Optimisation of fusion networks with linear resource states

William Cashman

University of Oxford

A thesis submitted for the degree of

Master of Mathematics and Foundations of Computer Science

Trinity 2024

Acknowledgements

I would like to thank my supervisors for their unwavering support throughout the development of this dissertation, as well as my classmates for many insightful discussions.

Abstract

Fusion-based quantum computing is a new method of implementing measurement-based quantum computing (MBQC) algorithms on photonic hardware. It constructs the highly entangled graph state by fusing together smaller resource states in a scalable, fault tolerant way. Fusions are inherently non-deterministic and so strategies must be employed to ensure failures are tolerated whilst minimising the required resources.

Here we consider the problem of implementing MBQC patterns with the fewest number of fusions using both bounded and unbounded linear cluster states. We focus on the cases of constructing MBQC patterns using solely X fusions, Y fusions, and both fusions separately and provide a mathematical reduction of each case to a graph minimisation and show that all cases are NP-hard except for the case of X fusions with unbounded resource states. We then present approximation algorithms for the NP-hard problems and prove tight bounds on their accuracy. In addition, we introduce a graph rewrite strategy for each case to reduce the number of required fusions. We have implemented our proposed algorithms in software and conclude this work with empirical results and show how our methods outperform existing benchmarks in key areas.

Contents

Chapter 1

Introduction

Recent breakthroughs have made photonic computing a promising platform for scalable quantum computing in the near term [1–3].

It is the natural candidate for executing algorithms formulated in the language of measurement-based quantum computing (MBQC) due to long coherence times and simplicity of performing single qubit rotations and measurements [4, 5]. Developments in photonic technologies have lead to the development of efficient cluster state generators[6, 7] and routers[8] with linear optics which has facilitated the development of full stack photonic architectures[9–11]. In addition, photon computing is unique in its ability to efficiently transmit qubits long distances which could power the quantum internet[12]. Unlike the commonly used quantum circuit model [13], MBQC has no classical counterpart and has been shown to provide a linear speed up for many real-world algorithms such as the Quantum Fourier Transform[14].

Fusion-based quantum computing (FBQC) is new strategy for efficiently executing MBQC patterns executing high fidelity on near-term photonic hardware[15]. Directly constructing the highly entangled graph state of an MBQC pattern is challenging due to the lack of deterministic entangling operations in photonics. FBQC details a method for performing 2-qubit entangling measurements to "fuse" together small, constant size graph states called *resource states* to construct the graph state.

Figure 1.1: *Fusing three star resource states with X fusions to create a larger graph state.*

Previous work has focused primarily on FBQC strategies using *star resource states* [15, 16] as these cluster states offer a large degree of flexibility when constructing the graph and may be efficiently generated by linear optical devices[17, 18]. In this paper, we instead focus on *linear resource states* which are being explored as an option for developing alternative fusion-based architectures which aim to have a lower resource overhead[11]. Quantum emitters are a promising class of devices for reliably constructing linear resource states and have been demonstrated with atomic systems [19, 20], superconducting circuits [21], and quantum dots [22, 23].

There are many ways to implement an MBQC pattern with fusions which have different resource and performance trade-offs. A particular choice of resource states and fusions is called a *fusion network*. Due to the probabilistic nature of fusions, reducing the number of fusions in the fusion network is critical to ensuring the MBQC pattern is

implemented with high probability on limited hardware. In this work, we address the problem of finding fusion networks with minimal fusions.

1.1 Structure and Novel Contributions

Chapter 2 presents the background on measurement-based and fusion-based quantum computing as well as relevant details regarding photonic computing.

Chapter 3 we formulate the problem of compiling MBQC patterns as fusion networks as a graph-theoretic minimisation problem. We define a new graph theoretic structure which we call a *trail cover* (Definition 8) of a graph and show that minimising fusions is equivalent to solving two graph theoretic problems (Theorem 4): that of finding minimum trail covers of a graph, and the problem of transforming the graph to minimise certain properties. The rest of this work considers the cases of X fusions, Y fusions, and both fusions, and the cases where the linear resource states are bounded or unbounded.

Chapter 4 address the problems of finding minimum path covers, trails decompositions, and trail covers as well as their bounded counterparts and prove their complexity classes. Minimum path covers have been studied extensively as well as their bounded counterparts, and so I merely present the most recent results to date. Trail decompositions are known of but not widely studied, in particular, the minimisation problems have never been studied and are first defined here (Definition 15 and Definition 17). As such, except Definition 14 and Theorem 7 (finding a minimum trail decomposition is in P), all other results are my own, including the proof that finding minimum 2-trail decomposition is solvable in polynomial time (Proposition 4), the test for determining whether a trail belongs to some minimum trail decomposition (Proposition 3), and the proof that finding bounded minimum trail decompositions is NP-hard (Theorem 9). Section 4.3 covers the problem of finding minimum trail covers to study linear XY-fusion networks in its most general form. The concept of a trail cover is my own and consequently all subsequent results are my own. This includes a proof that finding the minimum trail cover is NP-hard (Theorem 10), that there is an underlying subclass of minimum trail covers which have identical structure to minimum trail decompositions (Theorem 11). The definition of the bounded counterparts of minimum trail decompositions and minimum trail covers and all results from them are my own.

In Chapter 5, we describe novel approximation algorithms for the new NP-hard problems in the previous chapter. Concretely, for bounded minimum trail decompositions we provide an approximation algorithms (Algorithm 1) and prove tight bounded on its complexity and accuracy (Proposition 7). We additionally give an approximation algorithm for finding minimum trail covers (Algorithm 2) and for bounded minimum trail cover (Algorithm 3) and alternate algorithm based on a reduction to the travelling salesman problem (Proposition 11) which can leverage existing tools to find a solution.

In Chapter 6 we formulate the minimisation problem concerning graph rewrites and present novel graph rewrite strategies to minimise certain properties of the graph to reduce the number of required fusions. We present an algorithm for Y fusions (Algorithm 4) and for X fusions and show how these can be generalised to the case when considering both types of fusions.

In Chapter 7 we discuss the problems associated with implementing the fusion networks on real hardware and introduce photon loss as a potential source of error. We define a new calculus (Definition 22) for expressing computation in this class of architectures

defined in [10] to enable us to define the general optimisation problem for generating fusion networks on this architecture (Definition 24). We then define a concrete architecture (Definition 23) within the general class of architectures which serves as a compilation target and precisely evaluate the error associated with our fusion networks. We then give an approximation algorithm (Algorithm 5) for generating fusion networks with the aim of minimising the total error on this hardware.

We conclude in Chapter 8 with a discussion of empirical benchmarks obtained from the algorithms in previous chapters on random graph states, common error correcting codes, and real-world algorithms.

Appendix A contains — to the best of our knowledge — the first of proof the probability fusion success for arbitrary types of fusion in FBQC protocols.

1.2 Related work

Fused-based quantum computing was originally introduced by PsiQuantum in 2021 [24] and who are now manufacturing photonics chips with their technology at a large scale [25, 26]. This work considers the more general class of resource states with X and Y fusions from which the fusion networks with star resource states in the original proposed can be seen as a special case. Section 2.4.1 discusses this point in more detail and makes this claim precise.

The first proposal for a fusion-based architecture with linear resource states was given by Zilk et al. [27] which used both GHZ and linear resource states together. In contrast, our architecture uses only linear resource states with the aim of achieving a lower resource overhead by using longer resource states and by constraining the optimisation landscape to enable us to find better fusion networks efficiently. Chapter 8 compares benchmarks between the architecture in this work and that in [27]. Most recently, Quandela has proposed an architecture that works exactly as our proposed setup intends [11] and this was later mathematically formalised by De Felice et al. [10] who then outlined a repeat-until-success protocol for boosting the probability of success and mathematical tools for its analysis. This work builds on top of these work of these two papers, namely adopting the architecture originally proposed in [11] and using the theoretical results of [10] to allow us to formulate the problem of finding fusion networks with minimal fusions in the language of graph theory which forms the main contribution of this work.

Lee and Jeong [28] provided two graph rewrites for fusion networks the cases of bipartite subgraphs and complete subgraphs that reduced the total number of fusion required. Their rewrites in particular work in the case of star cluster states but do not always provide an improvement in the case of linear resource states. In this work, we precisely characterise the optimisation problem for minimising fusions as a graph theoretic problem and so we can better determine when certain rewrites will reduce the total number of fusions. This will allow the techniques of [28] to be applied to the more general case of linear cluster states.

Kumabe et al. [29] defined the graph theoretic notion of *CZ-complexity* to be the fewest number of edges in any graph that can be reach from another graph through local complementations or vertex insertions. They proved a lower bound on the CZ-complexity of a graph to be $n + r - 2$ where n is the number of vertices in a graph and r is the rank-width of the graph. In addition, they provided an algorithm for transforming graphs and showed that the number of edges in the resulting graph state is bounded above by

$O(nr \log n)$. Though reducing the number of edges in a graph is the primary heuristic used in our algorithms presented in this work, Theorem 3 demonstrates that this is not the entire picture and that the number of nodes and the number of trails used to cover the graph are equally important for reducing the number of fusions. Furthermore, graphs with the fewest number of edge need not have the smallest trail cover and hence will not lead to optimal fusion networks. We discuss this point more in Chapter 6.

Trail decompositions are not widely studied and are primarily mentioned in the study of Eulerian trails. The most comprehensive background can be found in [30]. The problem of finding a minimum trail decomposition is not formally stated but is proved in Theorem 2.3 of [31].

Chapter 2

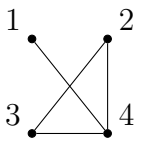
Background

This chapter covers relevant concepts for understanding the framework that FBQC operate. We assume familiarity with quantum computing fundamentals and refer the reader to [13] for a more comprehensive introduction.

2.1 Graph states

Graph states are quantum states comprised solely from preparing qubits in the $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ state and applying controlled-Z gates. They are known as graph states as they may be visualised as graphs where qubits are the vertices and CZ gates are the edges.

$$CZ_{1,4}CZ_{2,3}CZ_{3,4}CZ_{2,4}|+\rangle_{1234} \quad = \quad$$



Similarly, a graph $G = (V, E)$ corresponds to a graph state $|G\rangle$ constructed by preparing a $|+\rangle$ state for each vertex and a controlled-Z gates is applied between two qubits if there is an edge between their respective vertices in G.

$$|G\rangle = \prod_{(x,y)\in E} CZ_{x,y} \prod_{v\in V} |+\rangle_v$$

The connectivity of graph states may be changed through *local complementations*. Locally complementing a graph G about a vertex v is equivalent to toggling the edges between the neighbours of v (See Fig. 2.1). On graph states, local complementations are performed by applying an X rotation with angle $-\frac{\pi}{2}$ to the qubit corresponding to v and a Z rotation with angle $\frac{\pi}{2}$ to the qubits corresponding to the neighbours of v.

In addition to local complementations, we may introduce extra qubits into the graph state which are connected to any nodes provided they are measured in the Z basis and may introduce Pauli errors on the nodes.

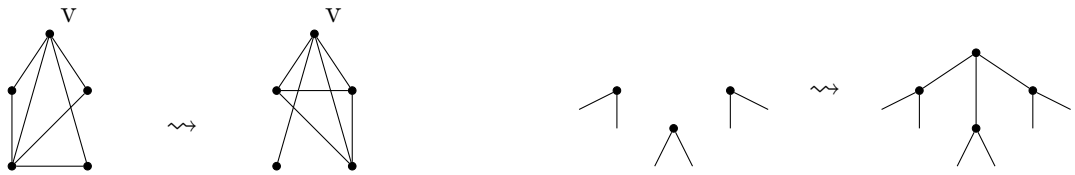


Figure 2.1: *Local complementation (left) of the node v toggles edges between its neighbours. Z-deletion (right) removes a node and all of its edges when it is measured in the Z basis with no phase.*

In fact, all single qubit Clifford operations on graph states correspond to local complementations [32] and so we may consider local complementations together with adding Z-measured nodes as the two fundamental operations with which to transform a graph state into an equivalent state up to local Clifford operations on the qubits. The problem of determining if two graphs can be obtained through local complementations alone is solvable in polynomial time [33]. However if we add the ability to remove nodes through Pauli measurements the problem becomes NP-complete [34].

2.2 Measurement-Based Quantum Computing

Measurement-based quantum computing (MBQC) [35] is a model of quantum computation with no classical analogue and is well suited for photonic architectures. In contrast with the quantum circuit model [13], which closely resembles thee classical circuit model, an MBQC algorithm consists of a highly entangled graph state and a procedure for adaptively measuring qubits to perform the computation. This type of computing is also known as "one-way computation" due to the destructive nature of quantum measurements.

Two qubit entangling operations are inherently non-deterministic in photonics which makes it challenging to reliably execute algorithms formulated in the quantum circuit model. In contrast, MBQC computations only require single qubit rotations and measurements to perform the computation after the initial graph state is constructed, which makes MBQC a more naturally computing paradigm for photonics [4, 5]. MBQC has additionally been shown to provide a linear performance increase over the circuit model for many real-world algorithms such as the Quantum Fourier Transform[14].

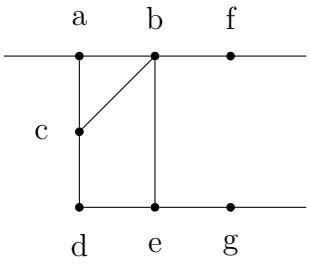
MBQC computations consists simply of a graph state and a procedure for perform single qubit measurements. These are commonly encoded in a single structure known as a *labelled open graph*.

Definition 1. A *labelled open graph* is a tuple $(G, I, O, \lambda, \alpha)$ consisting of a simple undirected graph $G = (V, E)$, a selection of input and output nodes $I, O \subseteq V$, and for all non-output nodes, an assignment of measurement planes $\lambda : V \backslash O \to \{XY, YZ, XZ\}$ and measurement angles $\alpha : V \backslash O \to [0, 2\pi)$.

A measurement performed in the λ plane with angle α has two possible outcomes, $\langle +_{\lambda,\alpha}|$ and $\langle -_{\lambda,\alpha}|$ which are given as

$$\sqrt{2}\langle \pm_{\lambda,\alpha}| = \begin{cases} \langle 0| \pm e^{i\alpha}\langle 1| & \text{if } \lambda = XY \\ \langle +| \pm e^{i\alpha}\langle -| & \text{if } \lambda = YZ \\ \langle i| \pm e^{i\alpha}\langle -i| & \text{if } \lambda = XZ \end{cases}$$

The *target linear map* of a labelled open graph $\mathcal{G} = (G, I, O, \lambda, \alpha)$, written $T(\mathcal{G})$, is the linear map obtained by initialising the graph state $|G\rangle$ and measuring all non-output nodes according to λ and α and post-selecting on the measurement outcome $\langle +_{\lambda,\alpha}|$. Considering the nodes in I and O to be the inputs and outputs respectively gives us the target linear map of the open graph.



$$\begin{array}{ll} \lambda(a) = YZ & \alpha(a) = 0 \\ \lambda(b) = XY & \alpha(b) = \frac{\pi}{2} \\ \lambda(c) = XZ & \alpha(c) = \pi \\ \lambda(d) = YZ & \alpha(d) = \frac{\pi}{3} \\ \lambda(e) = XY & \alpha(e) = \frac{\pi}{2} \end{array} \qquad \begin{array}{l} I = \{a\} \\ O = \{f, g\} \end{array}$$

Figure 2.2: *A labelled open graph given by a simple undirected graph with inputs and outputs together with measurement planes (λ) and measurement angles (α).*

In reality, measuring a qubit may produce different measurement outcomes which introduce Pauli errors into the graph state that may need to be corrected by future measurements. The existence of an adaptive measurement strategy capable of correcting all such Pauli errors on a graph is captured in the notion of *gflow flow* [36, 37].

Theorem 1 ([37])**.** *A labelled open graph may be executed to deterministically implement its target linear map up to local Cliffords if and only if it has gflow.*

The conditions of gflow are not relevant to the contents of this submission and so we will omit its definition and mention it solely as an essential criterion for realising the MBQC computations. In future chapters, we assume the labelled open graphs we wish to execute have gflow and we will merely show that all transformations that we use preserve the flow and thus the result of our pipeline may always deterministically implement the target linear map.

2.3 Photonic computing

Photonic computing uses photons as qubits to perform quantum computations, and is a strong candidate for realising near-term quantum computing due to their scalability, long coherence times and simple integration with quantum networks.

Qubits are usually encode from a pair of bosonic states associated with the photon, known as the *dual rail encoding*. The most common of these is the spatial encoding where the qubit states $|0\rangle$ and $|1\rangle$ correspond to the location of the photon. It is often useful to visualise this dual rail encoding with two lines representing the bosonic modes annotated with the number of photons present in the mode. These diagrams are not to be confused with the graph state diagrams drawn elsewhere.

$$|0\rangle = \begin{matrix} 1 \bullet\!\!-\!\!- \\ 0 \bullet\!\!-\!\!- \end{matrix} \qquad\qquad |1\rangle = \begin{matrix} 0 \bullet\!\!-\!\!- \\ 1 \bullet\!\!-\!\!- \end{matrix}$$

The primary focus of this work centers around linear cluster states which may be visualised as a graph state corresponding to a linear graph.

Figure 2.3: *Linear resource states of length zero, 1, and 4 respectively.*

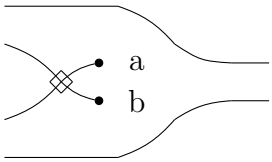
Quantum emitters are a promising class of devices for reliably constructing entangled resource states and have been demonstrated with atomic systems [19, 20], superconducting circuits [21], and quantum dots [22, 23]. These can be idealised as a devices that produces a stream of entangled photons which may optionally apply a single qubit unitary. Each vertex in the graph may contain multiple entangled photons in a GHZ state which are useful for performing multiple fusions on the node or realising repeat-until-success protocols. We will address the point in more detail in Section 2.4.1.

Since quantum emitters are currently produce only a limited number of entangled photons we have also considered the problem of finding fusion networks comprised of linear resource states of bounded length as an equally, if not more important, problem to solve.

Fusions are a class of non-deterministic 2-qubit entangling measurements. At the core of all fusions is the projector

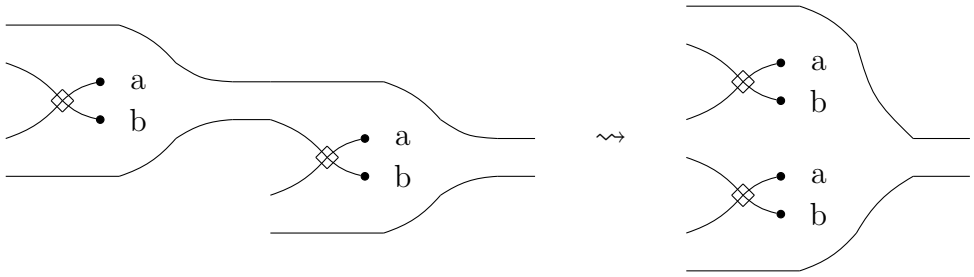
$$|0\rangle\langle 00| + (-1)^k |1\rangle\langle 11| \tag{2.1}$$

which is implemented by the following photonic experiment where $k = \frac{1}{2}(a + b)$ and $a, b \in \{0, 1\}$ and qubits are encoded in the dual rail encoding.



