Graphical calculus for Tensor Network Contractions

Candidate no: **1060926**

Word count: **19112** [1]

A dissertation submitted for the degree of

MSc in Advanced Computer Science

Trinity Term 2022

[1]This word count was computed by the command `pdftotext main.pdf -enc UTF-8 -| tr -cd '0-9A-Za-z \n' | wc -w`

Abstract

Quantum computing is an emerging field of research that harnesses the laws of quantum mechanics to solve computationally hard problems. In addition to developing better and bigger quantum computers, a significant challenge is developing techniques to benchmark these devices. Thus, pushing the barrier of classical simulation goes hand in hand with the development of quantum computers. In this dissertation, we hope to leverage the existing work on quantum graphical languages such as the ZX-calculus to aid one of the most prominent classical simulation tools, tensor network methods. By their very nature, graphical languages have a more modular representation of quantum circuits, which could be optimized using the associated rewrite rules of the calculi. We investigate how effective the existing procedures are at enhancing tensor network contractions and propose new strategies based on our observations. Furthermore, we evaluate our strategies using a variety of circuits, including the Sycamore circuits used by Google to demonstrate quantum supremacy in 2019.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Quantum computing refers to harnessing quantum mechanical phenomena such as superposition, interference, and entanglement to perform computation. It is an exciting field of research widely believed to help solve computationally hard problems much faster than classical computers [1–3]. As the size of quantum devices grows, benchmarking them becomes increasingly important. Improving the classical simulation of quantum circuits not only enables building better quantum computers but also helps in characterizing quantum advantange.

Tensor networks have been developed as a helpful framework for deriving mathematical descriptions of quantum many-body wavefunctions [4–9]. In addition to being a helpful formalism for describing quantum many-body systems, they also serve as powerful numerical tools and are the basis of many simulation algorithms. Since their inception, they have found use in applications such as quantum chemistry [10–14], holography [15–20], machine learning [21–25] and simulation of quantum circuits [26–31].

Quantum graphical languages like ZX-/ZH-calculus [32–34] have a similar motivation to Tensor Networks in that they use a graphical formalism to reason about quantum systems. Quantum circuits can be represented using a modular data structure by converting them

into ZX-/ZH-diagrams. Such transformations allow for better optimization of quantum circuits [35, 36] and often lead to a more efficient representation of the underlying computations with no circuit equivalent.

In this dissertation, we wish to enhance the tensor-network-based simulation of quantum circuits by using quantum graphical calculi. Our framework uses graphical-calculi-based representations as a bridge between quantum circuits and tensor networks. We convert quantum circuits into their equivalent ZX-/ZH-diagrams and then use graphical simplifications to obtain a compressed representation. This representation is then converted into a standard tensor network representation, potentially leading to more efficient contractions.

Existing graphical simplifications have proven to be useful for optimizing quantum circuits. However, using the same procedures for simplifying tensor networks might lead to an increase in the contraction complexity. In our work, we develop a heuristic measure to guide the application of simplification procedures and help find tensor networks with efficient contraction orders. We also develop new graphical simplification strategies using our proposed measure, designed explicitly for enhancing tensor-network-contractions.

The rest of the dissertation is organized as follows:

In **Chapter 2**, we give a background on tensor networks and quantum computation. Then we describe quantum circuit simulation and detail the Sycamore supremacy experiment. The chapter concludes with a description of a tensor-network-based simulation framework and a parameterization scheme for the contraction complexity of tensor networks. In **Chapter 3**, we give an overview of graphical calculi for quantum computation, namely ZX and ZH calculus. We present the rewrite rules of the calculi, which enable diagrammatic reasoning about quantum computation, and also describe graph-based simplification methods. In **Chapter 4**, we present our chosen representation of tensor networks enabling an efficient conversion between graphical diagrams and tensor networks. Then a description

of our proposed heuristic measure for quantifying contraction complexity is provided. We then present three heuristic-based-strategies to guide the application of graphical simplification procedures. Finally, we present the results of applying our strategies on Sycamore supremacy circuits and discuss observed trends. **Chapter 5** evaluates the proposed strategies on more classes of quantum circuits and concludes with a discussion on the trends we observed. **Chapter 6** summarizes our results and outlines possible research directions for future work.

Chapter 2

Background

In this chapter we give a brief background on topics relevant to the dissertation. We begin by describing Tensor Networks, the underlying graphical language for reasoning about them, and their ability to capture fundamental linear algebra concepts rigorously. We then summarize Quantum Computation with the relevant notation and its connection to Tensor Networks. Finally, we introduce Graphical Calculi for Quantum Computation and present fundamental rewrite rules for the calculi.

2.1 Tensor Networks

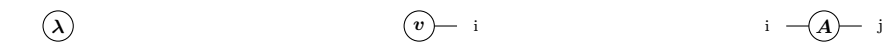
Tensors are objects that encapsulate and generalize the idea of multilinear maps, i.e., functions of multiple parameters that are linear with respect to every parameter. Formally, we define a tensor as follows:

Definition 2.1.1. (Tensor) A rank-r tensor T is an r-dimensional array of size $\mathbf{d} = (d_1, d_2, \cdots, d_r)$ indexed by the Cartesian product of some set of indices , such that

$$T_{i_1, i_2, \cdots, i_r} \in \mathbb{C}$$

where $(i_1, i_2, \cdots, i_r) \in [d_1] \times [d_2] \times \cdots [d_r]$ and $[d] = \{j \in \mathbb{Z} | 1 \leq j \leq d\}$

Although we presented the definition with all indices clubbed together in subscript, a widely followed convention is to place input indices as superscript and output indices as subscript. There exists an intuitive graphical language for reasoning about tensors with the earliest known use attributed to Roger Penrose in the early 1970s [4]. In such a formalism, a tensor can be denoted by any shape such as a box, oval or triangle, with zero or more legs attached. The legs correspond to the indices of the tensor and the *rank* of a tensor is the number of indices. Thus a rank-0 tensor is a scalar (λ), a rank-1 tensor is a vector (v_i) and a rank-2 tensor is a matrix (A^i_j). The dimension of an index specifies the number of values that index can take. For instance a vector with four entries can be represented by a rank-1 tensor with its index having dimension 4.



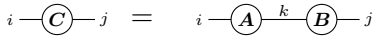
(a) scalar (b) vector (c) matrix

Figure 2.1: Tensor diagrams

A computation involving tensors generally involves some of the indices being *contracted*. A contraction refers to the sum over all the possible values of the repeated indices of a set of tensors. For example, the contraction of a rank-2 tensor A^i_k with another rank-2 tensor B^k_j is given by the following,

$$C_{ij} = \sum_k A^i_k B^k_j \tag{2.1}$$

the corresponding graphical notation for a contraction is denoted by



(2.2)

Here we contracted the common edge between tensors A and B to get a new tensor C. We can formalise this operation by defining an edge contraction on a graph:

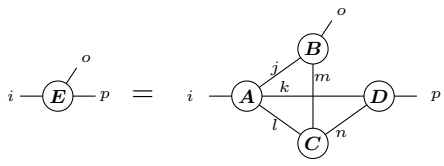
Definition 2.1.2. (Edge Contraction) Edge contraction on a graph G removes an edge uv

from the graph such that the vertices u and v are replaced by a new vertex w and all edges previously incident upon u and v, apart from the common edge, are now incident upon w.

Graphically, each edge contraction removes common edges between pairs of tensors, if any, and represents a product operation on the corresponding tensors, in which one takes the inner product over common indices or an outer product if there are no common indices. More complicated contractions like the following:

$$E^i_{op} = \sum_{jklmn} A^i_{jkl} B^{jm}_o C^{lm}_n D^{kn}_p \tag{2.3}$$

have an intuitive and easy to understand depiction in the graphical notation



(2.4)

Another fundamental operation on tensors, in addition to contraction of indices, is the *tensor product*. Tensor product is a generalisation of outer product of vectors where the value of the resultant tensor on a given set of indices is the element-wise product of the values of each constituent tensor. Explicitly, tensor product of two tensors A and B is given by the following:

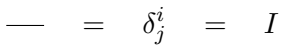
$$[A \otimes B]_{i_1,i_2\ldots,i_m,j_1,j_2\ldots,j_n} = A_{i_1,i_2\ldots,i_m} \cdot B_{j_1,j_2\ldots,j_n} \tag{2.5}$$

Tensor product in the diagrammatic notation is simply represented by two tensors being placed next to each other. The value of a network containing disjoint tensors is simply the product of the constituent values.

$$[A \otimes B] \quad = \quad \begin{array}{c} A \\ B \end{array} \tag{2.6}$$

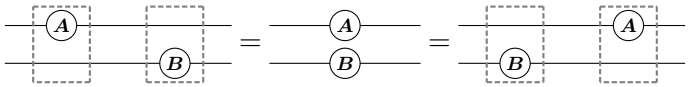
In the above diagrams, we have used wires to denote contraction between two tensors. But

we can interpret wires as tensors themselves. An idenity wire is defined as:



(2.7)

We can use tensor product with identity wires to deform the diagram such that tensors can freely move past each other:



(2.8)

which holds true because of the following relation:

$$[A \otimes I][I \otimes B] = [A \otimes B] = [I \otimes B][A \otimes I] \tag{2.9}$$

In addition to the identity wire, there are two more wire tensors that are commonly used in the graphical notation. The first is the *cup* tensor and the second is the *cap* tensor.

(a) Cup tensor

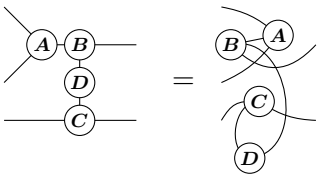
(b) Cap tensor

Figure 2.2: Wire tensors.

The cup and cap tensors allow us to deform a wire arbitrarily so long as its ends remain intact. Diagrammatically this is captured by the *snake equation*:

(2.10)

which enables us to move around tensors in our diagrams without changing their meaning. To demonstrate this, let's consider the contraction between four tensors A, B , C and D. The diagram obtained on the right represents the same computation as the diagram presented on left.



(2.11)

Thus a diagram representing a contraction of tensors can be represented by an undirected graph wherein vertices correspond to the tensors and edges correspond to contraction between the tensors. This graph is known as a *Tensor Network*. Formally:

Definition 2.1.3. (Tensor network) A tensor network (G, M) is an undirected graph G with edge weights w and a set of tensors $M = \{M_v | v \in V(G)\}$ such that M_v is a $|N_v|$-rank tensor having $2^{deg(v)}$ entries, where $|N_v|$ denotes the cardinality of neighbourhood of vertex v and $deg(v) = \sum_{u \in N(v)} w(\{u, v\})$ is the weighted degree of v. Each edge e corresponds to an index $i \in [2^{w(e)}]$ along with which the adjacent tensors are to be contracted.

2.2 Quantum Computation

In this section, we will discuss the basics of quantum computation namely the qubit, quantum gates and measurement and then describe the most widely used model for quantum computation, namely the *Quantum Circuit Model*. We also briefly outline how quantum circuits can be cast naturally as Tensor Networks.

