alpha)$ to the label, iff $\alpha \in \text{label}(\gamma)_k$ and $\beta \in \text{label}(\gamma)_{k+1}$.*

Example

In Fig. 4.2 we give an example on how the label is constructed. For α_1, the ID is α_1, and it is already on layer 0, thus the label is $[\{\alpha_1\}]$. A node on the second layer is γ_3, which has as parents α_2 and β_2. Because $\alpha_2 \in V_0$, the label stops here and results in $\text{label}(\gamma_3) = [\{\gamma_3\},\{\alpha_2,\beta_2\}]$. Located on the third layer is η_4, with parents $\{\gamma_3,\alpha_2\}$. Because $\alpha_2 \in V_0$, the label ends here, becoming $[\{\eta_4\},\{\gamma_3,\alpha_2\}]$. Lastly, η_2 is also located on the second layer, with parents $\{\gamma_1,\beta_2\}$. In turn β_2 has parents $\{\alpha_1,\alpha_2\}$ which are in V_0, thus the label stops here. The parents of γ_1 which are $\{\beta_2,\alpha_1\}$ are not added to the label again, because they are duplicates. First, α_1 is not added again, because β_2 already added it to the set $\text{label}(\eta_2)_2$. And secondly, β_2 is not added because it has already occurred previously in $\text{label}(\eta_2)_1$.

4.2.1. Simplifying the Label Further

We can simplify the labeling further, by removing what we define as *simple* vertices. A simple vertex is a vertex of which one of the parents has already been added to the label. Intuitively that means when routing from a high layer to a low layer you want to skip simple vertices, because their parent has already occurred in the label so we could have routed through that instead. A simple vertex occurs at the same or later index in the label than their parent, so there must be an equally short or shorter path to that parent.

Definition 4.2.2 (Simple Vertex)**.** *Consider a vertex $\alpha \in \text{label}^{\text{f}}(\alpha)_\ell$, for some integer ℓ. If there exists a vertex*

$$\beta \in \bigcup_{i=1}^{\ell} \text{label}^{\text{f}}(\alpha)_i$$

where $\beta \in p(\alpha)$, then we call α simple*.*

The most interesting property of simple vertices is that any simple vertex will only add at most one vertex to the filtered label, that must also be simple. This property of simple vertices is formalised in the following proposition.

Proposition 4.2.1 (Simple Vertex Properties)**.** *A simple vertex in $\text{label}^{\text{f}}(\alpha)_\ell$ will add no more than one vertex to $\text{label}^{\text{f}}(\alpha)_{\ell+1}$. Moreover, if a parent was added to the label by a simple vertex, it is also simple.*

That means that once a vertex is simple, it and its parents will never become non-simple, which in turn implies that we can remove them and still retain all possible non-simple vertices. Only if they would add non-simple parents, then it may be the case that routing through them would be interesting. So we can remove any simple vertex from the label.

$$\text{label}^{\text{s}}(\alpha)_k = \text{label}^{\text{f}}(\alpha)_k \setminus \{\beta \in \text{label}^{\text{f}}(\alpha)_k : \beta \text{ is simple}\}\,. \tag{4.11}$$

Example

An example for simple vertices is given in Fig. 4.3. The ancestry tree for ε is drawn, which is a fictitious child of η_2 and β_2 in Fig. 4.2. We can see that the simple vertices may occur at the same index as their parent, and that simple vertices only add simple vertices to the label when duplicates are filtered out. We can indeed see that duplicate entries for β_2 and α_1 are removed in accordance to Eq. (4.10). More importantly, when routing

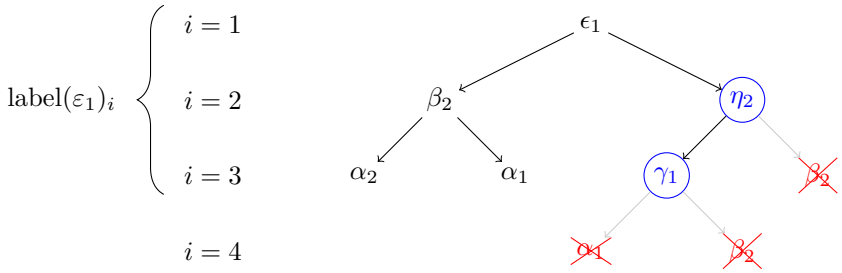


Figure 4.3: The ancestry tree is given of a node ε_1 which is the child node of η_2 and β_2 in Fig. 4.2. Only as an example the filtered label is continued for $i = 4$, where it would normally stop at $i = 3$. η_2 is simple (blue, circled), because its parent β_2 is in label$(\eta_2)_i$, for $i \leq 2$. It adds only one vertex to the filtered label, γ_2, which is also simple. The nodes β_2 and α_1 (red, crossed out) already occur in the tree previously, and are thus filtered out. A simple vertex always adds at most one vertex to the tree, which is also simple.

to α_1 from ε_1 we can see that it is faster to route through β_2 which is 2 hops, than it is through η_2 which takes 3 hops. That is because β_2 is a parent of η_2, so if going to ancestors that are far away, it is better to skip simple vertices.

Final Label

Sometimes there are non-simple vertices in $\text{label}^s(\alpha)$ which have no parents in the next label entry. Except in the last label entry, these vertices are also undesired. If routing to a far away ancestor, then we would like to traverse the ancestry tree as fast as possible, by continuously following the non-simple parents of vertices. However, if a vertex has no non-simple parents, then it is a dead end in this traversal. Thus, only vertices that do have non-simple parents should be chosen, and there is always at least one such choice, so we remove non-simple vertices that do not have parents in the label. As such we arrive at the final labelling

$$\begin{aligned} &\text{label} : V \to [V^*]\,, \\ &\text{label}(\alpha)_k = \begin{cases} \text{label}^s(\alpha)_k & \text{if } k = |\text{label}^s(\alpha)|, \\ \text{label}^s(\alpha)_k \setminus \{\beta \in \text{label}^s(\alpha)_k : p(\beta) \cap \text{label}(\alpha)_{k+1} = \emptyset\} & \text{otherwise,} \end{cases} \qquad (4.12) \\ &\text{label}(\alpha) = [\text{label}(\alpha)_1, \text{label}(\alpha)_2, \ldots, \text{label}(\alpha)_\ell]\,. \qquad (4.13) \end{aligned}$$

Pseudocode for the construction of the label is given in Algorithm 4.1. We may remove simple vertices while constructing the label, as they never add non-simple parents. This has been applied in Line 8 in Algorithm 4.1, the labelling algorithm. Moreover, we remove vertices that do not add any non-simple parents to the label by going backwards over the label and checking if any parents are in the label.

The remaining vertices in the label also have some interesting properties. Very important is the fact that $|\text{label}(\alpha)_i| \leq 3$, for any i. This greatly limits the size of the label, as desired. Furthermore, every two vertices for an entry in the label are adjacent.

Lemma 4.2.2 (Non-Simple Vertex Bound)**.** *The label size* $|\text{label}(\alpha)_\ell| \in \{1, 2, 3\}$*, for any* $\alpha \in V$ *and integer* ℓ*. Additionally, for any* $\alpha, \beta \in \text{label}(\alpha)_\ell : \alpha \neq \beta$ *it holds that* $\{\alpha, \beta\} \in E$*.*

4.3. Routing Algorithm

In this section we give a high-level routing algorithm for finding the shortest path on the sphere. We also need to keep in mind that the algorithm should eventually be transformed in a local form, where each node can route using only information about nearby nodes and the labelling, as we will see in Section 4.4. The general intuition of the routing algorithm is that when two vertices are not close to each other, then it is possible to route towards a lower layer as indicated in the Three Hops theorem (Theorem 4.1.3). The nodes in the lower layer will then be two steps closer to each other. We will show that the labelling takes care of the selection of nodes to a lower layer, so that it does not matter which vertex we choose from the label. All vertices in the label at a certain index are equivalent, as long as the destination is still far away.

But first we need to define what far away means. We found a sufficient condition while proving the equivalence of vertices in the label. Given the sender vertex $\alpha \in V$ and the receiver $\beta \in V$, and any $\gamma \in p(\alpha)$ and $\eta \in p(\beta)$, then it must hold that $d(\alpha, \beta) > 3$ and $d(\gamma, \eta) > 3$ for α and β to be far away. We give an algorithm that searches the neighbourhood of nodes using a brute force approach, guarantees that $d(\alpha, \beta) > 3$ and $d(\gamma, \eta) > 3$, and always returns a path of at most length 6. Thus we named this algorithm path6. We will later see that the neighbourhood is limited in such a way that the complexity of this algorithm is still acceptable. It results in the following lemma for path6:

Algorithm 4.1: Labelling of vertices. When the graph is constructed, each vertex is given a label that will allow efficient routing. The labelling is similar to IP addressing, because a hierarchical structure is used.

Data : $G = (V, E)$, the complete graph.
Input : $\alpha \in V$
Output : $\text{label}(\alpha)$, the label of α

Function $\text{label}(\alpha)$ **is**
  $\text{label}^{\text{s}}(\alpha)_1 = \{\alpha\}$ `// First we construct` $\text{label}^{\text{s}}(\alpha)$ `according to Eq. (4.11)`
  $\mathcal{D} \leftarrow \emptyset$ `// The set of vertices that have already occurred in the label`
  $i \leftarrow 2$
  while $p(\text{label}(\alpha)^{f}_{i-1})$ exists **do**
    $\text{label}^{\text{new}} \leftarrow p(\text{label}^{\text{s}}(\alpha)_{i-1}) \setminus \mathcal{D}$ `// Remove vertices that have already occurred (4.10)`
    $\mathcal{D} \leftarrow \mathcal{D} \cup \text{label}(\alpha)^{\text{new}}$
    $\text{label}^{\text{s}}(\alpha)_i \leftarrow \text{label}^{\text{new}} \setminus \{\beta : \beta \text{ is simple}\}$ `// Remove simple vertices (4.11)`
    $i \leftarrow i + 1$
  end
  $k \leftarrow |\text{label}(\alpha)|$ `// The length of the label`
  `// Remove vertices with no non-simple parents (4.12), to construct` $\text{label}(\alpha)$ `(4.13)`
  $\text{label}(\alpha)_k \leftarrow \text{label}^{\text{s}}(\alpha)_k$
  for $i \leftarrow k - 1$ **to** 1 **do**
    $\text{label}(\alpha)_i \leftarrow \text{label}^{\text{s}}(\alpha)_i \setminus \{\beta \in \text{label}^{\text{s}}(\alpha)_i : p(\beta) \cap \text{label}(\alpha)_{i+1} = \emptyset\}$
  end
  return $[\text{label}(\alpha)_1, \text{label}(\alpha)_2, \ldots, \text{label}(\alpha)_k]$
end

Lemma 4.3.1 (path6 Distance)**.** *Consider two vertices $\alpha, \beta \in V$ for which path6(α, β) (Algorithm 4.2) does not find a path. Then $d(\alpha, \beta) > 3$. Additionally, for any $\pi_\alpha \in p(\alpha)$ and $\pi_\beta \in p(\beta)$, $d(\pi_\alpha, \pi_\beta) > 3$.*

To guarantee termination of the routing algorithm, path6 must always find a path if given two vertices on the base layer $\alpha, \beta \in V_0$. However, the diameter of the icosahedron is 3, which means there are some pairs of α and β where $N(\alpha) \cap N(\beta)$ does not find a path, and $p(\alpha)$ nor $p(\beta)$ exist for these nodes. That is why there is a part in path6 handling exactly this situation. We use a standard algorithm for routing on the icosahedron called Dijkstra's algorithm, which runs in $O(|E| \log |V|)$ [KT06]. Moreover, we can restrict it to only G_0, according to Theorem 4.1.2. We know that the number of elements in $|E_0|$ is $O(1)$ and for $|V_0|$ it is also $O(1)$, so executing this will not have a significant impact on space or time complexity.

Algorithm 4.2: The algorithm path6 searches the neighbourhood of α and β for a path of maximum length 6. For simplicity, we assume $p(\gamma) = \emptyset$ given that $\gamma \in V_0$.

Input : $\alpha, \beta \in V$
Output : $P_{\alpha,\beta}$ the path from α to β, of maximum length 6. Otherwise nothing.
Function $\text{path6}(\alpha, \beta)$ **is**
  `// Calculate the neighbourhoods of` α `and` β`, and also of the parents and grandparents`
  $N^\alpha \leftarrow \{\alpha\} \cup N(\alpha) \cup N(p(\alpha)) \cup N(p(p(\alpha))$ `// Note: Includes non-simple parents`
  $N^\beta \leftarrow \{\beta\} \cup N(\beta) \cup N(p(\beta)) \cup N(p(p(\beta))$
  if $\exists \gamma : \gamma \in N^\alpha \cap N^\beta$ **then**
    $\gamma^{\min} \leftarrow \min_\gamma \{d(\alpha, \gamma) + d(\beta, \gamma)\}$ `// Take the closest` γ
    return $P_{\alpha,\gamma^{\min}} + P_{\gamma^{\min},\beta}$ `// And return the path to and from it`
  else if $d(\alpha, \beta) = 3 : \alpha, \beta \in L_0$ **then** `// Handle an edge case on the icosahedron`
    return dijkstra(α, β) `// Use Dijkstra's algorithm [KT06] on` G_0 `to find a path`
  end
end

We are then able to show that it does not matter which parent is taken from the label, if path6 does not find a path. With the guarantees on the distance, we are able to show that there always exists a shortest path going through the parents. It remains to show that any vertex in the label is indeed on a shortest path, and that not some other vertex is strictly faster. We prove exactly this, in the Label Routing Theorem.

