Master's Thesis (Academic Year 2023)

Optimizing Space-Time Overhead for Fault-Tolerant Graph State Generation

Keio University

Graduate School of Media and Governance

Sitong Liu

Abstract of Master's Thesis Academic Year 2023

Optimizing Space-Time Overhead for Fault-Tolerant Graph State Generation

The development of a reliable quantum computer has been a major research focus in recent decades. One approach to achieving fault-tolerant quantum computation is through measurement-based quantum computing using fault-tolerant graph states as computational resources. The generation of fault-tolerant graph states, a specialized form of entangled quantum state, entails sophisticated fault-tolerant logical operations, leading to considerable costs that are likely to dominate the entire computing process. This imposes substantial demands on quantum devices and presents architectural constraints, especially when dealing with large-scale graph states.

This thesis focuses on the Substrate Scheduler, a compiler module that aims to optimize fault-tolerant graph state compilation by minimizing the space-time overhead. It uses the stabilizer formalism for improved generation optimization. We study how different layout designs affect the generation of fault-tolerant graph states, and provides an enhanced qubit allocation method to further improve the process.

We compare the module's performance with non-optimized results and an alternative approach to generating graph states using CZ gates. The results demonstrate that the module achieves the lowest space-time overhead among these methods, surpassing the exponential space-time overhead growth rate using the unoptimized approach. Additionally, it enables specific graph types to maintain a constant-level space-time overhead. These findings suggest that the architecture/technology combination has the potential to efficiently generate large-scale graph states for various applications.

Keywords: 1. Fault-tolerant quantum computing, 2. Surface code, 3.Quantum compiling, 4. Graph states,

Keio University
Graduate School of Media and Governance

Sitong Liu

Contents

[1] This section is adopted from my first-authored paper, titled "A Substrate Scheduler for Compiling Arbitrary Fault-tolerant Graph States" (2023). See [1]

[2] Sections 2.3 and 2.4 are partially adopted from my first-authored paper, titled "A Substrate Scheduler for Compiling Arbitrary Fault-tolerant Graph States" (2023). See [1]

[3]This section, along with Section 4.3.2 is based on my first-authored paper, titled "A Substrate Scheduler for Compiling Arbitrary Fault-tolerant Graph States" (2023). See [1]

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background [1]

Spanning almost half a century, the development of quantum computing, which leverages the principles of quantum mechanics, has progressed dramatically [2]. Quantum computing has the potential to solve some problems that are either impossible or extremely difficult to solve with classical computers. However, an inevitable problem with quantum computers is that they are still small and highly susceptible to noise [3]. In order to fully realize the potential of quantum computing and achieve the so-called "quantum advantage", it is of vital importance to guarantee the high-fidelity execution of large-scale quantum programs. In the long term, the pressing challenge is how to transition from the current generation of noisy quantum devices to fault-tolerant quantum computing with quantum error correction (QEC).

The number and fidelity of qubits, both physical and logical, will remain a constraint on applications for the foreseeable future. In the meantime, significant resources must be allocated to error correction, as implementing quantum error correction involves using large numbers of physical qubits to encode a single fault-tolerant logical qubit [4, 5, 6]. As such, in order to hasten the advent of the scalable, fault-tolerant quantum computing era [7], efficient compilers that can implement applications using quantum error correction with minimal space-time cost are critical.

In recent years, engineers have gradually started to focus on how to construct and optimize fault-tolerant quantum computers and the related quantum error correction protocols [8]. One problem that needs to be solved is that current research is focused either on the logical level, assuming all operations are already fault-tolerant, or on constructing fault-tolerant qubits [9], i.e., how to encode and decode them. However, there is no integrated description to guide us on how to run programs on a fault-tolerant quantum computer.

Recent work has introduced a fault-tolerant measurement-based quantum com-

[1] This section is adopted from my first-authored paper, titled "A Substrate Scheduler for Compiling Arbitrary Fault-tolerant Graph States" (2023). See [1]

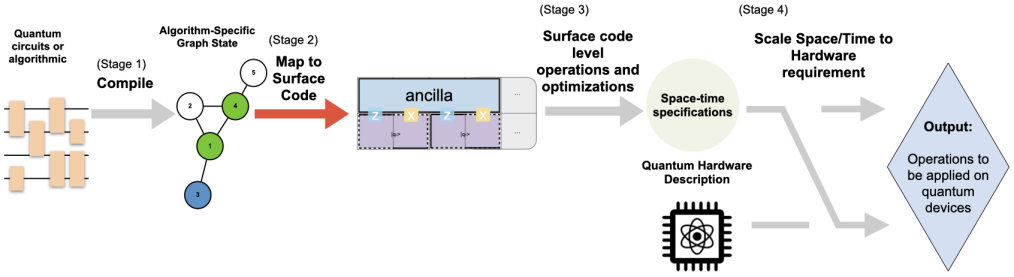


Figure 1.1: The toolchain for fault-tolerant quantum computation.

putation workflow that demonstrated arbitrary quantum circuits can be compiled into graph states that are believed to be amenable to further optimization and efficient execution as studied by Vijayan *et al.* [10]. As a generalization of cluster states, graph states [11, 12] have a variety of applications in quantum information, in particular as algorithmic resources in the context of measurement-based quantum computing (MBQC) [13, 14]. Recognizing their importance and flexibility, a group of researchers has proposed an end-to-end compilation toolchain based on the concept of fault-tolerant graph states, named **benchq** [2], which consists of four stages (see Figure 1.1). In the toolchain, stage 1 converts the quantum circuit as input into a circuit/algorithm-specific graph state (i.e., the work of Jabalizer [15]); the second stage involves compiling the graph state into a set of operations that can be executed on the surface code level; stage 3 involves operations and optimization at the surface code level, and after final integration with the hardware, operations can be executed to generate the fault-tolerant graph state, and finally conduct the fault-tolerant computation. The main focus of this thesis is on Stage 2, which addresses how to map algorithmic graph states onto the surface code, a crucial step in achieving fault tolerance.

In this thesis, we present the Substrate Scheduler, the compiler module that performs a fault-tolerant compilation of a graph state from the adjacency matrix to an optimized schedule of stabilizer measurements along with the logical qubit allocation.

Substrate Scheduler consists of three parts: stabilizer generator reduction, a heuristic algorithm for the vertex to qubit mapping, and scheduling of the stabilizer generator measurements. The stabilizer formalism [16] (see Section 2.4), which was originally developed for the analysis and design of quantum error correction codes, is used as it can provide further optimization of the generation process.

Substrate Scheduler is based on surface code [17, 18, 19], one of the most promising quantum error correction codes, with lattice surgery [20], using the rules introduced by Litinski's paper, "A Game of Surface Codes"(GoSC) [21] (see Section 2.4). GoSC uses a space-time resource model for evaluating performance. The primary space-

[2] `https://github.com/zapatacomputing/benchq`

time trade-off we observe is between the number of logical qubits (measured in units of "tiles") and the code cycles (known as "Tocks") spent in generating the graph state. In addition, three distinct layout designs are proposed in this thesis and are compared to find the minimal overhead. The primary goal of this thesis is to minimize the space-time overhead of generating the fault-tolerant graph states.

1.2 Research Contribution

The contributions of this thesis are summarized as follows:

- This thesis provides an overview of two methods for generating fault-tolerant quantum graph states based on surface codes. It discovers that utilizing stabilizer parity checks on the surface code requires a shorter minimum time step compared to preparing the graph state with CZ gates.
- This thesis introduces two evaluation methods, namely space-time cost, and the data-to-ancilla ratio for assessing the design of fault-tolerant qubits layouts. Building upon the operational principles of surface code presented in [21], three distinct layout designs are proposed and their data-ancilla ratios and resource needed are compared.
- This thesis investigates the impact of mapping methods under the constraint of a 2-tile-one-qubit one-bus layout, specifically analyzing their effects on time cost and space-time overhead. It summarizes the heuristic algorithms proposed in [1] and suggests further improvements.
- This thesis presents a procedure for mapping fault-tolerant graph states onto surface code-level operations. It utilizes optimization techniques including stabilizer generator reduction, qubit allocation, and operation scheduling to minimize time costs and space-time overhead. Evaluation shows that the procedure achieves a space-time overhead lower than the exponential growth rate of unoptimized methods, allowing specific graph types to maintain a constant-level space-time overhead.

1.3 Thesis Structure

The thesis is structured as follows: Chapter 2 serves as an introduction to quantum information processing and presents key concepts such as graph states, stabilizer formalism, and basic surface code operations that are utilized in this study as a preliminary to support the readers with minimal knowledge. Chapter 3 defines the problems addressed in this thesis and introduces various works related to fault-tolerant compilation workflows. Chapter 4 offers an overview of the Substrate Scheduler, a

compiler tool designed for fault-tolerant graph state compilation. In Chapter 5, the proposed tool is evaluated in its feasibility, use cases, and scalability. Lastly, Chapter 6 concludes the thesis with some discussions regarding future work.

Chapter 2

Preliminaries [1]

This chapter summarizes the basic concepts and notations of quantum computing. It begins with an exploration of the core principles of quantum information theory, along with the essential understanding of quantum error correction codes and the measurement-based quantum computing model. These concepts are indispensable for comprehending and engaging with the research presented in this thesis.

2.1 Quantum State and its Representation

2.1.1 Qubit

A quantum bit, often referred to as a *qubit*, can be understood as the quantum equivalent of the classical binary bit used in classical computers which is based on a two-state quantum-mechanical system. One of the significant differences of a qubit is its capability to exist in a superposition state, where it can represent both 0 and 1 simultaneously. This remarkable attribute arises from the quantum nature of the qubit. The qubit is the smallest unit of information used by a quantum computer.

Notation

To represent the states of qubits, the Dirac notation (also known as Bra-Ket notation) is commonly used. In the case of a single qubit, the quantum states can be expressed as linear combinations of the computational basis states,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,. \tag{2.1.1}$$

The notation $|k\rangle$ represents the possible states, known as *kets*. The coefficients α and β are complex numbers that represent the amplitudes. The computational basis states, $|1\rangle$ and $|0\rangle$, are special cases of the single qubit state where either α or β is

[1] Sections 2.3 and 2.4 are partially adopted from my first-authored paper, titled "A Substrate Scheduler for Compiling Arbitrary Fault-tolerant Graph States" (2023). See [1]

0. Another example, $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ indicates that the state $|\psi\rangle$ is in a equal superposition of the two states $|0\rangle$ and $|1\rangle$.

It is important to note that it forces the normalization condition $|\alpha|^2 + |\beta|^2 = 1$, i.e. the sum of the squared coefficients must equal 1. This holds true not only for single-qubit cases but also for multi-qubit cases.

Qubits can also be represented in terms of vectors in a Hilbert space, the vector representation of a single qubit is written as follows,

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{2.1.2}$$

where two basis states are:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Bloch Sphere

The geometric representation of a single qubit in spherical coordinates is often depicted using the *Bloch Sphere*, a unit three-dimensional sphere. Figure 2.1 provides an example illustration of the Bloch sphere.

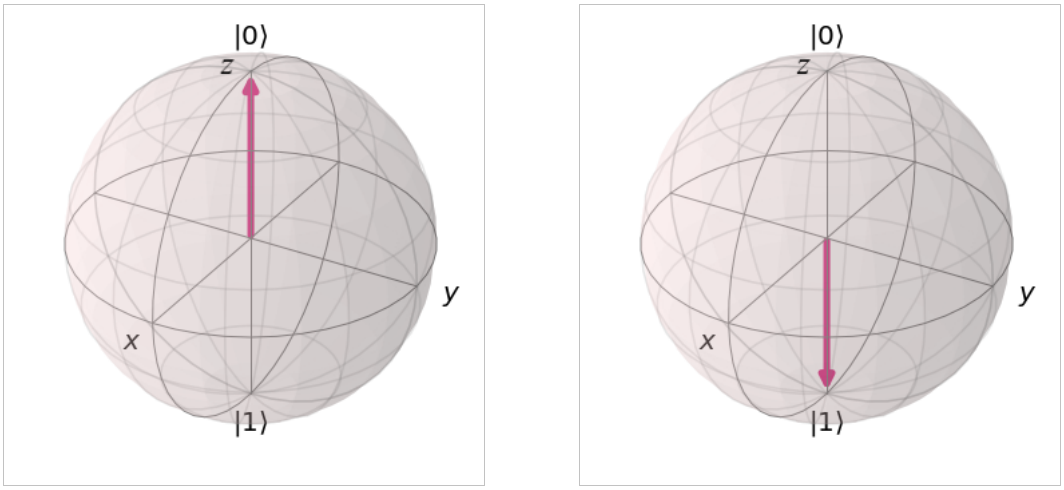


Figure 2.1: Bloch Sphere Representation of State 0 and 1

Multiple Qubits

When we extend to the multi-qubit quantum state, the multi-qubit quantum state can be constructed by tensor products of qubit states. As an example, a two qubits state $|01\rangle$ can be written as

$$|0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \tag{2.1.3}$$

where the resulting Hilbert space is the tensor product of two Hilbert spaces.

Similarly, an N-qubit system can be in a superposition of 2^n states simultaneously. In a more generalized way, can be written as:

$$|\psi\rangle = \sum_{i=0}^{2n-1} a_i |i\rangle \tag{2.1.4}$$

where $\sum_{i=0}^{2n-1} |a_i|^2 = 1$.