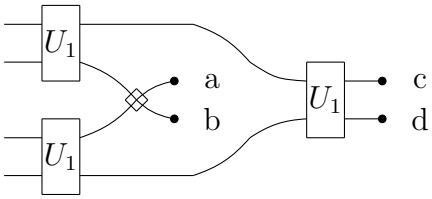
The resulting bosonic states that contain either two or zero photons are outside of the qubit subspace and mark fusion failure. Failure is heralded by the measurement outcomes $a, b \in \{0, 1\}$. Hence given a normalised state $\alpha_1|00\rangle + \alpha_2|01\rangle + \alpha_3|10\rangle + \alpha_4|11\rangle$ as input, fusion will fail with probability $|\alpha_2|^2 + |\alpha_3|^2$.

We may compose this experiment to form fusions between any number of qubits. This type of fusion is known as *n-ary fusion* and in some cases have a higher probability of success than multiple individual fusions.



In this work we only consider 2-ary fusion and leave the analysis with n-ary fusions as future work.

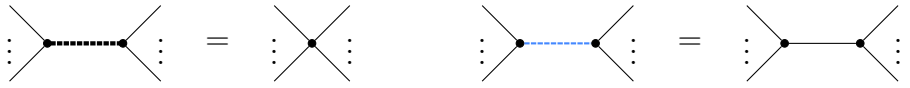
In general we may consider a general fusion to be specified by applying a unitary to each of the qubits followed by the projector (2.1) followed by a single qubit measurement. This can be drawn as the following photonics experiment.



De Felice et al. [10] categorised the class of fusions which preserve the original graph structure on fusion failure as *green fusions*. This class of fusions is important as it makes the compilation tolerant of fusion failures. Fusions of this kind were shown to have probability of exactly 50% of failing during an MBQC computation [10]. By only considering to symmetric fusions with Pauli errors that are constructed with Pauli measurements, we obtain only two distinct types of fusion: *X fusion* and *Y fusion*. These correspond to the two most commonly studied fusions, Type II [24, 27] and CZ fusion [38] respectively.

Definition 2. *X fusion* is when all unitaries U_1, U_2, and U_3 are Hadamard gates.
Y fusion is when $U_1 = U_2 = R_X(\frac{\pi}{2})$ and $U_3 = R_X(\frac{-\pi}{2})$.

X fusion corresponds to the projector $\langle 00| + \langle 11|$ up to a Pauli X error and has the effect of merging to nodes of a graph state into one. Y fusion has the effect of constructing a controlled-Z gate between the two qubits and then measuring in the X basis. On graph states, this corresponds to constructing an edge between two nodes.



X fusion Y fusion

We use the thick dashed line to denote X fusions and the blue dashed line for Y fusions throughout this paper.

2.4 Fusion-based Quantum Computing

Fusion-based quantum computing (FBQC) is new method of efficiently executing MBQC patterns on near-term photonic hardware[15]. Directly constructing the highly entangled graph state of an MBQC pattern is error prone due to the lack of deterministic entangling operations in photonics and so FBQC protocols construct the graph state by *fusing* small cluster states together with 2-qubit measurements in an way that is tolerant of the non-deterministic nature of the fusions.

This presents an additional challenge since fusions fail with a probability of 50% (see Theorem 2) and so FBQC protocols must be able to tolerant fusion failures. The probability of fusion success may be boosted to 75% with ancillary qubits [39] but most protocol feature some method to tolerant failures such as repeat-until-success protocols [10] or ballistic approaches [40].

Small, efficiently generated cluster states are the building blocks of any FBQC compilation strategy and are called *resource states*. The most commonly used resource state is the *star cluster state* [15, 16] as it offers a large degree of flexibility when constructing the graph and generating such state have already been extensively studied [17, 18]. Here we instead focus on the less studied *linear cluster states* which are produced efficiently by

quantum emitters. Architectures using these resource states may required fewer fusion and in fact subsume the case of star resource states as we show in Section 2.4.1.

Figure 2.4: *(Left) A 5-vertex star cluster state. (Right) A 4-vertex linear resource state.*

Whether a graph may be more efficiently implemented with X or Y fusions depends on its structure. In general, parts of a graph that are highly connected are more efficiently implemented with X fusions, and less connected parts are more efficiently implemented with Y fusions. Thus by using both kinds of fusions, we may be able to construct graph states in fewer fusions that if we were only allowed one type.

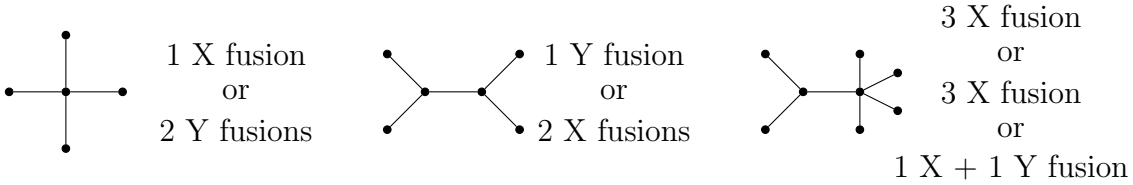


Figure 2.5: *Example of case when a graph may be implemented in fewer X fusions than Y fusions (left), fewer Y fusions than X fusions (middle), and fewer hybrid fusions than either type of fusion alone (right)*

In order to define an FBQC computation, we must specify the resource states and the fusions between them together with the measurement strategy on the graph. This is information is encoded in the notion of a *fusion network*.

Definition 3 (Fusion Network [10])**.** An XY-fusion network $\mathcal{F} = (G, I, O, X, Y, \lambda, \alpha)$ is given by the following:

1. a labelled open graph $(G, I, O, \lambda, \alpha)$ where $G = (V, E)$, and
2. sets of unordered pairs $X, Y \subseteq \{\{a, b\} \mid a, b \in V\}$ corresponding to X fusions and Y fusions on the nodes respectively.

We define *X-fusion networks* and *Y-fusion networks* to be XY-fusion networks comprised solely of X fusion and Y fusions respectively.

Since X fusions correspond to joining two vertices together in the underlying graph state, and Y fusions correspond to constructing an edge we can make the following definition.

We say that an XY-fusion network $\mathcal{F} = (G, I, O, X, Y, \lambda, \alpha)$ *implements the labelled open graph* $(G', I, O, \lambda, \alpha)$ where G' is defined as

$$G' = \frac{(V, E + Y)}{a \sim b \text{ if } \{a, b\} \in X} \tag{2.2}$$

where $+$ means the disjoint union. Informally, G' is created by turning Y fusions into edges, and my merging vertices that belong to X fusions.

The *target linear map* $T(\mathcal{F})$ of the fusion pattern $\mathcal{F}$ is the target linear map of labelled open graph implemented by $\mathcal{G}$. Implicitly, this is like post-selecting on the fusion success outcome with no induced Pauli errors.

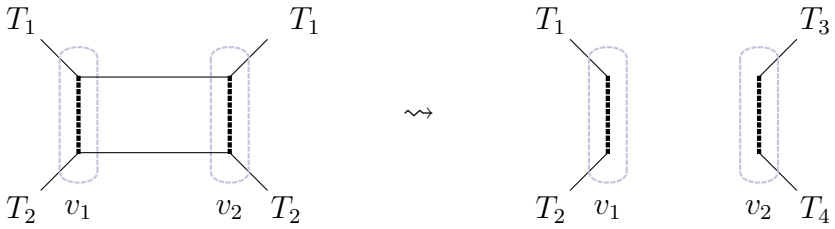
Despite introducing additional Pauli errors into the graph, De Felice et al. [10] showed that any fusion network that implements a labelled open graph with gflow is able to do so deterministically. Furthermore, it was also shown that no corrections need be performed at the fusions, which implies fusions may be performed any time before it's nodes are measured.

Since in our case we are focusing a on using linear resource states we define a special type of fusion network known as a *linear XY-fusion network*.

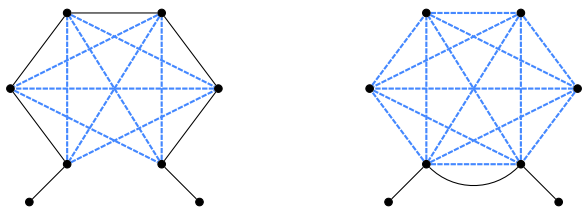
Definition 4. A *linear XY-fusion network* $\mathcal{F} = (G, I, O, X, Y, \lambda, \alpha)$ is an XY-fusion network where G is the disjoint union of linear graphs.

For the rest of this paper we will only consider simple undirected graphs because two edges in a graph correspond to two controlled-Z gates in the graph state which cancel. In fact, we can show that any linear XY-fusion network that implements a labelled open graph whose graph has a double edge, can be converted to an equivalent fusion network that implements a simple graph in the same number of fusions.

To see this, first note that if both edges in the double edge are created by Y fusions, then these fusions are unnecessary since removing both fusion results in the same graph state with fewer fusions. If one edge is created by a linear cluster state and one is a Y fusion. Then we could have split the linear cluster state into two pieces to avoid creating the edge in the first place, and omitting the Y fusion to obtain the same graph state with fewer fusions. If both edges are from linear cluster states, then it must be the case that the vertices at either edge of the edge must be merged by X fusions. In this case, we can again split the linear cluster states into multiple pieces to avoid any edge from being created in the first place and the number of fusions will remain the same. This is illustrated below with linear resource states T_1 and T_2 which are split to become T_1', T_2', T_3', T_4'.



We remark that this result does not hold in the case of n-ary Y fusions. An example is shown below that requires one 6-ary which has probability 2^{-5} of success, compared to using nine normal Y fusions which have a probability of 2^{-9} of success.



Five 9-ary Y fusion One 6-ary fusion

We remark that n-ary fusions are a promising avenue of research as it greatly improves the probability of success is certain cases, though we will leave it as future work.

2.4.1 Resource states for FBQC

Resource states are smaller, efficiently generated graph states that form the building blocks of fusion networks. The choice of resource state and how they are generated comes with trade-offs. We have already introduced star cluster states and linear cluster states at the beginning of this chapter, and so we will now discuss certain implementation details which affect their performance and show how the structure of fusion networks constructed from star resource states is mirrored by fusion networks with linear resource states of length zero.

Since fusions are two-qubit measurements, both photons are destroyed after the fusion. Therefore if a node in the graph state is implemented by a single photon then it would not be possible to perform any further operations on the node. This presents a significant problem as some nodes are involved in many fusions and all non-output nodes in a labelled open graph must be measured. To overcome this limitation, we can assume that every node in the graph state consists of an arbitrary number of photons entangled in a GHZ state. In this model, if we post-select on fusion success, we can perform an arbitrary number of fusions on each node as well as a final measurement.

Indeed quantum emitters are able to implement this resource state by applying a unitary operator on consecutive photons in the stream and is the foundation for the architecture proposed by Quandela in [11] and mathematically formalised in [10]. Since the photons emitted by the quantum emitter are originally in entangled in a GHZ state, applying no unitary operator will keep them in the GHZ state, and applying a CZ gate between them will create a graph state with the two photons belonging to separate nodes joined by an edge. Thus give a perfect quantum emitter capable of producing entangled streams of photons of arbitrary length, we can assume any node in the linear resource state is implemented with a GHZ state of any size. Therefore for the rest of this work we assume that fusions are *non-destructive*, meaning that we can perform as many fusions as we like before the final single qubit measurement. We will return to this detail in Chapter 8 where we discuss the number of photons required to implement a certain MBQC pattern with high probability with the fusion networks we generate.

This is in contrast to the most commonly studied fusion networks which use star resource states together with destructive X fusions. We will now show that these fusion networks may be modelled by linear XY-fusion network with resource states of length zero. To see this, note that there are three ways to perform X fusions on star resource states.

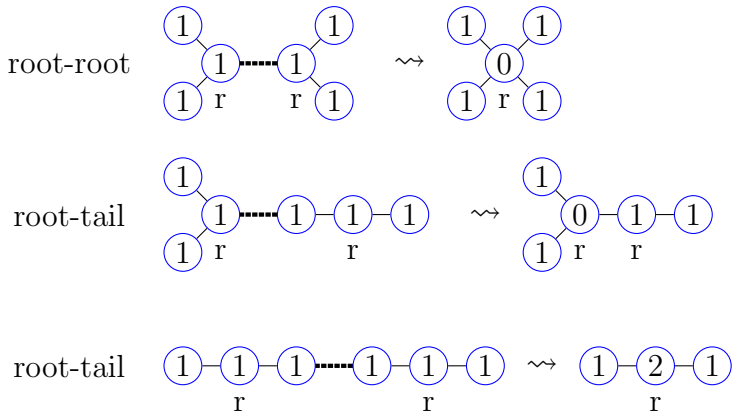


Figure 2.6: *The three distinct types of X fusion (Type II fusion) that can be performed on star resource states. Each circle is a node in the graph state which is annotated with the number of photons contained in it.*

The root-root fusion has the effect of entangling the two root nodes though the new root does not has any photons. The root-tail fusion has the effect of constructing a CZ between the two root nodes, and the tail-tail has the effect of merging the two root nodes together to create a new root node with two photons.

Looking at the actions that these fusions have on the root nodes, we see that tail-tail fusion merges root nodes, and root-tail fusion constructs an edge between them. This is therefore identical to performing X and Y fusion on the root nodes respectively. Thus any fusion network built with star resource states and destructive X fusions can be translated into a linear XY-fusion network with non-destructive fusions where each star state corresponds to a linear cluster state and root-root and tail-tail fusions correspond to X fusions, and root-tail fusions correspond to Y fusion. We note however that additional fusions may be required for the star resource states to replenish photons in the root node or the legs of the root node to facilitate further fusions.

Chapter 3

Problem formulation

Fusions are an inherently non-deterministic process which significantly affect the probability of successfully implementing an MBQC pattern on fusion-based architectures. It is therefore critical to find fusion networks that require minimal number of fusions. In this chapter we formalise this problem mathematically and prove its equivalence to a graph minimisations problem which will be analysed in more detail in the rest of this work.

Though the focus of this work centers around reducing fusions, Chapter 7 discusses the implementation of these fusion networks on real photonic hardware where we discuss the other major source of error: photon loss. We will then show how the design of the fusion network significantly impacts the probability of photon loss and propose alternate strategies for generating fusion networks which reach a trade-off between the number of fusions and the likelihood of photon loss. This discussion has been kept isolated in its own chapter and so we will for now focus solely on the problem of minimising fusions.

3.1 Minimising fusions

Fusions are a probabilistic process which fail with probability 50% when applied in fusion networks.

Theorem 2. *The probability of X and Y fusion success when executing on a fusion network is 50%.*

Proof. In Appendix A. □

In the case of quantum emitters, the probability of fusion success may be boosted with repeat-until-success protocols [10] and with ancillary qubits [39]. Since there are many possible fusion networks which implement a labelled open graph, it is natural to ask how we can find fusion networks that use the fewest number of fusions possible.

Assuming access to linear resource states of unbounded length and non-destructive fusions simplifies the analysis in later chapter, however we are not guaranteed such a resource state in practice. At the time of writing, quantum emitters have so far only been able to generate linear resource states with 14 photons [20]. Hence, even though we can expect the performance of quantum emitters to improve over time, it is likely that the number of photons will significantly constraint the length of the resource states we can use in practice.

It is therefore important to consider the case where the resource states are implemented with a bounded number of photons. Post-selecting of fusion success, in order

to facilitate F fusions and one single qubit measurement on a node in the graph state, the node must be implemented by a GHZ state with $F + 1$ photons. We then consider three variants of the fusion network minimisation problem, those are the cases where the resource states are: unbounded, bounded by the number of photons, bounded in the number of nodes. The third variant, though it does not accurately reflect the true constraints of a physical setup, it much more easily analysed in graph theory and we will show that complexity results and approximation algorithms that solve this problem may be adapted to the case with bounded photons.

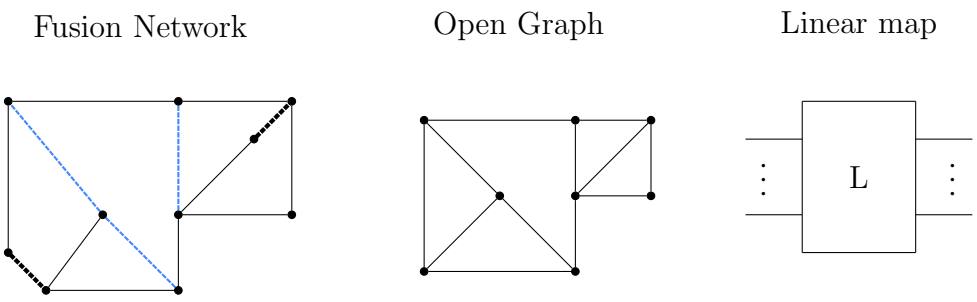
Definition 5. `MinXYFusionNetwork`
Input: A graph G.
Output: A linear XY-fusion network $\mathcal{F}$ which implements G using the fewest total number of fusions.

Definition 6. `MinBoundedPhotonXYFusionNetwork`
Input: A graph G and a positive integer L.
Output: A linear XY-fusion network $\mathcal{F}$ which implements G where every resource state of $\mathcal{F}$ contains at most L photons, and uses the fewest total number of fusions.

Definition 7. `MinBoundedXYFusionNetwork`
Input: A graph G and a positive integer L.
Output: A linear XY-fusion network $\mathcal{F}$ which implements G where every resource state of $\mathcal{F}$ contains at most L edges, and uses the fewest total number of fusions.

3.2 Linear XY-fusion networks as trail covers on graphs

There are three primary levels which we can see the computation: fusion networks, labelled open graphs (with gflow), and linear maps. The goal of an MBQC computation is to implement a certain linear map and to do so we need to specify a labelled open graph. In this section we will show that the set of all labelled open graph that implement a target linear map can be generated from a single labelled open graph through local complementations and Z-deletions. We then show that the set of all linear XY-fusion networks that implement a given labelled open graph $(G, I, O, \lambda, \alpha)$ corresponds to the set of trail covers on the graph G. We then conclude by equating the problem of finding fusion networks with minimum fusions to be one of transforming graphs and finding minimum trail covers that minimise a certain graph theoretic property.



It has been shown that graph rewrites are sufficient to show any two open graphs implement the same linear operator.

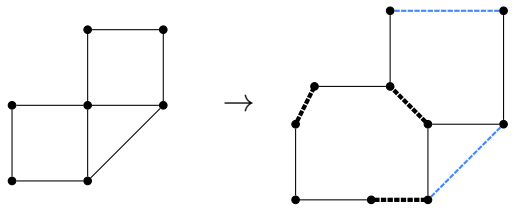
Lemma 1 (Backens and McElvanney [41])**.** *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two labelled open graphs with flow. Then $T(\mathcal{G}_1) = T(\mathcal{G}_2)$ if and only if $\mathcal{G}_1$ can be reached from $\mathcal{G}_2$ using local complementation and Z-deletions.*

This tells use the scope of graph transformations possible and to complete the graph theoretic reduction of fusion network we need to characterise all fusion networks that implement a specific labelled open graph.