2.2.1 Qubit

The fundamental building block of quantum computation is a *qubit* i.e. the quantum analogue of a (classical) bit. Where the state of a bit is simply represented by 0 or 1, the state of a qubit is essentially a unit vector in the two-dimensional complex Hilbert space. Quantum computation is often presented using Dirac notation which has two types of representation for a vector, namely *bra* and *ket*. A vector ψ is denoted by a *ket* like $|\psi\rangle$ and the dual of a vector ϕ is denoted by a *bra* like $\langle\phi|$. For a system of qubits, we have a computational basis

state comprising of $|0\rangle$, $|1\rangle$, and their duals $\langle 0|$, $\langle 1|$ which are defined as follows,

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \langle 0| = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad \langle 1| = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

In addition to being in one of the computational basis states, the state of a qubit $|\psi\rangle$ can be in a superposition of basis states:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

where α and β are complex numbers denoting the probability amplitudes. Measuring the state of a qubit with respect to the computational basis gives $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. Thus α and β are constrained by the relation:

$$|\alpha|^2 + |\beta|^2 = 1$$

2.2.2 Quantum Gates

Just as classical logical gates are used to perform computation on conventional digital computers, quantum logic gates (or simply quantum gates) are used to perform computations on the state of qubits. Unlike classical logic gates however, all quantum gates are reversible. Quantum gates are unitary operators and can be described as unitary matrices with respect to a basis. In the quantum circuit notation, gates are usually denoted by labelled boxes with input wires on the left and output wires on the right.

As an example, let's consider the Hadamard gate:

$$-\boxed{\text{H}}- = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.12}$$

Hadamard gate is interesting because it has the ability to transform a qubit in a computational

basis state into an equal superposition of computational basis states i.e. the probability of the qubit being in each state is equal.

$$\mathrm{H}\,|0\rangle \;=\; \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix} = |+\rangle$$

$$\mathrm{H}\,|1\rangle \;=\; \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ -1\end{bmatrix} = |-\rangle$$

Morevover, the Hadamard gives rise naturally to another basis called the Hadamard basis comprising of $|+\rangle$ and $|-\rangle$

$$|+\rangle \;=\; \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix} \qquad |-\rangle \;=\; \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ -1\end{bmatrix} \qquad \langle+| \;=\; \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\end{bmatrix} \qquad \langle-| \;=\; \frac{1}{\sqrt{2}}\begin{bmatrix}1 & -1\end{bmatrix}$$

An important family of single qubit gates are the phase shift gates. Phase shift gates are single qubit gates that map basis states $|0\rangle \to |0\rangle$ and $|1\rangle \to e^{i\phi}\,|1\rangle$. Thus they leave the probability of measuring the basis states unchanged but modify the phase of the quantum state. Phase shift gates are represented by:

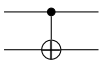
$$P(\phi) = \begin{bmatrix}1 & 0\\ 0 & e^{i\phi}\end{bmatrix} \tag{2.13}$$

A widely used gate-set belongs to the family of Pauli gates, which correspond to the Pauli Matrices:

Gate-set 1. (Pauli)

$$\left\{ -\boxed{\mathrm{I}}- = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix},\quad -\boxed{\mathrm{X}}- = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix},\quad -\boxed{\mathrm{Y}}- = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix},\quad -\boxed{\mathrm{Z}}- = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \right\}$$

So far we have seen examples of single-qubit gates. Gates can act on systems of multiple qubits such that they represent a unitary transformation on the tensor product of the state of the qubits. The most commonly used two-qubit gate is the Controlled-NOT gate (CNOT) :



$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.14}$$

Any unitary operation on a finite number of qubits can be expressed as a quantum circuit consisting of CNOT and single-qubit gates. An important set of gates is called the *Clifford group.*

Definition 2.2.1. (Clifford group) The Clifford group is the set of unitary operators C_n on a finite n-dimensional Hilbert space such that:

$$C_n = \left\{ V \in U_{2^n} | VPV^\dagger = P' \right\} \tag{2.15}$$

where P, P' are Pauli matrices and $\dagger$ denotes the complex conjugate of a matrix.

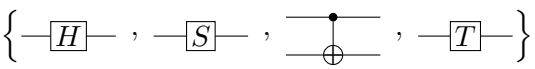
Simply put, the Clifford group consists of all matrices such their product with any Pauli matrix, further multiplied with the inverse of the matrix, results in a Pauli matrix. For single-qubit gates, the Clifford group consists of the gates $\{I, X, Y, Z, H, S\}$ where S is a phase shift gate with phase $\pi/2$:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{bmatrix} \tag{2.16}$$

Just as in classical computing, a finite set of gates suffices to construct a logic circuit computing any Boolean function, e.g. the NAND gate. For quantum computing, there also exists many finite gate sets that allow any arbitrary unitary operator to be approximated to arbitrary

accuracy [37]. One such set is the Clifford+T set [38] which consists of the following gates:

Gate-set 2. (Clifford+T)



where T is a phase shift gate with phase $\pi/4$:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{2.17}$$

Finally we introduce the family of Rotation operator gates which have the effect of rotating the state of a qubit inside the Bloch sphere about a particular axis. The three rotation operator gates $R_x(\theta)$, $R_y(\theta)$ and $R_z(\theta)$ are defined as follows:

Gate-set 3.

$$\left\{ R_x(\theta) = \begin{bmatrix} cos(\frac{\theta}{2}) & -isin(\frac{\theta}{2}) \\ -isin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} cos(\frac{\theta}{2}) & -sin(\frac{\theta}{2}) \\ -sin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{bmatrix}, \quad R_z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix} \right\}$$

2.2.3 Measurement

To get information out of the state of a quantum system, one needs to perform a measurement. A measurement is a non-reversible operation that collapses the quantum state of a system into one of its basis states. The probability of measuring a particular basis state is given by the square of the amplitude of that basis state. For example, if we measure the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ we will get the results $|0\rangle$ and $|1\rangle$ with equal probability. Measurements are often performed at the end of a quantum circuit to gather information about the final state

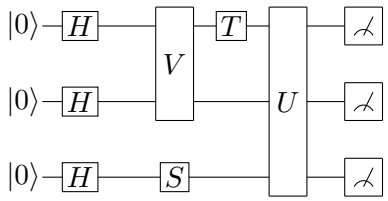


Figure 2.3: A quantum circuit diagram.

of a quantum system in the form of a discrete probability distribution in the computational basis.

2.2.4 Quantum Circuits

Quantum circuit notation is a well known diagrammatic notation for quantum computation associated with the circuit model of quantum computation. Quantum circuits are represented by a sequence of quantum gates and measurements applied to an initial quantum state encoded in a sequence of qubits. The quantum circuit model implicitly assumes that evolution of underlying quantum system happens in discrete steps, as represented by discrete gates and that a system remains in the same state unless acted upon by a gate. This enables the complexity of computations to be readily analysed: if every gate is assumed to take a fixed amount of time or some other quantifiable resource, then the number of gates in the circuit or the number of sequential layers of gates is a measure of complexity.

Gates can be combined by stacking them on top of each other, which denotes the tensor product of corresponding matrices. Alternatively, input of one gate can be plugged into the outputs of another, which denotes matrix multiplication. An example circuit with one, two and three qubit gates is shown in Figure 2.3.

2.2.5 Quantum Circuits as Tensor Networks

Quantum circuits are a special class of tensor networks in which the type and arrangement of tensors are restricted so as to preserve the geometry of the actual hardware implementation. Quantum gates are interpreted as tensors wherease the wires correspond to edges in the tensor network. In addition to gates, we also consider the initial qubit state and measurement as

tensors by interpreting them as a column vector and row vector respectively as shown in Figure 2.4 .

(a) State (b) Effect

Figure 2.4: Diagram (a) above represents the state $|\psi\rangle$ i.e. a tensor with single output index (a column vector) and diagram (b) the effect $\langle\phi|$ i.e. a tensor with a single input index (a row vector)

Measurements are often performed with respect to computational basis. As an example, we cast the quantum circuit in Figure 2.3 as a tensor network as shown in Figure 2.5.

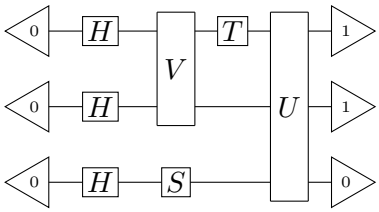


Figure 2.5: A quantum circuit diagram with $|000\rangle$ as the initial state, and $\langle 110|$ as the measured effect.

2.3 Quantum circuit simulation using tensor network methods

Classical simulation plays an indispensable role in understanding and designing quantum devices, as it often is the only means to validate and benchmark existing quantum devices. With quantum devices increasing in size and precision, classical simulation of the corresponding quantum system becomes increasingly challenging. As quantum technologies grow, an interesting crossing point called *quantum supremacy* or *quantum advantange* is after which quantum processors can outperform the most advanced classical computing systems on specialized tasks. A leading candidate for this is the sampling problem from a random quantum circuit. The key idea is that, unlike quantum algorithms (e.g., Shor [1] or Grover [2]) that require deep quantum circuits and high gate fidelities — inaccessible in the near future — to become manifestly advantageous, the task of sampling bit strings from the output of random quantum circuits is expected to be hard to simulate classically even for

lowdepth circuits and low-fidelity gates. Although that problem may not have practical value, it is straightforward to perform on a quantum processor: execute a random sequence of quantum gates and then output the measurement result of each qubit. From an engineering perspective, building a quantum processor of sufficient size and accuracy so that sampling becomes infeasible on a classical computer is still a challenge. The hardness of this task is mainly attributed [39–43] to the expectation that simulating random circuits takes time that grows exponentially with circuit size.

The precise threshold for observing a quantum advantage is nonuniversal and largely depends on the efficiency of classical simulation for each particular combination of circuit model and chip architecture. An important development in the field was when Google announced in 2019 to have demonstrated quantum supremacy using a 53-qubit Sycamore quantum chip [44]. Though initially this task was estimated to take thousands of years on the fastest classical supercomputers, later the simulation time was significantly reduced due to the recent progress on tensor network based quantum simulation algorithms [29, 45, 46]. Tensor network based simulations have been one of the main contenders at pushing the classical barrier. Thus, the race for development of higher qubit count devices with high gate fidelities runs in parallel with a race for development of high-performance simulation techniques for the corresponding quantum systems.

This section aims to provide a review of a tensor-network-based quantum circuit simulation framework that represents one of the state-of-the-art approaches in the field. We begin by first describing the Sycamore supremacy experiment in detail. Then we move our attention to tensor networks and detail a scheme for parameterizing tensor network contraction costs. Finally, we describe how this parameterization scheme was used by Gray and Kourtis to develop their quantum circuit simulation framework using tensor network contractions.