2.1.2 Engtanglement

Consider a quantum state involving two qubits

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{2.1.5}$$

When this quantum state is measured, we observe either $|00\rangle$ or $|11\rangle$ with an equal probability of 50%, and it cannot be observed as $|01\rangle$ or $|10\rangle$. Upon measuring the first qubit (e.g., the right one) and obtaining $|0\rangle$ or $|1\rangle$, the state of the second qubit is determined as $|0\rangle$ or $|1\rangle$, respectively, and vice versa. This indicates that the qubits' states are not independently changing, or we can say they are correlated. This phenomenon is known as *entanglement*.

Entanglement can occur between two or more qubits. When an operation is applied to one qubit in an entangled quantum state, the state of its entangled qubit is instantaneously impacted, irrespective of the physical separation between them.

A well-known example of an entangled state is the *Bell states*, also called the *EPR pair*. There are four types of Bell states:

$$\begin{aligned} |\Phi^+\rangle &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \\ |\Phi^-\rangle &= \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \\ |\Psi^+\rangle &= \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \\ |\Psi^-\rangle &= \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle). \end{aligned} \tag{2.1.6}$$

These four states can be used as a computational basis for two-qubit states.

2.2 Operation on the Quantum States

Quantum circuits are a widely utilized model for quantum computing as they provide a complete representation of quantum computations. However, it is important to note that alongside the quantum circuit model, there exists an alternative paradigm known as the measurement-based quantum computing model, which will be introduced in Section 2.5

2.2.1 Quantum Gates

Quantum gates are unitary operators utilized to manipulate and transform quantum states, which can be mathematically represented as unitary matrices. These gates can be conceptually likened to logical gates, with the notable distinction that all quantum gates are reversible. In this context, we overlook the physical implementation details and concentrate on comprehending the operation of quantum gates through an abstract and mathematical perspective. The application of a quantum gate can be represented as a straightforward matrix-vector multiplication. In this process, the matrix representing the gate is multiplied with the vector representing the quantum state. For gate that operate on n qubits, its action can be described by a matrix of size $2^n \times 2^n$, which operates on vector of size 2^n. To apply single-qubit gates on multiple qubits, we construct multi-qubit gates by taking the tensor product of known single-qubit gates.

2.2.2 Single Qubit Gates

From a geometric perspective, single qubit gates can be viewed as rotations about a specific axis n by an angle ϕ. These rotations are commonly denoted as $R_n(\phi)$. Among them, the Pauli operators play a crucial role in quantum computing as they are both highly significant and widely used gates. To see the effect a gate has on a qubit's state vector, a simple example of a Pauli-X gate

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.2.1}$$

operating on the state $|0\rangle$ of a single qubit is

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle . \tag{2.2.2}$$

This example demonstrates that the X gate effectively interchanges the amplitudes of the states $|0\rangle$ and $|1\rangle$, resulting in a flip from $|0\rangle$ to $|1\rangle$. In the representation of the Bloch sphere (see Figure 2.1), it corresponds to rotating the quantum state by 180 degrees (π radius) about the X-axis, transitioning from one pole of the Z-axis (left in Figure 2.1) to the other pole (right in Figure 2.1).

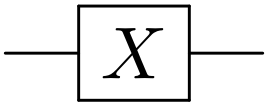


Figure 2.2: Circuit representation of a Pauli-X gate

In diagrams of the quantum circuit model, information flow occurs from left to right. The wires in the circuit symbolically represent qubits, which can be manipulated and entangled through the application of quantum gates. To facilitate calculations, it is often assumed that the initial state input into the circuit consists of all qubits initialized to the state $|0\rangle$.

Pauli-Y and Z gates are, respectively

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \tag{2.2.3}$$

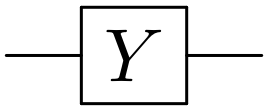


Figure 2.3: Circuit representation of a Pauli-Y gate

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.2.4}$$

where X, Y, and Z indicate the rotation axis and the rotation is π radius. The Pauli-Y

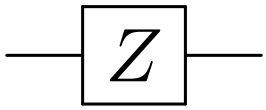


Figure 2.4: Circuit representation of a Pauli-Z gate

gate is a quantum gate that performs a rotation around the y-axis; when applied to a qubit, the Pauli-Z gate leaves the $|0\rangle$ state unchanged but apply a relative phase flip, transforming $|1\rangle$ state into $-|1\rangle$

Another important single qubit gate is the Hadamard gate

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \tag{2.2.5}$$

which plays a fundamental role in creating superposition, and transforming between

Figure 2.5: Circuit representation of an H gate

the different basis states.

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = |+\rangle \\ H|1\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = |-\rangle \\ H|+\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \\ H|-\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \end{aligned} \tag{2.2.6}$$

2.2.3 Controlled Gates

Beyond the single qubit gates, another category of gates allow us to manipulate two or more qubits simultaneously.

Controlled-NOT (X) Gate

Controlled NOT gate, also known as CNOT gate (see Figure 2.6) operates on two qubits, with one qubit serving as the control qubit and the other as the target qubit. When the control qubit is in the state $|1\rangle$, the CNOT gate applies the Pauli X gate to the target qubit.

The matrix representation of CNOT gate is

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.2.7}$$

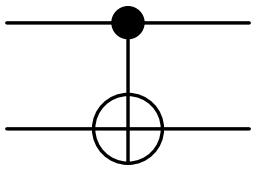


Figure 2.6: Circuit representation of a CNOT gate

Controlled-Z Gate

Similar to CNOT gate, Controlled-Z also known as the Controlled-phase or CZ gate performs the Z gate on the target qubit, when the controlled qubit's state is $|1\rangle$. The matrix representation of the CZ gate is

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}. \tag{2.2.8}$$

It is worth mentioning that the CZ gate can be constructed using two Hadamard gates and one CNOT gate, as illustrated in Figure 2.7.

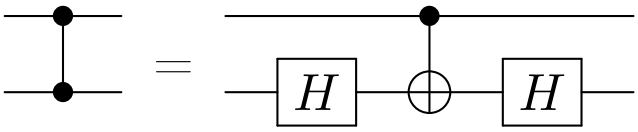


Figure 2.7: Circuit representation of a Controlled-Z gate

In addition to the quantum gates presented within this section, further commonly employed quantum gates can be found in quantum computation textbooks, such as Quantum Computation and Quantum Information authored by Nielsen and Chuang [22].

2.2.4 Measurement

Measurement (see Figure 2.8), also known as readout, plays a crucial role in quantum computing. It is inherently probabilistic, irreversible, and will cause the collapse of quantum superposition. When we refer to measurement, it often implies a measurement in the computational basis. This measurement operation causes the qubit to collapse to either the $|0\rangle$ or $|1\rangle$ state, with a probability equal to the squared modulus of the amplitude. As an simple, for $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, the probability of measuring the system to be in state $|0\rangle$ or $|1\rangle$ is 0.5.

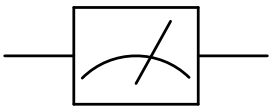


Figure 2.8: Circuit representation of measurement

Projective Measurement

Projective Measurement is based on the eigenvalues and eigenvectors of the observable being measured. For qubits, the most common observables are the Pauli matrices. Consider a collection of measurement operators M_b, where b corresponds to a computational basis state. The probability $p(b)$ of the state $|\psi\rangle$ being observed in basis state b can be expressed as

$$p(b) = \langle\psi| M_b^\dagger M_b |\psi\rangle . \tag{2.2.9}$$

The state of the system after measurement is

$$\frac{M_b |\psi\rangle}{\sqrt{\langle\psi| M_b^\dagger M_b |\psi\rangle}}. \tag{2.2.10}$$

A common projective measurement in quantum computing is the measurement in the computational basis, which uses the basis vectors $|0\rangle$ and $|1\rangle$. The measurement operators for this measurement are

$$\begin{aligned} P_0 &= |0\rangle \langle 0| \\ P_1 &= |1\rangle \langle 1| . \end{aligned} \tag{2.2.11}$$

2.3 Graph State

Graph states, which are a generalization of cluster states [11, 12] (see Figure2.9), play a vital role in Measurement-Based Quantum Computing (MBQC), an alternative quantum computational model to the circuit model (refer to Section 2.5). It is based on graph theory, providing a framework to depict the entanglement among qubits.

We begin with the notion of a *graph* $G = (V, E)$ [23], which is a pair of two sets, a vertex set $V = \{1, 2, 3, \ldots, n\}$ and an edge set $E \subseteq V^2$. The vertices correspond to the qubits in the system while edges correspond to interactions between pairs of qubits. We also denote $|V| = n$ and $|E|$ as the number of vertices and the total number of edges, respectively. Two vertices $a, b \in V$ are *adjacent* if they are connected by an edge, $\{a, b\} \in E$.

2.3.1 Cluster State

A cluster state is a special example of graph state when the underlying graph G is a $d-$ dimensional (normally two-dimensional) lattice. The cluster state is highly entangled, playing a crucial role in Measurement-Based Quantum Computing (MBQC).

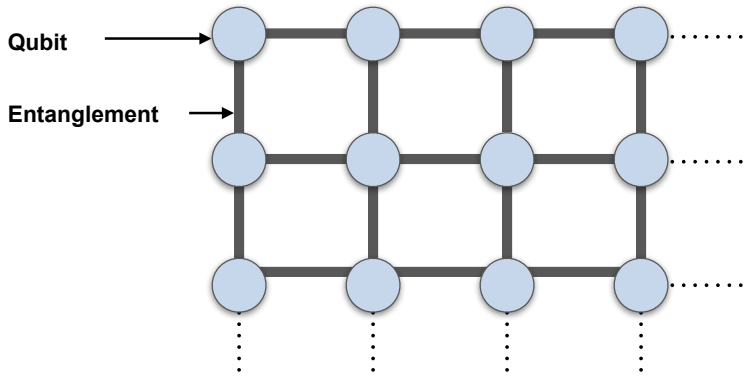


Figure 2.9: An example of a 2-dimensional cluster state.

2.3.2 Neighborhood and independent set

We frequently rely on the concept of the *neighborhood* $\mathrm{ngbr}(a)$ of a vertex $a \in V$,

$$\mathrm{ngbr}(a) = \{b \in V | \{a, b\} \in E\}. \tag{2.3.1}$$

The neighborhood is the set of vertices adjacent to a given vertex. A subset of vertices $U \subseteq V$ is *independent* if no two of its vertices are adjacent. The *maximum independent set*, $\alpha(G)$, is the largest such set, while a *maximal independent set* is an independent set to which no further vertex can be added without violating the independence condition.

2.3.3 Representation of Graph State

Commonly, graph states can be described as *undirected connected simple graphs* where only at most one edge is allowed per pair of vertices with no self-loops and there exists a path between any pair of vertices. This gives rise to the notion of an *adjacency matrix* $\Gamma = [\Gamma_{a,b}] \in \{0, 1\}^{n \times n}$ with elements

$$\Gamma_{a,b} = \begin{cases} 1, & \text{if } \{a, b\} \in E \\ 0, & \text{otherwise} \end{cases}. \tag{2.3.2}$$

For a given graph G, we can construct its corresponding graph state $|G\rangle$ by first initializing all qubits in the state $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, and for each pair of qubits which represent adjacent vertices in G, an entangling two-qubit *controlled-Z gate* (see Figure 2.7), is applied. Thus, the graph state can be represented as follows.

$$|G\rangle = \sum_{\{i,j\}\in E} CZ_{i,j} \, |+\rangle^{\otimes |V|} \tag{2.3.3}$$

Stabilizer Formalism

Since graph states are a subclass of more general stabilizer states [16], there is a compact description of a state using only n operators of length at most n, in contrast to the full state vector representation, which requires a complex amplitude for each of the 2^n basis states. The stabilizer formalism describes quantum states in terms of operators that *stabilize* the state.

A state $|\psi\rangle$ is stabilized by an operator K if $K|\psi\rangle = |\psi\rangle$, that is, the state $|\psi\rangle$ is a $+1$ eigenstate of the operator K. For example, state $|+\rangle$ is stabilized by the Pauli operator X since $X|+\rangle = |+\rangle$. We associate a multi-qubit Pauli operator with qubit i of a graph state with the following form,

$$g_i = X_i \bigotimes_{j\in \mathrm{ngbr}(i)} Z_j. \tag{2.3.4}$$

The n-qubit graph state $|G\rangle$ is then uniquely identified as the simultaneous $+1$ eigenstate of all n stabilizer operators from Eq. (2.3.4). These stabilizer operators generate an Abelian group referred to as the *stabilizer* $\mathcal{S} = \langle g_1, \ldots, g_n \rangle$. Stabilizer generators of Eq. (2.3.4) lead to an efficient description of the graph state in terms of n commuting operators.

We will make repeated use of measuring the stabilizer generators. Measurement of stabilizer generator g_i corresponds to the application of a projection operator onto the even/odd parity subspace for the stabilizer generator,

$$\Pi^{\pm}(g_i) = \frac{1}{2}(I \pm g_i). \tag{2.3.5}$$

If the measured parity is even, that is, we have projected onto the positive eigenspace of the stabilizer generator, we do not need to take further action. Projections onto the negative eigenspace can be further corrected to flip their parity, but in general, this is classically tracked without applying quantum gates during the generation process. Once the generation of the graph state is complete, in the absence of decoherence, it is clear from our previous discussion that measurement of any of the stabilizer generators of the graph state $|G\rangle$ produces a $+1$ outcome with unit probability. Similarly, projecting any initial state onto the common even parity eigenspace of all the stabilizer generators prepares the desired graph state $|G\rangle$.