Definition 8 (Trail cover)**.** A *trail* is a sequence of vertices where adjacent vertices are connected by an edge and where edges may not be repeated. The length of a trail T, denoted $|T|$ is the number of edges in the trail. A *trail cover* of a graph G is a set of edge disjoint trails that traverse each vertex of G at least once.

A trail can also be seen as a path that may revisit vertices by not edges. A trail that traverses the entire graph is called an *Eulerian trail* and an Eulerian trail that ends at the same vertex it started from is called an *Eulerian circuit*. A trail that has at most L edges is known as an *L-trail* and an *L-trail cover* is a trail cover consisting of L-trails.

Observe how a target graph below can be implemented with the following fusion network. This fusion network can be seen as a trail cover of G where each trails corresponds to a linear resource state where for every vertex that is traversed by more than one trail we use an X fusion to merge the two nodes, and we add a Y fusion for every edge in the graph that is not present in the trail cover.



Proposition 1. *Let $\mathcal{G} = (G, I, O, \lambda, \alpha)$ be a labelled open graph. Then linear XY-fusion networks that implement $\mathcal{G}$ are in one-to-one correspondence with trail covers of G.*

Proof. Suppose we have a linear XY-fusion network $\mathcal{F}$ that implements $\mathcal{G}$. Then by (2.2) we know that $\mathcal{F}$ must be of the form $\mathcal{F} = (G', I, O, X, Y, \lambda, \alpha)$ with $G' = (V, E)$ and where

$$G = \frac{(V, E + Y)}{a \sim b \text{ if } \{a, b\} \in X}. \tag{3.1}$$

Since $\mathcal{F}$ is a linear XY-fusion network, G' is the union of linear graphs $G = \bigcup_i G_i$ where $G_i \subseteq G$ is linear and each G_i is mutually disjoint. Then for any G_i, each vertex in G_i is mapped to a vertex in G under the transformation in (3.1) and vertices are adjacent in G_i if and only if they are adjacent in G. Since G is simple, the image of G_i corresponds to a trail in G. Since the transformation is surjective on the vertices of G, the image of G' is a trail cover in G.

Now suppose we have a trail cover $\mathcal{C}$ for G. Then we can construct a fusion network $\mathcal{F}$ consisting of a linear graph for each trail in $\mathcal{C}$, thus we can simply create the graph G' to be the disjoint union of the trails in $\mathcal{C}$. Denote the function that carries vertices in G' to vertices in G by f. Every time a trail traverses a vertex that has already been traversed by another trail, we add an X fusion between the two nodes. We then create the remaining edges of the graph with Y fusions. We can then create the measurement planes λ' and

α' for the fusion network as follows. For each $v \in G$, let $f^{-1}(v)$ be pre-image of v under the transformation. We then choose one vertex in the pre-image $v_0 \in f^{-1}(v)$ and define $\lambda'(v) = \lambda(v)$ and $\alpha'(v) = \alpha(v)$. For the remaining vertices in the pre-image $w \in f^{-1}(v)$ we define $\lambda'(w) = YZ$ and $\alpha'(w) = 0$. Then fusion network $(G', I, O, X, Y, \lambda', \alpha')$ implements $\mathcal{G}$. □

Then from Proposition 1 and Lemma 1 we obtain the full characterisation of all linear XY-fusion networks in graph theoretic terms.

Proposition 2. *Let $\mathcal{G} = (G, I, O, \lambda, \alpha)$ be a labelled open graph. Then every linear XY-fusion network $\mathcal{F}$ where $T(\mathcal{F}) = T(\mathcal{G})$ is in one-to-one correspondence with pairs $(G', \mathcal{C})$ where G' is a graph reachable from G using local complementations and Z-deletions, and $\mathcal{C}$ is a trail cover of G'.*

From Proposition 2, since every graph has many possible trail covers, there are many distinct fusion networks that implement it. Which fusion network has the highest probability of success implementing the MBQC on real hardware will depend on the hardware itself. There are several likely sources of error such as: holding many active qubits, photon coherence times, chance of photon loss in delay lines. Common to all hardware however, is the probability of fusion failure. We will restrict ourselves the problem of generating fusion networks with the smallest number of fusions.

Theorem 3. *Given a graph $G = (V, E)$ and trail cover $\mathcal{C}$. Let X_G and Y_G denote the number of X and Y fusions required to implement G with $\mathcal{C}$ respectively. Then*

$$X_G + Y_G = |E| - |V| + |\mathcal{C}|.$$

Proof. Let $T \in \mathcal{C}$ be a trail and let $E(T)$ and $V(T)$ denote the number of edges in T and $V(T)$ denote the number of vertices in T. Then $V(T) = E(T) + 1$ and summing over all trails gives

$$\sum_{T \in \mathcal{C}} V(T) = \sum_{T \in \mathcal{C}} (E(T) + 1) = \sum_{T \in \mathcal{C}} E(T) + |\mathcal{C}|. \tag{3.2}$$

Y fusions are required for edges that are not contained in the trail cover, so $\sum_{T \in \mathcal{C}} E(T) = |E| - F_Y$. Furthermore, each X fusion reduces the number of vertices by one, and so $\sum_{T \in \mathcal{C}} V(T) = |V| + F_X$. Substituting into (3.2) gives

$$F_X + F_Y = |E| - |V| + |\mathcal{C}|.$$

□

We can now state the main graph minimisation problems and show their equivalence to the original fusion network minimisation problems.

Definition 9. `MinEquivTrailCover`
Input: A graph G.
Output: A trail cover $\mathcal{C}$ of graph $G' = (V, E)$ reachable from G using local complementations and vertex insertions such that $|E| - |V| + |C|$ is minimised.

Definition 10. `MinEquivBoundedTrailCover`
Input: A graph G and a positive integer L.
Output: An L-trail cover $\mathcal{C}$ of graph $G' = (V, E)$ reachable from G using local complementations and vertex insertions such that $|E| - |V| + |C|$ is minimised.

Applying Theorem 3 to Proposition 2 proves the equivalence.

Theorem 4. `MinXY(Bounded)FusionNetwork` *is equivalent to* `MinEquiv(Bounded)TrailCover`.

We have not precisely formulated the `MinBoundedPhotonXYFusionNetwork` problem as a graph minimisation problem as it does not offer a natural representation in graph theory. However, in future chapters we will point out how complexity results and approximation algorithms can be carried over to the bounded photons case which may be of some practical utility.

We can now see that these problems have two components, namely: transforming the graph through local complementations and vertex insertions, and finding minimum trail covers. We will analyse the problem of finding minimum trail covers in Chapter 4 and Chapter 5, and the problem of reducing the graph in Chapter 6. We then present a benchmarks analysing the results of these algorithms for generating efficient fusion network for real-world algorithms in Chapter 8.

Chapter 4

Finding Trail Covers

We know from Theorem 3 that finding a linear XY-fusion network with minimum fusions requires finding a minimum trail cover of a graph. In this section, we first analyse the corresponding problems for the restricted cases of linear X-fusions networks and linear Y-fusion networks before presenting the general case for linear XY-fusion networks. All graphs in this section are simple and undirected unless stated otherwise.

4.1 Y Fusions

Without X fusion to join vertices together, every node in a linear Y-fusion network corresponds to a unique node in the target graph, and any edges that are not implemented by the resource states are implemented with Y fusions. Thus the linear resource states correspond to paths in the target graph, and so the trail cover constitutes a *path cover* of the graph.

Definition 11 (Path cover)**.** A *path* in a graph is a sequence of vertices such that adjacent vertices in the sequence are adjacent in the graph, and vertices are not repeated. A *path cover* of G is a set of vertex-disjoint paths where every vertex in G is contained in exactly one path in the cover.

Therefore in order to find linear Y-fusion networks with minimal fusions, we must be able to find *minimum path covers* of a graph.

Definition 12. `MinPathCover`
Input: A graph G.
Output: A path cover of G with the fewest number of paths.

A path of length at most L is called an *L-path* and an *L-path cover* is a path cover comprised of L-paths. The bounded variant of the minimum path cover problem is the *minimum L-path problem*.

Definition 13. `MinBoundedPathCover`
Input: A graph G an integer L.
Output: An L-path cover of G with the fewest number of paths.

The minimum path cover problem is NP-hard since the existence of a path cover containing a single path indicates that the graph has a Hamiltonian path, and deciding whether a graph has a Hamiltonian path is NP-complete [42]. Given a minimum path

cover, we cannot verify whether it is a valid solution without solving the minimisation problem and so `MinPathCover` is NP-hard. Classes of graphs for which the Hamiltonian path problem is known to be polynomial are cataloged [43] and may admit efficient algorithms for finding minimum path covers.

Moran et al.[44] introduced a $\frac{1}{2}$-approximation algorithm for finding path covers on weighted graphs where the total weight is maximised. This is equivalent to the minimum path cover problem when the weight of all edges is one.

Hence finding a minimum L-path cover is also NP-hard for $L > 1$. In the case where $L = 1$, the problem reduces to the maximum matching problem which is solvable in polynomial time [45]. Kobayashi et al.[46] generalised this problem to consider path covers where each path is associated with a weight based on its length. Setting the weight to be one regardless of its length gives us the bounded minimum path cover problem. They then showed that the bounded minimum path cover is solvable in polynomial time for graphs with bounded tree width.

Theorem 5 ([46])**.** *Let $G = (V, E)$ be an undirected graph with bounded treewidth at most W. Then the minimum L-path cover problem can be solved in time $O(2^{2W}W^{2W+2}(L + 2)^{2W+2}|V|)$.*

We present no new results on path cover problems and instead provide graph rewrites in Chapter 6 to reduce the number of Y fusion and the size of the minimum path covers.

4.2 X Fusions

In the case of linear X-fusion networks, resource states can traverse the same node in the graph twice by using an X fusion to merge two vertices together. Therefore linear resource states correspond to trails in the graph. Since we do not have Y fusions to construct edges, every edge in the graph state must be implemented by a resource state. This leads us to the notion of a *trail decomposition* of a graph.

Definition 14 (Trail Decomposition)**.** A *trail decomposition* of a graph is a set of edge disjoint trails that together contain every edge of the graph.

Trail decompositions are in one-to-one correspondence with linear X-fusion networks that implement the graph and by Theorem 3, fusions are minimised when the fusion network corresponds to a minimum trail decomposition.

4.2.1 Minimum trail decompositions

We can now give the corresponding minimisation problem for the case of linear X-fusion networks.

Definition 15. `MinTrailDecomposition`
Input: A graph G.
Output: A trail decomposition of G with the fewest number of trails.

Fortunately, there is an efficient algorithm finding a minimum trail decomposition by connecting vertices of odd degree and finding an Eulerian circuit of the resulting graph.

Lemma 2 (Euler [47])**.** *A connected graph has an Eulerian circuit if and only if every vertex in the graph is even.*

Definition 16. Let $G = (V, E)$ be a graph. Then $\texttt{Odd}(G) \subseteq V$ is the set of all vertices of G that have odd degree. We say such vertices are *odd* and all other vertices are *even*.

Theorem 6 (Theorem 2.3 [31])**.** *Let G be a connected graph. Then there exists a minimum trail decomposition of G that has $\frac{1}{2}|\texttt{Odd}(G)|$ trails if $|\texttt{Odd}(G)| > 0$ and a single trail otherwise.*

Proof. This is certainly a lower bound for the number of trails in a decomposition since any trail decomposition of G must have at least one trail end at every odd vertex. Thus a minimum trail decomposition has at least $\frac{1}{2}|\texttt{Odd}(G)|$ trails if $|\texttt{Odd}(G)| > 0$ and a single trail otherwise.

We can construct a decomposition that reaches this bound by connecting all odd vertices in G with edges in any arrangement to create a graph with only even vertices. Then by Lemma 2 we can find an Eulerian circuit of the modified graph and remove the introduced edges from the circuit to create $\frac{1}{2}|\texttt{Odd}(G)|$ trails if $|\texttt{Odd}(G)| > 0$ and one trail otherwise. □

Since there are efficient algorithms for finding an Eulerian circuit in time $O(|E|)$ such as Hierholzer's algorithm [30], we can conclude that the minimum trail decomposition can be found in polynomial time.

Theorem 7. `MinTrailDecomposition` *is in P.*

Minimum trail decompositions admit a particularly nice structure that allows us to test whether a trail belongs to some minimum trail decomposition efficiently. We will use this test extensively when proving future results and when devising heuristic algorithms for related NP-hard problems.

Theorem 6 may be generalised to disconnected graphs to obtain the following lemma.

Lemma 3. *Let G be a possibly disconnected graph. Then the minimum trail decomposition of G contains at least $\frac{1}{2}|\texttt{Odd}(G)|$ trails with equality if and only if every connected component of G has a non-zero odd vertices.*

We may now state the necessary criteria for a trail to belong to a minimum trail decomposition.

Proposition 3. *Let G be a connected graph with non-zero odd vertices. A trail T in G belongs to a minimum trail decomposition of G if and only if T begins and ends at distinct odd vertices and every connected component of $G \backslash T$ has non-zero odd vertices.*

Proof. Suppose T is a trail in a minimum trail decomposition $\mathcal{T}$ of G. Then $\mathcal{T} \backslash T$ must be a minimum trail decomposition for $G \backslash T$. Therefore by Theorem 6

$$|\texttt{Odd}(G \backslash T)| = 2|\mathcal{T} \backslash T| = 2(|\mathcal{T}| - 1) = |\texttt{Odd}(G)| - 2.$$

When removing T from G, the number of odd vertices can only decrease if T ends at an odd vertex. Hence if removing T from the graph decreases the number of odd vertices by two, T must end at distinct odd vertices. Lemma 3 also tells us that every connected component of $G \backslash T$ has non-zero odd vertices.

For the other direction, assume the trail T in G ends at odd vertices and that every connected component of $G \backslash T$ has non-zero odd vertices, and hence $|\texttt{Odd}(G \backslash T)| = |\texttt{Odd}(G)| - 2$ by Lemma 3. Since every connected component of $G \backslash T$ has non-zero odd

vertices, there exists a minimum trail decomposition $\mathcal{T}'$ of size $\frac{1}{2}|\texttt{Odd}(G\backslash T)|$. Then the trail decomposition $\mathcal{T} = \mathcal{T}' \cup \{T\}$ of G has size

$$|\mathcal{T}| = |\mathcal{T}'| + 1 = \frac{1}{2}(|\texttt{Odd}(G\backslash T)|) + 1 = \frac{1}{2}(|\texttt{Odd}(G)| - 2) + 1 = \frac{1}{2}|\texttt{Odd}(G)|.$$

This is the minimum from Theorem 6 and therefore T belongs to a minimum trail decomposition. □

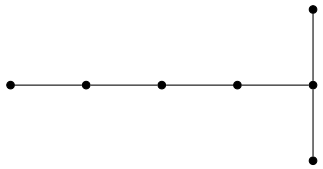
Remark 1. This also implies that in a graph with non-zero odd vertices, the minimum trail decomposition does not contain any closed trails.

4.2.2 The Bounded Minimum trail decomposition problem is NP-hard

We now consider the case where trails can have at most $L \geq 1$ edges. We call such trails, *L-trails*, their decompositions *L-trail decomposition* and their corresponding optimisation problem the *minimum L-trail decomposition problem*.

Definition 17. `MinBoundedTrailDecomposition`
Input: A graph G and positive integer L.
Output: An L-trail decomposition of G with the fewest number of trails.

One might suspect that there is always an L-trail decomposition of size $|E|/L$ or $\frac{1}{2}|\texttt{Odd}(G)|$, but this is not always the case as in the example below.



which for $L = 3$ has $|E|/3 = 2$, $\frac{1}{2}|\texttt{Odd}(G)| = 2$ but the minimum 3-trail decomposition has 3 trails.

The minimum 1-trail decomposition is simply the edge set E. There also exists an efficient algorithm for computing a minimum 2-trail decomposition by converting it to a matching problem.

Proposition 4. *Let $G = (V, E)$ be a graph. Then the minimum 2-trail decomposition of G contains $\left\lceil \frac{1}{2}|E| \right\rceil$ trails and can be found in polynomial time.*

Proof. Observe that 2-trails in G correspond to matchings on the line graph $L(G)$ of G. A 2-trail decomposition in G is minimal when it contains the most number of trails of length 2 of any 2-trail decomposition. This therefore corresponds to a maximum matching on $L(G)$. Since line graphs are connected and claw free, $L(G)$ has a perfect matching if the line graph has an even number of vertices [48], which corresponds to the case where G has an even number of edges. In this case, the line graph has $\frac{1}{2}|E|$ matches and thus there exists a 2-trail decomposition with $\frac{1}{2}|E|$ edges.

If G has an odd number of edges, we could remove a non-bridge edge and obtain a graph with an even number of edges and find a perfect matching of size $\frac{1}{2}(|E| - 1)$. Taking each match to be a 2-trail on the original graph and implementing the removed

edge with a 1-trail, gives us a minimum 2-trail decomposition of size $\frac{1}{2}(|E|-1)$. Thus in general, we can find a minimum 2-trail decomposition of size $\left\lceil \frac{1}{2}|E| \right\rceil$ which is minimal due to being comprised of the most number of 2-trails possible.

Maximal cardinality matchings can be found in polynomial time [45] and therefore minimum 2-trail decompositions can also be found in polynomial time. □

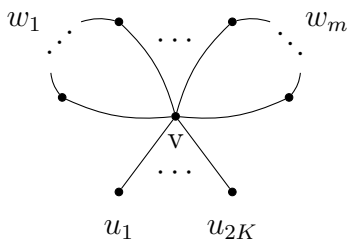
The situation is not as simple when $L \geq 3$ and is in fact NP-hard. To show this, we first proof the corresponding decision problem is NP-complete.

Theorem 8. *Given a graph $G = (V, E)$, determining whether there exists an L-trail decomposition of G of size $K \geq 1$ is NP-complete.*

Proof. First note that given a solution, we can verify whether it is a valid L-trail decomposition with K trails in polynomial time, so the problem is in NP. The rest of the proof follows constructing a polynomial reduction from the bin packing problem which is known to be NP-complete.

The bin packing problem can be stated as follows: given a set of n items with integer weights $(w_i)_{i=1}^n$ where $w_i > 1$ for all i, and positive integers C and K. Determine whether there exists a partition of the items into K disjoint sets $p_1, \ldots, p_K$ such that the total weight of all items in each partition is at most C, that is, $\sum_{j \in p_k} j \leq C$ for all $1 \leq k \leq K$.

We begin by constructing a graph G with $2K$ vertices $\{u_i\}_{i=1}^{2K}$, all connected to another vertex v. Then for each item j, construct a loop comprised of w_j edges which starts and ends at v. Since all weights are greater than one, there are no self loops and so the graph is simple.



We claim that solutions to the original bin packing problem are in one-to-one correspondence with $(C+2)$-trail decompositions of size K for G.