2.3.1 Sycamore Supremacy experiment

In 2019, Google's quantum computing group [44] released their Sycamore circuit experiments and demonstrated *quantum supremacy.* The authors sample from a family of random quantum circuits run on the 53-qubit Sycamore device at increasing depth, measured in terms of "cycles". Every random circuit is composed of m cycles, each consisting of a single-qubit gate layer and a two-qubit gate layer. The whole circuit then concludes with an additional single-qubit gate layer preceding measurement in the computational basis. In the first single-qubit gate layer, single-qubit gates are chosen for each individual qubit independently and uniformly as random from the gate set $\{\sqrt{X}, \sqrt{Y}, \sqrt{W}\}$, where upto a global phase these gates are

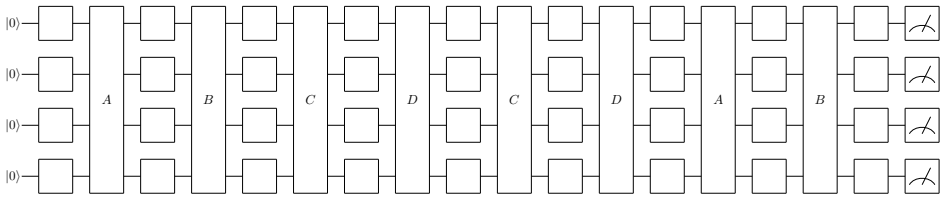
Gate-set 4.

$$\left\{ \sqrt{X} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix}, \quad \sqrt{Y} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}, \quad \sqrt{W} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -\sqrt{i} \\ \sqrt{-i} & 1 \end{bmatrix} \right\}$$

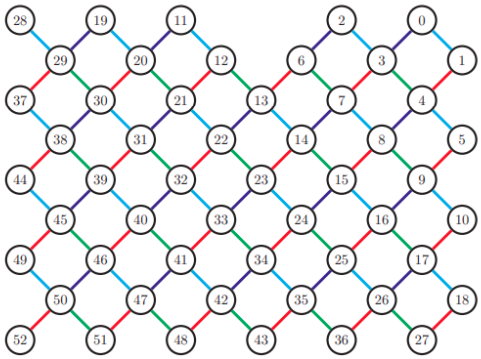
For each successive single-qubit layer, gates for each individual qubit are again chosen from the same set 4 but now excluding the gate applied on the same qubit in previous cycle. This prevents simplifications while simulation of the circuits. For the two-qubit gate layers, two-qubit gates are applied according to a specified pairing of qubits in different cycles. There are four different patterns of pairings, and the 8-cycle pattern ABCDCDAB is repeated.

Each gate in this family can be decomposed into four phase shift gates (see 2.13) described by three free parameters and the two-parameter fermionic simulation gate

$$fSim(\theta, \phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -isin(\theta) & 0 \\ 0 & -isin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & e^{-i\phi} \end{bmatrix} \tag{2.18}$$



(a)



(b)

Figure 2.6: (a) Schematic diagram of Sycamore circuit (b) Layout of 53 qubits and two-qubit gates between them.

For the experiments, the $fSim$ gates were tuned such that $\theta \approx \pi/2$ and $\phi \approx \pi/6$. $fSim(\pi/2, \pi/6)$ can be decomposed into the product of a fractional iSWAP and controlled phase gate [44] defined as follows:

$$iSWAP(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -isin(\theta) & 0 \\ 0 & -isin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$cphase(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix}$$

A schematic description of the complete two-qubit gate is shown in Figure 2.7.

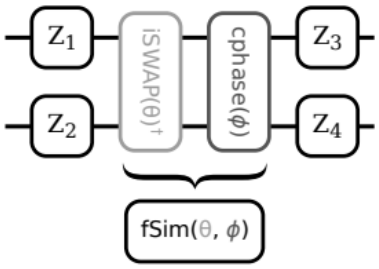


Figure 2.7: The two-qubit gate decomposition used in Sycamore circuits

The authors showed that their chosen gates constitute a universal gate-set and thus can be used to perform universal quantum computation (with arbitrary parameter values for gates). In these circuits, however, the two qubit gates are not completely random, but are determined by qubit pair and cycle number.

Due to the noise prevalent in quantum gates, the quantum device samples from a noisy version of the ideal distribution. However, the distribution can be reliably recovered by taking a sufficient number of samples. A measure is needed to assess the closeness of the output distribution to the ideal distribution. The ideal distibution for this problem is called the Porter-Thomas distribution. The authors in [44] used linear cross-entropy benchmarking fidelity (XEB) [47] for this purpose. XEB is defined as:

$$XEB = 2^n \langle p_I(x) \rangle - 1 \tag{2.19}$$

where $p_I(x)$ is the ideal distribution, x is the sample and n the number of qubits. The expectation is taken over the output distribution procured from the circuits. XEB is 0 when the output distribution is uniform and 1 when the output distribution is the ideal distribution. Simplified quantum circuits run on Sycamore achieved an XEB of approximately 0.2% at 20 cycles [44]. It was argued from numerical evidence that the aforementioned random quantum circuits had also achieved an XEB of approximately 0.2%. However, this was not directly verified as simulating these circuits was estimated to be infeasible.

2.3.2 Parameterization of tensor network contraction

Tensor networks are a powerful tool in the study of classical and quantum many-body systems. Tensor network based approaches often encode computations into a graphical structure such that the indices can be contracted to get a minimal representation of the computation.This contraction of indices to get the smallest possible tensor network representation is called *Tensor Network Contraction*. For the contraction procedure, different orders can lead to exponential, linearized or polynomial-time algorithms depending upon how the size of the tensor network decreases. Tensor network contraction is #-P hard in general [48].

We now present a parameterization scheme presented by Bryan O' Gorman [49] to characterize the complexity of contracting a tensor network with regards to its contraction orders. The author used tree embeddings of a tensor network, which they called *contraction trees*, such that they could express the time complexity of contraction directly as a property of contraction trees. Most of the notation used in this section is inspired from the original paper [49]. Formally, a tree embedding is defined as follows:

Definition 2.3.1. (Tree embedding) A tree embedding of a graph G is a tuple (T, b) of a binary tree T and a bijection $b : V(G) \to V(T)$ between vertices of G and leaves of T.

In their work, they [49] assume tensor network contractions are performed as a series of matrix multiplications. The tree embedding of a tensor network is created by assigning each leaf of the contraction tree to be a tensor in the original tensor network and each internal node to an intermediate contraction. We give an example tensor network along with two possible contraction trees in Figure 2.8.

Contraction trees exactly model the matrix multiplications done by a contraction algorithm in an abstract way. This enables directly expressing the time complexity of contraction as a property of contraction trees. Contraction trees also captures the the space needed by a matrix-multiplication-based contraction algorithm. Another important characterization used is called *routing*. Let $S_{u,v}$ be the unique path between the leaves $b(u)$ and $b(v)$ of T.

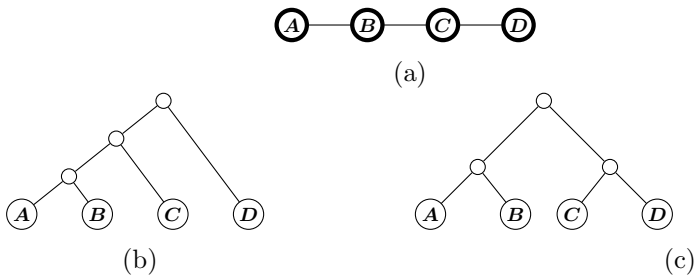


Figure 2.8: (a) A tensor network with 4 tensors (b) Contraction tree for the tensor network (c) Contraction tree for the tensor network with a different contraction order

Definition 2.3.2. (Routing) An edge e of a tensor network (G, M) between two vertices $u, v \in V(G)$ is said to be routed through a vertex (resp. an edge) of a contraction tree, if the vertex (resp. the edge) is included in the unique path $S_{u,v}$ joining the leaves of contraction tree corresponding to u and v.

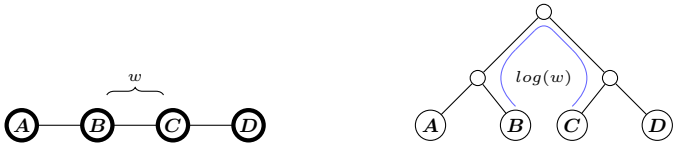


Figure 2.9: Logarithm of bond dimension being routed through the unique path connecting two leaves in contraction tree.

Each edge in the original tensor network connecting two tensors, is assigned a weight equal to the logarithm of the bond dimension. Figure 2.9 shows how an edge in the original tensor network is routed through in a corresponding contraction tree. The congestion of a vertex of the contraction tree is the sum of weights of edges in the original tensor network that are routed through it. Similarly the congestion of a contraction tree edge is the sum of weights of edges in the original tensor network routed through it. The vertex congestion of a graph G, denoted by $vc(G)$, is the minimum over contraction trees of the maximum congestion of a vertex, and similarly for the edge congestion, denoted by $ec(G)$.

We state the following result from the paper [49]

Theorem 2.3.3. A tensor network (G,M) can be contracted in time $n2^{vc(G)+1}$ and space $n2^{vc(G)+1}$.

We can now take an example and see if congestion of a tensor network is a good measure of the time and space requirements for a particular simulation. The space required to store a tensor network, (G, M) can be calculated by summing up the space required to store the matrices in the tensor network. Each tensor $\mathbf{M}_v$ contains 2^{deg_v} numbers. Now for each vertex $v \in G$, the congestion $con(e)$ of the edge $e \in E(T)$ adjacent to $b(v)$ gives the size of the tensor $\mathbf{M}_v$ because $con(e) = \sum_{u \in V(G)} w(v, u) = deg_v$ and therefore $2^{con(e)} = 2^{deg_v}$.

Now, for the contraction of two tensors, assume we are contracting a (d_L, d_M) dimensional tensor $\mathbf{M}_{v_1}$ with a (d_M, d_R) dimensional tensor $\mathbf{M}_{v_2}$ along the shared dimesnion d_M (see Figure 2.10). The space required to store the input tensors is $d_M(d_L + d_R)$ and the output tensors is $d_L d_R$. Assuming new memory is allocated for the output tensor total space is given by $d_M(d_L + d_R) + d_L d_R$. The author assumed the contractions were performed as a series of matrix multiplications which means that the implementation of the contraction algorithm will take time $d_L \cdot d_M \cdot d_R$. Now consider this contraction from the viewpoint of the tensor network. In the network, matrix $\mathbf{M}_{v_1}$ is represented by vertex v_1 and matrix $\mathbf{M}_{v_2}$ is represented by vertex v_2. The edge connecting v_1 and v_2 in G has a bond dimension of $2^{w(v1,v2)}$. The product of bond dimensions of $\mathbf{M}_{v_1}$ with tensors besides v_2 is $2^{deg_{v_1} - w(v_1,v_2)}$ and similarly product of bond dimensions of $\mathbf{M}_{v_2}$ with tensors besider v_1 is $2^{deg_{v_2} - w(v_1,v_2)}$. The contraction can be done in time $2^{deg_{v_1} - w(v_1,v_2)} \cdot 2^{w(v_1,v_2)} \cdot 2^{deg_{v_2} - w(v_1,v_2)} = 2^{w(v_1,v_2,V(G)\backslash\{v_1,v_2\})}$ where $w(v_1, v_2, V(G)\backslash\{v_1, v_2\})$ is the total weight of edges across the tripartite cut. This is corresponds to the congestion of the vertex $t \in V(T)$ adjacent to both $b(v_1)$ and $b(v_2)$. The output tensor is represented by the contracted vertex $v_{1,2}$ and the size of the correspond output tensor $\mathbf{M}_{v_{1,2}}$ is $2^{w(v_1,v_2,V(G)\backslash\{v_1,v_2\})} = 2^{con(\{t1,t2\})}$

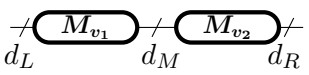


Figure 2.10: Tensor network representing the contraction of a (d_L, d_M) dimensional tensor with a (d_M, d_R) dimensional tensor.