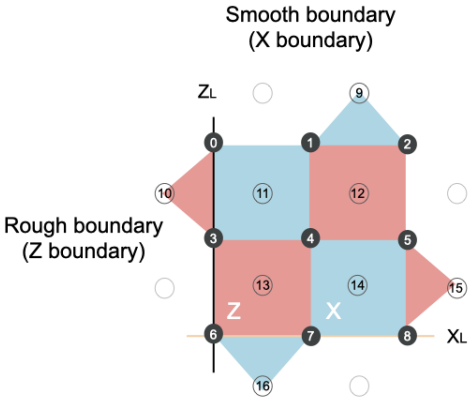
2.4 Quantum Error Correction

2.4.1 Surface Code

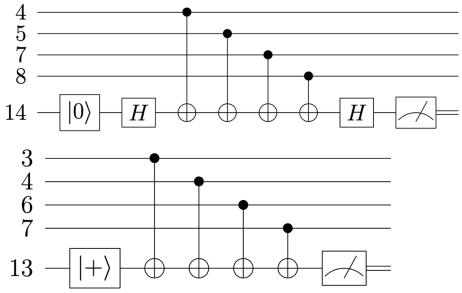
Quantum error correction codes (QECC) are employed to protect quantum states from decoherence, ensuring the accuracy of quantum computations. This thesis focuses on utilizing the surface code [17, 18, 19] as the chosen method for quantum error correction. The surface code (see Figure 2.10) employs a 2-D lattice composed of entangled physical qubits to encode logical qubit states with each physical qubit within the lattice interacting solely with its nearest neighbors. The surface code's nearest neighbor interaction structure in a two-dimensional plane offers natural and practical realizability compared to non-local codes or codes that require transversal operations. Additionally, its high tolerance to errors [24, 25] makes it an ideal candidate for error correction in quantum computing.

The code distance, denoted as d, is a significant parameter of the surface code. It not only determines the error correction ability by determining how many errors it would take to result in an incorrect logical qubit value but is also closely related to the number of physical qubits needed for encoding a logical qubit.

There are many variations of surface codes, such as the defect-based [17], the twist-based [26], or the patch-based [20] encodings. We focus on the patch-based surface code systems with the simplified rules of tile-based board games introduced in [21].



(a) An example of a distance 3 surface code.



(b) X (above) and Z (below) syndrome extraction circuit.

Figure 2.10: (a)An example of a distance 3 surface code, where the code distance is defined by the minimum weight of the logical operator. The d^2 data qubits, depicted as solid dots, are positioned on the edges, while the $d^2 - 1$ X − and Z −syndrome qubits, represented by open circles, reside on the plaquettes. The red plaquettes correspond to the Z-stabilizer, while the blue ones refer to the X-stabilizer. Each measurement qubit is specifically linked to a stabilizer for its respective measurement process. The logical operators Z_L and X_L are represented as chains (which may take other forms) of operators that are paired across corresponding stabilizers. By applying physical Pauli X(Z) operation on the chain, logical X or logical Z operations are performed. (b)X (above) and Z (below) syndrome extraction circuit. X(Z) physical errors were detected via Z(X) stabilizer measurement.

2.4.2 Tile-based Surface Code

Here, we briefly introduce the simplified rules of tile-based board games defined in [21]. In this framework, we can understand the quantum device responsible for storing the qubits as a board-like structure. The board is partitioned into a number of tiles that can host *patches* representing logical qubits.

Correspondence to surface code lattice surgery

Assuming that we are using the surface code with a code distance d, a tile on the board represents $\sim 2d^2$ physical qubits. Approximately half of these qubits are used for the data state, while the other half are used for error syndrome extraction. The patch's boundaries are represented by solid and dashed edges, indicating smooth and rough boundaries within the surface code patches (see Figure 2.10), respectively.

In this tile-based game, the performance metric is determined by the space-time volume, which corresponds to the board area and the unit of time equivalent to d error-check code cycles. To describe the time required for operations on patches, [21]

introduced a time unit called *Tock*, which corresponds directly to d rounds of code cycles. 0 *Tock* is a special case that does not mean 0 code cycles; rather, it represents operations that have a constant time cost and does not scale with the code distance d. Hence, it should be noted that the constant time cost may be non-zero.

Logical qubit representation

A *patch* is a contiguous area, which can span over multiple tiles, used to represent one or more logical qubits. In this work, we will only use one logical qubit per patch.

An example of two different kinds of one-qubit patches is shown In Figure 2.11. On the boundary of the patch, there can be solid or dashed edges representing the Pauli Z and X operators, respectively. The point at which the solid and dashed edges meet is called a corner or X/Z corner, even if the two edges are on the same line, for historical reasons. We will see their importance when we discuss operations on patches.

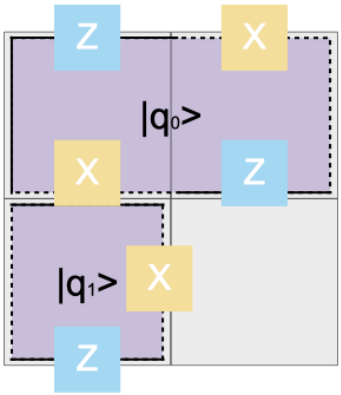


Figure 2.11: Examples of two different kinds of one-qubit patches in a 2×2 grid of tiles.

Patch operations

We will now describe the native operations available in this tile-based game with their corresponding time cost in units of *Tocks*.

Qubit initialization One-qubit patches can be initialized in the $+1$ eigenstates of X_L or Z_L basis ($|+\rangle_L$ or $|0\rangle_L$) in 0 Tocks, where subscript L is used to denote that these are logical Pauli operations and states.

Single-patch measurements One single patch can be measured in either the X or Z basis (see 2.2.4). After the measurement, the patch will be removed from the board, freeing up previously occupied tiles. This operation has a cost of 0 Tocks.

Multi-qubit Pauli product measurement/parity measurement A parity check measurement can also be measured on multiple patches (see Fig. 2.12 for an example). This measurement procedure is performed by first initializing an ancilla patch. The product of the operators on the boundaries of qubit patches can be measured only if they share a border with (are adjacent to) the ancilla patch. For two-qubit cases, we may wish to measure any of the four possible combinations XX, XZ, ZX, or ZZ, but this is only possible if the corresponding X and Z operators border the ancilla patch. After the chosen X or Z boundaries are merged with and then split from the ancilla patch, the ancilla patch is measured. (Combinations with Y are also possible if both X and Z boundaries for a patch border the ancilla.)

The ancilla patch measurement projects the qubits onto the $+1$ or -1 eigenspace of the multi-qubit Pauli product operators and removes the ancilla patch from the board. This operation costs 1 Tock.

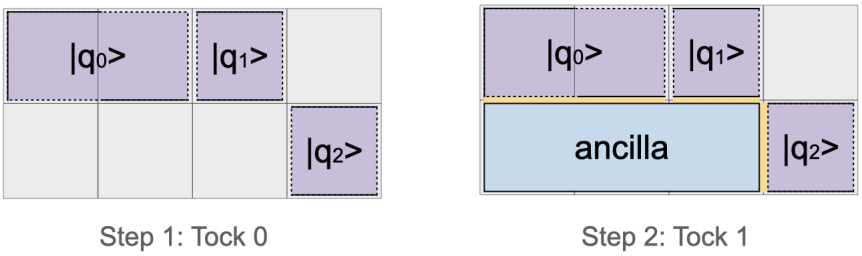


Figure 2.12: A multi-qubit $Y|q_0\rangle \otimes Z|q_1\rangle \otimes X|q_2\rangle$ measurement in one time step.

Patch deformation and rotation A patch offers the flexibility to expand (1 Tock) or shrink (0 Tocks) its tile coverage. Additionally, the X/Z corners can be relocated along the patch boundaries (1 Tock) for patch reformation.

This deformation capability proves valuable for exposing additional or different operators on the patch boundary, facilitating multi-patch measurements.

2.5 Measurement-based Quantum Computing

Measurement-Based Quantum Computation (MBQC), also known as one-way quantum computing, was introduced as an alternative universal computation approach by Raussendorf and Briegel in 2001 [13]. Universal quantum computation is accomplished through MBQC by applying adaptive single-qubit measurements to appro-

priately prepared entangled graph states. One noteworthy example of such a state is the cluster state (see Figure 2.9), thus it is also called *resource state*.

2.5.1 One-bit teleportation

Measurement-based quantum computation relies heavily on the concept of information propagation. A crucial protocol utilized in this approach is known as "quantum teleportation," which was introduced by Zhou, Leung, and Chuang in 2000 [27]. To better understand the principles underlying quantum teleportation, we will look into a simple example involving the teleportation of a one-qubit state (see Figure 2.13). Initially, we have one qubit in the state $|\psi\rangle$ that we aim to transfer to the second qubit.

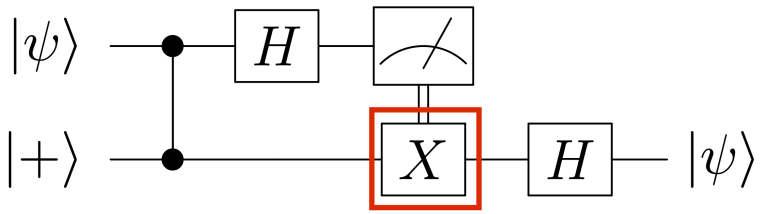


Figure 2.13: Quantum circuit for one-bit teleportation. The use of the CZ gate is for entangling qubits to create a graph state. Measurement on the first qubit has the effect of teleporting the state $|\psi\rangle$ to the next qubit. The using of the X gate and the end perform the necessary Pauli byproduct correction based on the measurement outcome.

2.5.2 Single-qubit gate

Any single-qubit gate can be represented by Euler representation,

$$U(\zeta, \eta, \xi) = He^{-i\frac{\zeta}{2}Z}e^{-i\frac{\eta}{2}X}e^{-i\frac{\xi}{2}Z}, \tag{2.5.1}$$

i.e. a composition of three rotations along two different axes, where X and Z represent rotations around the X and Z axis, respectively.

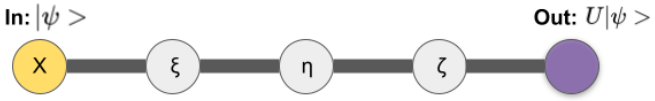


Figure 2.14: Single qubit rotation

In the MBQC model, the implementation of such single-qubit quantum gates can be achieved as follows (see Figure 2.14). Prepare five qubits, with the leftmost one

being the input qubit and the rightmost one as the output qubit. The measurement outcomes for the first four qubits are denoted as s_0, s_1, s_2, and s_3, respectively. The first qubit undergoes an X measurement, and the middle three qubits are sequentially measured adaptively. i.e. depending on the outcome of earlier measurements in the chain. Based on these measurement outcomes, the resulting quantum state is corrected for commutation byproducts.

2.5.3 Two-qubit gate: CNOT

The CNOT operation can be implemented through a 2-dimensional sequence of measurements, as illustrated in Figure 2.15.

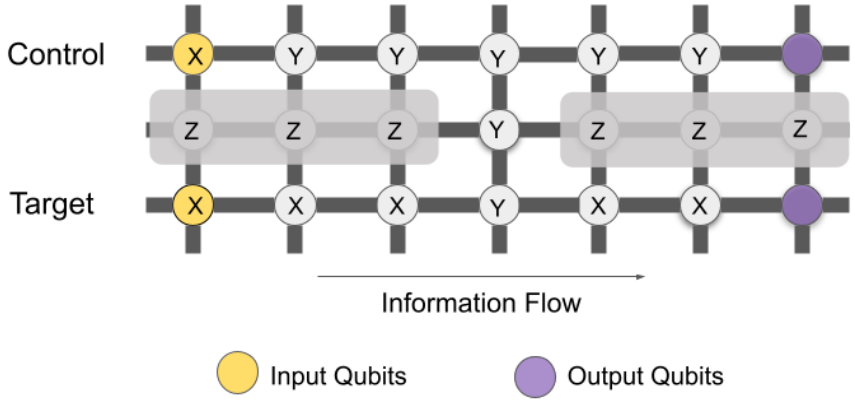


Figure 2.15: Implementation of a CNOT operation in the measurement-based framework. A CNOT gate can be performed between an input qubit and an output qubit using a 15-qubit cluster state. Simultaneous Pauli measurements are applied to each qubit based on the indicated pattern in the cluster state. Before the CNOT sequence, Z-basis measurements are conducted, depicted as gray blocks, to remove unwanted qubits and disconnect them from the graph.

2.5.4 Universal Computing by MBQC

We can convert a quantum circuit into the MBQC model with a 2D lattice cluster state of sufficient size, as MBQC is universal. The first step of the operations is to etch the circuit topology out of the lattice cluster state using Pauli Z measurements. These measurements disconnect the measured node from the cluster, as shown in Figure 2.15. If starting with an algorithmic-specific graph state, refer to [10].