Theorem 4.3.2 (Label Routing). *Consider two nodes $\alpha, \beta \in V, l(\alpha) \geq l(\beta)$ so that path6(α, β) (Algorithm 4.2) cannot determine a path between them. Then for every node $\pi_1 \in \text{label}(\alpha)_2$ holds that $d(\pi_1, \beta) = d(\alpha, \beta) - 1$, i.e. they are on a shortest path between α and β.*

Using this theorem, we can create a routing algorithm that constructs a shortest path. If path6 is not able to find a path, we use Label Routing to show that we can route to a parent in the label. Additionally, we pick a parent at random, to more uniformly spread the routes over the nodes. Since connections are single-use it is desirable to have many different paths, as a high load will result in more delay. The pseudocode of the routing algorithm is given in Algorithm 4.3. Note that Theorem 4.3.2 assumes that α is on at least as high a layer as β. Nonetheless if α is on a lower layer, we can swap α and β so that the theorem still applies, so that we just have to take $\pi_1 \in \text{label}(\beta)_2$. Which is exactly what the algorithm does, thus allowing us to prove the optimality.

Theorem 4.3.3 (Sphere Routing Optimality). *Consider vertices $\alpha, \beta \in V$ with a distance m, the sphere routing algorithm (Algorithm 4.3) will find a path $P_{\alpha,\beta}$ also of length m.*

Algorithm 4.3: High-Level Routing Algorithm. The paths to α and β are accumulated in P_α and P_β. Once α and β are close enough, path6 will connect them.

Input : $\alpha, \beta \in V$, sender and receveiver respectively.
Output: $P_{\alpha,\beta}$ the path from α to β
Function path(α, β) **is**
  return pathRecursive$(\alpha, \beta, [\,], [\,])$
end
Input : $\alpha, \beta \in V$, sender and receiver respectively.
  $P_\beta, P_\beta \in [V]$ the path from the starting vertices to α and β
Output: $P_{\alpha,\beta}$ the path from α to β
Function pathRecursive$(\alpha, \beta, P_\alpha, P_\beta)$ **is**
  if $\exists$path6(α, β) **then**
    return $P_\alpha +\!\!+$ path6(α, β) $+\!\!+$reverse(P_β)
  else
    `// Increment the path to a parent in the label of` α `or` β`.`
    if $l(\beta) > l(\alpha)$ **then**
      $\beta^{\text{new}} \leftarrow$ randomElement(label$(\beta)_2$) `// Choose randomly on multiple options`
      return path$(\alpha, \beta^{\text{new}}, P_\alpha, P_\beta +\!\!+ [\beta])$
    else `// Else` $l(\alpha) \geq l(\beta)$
      $\alpha^{\text{new}} \leftarrow$ randomElement(label$(\alpha)_2$) `// Choose randomly on multiple options`
      return path$(\alpha^{\text{new}}, \beta, P_\alpha +\!\!+ [\alpha], P_\beta)$
    end
  end
end

4.4. Local Routing Algorithm

We will adapt the algorithm given in Algorithm 4.3 to an algorithm which does not require global information to route. This is important for any routing algorithm that is used in real world networks. If every network node has to able to calculate a shortest route, then each vertex must store the entire network structure. That would mean the locally stored data grows linearly with the network size in every vertex. However, we want an algorithm that will perform well even when the number of nodes grows large. Thus we cannot store the entire network structure, but must restrict the data to what every vertex knows about itself and its limited surroundings. In addition, we argue why the local algorithms are equivalent to their global counterparts. Then the proof of optimality for the global algorithm will also guarantee the optimality of the local algorithms.

In a local algorithm we assume the perspective of a single network node, that must be able to route any incoming data packets towards the next hop that on a shortest path. Once some vertex α wants to route to β and sends his data packets to α', it is as if α' wants to route to β and performs the same operations as α. This continues until β has been reached. That means that the α in the algorithm will refer to the node the data packet is currently at. The only exception will be the result of path6, which precalculates a path of at most 6 hops that can be blindly followed by any vertex on this path.

4.4.1. Local path6 Algorithm

First, the path6 algorithm in Algorithm 4.2 has to be adapted for a neighbourhood search from the perspective of the sender $\alpha \in V$ as seen in the global path6 algorithm (Algorithm 4.2). The first issue we encounter, is that the current position of the receiver $\beta \in V$ is unkown, as the receiver β also moves in the global algorithm. Let β' be the current value of the receiver β in the global algorithm. The movement of β' is restricted to label(β), so we can assume that $\beta' \in \text{label}(\beta)$. All possible possitions of β' can thus be contained in in the set

$$L_\beta = \bigcup_i \text{label}(\beta)_i \,. \tag{4.14}$$

Furthermore, the global path6 algorithm calculates $N(\beta)$. But it is not possible for α to calculate $N(\beta)$, because that would require global information about the network structure, as β is not known before communication occurs. Let N^α be the entire neighbourhood that is searched by the global path6 algorithm

$$N^\alpha = \{\alpha\} \cup N(\alpha) \cup N(p(\alpha)) \cup N(p(p(\alpha)) \,. \tag{4.15}$$

Then we can see that if there is some node $\gamma \in N^\alpha \cap N(\beta)$, then $\beta \in N(\gamma)$ and $\gamma \in N^\alpha$. Thus we can find β using $N(N^\alpha)$.

Finally, the global algorithm also uses $p(\beta)$. But α only knows the parents of β that are in label(β) and not the simple parents, which have to be included if the algorithm is to be correct. Again, we can see that if there is a node $\gamma \in N^\alpha \cap p(\beta)$, then $\beta \in \text{children}(\gamma)$ and $\gamma \in N^\alpha$. Thus $\beta \in \text{children}(N^\alpha)$. The same applies to $p(p(\beta))$, but we require children(children(N^α)).

Using these three changes, we can create a local algorithm of path6 that is given in Algorithm 4.4. If for any $\beta' \in L_\beta$ there is a path6, that means that there is path of maximum length 6 to the ancestry tree of β (as introduced in Section 4.2). A nice property of the distance $d(\gamma, \beta)$ (as defined Eq. (2.3)) is that it is equal to the index γ occurs in label(β)

$$d(\gamma, \beta) = i \iff \gamma \in \text{label}(\beta)_i \,, \tag{4.16}$$

which can be calculated with a linear search over label(β), and the label is small as seen in Lemma 4.2.2, so this can be done efficiently. Furthermore, $d(\gamma, \alpha)$ can also be calculated quickly as shown in the time and space complexity analyses (Section 4.4.3).

Example

We give some examples to gain insight into the transformation given. Some of the cases that we encounter in the global path6 algorithm include $N(\alpha) \cap N(p(\beta'))$, or $N(\alpha) \cap p(p(\beta'))$, or even $N(p(p(\alpha)) \cap N(p(p(\beta'))$. The first case can be translated as

$$N(\alpha) \cap N(p(\beta')) \implies N(N(\alpha)) \cap p(\beta') \implies \beta' \in \text{children}(N(N(\alpha))) \,. \tag{4.17}$$

The second case

$$N(\alpha) \cap p(p(\beta')) \implies \beta' \in \text{children}(\text{children}(N(\alpha))) \,, \tag{4.18}$$

while the last case is equivalent to

$$N(p(p(\alpha)))) \cap N(p(p(\beta'))) \implies \beta' \in \text{children}(\text{children}(N(N(p(p(\alpha))))))) \,. \tag{4.19}$$

4.4.2. Local Routing Algorithm

Now that we have transformed the global path6 algorithm into a local version, we can look at the remainder of the routing algorithm (Algorithm 4.3). If path6 is not applied, either α or β should perform a step through their label depending on their relative layer. However, we are now routing from α to β and do not know at what layer β currently is. Suppose that β should perform a step in the global algorithm, and not α. If it did, then it still would not change L_β, since we assumed β can be anywhere in its label. So we can assume that β is at a lower or equal layer to α, which then allows us to perform a step towards some $\alpha' \in \text{label}(\alpha)$. This will repeat until path6 has found a path. And as shown in Section 4.3 a path6 will eventually be found.

But what happens once we have found a vertex $\beta' \in L_\beta$ that is on the ancestry tree of β? Routing should go up the layers now, instead of down. More specifically, up the layers on a shortest path towards β. First the node must know that it is now on the ancestor tree of β, which is indicated by the flag d. Then routing can be performed towards β. We know that $\beta' \in \text{label}(\beta)_i$ for some i. We also know that it does not matter which $\gamma \in \text{label}(\beta)_{i-1}$ is chosen, because the global algorithm also chooses γ at random when routing from label(β)$_{i-2}$. Thus any $\beta^{\text{new}} \in \text{label}(\beta)_{i-1} \cap N(\beta')$ will be reachable by β', and is still on a shortest path towards β. Such a β^{new} exists, because there must be a child of β' that added β' to label(β) and is non-simple (Proposition 4.2.1). Since $\beta' \in \text{label}(\beta)_i$ it also cannot be the case that this child was removed in any of the label filtering steps (Eqs. (4.11) and (4.12)).

Algorithm 4.4: A local variant of path6 (Algorithm 4.2). To perform path6 α can either store its path6-neighbourhood (as shown), or request it through the classical network.

Data : $\alpha \in V$, the current node.
$N^\alpha \leftarrow \{\alpha\} \cup N(\alpha) \cup N(p(\alpha)) \cup N(p(p(\alpha))$, the neighbourhood of α.
$N_2^\alpha \leftarrow N^\alpha \cup N(N^\alpha)$. Include $N(\alpha) \cap N(\beta)$ seen from α's side.
$N_3^\alpha \leftarrow N_2^\alpha \cup \text{children}(N_2^\alpha) \cup \text{children}(\text{children}(N_2^\alpha))$. Finally, include $p(\beta)$ and $p(p(\beta))$ in the search from the perspective of α.
Input : $\beta \in V$, the destination.
Output : $P_{\alpha,\gamma}$ the path from α to a vertex $\gamma \in V$ towards β of maximum length 6. Otherwise nothing.
Function path6(β) **is**
  $L_\beta \leftarrow \bigcup_i \text{label}(\beta)_i$ `// The entire ancestry tree of` β
  if $\exists \gamma : \gamma \in N_3^\alpha \cap L_\beta$ **then**
    $\gamma^{\min} \leftarrow \min_\gamma \{d(\alpha,\gamma) + d(\gamma,\beta)\}$ `// Take the closest` γ
    return $P_{\alpha,\gamma^{\min}}$
  else if $d(\alpha,\beta') = 3 : \alpha \in L_0, \beta' \in L_0 \cap L_\beta$ **then** `// Handle an edge case on the icosahedron`
    return $\min_{\beta' \in L_0 \cap L_\beta}\{\text{dijkstra}(\alpha,\beta)\}$ `// Use Dijkstra's algorithm on` G_0 `to find a path`
  end
end

Algorithm 4.5: Local Routing Algorithm. A vertex α receives a data packet (β, d) and sends it along according to the return value.

Data : $\alpha \in V$, the current node.
$N(\alpha) \in V^*$, the IDs of the direct neighbourhood of α.
Input : $\beta \in V$, the destination.
$d \in \{0,1\}$, d stands for 'down', signalling if a path6 has been reached.
Output : $\gamma \in [V]$, the next nodes to route to.
Function localPath(β, u) **is**
  if $\alpha = \beta$ **then**
    Destination reached
  else if $u = 0$ **then**
    if $\exists$path6(α,β) **then**
      $u \leftarrow 1$
      return path6(α,β)
    else
      `// Decrease the layer to a parent in the label of` α.
      $\alpha^{\text{new}} \leftarrow \text{randomElement}(\text{label}(\alpha)_2)$ `// Choose randomly on multiple options`
      return $[\alpha^{\text{new}}]$
    end
  else
    `// Increase the layer to a child that is in the label of` β.
    $i : \alpha \in \text{label}(\beta)_i$
    $\beta^{\text{new}} \leftarrow \text{randomElement}(\text{label}(\beta)_{i-1} \cap N(\alpha))$ `// Choose randomly on multiple options`
    return $[\beta^{\text{new}}]$
  end
end

4.4.3. Complexity Analysis

The local algorithms path6 and the routing algorithm must also be fast and use little memory. This can be checked by performing a time- and space-complexity analysis. Because the algorithm is local, we look at the complexities per vertex. The space complexity is related the amount of memory stored per vertex, and we show that it scales poly-logarithmically with the number of vertices $|V|$.

Theorem 4.4.1 (Local Space Complexity)**.** *The space complexity per vertex of the local routing algorithm is* $O(\log^5 |V|)$.

Furthermore, we also show that the time complexity, related to the running time, scales poly-logarithmically in the number of vertices $|V|$.

Theorem 4.4.2 (Local Time Complexity)**.** *The time complexity of the local routing algorithm is* $O(\log^2 |V|)$ *per vertex.*

Since the diameter of the graph $D(G) = 4\log_2\left(\frac{n-2}{10}\right) + 3 = O(\log |V|)$ (Proposition 3.1.1) and every vertex uses $O(\log^2 |V|)$ run-time, the complete routing of any packet will take at most $O(\log^3 |V|)$ time.

5

Routing Technical Details

In this chapter we will go into detail of the theorems used in Chapter 4, proving their correctness. But first, we give an outline of our proving strategy that may be used as a guide through our proofs (Section 5.1). Only then we follow with the proofs of the graph properties (Section 5.2), then the labelling (Section 5.3), the routing algorithm optimality (Section 5.4) and finally the complexity analysis of the algorithm (Section 5.5).

5.1. Proof Outline

The proofs are structured into four sections: Graph Properties, Labelling, Proof of Optimality and Complexity Analysis. Each section builds on the theorems of the previous. We give an outline of the theorems proved in each section and how they result in the next theorem.

Graph Properties The parents of a node have a few simple properties that are formalised in propositions, that of the common parent, the connectedness of parents and the layer of a vertex. We use these properties in proofs to show that there exist certain parents that can be routed through.