Thus, we see that the congestion of a vertex serves as a metric to approximate the time to do the contraction. and the congestion of the adjacent edge approximates the space of the

resulting contracted tensor. The authors also showed that their framework naturally extends to tensor networks where the underlying graphical structure is a hypergraph i.e. a graph with edges connecting arbitrary number of vertices.

2.3.3 Hyper optimized tensor network contractions

In [45], Gray and Kourtis developed randomized protocols that find contraction paths for arbitrary and large scale tensor networks. They used a hyper-optimization algorithm to enable different contraction strategies for different geometries of tensor networks. Their experiments showed that their model was able to perform state-of-the-art simulations of quantum circuits including simulating Sycamore Supremacy experiments with a speedup of over 10000x compared to Google's earlier estimate.

The authors start by constructing a tensor network representing the computation needed to be performed by the quantum circuit. As we have already described in the previous sections, quantum circuits can be naturally cast as tensor networks and then simulated via contraction. The simplest scalar quantity that can be represented by the tensor network is a "transition amplitude" of one computational basis state to another through a unitary describing the quantum circuit. Computing the transition amplitude is also called *strong simulation*. Assuming that the initial state is the N qubit all-zero bit string $|0\rangle^{\otimes N}$. The transition amplitude for any arbitrary output bitstring x can then be written as:

$$A_x = \langle x|U_d U_{d-1} \cdots U_2 U_1 |0^{\otimes N}\rangle \tag{2.20}$$

where the circuit has depth d and the unitary U_i is the unitary that describes the linear algebra operation for the i^{th} layer of the circuit. Figure 2.11 depicts a tensor network representation of the transition amplitude of the circuit.

They then use the parameterization developed by Gorman [49] and try to find a contraction order that minimzes edge and vertex congestion over the possible tree embeddings of the

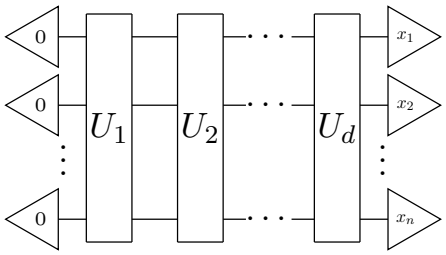


Figure 2.11: Tensor network representing transition amplitude A_x of a d-depth circuit with respect to an output bitstring x encoded in computational basis.

network. Although they don't explicitly use the congestion measures while performing optimization, they use quantities proxy to them and employ heuristics based on graph theoretic properties of the network. However, before performing the optimization they use a series of simplifications based on the tensor network structure and sparsity of tensors. They perform such simplifications iteratively until no more operations are possible. These simplifications decrease the complexity of the tensor network priod to invoking full contraction path finders and can be performed efficiently as a series of local searches.

Thus the process for computing A_x takes the following steps (see Figure 2.12);

(a) Create the tensor network representation of the circuit

(b) Perform structural simplifications of the tensor network

(c) Find the contraction path for this simplified network

(d) Perform the contraction using the found path

Step (a) is very cheap and the path found in step (c) could be reused for computing the probability amplitude for different bitstrings. To find a contraction path of the simplified network, they explicitly construct the contraction tree for the tensor network. The generation of these trees are driven by various state of the art graph manipulation algorithms combining agglomerative, divisive and optimal strategies. These algorithm are then hyper-optimized with respect to the total contraction cost such that both the method applied and its parameters are varied throughout the contraction path search. The framework has been implemented by the

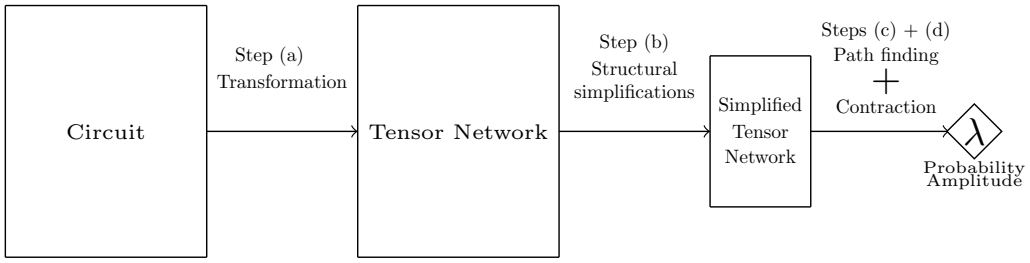


Figure 2.12: Pipeline for strong simulation of quantum circuits

authors using a python library called `quimb`. `quimb` creates the tensor network representation of the circuit and applies local simplifications after which it conducts search over contraction order using another python library `cotengra`. We will be using `quimb` both as an evalutation framework and as a baseline throughout this dissertation.

In this dissertation we focus mainly on step (b) of the pipeline, i.e. the structural simplifications applied to the tensor network, and consider the rest as a black-box. In the paper [45] Gray and Kourtis come up with some heuristical simplification inspired from the properties of quantum circuits and notice that local, cheap to apply structural simplifications are able to make considerable impact on both the size of the tensor network and the associated contraction costs. Their heuristics often utilize some known properties of quantum circuits cast as tensor networks to create a more efficient representation with respect to contraction order search. We aim to enhance such simplifications by using quantum graphical languages which could potentially exploit the inherent structure of quantum circuits better using their rich graphical representation. In the subsequent chapter, we provide a description of the quantum graphical languages examined in this dissertation.

Chapter 3

Existing work on Quantum graphical languages

The most widely used formalization of quantum computing is the quantum circuit model, a diagrammatical language for representing unitary matrices in two dimensional Hilbert space. The quantum circuit model is used to describe quantum algorithms, quantifying the resources they use in their experimental implementations(for e.g. by counting the number of qubits required or the depth of the circuit) and to classify the entangling properties and expressive power of specific gate families. While preserving the structure of actual hardware implementation is helpful, the rigid structure and lack of diagrammatic transforms imposes restrictions on reasoning about quantum systems.

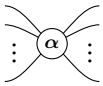
Quantum graphical languages have been developed to allow a more flexible formalism for reasoning about quantum systems. These languages relax the unitarity condition and allow all linear maps to be represented by their generators and associated relations. The ZX-calculus is one such graphical language, that was introduced in [32, 33] as a part of the categorical quantum mechanics program. It relies on the interaction between two complementary observables and is able to capture many important concepts in quantum

mechanics intuitively. Another graphical language, called the ZH-calculus was later introduced to include compact encodings of non-linear classical functions and hypergraph-states [34]. Quantum languages like the ZX- and ZH-calculus share an important advantage over the quantum circuit model. Processes and matrices are not represented merely by diagrams, but by *graphs* (hence the term *graphical language*). This allows for a more modular representation of quantum circuits.

In this chapter, we will describe the generators and fundamental rewrite rules associated with ZX- and ZH-calculus. The initial sections serve as a summary of the excellent introduction to ZX- and ZH- calculus given by John Van de Wetering in "ZX-calculus for the working quantum computer scientist" [50]. We then describe more advanced graph-based simplification procedures developed for the calculi and conclude by briefly discussing the implementation of these procedures using the `PyZX` library [51], an open source library programmed in python for creating, visualizing and rewriting ZX-diagrams.

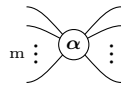
3.1 ZX-calculus

In the following section we give a short introduction to ZX-calculus. A more in-depth coverage of the topic can be found in [50] and [52] . The fundamental building blocks of ZX-diagrams are spiders and wires. A spider is a restriction on the type of tensors that this graphical language supports. The first kind of spider is called a Z spider which is represented by a white (also represented as green) dot with arbitrary number of legs attached. A Z-spider is defined with respect to the eigenbasis of Pauli Z.

Z-spider: 

$$:= \quad |0\cdots0\rangle\langle 0\cdots 0| + e^{i\alpha}\,|1\cdots1\rangle\langle 1\cdots 1|$$

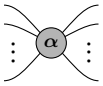
The variable α denotes an angle as a fraction modulo 2π and the "$\cdots$" notation is translated as 0 or arbitrary many wires depending on the number of incoming (wires coming in from the left) and outgoing wires (wires coming out of the right). In the standard linear map representation, a Z spider is represented by a matrix of the form



$$:= 2^n \overbrace{\begin{Bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & e^{i\alpha} \end{Bmatrix}}^{2^m} := \overbrace{|0\cdots0\rangle}^{n}\overbrace{\langle 0\cdots 0|}^{m} + e^{i\alpha}\overbrace{|1\cdots1\rangle}^{n}\overbrace{\langle 1\cdots 1|}^{m}$$

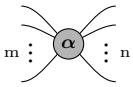
The other fundamental generator is a X-spider denoted by a grey (also represented as red) dot with arbitrary number of legs attached. A X-spider is defined with respect to the eigenbasis of Pauli X.

X-spider:



$$:= |+\cdots+\rangle\langle+\cdots+| + e^{i\alpha}\,|-\cdots-\rangle\langle-\cdots-|$$

The linear map representation of X-spider can be constructed by Z-spider's matrix representation, with a simple change of basis.



$$:= \left(\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\right)^{\otimes n} \circ 2^n \overbrace{\begin{Bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & e^{i\alpha} \end{Bmatrix}}^{2^m} \circ \left(\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\right)^{\otimes m}$$

$$:= \overbrace{|+\cdots+\rangle}^{n}\overbrace{\langle+\cdots+|}^{m} + e^{i\alpha}\overbrace{|-\cdots-\rangle}^{n}\overbrace{\langle-\cdots-|}^{m}$$

From a tensor network viewpoint, we see that Z and X spiders are just tensor networks with some additional constraints. Each wire corresponds to a 2-dimensional index and the tensors are defined as follows:

$$\left(\ \textcircled{\alpha}\ \right)^{j_1 \ldots j_n}_{i_1 \ldots i_m} = \begin{cases} 1 & \text{if } i_1 = \ldots = i_m = j_1 = \ldots = j_n = 0 \\ e^{i\alpha} & \text{if } i_1 = \ldots = i_m = j_1 = \ldots = j_n = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\left(\ \textcircled{\alpha}\ \right)^{j_1 \ldots j_n}_{i_1 \ldots i_m} = \frac{1}{\sqrt{2}} \cdot \begin{cases} 1 + e^{i\alpha} & \text{if } \bigoplus_k i_k \oplus \bigoplus_l j_l = 0 \\ 1 - e^{i\alpha} & \text{if } \bigoplus_k i_k \oplus \bigoplus_l j_l = 1 \end{cases}$$

where i_k, j_l range over $\{0, 1\}$ and $\oplus$ is addition modulo 2 (i.e. XOR).