Suppose we have a $(C+2)$-trail decomposition $\mathcal{T}$ of G of size K. First observe that each trail in $\mathcal{T}$ must start and end at one of the vertices in $\{u_i\}_{i=1}^{2K}$. Therefore every trail either fully traverses a particular loop or doesn't traverse any edge in the loop. Since the trail has length at most $C+2$, subtracting the first and last edge of the trail between v and its endpoints in $\{u_i\}_{i=1}^{2K}$, the sum of the edges of the loops it traverses must not exceed C.

Each trail therefore corresponds to a partition of the items, namely the items associated with the loops it traverses that solves the original bin-packing problem. Conversely, it is straightforward to see that any partition of the items can be converted into a $(C+2)$-trail decomposition of the graph by mapping each partition to a trail that begins and ends at one of the vertices in $\{u_i\}_{i=1}^{2K}$, and traverses every loop associated with the items in the partition.

Therefore since the bin packing problem is NP-complete, the L-trail decomposition decision problem is also NP-complete. □

Given that the decision problem is NP-complete, we can infer that the corresponding minimisation problem is NP-hard.

Theorem 9. `MinBoundedTrailDecomposition` *is NP-hard.*

Approximating the bin packing problem with ratio smaller than $\frac{3}{2}$ is NP-hard[49]. This therefore equally applies to the minimum L-decomposition problem as well, though we will show in Chapter 5 that in general, the accuracy of approximation algorithms depends on the number of odd vertices and offer a heuristic algorithm that returns an L-trail decomposition containing no more than $\frac{1}{4}|\texttt{Odd}(G)|$ trail on average.

4.3 XY Fusions

Chapter 3 already outlined the equivalence of minimising fusions in linear XY-fusion networks and finding minimum trail covers. In this section, we will use the results from our discussion of path covers and trail decompositions to prove the complexity of finding minimum trail covers and develop the theory that underpins the approximation algorithms in Chapter 6.

4.3.1 Minimum trail covers

Recalling the definition of a trail cover from Definition 8, we can precisely state the general minimisation problem for trail cover and its bounded counterpart.

Definition 18. `MinTrailCover`
Input: A graph G.
Output: A trail cover of G with the fewest number of trails.

Definition 19. `MinBoundedTrailCover`
Input: A graph G an integer L.
Output: An L-trail cover of G with the fewest number of trails.

To see how trail covers may be smaller than both trail decompositions and path covers, consider that if we can find two adjacent odd vertices where removing their common edge does not break the graph into connect components with non-zero odd vertices, then from Proposition 3, the 1-trail corresponding to this edge belongs to a minimum trail decomposition. Therefore the trail cover obtained by removing this 1-trail from the decomposition produces a trail cover with strictly less trail than is possible from any trail decomposition.

This naturally leads to a heuristic algorithm for finding trail covers. However, selecting edges at random will not always lead to a minimum trail cover as we can see in the graph below.

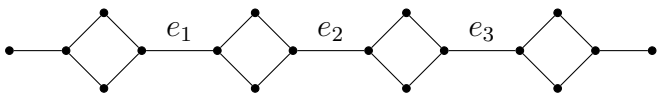


Figure 4.1: *If we removed e_2, the minimum trail cover of the resulting graph has four trails. If we instead removed edges e_1 and e_3 from the graph, the resulting trail cover only has three trails and is in fact minimal. By choosing e_2 we are no longer able to chose either e_1 or e_3 since taking them would produce a connected component with no non-zero odd vertices.*

4.3.2 The minimum trail cover problem is NP-hard

Since there are efficient algorithms for finding minimum trail decompositions, it is natural to ask if there exist efficient algorithms for finding minimum trail covers. Here we show that in general the problem of finding the optimal trail cover is NP-hard. We might expect that the minimum trail cover problem should be at least as hard as the minimum path cover problem. We confirm this intuition by showing that solutions to `MinTrailCover` can produce solutions to `MinPathCover` on cubic graphs.

Theorem 10. `MinTrailCover` *is NP-hard.*

Proof. Let G be a cubic graph and suppose $\mathcal{C}$ is a minimum trail cover for G. Then for any vertex traversed by two trails in $\mathcal{C}$, one of the trails must have its endpoint at the vertex since the degree of the vertex is three. Then retracting the trail by removing the final edge producing a trail cover of the same size. By performing these retractions wherever possible, we obtain a trail cover where no two trails traverse the same vertex. Thus each trail in now a path and we have found a minimum path cover of G.

Since finding a Hamiltonian path in a cubic graph is NP-hard [50], finding a minimum path cover is also NP-hard and therefore `MinTrailCover` is NP-hard. □

It naturally follows that the `MinBoundedTrailCover` is also NP-hard.

Remark 2. This also implies the corresponding graph problem for minimising fusion networks with linear resource states of bounded photon length is NP-hard.

Suppose we have a trail T in a cubic graph where each trail corresponds to a linear resource state of length L. Then by the proof of Theorem 10, we see that we take T to be a path. Then each node in the resource state has one photon for a measurement (or for output) and one photon for every fusion that occurs at the node. Since T is in a cubic graph, each intermediate node has one fusion and each endpoint has two fusions. Thus the resource state consists of at most $2(L-1)+2*3 = 2L+4$ photons. Therefore a solution to the bounded photon fusion network problem of size $2L+4$ is a solution to the minimum L-trail cover on cubic graphs which we know to be NP-hard from the proof of Theorem 10. Observe that if we had $2L+5$ photons, the size of our resource states would not change since any additional intermediate node required two photons. Therefore we can conclude this problem is also NP-hard.

4.3.3 Maximal trail covers

We now introduce a special class of trail cover than have a remarkably similar structure to minimum trail decompositions we call *maximal trail covers*. This will help guide the development of efficient approximation algorithms in Chapter 5.

We denote the subgraph of G covered by the trail T by $G(\mathcal{T})$, and similarly, denote the subgraph of G covered by the trail cover $\mathcal{C}$ by $G(\mathcal{C})$.

Definition 20. A trail cover $\mathcal{C}$ of a graph G is *maximal* if $\mathcal{C}$ is a minimum trail decomposition of $G(\mathcal{C})$ and for any trail cover $\mathcal{C}'$ of G where $G(\mathcal{C}) \subsetneq G(\mathcal{C}')$, $|\mathcal{C}| < |\mathcal{C}'|$.

Informally, this means that any trail cover that covers more than the subgraph of a maximal trail cover must have strictly more trails.

The main insight in this section is that every maximal trail cover is a subset of some minimum trail decomposition. This is important as minimum trail decompositions are

highly structured and by establishing this link, we can carry over results and heuristics for minimum trail decompositions to the task of finding minimum trail covers.

A *closed trail* is a trail whose endpoint are located at the same vertex, otherwise we say it is *open*. We define the *rotations* of a closed trail T to be all the trails that can be obtained by moving the end point to another vertex traversed by the trail. The rotations of a trail is simply just the trail itself.

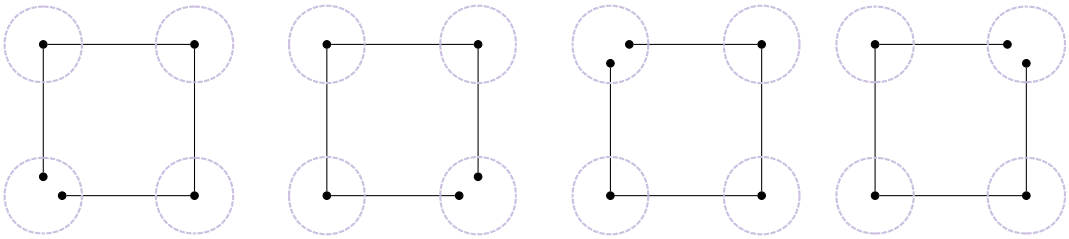


Figure 4.2: *All possible rotations of a closed trail. The grey dashed edge denotes a vertex in the graph where potentially multiple trails may end at.*

Note that if a graph G is connected and has no odd vertices, then the minimum trail decomposition is an Eulerian circuit which a is a minimum trail cover that is maximal. For all other cases, we have the following proposition.

Proposition 5. *Let $\mathcal{C}$ be a maximal trail cover of a connected graph G with non-zero odd vertices. Then for all $T \in \mathcal{C}$, neither T nor any rotation of T ends at a vertex with adjacent edges in $G \backslash G(\mathcal{C})$.*

Proof. Let $\mathcal{C}$ be a maximal trail cover for G. Then if a trail or its rotation in $\mathcal{C}$ ends at a node with edges in $G \backslash G(\mathcal{C})$, we could extend the trail to include one of the edges and obtain a trail cover of the same size which covers a larger subgraph, hence contradicting the maximality of $\mathcal{C}$. □

From this proposition, we obtain two simple lemmas.

Lemma 4. *Let $\mathcal{C}$ be a maximal trail cover of a connected graph G. Then every connected component of $G(\mathcal{C})$ has non-zero odd vertices if G has non-zero odd vertices.*

Proof. Since $\mathcal{C}$ is a minimum trail decomposition on $G(\mathcal{C})$, if there exists a connected component S of $G(\mathcal{C})$ with zero odd vertices, then there is a trail $T \in \mathcal{C}$ which is an Eulerian circuit of S. Then if Proposition 5 holds, then no vertex in S has is adjacent to a vertex in $G \backslash S$. Since G is assumed to be connected, this implies $S = G$, and thus G has zero odd vertices. Thus by the contrapositive, if G has non-zero odd vertices, then every connected component of $G(\mathcal{C})$ has non-zero odd vertices. □

Lemma 5. *Let $\mathcal{C}$ be a maximal trail cover of a connected graph G. Then vertices that are odd in $G(\mathcal{C})$ are odd in G.*

Proof. Now assume that G has non-zero odd vertices. Then by Lemma 4, any trail in $\mathcal{C}$ belongs to a connected component with non-zero odd vertices. Therefore since $\mathcal{C}$ is a maximal trail cover which is a minimum trail decomposition on $G(\mathcal{C})$, each trail in $\mathcal{C}$ ends at distinct vertices that are odd in $G(\mathcal{C})$. By Proposition 5 these vertices have no adjacent edges in $G \backslash \mathcal{C}$ and therefore they are odd in G as well. □

We are now able to proof the main theorem of this section which demonstrates the link between maximal trail covers and minimum trail decompositions.

Theorem 11. *A trail cover $\mathcal{C}$ of a graph G is maximal if and only if it is a subset of a minimum trail decomposition of G and $G \backslash G(C)$ is a tree.*

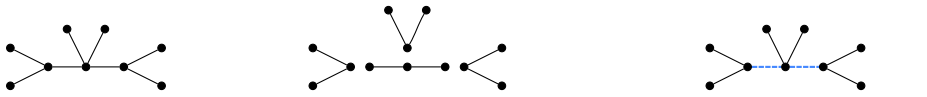
Proof. In the case where G has no odd vertices, maximal trail covers are exactly minimum trail decompositions consisting of a single Eulerian tour of G and so the theorem holds. For the rest of the proof we will consider the case where G has non-zero odd vertices.

Let $\mathcal{C}$ be a maximal trail cover of G and let T be a trail in $\mathcal{C}$. Then from Lemma 4, the connected component of $G(\mathcal{C})$ that contains T has non-zero odd vertices. Since $\mathcal{C}$ is a minimum trail decomposition on $G(\mathcal{C})$, T ends at vertices that are odd in $G(\mathcal{C})$ which by Lemma 5, are odd in G as well.

Similarly, it follows that since T is in a minimum trail decomposition of $G(\mathcal{C})$, every connected component of $G(\mathcal{C}) \backslash T$ has odd vertices and since odd vertices in $G(\mathcal{C})$ are odd in G, every connected component of $G \backslash T$ must also have odd vertices. Therefore by Proposition 3, T belongs to a minimum trail decomposition of G. Since T was arbitrary, we conclude that $\mathcal{C}$ is a subset of some minimum trail decomposition of G.

Now suppose $\mathcal{C}$ is a trail cover and is a subset of some minimum trail decomposition of G and $G \backslash G(\mathcal{C})$ is a tree. Then naturally $\mathcal{C}$ is a minimum trail decomposition of $G(\mathcal{C})$. For any trail cover $\mathcal{C}'$ that is a minimum trail decomposition on $G(\mathcal{C}')$ where $G(\mathcal{C}) \subsetneq G(\mathcal{C}')$, Proposition 5 tells us that every edge in $G(\mathcal{C}') \backslash G(\mathcal{C})$ is adjacent to a vertex that is even in $G(\mathcal{C})$. Since $G \backslash G(\mathcal{C})$ is a tree, any subgraph of $G \backslash G(\mathcal{C})$ has non-zero odd vertices and therefore $G(\mathcal{C}')$ must have strictly more odd vertices than $G(\mathcal{C})$. Since $\mathcal{C}$ and $\mathcal{C}'$ are both minimum trail decompositions on a graph with non-zero odd vertices, we have $|\mathcal{C}| = \frac{1}{2}|\texttt{Odd}(G(\mathcal{C}))| < \frac{1}{2}|\texttt{Odd}(G(\mathcal{C}'))| = |\mathcal{C}'|$. Therefore $\mathcal{C}$ is maximal. □

Note that any set of edge disjoint paths that belong to a minimum trail decomposition form a tree. Therefore we can find maximal trail covers by removing paths that belong to a minimum trail decomposition and finding minimum trail decompositions of the remaining graph. For example:



Original Minimum trail decomposition Minimum trail cover

We will formalises this intuition into a heuristic algorithm in Chapter 5.

As a final note on maximal trail covers, every trail cover can be converted into a maximal trail cover in polynomial time of the same size. Therefore the problem of finding minimum trail covers and that of finding minimum trail covers that are maximal are polynomially equivalent.

Proposition 6. *Given a trail cover $\mathcal{C}$ of a graph G, we can find a maximal trail cover $\mathcal{C}'$ such that $|\mathcal{C}'| \leq |\mathcal{C}|$ in polynomial time.*

Proof. Suppose we are given a trail cover $\mathcal{C}$ of G. First replace $\mathcal{C}$ with a minimum trail decomposition on $G((C))$, denoted $\mathcal{C}'$. Then for each trail in $\mathcal{C}'$, check if it can be extended to traverse an edge in $G \backslash \mathcal{C}$. We continue this until it is no longer possible and obtain a new trail cover $\mathcal{C}''$. Then for all trails in $\mathcal{C}''$ that traverse a vertex which belongs to a closed loop in $G \backslash \mathcal{C}''$, we modify the trail to traverse this loop whilst remaining the same everywhere

else. We then replace the remaining trail cover again with a minimum trail decomposition on $G(\mathcal{C}'')$ to obtain the trail cover $\mathcal{C}'''$. This trail cover has the property that all of its trails end at vertices with no adjacent trails in $G \backslash \mathcal{C}'''$. Therefore odd vertices in $G(\mathcal{C}''')$ are odd in G. Since trails in $\mathcal{C}'''$ belong to a minimum trail decomposition removing any of them from $G(\mathcal{C}''')$ will produce connected components with non-zero odd vertices, and thus the same holds if we were to remove the trail from G. Therefore trails in $\mathcal{C}'''$ are part of a minimum trail decomposition in G. Since we modified our trail cover to traverse any loops in the complement graph, $G \backslash \mathcal{C}'''$ is loop-free and if therefore a tree.

Thus by Theorem 11, $\mathcal{C}'''$ is a maximal trail cover. This constitutes a polynomial reduction since each of the operations can be performed in polynomial time. □

Chapter 5

Approximating Trail covers

Previous chapters have show the NP-hardness of finding minimum trail covers and its related problems. In this chapter we will present approximation algorithms for each problem and prove bounds on their complexity and accuracy.

5.1 Approximating minimum L-trail decompositions

Since the minimum L-trail decomposition problem is NP-hard (Theorem 9), we will now focus our attention on finding efficient approximation algorithms. A natural choice would be to find a minimum trail decomposition of unbounded length of the graph and subdivide the trails into L-trails. We will prove tight bounds on this algorithm and show that the accuracy depends linearly on the number of odd vertices and discuss more advanced heuristics.

Lemma 6. *Let $(t_i)_{i=1}^N$ and L be positive integers. Then*

$$\sum_{i=1}^{N} \left\lceil \frac{t_i}{L} \right\rceil - \left\lceil \sum_{i=1}^{N} \frac{t_i}{L} \right\rceil = \sum_{i=1}^{N} \left\lceil \frac{t_i \bmod L}{L} \right\rceil - \left\lceil \sum_{i=1}^{N} \frac{t_i \bmod L}{L} \right\rceil. \tag{5.1}$$

Proof. For positive integers a and b, we have $\frac{a}{b} = \left\lfloor \frac{a}{b} \right\rfloor + \frac{a \bmod b}{b}$. Therefore

$$\sum_{i=1}^{N} \left\lceil \frac{t_i}{L} \right\rceil = \sum_{i=1}^{N} \left\lfloor \frac{t_i}{L} \right\rfloor + \sum_{i=1}^{N} \left\lceil \frac{t_i \bmod L}{L} \right\rceil. \tag{5.2}$$

and

$$\left\lceil \sum_{i=1}^{N} \frac{t_i}{L} \right\rceil = \sum_{i=1}^{N} \left\lfloor \frac{t_i}{L} \right\rfloor + \left\lceil \sum_{i=1}^{N} \frac{t_i \bmod L}{L} \right\rceil \tag{5.3}$$

Subtracting (5.3) from (5.2) gives

$$\begin{aligned} \sum_{i=1}^{N} \left\lceil \frac{t_i}{L} \right\rceil - \left\lceil \sum_{i=1}^{N} \frac{t_i}{L} \right\rceil &= \sum_{i=1}^{N} \left\lfloor \frac{t_i}{L} \right\rfloor + \sum_{i=1}^{N} \left\lceil \frac{t_i \bmod L}{L} \right\rceil - \left(\sum_{i=1}^{N} \left\lfloor \frac{t_i}{L} \right\rfloor + \left\lceil \sum_{i=1}^{N} \frac{t_i \bmod L}{L} \right\rceil \right) \\ &= \sum_{i=1}^{N} \left\lceil \frac{t_i \bmod L}{L} \right\rceil - \left\lceil \sum_{i=1}^{N} \frac{t_i \bmod L}{L} \right\rceil. \end{aligned}$$

□

This sum is bounded above by the following inequality.

Corollary 1. *Let $(t_i)_{i=1}^N$ and L be positive integers. Then the following inequality holds and is tight.*

$$\sum_{i=1}^{N} \left\lceil \frac{t_i}{L} \right\rceil - \left\lceil \sum_{i=1}^{N} \frac{t_i}{L} \right\rceil \leq N - \left\lceil \frac{N}{L} \right\rceil .$$

Proof. By Lemma 6, it is sufficient to show that

$$\sum_{i=1}^{N} \left\lceil \frac{t_i \bmod L}{L} \right\rceil - \left\lceil \sum_{i=1}^{N} \frac{t_i \bmod L}{L} \right\rceil \tag{5.4}$$

maximised when $t_i \bmod L = 1$ for all $1 \leq i \leq N$ and is equal to $N - \left\lceil \frac{N}{L} \right\rceil$.