These are followed by two key theorems for routing, Theorem 5.2.4 (No Higher Edge) and Theorem 5.2.5 (Three Hops). No Higher Edge shows that any shortest path between vertices may ignore edges on higher layers than either vertex, greatly reducing the number of possibilities since we can ignore children. Three Hops furthermore shows that if two vertices are on the same layer and at least 3 hops apart, then there must be a shortest path that goes through the parents. This can then be extended to two vertices on different layer of any distance in Corollary 5.2.6. These Theorems already hint towards a routing algorithm that makes use of the parents of nodes to route, as there almost always seems to be some parent that is on a shortest path, except when they are already nearby.

Labelling We have seen that parents are interesting for finding the shortest path, but since each node has two parents we do not know which parent is on the shortest path. As it turns out, each vertex has at least one parent that must always be on the shortest path, if there is any parent on the shortest path. We show this by constructing a label and filtering it. Furthermore, it is necessary for another vertex to know who to route to. The labelling of vertices must also indicate the location of each vertex if a node does not have access to global information.

The first type of vertices that are not necessarily on a shortest path are duplicates in the label, vertices that have already occurred in the label. The second type are simple vertices (Definition 4.2.2) which can be removed from the label without trouble as shown in Proposition 5.3.1 (Simple Vertex Properties). Lastly, vertices that do not add parents to the label can also be removed, because they are a dead-end when routing to lower layers. Then in Lemma 5.3.2 (Non-Simple Vertex Bound) we show that there are no more than 3 remaining non-simple vertices at any label index, greatly limiting the size of the label. Furthermore, the case analysis in the proof gives insight into the possible configurations of each label entry. As it turns out, all vertices in a label entry must be connected. Because they are connected, they also must be on the same layer (Corollary 5.3.3).

Proof of Optimality Here we show that the global routing algorithm (Algorithm 4.3) produces a shortest path. To do so, the path6 algorithm (Algorithm 4.2) first makes sure that the sender and receiver are at least a minimum distance apart as shown in Lemma 5.4.1 (path6 Distance). Then we can use the Three Hops Theorem to guarantee that some parent is in the shortest path.

The algorithm blindly picks some vertex in the label, if path6 does not find a path. It remains to show that the remaining vertices in the label after filtering are always on a shortest path, if a minimum distance is guaranteed. In Theorem 5.4.2 (Label Routing) we show that picking some vertex from the label is never worse than a simple vertex (which were removed from the label), nor is it worse than picking any other non-simple vertex where the necessity of the third label filtering is shown (4.12). We have argued in Section 4.3 that the path6 algorithm is optimal, and we have shown that routing to the parents in the label is optimal if path6 does not find a path. Thus in all cases the algorithm finds an optimal path, as shown in Theorem 5.4.3.

Complexity Analysis We have argued that the local algorithm is equivalent to the global algorithm in Section 4.4. But it is also important that the local routing algorithm is efficient and uses little memory. Because the algorithm is local, we look at the complexities per vertex. The space complexity is related to the amount of memory stored per vertex, and we show that it scales poly-logarithmically with the number of vertices $|V|$ (Theorem 5.5.1). Furthermore, we also show that the time complexity, related to the running time, scales poly-logarithmically in the number of vertices $|V|$ (Theorem 5.5.2).

5.2. Graph Properties

In this section we prove properties of the graph that are very useful for analysis. First we look at vertices, their parents and layering. Later, we look at how shortest paths are structured in the graph. Which nodes these paths use, and how the distance and layer of nodes dictate what nodes the path uses.

5.2.1. Vertex Properties

In a path, a sequence of vertices that are adjacent follow each other. We show that adjacent vertices have a *common parent*, which is of interest when we want to reroute path through the parents. A common parent of vertices $\alpha, \beta \in V$ is

$$\pi_{\alpha,\beta} \in \Pi_{\alpha,\beta} = \begin{cases} \{\alpha\} & \text{if } \alpha = \beta\,, \\ (p(\alpha) \cup \{\alpha\}) \cap (p(\beta) \cup \{\beta\}) & \text{otherwise.} \end{cases} \tag{5.1}$$

Which is defined in this way so that the common parent of a node and itself, is itself. And the common parent of two different nodes is either a common direct parent in $p(\alpha) \cap p(\beta)$. Or if $\alpha \in p_k(\beta)$ then $\alpha = \pi_{\alpha,\beta}$ and vice versa. We will now pose some facts of the graph, that that follow from the generation algorithm and will be used often in later proofs. First, is that two adjacent vertices always share a parent, their common parent. Furthermore, we show that the two parents of a node are always connected to each other. And finally, that the layer of a vertex depends on the maximum layer of its parents.

Proposition 5.2.1 (Common Parent)**.** *If any two vertices of $\alpha_1, \alpha_2 \in V$ are connected by an edge, they have some unique common parent π_{α_1,α_2}.*

Proof. If $\alpha_1 = \alpha_2$ then $\pi_{\alpha_1,\alpha_2} = \alpha_1$. Otherwise, we consider two cases, depending on the layers of α_1 and α_2.

1. If $l(\alpha_1) = l(\alpha_2)$ then on Line 18 of the subdivision algorithm (Algorithm 3.1) the vertex α_1 is connected to its neighbours $\Gamma = (N(\alpha) \cap L_k)$ of layer k. In this step two triangles are defined to connect α_1 to its neighbours, where each triangle only contains edges with at least one endpoint being in $p(\alpha_1)$. The edges of these triangles generate the set Γ and α_1. Then by construction it holds that all neighbours Γ have a parent in common
$$p(\alpha_1) \cap p(\gamma) \neq \emptyset \text{ for all } \gamma \in \Gamma.$$
Since $\alpha_1 \neq \alpha_2$, it must furthermore be the case that $p(\alpha_1) \neq p(\alpha_2)$, because the edge $p(\alpha_1) \in E$ only generates one vertex. Thus there is a unique common parent
$$\pi_{\alpha_1,\alpha_2} \in p(\alpha_1) \cap p(\alpha_2)\,.$$
2. When $l(\alpha_1) \neq l(\alpha_2)$, then assume without loss of generality (w.l.o.g.) that $l(\alpha_1) > l(\alpha_2)$. Because the vertices are connected and $l(\alpha_1) > l(\alpha_2)$ it must be the case according to Eq. (4.4) that $\alpha_2 \in p(\alpha_1)$. Thus $\alpha_2 = \pi_{\alpha_1,\alpha_2}$. □

Proposition 5.2.2 (Parents Connected)**.** *Consider some vertex $\alpha \in V_k, k > 0$, then for its parents $\{\beta_1, \beta_2\} = p(\alpha)$ holds that there is an edge $\{\beta_1, \beta_2\} \in E$.*

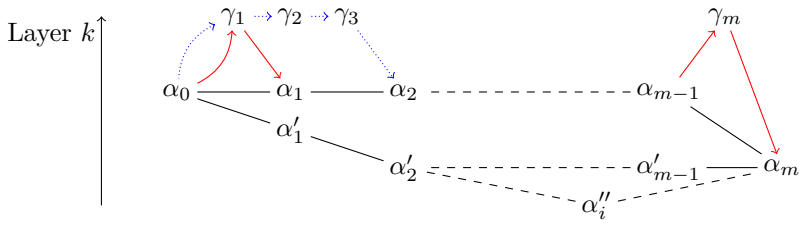


Figure 5.1: An illustration of the proof of Theorem 5.2.4. In black are given three possible shortest paths. It is not possible to construct a shortest path by replacing a hop with going through a higher layer (red, solid arrow). Additionally, any number of hops through a higher layer, may be replaced by going through a lower layer instead (blue, dotted arrow).

Proof. By construction of the graph, specifically on Line 13 of the graph generation algorithm (Algorithm 3.1), α is connected to its parents. These parents are chosen such that they are connected on Line 5 of the graph generation algorithm, because α is placed as a midpoint on the edge. Thus proving that there is an edge $\{\beta_1, \beta_2\} \in E$. □

Proposition 5.2.3 (Vertex Layer)**.** *Consider a vertex $\alpha \in L_k : k > 0$, then*

$$l(\alpha) = \max_{\beta \in p(\alpha)} \{l(\beta)\} + 1\,. \tag{5.2}$$

Proof. Let $k = l(\alpha)$. Then by construction of the graph (Line 5 of Algorithm 3.1), there is an edge $e = \{\beta, \gamma\} \in E_{k-1}$ that α is midpoint on. The endpoints of this edge are the parents of α, $\{\beta, \gamma\} = p(\alpha)$. Because $e \in E_{k-1}$, we know that either $l(\beta) = k-1$ or $l(\gamma) = k-1$. But $l(\beta) < l(\alpha)$ and $l(\gamma) < l(\alpha)$, so $l(\alpha) = \max(l(\beta), l(\gamma)) + 1$. □

5.2.2. Properties for Routing

An important property of the sphere graph, is that when routing between vertices it is never useful to use edges in E_k for higher k than either vertex. Let $\alpha \in V$ be a sender and $\beta \in V$ a receiver. Then this result allows us to ignore the possibility of routing through a higher layer than $l(\alpha)$, when analysing the neighbourhood $N(\alpha)$. That is, if we know $l(\alpha) \geq l(\beta)$. Any path is symmetric, so when $l(\alpha) < l(\beta)$ we can analyse the inverted situation instead, where β is the transmitter and α the receiver. In proofs by contradiction we will use ※ to signal a contradictory statement.

Theorem 5.2.4 (No Higher Edge)**.** *If m is the distance between vertices $\alpha_0, \alpha_m \in V$, and $k = \max(l(\alpha_0), l(\alpha_m))$. Then for all paths between α_0 and α_m of length m holds that they do not use an edge in E_h, where $h > k$.*

Proof. We give a proof by induction over m, the distance between α_0 and α_m. An illustration of the proof is given in Fig. 5.1.
Basis: $m = 1$. By construction of the graph, we know that all edges in E_i are subdivided to obtain E_{i+1}. This continues recursively until the highest layer is reached. This implies that

$$\forall \alpha_0 \forall \alpha_1 \in V_k\,, \quad \{\alpha_0, \alpha_1\} \notin E_h\,. \tag{5.3}$$

Thus there is no shortest path of length 1 that use an edge in E_h.
Induction hypothesis: If there are two vertices $\alpha, \alpha_m \in V$ such that the distance between them is of length $m = \ell - 1$ and $k = \max(l(\alpha_0), l(\alpha_m))$, then there exists no path of length $\ell - 1$ that uses an edge in $E_h, h > k$.
Induction: $m = \ell$. First consider all shortest paths $P_{\alpha_0, \beta_{m-1}}$ of length $m-1$ from α_0 to a vertex β_{m-1} that satisfies

$$\beta_{m-1} \in (V_k \cap N(\alpha_m))\,.$$

By the Induction Hypothesis (IH) all $P_{\alpha_0, \beta_{m-1}}$ cannot contain an edge in E_h, or they would be longer than $m-1$. That means that for any edge in E_k, two or more edges have to be traversed in E_h. When $P_{\alpha_0, \beta_{m-1}}$ is extended using an edge in E_h, then the destination cannot be reached in one hop. Thus the path using an edge in E_h is longer than m hops.

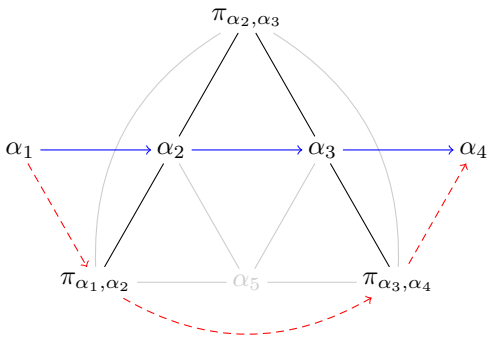


Figure 5.2: An illustration for the proof of Theorem 5.2.5, in the case that $l(\alpha_2) = l(\alpha_3)$. There exists a triangle $\{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_2,\alpha_3}, \pi_{\alpha_3,\alpha_4}\}$ such that there is an edge $\{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_3,\alpha_4}\} \in E$. Now the original path P_{α_1,α_4} (blue, solid arrow) can be rerouted through a lower layer $\hat{P}_{\alpha_1,\alpha_4}$ (red, dashed arrow).

Let us prove by contradiction that there are no paths of length $m-1$ to some vertex on a higher layer $\gamma_{m-1} \in L_i : i > k$

$$P_{\alpha_0,\gamma_{m-1}} = [\alpha_0, \gamma_1, \ldots, \gamma_{m-2}, \gamma_{m-1}]$$

which satisfy $\gamma_{m-1} \in N(\alpha_m)$, such that α_m can be reached in m hops through γ_{m-1}. For a contradiction assume there is some path $P_{\alpha_0,\gamma_{m-1}}$. Then we know that $\gamma_{m-2} \notin p(\gamma_{m-1})$, or else $\{\gamma_{m-2}, \alpha_m\} \in E$ according to Proposition 5.2.2 because $\alpha_m \in p(\gamma_{m-1})$ as $l(\alpha_m) < l(\gamma_{m-1})$. This would contradict that the distance between α_0 and α_m is m. Nor is $\gamma_{m-1} \in p(\gamma_{m-2})$ because that would contradict the IH for the path $P_{\alpha_0,\gamma_{m-1}}$, as it would use an edge to a higher layer than $l(\gamma_{m-1})$. So $l(\gamma_{m-2}) = l(\gamma_{m-1})$. Together with Proposition 5.2.1, that implies there is some common parent $\pi_{\gamma_{m-2},\gamma_{m-1}}$ for which holds that $\gamma_{m-2} \neq \pi_{\gamma_{m-2},\gamma_{m-1}} \neq \gamma_{m-1}$. Let us call this parent $\pi_{\gamma_{m-2},\gamma_{m-1}} = \pi_{2,1}$ for simplicity.