Some common ZX-diagrams and their linear map representations are given below.

A 1-input, 1-output Z-spider with a phase α corresponds to the $R_Z(\alpha)$ phase gate (ref. 3) as shown below:

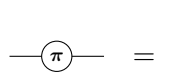
$$\text{—}\textcircled{\alpha}\text{—} \quad = \quad |0\rangle\langle 0| + e^{i\alpha}\,|1\rangle\langle 1| \quad = \quad \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & e^{i\alpha} \end{bmatrix} \quad = \quad \begin{bmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{bmatrix} \tag{3.1}$$

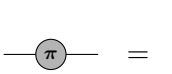
Similarly a 1-input, 1-output X-spider with a phase α corresponds to the $R_X(\alpha)$ phase gate (ref. 3) as shown below:

$$\text{—}\textcircled{\alpha}\text{—} \quad = \quad |+\rangle\langle +| + e^{i\alpha}\,|-\rangle\langle -| \quad = \quad \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \frac{1}{2}e^{i\alpha}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad = \quad \frac{1}{2}\begin{bmatrix} 1 + e^{i\alpha} & 1 - e^{i\alpha} \\ 1 - e^{i\alpha} & 1 + e^{i\alpha} \end{bmatrix} \tag{3.2}$$

For both the above diagrams, we can choose the phase $\alpha = \pi$ which gives us the familiar Pauli

Z and Pauli X matrices respectively (ref. 1).


$$= |0\rangle\langle 0| - |1\rangle\langle 1| = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \text{Pauli Z} \tag{3.3}$$


$$= |+\rangle\langle +| - |-\rangle\langle -| = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \text{Pauli X} \tag{3.4}$$

We can also represent the Pauli Z and Pauli X basis states (upto a scalar) using spiders:

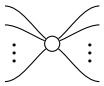

$$= |+\rangle + |-\rangle = \sqrt{2}\,|0\rangle \qquad \text{(3.5)}$$

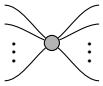

$$= |0\rangle + |1\rangle = \sqrt{2}\,|+\rangle \qquad \text{(3.5)}$$


$$= |+\rangle - |-\rangle = \sqrt{2}\,|1\rangle \qquad \text{(3.6)}$$

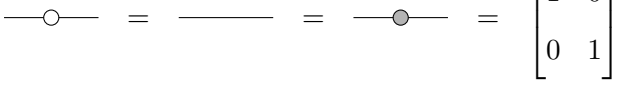

$$= |0\rangle - |1\rangle = \sqrt{2}\,|-\rangle \qquad \text{(3.6)}$$

Spiders without any labelled phases are assumed to have a phase $\alpha = 0$.


$$= |0\cdots 0\rangle\langle 0\cdots 0| + |1\cdots 1\rangle\langle 1\cdots 1|$$


$$= |+\cdots +\rangle\langle +\cdots +| + |-\cdots -\rangle\langle -\cdots -|$$

which implies that an identity map in ZX is represented as follows:

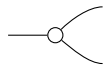

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.7}$$

ZX-diagrams have compositions similar to tensor networks where each wire connecting two tensors corresponds to a contraction. We defined the following two compositions for ZX-diagrams D1, D2:

Spatial composition D1 $\otimes$ D2: Putting the diagrams on top of each other corresponds

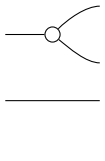
to taking the kronecker product of the respective linear maps

As an example let us consider the following diagram:



$$= \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.8}$$

Suppose we wish to spatially compose this diagram with the identity wire. The way we do this is by calculating the kronecker product:

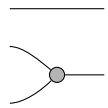


$$= \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Sequential composition D1 · D2: Placing D1 side by side D2 and connecting the outputs of D1 to the inputs of D2. This is only possible if the number of outputs of D1 is equal to the number of inputs of D2 and corresponds to matrix multiplication.

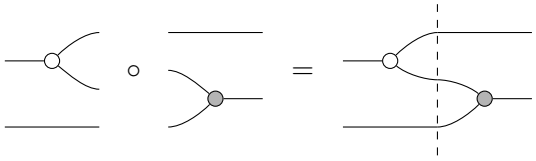
To see this in action, let us first consider a similar diagram to 3.8, but using the X

spider, and its spatial composition with identity:



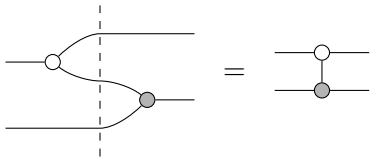
$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Now, if we want to sequentially compose this diagram with the previously constructed diagram, we can do this by horizontally stacking the diagrams:



(3.9)

ZX-diagrams are invariant under spatial isotopy. This means that if one diagram can be deformed into another, then they are equal. By straightening the wires in the above diagram, we get a more canonical representaiton of the CNOT gate which resembles the CNOT gate in conventional circuit notation.



(3.10)

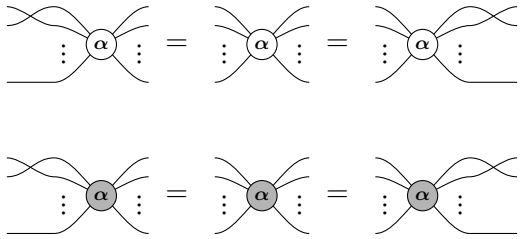
Considering the linear map representation, horizontal composition of diagrams

corresponds to matrix multiplication:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1&0&0&1&0&0&0&0\\0&1&1&0&0&0&0&0\\0&0&0&0&1&0&0&1\\0&0&0&0&0&1&1&0\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$$

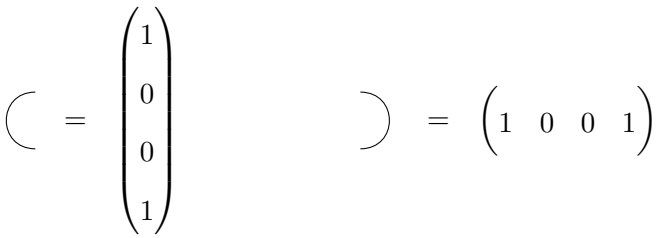
which is indeed the matrix of a CNOT gate upto a scalar factor (ref. 2.14).

ZX-diagrams also posess some useful symmetries. For instance, the spiders are symmetric tensors which means that they are invariant under swapping of wires:



(3.11)

The calculus also admits the familiar cup and cap tensors

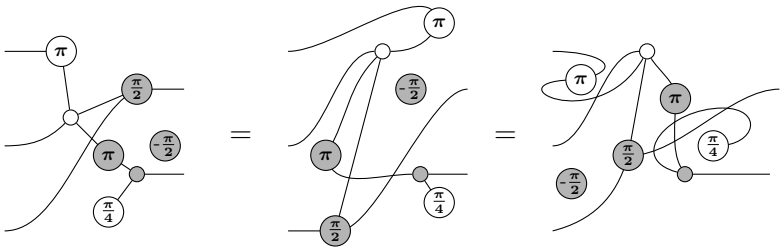


(3.12)

which give rise to the *yanking equations* of the calculus:

(3.13)

This enables a fundamental property in ZX-diagrams, often termed as as *only connectivity matters*, which means it is neither important in which order the spiders are arranged nor how the wires are formed as long as the order of inputs and outputs of a diagram is preserved. The following ZX-diagrams illustrate this property:



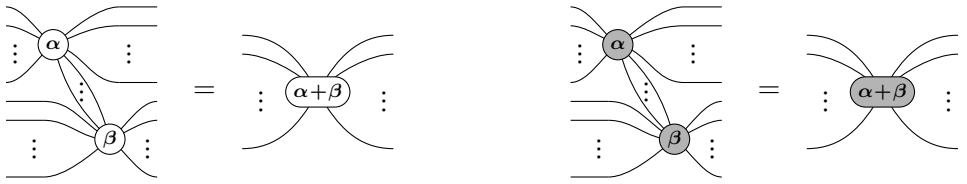
(3.14)

Because only connectivity matters, all the three diagrams in the above equation represent the same linear map. In general we can decompose every unitary operation on multiple qubits into a combination of CNOTs and single qubit unitary operations. Since we are able to represent CNOT and arbitrary single qubit phase gates with Z and X spiders, Z and X spiders are the universal building blocks for quantum computation and all quantum operations can be constructed using these two spiders.

Although ZX-diagrams are just tensor networks, the additional contraints on the type of tensors and the wires connecting them give rise to associated *rewrite rules* that allow transformations on ZX-diagrams. The rules are sound, which means if two diagrams can be transformed into each other, then their linear map representation is equal. Below we list a succint description of the ZX-calculus rules set:

Spider fusion The most fundamental rule in ZX-calculus deals with the fusion of spiders. This rule allows us to merge connected spiders of the same color into a single spider with the phases summing up (module 2π). Taken the other way, this rule also enables splitting a spider into any nunmber of spiders of the same color with the phases summing up to the

original phase.



(sf)

Identity rule An empty spider with two connected wires equals an identity.

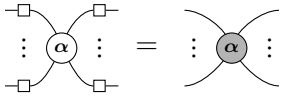
(id)

Moreover, two hadamards in a row cancel each other

(hh)

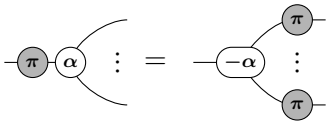
These rules are also familiar in conventional quantum computing where two hadamards cancel each other and a rotation around an axis by zero has no effect on the corresponding qubit.

Hadamard rule The hadamard rule allows us to swap colors of any spider by adding hadamards on all the input and output legs of the spider. This essentially amounts to changing from one basis to the other and was used in the definition of X spider in the begining of this section.



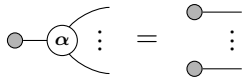
(h)

π **rule** π rule, as the name suggests deals with π phase gates and their interaction with spiders. In general, a π phase gate copies through a spider of opposite color and negates the phase of the spider.



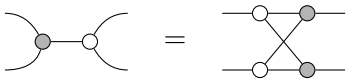
(π)

copy rule The copy rule is similar to π rule but ends up killing the original spider



(c)

Bi-algebra rule The last major rule of ZX-calculus is the bialgebra rule which deals with the interaction of opposite color spiders.