After establishing the topology to support computation, the remaining qubits are measured in an adaptive manner, following the information flow. It is worth noting

that implementing the gates in the same order as the circuit description is unnecessary in the MBQC model. Non-adaptive measurements (Clifford operations) can be interchanged or performed simultaneously on the previously prepared graph state, as they are independent. Next, single-qubit measurements are carried out according to a specific measurement scheme. The measurement angles can be adjusted based on the outcomes of previous measurements. These results are then used to determine the measurement basis for subsequent qubits in subsequent rounds, continuing until the computation is completed. This approach allows us to avoid the need to prepare all of the entanglement at once and instead dynamically consume the cluster state as we generate it in correspondence with the measurement rounds. To perform multi-qubit operations, entanglement between multiple rows must be maintained.

Once all qubits have been measured, the last step is to perform Pauli corrections on the final state to correct the byproduct.

Chapter 3

Problem Definition

3.1 Problem Definition

This research focuses on efficiently generating a fault-tolerant arbitrary graph state using the rules outlined in [21]. The primary objective is to minimize the space-time volume cost associated with this generation process.

The significance of generating an arbitrary graph state lies in its wide range of applications. For example, a recent study [10] introduced an innovative approach for the measurement-based quantum computing model, utilizing an algorithm-specific graph state which is a local Clifford equivalent of the measurement-based quantum computing system after applying all the Clifford gates in the corresponding quantum circuit, as a resource state instead of the conventional lattice cluster state.

By preparing the algorithm-specific graph state, computations can be performed through non-Pauli measurements on the qubits, eliminating the need for applying Clifford gates. Additionally, this approach offers advantages in logical-level compilation for fault-tolerant error-correction codes. It eliminates the requirement of complex state evolution in error-corrected qubits, as the focus shifts solely to generating the graph state using logical qubit measurements.

3.2 Evaluation Metrics

3.2.1 Space-time Overhead

Within the framework described in [21], a space-time resource model is utilized to evaluate performance. The primary trade-off observed is between the number of logical qubits (measured in *tiles*), representing the board area, and the time unit related to code cycles spent (referred to as *Tocks*). The space-time overhead of a protocol utilizing a layout with s *tiles* for t *Tocks* can be expressed as

$$\text{overhead}_L = s \times t. \tag{3.2.1}$$

When considering a more fundamental level, specifically the level of physical qubits, the total cost is

$$\text{overhead}_P = s \times t \times d^3 \tag{3.2.2}$$

Here, d denotes the number of code cycles, where each tile (i.e., logical qubit) contains d^2 physical data qubits, and one Tock corresponds to d code cycles.

Similar evaluations involving resource overhead related to time and space have been applied in various quantum computing studies [28, 29, 30, 31]. Therefore, this work also adopts the same overhead evaluation metric, with the primary goal of minimizing the space-time volume cost of graph state generation.

3.2.2 Data-to-Ancilla Ratio

The analysis of the layout design can be conducted by calculating the ratio between the number of logical qubits utilized for representing vertices in the graph state and the number of logical qubits employed as ancilla. This ratio, denoted as *ratio*, can be expressed as:

$$\text{ratio} = \frac{\text{vertices\_tiles}}{\text{vertices\_tiles} + \text{ancilla\_tiles}} \tag{3.2.3}$$

Here, vertices_tiles represents the number of tiles employed to represent vertices in the graph state, while ancilla_tiles corresponds to the number of tiles utilized as ancilla.

This ratio provides insight into the "compactness" of the layout design, revealing how efficiently the resources are utilized in the representation of the graph state. Note that the implementation of magic state distillation is not considered in our cost calculation for the time being.

3.3 Related Work

3.3.1 Fault-tolerant Schemes for MBQC

We briefly summarize the relevant work from various stages of fault-tolerant quantum computing based on measurement-based quantum computing (see Figure 1.1). The first stage, which converts the quantum circuit as input into a circuit/algorithm-specific graph state, as mentioned in Section 1, has been implemented by Jabalizer. The focus of our work is the second stage, to compile the graph state into a set of operations that can be executed on the quantum error correction. Raussendorf et al [32, 33] suggested that measurement-based quantum computing can be achieved by creating topologically entangled 3D cluster states, and MBQC can be fault-tolerantly performed on the 3D clusters via sequential single-qubit measurements. Although

it only requires local and next-neighbor interaction of qubits, the 3D structure still imposes some specific hardware requirements; while Newman, de Castro, and Brown [34] also proposed an approach using degree-3 connectivity.

A widely utilized technique for generating graph states, which are suitable for comparison with the approach presented in this thesis, involves utilizing the CZ gate to generate the graph state. Table 5.1 presents a comparison of the time cost required for specific graph types.

In the third stage, one potential approach that can be considered is the optimization of magic state distillation [31]. In the fourth stage, it is possible to consider different quantum device architectures, as in [35]. Alternatively, if we take into account the limitations of the devices at an earlier stage, we can consider the dynamic version mentioned in Section 1. The dynamic version allows for the use of smaller devices and more flexible implementations. By considering these factors from the outset, we can design fault-tolerant quantum computing approaches that are tailored to the specific constraints and capabilities of the devices being used.

Chapter 4

Substrate Scheduler: Protocol Design

The Substrate Scheduler, a compiler module specifically designed for fault-tolerant graph state compilation, has initially been proposed in [1]. Building upon the initial proposal, this thesis aims to provide additional insight and enhanced functionalities.

4.1 The Framework of Substrate Scheduler

Fig. 4.1 summarizes the key components of the Substrate Scheduler. By utilizing the adjacency matrix as input, the Substrate Scheduler is theoretically able to process any graph state.

4.2 Design of the Layout

4.2.1 One Bus Design

We propose a layout design where the ancilla patch consists of a row (one bus) of logical qubits, with the same length as the patches representing vertices.

One-bus layout design using 2-tile one-qubit patches Given the space-time trade-offs, one primitive design is proposed using the 2-tile one-qubit patches in [21] (refer to Section 2.4) as logical qubits to represent the vertices in the graph state. This design aligns the ancilla bus with both the X and Z boundaries of the logical qubits. Consequently, stabilizer generators can be measured using the ancilla patch without the need for expensive patch rotations, as introduced in Section 2.4.

The structure utilized is a block of 2 rows by $2n$ columns, representing n logical qubits and the corresponding ancilla patch, as illustrated in Figure 4.4. The spatial

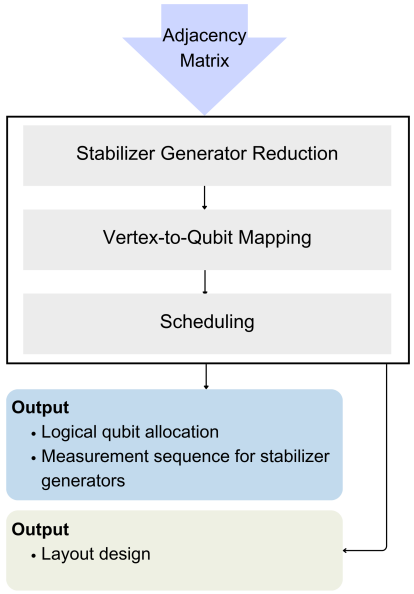


Figure 4.1: Workflow of the Substrate Scheduler.

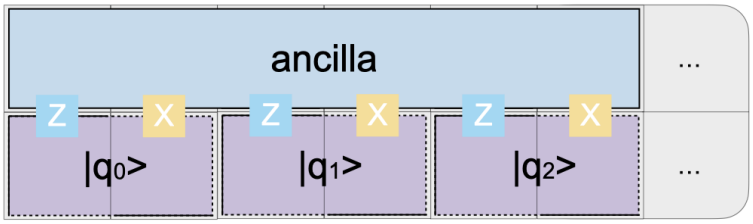


Figure 4.2: The one-bus layout design using 2-tile one-qubit patches.

cost of this layout design is $4n$ tiles, where n denotes the number of vertices in the graph corresponding to $\sim 8d^2$ physical qubits (refer to Section 2.4). The data-to-ancilla ratio in this design is therefore 0.5 (see Section 3.1). For the time being, our cost calculation does not consider the implementation of magic state distillation.

As mentioned in Section 2.3, generating a graph state with n vertices requires the measurement of n stabilizer generators, and the time required to perform a multi-qubit Pauli product measurement/parity measurement is 1 Tock. Consequently, in the worst-case scenario (without optimization), the space-time overhead of constructing a graph state using this layout design amounts is $4n^2$. It should be noted that this spatial cost is not optimal when employing pre-mapping stabilizer generator reduction and allowing the use of 1-tile one-qubit patches, which will be discussed later in Section 4.3.

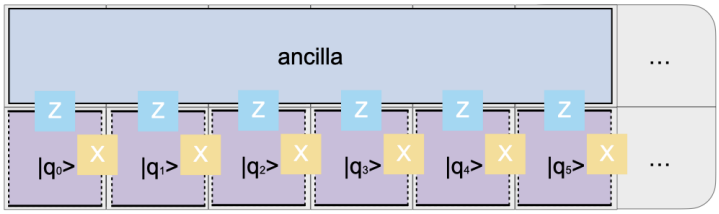


Figure 4.3: The one-bus layout design using 1-tile one-qubit patches.

One-bus layout design using 1-tile one-qubit patches When utilizing 1-tile one-qubit patches in this design, the data-to-ancilla ratio remains at 0.5, and the spatial cost instead is $2n$ tiles. Additionally, considering the qubit reformation process described in Section 2.4, which takes 1 Tock, each stabilizer measurement in the worst-case scenario will require 2 Tocks: one for qubit reformation and one for multi-qubit Pauli product measurement/parity measurement. Consequently, the space-time overhead of constructing a graph state using this layout design is also $4n^2$. This is equal to using the 2-tile one-qubit patches. However, taking into account the optimizations offered by pre-mapping reduction for 2-tile one-qubit patches (refer to Figure 4.9), the space-time overhead is hignger using 1-tile one-qubit patches. Therefore, in this study, we primarily adopted the layout design of 2-tile one-qubit patches.

4.2.2 Two Bus Design

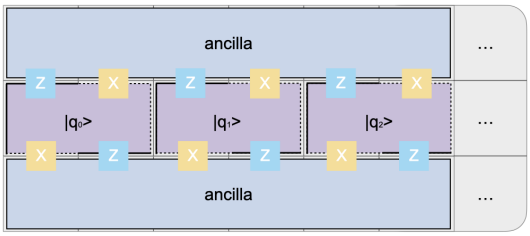


Figure 4.4: The two-bus layout design using 1-tile one-qubit patches.

Due to the possibility of parallel measurements of stabilizer generators (as discussed in Section 4.3), the two-bus layout design has an advantage over the one-bus layout design. When employing the one-bus ancilla layout, it is crucial to ensure that no two measurements occur simultaneously on a single logical qubit, and also

to prevent any overlap among the stabilizer generators being measured concurrently. This requirement is reflected in the need to prevent any overlap between the leftmost and rightmost positions of any ancilla used by stabilizer generators measured simultaneously. However, for the two-bus design, this constraint is no longer a strict requirement.

The space-time overhead of constructing a graph state using this layout design, in the worst-case scenario without any optimization, amounts to $6n^2$. Furthermore, the data-to-ancilla ratio in this design is $\frac{1}{3}$.

At the current stage, the Substrate Scheduler only handles one-bus layout designs.

4.2.3 Other Layout Designs

Other layout designs can be considered from two perspectives: enhancing the possibility of parallel measurements of stabilizer generators and accommodating hardware requirements. Both one-bus and two-bus designs involve a considerable number of tiles, where each tile represents approximately $\sim 2d^2$ physical qubits. It is also worth noting that the strip-like device structure required by these designs poses challenges in terms of current hardware capabilities. Achieving such specific configurations with existing hardware remains difficult.

4.3 Three-Phase Process of Optimization

We split the time step reduction process into 3 phases, namely: stabilizer generator reduction, vertex-to-qubit mapping, and scheduling of the stabilizer generator measurements. The first step in the process is to reduce the number of stabilizer generators that need to be actively measured. This is achieved by appropriately initializing the qubits so that they are already stabilized by a subset of the stabilizer generators. Subsequently, the vertex-to-qubit mapping is determined using a proposed heuristic algorithm that demonstrates effectiveness for certain types of graphs. In the final step, the Substrate Scheduler optimizes the schedule for stabilizer generators to minimize time costs by concurrently measuring as many stabilizer generators as possible.

This workflow produces an optimized schedule of operations and logical qubit allocation, which, in combination with the layout design, is then passed to the final compilation phase. Following this, the target quantum device generates the fault-tolerant quantum graph state.

Despite the natural sequence of processes in the Substrate Scheduler, as depicted in Figure 4.1, which follows reduction, mapping, and then scheduling, this thesis has chosen an alternative order for explanation, namely reduction -> scheduling -> mapping, as it offers greater clarity and ease of understanding.

4.3.1 Stabilizer Generator Reduction[1]

As mentioned in Section 2.3, an n-qubit graph state $|G\rangle$ is uniquely identified as the simultaneous $+1$ eigenstate of all n stabilizer operators, i.e. we need to perform n stabilizer generator measurements to generate the graph state.