The path

$$P_{\alpha_0,\gamma_{m-2}} = [\alpha_0, \gamma_1, \ldots, \gamma_{m-2}]$$

has length $m-2$, and Proposition 5.2.2 implies that $\{\pi_{2,1}, \alpha_m\} \in E$, because $\alpha_m \in p(\gamma_{m-1})$. Thus it is possible to construct a shortest path of length $m-1$

$$P_{\alpha_0,\pi_{2,1}} = P_{\alpha_0,\gamma_{m-2}} + [\pi_{2,1}]\,.$$

Which must be a shortest path, or else the distance between α_0 and α_m would be less than m. So $P_{\alpha_0,\pi_{2,1}}$ contains an edge

$$\{\gamma_{m-2}, \pi_{2,1}\} \in E_i, \qquad \text{where } i = l(\gamma_{m-2}) = l(\gamma_{m-1}) > l(\alpha_0)\,. \tag{5.4}$$

But $i > l(\pi_{1,2})$, so $i > \max\{l(\pi_{2,1}), l(\alpha_0)\}$, thus contradicting the IH. Because the IH implies there is no shortest path from α_0 to $\pi_{2,1}$ which uses an edge on a higher layer than $\max\{\alpha_0, \pi_{2,1}\}$. ※

Thus we can conclude that the only paths of length $m-1$ are $P_{\alpha_0,\beta_{m-1}}$, for which we have already proven that the Theorem holds. □

Now that we know higher layers are uninteresting for routing, we can start analysing what happens if two vertices are far apart. We show that when two vertices $\alpha, \beta \in V$ are on the same layer and $d(\alpha, \beta) \geq 3$, then we there is some path between α and β that uses only lower layer vertices, except α and β. So when we can guarantee that two vertices are some distance apart, we may assume that there is a parent in $\pi_\alpha \in p(\alpha)$ and a parent in $\pi_\beta \in p(\beta)$ that have $d(\pi_\alpha, \pi_\beta) = d(\alpha, \beta) - 2$. Thus there are some parents that are on a shortest path between α and β.

Theorem 5.2.5 (Three Hops)**.** *Consider a shortest path $P_{\alpha_1,\alpha_{m+1}}$ of length $m \geq 3$ between two vertices on the same layer $\alpha_1, \alpha_{m+1} \in L_k, k > 0$. Then there exists a path also of length m that contains only vertices in $V_h, h < k$ between α_1 and α_{m+1}, except for α_1 and α_{m+1}.*

Proof. We give an inductive proof over m, the shortest path length between a pair of vertices on the same layer k.

Basis: Let there be a shortest path of length $m = 3$ given by $[\alpha_1, \alpha_2, \alpha_3, \alpha_4]$. The way vertices are generated,

any vertex on layer $k > 0$ is connected to two parents. It is sufficient to prove that the path

$$P'_{\alpha_1,\alpha_4} = [\alpha_1, \pi_{\alpha_1,\alpha_2}, \pi_{\alpha_3,\alpha_4}, \alpha_4] \tag{5.5}$$

exists, where $\pi_{\alpha,\beta}$ is the common parent defined in Eq. (5.1). Additionally, we know that

$$\pi_{\alpha_1,\alpha_2} \neq \pi_{\alpha_3,\alpha_4}\,, \tag{5.6}$$

or the shortest path would only be 2 hops. Let us prove by contradiction that this path does exist.

Assume that P'_{α_1,α_4} does not exist. This implies

$$\{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_3,\alpha_4}\} \notin E\,. \tag{5.7}$$

Because of Proposition 5.2.2 we know that if $\pi_{\alpha_1,\alpha_2} \neq \pi_{\alpha_2,\alpha_3}$ then $\{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_2,\alpha_3}\} \in E$, and if $\pi_{\alpha_3,\alpha_4} \neq \pi_{\alpha_3,\alpha_4}$ then $\{\pi_{\alpha_2,\alpha_3}, \pi_{\alpha_3,\alpha_4}\} \in E$. Therefore, together with Eq. (5.6) if

$$\pi_{\alpha_1,\alpha_2} = \pi_{\alpha_2,\alpha_3} \implies \{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_3,\alpha_4}\} \in E\,. \qquad ※ \tag{5.8}$$

$$\pi_{\alpha_2,\alpha_3} = \pi_{\alpha_3,\alpha_4} \implies \{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_3,\alpha_4}\} \in E\,. \qquad ※ \tag{5.9}$$

The final case is $\pi_{\alpha_1,\alpha_2} \neq \pi_{\alpha_2,\alpha_3} \neq \pi_{\alpha_3,\alpha_4}$. Let us show that the following holds

$$l(\alpha_2) = l(\alpha_3)\,. \tag{5.10}$$

For contradiction assume $l(\alpha_2) \neq l(\alpha_3)$. W.l.o.g. assume $l(\alpha_2) < l(\alpha_3)$, then $\pi_{\alpha_2,\alpha_3} = \alpha_2$. That also means that $l(\alpha_3) = l(\alpha_4) = k = l(\alpha_1)$, or the path would already satisfy Eq. (5.5). Thus $l(\alpha_2) < l(\alpha_1)$, and so $\pi_{\alpha_1,\alpha_2} = \alpha_2$. But this contradicts the assumption that $\pi_{\alpha_1,\alpha_2} \neq \pi_{\alpha_2,\alpha_3}$, which means that Eq. (5.10) holds.

Thus $p(\alpha_2) = \{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_2,\alpha_3}\}$ and $p(\alpha_3) = \{\pi_{\alpha_2,\alpha_3}, \pi_{\alpha_3,\alpha_4}\}$, because the common parents are distinct. Together with Eq. (5.10), we can say that because on Line 18 of the subdivision algorithm (Algorithm 3.1) the following triangle must exist or there would be no edge (α_2, α_3)

$$\{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_2,\alpha_3}, \pi_{\alpha_3,\alpha_4}\} \implies \{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_3,\alpha_4}\} \in E\,. \qquad ※ \tag{5.11}$$

This is also illustrated in Fig. 5.2. Thus in all cases for P_{α_1,α_4} there exists an edge $\{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_3,\alpha_4}\} \in E$, so it is always possible to construct P'_{α_1,α_4}.

Induction hypothesis: Assume that for all shortest path of length $3 \leq m < \ell - 1$ there exists a shortest path that contains only vertices $\beta \in V_h, h < k$ between $\alpha_1 \in L_k$ and $\alpha_m \in L_k$, except for α_1 and α_m.

Induction: $m = \ell - 1$: Let there be a shortest path between α_1 and α_ℓ of length $\ell - 1$

$$P_{\alpha_1,\alpha_\ell} = [\alpha_1, \cdots, \alpha_{\ell-1}, \alpha_\ell]\,. \tag{5.12}$$

We distinguish cases by the relation between $l(\alpha_{\ell-1})$ and $l(\alpha_\ell)$. Note that $l(\alpha_{\ell-1}) \not> l(\alpha_\ell)$ because of Theorem 5.2.4.

1. Consider the case that $l(\alpha_{\ell-1}) = l(\alpha_\ell)$. Then we can create a shortest path of length $\ell - 2$ between α_1 and $\alpha_{\ell-1}$ using the Induction Hypothesis (IH)

   $$P_{\alpha_1,\alpha_{\ell-1}} = [\alpha_1, \beta_2, \cdots, \beta_{\ell-2}, \alpha_{\ell-1}] \tag{5.13}$$

   that uses edges $e \in E_h$, except the first and last hop. Extend $P_{\alpha_1,\alpha_{\ell-1}}$ with a hop to α_ℓ so that we get a new shortest path from α_1 to α_ℓ of length $\ell - 1$

   $$P'_{\alpha_1,\alpha_\ell} = [\alpha_1, \cdots, \beta_{\ell-2}, \alpha_{\ell-1}, \alpha_\ell]. \tag{5.14}$$

   We know that $\beta_{\ell-2} \in p(\alpha_{\ell-1})$, because it must be in a lower layer than $\alpha_{\ell-1}$ (IH). Additionally, there must be some common parent $\pi_{\alpha_{\ell-1},\alpha_\ell}$. Parents are connected according to Proposition 5.2.2. Thus we can construct a shortest path of length ℓ which meets the criteria of the theorem by replacing $\alpha_{\ell-1}$ with $\pi_{\alpha_{\ell-1},\alpha_\ell}$

   $$\hat{P}_{\alpha_1,\alpha_\ell} = [\alpha_1, \cdots, \beta_{\ell-2}, \pi_{\alpha_{\ell-1},\alpha_\ell}, \alpha_\ell]\,.$$

2. Otherwise $l(\alpha_{\ell-1}) < l(\alpha_\ell)$. Then there must be some minimal i so that for the set $\mathcal{A} = \{\alpha_i, \alpha_{i+1}, \dots \alpha_{\ell-1}\}$ it holds that $\forall \alpha \in \mathcal{A} : l(\alpha) < l(\alpha_\ell)$. For all vertices in the subpath $\alpha' \in P_{\alpha_1,\alpha_{i-1}}$ holds that $l(\alpha') = l(\alpha_\ell)$, or else i is not minimal or Theorem 5.2.4 would be violated for a path between α' and $\alpha_{\ell-1}$. For the same reason it cannot be the case that there is some $i < j < \ell$ for which $l(\alpha_j) > l(\alpha_\ell)$.

   It is now possible to transform the vertices in the subpath $P_{\alpha_1,\alpha_{i-1}} = [\alpha_1, \dots, \alpha_{i-1}]$ to vertices for which the theorem holds. It must be the case that $l(\alpha_1) = l(\alpha_2) = \dots = l(\alpha_{i-1})$, or Theorem 5.2.4 would be violated. We first show that $i \ngeq 5$. For a contradiction assume $i \geq 5$, then we can apply the IH to create a path

$$P'_{\alpha_1,\alpha_{i-1}} = [\alpha_1, \beta_2, \dots, \beta_{i-2}, \alpha_{i-1}] \tag{5.15}$$

   through a lower layer. But β_{i-2} and α_i are both parents of α_{i-1}, so they are connected according to Proposition 5.2.2. We can thus remove α_{i-1} to create a shorter path from α_1 to α_ℓ, contradicting the assumption that P_{α_1,α_ℓ} was a shortest path. Thus it must be the case that $i < 5$.

   There remain only three cases, $i = 2$, $i = 3$ and $i = 4$. If $i = 2$ then the path already fulfills the theorem. If $i = 3$, we replace α_2 with π_{α_1,α_2} which must be connected to α_3, because α_3 is also a parent of α_2. If $i = 4$, we do the same and replace α_2 with π_{α_1,α_2}, and α_3 with π_{α_2,α_3} which again must be connected. Finally, π_{α_2,α_3} must be connected to α_4 to complete the path.

In all cases it is possible to construct a path according to the theorem. □

We have shown for vertices on the same layer that it is possible to route through their parents, if they are at least 3 hops apart. However, we want to generalise this statement to vertices on any layer. Using the proof in Theorem 5.2.5 we can show that if routing to a lower layer, there is always some parent which is on the shortest path.

Corollary 5.2.6 (Lower Layer Path). *Consider vertices*

$$\alpha_0, \alpha_m \in V : l(\alpha_1) < l(\alpha_m)\,. \tag{5.16}$$

with $d(\alpha_1, \alpha_m) = m$. Then there exists a path also of length m that contains only vertices $\beta \in V_h, h < l(\alpha_1)$ between α_0 and α_m, except α_0.

Proof. We give a proof by induction over m, the length of the shortest path.
Basis: $m = 1$: Then there is a direct hop from α_0 to α_1. This path also satisfies the corollary.
Induction hypothesis: If there is a shortest path $P_{\alpha_0,\alpha_{\ell-1}}$ of length $m = \ell - 1$ between vertices

$$\alpha_0, \alpha_m \in V : l(\alpha_0) < l(\alpha_m)\,. \tag{5.17}$$

Then there exists a shortest path also of length $\ell - 1$ that contains only vertices $\beta \in V_h, h < l(\alpha_0)$.
Induction: $m = \ell$: The proof is similar to Item 2 of Theorem 5.2.5. There is a shortest path of length ℓ

$$P_{\alpha_0,\alpha_\ell} = [\alpha_0, \alpha_1, \dots, \alpha_\ell]\,. \tag{5.18}$$

Then there must be some minimal i so that for the set $\mathcal{A} = \{\alpha_i, \alpha_{i+1}, \dots \alpha_\ell\}$ holds that $\forall \alpha \in \mathcal{A} : l(\alpha) \leq l(\alpha_\ell)$. For all vertices in the subpath $\alpha' \in P_{\alpha_0,\alpha_{i-1}}$ holds that $l(\alpha') > l(\alpha_\ell)$. Or either i is not minimal if we can add an α_{i-1} to $\mathcal{A}$, or Theorem 5.2.4 would be violated for a path between α' and α_ℓ if there is vertex β between α' and α_ℓ which has $l(\beta) > l(\alpha')$. For the same reason it cannot be the case that there is some $i < j < \ell$ for which $l(\alpha_j) > l(\alpha_\ell)$.

It is possible to transform the vertices in the subpath $P_{\alpha_0,\alpha_{i-1}} = [\alpha_0, \dots, \alpha_{i-1}]$ to vertices in a path $P'_{\alpha_0,\alpha_{i-1}}$ for which the Corollary holds, in exactly the same manner as given in Theorem 5.2.5. The path $P'_{\alpha_0,\alpha_{i-1}} +\!\!+ P_{\alpha_i,\alpha_\ell}$ fulfills the Corollary. □

5.3. Labelling

In Definition 4.2.2 we have defined the simple vertex, which have some interesting properties that are formally proved. The most interesting property is that any simple vertex will only add at most one simple vertices to the filtered label. That means that once a vertex is simple, it and its parents will never become non-simple. Which in turn implies that we can remove them and still retain all possible non-simple vertices (as we shall see later, this is indeed what we do).