Suppose that for some $1 \leq i \leq N$ we have $t_i \bmod L = 0$. Then if instead we had $t_i \bmod L = 1$, the first sum of (5.4) increases by one and the second sum increase by at most one. Hence (5.4) does not decrease. Suppose now that $t_i \bmod L > 1$. Then if $t_i \bmod L = 1$, first sum in (5.4) will not change and the second sum may decrease by one. Hence (5.4) will not decrease in this case either.

Therefore the configuration where $t_1 \bmod L = \cdots = t_N \bmod L = 1$ is no less than any other assignment of $(t_i)_{i=1}^N$ and is therefore the maximum. Substituting these values into (5.4) gives us the desired equality. □

Now we can present our approximation algorithm and prove tight bounds on its accuracy.

Proposition 7. *We can find an L-trail decomposition of a graph G in polynomial time which has at most $\left\lfloor \frac{1}{2}|\texttt{Odd}(G)|(1 - \frac{1}{L}) \right\rfloor$ more trails than the minimum.*

Proof. Suppose we have a minimum trail decomposition $\mathcal{T} = \{T_1, \ldots, T_K\}$ for some integer K. Then subdividing each trail into L-trails produces an L-trail decomposition of size $\sum_{i=1}^{K} \left\lceil \frac{|T_i|}{L} \right\rceil$ where $|T_i|$ is the number of edges in the trail T_i.

A lower bound for the minimum number of trails in an L-trail decomposition is $\left\lceil \frac{|E|}{L} \right\rceil$, or equivalently $\left\lceil \frac{1}{L} \sum_{i=1}^{K} |T_i| \right\rceil$. Hence the difference between the number of trails in $\mathcal{T}$ and in the minimum L-trail decomposition is at most

$$\sum_{i=1}^{K} \left\lceil \frac{|T_i|}{L} \right\rceil - \left\lceil \sum_{i=1}^{K} \frac{|T_i|}{L} \right\rceil \leq K - \left\lceil \frac{K}{L} \right\rceil = \left\lfloor K \left(1 - \frac{1}{L} \right) \right\rfloor .$$

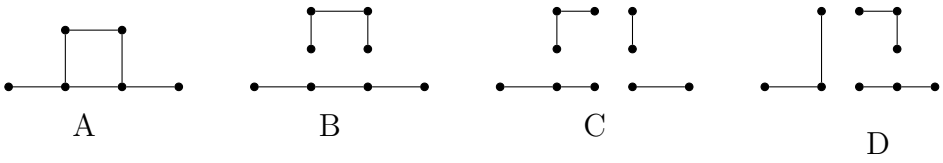
where the inequality follows from an application of Corollary 1.

From Theorem 6 we know that $K = \frac{1}{2}|\texttt{Odd}(G)|$ if $|\texttt{Odd}(G)| > 0$ and $K = 1$ otherwise. However, in the case where $|\texttt{Odd}(G)| = 0$, this subdivision gives a minimum L-trail decomposition anyway, so the proposition still holds. □

Note that when L is larger than the length of any trail in the decomposition, it is impossible for the case outlined in the proof of Proposition 7 to occur, and naturally there is no error in our approximation since it is exactly the optimal solution with no subdivisions.

This is a tight bound as can be seen in the example below where we illustrate this subdivision algorithm. Suppose we wish to find a minimum 2-trail decomposition for the

graph (A). We first find a minimum trail decomposition (B) and subdivide it to get 4 2-trails (C). However the minimum 2-trail decomposition has size 3 (D). This achieves the maximum error bound stated in Proposition 7, namely $\frac{1}{2}|\texttt{Odd}(G)|(1-\frac{1}{L}) = \frac{1}{2}\times 4(1-\frac{1}{2}) = 1$.



Thus reducing the number of odd vertices increases the accuracy of the approximation. Indeed if there are less than 3 odd vertices, then we obtain a minimum trail decomposition. We now show that on average this trail performs twice as good as the worst case.

Proposition 8. *The result of Proposition 7 on average contains at most* $\frac{1}{4}|\texttt{Odd}(G)|$ *trails more than the minimum.*

Proof. Let $\mathcal{T}$ be a minimum trail decomposition, and suppose that suppose that N of the trails in $\mathcal{T}$ are a multiple of L. Then on average the remaining trails T_i have length $T_i \mod L = \frac{1}{2}L$. Therefore substituting into (5.4) we have at most

$$K - N - \frac{(K-N)(\frac{L}{2})}{L} = \frac{K-N}{2}$$

more trails than the minimum. This reaches a maximum of $\frac{K}{2}$ when $N = 0$. Therefore the expected number of trails is at most $\frac{K}{2}$ more than the minimum. Substituting $K = \frac{1}{2}|\texttt{Odd}(G)|$ gives the result. □

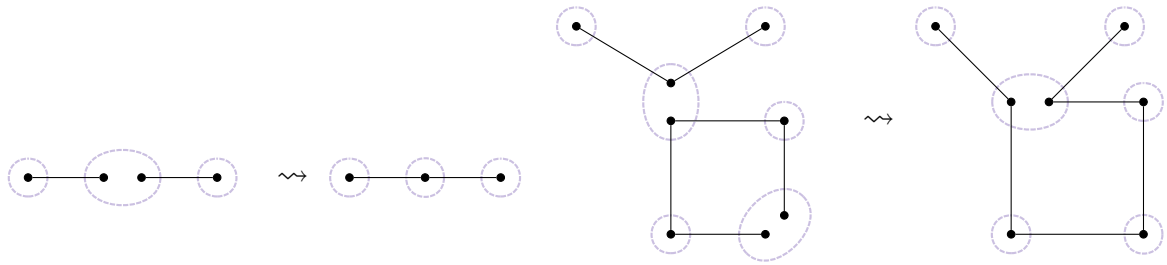
Remark 3. This subdivision algorithm can be easily modified for the case of linear resource states with a bounded number of photons. Instead of subdividing the trails based on the length of the trail, we can subdivide based on the number of photons required in the linear resource states implementing them. Each node in the resource state requires one photon for each fusion that acts on it and one photon for a measurement (or for output). We can then search for trails that belong to a minimum trail decomposition and be subdivided based on the number of photons they require. We can then find a minimum trail decomposition of the remaining graph and subdivide again to obtain the entire trail decomposition.

We now establish a link between bounded minimum trail decompositions and unbounded counterparts.

Proposition 9. *Any trail decomposition can be transformed into a minimum trail decomposition with two rules:*

- *if a trail ends where another trail ends then join them together, and*
- *if a closed trail traverses the same vertex as another trail, join them together*

These two rules are illustrated below. The dash lines represent a node in the graph which may be traversed by multiple trails.



Proof. Let $\mathcal{T}$ be a trail decomposition of G obtained by applying the two rules on some trail decomposition. We will show that all trails in the decomposition satisfy the definition of belonging to a minimum trail decomposition.

Let $T \in \mathcal{T}$. If T is the only trail in its connected component, then it belongs to a minimum trail decomposition and we are done. Now suppose that T is not the only trail in its connected component. Then no other trail in $\mathcal{T}$ can end at the same vertices that T ends at since otherwise we could have joined them. This also implies that T ends at distinct odd vertices or both endpoints are at an even vertex and T is closed. However, if T is closed we would have been able to apply the second rule to join it with another trail in the connected component. Thus T ends at distinct odd vertices. Since T was arbitrary, this implies that every trail in the connected component of T also ends at distinct odd vertices and therefore $\mathcal{T}$ is a minimum trail decomposition on the connected component of T. We can therefore conclude that every $\mathcal{C}T$ is minimum trail decomposition on any connected component of G and thus $\mathcal{T}$ is a minimum trail decomposition on G. □

Remark 4. The reduction in Proposition 9 can be used to prove Theorem 6 without the need to invoke Euler's Theorem.

We now present the key result that will enable us to provide a better heuristic for finding minimum L-trail decompositions.

Proposition 10. *For any graph G, there exists a minimum L-trail decomposition of G that is a subdivision of a minimum trail decomposition of G.*

Proof. Suppose we have a minimum L-trail decomposition $\mathcal{T}$ of G. Then by using the reduction in Proposition 9, we are able to convert it into a minimum trail decomposition $\mathcal{T}'$ of G. Since for each trail T in $\mathcal{T}$, every edge in T is included in exactly one trail in $\mathcal{T}'$, subdividing each trail of $\mathcal{T}'$ will produce a minimum L-trail decomposition. □

To see how we may formulate this into a heuristic, suppose we have a minimum trail decomposition $\mathcal{T} = \{T_1, \ldots, T_K\}$ of a graph G which has $K = \frac{1}{2}|\texttt{Odd}(G)|$ if $|\texttt{Odd}(G)| > 0$ and $K = 1$ otherwise. Then subdividing it into L-trails produces

$$\sum_{i=1}^{K} \left\lceil \frac{|T_i|}{L} \right\rceil$$

trails. This sum is minimised when the number of trails whose length is a multiple of L is maximum.

We can then summarise this finding by saying that `MinBoundedTrailDecomposition` is equivalent to the problem of finding an unbounded minimum trail decomposition which has a maximum number of trails whose length is a multiple of L. Hence generating all minimum trail decompositions is NP-hard since other we could use it to find a solution for `MinBoundedTrailDecomposition` which we know to be NP-hard.

Algorithm 1.
Input: *A graph G.*
Output: *An L-trail decomposition of G.*

1. *If $|\texttt{Odd}(G)| \leq 2$, then find an Eulerian path and subdivide into L-trails and return.*
2. *Otherwise, search for a trail of length some multiple of L that satisfies the conditions of being in a minimum trail decomposition.*
3. *Repeat Step 1 and 2 again until we can not find any more such trails.*
4. *Remove these trails from the graph and find a minimum unbounded trail decomposition on the remaining graph.*
5. *Subdivide the overall trail decomposition intro L-trails and return.*

Algorithm 1 terminates with a solution in polynomial time if the search algorithm terminates in polynomial time. Since the search algorithm may fail to find a suitable trail despite one existing, Algorithm 1 has the same approximation ratio as the original subdivision algorithm in Proposition 7.

The search algorithm may be implemented using a breadth first search which terminates after a predetermined time if no suitable trail is found. Though the search space consists of all possible paths of a graph and is therefore exponential in size, the search algorithm can exploit the structure of the minimum trails to speed up the search.

5.2 Approximation Minimum trail covers

We now turn our attention to the problem of finding heuristic algorithms for minimum trail covers. From Proposition 6 we know that it is sufficient to search in the space of maximal trail covers which have the structure of minimum trail decompositions.

We may then formalise this in to a greedy algorithm for finding a trail cover of a given graph.

Algorithm 2.
Input: *A graph G.*
Output: *A maximal trail cover for G.*

1. *Search for paths belonging to a minimum trail decomposition that traverse only edges with degree at least two taking the shortest possible path.*
2. *Remove this path from the graph and continue until we can not remove any more paths*
3. *Find a minimum trail decomposition of the remaining graph.*
4. *Return the trail decomposition which is a maximal trail cover by Theorem 11.*

This algorithm can be easily modified to find L-trail covers.

Algorithm 3.
Input: *A graph G and an integer L.*
Output: *An L-trail cover for G.*

1. *Execute Steps 1 and 2 of Algorithm 2 to remove paths from the graph.*

2. *Modify Algorithm 1 to attempt to find a trail decomposition of the remaining graph where the length of the trails are of the form $L + k(L + 1)$ for some non-negative integer k.*

3. *Subdivide the trails into L-trails and return.*

The reason we search for trails with length of the form $L + k(L + 1)$ is because when subdividing we can break the trail into k trails of length L by omitting the edge between successive trails and still have a trail decomposition. See the example below for deconstructing a trail of length 5 into three 1-trails.

Algorithm 2 may be executed in polynomial time and Algorithm 3 may be executed in polynomial time if the search algorithm used in the invocation of Algorithm 1 is executed in polynomial time. In the worst case we do not find any suitable trails with Algorithm 1 and so we just return a minimum trail decomposition of size $\frac{1}{2}|\texttt{Odd}(G)|$. It may be the case that the minimum trail cover contains only one trail, and therefore our solution contains at most $\frac{1}{2}|\texttt{Odd}(G)| - 1$ more trails than the minimum.

Remark 5. Algorithm 3 is based on subdivision and is therefore easily adapted to the problem of minimising fusions fusion networks with resource states with bounded photons in the same way as previously outlined in the case of the heuristic algorithm for minimum L-trail decompositions.

5.2.1 Reduction to Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is one of the most well known and widely studied problems in computer science. The problem is to find a Hamiltonian cycle in a weighted graph that minimises the sum of the edges it traversed. Despite being NP-hard, many advanced heuristics and optimisations exist to allow the TSP to be solved efficiently for graphs containing thousands of vertices [51].

We will show how the minimum trail cover problem can be reduced to an instance of the graph-theoretic TSP and therefore may leverage existing solvers. We define the *multi-visit TSP* to be a variant of the TSP where we relax the requirement of finding a Hamiltonian cycle to that of finding a trail that visits every vertex in the graph. We then show how the multi-visit TSP can be solved with the original TSP to conclude the reduction.

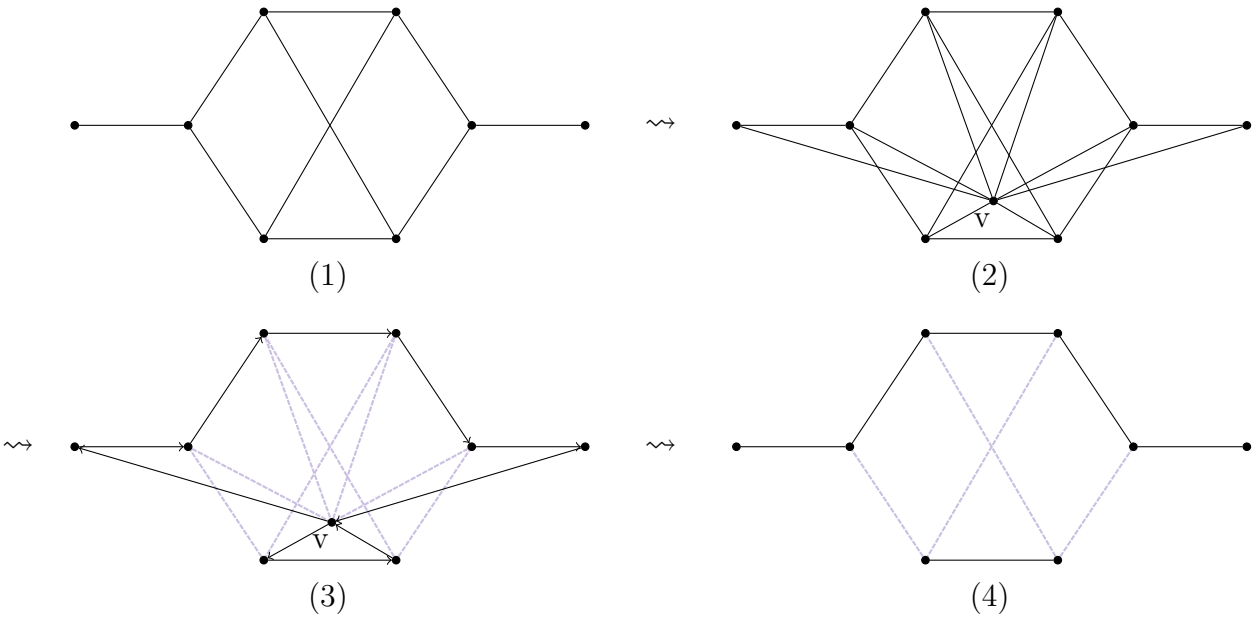
Proposition 11. *Let $G = (V, E)$ be a graph with non-zero odd vertices. Define $G' = (V', E')$ to be the weighted undirected graph obtained by starting with G and adding an addition vertex v, $V' = V \cup \{v\}$, additional edges between v and odd vertices of G, $E' = E \cup \{(v, w) \mid w \in \texttt{Odd}(G)\}$, and setting the weight of edges in the original graph G to be zero, and the weight of the new edges adjacent to v to be one. Then solutions to the multi-visit TSP on G' correspond to minimum trail covers on G that are maximal.*

Proof. Let T be a solution to the multi-visit TSP on G' with total weight W. Then by removing edges from T that are adjacent to the added vertex v, we obtain a trail cover on $G \subset G'$ of size $W/2$.

Similarly, given any minimum trail cover that is maximal, we know from Theorem 11 that trails in $\mathcal{C}$ belong to some minimum trail decomposition and hence each trail ends

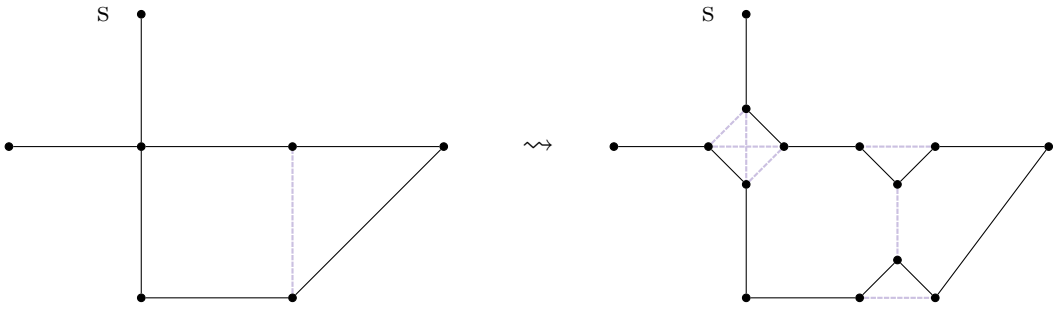
at distinct odd vertices (Proposition 5). Thus we can construct a solution to the multi-visit TSP problem by connecting trails through the added vertex v. Since there are $|\mathcal{C}|$ trails and each edge adjacent to v has weight one, the resulting trail has a total weight of $2|\mathcal{C}|$. This is the same weight as was obtained by the solution to the multi-visit TSP and therefore the trail cover obtained from the solution is a minimum trail cover that is maximal. □

This procedure is illustrated in the example below where we begin with the original graph (1), then add the vertex v with edges of weight one (2), then find a solution to the multi-visit TSP (3) and finally take the image of the solution of the original graph to be our minimum trail cover (4).



Note that we can adapt this result to the normal TSP by replacing each node in G' with degree $d > 2$ with a complete subgraph of size d. Under this transformation, all trails in the original graph become paths and so solutions to the multi-visit TSP become solutions to the TSP on the new graph.

An example is drawn below where the original graph on the left has a trail that begins and ends at the vertex s and covers every vertex in the graph, however it traverses one vertex twice and so is not a solution to the TSP. The graph on the right has undergone the transformation outlined above and so the solution to the multi-visit TSP now corresponds to a solution to the normal TSP and hence to a minimum trail cover.



Chapter 6

Graph Rewrites for fusion networks

In this section we outline several strategies for transforming graphs to generate fusion networks with fewer fusions. We will use the techniques from the previous chapters to develop graph rewrites strategies which modify the graph state to reduce the number of required fusions and enhance the performance of the approximation algorithms developed in Chapter 5. The algorithms here will normally reduce the number of edges in the graph state and so will be of some utility for fusion networks created with star resource states, however we note that the algorithms presenting in [29] would be more useful in this case.