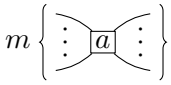


(ba)

3.2 ZH-calculus

ZH-calculus is a related graphical calculus to ZX, but was developed with the motivation of natively representing hypergraph states [34]. The fundamental generators of ZH-calculus are Z spiders and H boxes.

An H-box is defined as follows



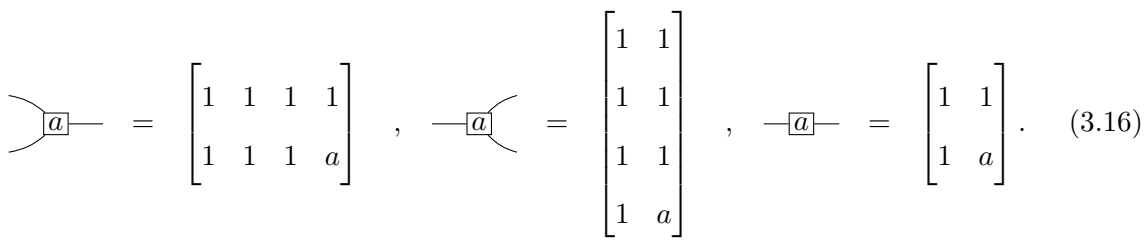
$$:= \quad \sum a^{i_1 \ldots i_m j_1 \ldots j_n} \left| j_1 \ldots j_n \right\rangle \left\langle i_1 \ldots i_m \right| \tag{3.15}$$

The sum in this equation is over all $i_1, \ldots, i_m, j_1, \ldots, j_n \in \{0, 1\}$ and a is an arbitrary complex number. An H-box in its tensor network representation is denoted as follows

$$\left(\boxed{a} \right)_{i_1 \ldots i_m}^{j_1 \ldots j_n} = \begin{cases} e^{i\alpha} & \text{if } i_1 = \ldots = i_m = j_1 = \ldots = j_n = 1 \\ 1 & \text{otherwise} \end{cases}$$

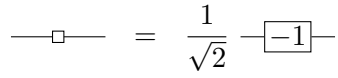
Hence, an H-box represents a matrix with all entries equal to 1, except the bottom right element, which is equal to a. Some examples of H-boxes with their linear map representation

are as follows:



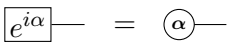
$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & a \end{bmatrix} \quad , \quad = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & a \end{bmatrix} \quad , \quad = \begin{bmatrix} 1 & 1 \\ 1 & a \end{bmatrix}. \tag{3.16}$$

We see that when $a = -1$, the 1-input 1-output H-box becomes a Hadamard upto some scalar factor.



$$= \frac{1}{\sqrt{2}} \tag{3.17}$$

We will follow the convention of unlabelled H-boxes are assumed to have label -1. Thus, the H-boxes serve as a generalization of hadamards for arbitrary number of input and output wires. Now that we have the basic definitions in place, we can move on to stating the rewrite rules of this calculus. First, note the following relation between an H-box and Z spider.



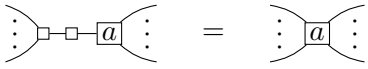
(3.18)

Taking $\alpha = 0$ and $\alpha = \pi$, we get the following useful rules

(3.19)

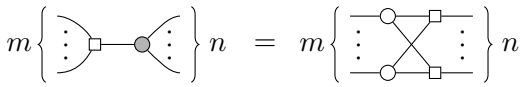
We now present the phase-free rewrite rules of ZH calculus.

H-box fusion rule We have seen in the previous section that two hadamards next to each other cancel. This further extends to H-boxes leading to the H-box fusion rule stated as follows:



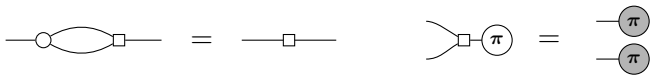
(zhf)

ZH-Bialgebra rule The bialgebra rule in ZX-calculus has a following presentation in ZH:



(zhba)

Additional phase-free rules The final rules complete the phase-free rules for ZH-calculus:



(zhpf)

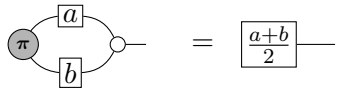
It is interesting to note that these rules are complete for Hadamard-Toffoli fragment of ZH-calculus [53]. This implied that any two circuits only consisting of Hadamards and Toffolis can be reasoned about completely using the stated rules. As Hadamard+Toffoli circuits are approximately universal [54,55], we can use the rules stated above to do arbitrary calculations for a approximately universal model of quantum computation.

The second set of rules for ZH-calculus deal with labelled H-boxes and their phases.

Multiply rule The multiply rule enables fusing of phases when a set of H-boxes are connected to the same set of Z-spiders. The multiply rule and its generalization to arbitrary arity H-boxes are stated as follows:

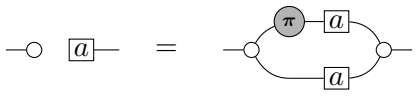
(zhm)

Average rule The average rule is another rule which deals with the phases of H-boxes. It is stated as follows:



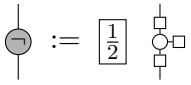
(zha)

Introduction rule The introduction allows the introduction of additional edges to an H-box at the cost of copying the H-box.

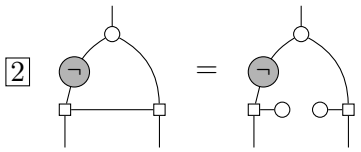


(zhi)

Ortho rule The final rule for ZH-calculus is called the ortho rule. It is stated with the help of a NOT spider defined as:



The rule is stated as follows:



(zho)

3.3 Graph-based simplifications

As we have seen in the previous sections, ZX-/ZH-calculi come equipped with a set of rewrite rules that help reason about quantum circuits and their properties using a lower-level graphical language. In addition to the rewrite rules, recent works have found developing graph-based simplification procedures on top of the rewrite rules to be very effective at optimising quantum circuits [35, 36]. One can use such procedures to create an equivalent tensor network representation of a quantum computation task but with fewer tensors (spiders) and fewer indices (legs). Thus graphical calculus could enable us to produce a compressed tensor network representation by applying cheap transformations to the graphical structure.

In this chapter, we review some of the existing graph-based transformations and the resulting simplification procedures in the literature which have been applied to simplify ZX-/ZH-diagrams.

3.3.1 Graph-based ZX-simplifications

This section provides a condensed overview of the ideas presented in [35, 36]. The core of most ZX-calculus based simplification strategies are two rules named *local complementation* and *pivoting*. These rules are rooted in graph theory and apply when the diagrams are in a special normal form called *graph-like*. We begin this section by presenting this normal form's definition and then describe the graph-theoretic rewrite rules along with their ZX counterparts.

Definition 3.3.1. (Graph-like diagrams) We call a diagram to be *graph-like* when

- Every spider is a Z-spider
- Every spider is connected only via Hadamard edges
- There are no parallel edges or self-loops.
- Every input and output is connected to a spider and every spider is connected to at most one input or output.

Every ZX-diagram can be efficiently converted to a graph-like diagrams using the rules from section 3.1. A special instance of graph-like diagrams called *graph-states* are those that are in this normal form but with additional constraints.

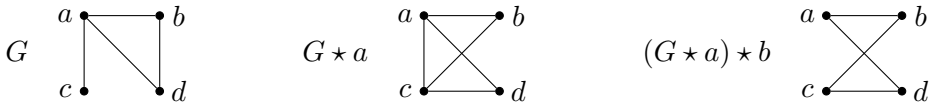
Definition 3.3.2. (Graph-states) A graph-like diagram is called a graph-state if

- All spiders must be phaseless.
- There are no inputs
- Every spider is connected to a unique output

We can see that the spider unfusion rule can convert any subgraph of graph-like diagram to a graph state. Graph states have a nice property that two graph states are equal up to a unitary clifford (ref. 2.2.1) matrix if and only if the underlying graphs can be transformed

into one another using a series of graphical transformation steps called *local complementation*. Thus, we can create an equivalent linear map representaiton of graph states using local-complementation and some phase manipulation according to the unitary clifford matrix. We will now describe the transformations briefly.

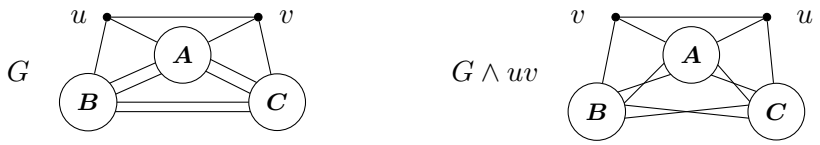
Local complementation Let G be the underlying graph and u be a vertex in G. The local complementation of G about u, $G \star u$ is defined as a transformation where the vertices remain the same but the edges in the neighbourhood of u are toggled. An edge between the neighbours of u is added if it does not exist and removed if it does.



Pivoting Pivoting in another transformation for the graph which occurs about an edge $(u, v) \in E$ and is defined as three successive local complementation steps on alternating vertices u and v i.e. $G \wedge uv = G \star u \star v \star u = G \star v \star u \star v$. Pivoting can be understood by separating the rest of the vertices into three sets:

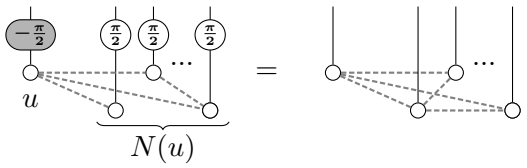
- $N_G(u) \cap N_G(v)$ i.e. one where all vertices are connected to both u and v,
- $N_G(u) \backslash (N_G(v) \cap \{v\})$ i.e. one where all vertices are connected to only u and
- $N_G(v) \backslash (N_G(u) \cap \{u\})$ i.e. one where all vertices are connected to only v.

In a pivoted graph $G \wedge uv$, all vertices between those sets are connected which were not connected in G. Any connections within a set is not modified.