From the rule set in [21], we know that the time cost to initialize logical qubits to $|+\rangle$ or $|0\rangle$ are both 0 time steps. This allows us to initialize some qubits in such a way that they are already stabilized by a subset of the stabilizer generators. For example, the three-vertex line graph G is stabilized by the following generators,

$$\begin{aligned} g_0 &= X \otimes Z \otimes I, \\ g_1 &= Z \otimes X \otimes Z, \\ g_2 &= I \otimes Z \otimes X. \end{aligned}$$

Initializing q_0 in $|+\rangle$ and q_1 in $|0\rangle$ prepares a state that is automatically stabilized by g_0 (here we assume that the mapping of the vertex-to-qubit is also sequential from left to right). We can go further and initialize qubit q_2 in the state $|+\rangle$, which will ensure that the three-qubit state is a simultaneous $+1$ eigenstate of both g_0 and g_2,

$$g_0 \,|+0+\rangle = |+0+\rangle = g_2 \,|+0+\rangle \,. \tag{4.3.1}$$

This results in the reduction of total stabilizer generator measurements that we need to perform via multi-Pauli product measurement. In our example, the only stabilizer generator that remains to be measured in order to prepare the three-qubit graph state is g_1.

For a simple argument, to stabilize g_a, we must initialize qubit a in the state $|+\rangle$, and all of its neighboring qubits in $|0\rangle$. This initialization strategy prohibits us from simultaneously stabilizing any of the neighboring qubits in $\mathrm{ngbr}(a)$. However, we can stabilize qubits inside $\mathrm{ngbr}(\mathrm{ngbr}(a))$ that are not adjacent to a concurrently with g_a. As a result, the optimal reduction problem can be reduced to the maximum independent set problem. Knowing the maximum independent set $\alpha(G)$ identifies the qubits which must be initialized in $|+\rangle$, while all the remaining qubits are initialized in $|0\rangle$. This means that the number of stabilizer generators that need to be measured decreases to $|V| - |\alpha(G)|$. In practice, we compute a maximal independent set to achieve a reasonable time complexity.

4.3.2 Scheduling of the Stabilizer Generator Measurements

Given that each vertex in the graph is assigned to a specific logical qubit as depicted in Fig. 4.8, along with a fixed stabilizer generator reduction, we can decide the measurement sequence for the stabilizer generators by exploiting their commutativity.

[1]This section, along with Section 4.3.2 is based on my first-authored paper, titled "A Substrate Scheduler for Compiling Arbitrary Fault-tolerant Graph States" (2023). See [1]

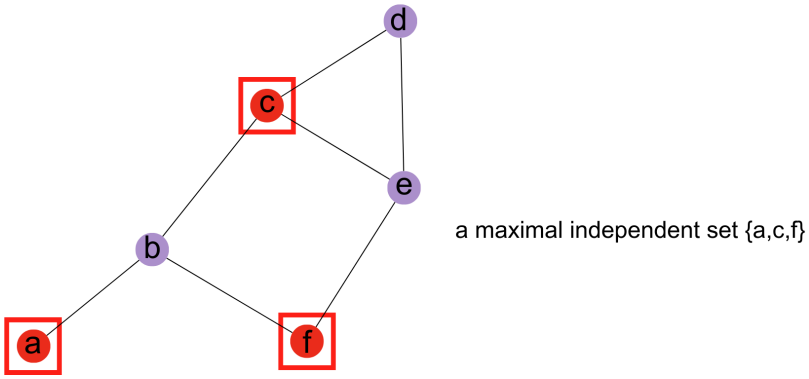


Figure 4.5: An example of a maximal independent set. By initializing the logical qubits corresponding to the vertice a in $|+\rangle$, the requirement to measure the stabilizer generator g_a associated with this vertex can be omitted.

In order to measure a stabilizer generator g_a, the ancilla is required to cover patches that represent vertices a and ngbr(a), without the need to cover the remaining qubits. Thus, we can define an ancilla block for measuring the stabilizer generator g_i by a pair of two numbers (L_i, R_i) with $L_i, R_i \in [1, 2n]$ where L_i and R_i denotes the leftmost and the rightmost qubits that the ancilla needs to cover, respectively.

Consequently, in a one-bus design, simultaneous measurement of stabilizer generators that share an overlapping ancilla region from the leftmost to the rightmost mapped vertices is unfeasible (refer to Figure 4.6 for an illustrative example).

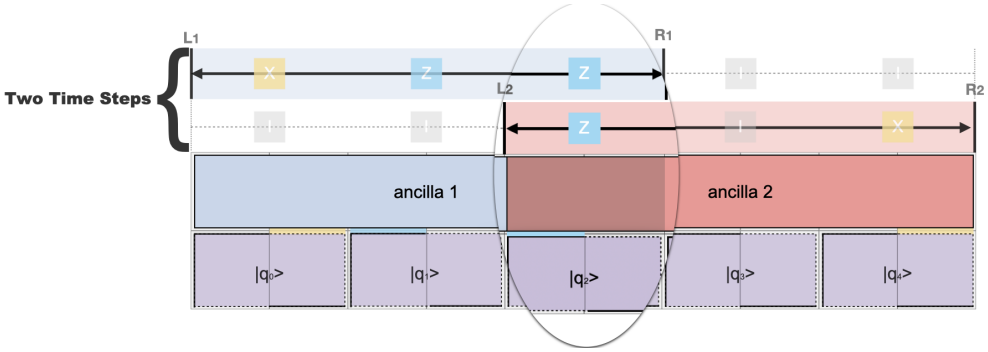


Figure 4.6: An example of the overlapping of two stabilizer generators.

The challenge of maximizing the simultaneous measurement of stabilizer generators can be reframed as minimizing the overall height of the stacked ancilla blocks. Below is the step-by-step explanation:

1. Sort the list of ancillas, $[(L_i, R_i)]$, in ascending order of R_i. If two pairs have the same R_i value, sort them in non-decreasing order of L_i as well.

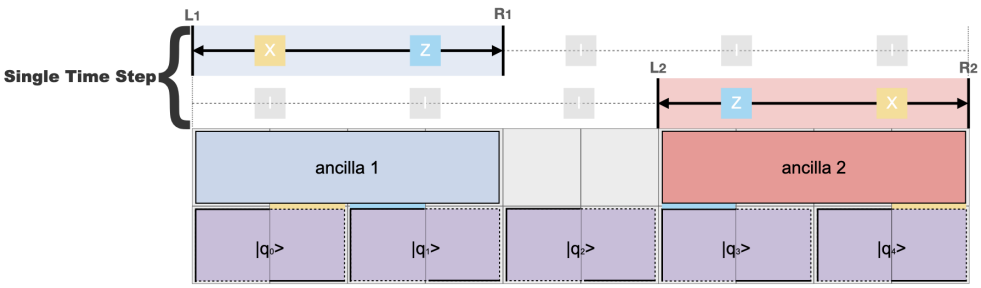


Figure 4.7: Measuring two stabilizers in one time step.

2. For each time step, initialize an empty set to store the generators that can be measured concurrently $Set\_t = \{\}$.

3. Traverse the sorted list from the beginning. For each ancilla represented by the pair (L_i, R_i), check if $R_j < L_i$ for the most recently added pair (L_j, R_j) in the set Set_t. If this condition holds, remove the pair (L_i, R_i) from the unmeasured list and add it to Set_t. Proceed with traversing the list until reaching the end.

4. Repeat steps 2 and 3 until the list is empty. At each iteration, the set Set_tobtained corresponds to the stabilizer measurements that can be performed simultaneously in each time step.

The scheduling algorithm provides an optimal solution and has an average time complexity of $O(n \log n)$ and a space complexity of $O(n)$.

4.3.3 Vertex-to-Qubit Mapping

Qubit mapping (see Figure 4.8), which involves assigning vertices to logical qubits, plays a crucial role in determining the number of stabilizers that can be measured simultaneously. The objective is to optimize the vertex-to-patch assignment to maximize parallelism during stabilizer scheduling, ultimately minimizing the resulting time step after scheduling.

It is important to note that the mapping for the one-bus ancilla layout design differs from other layout designs. In the case of the one-bus ancilla layout, not only do no two measurements occurring simultaneously on a single logical qubit crucial, but it is also important to prevent any overlap among the stabilizer generators being measured concurrently. This is reflected in the requirement that there should be no overlap between the leftmost and rightmost positions of any ancilla used by stabilizer generators measured simultaneously.

This additional constraint simplifies the task of finding an optimal bound for potential solutions in the one-bus ancilla mapping problem, thereby facilitating the identification of the most suitable mapping.

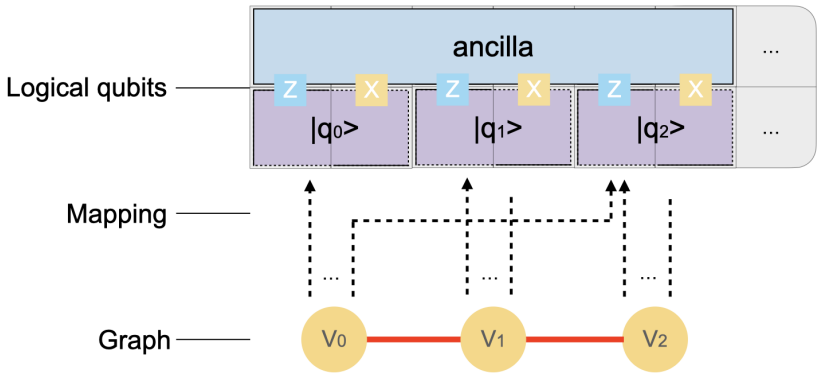


Figure 4.8: A schematic diagram of the mapping process for one-bus ancilla layout design. The dashed lines represent the mapping process, where vertices are assigned to potential logical qubits.

MinCut mapper

As discussed earlier, our goal in qubit mapping is to minimize the overlap of stabilizer generators (also ancillas) to maximize the potential for simultaneous multi-Pauli measurements. To achieve this, we can intuitively position the dense components (i.e. subgraphs that are highly connected) of the graph G close to each other, minimizing the distance between the start and end positions (L, R) of the stabilizer generators.

To accomplish this, we propose an iterative mapping method that involves finding the minimum cut of the graph (refer to Appendix A for details). In graph theory, a minimum cut refers to partitioning the vertices of graph G into two sets, minimizing the number of edges that cross the partition. In the case of the MinCut mapper, we repeat the process of cutting the graph until we obtain a subgraph with two or fewer vertices. We then map these vertices in the subgraph to adjacent logical qubits at the end of the row and remove the subgraph. Then, we continue processing the first subgraph until no subgraphs with more than two vertices remain.

To solve the minimum cut problem, a randomized algorithm known as Karger's algorithm [36] can be employed. Karger's algorithm exhibits a time complexity of $O(n^2)$ for a single run. To obtain the optimal solution, we can run the algorithm $n^2 \log(n)$ times, resulting in an overall time complexity of $O(n^4 \log(n))$.

Further Layout Optimization After the initial stabilizer generator reduction and vertex-to-qubit mapping, further optimization can be achieved in the layout design

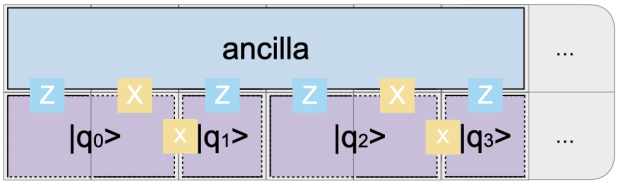


Figure 4.9: The one-bus layout design using 2-tile one-qubit patches after the pre-mapping stabilizer generator reduction.

by utilizing both two-tile one-qubit patches and one-tile one-qubit patches for logical qubits.

In particular, any qubit $a \in \alpha(G)$ is initialized in the $|+\rangle$ state, indicating that it will not participate in stabilizer measurements requiring measurement in the X basis. As a result, these qubits can be represented by patches that occupy only a single tile, as depicted in Fig. 4.9.

This optimization results in a reduction in the space complexity of the 2 two-row layout design, ultimately reaching a value of $2 \times 2n - |\alpha(G)|$.

Two-sided crossing

In this work, we explore methods to optimize the mapping, as it has a vital impact on scheduling.

We propose a new approach by dividing the stabilizer generator and vertices into a bipartite graph with two layers: L_g and L_v (as shown in Figure 4.10). The stabilizer generator is connected to its corresponding vertices, allowing only between-layer connections and no connections within a single layer. The number of crossings in this bipartite graph depends on how the vertices are arranged in both layers. In the one-sided crossing minimization problem, we keep the coordinates of one layer fixed while permutating another layer, while in the two-sided crossing minimization allowing for permutations in both layers.

The idea is that reducing the crossing of edges decreases the chance of overlap among ancillas, which can be represented by their positions $[L_i, R_i]$. As a result, more stabilizer generators can be simultaneously measured, improving the overall mapping process. In this minimization process, the crossings on the nodes (multi nodes in L_g connected to one node in L_v) are not taken into account, which means that the simultaneous use of the same logical qubit cannot be considered in the optimization process. This introduces a certain limitation to the optimization. For example, if we have a pair of ancilla with positions represented as $[L_1, R_1]$ and $[L_2, R_2]$, where R_1 is less than or equal toL_2. If there is an overlap at the endpoints, i.e., R_1 and L_2, this situation cannot be considered in the optimization process because they do not have crossings between layers. However, we propose an assumption that there exists an

optimal permutation π_o for layer L_v that allows us to perform the maximum number (greater than 0) of simultaneous measurements. In this case, the number of crossings between layers must be minimized, meaning that minimizing the number of crossings is both a necessary but not sufficient condition for achieving the optimal mapping.