From Chapter 4 we know that without graph rewrites, finding linear XY-fusion networks with the minimum number of fusions is NP-hard in all cases except for the case of X fusions with resource states of unbounded length. It is unknown whether these problems remain NP-hard even when allowing rewrites, and so we will instead present rewrite strategies which heuristically reduce the total number of fusions.

In order for the open graph to be deterministically implementable, we require that our rewrites preserve gflow in the graph. Backens and McElvanney[41] showed that local complementation and Z-deletion (and their inverses) preserve gflow and are sufficient to transform any two labeled open graph with gflow and the same target linear map into each other, and so we these are the only two graph transformations we will consider here.

Our main optimisation stems from the fact that the number of fusions required to implement a graph state with a given trail cover $\mathcal{C}$ is

$$|E| - |V| + |\mathcal{C}|. \tag{6.1}$$

From (6.1) we can reduce the number of fusions in a linear XY-fusion network by either decreasing $|E|$ and $|\mathcal{C}|$ or increasing $|V|$. Finding a trail cover $\mathcal{C}$ such that $|\mathcal{C}|$ is minimal was the topic of Chapter 4. In this section we primarily focus on reducing the number of edges in the graph state.

We remark that one may ask whether we can transform any graph into one with bounded tree-width and so we could solve the trail cover problems in polynomial time. This is not possible however as tree-width is bounded below by rank-width [29] and since the rank-width is invariant under local complementation then we can't reduce it to an arbitrary size.

6.1 Reducing Y Fusions

In this section, we look at rewriting graphs to reduce the number of Y fusions required to implement linear Y-fusion networks. In general, Z-deletions increase the number of edges

in the graph and hence the number of fusions. However, there is one special case which we can show will never increase the number of Y fusions required in a linear Y-fusion network.

Definition 21. Given a graph with a triangle, *complementing the triangle* means using inverse Z-deletion to add a vertex to the graph that is connected to the three vertices of the triangle. Then performing local complementation on the vertex.

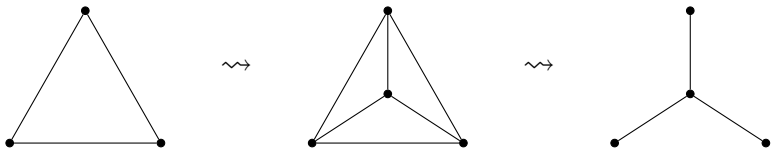


Figure 6.1: *Complementing a triangle by first adding a new node connected to the vertices of the triangle and performing a local complementation about it.*

Complementing the triangle keeps the number of edges the same but increases the vertices by one. Therefore by (6.1) it will decrease the number of fusions if the path cover stays the same size or increases by one. We will now show that indeed this is always the case.

Theorem 12. *Complementing a triangle will never increase the minimum number of required fusions to construct a graph state from an unbounded linear resource state using Y fusions.*

Proof. Assume we have a triangle and some minimum path cover P. We will show that we can always update P after complementing the triangle such that the number of fusions never increases.

Consider the cases where the triangle contains either 0, 1, or 2 edges from the cover. It cannot contain 3 edges as this would imply the paths are not vertex disjoint. These cases are illustrated in Figure 6.2 alongside a modified path flowing through the triangle which does not increase the number of fusions.

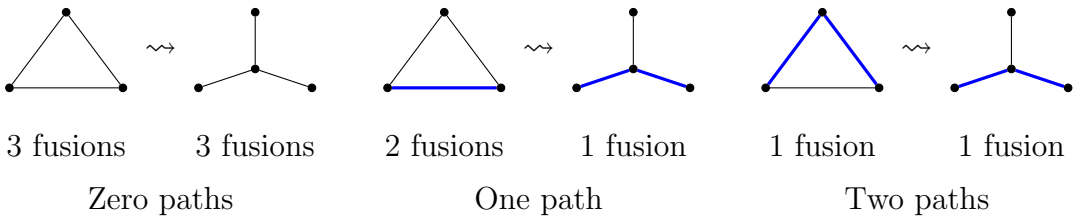


Figure 6.2: *Complementing the triangle never increases the number of fusions. The thick blue lines represent a edges belonging to the path cover, and the dashed lines are edges that must be fused.*

□

We note that this result may not hold in general for the case of bounded path covers. Although it is easily to generalise this technique to cliques of arbitrary size. For cliques of size four or greater, it is easy to see that the number of fusions does not increase since when $n = 4$ the number of edges reduces by 2 and vertices increases by 1. However note that edges in 4-clique could be traversed by at most two paths, so complementing

the clique will at most, split one path into two and increase the size of the path cover by one. Therefore from (6.1) we will always strictly decrease the number of fusions by complementing cliques of size four or greater.

We can then formulate these and the general local complementations into an algorithm.

Algorithm 4.
Input: *An open graph.*
Output: *A fusion network with linear resource states and Y fusions.*

1. *Apply local complementation whenever it strictly reduces the number of edges.*

2. *Complement all cliques.*

3. *Find a path cover of the resulting graph*

4. *Convert the path cover into a fusion network and return.*

Algorithm 4 will terminate in a finite number of steps since Step 1 will terminate because it strictly reduces the number of edges in the graph at each step, and Step 2 will terminate because there is a finite number of triangles in the graph and complementing a triangle does not induce any other triangles in the graph. This can be seen from Figure 6.1 where there can not be any new triangles formed by the new edges.

Step 1 is a heuristic and will not always find the optimal equivalent graph state that requires the fewest fusions to implement. An example of such a situation is illustrated in Figure 6.3.

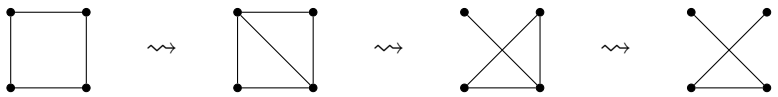


Figure 6.3: *The ring graph state is transformed into a linear graph state requiring fewer fusions through a series of local complementation. However, any local complementation applied to the ring will increase the number of edges in the graph state and so our Step 1 procedure would not apply any rewrites, and therefore does not always find the optimal graph state.*

We also obtain a slight reduction in the number of edges in the graph, which may reduce the time required to find the minimum path cover, however the there is a trade-off in the number nodes and number of edges.

6.2 Reducing X fusions

Reducing the number of edges will also help reduce the number of X fusion and so we may use the same rewrites as in the case of Y fusions. Though in the case of X fusions, we have the advantage that we know the number of odd vertices will increase the number of fusions in the unbounded case (Theorem 6) and will reduce the accuracy of our heuristic algorithm in the case of bounded trails (Algorithm 1). Therefore our rewrites should aim to reduce both the number of edges and the number of odd vertices, which will require us to add additional preconditions on the rewrites from the previous section.

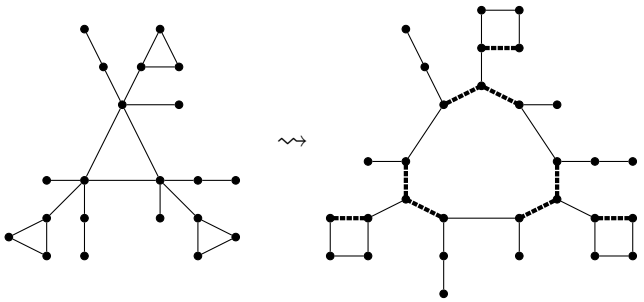
Proposition 12. *Complementing the triangle reduces the number of X fusions required with resource states of unbounded length.*

Proof. Let $G = (V, E)$ be the graph and $S \subset G$ be a triangle in G. Then after complementing the triangle we obtain the graph $G' = (V', E')$ and the number of odd vertices will increase by $4 - 2|\texttt{Odd}(S)|$. The total number of edges in the graph have stayed the same, so $|E'| = |E|$, but the number of vertices has increased by one, so $|V'| = |V| + 1$. Let $\mathcal{T}$ be a minimum trail decomposition for G. Then the number of fusions required to implement the graph will increase by

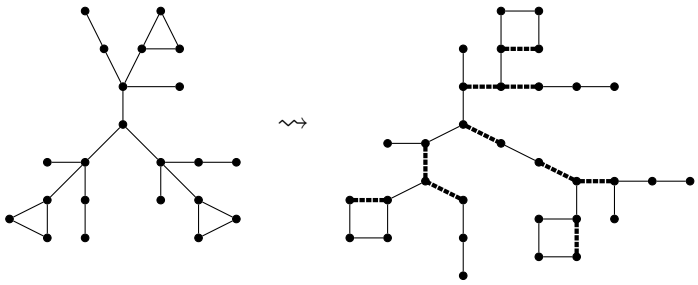
$$|E'| - |V'| + |\mathcal{T}| + \frac{1}{2}(4 - 2|\texttt{Odd}(S)|) - (|E| - |V| + |\mathcal{T}|) = 1 - |\texttt{Odd}(S)|$$

Therefore the number of required fusions only decreases when there are two or more odd vertices in the triangle. □

However, this does not hold in general for bounded trail decomposition as we can see in the counterexample below which is a minimum trail decomposition if therefore the optimal 4-trail decomposition.

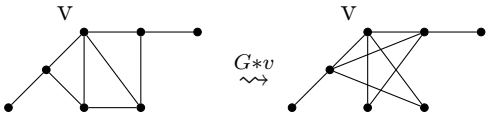


If we were to complement the triangle we get the following graph whose minimum 4-trail decomposition.



The original graph required six 4-trails whereas after complementing the triangle, the new graph required eight.

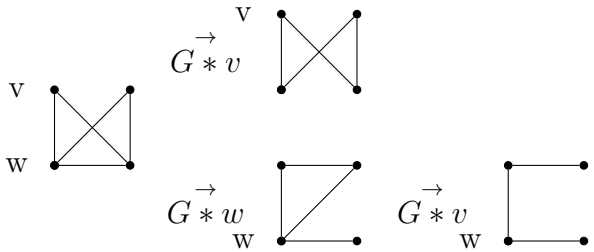
In addition to complementing cliques, we may also greedily locally complement vertices whenever it reduces the number of edges plus odd vertices. The following example illustrates the important of reducing odd vertices well.



Local complementation about the vertex v keeps the number of edges the same but reduces the number of odd vertices from 6 to 2. This means we can implement the graph using fewer unbounded trails and hence less X fusions, and also that we can find

the optimal solution for the bounded trail case by simply subdividing a minimum trail decomposition. The latter reason is important and may grounds to prioritise reducing the number of odd vertices over reducing the number of edges.

We still suffer from the same local minima problem as with the rewrite strategy for Y fusions however. For example, in the following figure, if we first complement by node v, then we get stuck in a minima, whereas it would have been better to first complement by w.



To help overcome this limitation, we experimented with a simulated annealing approach with moderate success and tabulated the results in Chapter 8.

6.2.1 Reducing XY Fusions

To reduce the number of fusions in linear XY-fusion networks, we may use the rewrite rules for the X fusions case as they are the same as the case for Y fusions but with more preconditions. We do not prove that these will always reduce the number of fusions but instead use them as a heuristic, which the benchmarks in Chapter 8 show to be quite effective.

Chapter 7

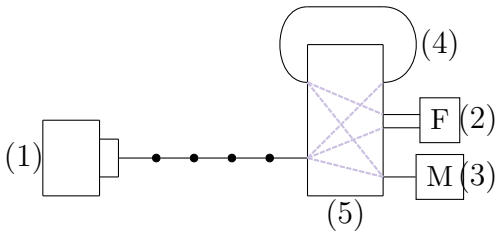
Full-stack compiler for FBQC with linear resource states

In this section we describe a photonic compiler for a specific type of fusion-based architecture proposed in [10]. In the paper, the authors present three main stages in their compilation pipeline: open graph, fusion network, and the optical protocol.

The open graph is the original formulation of the MBQC algorithm at a high level. The fusion network is the implementation strategy used to execute the open graph and is hardware agnostic, though different hardware may work better with different types of fusion networks. The optical protocol is the concrete set of hardware instructions that are executed to implement the fusion network.

Previous chapters have addressed the problem of translating open graphs to fusion networks. Here we will present the problem of converting fusion networks to optical protocols as well as highlighting architectural details which we can use to find better fusion networks which have more performant optical protocols.

Since computations in near-term quantum computing occur incredibly fast, the main factor we wish to optimise in our compilation procedure minimising error. Here we present the class of architectures proposed in [10] but restricted to the case of X and Y fusions.



This architectures consists of the following components: (1) linear graph state generator, (2) fusion module, (3) a measurement module, (4) line, and (5) a photon router.

In these architectures, a single quantum emitter emits a stream of entangled photons in the form of linear cluster states of arbitrary length. One photon is released per timestep during which it is fed into a photon router which directs the photons into either a measurement module, fusion module, or delay line.

The measurement module performs an arbitrary single qubit measurement on the photon, the fusion module performs either an X or Y fusion on the photon and another photon from the delay line, and the delay line holds the photon for a predetermined period of time before directing it into the fusion or measurement modules.

We refer to this as a class of architectures as there are many possible constraints a physical setup could place of the operations we can perform. The primary choices to be made are: how many photons can be held in a delay line and for how long, whether a measurements and a fusion can be performed on separate photons during the same time step, and the number of photons that can be measured simultaneously during a given time step. We will fix a concrete architecture in Section 7.1 to act a compilation target which we can then analyse and optimise for.

We note that the primary sources of error in this model arise from photon loss in the router and delay line, and fusion failures. The probability of photon loss when passing through the router is p and there is a $e^{-\mu t}$ chance of photon loss in the delay line when held for time t and μ is some constant.

Assuming the probability of fusion failure is 50% and the probability of photon loss from the delay loop is $\exp(-\mu t)$ where t is the delay time and $\mu > 0$ is a small constant. The error can therefore be formulated as:

$$P(\mu, p, d_1, \ldots, d_n) = p^n 2^{-F} \prod_{i=1}^{n} \exp(-\mu t_i) = 2^{-F} \exp(-\mu \sum_{i=1}^{n} t_i)$$

where there are n photons and t_i is the delay for photon i. Note that p^n is minimised when the number of fusions is minimised. Therefore our optimisation problem is find to minimise $P(\mathcal{F}, \mu)$ which amounts to minimising fusions and minimising the sum of the delay times.

We will now define a set of instructions called an *optical pattern* that can characterise any computation on this class of architectures and which allows us to precisely evolute the error associated with a particular implementation in our simplified error model.

Definition 22 (Optical Pattern)**.** An *optical pattern* is a sequence of pairs. The first element the pair is a *photon emission command*

- N: Emit an entangled photon.
- N_H: Emit a photon that has changed basis.

and the second element is a *router command*.

- $F[d]$: Fuse a photon. d is an integer denoting an optional time to delay the photon before it is fused.
- $M[d]$: Measure a photon. d is an integer denoting an optional time to delay the photon before it is measured.

The only rule regarding the ordering of the sequence is that when fusing a photon, there must be another photon coming out of the delay at the same time.

We need not consider the case where we need to fuse two delayed photons since De Felice et al. [10] showed that correction never need to be performed at fusions, and so there is no need to delay the second photon before performing the fusion.

This calculus bears a close resemblance to the commonly used measurement calculus [52]. However the main difference is the photon emission command and the X fusion operation, which is a non-unitary projection and so isn't easily expressed in the measurement calculus.

Note that we have not included the measurement angles or fusion types necessary for the calculus to completely describe any quantum computation. This is because our purpose is to use to analyse errors arising from delays and fusion and leave the full characterisation as future work.

Given an optical pattern P, we define the *total delay* of P to be the sum of the total delays of each photon. If D is the total delay of P, then we define the *total error* of P

$$E(P, \mu, p) = p^n 2^{-F} \exp(\mu D)$$

where μ and p are constants and F is the number of fusions in P.

If the probability of photon loss in the delay line is low, the problem of minimising error simply becomes to minimise the number of fusions. In this case, the problem is exactly that considered in previous sections. We will now consider the case where the probability photon loss is high.

7.1 Minimising delays on SEMM devices

In order to precisely state the error associated with a fusion network, we must first fix a concrete architecture to act as a compilation target.

Here we define the notion of a *single emitter-multiple measure* device as a simple concrete architecture with a well defined error model with which to analyse the problem of reducing errors.

Definition 23 (SEMM)**.** A *single emitter-multiple measure* device is a single emitter architecture which does not restrict the number of number of photons we can measure simultaneously or hold in a delay line. An optical protocol on such a device has no additional constraints.

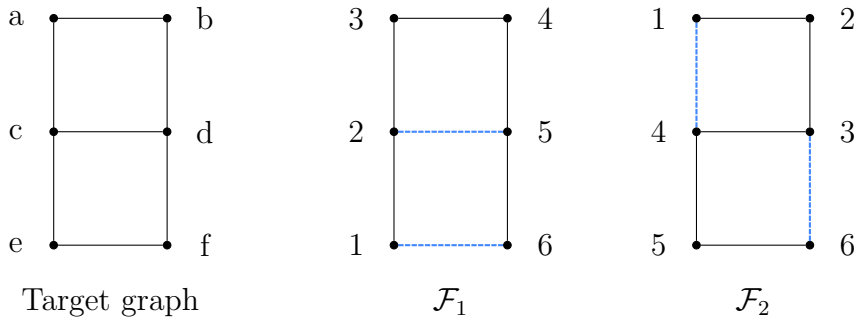
Relieving any restriction on the number of photons which can be delayed at the same time prevents a cascade of photons from preventing others from being measured and greatly simplifies the analysis and allows us to define the minimisation problem.

Definition 24. `MinErrorFusionNetwork`
Input: An open graph $\mathcal{G}$, and real constants μ and p.
Output: An optical pattern P that is executable on an SEMM machine and implements $\mathcal{G}$ such that $E(P, \mu, p)$ is minimised.

If you have a variable number of emitters then the problem becomes much simpler since the partial order given by gflow induces a DAG on the graph and would be able to be efficiently implemented if we have the same number of emitters as the width of the DAG. In this case, no photon would need to enter a delay line if photons could be emitted asynchronously.

Note that there is not a canonical translation between fusion networks and optical patterns. Since given a linear XY-fusion network $\mathcal{F} = (G, I, O, X, Y, \lambda, \alpha)$ where G is the disjoint union of linear graphs $R_1, \ldots, R_k$, there is a choice to be made as to which order to emit the resource states in (there are $k!$ possible combinations), and from what end of each linear graph we should start emitting from. Hence, given a linear XY-fusion network, creating an optical pattern which have minimum delay is also a minimisation problem with an exponential search space.

Not every minimum trail cover will produce an optical pattern with the smallest delay. For instance, consider the fusion networks below which have inputs a and b and outputs e and f. The fusion networks $\mathcal{F}_1$ and $\mathcal{F}_2$ drawn below are both minimum trail covers but have different delay structures. This is because in order for fusion to occur, one photon must wait for the other photon to be emitted. The vertices in the graphs below have been annotated with the time stamp at which the photon for each vertex was emitted.