Proposition 5.3.1 (Simple Vertex Properties)**.** *A simple vertex in* $\text{label}^{\text{f}}(\alpha)_\ell$ *will add no more than one vertex to* $\text{label}^{\text{f}}(\alpha)_{\ell+1}$*. Moreover, if a parent was added to the label by a simple vertex, it is also simple.*

Proof. We will show that once a vertex is simple, it must have some parent that does not get added to the filtered label at index $\ell+1$. Given is a simple vertex $\gamma \in \text{label}^{\text{f}}(\alpha)_\ell$ for some $\ell > 1$. Using the definition of a simple vertex (Definition 4.2.2), we recall a parent $\beta \in p(\gamma)$ has already occurred in the filtered label. The parent β is not added once again to the filtered label, because duplicates are filtered out. There must also be a second parent β_2:

$$\{\beta, \beta_2\} = p(\gamma).$$

According to Proposition 5.2.2 $\{\beta, \beta_2\} \in E$. We will analyze this case-by-case:

1. If $l(\beta) > l(\beta_2)$, then both γ and β will add β_2, because $\beta_2 \in p(\gamma)$ and $\beta_2 \in p(\beta)$. Because each label index is a set, we can view this as only β adding β_2 to the filtered label. Thus the simple node γ does not add any nodes to the filtered label.

   For this argument to hold, it must not be the case that β uses this same argument to defer the parent to γ, which would mean that the argument is cyclical. But $l(\beta) < l(\gamma)$ as $\beta \in p(\gamma)$, so $\gamma \notin p(\beta)$. Thus it is impossible for the argument to become cyclical.

2. If $l(\beta) = l(\beta_2)$, then the fact that β and β_2 are connected allows us to conclude that there is a common parent $\beta \neq \pi_{\beta,\beta_2} \neq \beta_2$ (Proposition 5.2.1) which will be added to the filtered label by β. This occurs in the same step where γ adds β_2 to the filtered label. That causes β_2 to also be simple, because β_2 is adjacent to its parent π_{β,β_2}.

3. If $l(\beta) < l(\beta_2)$, then $\beta \in p(\beta_2)$. Thus β_2 will also be simple, as it is adjacent to its parent β.

Thus for all cases, γ adds at most one parent to $\text{label}^{\text{f}}(\alpha)_{\ell+1}$, which is also simple. □

The labelling as defined in Eq. (4.11) for $\text{label}^{\text{s}}(\alpha)$ and Eq. (4.13) for $\text{label}(\alpha)$ has some properties that we will use often in later proofs. Very important is the fact that $|\text{label}^{\text{s}}(\alpha)_i| \leq 3$, for any i. This greatly limits the size of the labelling, since $|\text{label}(\alpha)_i| \leq |\text{label}^{\text{s}}(\alpha)_i|$. In the proof we perform a case analysis, that we will refer to in later proofs.

Lemma 5.3.2 (Non-Simple Vertex Bound)**.** *The label size* $|\text{label}^{\text{s}}(\alpha)_\ell| \in \{1, 2, 3\}$*, for any* $\alpha \in V$ *and integer* ℓ*. Additionally, for any* $\alpha, \beta \in \text{label}^{\text{s}}(\alpha)_\ell : \alpha \neq \beta$ *it holds that* $\{\alpha, \beta\} \in E$*.*

Proof. We will prove this lemma using induction over a constrained set of cases.
Basis: We know that the label starts with only one non-simple node α, at $\text{label}^{\text{s}}(\alpha)_1$. The parents $\{\beta_1, \beta_2\} = p(\alpha)$ will be added to the label at the next index, if $l(\alpha) > 1$. Either α has one non-simple parent and one simple parent, or it will have two non-simple parents. Simple vertices are removed according Eq. (4.11), leading to a label size of 1 or 2. The parents are also adjacent, according to Proposition 5.2.2. The lemma thus holds for the base case, and results in one of the induction step cases.
Induction hypothesis: Assume that $|\text{label}^{\text{s}}(\alpha)_{\ell-1}| \in \{1, 2, 3\}$, and that all vertices in $\text{label}^{\text{s}}(\alpha)_{\ell-1}$ are pairwise adjacent. The label is thus in one of the following configurations:

1. One non-simple vertex.
2. Two non-simple and adjacent vertices.
3. Three non-simple and pairwise adjacent vertices.

We will show that the induction step will only result in one of these cases.
Induction: According to the Induction Hypothesis (IH) $\text{label}^{\text{s}}(\alpha)_{\ell-1}$ is in one of three configurations, we will handle each case individually.

1. The number of non-simple vertices is 1 for $\text{label}^{\text{s}}(\alpha)_{\ell-1}$. Given that there is only one non-simple vertex that can add two parents, we know that this can result in:
   (a) One non-simple vertex and one simple vertex, where the simple vertex is filtered out according to Eq. (4.13).
   (b) Two non-simple and adjacent vertices,

2. There are 2 non-simple and adjacent vertices in the label. Let the two non-simple vertices be called $\alpha_1, \alpha_2 \in \text{label}^s(\alpha)_{\ell-1}$. Because they are adjacent, there must be a common parent π_{α_1,α_2} according to Proposition 5.2.1. This common parent is not equal to either α_1 or α_2, or else one of them would be a simple vertex (as its parent is in the label) Furthermore, there are parents $\beta_1 \in p(\alpha_1)$ and $\beta_2 \in p(\alpha_2)$ which are distinct from the common parent $\beta_1 \neq \pi_{\alpha_1,\alpha_2}$ and $\beta_2 \neq \pi_{\alpha_1,\alpha_2}$.

   We will show that β_1, β_2 and π_{α_1,α_2} are all pairwise adjacent. There exist edges $\{\pi_{\alpha_1,\alpha_2}, \beta_1\}, \{\pi_{\alpha_1,\alpha_2}, \beta_2\} \in E$ according to Proposition 5.2.2. The edge $\{\beta_1, \beta_2\}$ also exists in E, because $l(\alpha_1) = l(\alpha_2)$ and the way vertices are connected on Line 18 of the graph construction algorithm (Algorithm 3.1). The construction algorithm step implies there is a triangle $\{\beta_1, \pi_{\alpha_1,\alpha_2}, \beta_x\}$, where β_x is a parent of α_2. Since α_2 has only one other parent besides π_{α_1,α_2}, we must have that $\beta_2 = \beta_x$ and thus β_1 and β_2 are adjacent.

   Apart from that, we will also show that this case may only results in either 1 non-simple vertex plus 2 simple vertices, or 3 non-simple vertices. A fact that is used later in a proof.

   Subproof No 2 Non-Simple. For a contradiction, assume that 2 vertices $\gamma_1, \gamma_2 \in \{\beta_1, \pi_{\alpha_1,\alpha_2}, \beta_2\}$ are non-simple, and the last vertex, γ_3, is simple. That means that either $\gamma_1 \in p(\gamma_3)$ or $\gamma_2 \in p(\gamma_3)$. We also know that $l(\gamma_i) < l(\alpha_1) = l(\alpha_2)$ for $i \in \{1, 2, 3\}$. Together with the fact that $l(\gamma_1) = l(\gamma_2)$ (or one of them would be simple, see also the proof of Corollary 5.3.3), we know that

$$l(\gamma_1) = l(\gamma_2) < l(\gamma_3)\,. \tag{5.19}$$

   Assume w.l.o.g. $\{\gamma_1, \gamma_2\} = p(\alpha_1)$. From Eq. (5.19) and Proposition 5.2.3 we can see that $l(\alpha_2) = l(\gamma_3) + 1$, while $l(\alpha_1) = l(\gamma_1) + 1$. Thus

$$l(\alpha_1) = l(\gamma_1) + 1 < l(\gamma_3) + 1 = l(\alpha_2)\,. \tag{5.20}$$

   However, that would mean $\alpha_1 \in p(\alpha_2)$, so α_2 is simple, contradicting the assumption that it is non-simple. A similar contradiction is reached for $\{\gamma_1, \gamma_2\} = p(\alpha_2)$. Thus $\gamma_1 \neq \pi_{\alpha_1,\alpha_2} \neq \gamma_2$, or one of these cases would occur.

   The only remaining case is that $\gamma_1 = \beta_1$ and $\gamma_2 = \beta_2$ (or the inverse). Then there must be some child $\alpha_3 \in V$ on the edge between γ_1 and γ_2, with $l(\alpha_3) = l(\gamma_1) + 1$. It must also be the case that $l(\gamma_1) < l(\pi_{\alpha_1,\alpha_2})$, or it would not be simple. So $l(\alpha_1) = l(\alpha_2) = l(\pi_{\alpha_1,\alpha_2}) + 1$ according to Proposition 5.2.3. As before, we have seen that $\{\beta_1, \pi_{\alpha_1,\alpha_2}, \beta_2\}$ form a triangle, so by construction there must be edges $\{\alpha_1, \alpha_3\}, \{\alpha_2, \alpha_3\} \in E$. But $l(\alpha_3) = l(\gamma_1) + 1 < l(\pi_{\alpha_1,\alpha_2}) + 1 = l(\alpha_1)$, so α_3 is also a parent of α_1 and α_2. This contradicts the assumption that only β_1, β_2 and π_{α_1,α_2} are the parents. ■

   Thus this can result in the following cases:

   (a) One non-simple vertex and the rest is simple. Thus two of the common parents in the triangle were simple and are removed from the label.

   (b) Three non-simple vertices that are all pairwise adjacent.

3. The number of non-simple vertices is 3 and they are all pairwise adjacent. Let the non-simple vertices be called $\alpha_1, \alpha_2, \alpha_3$. Because these vertices are all pairwise adjacent, they form a triangle. Additionally, with the same reasoning as in Item 2 of the induction step

$$l(\alpha_1) = l(\alpha_2) = l(\alpha_3)\,, \tag{5.21}$$

   thus these vertices were generated in the same iteration of the graph generation. A triangle $\{\alpha_1, \alpha_2, \alpha_3\}$ with Eq. (5.21) is only formed by three parents in a triangle $\{\beta_1, \beta_2, \beta_3\}$, according to Line 18 of the graph generation algorithm (Algorithm 3.1). Thus there exist three common parents $\{\pi_{\alpha_1,\alpha_2}, \pi_{\alpha_1,\alpha_3}, \pi_{\alpha_2,\alpha_3}\} = \{\beta_1, \beta_2, \beta_3\}$ that are distinct and also adjacent.

   This can result in the following cases:

   (a) One non-simple vertex and the rest is simple.

(b) Two non-simple and adjacent vertices, and the rest simple.

(c) Three non-simple vertices that are pairwise adjacent.

So for all given cases the lemma holds, and each case results in one given in the IH. □

With the non-simple vertex properties in the label we can give a succinct proof that any two vertices at the same label index must be on the same layer.

Corollary 5.3.3 (Same Layer Label)**.** *Consider two vertices $\alpha, \beta \in \text{label}^s(\gamma)_i$ for some integer i. Then $l(\alpha) = l(\gamma)$.*

Proof by Contradiction. Assume w.l.o.g. $l(\alpha) < l(\beta)$. Then $\alpha \in p(\beta)$ as they are connected (Lemma 5.3.2). But that implies β is simple, as α has occurred in the label, leading to a contradiction. □

Corollary 5.3.3 together with Proposition 5.2.3 also implies that if $\beta \in \text{label}^s(\alpha)_i$, and $p(\beta) \subseteq \text{label}^s(\alpha)_{i+1}$, then $l(\gamma) = l(\beta) - 1 : \gamma \in p(\beta)$. As $l(\pi_1) = l(\pi_2) : \{\pi_1, \pi_2\} = p(\beta)$. So it gives a direct restriction on the layer of the parents in the label. Furthermore Lemma 5.3.2 and Corollary 5.3.3 also hold for the final labelling, since vertices are only removed from the label.

5.4. Proof of Optimality

Given the Algorithm 4.3 we will now show that this algorithm produces a shortest path. We will first analyse the path6 algorithm (Algorithm 4.2), which will give us a lower bound on the distance between the sender α and the receiver β (Lemma 5.4.1) If we know that $d(\alpha, \beta) \geq 3$ we can then apply Theorem 5.2.5 and Corollary 5.2.6 to ensure there is some path through the parents of α and β. We will show that the filtering we have applied to the labelling has resulted in non-simple vertices which are all equally good to route through, if α and β are far apart (Theorem 5.4.2). Finally, we can combine these facts to show that Algorithm 4.3 produces a shortest path in Theorem 5.4.3.

We start off by giving a lower bound on the distance between α and β if path6 does not find a path.

Lemma 5.4.1 (path6 Distance)**.** *Consider two vertices $\alpha, \beta \in V$ for which path6(α, β) (Algorithm 4.2) does not find a path. Then $d(\alpha, \beta) > 3$. Additionally, for any $\pi_\alpha \in p(\alpha)$ and $\pi_\beta \in p(\beta)$, $d(\pi_\alpha, \pi_\beta) > 3$. Finally, for grandparents $\gamma_\alpha \in p(\pi_\alpha)$ and $\gamma_\beta \in p(\pi_\beta)$, $d(\gamma_\alpha, \gamma_\beta) \geq 3$.*

Proof. For the case $d(\alpha, \beta) = 0$ or $d(\alpha, \beta) = 1$, a solution is found with $\{\alpha\} \cap \{\beta\}$, and $\{\alpha\} \cap N(\beta)$ respectively. If $d(\alpha, \beta) = 2$ then $N(\alpha) \cap N(\beta)$ will find it.