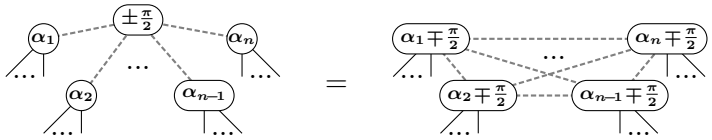
Local complementation and Pivoting in ZX

Local complementation and pivoting enable us to apply some very interesting transformations to ZX-diagrams. Consider a ZX-diagram D which is a graph state. We can transform D into D' such that $G(D') = G(D) \star u$ by applying $X_{-\pi/2}$ on the spider corresponding to u and $Z_{\pi/2}$ on spiders in the neighbourhood of u:



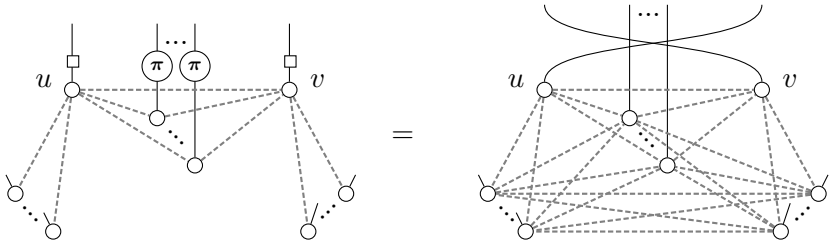
(3.20)

The derivation of this tranformation was given in [56]. We can apply this rule to any spider in a graph-like diagram since we can always pull the phases out of the spiders via the unfusion rule. Moreover, for spiders with phase $\pm\pi/2$ we can apply the rule to remove the spider u completely:



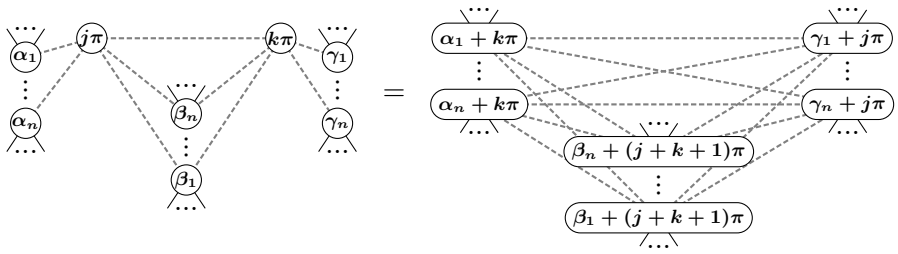
(3.21)

Similarly, a pivot $G(D') = G(D) \wedge uv$ can be introduced by applying Hadamard gates on u and v and Z_π gates on $N_{(u)} \cap N_{(v)}$ [57] :



(3.22)

In case of pivoting, we can remove two adjacent spiders with a phase of 0 or π.



(3.23)

Clifford simplification algorithm

We have seen the Clifford group and Clifford gates in section 2.2.2 (ref. 2.2.1). An important property about spiders is that the spider types representing the Clifford subset of quantum gates are exactly all spiders with phase $k\pi/2$ for an integer k. Thus we see that these two transformation relating to the graph theoretic notions of *local complementation* and *pivoting* can be used to elimiate Clifford spiders from our ZX-diagrams in turn simplifying them. The following simplification algorithm can be used for eliminating Clifford spiders in ZX-diagrams:

1. Tranform the diagram into a graph-like diagram.

2. Apply identity and fusion rules to maintain graph-like diagram.

3. Apply local-complementation on every spider of phase $\pm\pi/2$ and pivoting on every pairs of spider having 0 or π phase as long as possible.

4. Terminate if step 3 didn't modify the diagram else go to step 2.

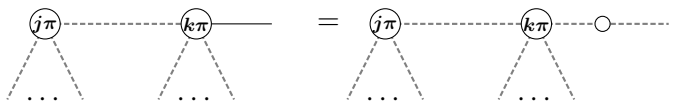
This simplification procedure leads to a ZX-diagram that contains no interior spiders with phase $\pm\pi/2$ and the only interior Clifford spiders left are spiders with phase 0 or π which are either adjacent to boundary or to non-Clifford spiders.

Additional simplifications

Although these algorithms already simplify the diagrams drastically, there are some more strategies that we can apply in addition to the procedure laid above. We now discuss *pivot boundary simplification*, which can help further reduce interior Clifford spiders with phase of 0 or π, and *phase gadget simplification*, which aims to reduce non-Clifford spiders.

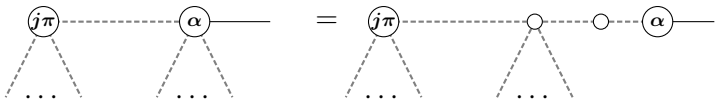
Pivot boundary simplification We described how pivoting can be applied on two connected interior spiders with a phase equal to 0 or π. We can also use pivoting to remove an internal Pauli spider that is adjacent to a boundary spider with arbitrary phase. We first use identity rules to replace the non-Hadamard wire with an empty spider surrounded by

two hadamard wires:



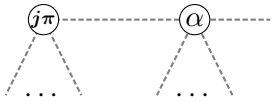
(3.24)

This transforms the original boundary spider into an interior spider where a pivot rule application is possible.



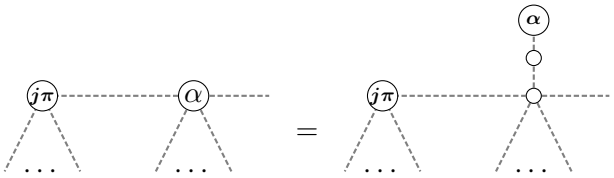
(3.25)

Phase Gadget simplification If we have a spider with $0/\pi$ phase conntected only to non-clifford spiders, like as shown here:



(3.26)

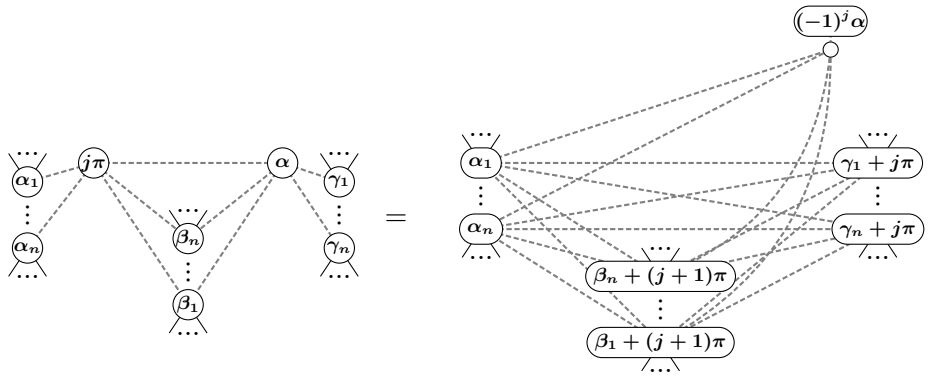
we can use phase gadget simplifications combined with pivoting to simplify the diagram. First we modify one of these spiders as follows to make the originally non-clifford spider, a spider with phase 0 connected to a phase gadget with the original phase α:



(3.27)

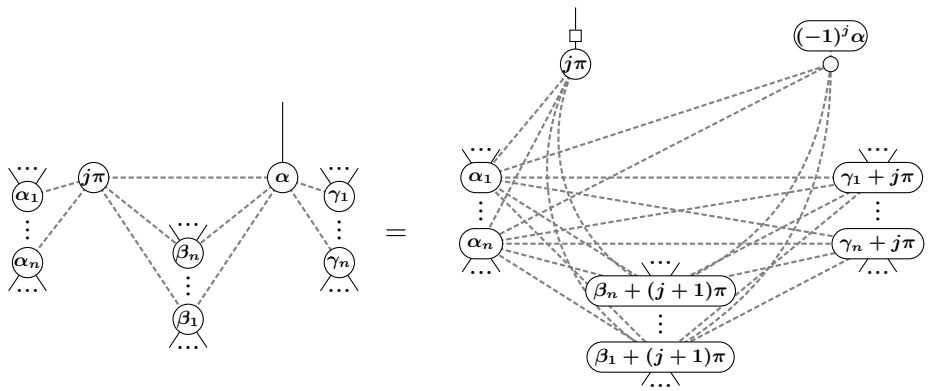
Now we can perform pivoting/pivot boundary simplification based on the internal spider's location and simplify our diagram. For the case when the non-Clifford spider is internal, we

can apply the following rule:



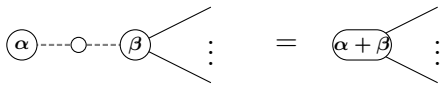
(3.28)

and correspondingly when the non-Clifford spider is a boundary spider, we apply the following transformation:



(3.29)

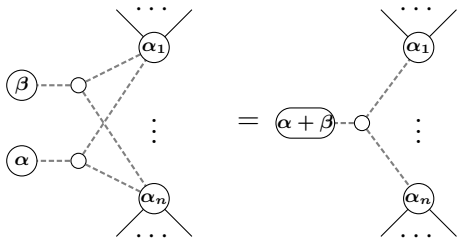
Morevoer, phase gadgets have some additional simplifications associated with them. When a phase gadget is connected to exactly one other spider, its phase can be combined with the phase on that spider via (ID). This is essentially an application of the rules (i1) and (i2).



(3.30)

When two phase gadgets are connected to exactly the same set of spiders, they can be fused

into one via the gadget-fusion rule (GF).



(3.31)

3.3.2 Graph-based ZH-simplifications

In this section, we will summarize the notation and the simplification procedures for ZH-diagrams laid out in [58]. We begin by describing a new class of ZH-diagrams that generalizes the definition of graph-like ZX-diagrams from [36].

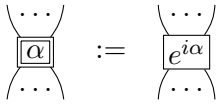
Definition 3.3.3. (Hypergraph-like diagrams) We call a diagram to be *hypergraph-like* when

- Every spider is a Z-spider
- Every input and output is connected to a spider and H-boxes don't have any direct connection to inputs/outputs.
- Every wire except the input and output wire connects a z spider and a H-box.
- There are no parallel edges between H-box and Z-spider.
- No H-boxes are connected to the same set of Z-spiders.

The underlying hypergraph of a hypergraph-like ZH-diagram is a simple graph iff all H-boxes have arity 2. Such diagrams are graph-like as defined in [36]. Every ZH-diagram can be efficiently reduced to a hypergraph-like ZH-diagram representing the same linear map as shown in [58].

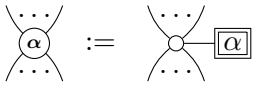
Hypergraph-like diagrams are often represented with the help of *exponentiated H-boxes* which

are represented as double bordered boxes annoted with the phase of the H-box:



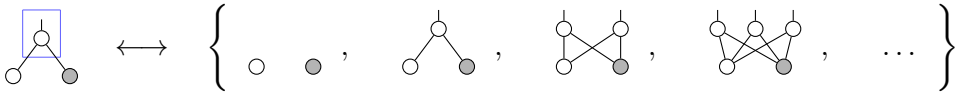
(3.32)

We can use the exponentiated H-boxes to pull phases out of a Z-spider as follows:



(3.33)

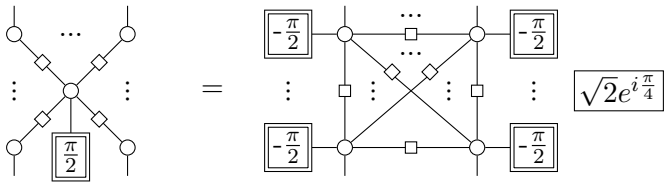
We also introduce the notation of a !-box. A !-box, drawn as a blue square around a piece of a ZH-diagram, represents a part of the diagram that may be replicated an arbitrary number of times, and hence allows one to express a whole family of diagrams at once:



Hyper-local complementation

As shown in the previous section, by using equation 3.21 a Z-spider labelled by a phase of $\pm\frac{\pi}{2}$ can be deleted from a ZX-diagram without changing the linear map, by performing a local complementation about the vertex.