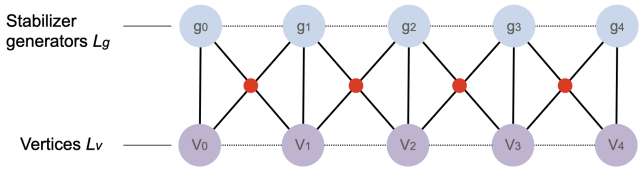


Figure 4.10: An example of the two layers representation of a 5-vertex path graph. The red points show the between-layer crossings.

This problem can be seen as a two-layer crossing minimization problem, which has been proven to be NP-complete in both the one-sided and two-sided scenarios by Garey and Johnson [37], and Eades and Wormald[38], respectively. However, we can use heuristic algorithms to approximate the optimal solution. In our study, we keep L_g a fixed permutation π_0 while allowing flexibility in rearranging L_v and trying to find a permutation π_1 of L_v such that the inter-layer crossing number $cr(G, \pi_0, \pi_1)$ is minimized, similar to a labeling process. We employ the Barycenter heuristic [39], which is considered one of the best algorithms for obtaining an approximate optimal solution[40] with an approximation ratio of $d-1$ [41], where d represents the maximum degree of a vertex in L_v. By applying the Barycenter heuristic, we simply compute the barycenters of vertices in the free layer L_v, which is the average of its neighbors' position in the fixed opposing layer L_g. We then determine the order of L_v, denoted as π_1, by sorting based on increasing barycenters. This algorithm allows us to efficiently find a solution within a short time complexity of $\theta(|L_g| + |L_v|log(|L_v|))$ [42].

Chapter 5

Evaluation

In this chapter, the functionality and performance of the proposed tool is evaluated. The evaluation metrics used in this section were introduced in Section 3.1.

As mentioned in the Chapter 1, one significant application of graph states is as resource states in the MBQC model. Describing algorithm-specific graph states corresponding to different quantum circuits or algorithms can be challenging due to their distinct characteristics, such as size and density, especially after undergoing optimizations like local complementation [11, 43]. It is not yet clear what structural properties we should expect from a typical instance of an algorithm-specific graph state.

Thus, in order to gain insight into the performance of the Substrate Scheduler in its application, we chose the following testing strategy. First, we verify that the Substrate Scheduler produces the correct and expected output by testing specific regular types of graphs in Section 5.1. Then, we explore how the sparsity and size of the graph affect the space-time overhead required to prepare the target graph state in Section 5.2 and Section 5.3, respectively.

5.1 Evaluation of Specific Types of Graphs

We start by assessing the Substrate Scheduler's performance on various types of graphs, such as line graphs, star graphs, tree graphs, and complete graphs. These specific graph classes were selected to facilitate an analysis of the optimization process for graph state preparation, enabling us to verify the correctness of the Substrate Scheduler's output. By studying how our approach minimizes the overhead associated with graph state preparation in these basic examples, we gain valuable insights into what we can anticipate when testing the Substrate Scheduler on random graphs.

To begin with, we summarize the analysis of the optimization process in Table 5.1.

To assess the performance of our method, we compare the time cost of creating the graph state after applying the Substrate Scheduler with the time cost of generating the

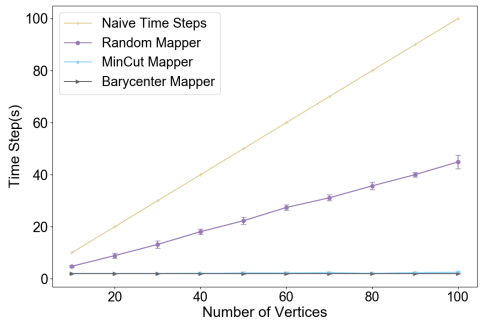
Graph Type	CZ Preparation Depth	Maximum Stabilizer Reduction	Parity Check Preparation Depth
Line Graph	2	$\|V\| \to \|V\|/2$	2
Star Graph	$\|V\| - 1$	$\|V\| \to 1$	1
Random Tree	$\Delta(G)$	$\|V\| \to \|V\|/2$	$\Delta(G)$
Complete Graph	$\|V\|$ or $\|V\| - 1$	$\|V\| \to \|V\| - 1$	$\|V\| - 1$

Table 5.1: Preparation depth associated with generating different types of graph states, where $\Delta(G)$ is the highest degree of any vertex in G.

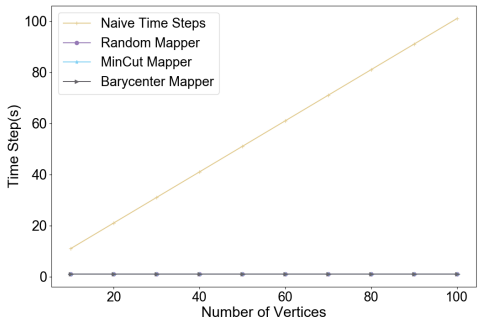
graph state using CZ gates. The second column in Table 5.1represents the optimal preparation depth required for constructing the graph state using CZ gates. This depth corresponds to the minimum number of time steps necessary to prepare the graph state when ancilla buses are not restricted by spatial layout. It is proportional to the chromatic index of the graph.

The third column denotes the number of parity checks that need to be performed before and after initializing the maximum independent set of the given graph class in the $|+\rangle$ state for stabilizer generator reduction.

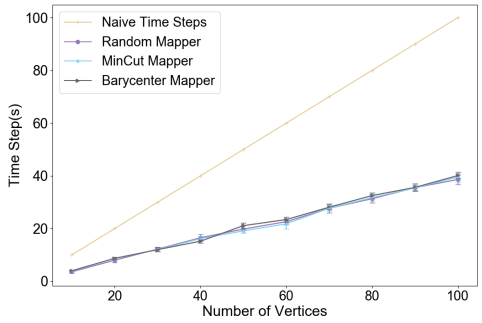
The fourth column is the minimum number of time steps required for performing the stabilizer parity checks when allowing non-overlapping parity checks to be applied in parallel. It is worth noting that the naive time step cost for generating graph states using the stabilizer formalism is determined by the number of vertices in the graph, denoted as $|V|$. The preparation depth for a complete graph is $|V|$ when the number of vertices is odd and $|V|-1$ when it is even. In the case of random trees, the maximum stabilizer reduction can reach the lower bound since trees are bipartite, and selecting the maximal independent set as the larger of the two bipartitions results in optimal reduction.



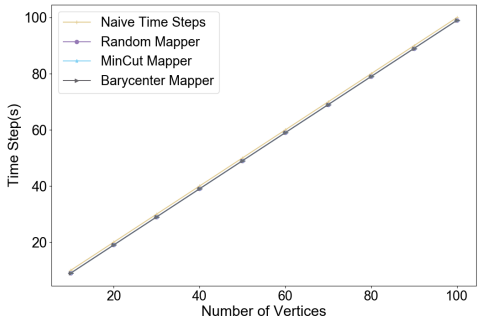
(a) Line Graph



(b) Star Graph

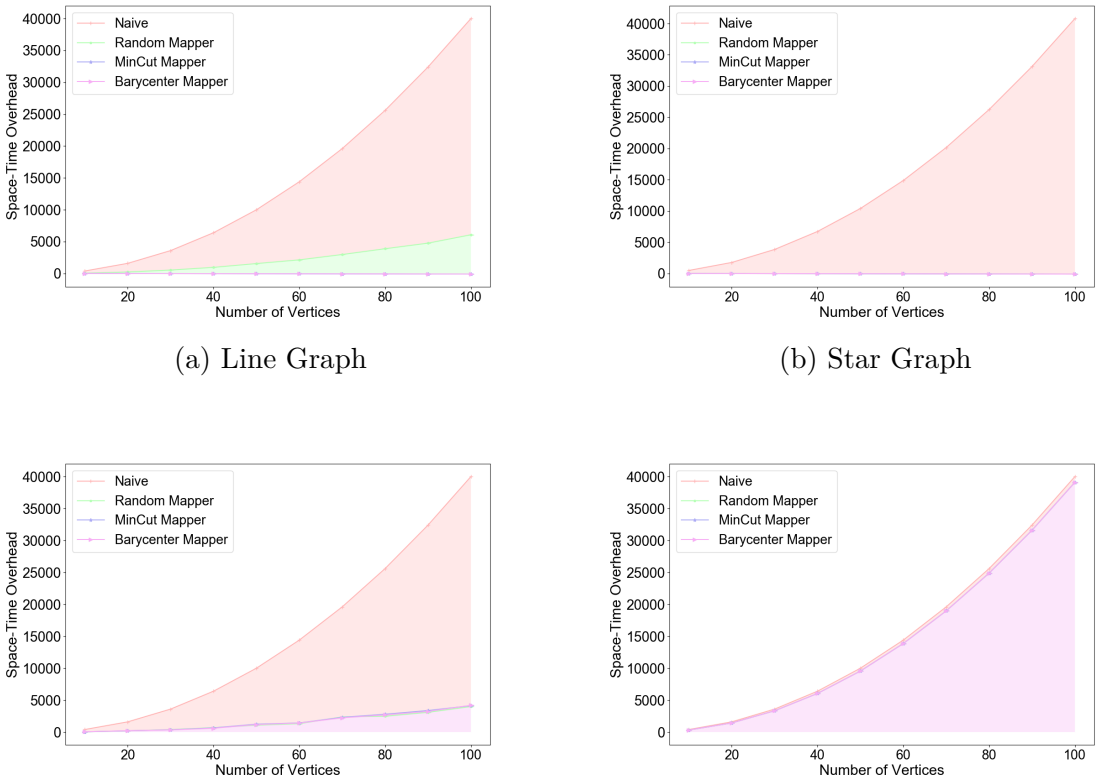


(c) Random Tree Graph



(d) Complete Graph

Figure 5.1: The performance of the Substrate Scheduler is evaluated based on the time cost across four distinct types of graphs. The horizontal axis represents the graph size (number of vertices), while the vertical axis displays the number of time steps needed.



(a) Line Graph

(b) Star Graph

(c) Random Tree Graph

(d) Complete Graph

Figure 5.2: The performance of the Substrate Scheduler is evaluated based on the space-time overhead (refer to Section 3.1) across four distinct types of graphs. The horizontal axis represents the graph size (number of vertices), while the vertical axis displays the space-time overhead.

We evaluated the performance of the Substrate Scheduler, under the one-bus ancilla layout design, on two aspects: 1. time steps: the time steps required to generate the graph state were measured and depicted in Figure 5.1; 2. space-time overhead (see Section 3.1): we also assessed the space-time overhead of the generation process using the 2-tiles one-bus layout design, as shown in Figure 5.2. To compare different mapping methods, we examined the results of the Barycenter heuristic proposed in this thesis, the MinCut mapping, and a random qubit assignment.

In Figure 5.1, grey lines represent experiments using the Barycenter Mapper, which aims to minimize one-sided crossing. Light blue lines indicate experiments with the MinCut mapping method. Purple lines depict experiments conducted with a random mapping method without any mapping optimization. The experiment was conducted 10 times on the same examples, with each point on the plot representing the average value obtained from these 10 experiments. The error bars indicate the standard deviation.

Overall, the Substrate Scheduler achieves a significant speedup in terms of time steps across three out of four graph types, with line graphs and star graphs experiencing a reduction in time steps from linear growth to constant levels. The obtained results align with the expected outcomes and can be compared to the theoretical minimum (refer to Table 5.1) when using the Barycenter and MinCut mapper. The time step growth of the random tree depends on the maximum dimension of the graph, as previously discussed. In the complete graph, there is no discernible improvement.

Results in Figure 5.2 show that the time cost reduction contributes to a reduction in the space-time overhead required for generating fault-tolerant graph states, as the layout design is fixed. For the tree graph, the space-time overhead has been reduced to a level below the exponential growth rate observed in unoptimized methods. Additionally, the utilization of Barycenter Mapper and MinCut Mapper ensures that line graphs and star graphs can maintain a constant-level space-time overhead, further enhancing efficiency.

5.2 Evaluation of Random Graphs with Different Density

The overall overhead of generating graph states is influenced by several significant factors, including the type of graph, its size, and its density. The density of graphs is represented by the ratio between the number of edges in a graph $|E|$ and the maximum number of edges that the graph can contain. Thus when the number of vertices is fixed, the number of edges has determined the density of a graph. For undirected simple graphs, we define the graph density as:

$$\textbf{Density} = \frac{2|E|}{|V| \times (|V| - 1)} \tag{5.2.1}$$

Figure 5.3 and 5.4 presents experimental results illustrating the impact of various graph densities on the time cost and space-time overhead. To generate the graphs for the experiments, the NetworkX Python package [44] is utilized. The graphs were created by uniformly selecting from the collection of all graphs that consist of 100 vertices and $|E|$ edges.

In Figure 5.3 and 5.4, each point represents an average result of 100 distinct randomly connected graphs consisting of 100 vertices. The error bars indicate the standard deviation. The horizontal axis denotes the graph density, where the region to the left of the gray dashed line represents sparse graphs, the region between the gray and red lines represents intermediate graphs, and the region to the right of the red dashed line represents dense graphs.