For $d(\alpha, \beta) = 3$ there are two cases:

1. If $l(\alpha) = l(\beta)$ and $d(\alpha, \beta) = 3$. Then we can use the Theorem 5.2.5 Theorem to show that there must be some parents $\gamma_\alpha \in p(\alpha)$ and $\gamma_\beta \in p(\beta)$, so that $\gamma_\alpha \in N(\gamma_\beta)$ because $d(\gamma_\alpha, \gamma_\beta) = 1$. Thus $N(\alpha) \cap N(p(\beta)) \neq \emptyset$ for $d(\alpha, \beta) = 3$.

2. Otherwise $l(\alpha) \neq l(\beta)$. Assume w.l.o.g. that $l(\alpha) > l(\beta)$. By Corollary 5.2.6 we know there is some vertex in $p(\alpha)$ on a shortest path $P_{\alpha,\beta}$. Thus $N(p(\alpha)) \cap N(\beta)$ will always find the shortest path $P_{\alpha,\beta}$.

To show that $d(\pi_\alpha, \pi_\beta) > 3$ we closely investigate N^α and N^β. We remove the entries not containing $p(\alpha)$ and $p(\beta)$

$$\hat{N}^\alpha = N(p(\alpha)) \cup N(p(p(\alpha)))\,, \tag{5.22}$$

$$\hat{N}^\beta = N(p(\beta)) \cup N(p(p(\beta)))\,. \tag{5.23}$$

Then we can see that the above proof also holds for $p(\alpha)$ and $p(\beta)$. Since $p(p(\alpha))$ is never used in the proof, we can replace $p(\alpha)$ with some $\gamma \in p(\alpha)$ and $\eta \in p(\beta)$ to create

$$\hat{N}^\alpha = N(\gamma) \cup N(p(\gamma))\,, \tag{5.24}$$

$$\hat{N}^\beta = N(\eta) \cup N(p(\eta))\,. \tag{5.25}$$

For which we can see that the same proof applies if $\alpha = \gamma$ and $\beta = \eta$.

Finally, we need to show that $d(\gamma_\alpha, \gamma_\beta) \geq 3$. Let us look at the entries containing γ_α and γ_β

$$\widetilde{N}^\alpha = N(p(p(\alpha)))\,, \tag{5.26}$$
$$\widetilde{N}^\beta = N(p(p(\beta)))\,. \tag{5.27}$$

With the same steps as for α and β for $d(\alpha, \beta) \leq 2$ we can show that $N(p(p(\alpha)) \cap N(p(p(\beta)$ finds all paths if $d(\gamma_\alpha, \gamma_\beta) < 3$. Thus $d(\gamma_\alpha, \gamma_\beta \geq 3$ if path6 does not find a path. □

Using the guarantee on the distance between α and β that path6 gives, we show that if path6 did not find a path then all vertices in $\text{label}(\alpha)_2$ are equidistant from β. This allows us to chose any vertex in $\text{label}(\alpha)_2$ at random. Conversely, this will also be useful when routing up the layers towards β in the local algorithm (Section 4.4). Because any vertex in $\text{label}(\alpha)_2$ may be chosen to be in the shortest path, it also does not matter which vertex is chosen if coming from a vertex $\gamma \in \text{label}(\beta)_3$ and routing towards β.

Theorem 5.4.2 (Label Routing). *Consider two nodes $\alpha, \beta \in V, l(\alpha) \geq l(\beta)$ so that path6(α, β) (Algorithm 4.2) cannot determine a path between them. Then for every node $\pi_1 \in \text{label}(\alpha)_2$ holds that $d(\pi_1, \beta) = d(\alpha, \beta) - 1$, i.e. they are on a shortest path between α and β.*

Proof. By Lemma 5.4.1 we know that $d(\alpha, \beta) > 3$, or path6(α, β) would have found a path. The same holds for the parents

$$d(\pi_\alpha, \pi_\beta) > 3 : \pi_\alpha \in p(\alpha), \pi_\beta \in p(\beta)\,. \tag{5.28}$$

From Theorem 5.2.4 we also know that only vertices $\eta \in N(\alpha) : l(\alpha) \geq l(\eta)$ are relevant. We first show that routing through π_1 is not worse than routing through some simple parent $\pi_2 \in p(\alpha)$, if it exists. Then we show that if there are two non-simple parents, then routing through π_1 is not worse than the other parent. We first assume for a proof by contradiction that $d(\pi_2, \beta) < d(\pi_1, \beta)$.

Because $l(\beta) \leq l(\alpha)$ we can apply either Theorem 5.2.5 or Corollary 5.2.6 to see that there is a shortest path that includes π_2. Furthermore, if $l(\beta) = l(\alpha)$ then there is some parent $\pi_\beta \in p(\beta)$ in the shortest path, otherwise let $\pi_\beta = \beta$. In both cases $d(\pi_2, \pi_\beta) \geq 3$, because if $\pi_\beta = \beta$ then $d(\pi_2, \pi_\beta) = d(\alpha, \beta) - 1$, otherwise the distance is guaranteed by Eq. (5.28).

We can then show that there is a shortest path through the parents of π_2. If π_2 is simple, then $l(\alpha) = l(\pi_2) + 1$ (Proposition 5.2.3), if it is non-simple then still $l(\alpha) = l(\pi_2) + 1$. Thus it also holds that

$$l(\pi_2) \geq l(\pi_\beta)\,. \tag{5.29}$$

since $l(\alpha) > l(\pi_\beta)$. Once again, since we do not know whether $l(\pi_\beta) = l(\pi_2)$ or $l(\pi_\beta) < l(\pi_2)$. In the former case, let us call $\gamma_\beta \in p(\beta)$ the parent the shortest path goes through, otherwise $\gamma_\beta = \pi_\beta$. We can then apply Theorem 5.2.5 or Corollary 5.2.6 once more to see that there is a parent $\gamma_{\pi_2} \in p(\pi_2)$ which has $d(\gamma_{\pi_2}, \beta) = d(\pi_2, \beta) - 1$ on the path.

$$P_{\pi_2, \gamma_\beta} = [\pi_2, \gamma_{\pi_2}, \gamma_1, \gamma_2, \ldots, \gamma_\beta, \pi_\beta]\,, \tag{5.30}$$

where for convenience we have assumed that if $\gamma_\beta = \pi_\beta$ the last hop is not performed.

Subproof Simple Parents. Let π_2 be a simple parent, then it must have a parent already in the label. The parent in the label can only be π_1, as π_1 and α are the only other vertices in the label up to index 2. From Eq. (5.30) we can see that γ_{π_2} is on the shortest path. However, parents of a node are connected (Proposition 5.2.2), i.e. $\{\pi_\pi, \pi_1\} \in E$, so at most $d(\pi_1, \beta) = d(\gamma_{\pi_2}, \beta) + 1 = d(\pi_2, \beta)$, which contradicts the assumption $d(\pi_2, \beta) < d(\pi_1, \beta)$. Thus, for all cases, routing through π_1 is never worse than routing through π_2. ■

Subproof Non-Simple Parents. We will show that it does not matter which non-simple parent is routed to. Consider the case that $\{\pi_1, \pi_2\} = p(\alpha)$ and π_1, π_2 are non-simple, so $\{\pi_1, \pi_2\} = \text{label}^s(\alpha)_2$. Note that $\pi_1 \in \text{label}(\alpha)_2$, as per the theorem. We know that $l(\pi_1) = l(\pi_2)$, according to Corollary 5.3.3. For a contradiction assume that $d(\pi_2, \beta) < d(\pi_1, \beta)$.

Consider the path in Eq. (5.28). It is the case that $\gamma_{\pi_2} \neq \pi_{\pi_2, \pi_1}$, or the assumption $d(\pi_2, \beta) < d(\pi_1, \beta)$ would not hold as they would have an equal distance to β. The label filtering of Eq. (4.13) implies that π_1 has at least one non-simple parent. So there must be three non-simple parents if $\gamma_{\pi_2} \neq \pi_{\pi_2, \pi_1}$ according to Item 2 in Lemma 5.3.2, because π_2 is also non-simple and thus adds a distinct non-simple parent to

$\text{label}^s(\alpha)_2$. Note that it is not known whether

$$\pi_{\pi_2,\pi_1} \overset{?}{\in} \text{label}(\alpha)_2\,, \text{or} \tag{5.31}$$

$$\gamma_{\pi_2} \overset{?}{\in} \text{label}(\alpha)_2\,. \tag{5.32}$$

Let us call the other parent of π_1, γ_{π_1}, where

$$\gamma_{\pi_1} \in p(\pi_1) : \gamma_{\pi_1} \neq \pi_{\pi_2,\pi_1}\,. \tag{5.33}$$

The vertices $\{\gamma_{\pi_1}, \pi_{\pi_2,\pi_1}, \gamma_{\pi_2}\}$ form a triangle, as shown in Item 2 in Lemma 5.3.2. They are also at the same index in $\text{label}^s(\alpha)$, so

$$l(\gamma_{\pi_2}) = l(\gamma_{\pi_1}) = l(\pi_{\pi_2,\pi_1})\,, \tag{5.34}$$

as shown in Corollary 5.3.3. Thus, the layer of these vertices is fixed at $l(\pi_2) = l(\gamma_{\pi_2}) + 1$ as shown in Proposition 5.2.3. From that we can conclude that $l(\gamma_{\pi_2}) \geq l(\gamma_\beta)$, since $l(\pi_2) > l(\gamma_\beta)$. This in turn shows that for the vertices in the path of Eq. (5.30)

$$l(\gamma_{\pi_2}) \geq l(\gamma_i)\,, \tag{5.35}$$

for integer i, according to Theorem 5.2.4.

We need to show that $d(\gamma_{\pi_2}, \gamma_\beta) \geq 3$, so that there is a path through a lower layer. From Lemma 5.4.1 we know that if $\gamma_\beta \in p(p(\beta))$ then this holds, since $\gamma_{\pi_2} \in p(p(\alpha))$. However, if $\gamma_\beta = \pi_\beta$ then still $N(p(p(\alpha)) \cap N(p(\beta))$ will guarantee that they are separated by a distance of 3 or more. The same holds for $\pi_\beta = \beta$. Finally, if $\gamma_\beta = \pi_\beta = \beta$, then $N(p(p(\alpha)) \cap N(\beta)$ will guarantee the distance. Thus know that there must be a γ_1 and γ_2 in $P_{\pi_2,\beta}$, and $\gamma_2 \neq \gamma_\beta$. But it could be the case that $\gamma_3 = \gamma_\beta$.

1. If $l(\gamma_1) < l(\gamma_{\pi_2})$ then γ_1 is a parent of γ_{π_2}, and thus also of either π_{π_2,π_1} or γ_{π_1} (Item 3 of Lemma 5.3.2). That means there exists a path through π_1 of equal distance to β, if we assume w.l.o.g. that γ_{π_1} is adjacent to γ_1 then it is possible to traverse

   $$[\pi_1, \gamma_{\pi_1}, \gamma_1]\,. \tag{5.36}$$

   This is equal in length to

   $$[\pi_2, \gamma_{\pi_2}, \gamma_1]\,. \tag{5.37}$$

2. Otherwise if $l(\gamma_{\pi_2}) = l(\gamma_1)$ or $l(\gamma_{\pi_2}) = l(\gamma_1) = l(\gamma_2)$. Then we can transform the path using Corollary 5.2.6 to another shortest path which starts with a vertex in $p(\gamma_{\pi_2})$. Then we can apply the same reasoning as in Item 1 to show that there is a shortest path through π_1.

3. Finally, if $l(\gamma_{\pi_2}) = l(\gamma_i)$ for $i \in \{1, 2, 3\}$, we can use Theorem 5.2.5 to create a lower layer path

   $$P_{\gamma_{\pi_2},\gamma_3} = [\gamma_{\pi_2}, \eta_1, \eta_2, \gamma_3]\,. \tag{5.38}$$

   Again $\eta_1 \in p(\gamma_{\pi_2})$ so we can apply Item 1.

In all cases of $P_{\pi_2,\beta}$ there is an equivalent length path starting at π_1 so both non-simple parents π_2 and π_1 must have an equal distance to β. ■

□

We know that path6 does a neighbourhood search and finds the minimal path in that neighbourhood. Otherwise, if Algorithm 4.3 routes to a lower layer using the label then that is also on a shortest path. So in all cases the algorithm performs a move that is on the shortest path, as we will show.

Theorem 5.4.3 (Sphere Routing Optimality)**.** *Consider vertices $\alpha, \beta \in V$ with a distance m, the sphere routing algorithm (Algorithm 4.3) will find a path $P_{\alpha,\beta}$ also of length m.*

Proof. The routing algorithm Algorithm 4.3 consists of two cases. If α and β are close to each other, then path6(α, β) will apply and find a shortest path using a brute force search in the neighbourhood of α and β. Otherwise, Theorem 5.4.2 applies. If $l(\alpha) = l(\beta)$ then α may always route to a parent $\gamma \in \text{label}(\alpha)_2$. The cases $l(\alpha) < l(\beta)$ and $l(\alpha) > l(\beta)$ are synonymous, in the former β routes to a parent in $\text{label}(\beta)_2$, and in the latter α does to a parent in $\text{label}(\alpha)_2$. In all cases the routing algorithm performs a step which is on

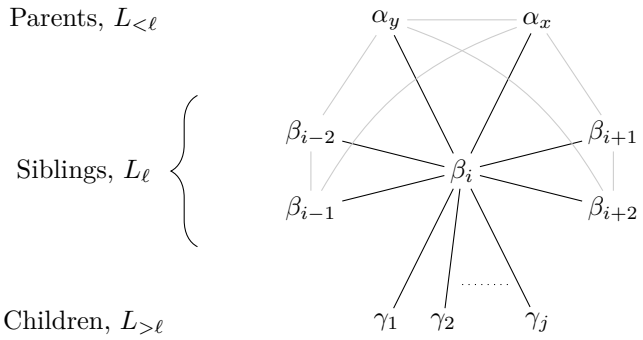


Figure 5.3: The neighbourhood of a node, which is directly related to the size complexity. Some edges of the neighbours are drawn using a light gray line. The node β_i stores the IDs of 2 parents, 4 siblings, and $O(k)$ children, where k is the number of layers.

a shortest path between α and β. Lastly, the algorithm will always terminate by path6, as shown in the discussion in Section 4.3. Thus the routing algorithm returns the shortest path between α and β.