Target graph $\mathcal{F}_1$ $\mathcal{F}_2$

We can see in this example that in the first fusion network $\mathcal{F}_1$ the fusion photon in node e are delayed for 5 time stamps and the photons in c are delayed for three time stamps which gives a total delay of 8. In the second fusion network $\mathcal{F}_2$, the photons a and d are delayed for 3 timesteps which gives a total delay of 6. Therefore the second fusion network is when the probability of photon loss is high.

In general, emitting photons that are lower in the partial order given by gflow on the open graph (such as the inputs) will reduce the total delay of the optical protocol. We can encode this fact into a heuristic algorithm for finding fusions networks with the goal of minimising the total delay.

Algorithm 5.
Input: *A graph G and a partial order $\prec$ of the vertices of G.*
Output: *A fusion network $\mathcal{F}$ that implements G.*
Hyperparameters: *A loss function which takes a tuple of integers as inputs*

1. *Assign a number called the "layer number" of a node based on the partial order $\prec$. Higher layers require nodes from the lower layers to be measured first. Layer 0 can be measured immediately.*

2. *Start at a node with the smallest layer number and search for a trail of length L that minimises the loss function when supplied with the layer numbers associates with all the nodes in the trail.*

3. *Remove the trail from the graph and repeat Step 2 until the graph is covered.*

4. *Return the trail cover*

We experimented with loss functions corresponding to the L_1, L_2 and L_{max} norms and found the L_{max} norm to be the most successful for small L and L_2 to be more effective for larger values of L. The intuition is the guide the algorithm into finding paths with nodes that have small layer numbers. These nodes correspond to vertices lower in the partial order and as such, do not have to be delayed as long before they are measured.

$$L_1(x_1,\ldots,x_n) = \sum_{i=1}^{n} x_i L_2(x_1,\ldots,x_n) = \sum_{i=1}^{n} x_i^2 L_{max}(x_1,\ldots,x_n) = \max_i x_i$$

Chapter 8

Benchmarks

8.1 Graph rewrites

8.1.1 Y fusions

To test the effectiveness of Algorithm 4 for transforming graph to reduce the number of Y fusions, we benchmarked its performance on all connected graphs with less than 10 vertices. For every graph we calculated a minimum path cover before and after applying Algorithm 4 and recorded the results in Table 8.1.

Additionally, we were able to experimentally verify that greedily locally complementing vertices whenever it decrease the total number of edges never increases the size of the minimum path cover when the graph has less than 9 vertices. There is does exist a graph with 9 vertices however that fails to hold.

We can see that on average Algorithm 4 reduces the number of required fusions to construct the graph state by more than 50%. Though the data is limited, it shows a trend that that the algorithm becomes less effective at reducing fusions as the size of the graph increases. It is therefore possible that it will cease to be an effective heuristic beyond a certain graph size.

Vertices	No. Fusions			No. Edges			No. Nodes		
	Before	After	Perc	Before	After	Perc	Before	After	Perc
3	0.50	0.00	0%	2.50	2.00	80%	3.00	3.00	100%
4	1.33	0.50	38%	4.17	3.17	76%	4.00	4.00	100%
5	2.38	0.90	38%	6.19	4.57	74%	5.00	5.00	100%
6	3.71	1.50	40%	8.49	6.28	74%	6.00	6.19	103%
7	5.37	2.17	40%	11.20	8.29	74%	7.00	7.46	107%
8	7.53	3.19	42%	14.41	10.83	75%	8.00	8.92	112%
9	10.28	4.53	44%	18.22	13.92	76%	9.00	10.57	117%

Table 8.1: *Change in the average number of fusions, edges, and vertices before and after applying Algorithm 4 to a graph state.*

To analyse the performance on larger graph, we ran Algorithm 4 on randomly generated graphs with different levels of sparsity. A graph with edge density p means that each edge has probability p of being present in the graph. We repeated the experiment for graphs with 20 vertices and 50 vertices.

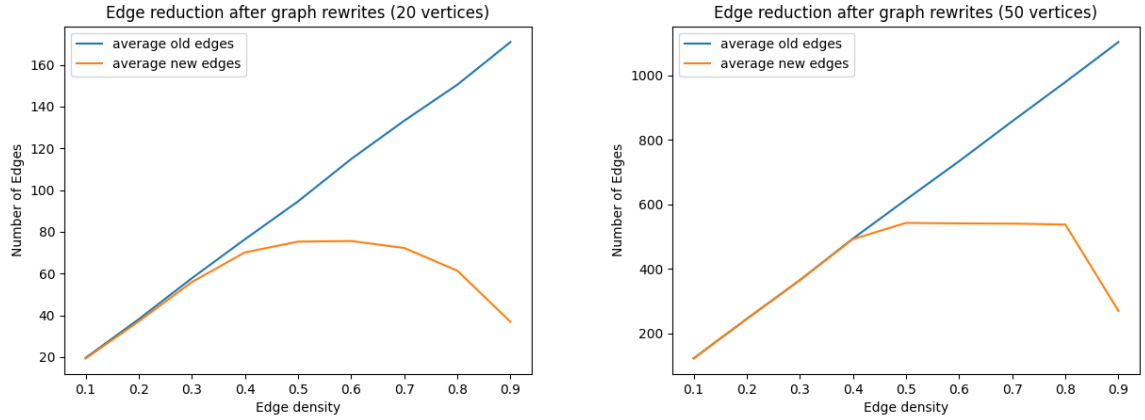


Figure 8.1: *Average reduction in the number of edges after applying Algorithm 4.*

It is clear that the algorithm is most effective on dense graphs. This is because local complementations are only performed when the number of edges strictly decreases. In fact, using this simple procedure any graph can be reduced to one with density less than 0.5. For dense graphs, we see a much greater reduction in the number of edges due to tendency for local complementations to significantly reduce the number of edges. This is most easily seen in the case of complete graphs where locally complementing any vertex will transform the graph into a star graph — the sparsest connected graph.

It is also worth pointing out that the graphs with 50 vertices were not reduced as much as the graphs on 20 vertices, and they appear to plateau around an edge density of 0.5. This could be due to the algorithm being caught in local minima which may be mitigated by more advanced algorithms such as simulated annealing.

8.1.2 X fusions

Here we applied the graph rewriting procedure for reducing X fusions (Algorithm 1) on all graphs with a fixed number of vertices and found the average number of fusions required to implement the resulting graph. We display results from two variations of our algorithm, one using a greedy approach to perform local complementations and one using simulated annealing. Alongside these results we have included the theoretical optimum that could be obtained by using the most efficient rewrites.

Vertices	Before Optim	Greedy	Sim. Annealing	Theoretical Optimum
3	0.50	0.00	0.00	0.00
4	1.50	0.50	0.33	0.33
5	2.52	0.71	0.71	0.52
6	4.14	1.67	1.50	1.05
7	5.98	2.80	2.46	1.65

Table 8.2: *Average minimum number of X fusions required to implement a graph state before and after applying the graph rewrites.*

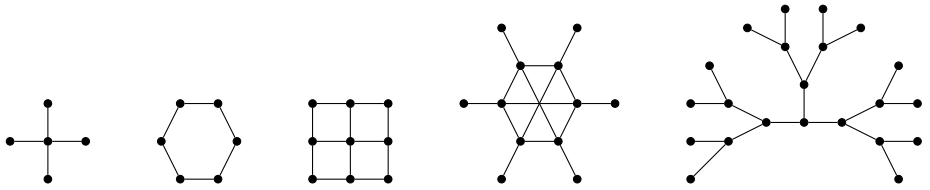
We can see that the simulated annealing approach outperforms the greedy approach in all situations which is to be expected. On average the number of fusions reduces by

significantly for smaller graphs but begins to move closer to 50% for larger graphs which reflects the results from Y fusions case.

8.2 Approximation Algorithms

We have implemented the heuristic trail cover finding algorithms from Chapter 5 and benchmarked their performance by running then on common error correcting codes to compare how the graph rewrites reduce the number of fusions.

Here we use common error correcting codes, namely star graphs [53, 54] — which are used in the original formulation of FBQC [24] — lattices [55], trees [56, 57] and repeaters [58].



Graph	Y	X	XY
6-vertex star	3	2	2
12-vertex star	9	5	5
18-vertex star	15	8	8
24-vertex star	21	11	11
(3,3)-lattice	4	5	4
(4,4)-lattice	9	12	9
(5,5)-lattice	16	21	16
(6,6)-lattice	25	32	25
(2,2,2)-tree	7	6	4
(3,3,3)-tree	26	13	13
(4,4,4)-tree	63	41	33
3-repeater	13	12	12
4-repeater	26	24	24
6-repeater	63	60	60

Table 8.3: *Benchmarks for unbounded length resource states*

This table illustrates well the various scenarios in which Y fusions can outperform X fusions and vice versa. We can see quite clearly that X fusions greatly outperform Y fusions on the star graphs due to the high degree of the central vertex of the graph. However in the case of lattices, Y fusions outperform X fusions since there is a Hamiltonian path of the graph which means the minimum path cover of the graph consists of a single path, whereas any trail decomposition must contain many trails due to the high number of odd vertices. It is also worth mentioning that we can see that the minimum trail covers we find do no worse than either of the trail decompositions or path covers which is to be expected.

Since most error correcting codes are sparse, there were no opportunities to apply effective graph rewrites except in the case of k-repeater codes which contain a complete subgraph of size $2k$ which can be locally complemented to greatly reduce the number of fusions required to implement it.

Graph	Before reduction			After Reduction		
	Y	X	XY	Y	X	XY
3-repeater	13	12	12	8	9	7
4-repeater	26	24	24	12	13	10
6-repeater	63	60	60	20	21	16

Table 8.4: *Fusions required to implement repeater codes before and after applying graph rewrites.*

We remark that repeater graphs would be benefit from n-ary Y fusions which apply a Y fusion between all nodes with probability $2^{-(n-1)}$ of success. This is contrast to manually performing $\frac{1}{2}n(n-1)$ separate Y fusions or introducing a node and locally complementing to get n edges which would have a probability of success of 2^{-n}. We leave an analysis of n-ary fusions as future work.

Graph	Y	X	XY
6-vertex star	3	2	2
12-vertex star	9	5	5
18-vertex star	15	8	8
24-vertex star	21	11	11
(3,3)-lattice	6	9	7
(4,4)-lattice	14	21	15
(5,5)-lattice	24	37	26
(6,6)-lattice	36	56	42
(2,2,2)-tree	8	8	5
(3,3,3)-tree	27	19	19
(4,4,4)-tree	64	49	36
3-repeater	15	20	18
4-repeater	28	39	35
6-repeater	66	94	84

Table 8.5: *Benchmarks with $L = 2$*

Observer that X fusions does far worse for smaller resource states. This is because it is far more economical to have Y fusions introduce an edge which is already half the size of the resource state. A linear graph state of length 5 can be implemented with 2 Y fusions and 3 1-trails, but requires 4 X fusions when using 1-trails as illustrated below.

Y fusions

X fusions

In general, a line of length $|E|$ edges could be implemented with $|E|/2$ trails (ideally), but $|E|/3$ paths with Y fusion. Hence in the case of many graphs that contain long lines

such as lattices and repeaters, the number of X fusion requires is approximately 50% higher than Y fusions.

Since our XY algorithm is built on top of the algorithm for X fusions, it tends to fair worse in these cases as well, resulting in a fusion network with slightly higher fusions.

Chapter 9

Future work

One simple way to reduce the number of photons required would be to use n-ary fusions. n-ary X fusion for example has the same probability of success of normal 2-ary fusions, but fusing n nodes together with one n-ary fusion required n photons, whereas with normal 2-ary fusion it would require a total of $2n$ photons. However the architecture described in Chapter 7 would need to change significantly to support these fusions and it would impact the time photons spend in delay lines which may increase the overall error of the compilation pipeline. n-ary Y fusions are able to implement the complete graph on n nodes with probability 2^{-n+1} which outperforms the normal 2-ary Y fusion case which has probability 2^{-n} of success (by using a Z-deleted node), though it suffers from the same architectural concerns as the n-ary X fusion case.

The complexity classes of the original problems we defined in Chapter 3 regarding minimising fusions in fusion networks, that is `MinXYFusionNetwork`, `MinBoundedXYFusionNetwork`, and `MinBoundedPhotonXYFusionNetwork` remain unknown despite knowing that the problem of finding minimum trail covers in NP-hard. We conjecture that these problem are also NP-hard but have not yet found an suitable argument to support it.

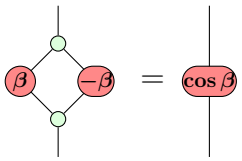
Recent fusion-based architectures include the ability to locally complement a resource state before fusions are applied which may yield fusion networks with fewer total fusions [27, 28]. As a trivial example, any graph state that can be reach from a linear graph state through local complementation could be realised with zero fusions in this architecture. This is incompatible with the architecture detailed in Chapter 7 and the definition of fusion network in Chapter 2 and so further work required to extend the results to this case.

Appendix A

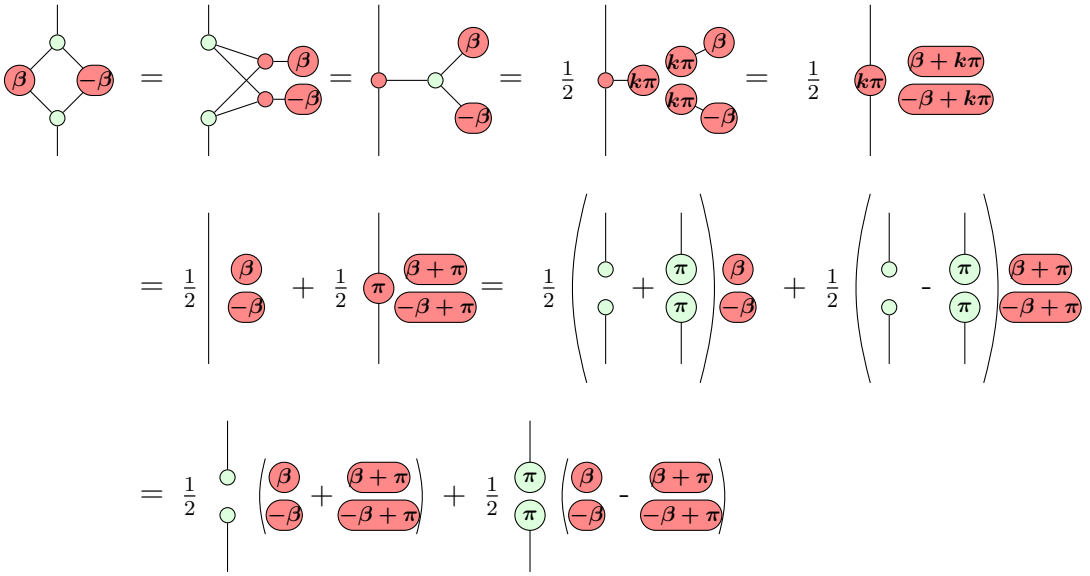
Probability of Fusion success

Here we will proof the probability of fusion success for any general type of fusion used in a FBQC protocol. The proof is forumated in the ZX calculus which we will not introduce and instead refer the reader to [59] for a more thorough introduction.

Lemma 7. *For any angle* $\beta \in [0, 2\pi)$.



Proof. We can simplify the diagram as follows. Here we use an implicit summation notation where every term containing the value of k is actually a sum over $k \in \{0, 1\}$.

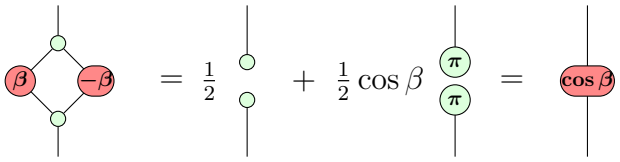


Then the last constant simplifies from the fact that

and hence

$$\begin{aligned}
\beta \\ -\beta
+
\beta+\pi \\ -\beta+\pi
&= \frac{1}{4}(1+e^{i\beta})(1+e^{-i\beta}) - \frac{1}{4}(1+e^{i(\beta+\pi)})(1+e^{i(-\beta+\pi)}) \\
&= \frac{1}{2}(e^{i\beta}+e^{-i\beta}) \\
&= \cos\beta
\end{aligned}$$

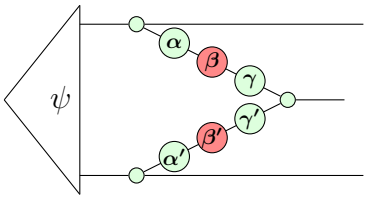
Therefore



□

In our case we consider the repeat until success model where the two photons we are fusing have output wires, this is necessary because if fusion fails then we need to retry it.

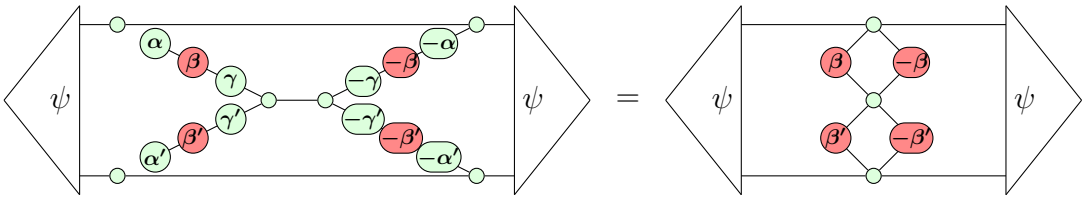
Proposition 13. *Consider the diagram below where U and V are unitary operations with Euler decompositions $U = Z(\gamma)X(\beta)Z(\alpha)$ and $V = Z(\gamma')X(\beta')Z(\alpha')$, and the inner Z spider is a probabilistic operation referred to as fusion.*



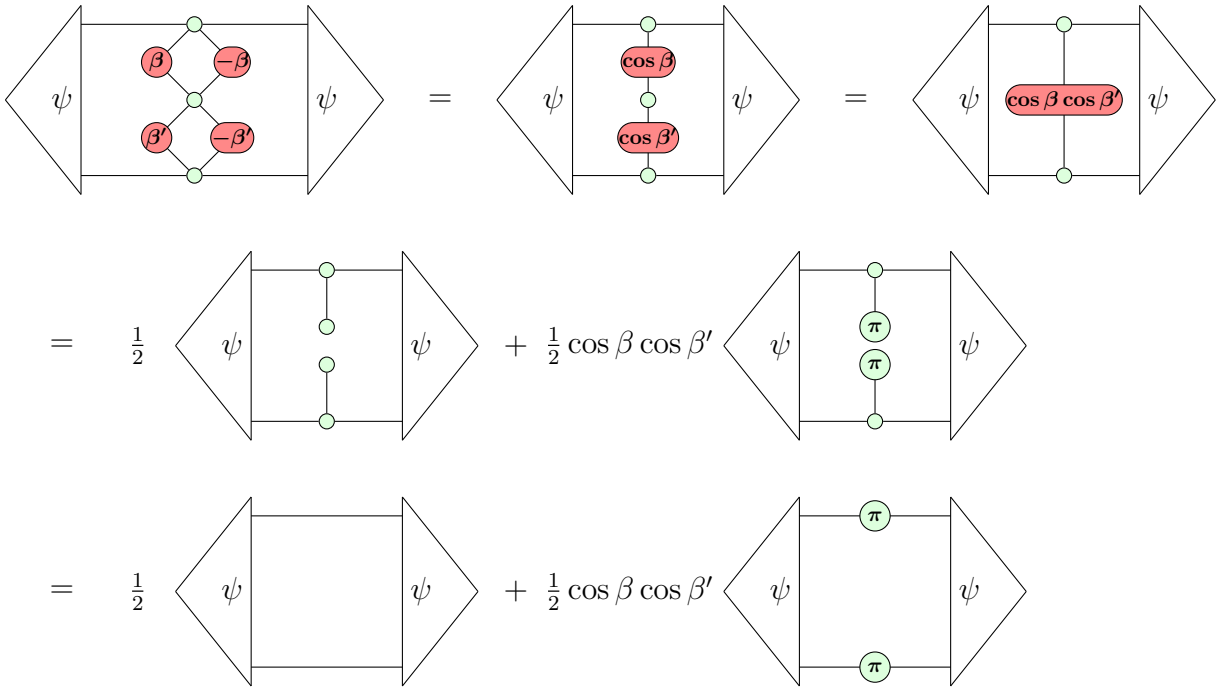
Then the probability of fusion success given the state $|\psi\rangle$ is

$$P(\psi) = \frac{1}{2} + \frac{1}{2}\cos\beta\cos\beta' \langle\psi|Z\otimes Z|\psi\rangle$$

Proof. The inner Z spider is a (non-deterministic?) projector and so discarding the outputs and undoubling the wires gives us the probability of the projector (and hence fusion) succeeding.