The results demonstrate the Substrate Scheduler's ability to reduce the space-time overhead, which becomes less significant as the graph density increases. Nevertheless, as the graph density increases, the overhead after optimization demonstrates a pattern

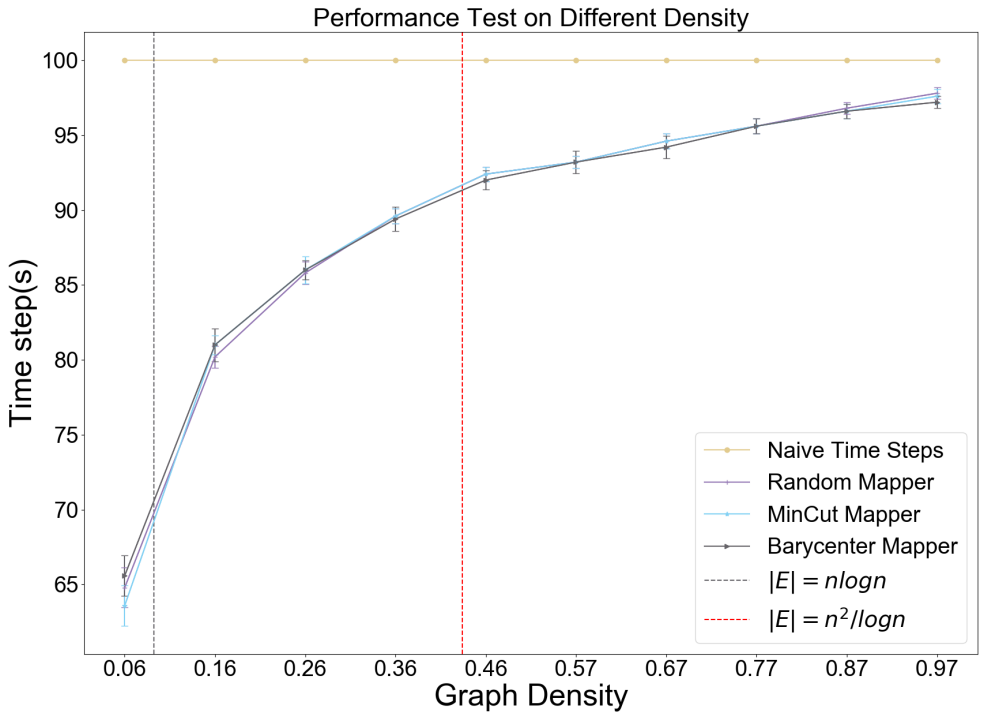


Figure 5.3: The performance of the Substrate Scheduler is analyzed across a range of densities. The horizontal axis denotes the graph density, while the vertical axis represents the corresponding number of time steps.

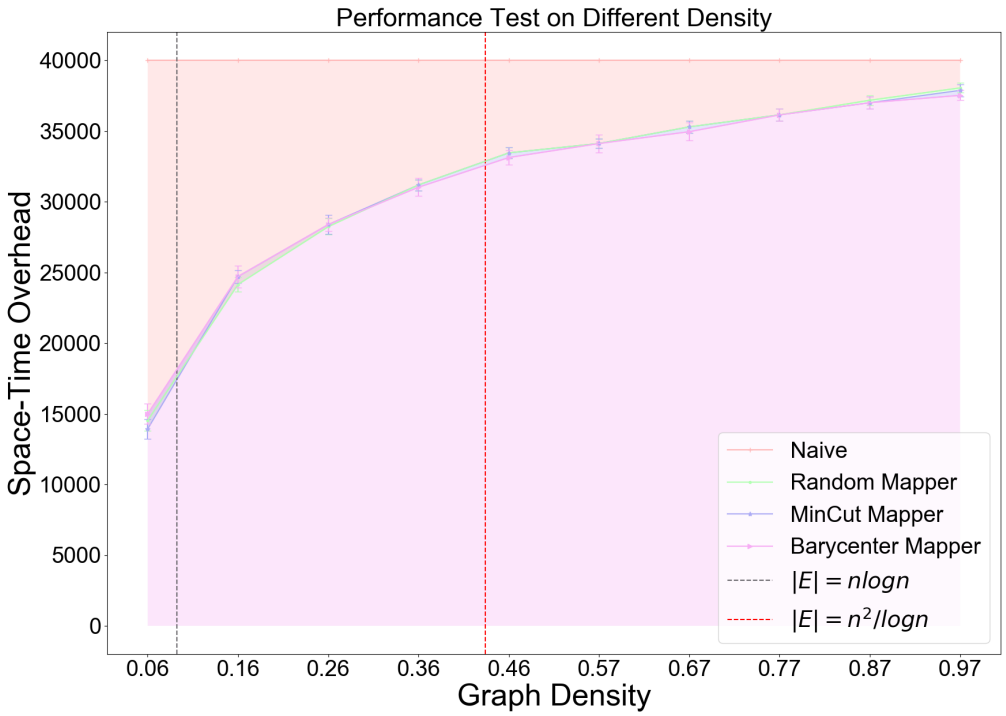


Figure 5.4: The space-time overhead under different graph densities. The horizontal axis denotes the graph density, while the vertical axis displays the space-time overhead.

of growth that closely resembles a logarithmic curve. Initially, the increase is rapid, but it gradually slows down as the graph transitions from being sparse to dense. This observation implies that the Substrate Scheduler's performance is less significantly affected as the graph density surpasses a certain threshold.

Moreover, it is important to mention that the MinCut mapping method demonstrated similar performance to the random mapping method, both generally outperforming the Barycenter Mapper when dealing with less dense graphs. However, as the graph density surpasses a certain threshold, the Barycenter Mapper proposed in this thesis exhibits superior optimization outcomes.

5.3 Scalability Testing

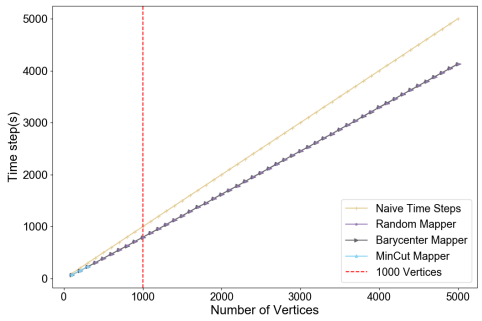
To assess the Substrate Scheduler's suitability for promising near-term applications, experiments to evaluate its performance across graphs of varying sizes are conducted, as illustrated in Figure Fig. 5.5 and Fig. 5.6.

In order to obtain representative results, we selected two distinct types of graphs based on their densities, as defined in this work: sparse graphs (with $O(n \log n)$ edges) and dense graphs (with $O(n^2/\log n)$ edges).

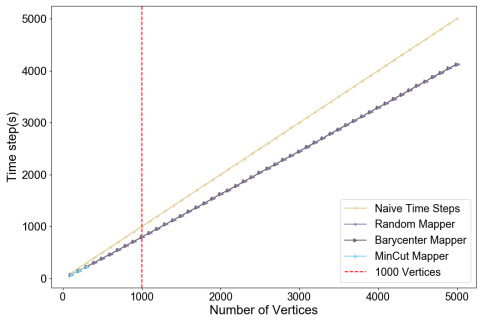
In Figure 5.5 and 5.6, each point on the plot represents the average performance over 2 randomly generated instances, the error bars are too small to be seen. To maintain practicality and avoid excessive compilation times in real-world scenarios, we limited our testing with the MinCut Mapper to graphs containing a maximum of 300 vertices. The red vertical dashed line indicates the specific target for this study of 1000 vertices, which enabled a reasonable evaluation of the Substrate Scheduler's overall behavior and compatibility with other compiler tools.

The experimental results demonstrated the Substrate Scheduler's effectiveness in handling sparse graphs, suggesting its potential applicability to even larger graphs in the future. However, for dense graphs, we did not observe any significant reduction in the number of time steps required and corresponding space-time overhead. This limitation, however, can be mitigated by transforming dense graphs into sparse ones using local complementation, as discussed in Section 6.2.

The testing at the current scale shows comparable results between the MinCut Mapper and the random mapper. Also, the performance of Barycenter Mapper and Random Mapper on dense graphs is difficult to determine which one performs better since one may exhibit slight advantages in certain sizes while the other may excel in others, though the scale of the graph makes it not easy to discern.

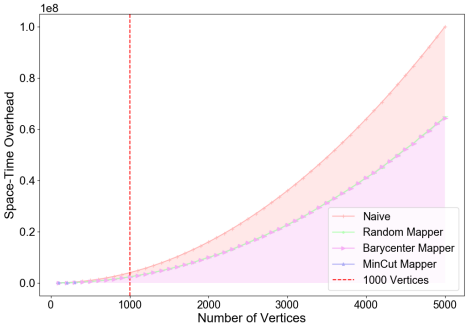


(a) Sparse graph with $n \log n$ edges

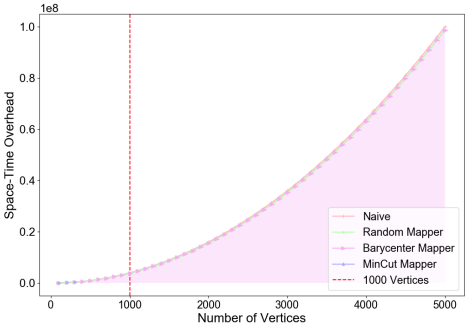


(b) Dense graph with $n^2/\log n$ edges

Figure 5.5: Performance of Substrate Scheduler on graphs of varying sizes. The horizontal axis denotes the number of vertices present in the graph, while the vertical axis represents the corresponding number of time steps.



(a) Sparse graph with $n \log n$ edges



(b) Dense graph with $n^2/\log n$ edges

Figure 5.6: Performance of Substrate Scheduler on graphs of varying sizes based on their space-time overhead. The horizontal axis represents the graph size (number of vertices), while the vertical axis displays the space-time overhead.

Chapter 6

Conclusion

6.1 Conclusion

To the best of our knowledge, this study is one of the earliest explorations of this problem, introducing a feasible approach to optimize the overhead of generating fault-tolerant graph states.

Regarding the spatial aspect, this thesis introduces and compares three distinct layout designs, evaluating their data-ancilla ratios and associated overhead.

In terms of time cost, this thesis proposes a compiler module that maps fault-tolerant graph states onto surface code-level operations. Employing optimization techniques such as stabilizer generator reduction, qubit allocation, and operation scheduling, the aim is to minimize time costs and space-time overhead. Evaluation results demonstrate that this approach, with chosen layout design, reduces the space-time overhead required for generating fault-tolerant graph states to a level below the exponential growth rate of unoptimized methods, and allows specific graph types to maintain a constant-level space-time overhead. Furthermore, the result shows that this approach scales well for sparse graphs, making it suitable for handling large-scale graph states.

This thesis also investigates the impact of mapping methods within the constraint of a 2-tile-one-qubit one-bus layout. Two heuristics, namely the Barycenter Mapper and the MinCut Mapper, are developed for mapping purposes. By analyzing their effects on time cost and space-time overhead, it is evident that the proposed methods generally outperform the random mapping method in terms of lower time cost, which reaches a theoretical minimum, especially for certain graph types like the path graph. In other graph types, their performance is comparable to random mapping. It should be noted that for graphs that have a density lower than $n^2/logn$, it becomes challenging to determine a clear advantage for either mapper, as they both demonstrate instances of good and poor performance. However, when the graph density reaches a certain threshold, the Barycenter Mapper proposed in this thesis demonstrates the best optimization results. This to some extent indicates that the proposed method

excels in handling dense graphs. However, overall, compared to sparse graphs, the reduction in overhead resulting from this optimization is not significant.

6.2 Future Work

To further enhance the performance of the implementation in this thesis and establish more robust optimality bounds, we propose several promising extensions to this work:

Stabilizer generator reduction In this work, stabilizer generator reduction is employed as an optimization approach to minimize the time cost associated with graph state generation. This involves approximating the maximum independent set of the input graph state with a maximal independent set, effectively reducing the required number of stabilizer checks. However, it is important to note that although this minimizes the number of stabilizer checks, it does not necessarily minimize the preparation depth itself. For certain algorithm-specific graph states, it is plausible that another stabilizer generator reduction exists, which involves more stabilizer checks but can be executed in parallel over fewer time steps, resulting in a lower overall preparation depth. Future research could investigate the integration of stabilizer generator reduction with alternative criteria to establish more robust guarantees for achieving optimal preparation depth.

Local complementation Local complementation (LC) is an operation that transforms graph states into a diverse set of equivalent graph states with varying structures, using local Clifford gates [45]. In the context of the surface code, single qubit local Clifford gates have low overhead and can be absorbed by the final measurement, making the additional overhead negligible.

An optimization potential in our method is to reduce the preparation depth of the algorithm-specific graph state by applying LC to modify the input state. While computing the local minimum degree of a graph is NP-complete and hard to approximate [46], LC can effectively reduce the generation cost of graph states by altering the graph's feature such as connectivity [47]. For instance, transforming a complete graph into a star graph can reduce the number of time steps required for its preparation.

Following stabilizer generator reduction on an LC-optimized graph state, further optimization can be achieved by using LC to minimize the connectivity constraints necessary for stabilizer checks. Whether these two LC optimization steps differ significantly is currently unknown.

Other ancilla bus architectures As demonstrated in 4.2, the constraints of the one-bus ancilla bus architecture enable precise optimization of vertex-to-qubit mappings and have a high data-to-ancilla qubit ratio of 50%. However, alternative layout

designs can be explored from two perspectives: To enhance the potential for parallel measurements of stabilizer generators or accommodate specific hardware requirements.