We have proved for all cases that routing through π_1 is never worse than through a different vertex in $p(\alpha)$, if path6(α, β) cannot determine a path. Theorem 5.2.5 and Corollary 5.2.6 with $d(\alpha, \beta) > 3$ imply that there is no sibling $\gamma \in N(\alpha) : l(\gamma) = l(\alpha)$, which is strictly closer to β than some parent in $p(\alpha)$. Thus any $\pi_1 \in \text{label}(\alpha)_2$ will be on a shortest path to β if `path6` does not find a path. □

5.5. Complexity Analysis

The size of the node ID of $\alpha \in V$ can be expressed as

$$|\text{ID}_\alpha| = \lceil \log_2(|V|) \rceil = O(k)\,, \tag{5.39}$$

where the number of layers is (3.23)

$$k = 2 \log_2 \left(\frac{|V| - 2}{10} \right) . \tag{5.40}$$

The ID is saved as a unique binary number in the graph. Since there are $|V|$ IDs, it has size $\lceil \log_2(|V|) \rceil$. We will first analyse the space complexity of the data stored in every vertex. Using that information we are able to analyse the time complexity of routing per vertex.

Theorem 5.5.1 (Local Space Complexity)**.** *The space complexity per vertex of the local routing algorithm is* $O(\log^5 |V|)$.

Proof. First we analyze the space complexity of the labelling, followed by that of N_3^α with in between less significant contributors to the space complexity.

For every $i \in \{1, \ldots, l(\alpha)\}, |\text{label}(\alpha)_i| \leq 3$ (Lemma 5.3.2). Where every entry is an ID, resulting in a label size of (5.39)

$$3 \cdot l(\alpha) \cdot \log_2 |V| = O(k^2)\,. \tag{5.41}$$

A vertex must store its own label, and receive a label of the target node β, which has a space complexity of $O(k^2)$.

We assume that the data N_3^α for the local path6 algorithm (Algorithm 4.4) is stored in the network node. Alternatively, it may also request it over the network, but this costs time instead of space. The neighbourhood of a node is illustrated in Fig. 5.3. The number of children of a node is (Section 5.2)

$$\sum_{i=l(\alpha)+1}^{k} 6 = 6(k - l(\alpha)) = O(k)\,. \tag{5.42}$$

The neighbourhood of a node $N(\alpha)$ thus has $2+4+6(k-l(\alpha)) = O(k)$ vertices in it, and children(α) has at most $6(k - l(\alpha)) = O(k)$ vertices in it. Thus N_3^α, the largest set contains $O(k^2)$ neighbours, each of which have $O(k^2)$ children. So in total, with the ID size, $|N_3^\alpha| = O(k^2 \cdot k^2 \cdot k) = O(k^5)$, because we only need to store the IDs of nodes in this set and not their labels. Since $|N^\alpha| < |N_2^\alpha| < |N_3^\alpha|$ the space complexity of the neighbourhood is

$$|N_3^\alpha| = O(k^5)\,. \tag{5.43}$$

For $\alpha \in V_0$, it is also necessary to store G_0 to perform Dijkstra's algorithm on it, according to the last case of path6. But $|G_0| = O(1)$, so this is negligible. The d flag is also of negligible size, $O(1)$. The main routing algorithm in Algorithm 4.5 stores the label of α and β, but those have already been taken into account. Storing the IDs of the neighbourhood of α is $O(k^2)$. Thus in total the space complexity per vertex is $O(k^2 + k^5 + 1 + k^2) = O(k^5)$. □

Now that we know the sizes of the data structures, we can use that information in the time complexity analysis. Given two sets $\mathcal{A}$ and $\mathcal{B}$, where the size complexity is $|\mathcal{A}| = O(x)$ and $|\mathcal{B}| = O(y)$. Then the time complexity of $A \cap B$ is

$$O(\min(x, y)) . \tag{5.44}$$

We can achieve this by taking each element from the smallest set, and checking presence in the other set. If we assume each element of the smallest set has size $O(z)$, then hashing it will takes $O(z)$, finding the element in a hash set amortized $O(1)$, and an equality comparison $O(z)$. However, even then the time complexity is equal to the set size $O(n \cdot (z + z + 1 + z)) = O(nz)$, if n is the number of elements in the smallest set.

Theorem 5.5.2 (Local Time Complexity)**.** *The time complexity of the local routing algorithm is* $O(\log^2 |V|)$ *per vertex.*

Proof. First we investigate the time complexity of path6 (Algorithm 4.4), going from top to bottom. It is assumed that N_3^α is already stored on the network node. The largest contributor to the time complexity is computing the intersection $N_3^\alpha \cap L_\beta$. As shown in the space complexity analysis (Theorem 5.5.1), the size of the N_3^α set is $O(k^5)$ (5.43) and of L_β is $O(k^2)$ (5.41). The time complexity of the intersection $N_3^\alpha \cap L_\beta$ then is $O(k^2)$ (Eq. (5.44))

Once an intersection has been found we know the set of possible $\gamma \in N_3^\alpha \cap L_\beta$, which we will call Γ. We know that $\Gamma \subseteq L_\beta$, so $|\Gamma| = O(k^2)$. To calculate $d(\alpha, \gamma)$ and $\delta(\gamma, \beta)$ we perform the following steps. For every $\gamma \in \Gamma$ we assume the implementation has stored a *breadcrumb trail* to α, which is a path $P_{\gamma,\alpha}$ of size at most 6 indicating the shortest path to α (of size $O(1)$). Thus we can calculate $d(\alpha, \gamma)$ in $O(1)$. So mapping $\forall \gamma \in \Gamma \to d(\alpha, \gamma)$ takes $O(k^2)$ time complexity.

To calculate $d(\gamma, \beta)$ an iterative step is performed through $\text{label}(\beta)_i$ for $i = 1, 2, \ldots, |\,\text{label}(\beta)|$. For every i $\beta' \in \Gamma : \beta' \in \text{label}(\beta)_i$ is checked, and if so $d(\gamma, \beta) = i$. Again, looking up β' in Γ takes $O(k(k + k)) = O(k^2)$ because of the ID size, hashing, and the amortized $O(1)$ lookup in a hash set. In this way we map $\forall \gamma \in \Gamma \to d(\beta, \gamma)$ in $O(k^2)$.

Summing both these maps to $d(\alpha, \gamma) + d(\beta, \gamma)$ takes no longer than $O(k^2)$. Finding the minimum in this data structure can be done through a linear traversal in $O(k^2)$, because the data is of size $O(k^2)$. Thus it is possible to calculate $\min_\gamma \{d(\alpha, \gamma) + d(\beta, \gamma)\}$ in $O(k^2)$. As seen before, the path from α to $\gamma^{\min}$ is readily available for a given γ in $O(1)$ by a breadcrumb trail. Finally, there is an edge case in which Dijkstra's algorithm is required. First we need to calculate $L_0 \cap L_\beta$, which takes $O(k^2)$ because L_0 is of size $O(k)$ elements, and L_β is $O(k^2)$. The Dijkstra algorithm has a time complexity of $O(|E_0| \log |V_0|)$ [KT06] because it runs on G_0. However, $|V_0| = O(k)$ and $|E_0| = O(k)$ as they contain IDs, thus Dijkstra's algorithm runs in $O(k)$. Finding the minimum path over $\beta' \in L_0 \cap L_\beta$ is only $O(k \cdot k) = O(k)$, as the set $L_0 \cap L_\beta$ is has at most 3 elements and Dijkstra runs in $O(k)$. Thus the total time complexity of path6 is

$$O(k^2) = O(\log^2 |V|) . \tag{5.45}$$

Using the time complexity of path6, we can calculate the complexity of the main routing algorithm. First is a check $\alpha = \beta$ which is $O(k)$ because of the ID size. Then $u = 0$ is insignificant with a complexity of $O(1)$. Followed by a call to path6, of which we have seen that the time complexity is $O(k^2)$. Finding α^{new} is only $O(k)$ because $\text{label}(\alpha)_2 \leq 2$, and each entry is an ID of $O(k)$. Finally, finding $i : \alpha \in \text{label}(\beta)_i$ is $O(k^2)$, because we have to do a linear search through $\text{label}(\beta)$ and compare each element with the ID of α in $O(k)$. The neighbourhood search $\text{label}(\beta)_{i-1} \cap N(\alpha)$ is $O(3(k + k)) = O(k)$, because we have to look up at most 3 IDs in the $N(\alpha)$ set. Finally, creating the lists $[\alpha^{\text{new}}]$ and $[\beta^{\text{new}}]$ is $O(k)$. Thus the total time complexity of the algorithm is

$$O(k^2) = O(\log^2 |V|) , \tag{5.46}$$

proving the theorem. □

Since the diameter of the graph $D(G) = 2k + 3 = O(k)$ (Proposition 3.2.1) and every vertex takes $O(k^2)$, the routing of any packet will take at most $O(k \cdot k^2) = O(\log^3 |V|)$ time.

6 Conclusion

Research into quantum mechanics has shown that it is possible to communicate with perfect secrecy. To do so, quantum entanglement is required between communicating parties. Even though it is possible to establish a direct physical connection between all parties on a small scale, this quickly becomes infeasible for a large number of participants. A solution to this problem is a quantum network, where entanglement between distant parties may be established by performing entanglement swaps. We approached the novel problem of the structure of quantum networks by investigating the case of satellites with quantum communication capabilities. We modelled satellites around the earth as network nodes and suggested improvements to the resulting network.

The network created by satellites can be improved by creating entanglement between distant nodes, so that the diameter of the graph is reduced significantly. We give a subdivision algorithm which starts from a base icosahedron and adds more and more nodes to the graph, until the desired number of nodes is reached. The graphs that are generated in this way are used to structure the physical and virtual links in the network in a recursive and structured fashion. The resulting network has a diameter that is upper bounded by a logarithm in the number of nodes. With a lower diameter the communication time decreases, since less entanglement swaps are necessary to establish entanglement between endpoints, and each swap may fail. As such, it is less likely that retries are necessary to establish a connection with a lower failure rate. Furthermore, the network has a logarithmic upper bound on the quantum memory size that is required of every network node, so that the network can be physically implemented even with the constraints on quantum memory size.

Because of the structure in the network it is also possible to give a routing algorithm for the shortest path that uses only local information. Standard routing algorithms such as Dijkstra's algorithm require every network node to have global knowledge of the network, but for large networks this comes at a cost of a large classical memory size. We instead proposed an algorithm which only requires information about its direct surroundings, and a memory size that is an exponential factor smaller, which is logarithmic in the number of nodes. To do this, we first analysed the structure of the network model and proved that long paths can always be routed through the parents of nodes. We then created a labelling that uniquely identifies nodes in the graph, and allows nodes to route with local knowledge even though they are not in the neighbourhood of the destination. With the labelling, we proposed a global routing algorithm for finding shortest paths that we proved the optimality of. This global algorithm can then be rewritten into a local algorithm that requires only local knowledge of a network router to route. Furthermore, the local routing algorithm is also exponentially faster than a standard routing algorithm such as Dijkstra's algorithm, since it has a run-time that is logarithmic in the number of nodes.

6.1. Discussion and Future Work

During this project we encountered some interesting extensions to our problem that we would like to discuss and may lead to future lines of research.

6.1.1. Implementation Parameters

We have greatly restricted the physical parameters involved in creating entanglement, and left the distribution of entanglement to some background process. Some of the parameters that are relevant to the distribution of entanglement are:

Coherence time of entanglement Once entanglement has been established, the entangled state will slowly lose its entanglement until it is no longer usable. Depending on this parameter, it may be impossible to create very long links since states could decohere too much before proper entanglement can be established.

Entanglement event time The time it takes to create entanglement between two physically adjacent vertices also influences how easily a path can be established. If the event time is very long, then any previously generated entanglement will have already decohered before an entanglement swap can be made.

Noise introduced by local operations We have talked about how entanglement swaps can introduce noise, so that it is better to have less entanglement swaps for any communication. Besides entanglement swaps, noise may also be introduced by other operations performed on the quantum states.

For one, we assumed that every step in the entanglement distribution process results in only perfectly entangled states, because of purification. However, because of local errors, purification cannot create perfectly entangled states. Depending on the amount of error introduced, it may be less feasible to construct longer-distance links, since the number of states to perform successful purification increases with increasing noise. For long-distance links this means more repeated entanglement swapping and purification steps to create multiple redundant states, so that they can be purified.

Other operations that may also introduce noise include measurements and correction rotations that have to be performed after a teleportation.

Thus the process of distributing entanglement depends on the underlying implementation. This may also influence the design of the graph, since some structures may not be feasible. It would be interesting to see the influence of these parameters on quantum networks, and which requirements are essential for correct functioning.

6.1.2. Multipartite Entanglement

In this thesis we assume that entanglement is only distributed in pairs, but it is also possible to create entanglement between an arbitrary number of nodes, which is called multipartite entanglement. This could reduce the number of qubits necessary to store for the virtual links, since only one qubit has to be stored per multipartite entanglement instead of one per virtual link. There has also been some related research into entanglement percolation which we have seen in Section 2.4.1. Maybe there is some way to transform multipartite entanglement to quickly establish connections between a sender and a receiver. However, multipartite states decohere faster than two-party entangled states, where the time to decohere decreases for increasing number of participants. This has to be taken into account for the feasibility of a network making use of multipartite states, and especially large multipartite states.