Then apply Lemma 7 to obtain



Since we $|\psi\rangle$ is normalised, the first term in the sum equals the constant $\frac{1}{2}$. Therefore the probability of fusion success for the given state $|\psi\rangle$ is

$$P_\psi = \frac{1}{2} + \frac{1}{2}\cos\beta\cos\beta'\langle\psi|Z\otimes Z|\psi\rangle.$$

□

Corollary 2. *The angles $\alpha, \gamma, \alpha', \gamma'$ do not affect the probability of fusion success.*

Corollary 3. *Write $|\psi\rangle = a_0|00\rangle + a_1|01\rangle + a_2|10\rangle + a_3|11\rangle$. Then the probability of fusion success is always 50% for all angles β and β' if and only if $|a_0|^2 + |a_3|^2 = |a_1|^2 + |a_2|^2$.*

Proof. Observe that $\langle\psi|Z\otimes Z|\psi\rangle = |a_0|^2 - |a_1|^2 - |a_2|^2 + |a_3|^2 = 0$. Therefore $P(\psi) = \frac{1}{2}$ □

Corollary 4. *Every graph state has 50% chance of fusion success regardless of β or β'.*

Proof. Every entry in the vector corresponding to a graph state has equal modulus, therefore Corollary 3 applies. □

Corollary 5. *The probability of fusion success is constant for all inputs if any only if either $\beta = \frac{\pi}{2}$ or $\beta' = \frac{\pi}{2}$, and the probability is 50%.*

Proof. $P(\psi)$ can only be constant for all inputs if the term $\cos\beta\cos\beta'\langle\psi|Z\otimes Z|\psi\rangle$ is constant. Since $\langle\psi|Z\otimes Z|\psi\rangle$ is not constant for all inputs $|\psi\rangle$ and therefore $\cos\beta\cos\beta' = 0$ which implies either $\beta = \frac{\pi}{2}$ or $\beta' = \frac{\pi}{2}$. □

There are some limitations on the fusions we consider for implementation since it is preferred to be "green fusions", where if there is fusion failure (something I haven't discussed yet) then it should not destroy the rest of the graph.

Corollary 6. *Green fusion succeeds with constant probability 50% for all inputs.*

Proof. Green fusion is a special type of fusion where $\beta = \beta' = \frac{\pi}{2}$. □

Theorem 2. *The probability of X and Y fusion success when executing on a fusion network is 50%.*

Bibliography

[1] Wei Luo et al. "Recent progress in quantum photonic chips for quantum communication and internet". In: *Light: Science & Applications* 12.1 (July 14, 2023), p. 175. DOI: 10.1038/s41377-023-01173-8. URL: https://doi.org/10.1038/s41377-023-01173-8.

[2] Koen Alexander et al. *A manufacturable platform for photonic quantum computing*. 2024. arXiv: 2404.17570 [quant-ph]. URL: https://arxiv.org/abs/2404.17570.

[3] Pieter Kok et al. "Linear optical quantum computing with photonic qubits". In: *Rev. Mod. Phys.* 79 (1 2007), pp. 135–174. DOI: 10.1103/RevModPhys.79.135. URL: https://link.aps.org/doi/10.1103/RevModPhys.79.135.

[4] S. Bogdanov et al. "Material platforms for integrated quantum photonics". In: *Opt. Mater. Express* 7.1 (2017), pp. 111–132. DOI: 10.1364/OME.7.000111. URL: https://opg.optica.org/ome/abstract.cfm?URI=ome-7-1-111.

[5] Jeremy L. O'Brien, Akira Furusawa, and Jelena Vučković. "Photonic quantum technologies". In: *Nature Photonics* 3.12 (Dec. 1, 2009), pp. 687–695. DOI: 10.1038/nphoton.2009.229. URL: https://doi.org/10.1038/nphoton.2009.229.

[6] D. Istrati et al. "Sequential generation of linear cluster states from a single photon emitter". In: *Nature Communications* 11.1 (Oct. 30, 2020), p. 5501. DOI: 10.1038/s41467-020-19341-4. URL: https://doi.org/10.1038/s41467-020-19341-4.

[7] Dmitry B. Uskov et al. "Resource-efficient generation of linear cluster states by linear optics with postselection". In: *Journal of Physics B: Atomic, Molecular and Optical Physics* 48 (2015). URL: https://api.semanticscholar.org/CorpusID:119452179.

[8] Stefano Paesani and Benjamin J. Brown. "High-Threshold Quantum Computing by Fusing One-Dimensional Cluster States". In: *Physical Review Letters* 131.12 (2023). ISSN: 1079-7114. DOI: 10.1103/physrevlett.131.120603. URL: http://dx.doi.org/10.1103/PhysRevLett.131.120603.

[9] Jianwei Wang et al. "Integrated photonic quantum technologies". In: *Nature Photonics* 14.5 (May 1, 2020), pp. 273–284. DOI: 10.1038/s41566-019-0532-1. URL: https://doi.org/10.1038/s41566-019-0532-1.

[10] Giovanni de Felice, Boldizsár Poór, and Lia Yeh. "Fusion and flow: formal protocols to reliably build photonic graph states". In: *ArXiv* (2024). DOI: http. URL: http.

[11] Nicolas Maring et al. "A versatile single-photon-based quantum computing platform". In: *Nature Photonics* (Mar. 26, 2024). DOI: 10.1038/s41566-024-01403-4. URL: https://doi.org/10.1038/s41566-024-01403-4.

[12] H. J. Kimble. "The quantum internet". In: *Nature* 453.7198 (2008), pp. 1023–1030. DOI: https://doi.org/10.1038/nature07127.

[13] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10th. USA: Cambridge University Press, 2011.

[14] Dan E. Browne, Elham Kashefi, and Simon Perdrix. *Computational depth complexity of measurement-based quantum computation*. 2009. arXiv: 0909.4673 [quant-ph]. URL: https://arxiv.org/abs/0909.4673.

[15] Sara Bartolucci et al. *Fusion-based quantum computation*. 2021. arXiv: 2101.09310.

[16] Brendan Pankovich et al. *Flexible entangled state generation in linear optics*. 2023. arXiv: 2310.06832 [quant-ph].

[17] Deepesh Singh, Austin P. Lund, and Peter P. Rohde. *Optical cluster-state generation with unitary averaging*. 2022. arXiv: 2209.15282 [quant-ph].

[18] Mercedes Gimeno-Segovia, Terry Rudolph, and Sophia E. Economou. "Deterministic Generation of Large-Scale Entangled Photonic Cluster State from Interacting Solid State Emitters". In: *Phys. Rev. Lett.* 123 (7 2019), p. 070501. DOI: 10.1103/PhysRevLett.123.070501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.123.070501.

[19] Chao-Wei Yang et al. "Sequential generation of multiphoton entanglement with a Rydberg superatom". In: *Nature Photonics* 16.9 (Sept. 1, 2022), pp. 658–661. DOI: 10.1038/s41566-022-01054-3. URL: https://doi.org/10.1038/s41566-022-01054-3.

[20] Philip Thomas et al. "Efficient generation of entangled multiphoton graph states from a single atom". In: *Nature* 608.7924 (Aug. 1, 2022), pp. 677–681. DOI: 10.1038/s41586-022-04987-5. URL: https://doi.org/10.1038/s41586-022-04987-5.

[21] Jean-Claude Besse et al. "Realizing a deterministic source of multipartite-entangled photonic qubits". In: *Nature Communications* 11.1 (Sept. 28, 2020), p. 4877. DOI: 10.1038/s41467-020-18635-x. URL: https://doi.org/10.1038/s41467-020-18635-x.

[22] N. Coste et al. "High-rate entanglement between a semiconductor spin and indistinguishable photons". In: *Nature Photonics* 17.7 (July 1, 2023), pp. 582–587. DOI: 10.1038/s41566-023-01186-0. URL: https://doi.org/10.1038/s41566-023-01186-0.

[23] I. Schwartz et al. "Deterministic generation of a cluster state of entangled photons". In: *Science* 354.6311 (2016), pp. 434–437. DOI: 10.1126/science.aah4758. eprint: https://www.science.org/doi/pdf/10.1126/science.aah4758. URL: https://www.science.org/doi/abs/10.1126/science.aah4758.

[24] Sara Bartolucci et al. *Fusion-based quantum computation*. 2021. arXiv: 2101.09310 [quant-ph].

[25] Sara Bartolucci et al. *Switch networks for photonic fusion-based quantum computing*. 2021. arXiv: 2109.13760 [quant-ph]. URL: https://arxiv.org/abs/2109.13760.

[26] Hector Bombin et al. *Interleaving: Modular architectures for fault-tolerant photonic quantum computing*. 2021. arXiv: 2103.08612 [quant-ph]. URL: https://arxiv.org/abs/2103.08612.

[27] Felix Zilk et al. "A compiler for universal photonic quantum computers". In: *2022 IEEE/ACM Third International Workshop on Quantum Computing Software (QCS)*. IEEE, 2022. DOI: 10.1109/qcs56647.2022.00012. URL: http://dx.doi.org/10.1109/QCS56647.2022.00012.

[28] Seok-Hyung Lee and Hyunseok Jeong. "Graph-theoretical optimization of fusion-based graph state generation". In: *Quantum* 7 (2023), p. 1212. ISSN: 2521-327X. DOI: 10.22331/q-2023-12-20-1212. URL: https://doi.org/10.22331/q-2023-12-20-1212.

[29] Soh Kumabe, Ryuhei Mori, and Yusei Yoshimura. *Complexity of graph-state preparation by Clifford circuits*. 2024. arXiv: 2402.05874 [quant-ph]. URL: https://arxiv.org/abs/2402.05874.

[30] Herbert Fleischner. *Eulerian Graphs and Related Topics: Part 1, Volume 2*. Annals of Discrete Mathematics, 1991. URL: https://api.semanticscholar.org/CorpusID:118183786.

[31] Oystein Ore and Robin J. Wilson. *Graphs and Their Uses*. Anneli Lax New Mathematical Library. Mathematical Association of America, 1990.

[32] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. "Graphical description of the action of local Clifford transformations on graph states". In: *Physical Review A* 69.2 (Feb. 2004). ISSN: 1094-1622. DOI: 10.1103/physreva.69.022316. URL: http://dx.doi.org/10.1103/PhysRevA.69.022316.

[33] André Bouchet. "An efficient algorithm to recognize locally equivalent graphs". In: *Combinatorica* 11.4 (1991), pp. 315–329. DOI: 10.1007/BF01275668. URL: https://doi.org/10.1007/BF01275668.

[34] Axel Dahlberg, Jonas Helsen, and Stephanie Wehner. *The complexity of the vertex-minor problem*. 2019. arXiv: 1906.05689 [math.CO]. URL: https://arxiv.org/abs/1906.05689.

[35] Robert Raussendorf, Daniel Browne, and Hans Briegel. "The one-way quantum computer–a non-network model of quantum computation". In: *Journal of Modern Optics* 49.8 (July 2002), pp. 1299–1306. ISSN: 1362-3044. DOI: 10.1080/09500340110107487. URL: http://dx.doi.org/10.1080/09500340110107487.

[36] Daniel E Browne et al. "Generalized flow and determinism in measurement-based quantum computation". In: *New Journal of Physics* 9.8 (Aug. 2007), pp. 250–250. ISSN: 1367-2630. DOI: 10.1088/1367-2630/9/8/250. URL: http://dx.doi.org/10.1088/1367-2630/9/8/250.

[37] Will Simmons. "Relating Measurement Patterns to Circuits via Pauli Flow". In: *ArXiv* abs/2109.05654 (2021). URL: https://api.semanticscholar.org/CorpusID:237491808.

[38] Daniel E. Browne and Terry Rudolph. "Resource-Efficient Linear Optical Quantum Computation". In: *Physical Review Letters* 95.1 (2005). ISSN: 1079-7114. DOI: 10.1103/physrevlett.95.010501. URL: http://dx.doi.org/10.1103/PhysRevLett.95.010501.

[39] Matthias J. Bayerbach et al. “Bell-state measurement exceeding 50% success probability with linear optics”. In: *Science Advances* 9.32 (2023), eadf4080. DOI: 10.1126/sciadv.adf4080. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.adf4080. URL: https://www.science.org/doi/abs/10.1126/sciadv.adf4080.

[40] Hezi Zhang et al. *OnePerc: A Randomness-aware Compiler for Photonic Quantum Computing*. 2024. arXiv: 2403.01829 [quant-ph]. URL: https://arxiv.org/abs/2403.01829.

[41] Tommy McElvanney and Miriam Backens. “Complete Flow-Preserving Rewrite Rules for MBQC Patterns with Pauli Measurements”. In: *Electronic Proceedings in Theoretical Computer Science* 394 (Nov. 2023), pp. 66–82. ISSN: 2075-2180. DOI: 10.4204/eptcs.394.5. URL: http://dx.doi.org/10.4204/EPTCS.394.5.

[42] Richard M. Karp. “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9. URL: https://doi.org/10.1007/978-1-4684-2001-2_9.

[43] “List of graphs for which the Hamiltonian path problem is in P”. In: (). URL: https://www.graphclasses.org/classes/problem_Hamiltonian_path.html.

[44] Shlomo Moran, Ilan Newman, and Yaron Wolfsthal. “Approximation algorithms for covering a graph by vertex-disjoint paths of maximum total weight”. In: *Networks* 20 (1990), pp. 55–64. URL: https://api.semanticscholar.org/CorpusID:8146642.

[45] Silvio Micali and Vijay V. Vazirani. “An O(v—v— c —E—) algoithm for finding maximum matching in general graphs”. In: *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*. 1980, pp. 17–27. DOI: 10.1109/SFCS.1980.12.

[46] Kenya Kobayashi et al. “Path Cover Problems with Length Cost”. In: *Algorithmica* 85.11 (Nov. 1, 2023), pp. 3348–3375. DOI: 10.1007/s00453-023-01106-2. URL: https://doi.org/10.1007/s00453-023-01106-2.

[47] Norman L. Biggs, E. Keith Lloyd, and Robin J. Wilson. *Graph Theory 1736-1936*. 1976. URL: https://api.semanticscholar.org/CorpusID:118963253.

[48] David P. Sumner. “Graphs with 1-Factors”. In: *Proceedings of the American Mathematical Society* 42.1 (1974), pp. 8–12. ISSN: 00029939, 10886826. URL: http://www.jstor.org/stable/2039666 (visited on 08/06/2024).

[49] Vijay V. Vazirani. “Approximation Algorithms”. In: Springer Berlin, Heidelberg, 2001, p. 380. DOI: https://doi.org/10.1007/978-3-662-04565-7.

[50] M. R. Garey, D. S. Johnson, and R. Endre Tarjan. “The Planar Hamiltonian Circuit Problem is NP-Complete”. In: *SIAM Journal on Computing* 5.4 (1976), pp. 704–714. DOI: 10.1137/0205049. eprint: https://doi.org/10.1137/0205049. URL: https://doi.org/10.1137/0205049.

[51] Pouya Baniasadi et al. *A new benchmark set for Traveling salesman problem and Hamiltonian cycle problem*. 2018. arXiv: 1806.09285 [cs.DS]. URL: https://arxiv.org/abs/1806.09285.

[52] Vincent Danos, Elham Kashefi, and Prakash Panangaden. *The Measurement Calculus*. 2004. arXiv: quant-ph/0412135 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/0412135.

[53] Ying Li et al. "Fault Tolerant Quantum Computation with Nondeterministic Gates". In: *Phys. Rev. Lett.* 105 (25 Dec. 2010), p. 250502. DOI: 10.1103/PhysRevLett.105.250502. URL: https://link.aps.org/doi/10.1103/PhysRevLett.105.250502.

[54] Srikrishna Omkar et al. "All-Photonic Architecture for Scalable Quantum Computing with Greenberger-Horne-Zeilinger States". In: *PRX Quantum* 3 (3 July 2022), p. 030309. DOI: 10.1103/PRXQuantum.3.030309. URL: https://link.aps.org/doi/10.1103/PRXQuantum.3.030309.

[55] Robert Raussendorf, Daniel E. Browne, and Hans J. Briegel. "Measurement-based quantum computation on cluster states". In: *Phys. Rev. A* 68 (2 Aug. 2003), p. 022312. DOI: 10.1103/PhysRevA.68.022312. URL: https://link.aps.org/doi/10.1103/PhysRevA.68.022312.

[56] Ying Li et al. "Resource Costs for Fault-Tolerant Linear Optical Quantum Computing". In: *Phys. Rev. X* 5 (4 Oct. 2015), p. 041007. DOI: 10.1103/PhysRevX.5.041007. URL: https://link.aps.org/doi/10.1103/PhysRevX.5.041007.

[57] Michael Varnava, Daniel E. Browne, and Terry Rudolph. "Loss Tolerance in One-Way Quantum Computation via Counterfactual Error Correction". In: *Phys. Rev. Lett.* 97 (12 Sept. 2006), p. 120501. DOI: 10.1103/PhysRevLett.97.120501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.97.120501.

[58] Koji Azuma, Kiyoshi Tamaki, and Hoi-Kwong Lo. "All-photonic quantum repeaters". In: *Nature Communications* 6.1 (Apr. 15, 2015), p. 6787. DOI: 10.1038/ncomms7787. URL: https://doi.org/10.1038/ncomms7787.

[59] John van de Wetering. "ZX-calculus for the working quantum computer scientist". In: 1 (2020), p. 75. DOI: https://doi.org/10.48550/arXiv.2012.13966. arXiv: 2012.13966.