While some alternative architectures may have a lower data-to-ancilla qubit ratio, they might be capable of preparing graph states with fewer time steps. Thus quantitative comparisons between different architectures regarding data-to-ancilla qubit ratio and space-time overhead require further research.

Furthermore, in future advancements, a dynamic version of the algorithm could be created to generate graph states that surpass the capacity of available tiles, which is equal to the number of logical qubits encoded using the existing physical qubits of the quantum device. To handle the augmented complexity and scalability of the system, a distinct mapping strategy would be necessary.

Optimizing mapping methods For the existing one-bus layout design, a potential improvement approach could involve weighting the connectivity of the graph after stabilizer generator reduction. For instance, one possibility is to utilize a weighted version of the Barycenter heuristic or other similar techniques. Currently, the existence of an algorithm that can optimally map the vertices of an arbitrary graph state to qubits, minimizing preparation depth, is uncertain. This uncertainty applies within the proposed one-bus layout design, which restricts the simultaneous execution of parity checks. While more structurally complex ancilla bus architectures, such as two buses, may not face the same limitations as the one-bus design due to their geometric characteristics, the extent of the intractability of this problem for such intricate architectures is still unknown.

6.3 Code Availability

Substrate Scheduler's source code, documentation, and sample configurations are fully available online [1], under the MIT license.

[1] `https://github.com/sfc-aqua/gosc-graph-state-generation`

Acknowledgement

First, I would like to express my deepest gratitude to my advisor, Professor Rodney Van Meter, for his unwavering support, guidance, and words of encouragement during my bachelor's and master's studies. He showed me the world of quantum computing, and the experience of learning alongside him in courses, discussions, and research projects has ignited my passion to pursue a career as a researcher. He is always very supportive and has provided me with numerous amazing opportunities that I could never have gained without him. I can confidently say that I would not be the person I am today without him. I would also like to express my gratitude to the other faculty members of AQUA (Advancing Quantum Architecture Research Group), Dr. Kusumoto Hiroyuki, Dr. Shigeya Suzuki, Dr. Shota Nagayama, Dr. Takahiko Satoh, Dr. Bernard Ousmane Sane, and especially Dr. Michal Hajdušek for his significant assistance in completing this project and helping me with the previous paper. He has patiently answered numerous questions that arose during the process.

I would also like to express my gratitude to all the members of AQUA, both current and former. I used to be able to list each of your names individually, but our team has grown so large that it is now impossible for me to include everyone and I know that our team will continue to grow stronger in the future. I am proud to be a member of AQUA. The experiences I have shared with all of you have, without a doubt, been the most valuable moments in my life. I would like to extend a special thanks to Naphan Benchasattabuse for his guidance and assistance in completing this project. I would also like to thank them for the support they have given me during my time as the AQUA KG leader.

I would like to extend my gratitude to the faculty and all members of the RG Joint Research Group and faculties of Keio University for their invaluable advice and guidance during my time at Keio. I am especially grateful to Professor Mitsugi Jin for his belief in my abilities, which led to the opportunity to work as a teaching assistant.

In addition, I would like to express my gratitude to Professor Simon Deviitt and PhD student Darcy Morgan from the University of Technology Sydney for their support and efforts in completing this research project. I also want to thank Zapata Computing for their assistance in code integration.

I would also like to thank all the incredible artists whose work has had a profound impact on me, offering immense inspiration and encouragement in both my research and personal life. Their artistic creations have transformed my perspective of the

world, and reminded me that ”there must be some way out of here.”

I would like to express my gratitude to my friends, both within and outside of Keio University, for being my friends. Lastly, I want to extend my deepest appreciation to my family, especially my mother. Their selfless support is the best thing I ever have had. Though some of them are no longer with us, their kind and gentle nature will forever remain in my heart. I hope that they can witness the accomplishments I will achieve. I am also grateful for the companionship of my cat during the dissertation writing process.

Appendix A

Appendix

The pseudocode for the MinCut mapper (see Section 4.3.3) is provided below as Algorithm 1.

Algorithm 1 Algorithm for MinCut Mapper

Input: g: the input graph

Output: *mapping*: the indexes of the vertices that have been mapped to the logical qubits

```
mapping ← Array[]                                   ▷ Initialize the mapping array
G ← deepcopy(g)                                     ▷ Create a copy of the input graph
function MIN_CUT(graph, component)
    cut_edges_list ← Array[]                        ▷ Initialize the list of cut edges
    shortestLen = very_large_value
    shortestV ← Array[]
    for i = 0 to num_of_repetitions do              ▷ Loop over iterations
        currentLen, currentV ← karger(graph)        ▷
Apply Karger's algorithm to find the minimum cut. The function karger takes a
graph as input and returns two values: the number of edges that need to be cut
currentLen, and the vertices of those edges currentV.
        if currentLen < shortestLen then
            shortestLen ← currentLen
            shortestV ← deepcopy(currentV)
        end if
    end for
    for edges in currentV do
        cut_edges_list.append(edges)
    end for
    graph.remove_edges_from(cut_edge)               ▷ Remove the cut edges from the
graph
    return graph
end function
function MAPPING_MIN(G)
    for component in connected_subgraphs(G) do
        if (number_of_vertices (component) > 2) then
            G ← min_cut(G, component)
            return mapping_min(G)                   ▷ Recursively call the function on the
updated graph
        else
            mapping.append(component.vertices)
            G.remove_vertices_from(component.vertices)
            return mapping_min(G)                   ▷ Recursively call the function on the
updated graph
        end if
    end for
end function
mapping_min(G)
return mapping
```

Bibliography

[1] Sitong Liu, Naphan Benchasattabuse, Darcy QC Morgan, Michal Hajdušek, Simon J. Devitt, and Rodney Van Meter. A substrate scheduler for compiling arbitrary fault-tolerant graph states, 2023. arXiv:1808.02892.

[2] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien. Quantum computers. *Nature*, 464(7285):45–53, 2010.

[3] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[4] Daniel Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation. In *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*, volume 68, pages 13–58, 2010.

[5] Simon J Devitt, William J Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013.

[6] Barbara M. Terhal. Quantum error correction for quantum memories. *Rev. Mod. Phys.*, 87:307–346, Apr 2015.

[7] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2, 01 2018.

[8] Christopher Chamberland and Kyungjoo Noh. Very low overhead fault-tolerant magic state preparation using redundant ancilla encoding and flag qubits. *npj Quantum Information*, 6(1):91, 2020.

[9] K. J. Satzinger, Y. J Liu, A. Smith, C. Knapp, M. Newman, C. Jones, Z. Chen, C. Quintana, X. Mi, A. Dunsworth, C. Gidney, I. Aleiner, F. Arute, K. Arya, J. Atalaya, R. Babbush, J. C. Bardin, R. Barends, J. Basso, A. Bengtsson, A. Bilmes, M. Broughton, B. B. Buckley, D. A. Buell, B. Burkett, N. Bushnell, B. Chiaro, R. Collins, W. Courtney, S. Demura, A. R. Derk, D. Eppens, C. Erickson, L. Faoro, E. Farhi, A. G. Fowler, B. Foxen, M. Giustina, A. Greene, J. A. Gross, M. P. Harrigan, S. D. Harrington, J. Hilton, S. Hong, T. Huang, W. J. Huggins, L. B. Ioffe, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi,

T. Khattar, S. Kim, P. V. Klimov, A. N. Korotkov, F. Kostritsa, D. Landhuis, P. Laptev, A. Locharla, E. Lucero, O. Martin, J. R. McClean, M. McEwen, K. C. Miao, M. Mohseni, S. Montazeri, W. Mruczkiewicz, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, T. E. O'Brien, A. Opremcak, B. Pató, A. Petukhov, N. C. Rubin, D. Sank, V. Shvarts, D. Strain, M. Szalay, B. Villalonga, T. C. White, Z. Yao, P. Yeh, J. Yoo, A. Zalcman, H. Neven, S. Boixo, A. Megrant, Y. Chen, J. Kelly, V. Smelyanskiy, A. Kitaev, M. Knap, F. Pollmann, and P. Roushan. Realizing topologically ordered states on a quantum processor. *Science*, 374(6572):1237–1241, 2021.

[10] Madhav Krishnan Vijayan, Alexandru Paler, Jason Gavriel, Casey R. Myers, Peter P. Rohde, and Simon J. Devitt. Compilation of algorithm-specific graph states for quantum circuits, 2022.

[11] M. Hein, J. Eisert, and H. J. Briegel. Multiparty entanglement in graph states. *Phys. Rev. A*, 69:062311, Jun 2004.

[12] M. Hein, W. Dür, J. Eisert, R. Raussendorf, M. Van den Nest, and H. J. Briegel. Entanglement in graph states and its applications. *ArXiv:quant-ph/0602096*, 2006.

[13] Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86:5188–5191, May 2001.

[14] Michael A. Nielsen. Cluster-state quantum computation. *Reports on Mathematical Physics*, 57(1):147–161, 2006.

[15] Madhav Krishnan Vijayan, Simon Devitt, Peter Rohde, Casey Myers Alexandru Paler, Jason Gavriel, Michał Stęchły, Scott Jones, and Athena Caesura. Jabalizer.jl. https://github.com/QSI-BAQS/Jabalizer.jl.

[16] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. PhD Thesis Caltech, 1997.

[17] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, Sep 2012.

[18] A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, Jan 2003.

[19] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, sep 2002.

[20] D. Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, Dec 2012.

[21] Daniel Litinski. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum*, 3:128, March 2019.

[22] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[23] Douglas Brent West. *Introduction to graph theory*, volume 2. Prentice Hall, 2001.

[24] Robert Raussendorf and Jim Harrington. Fault-tolerant quantum computation with high threshold in two dimensions. *Phys. Rev. Lett.*, 98:190504, May 2007.

[25] David S. Wang, Austin G. Fowler, and Lloyd C. L. Hollenberg. Surface code quantum computing with error rates over 1%. *Phys. Rev. A*, 83:020302, Feb 2011.

[26] H. Bombin. Topological order with a twist: Ising anyons from an abelian model. *Physical Review Letters*, 105(3):030403, Jul 2010.

[27] Xinlan Zhou, Debbie W. Leung, and Isaac L. Chuang. Methodology for quantum logic gate construction. *Phys. Rev. A*, 62:052316, Oct 2000.

[28] Andrew M. Steane. Overhead and noise threshold of fault-tolerant quantum error correction. *Physical Review A*, 68:042322, 2003.

[29] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. Validating quantum computers using randomized model circuits. *Phys. Rev. A*, 100:032328, Sep 2019.

[30] Alexandru Paler, Ilia Polian, Kae Nemoto, and Simon J Devitt. Fault-tolerant, high-level quantum circuits: form, compilation and description. *Quantum Science and Technology*, 2(2):025003, apr 2017.

[31] Adam Holmes, Yongshan Ding, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic T. Chong. Resource optimized quantum architectures for surface code implementations of magic-state distillation. *Microprocessors and Microsystems*, 67:56–70, 2019.

[32] R Raussendorf, J Harrington, and K Goyal. Topological fault-tolerance in cluster state quantum computation. *New Journal of Physics*, 9(6):199, jun 2007.

[33] R. Raussendorf, J. Harrington, and K. Goyal. A fault-tolerant one-way quantum computer. *Annals of Physics*, 321(9):2242–2270, 2006.

[34] Michael Newman, Leonardo Andreta de Castro, and Kenneth R. Brown. Generating fault-tolerant cluster states from crystal structures. *Quantum*, 4:295, jul 2020.

[35] Anbang Wu, Gushu Li, Hezi Zhang, Gian Giacomo Guerreschi, Yufei Ding, and Yuan Xie. Mapping surface code to superconducting quantum processors, 2021.

[36] David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, page 21–30, USA, 1993. Society for Industrial and Applied Mathematics.

[37] M. R. Garey and D. S. Johnson. Crossing number is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[38] Peter Eades and Nicholas C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.

[39] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11:109–125, 1981.

[40] Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.*, 1:1–25, 1997.

[41] Xiao Yu Li and Matthias F. Stallmann. New bounds on the barycenter heuristic for bipartite graph drawing. *Information Processing Letters*, 82(6):293–298, 2002.

[42] Erkki Mäkinen and Harri Siirtola. The barycenter heuristic and the reorderable matrix. 29:357–363, 2005.

[43] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. Graphical description of the action of local clifford transformations on graph states. *Phys. Rev. A*, 69:022316, Feb 2004.

[44] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, 2008.

[45] Jeremy C. Adcock, Sam Morley-Short, Axel Dahlberg, and Joshua W. Silverstone. Mapping graph state orbits under local complementation. *Quantum*, 4:305, August 2020. arXiv: 1910.03969.

[46] David Cattanéo and Simon Perdrix. Minimum Degree up to Local Complementation: Bounds, Parameterized Complexity, and Exact Algorithms. *arXiv:1503.04702 [quant-ph]*, 9472:259–270, 2015. arXiv: 1503.04702.

[47] Adan Cabello, Lars Eirik Danielsen, Antonio J. Lopez-Tarrida, and Jose R. Portillo. Optimal preparation of graph states. *Physical Review A*, 83(4):042314, April 2011. arXiv: 1011.5464.