6.1.3. Robustness of the Graph

It is also interesting to see how well the graph and routing algorithm perform with many simultaneous requests. Since each entangled pair is single-use only, it is desirable to spread paths so that more valuable resources are used less. Longer distance virtual links are harder to create, so we would like them to be used less than shorter links. However, since the shortest path is usually through some ancestors of nodes, it seems likely that edges on lower layers will have a higher demand than on higher layers for randomly selected senders and receivers.

To analyse this problem, we have attempted to analytically express the total number of paths that pass through any one edge. But this analysis was severely complicated by the fact that many edges, even within one layer, have different numbers of paths passing through them. Thus edges on a layer are not uniform, nor were the vertices. We can see this from the simple and non-simple vertices. Non-simple vertices will have more paths using them, because they are faster to reach lower layers. Thus we decided to perform simulations instead, to gain some insight into the performance. Unfortunately, these simulations are not in a presentable form yet, but may be referenced at [Sch15].

Furthermore, it may be possible to specify what happens when a virtual link is unavailable. One solution is to route through a higher layer instead. If a lower layer edge is unavailable, there could be a replacement path through two higher layer edges. If these are unavailable, then even higher layer edges could be used. Since all faces form triangles it may also be trivially possible to route through using the other two edges in the triangle instead.

6.1.4. Varying Level of Detail

It may also be interesting to see what happens when some part of the sphere is less densely or more densely populated by vertices. For example, some area above Europe or America may have more satellites than the Great Pacific ocean. In the subdivided graph this can be achieved by subdividing some faces of the graph a different amount of times than others.

The algorithm uses only local information, except for the label of the destination. Since the label is still of the same structure for all areas of the graph, no matter the amount of subdivisions of other areas on the graph, we believe that the routing scheme is able to handle these differences in detail.

6.1.5. Generalisation of Recursive Graphs

The subdivided icosahedron has a recursive structure, where each triangle contains three triangles, that again contain three triangles, etcetera. If we have some other graph that also contains such a recursive structure, would the same entanglement distribution strategy and a similar routing algorithm then also apply? For example, the grid network can be seen as a recursive graph that starts as a square, and adds more and more internal squares. We do not know if it is possible to generalise these recursive graphs.

However, we do believe that the approach followed in this thesis may apply to more recursive graphs than just the sphere. The labelling only relies on the set of parents that generate a node, which can be extended to any number of parents that generate a node. Then the question remains if variations of No Higher Edge (Theorem 5.2.4) and Three Hops (Theorem 5.2.5) also hold on these graphs. If so, then it may be possible to show that routing through the parents is also a feasible strategy on these graphs. But if the number of parents is not a constant, then all operations involving the label will at least also have an equivalent complexity increase.

Bibliography

[ACL07] Antonio Acín, J. Ignacio Cirac, and Maciej Lewenstein. "Entanglement percolation in quantum networks". In: *Nature Physics* 3.4 (2007), pp. 256–259. DOI: 10.1038/nphys549.

[ADR82] Alain Aspect, Jean Dalibard, and Gérard Roger. "Experimental Test of Bell's Inequalities Using Time- Varying Analyzers". In: *Phys. Rev. Lett.* 49 (25 1982-12), pp. 1804–1807. DOI: 10.1103/PhysRevLett.49.1804.

[AGR81] Alain Aspect, Philippe Grangier, and Gérard Roger. "Experimental Tests of Realistic Local Theories via Bell's Theorem". In: *Phys. Rev. Lett.* 47 (7 1981-08), pp. 460–463. DOI: 10.1103/PhysRevLett.47.460.

[Alg15] Algemene Inlichtingen- en Veiligheidsdienst. *Bereid u voor op de komst van Quantumcomputers*. (Dutch). 2015-04-15. URL: https://www.aivd.nl/onderwerpen/infobeveiliging/publicaties/bereid-komst/ (visited on 2015-07-16).

[Bak94] Brenda S. Baker. "Approximation algorithms for NP-complete problems on planar graphs". In: *Journal of the ACM* 41.1 (1994-01), pp. 153–180. DOI: 10.1145/174644.174650.

[BB84] C.H. Bennett and Gilles Brassard. "Quantum cryptography: Public key distribution and coin tossing." In: *Proc. IEEE Int. Conf. on Comp., Sys. and Signal Process.* 1 (1984), pp. 175–179.

[Bea76] Dwight R. Bean. "Effective coloration". In: *The Journal of Symbolic Logic* 41 (02 1976-6), pp. 469–480. DOI: 10.1017/S0022481200051549.

[Bel64] John S Bell. "On the einstein-podolsky-rosen paradox". In: *Physics* 1.3 (1964), pp. 195–200.

[Ben+96] Charles H. Bennett et al. "Purification of Noisy Entanglement and Faithful Teleportation via Noisy Channels". In: *Phys. Rev. Lett.* 76 (5 1996-01), pp. 722–725. DOI: 10.1103/PhysRevLett.76.722.

[Ber+13] H Bernien et al. "Heralded entanglement between solid-state qubits separated by three metres". In: *Nature* 497.7447 (2013), pp. 86–90. DOI: 10.1038/nature12016.

[BHK05] Jonathan Barrett, Lucien Hardy, and Adrian Kent. "No Signaling and Quantum Key Distribution". In: *Phys. Rev. Lett.* 95 (1 2005-06), p. 010503. DOI: 10.1103/PhysRevLett.95.010503.

[BPT92] Richard B. Borie, R. Gary Parker, and Craig A. Tovey. "Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families". In: *Algorithmica* 7.1-6 (1992-06), pp. 555–581. DOI: 10.1007/bf01758777.

[Bri+98] H.-J. Briegel et al. "Quantum Repeaters: The Role of Imperfect Local Operations in Quantum Communication". In: *Phys. Rev. Lett.* 81 (26 1998-12), pp. 5932–5935. DOI: 10.1103/PhysRevLett.81.5932.

[BVK98] S. Bose, V. Vedral, and P. L. Knight. "Multiparticle generalization of entanglement swapping". In: *Physical Review A* 57.2 (1998-02), pp. 822–829. DOI: 10.1103/physreva.57.822.

[CC09] Martí Cuquet and John Calsamiglia. "Entanglement Percolation in Quantum Complex Networks". In: *Phys. Rev. Lett.* 103 (24 2009-12), p. 240503. DOI: 10.1103/PhysRevLett.103.240503.

[Cla+69] John F Clauser et al. "Proposed experiment to test local hidden-variable theories". In: *Physical review letters* 23.15 (1969), p. 880. DOI: 10.1103/PhysRevLett.23.880.

[Cre+05] P. Crescenzi et al. *A compendium of NP optimization problems*. 2005-07. URL: http://www.nada.kth.se/~viggo/problemlist/compendium.html (visited on 2015-07-16).

[DJ92] David Deutsch and Richard Jozsa. "Rapid Solution of Problems by Quantum Computation". In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 439.1907 (1992), pp. 553–558. DOI: 10.1098/rspa.1992.0167.

[DM10] L.-M. Duan and C. Monroe. "*Colloquium* : Quantum networks with trapped ions". In: *Rev. Mod. Phys.* 82 (2 2010-04), pp. 1209–1224. DOI: 10.1103/RevModPhys.82.1209.

[DR99] Joan Daemen and Vincent Rijmen. *AES proposal: Rijndael*. 1999.

[Dür+99] W. Dür et al. "Quantum repeaters based on entanglement purification". In: *Physical Review A* 59.1 (1999-01), pp. 169–181. DOI: 10.1103/physreva.59.169.

[EPR35] A. Einstein, B. Podolsky, and N. Rosen. “Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?” In: *Phys. Rev.* 47 (10 1935-05), pp. 777–780. DOI: 10.1103/PhysRev.47.777.

[FHW80] Steven Fortune, John Hopcroft, and James Wyllie. “The directed subgraph homeomorphism problem”. In: *Theoretical Computer Science* 10.2 (1980-02), pp. 111–121. DOI: 10.1016/0304-3975(80)90009-2.

[Flo62] Robert W. Floyd. “Algorithm 97: Shortest Path”. In: *Commun. ACM* 5.6 (1962-06), pp. 345–. ISSN: 0001-0782. DOI: 10.1145/367766.368168. URL: http://doi.acm.org/10.1145/367766.368168.

[Fra+13] Fabrizio Frati et al. “Augmenting Graphs to Minimize the Diameter”. English. In: *Algorithms and Computation*. Ed. by Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam. Vol. 8283. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 383–393. ISBN: 978-3-642-45029-7. DOI: 10.1007/978-3-642-45030-3_36.

[Gei+08] Robert Geisberger et al. “Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks”. In: *Experimental Algorithms*. Springer Science and Business Media, 2008, pp. 319–333. DOI: 10.1007/978-3-540-68552-4_24.

[GL98] William I. Gasarch and Andrew C.Y. Lee. “On the finiteness of the recursive chromatic number”. In: *Annals of Pure and Applied Logic* 93.1–3 (1998). Computability Theory, pp. 73–81. DOI: 10.1016/S0168-0072(98)00054-2.

[Gre+90] Daniel M. Greenberger et al. “Bell’s theorem without inequalities”. In: *American Journal of Physics* 58.12 (1990), pp. 1131–1143. DOI: 10.1119/1.16243.

[Gro96] Lov K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*. STOC ’96. Philadelphia, Pennsylvania, USA: ACM, 1996, pp. 212–219. ISBN: 0-89791-785-5. DOI: 10.1145/237814.237866.

[GT10] M.T. Goodrich and R. Tamassia. *Data Structures and Algorithms in Java*. J. Wiley & Sons, 2010. ISBN: 9780470398807.

[GT87] J.L. Gross and T.W. Tucker. *Topological Graph Theory*. Dover books on mathematics. Dover Publications, 1987. ISBN: 9780486417417.

[GYZ13] J.L. Gross, J. Yellen, and P. Zhang. *Handbook of Graph Theory, Second Edition*. Discrete Mathematics and Its Applications. CRC Press, 2013. ISBN: 9781439880197.

[Hor+09] Ryszard Horodecki et al. “Quantum entanglement”. In: *Rev. Mod. Phys.* 81 (2 2009-06), pp. 865–942. DOI: 10.1103/RevModPhys.81.865.

[Kim08] H. J. Kimble. “The quantum internet”. In: *Nature* 453.7198 (2008-06), pp. 1023–1030. DOI: 10.1038/nature07127.

[Kom+14] Peter Komar et al. “A quantum network of clocks”. In: *Nature Physics* (2014). DOI: 10.1038/nphys3000.

[KT06] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN: 0-321-327291-3.

[LC99] Hoi-Kwong Lo and H. F. Chau. “Unconditional Security of Quantum Key Distribution over Arbitrarily Long Distances”. In: *Science* 283.5410 (1999), pp. 2050–2056. DOI: 10.1126/science.283.5410.2050.

[LMS92] Chung-Lun Li, S.Thomas McCormick, and David Simchi-Levi. “On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems”. In: *Operations Research Letters* 11.5 (1992-06), pp. 303–308. DOI: 10.1016/0167-6377(92)90007-p.

[Loo87] Charles Loop. “Smooth subdivision surfaces based on triangles”. Master thesis. Univeristy of Utah, 1987-08.

[Mie06] P. van Mieghem. *Data Communications Networking*. Techne Press, 2006. ISBN: 9789085940081.

[Moe+07] D. L. Moehring et al. “Entanglement of single-atom quantum bits at a distance”. In: *Nature* 449.7158 (2007-09-06), pp. 68–71. DOI: 10.1038/nature06118.

[Mun+15] W.J. Munro et al. “Inside Quantum Repeaters”. In: *Selected Topics in Quantum Electronics, IEEE Journal of* 21.3 (2015-05), pp. 78–90. DOI: 10.1109/JSTQE.2015.2392076.

[NC10] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. 10th Anniversary Edition. Cambridge university press, 2010. ISBN: 9781107002173.

[Pfa+14] W. Pfaff et al. “Unconditional quantum teleportation between distant solid-state quantum bits”. In: *Science* 345.6196 (2014), pp. 532–535. DOI: 10.1126/science.1253512.

[RLH06] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. Tech. rep. 2006-01. DOI: 10.17487/rfc4271.

[SA94] Dietrich Stauffer and Ammon Aharony. *Introduction to percolation theory*. CRC press, 1994. ISBN: 978-0748402533.

[San+11] Nicolas Sangouard et al. “Quantum repeaters based on atomic ensembles and linear optics”. In: *Rev. Mod. Phys.* 83 (1 2011-03), pp. 33–80. DOI: 10.1103/RevModPhys.83.33.

[Sch15] Eddie Schoute. *Spherical Routing*. 2015. URL: https://github.com/Calavoow/spherical-routing (visited on 2015-08-19).

[Sho97] Peter W. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation*. Second Edition. Thomson Course Technology, 2006. DOI: 10.1145/230514.571645.

[SP00] Peter W. Shor and John Preskill. “Simple Proof of Security of the BB84 Quantum Key Distribution Protocol”. In: *Phys. Rev. Lett.* 85 (2 2000-07), pp. 441–444. DOI: 10.1103/PhysRevLett.85.441.

[Weh08] Stephanie Wehner. “Cryptography in a quantum world”. PhD thesis. Plantage Muidergracht 24, 1018 TV Amsterdam: University of Amsterdam, 2008.

[Weh15] Stephanie Wehner. “Quantum computing and communication”. Lecture Notes. 2015.

[Wil13] Mark M Wilde. *Quantum information theory*. Cambridge University Press, 2013. ISBN: 9781107034259.

[WZ82] W. K. Wootters and W. H. Zurek. “A single quantum cannot be cloned”. In: *Nature* 299.5886 (1982-10-28), pp. 802–803. DOI: 10.1038/299802a0.