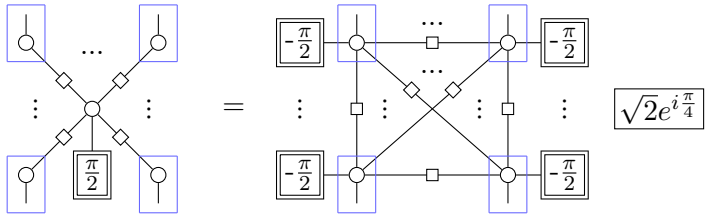
Translating the rule to a ZH-friendly notation we obtain:



(3.34)

As this rule is already proven in ZX-calculus, by completeness, it is also valid in ZH-calculus. We extend this rule to *hyper-local complementation* by using a bang-box on each of the Z-

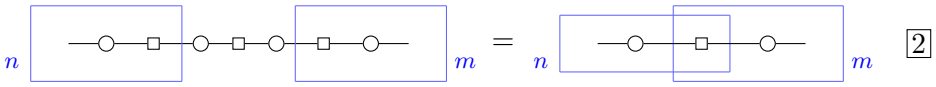
spiders at the boundary:



(3.35)

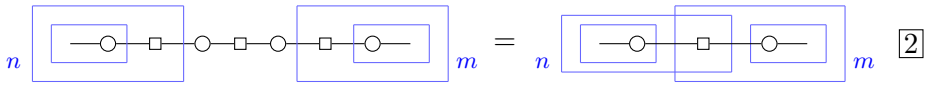
Fourier Hyper pivot

As we saw in the previous section, local complementation complements the connectivity of the neighbours of a vertex. On the other hand, a pivot along an edge complements between three groups of vertices: those connected to both the neighbors and those connected to one but not the other. The following pivoting rule holds in the ZH-calculus for any n,m $\in \mathbb{N}$.



(3.36)

We can see that this corresponds to a pivot followed by vertex deletions as every vertex (spider) connected to the pivoted vertex on the left becomes connected via a 2-ary H-box to every neighbour of the right pivoted vertex. We also present a slight generalization of the above rule by allowing each H-box to have arbitrary arity.

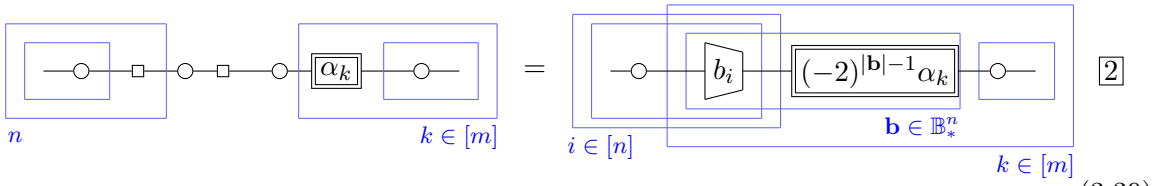


(3.37)

Similar to how a pivot is implemented by a combination of three local complementations, the hyperpivot rule can be implemented by three applications of hyperlocal complementations. We can go even further and allow arbitrary phase on the H-boxes. But before that, we construct *disconnect boxes* from [59].

(3.38)

The following equation holds for any set of real numbers $\alpha_1, \alpha_2, \cdots \alpha_m$



(3.39)

We can also use the RHP rule to create a more compact ruleset for ZH-calculus as rules (ba) (zhba) and (zhf) can be proved using Fourier hyperpivot with (sf) and (hh).

3.4 PyZX Implementation

This section gives an overview of the simplification strategies present in the `PyZX` library [51]. `PyZX` is a Python library designed to enable graphical calculus based simplifications to quantum circuits. At its core, `PyZX` implements two basic classes: *Circuits* for representing quantum circuits and *graphs* for representing ZX-/ZH-diagrams.

Simplification procedures

There are a number of simplification procedures implemented in `PyZX` using the rewrite rules and simplification algorithms we have discussed so far. Each procedure works by first calculating all suitable non-intersecting sets of spiders where a rule application is possible, called matches and then applying the rule on the found matches by using a rule application method. For example, the procedure `lcomp_simp` consists of the method `match_lcomp` which finds all spiders with phase $\pm\pi/2$ and the method `lcomp` (ref. 3.21) which receives a list of the spiders and applies the local complementation rule on all of them. All the available simplification procedures can be grouped into the following terminating algorithms:

- `interior_clifford_simp`: This function implements the algorithm laid out in 3.3.1. It begins by converting the given diagram into a graph-like diagram, then applies `id_simp` (ref. id), `spider_simp` (ref. sf), `pivot_simp` (ref. 3.23), `lcomp_simp` (ref. 3.21) in a loop until no more rule application is possible.

- `clifford_simp`: This function adds pivot boundary simplification (ref. 3.25), implemented as a phase gadget, to `interior_clifford_simp`. Each subroutine is applied in a loop until no more changes occur.

- `zx_simp (full_reduce)`: This algorithm additionally applies phase gadget simplifications (ref. 3.28,3.29) to `clifford_simp` and is the most powerful simplification algorithm in `PyZX` for ZX-diagrams. This function is implemented as `full_reduce` in `PyZX` but we will refer to it as `zx_simp` in this dissertation for clarity.

- `zh_simp`: This function combines ZH-simplification routines with `zx_simp` but with a change in the order of application of rules. It applies simpler rules like `spider_simp`, `hspider_simp` (ref. zhf), `id_simp`, `par_hbox_simp` (ref. zhm) before applying more complicated routines like `pivot_simp` (ref. 3.23) and `lcomp_simp` (ref. 3.21). Lastly, the function applies `hpivot_simp` (ref. 3.37) along with `par_hbox_simp` and moves to the next iteration. All the rules are then again applied in a loop until no more application is possible.

Chapter 4

Enhancing Quantum Circuit Simulation

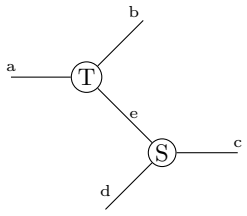
This chapter provides a description of our proposed strategies for quantum circuit simulation by enhancing tensor network contractions. We follow the approach used by Gray and Kourtis [45] where they use tensor network contractions to perform *strong simulation*. We try to improve the contraction procedure using graphical-calculi-based simplification procedures introduced in earlier chapters. Our first step is to describe our chosen representation of tensor networks, making it possible to efficiently convert between ZX/ZH diagrams and a more general representation of tensor networks. Then we outline the modified circuit simulation pipeline to enable the use of graphical-calculi-based simplifications. We present the results of using existing graphical-calculi-based simplification procedures and compare that against the original results. We restrict our analysis in this chapter to Sycamore supremacy circuits and deal with more circuit classes in the following chapter. To motivate better simplification procedures, we discuss the issues encountered while applying already existing strategies in the pipeline. We then present our framework to circumvent these issues by developing a heuristic measure to help guide the application of simplification subroutines to our diagrams. This chapter concludes with a discussion of the

results obtained after evaluating the proposed strategies.

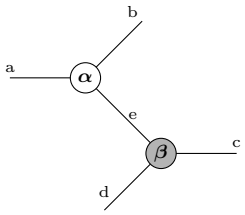
4.1 Hypergraph-like Tensor Networks

We will now describe how we can represent a ZX/ZH diagram as a tensor network. To begin, let's start with an example of a two tensor contraction. Consider a rank-3 tensor T_{abe} contracting with another rank-3 tensor S_{cde} to give a rank-4 tensor U_{abcd}. The tensor network representation of this contraction is as follows:

$$U_{abcd} = \sum_{e} T_{abe} S_{cde} = \tag{4.1}$$



Now, let's consider a similar ZX diagram with a Z-spider inplace of the T tensor and an X-spider inplace of the S tensor.



(4.2)

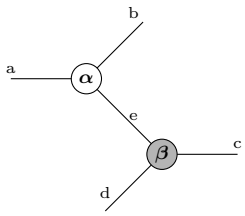
We can treat indices a, b as the input to the Z spider and index e as its output. Similarly, indices c, d can be considered as output for the X spider with index e being the input. To represent this diagram as a tensor network contraction, we begin by recalling the tensor network representation of generators Z and X.

$$Z^{j_1 \ldots j_n}_{i_1 \ldots i_m}(\alpha) = (\ \textcircled{\alpha}\ )^{j_1 \ldots j_n}_{i_1 \ldots i_m} = \begin{cases} 1 & \text{if } i_1 = \ldots = i_m = j_1 = \ldots = j_n = 0 \\ e^{i\alpha} & \text{if } i_1 = \ldots = i_m = j_1 = \ldots = j_n = 1 \\ 0 & \text{otherwise} \end{cases}$$

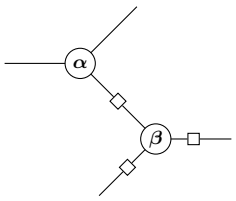
$$X^{j_1\ldots j_n}_{i_1\ldots i_m}(\alpha) = (\ \boxed{\alpha}\ )^{j_1\ldots j_n}_{i_1\ldots i_m} = \frac{1}{\sqrt{2}} \cdot \begin{cases} 1 + e^{i\alpha} & \text{if } \bigoplus_k i_k \oplus \bigoplus_l j_l = 0 \\ 1 - e^{i\alpha} & \text{if } \bigoplus_k i_k \oplus \bigoplus_l j_l = 1 \end{cases}$$

Thus, the tensor network contraction represented by the ZX diagram equals:

$$U^{abcd} = \sum_e Z^{ab}_e(\alpha) X^{cd}_e(\beta) = \tag{4.3}$$

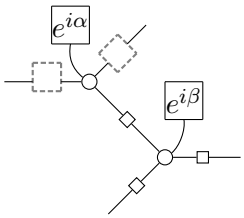


Now let's consider the tensor network representation of a ZH-diagram. As an example, we take the same diagram in (4.2) and apply the color change rule to get an equivalent ZH diagram.



(4.4)

This diagram has hadamard edges between any two internal spiders and a simple edge connecting spiders to inputs/outputs. To convert this into a tensor network we first take the phases out of the Z spiders using 1-ary phase H-boxes. Then, we let every hadamard edge stay as it is, but place a kronecker-box (represented by a dashed box) at every simple edge. We get the following diagram:



(4.5)

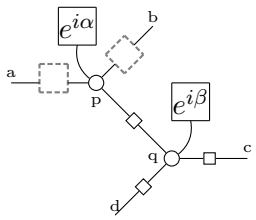
So now we have a diagram with phase-free Z spiders and H-boxes. Every edge now either

contains an H-box or a kronecker-box. The tensor network representation of H-box and kronecker box is as follows

$$H^{j_1\ldots j_n}_{i_1\ldots i_m}(e^{i\alpha}) = \Big(\boxed{e^{i\alpha}} \Big)^{j_1\ldots j_n}_{i_1\ldots i_m} = \begin{cases} e^{i\alpha} & \text{if } i_1 = \ldots = i_m = j_1 = \ldots = j_n = 1 \\ 1 & \text{otherwise} \end{cases}$$

$$\delta^{j_1\ldots j_n}_{i_1\ldots i_m} = \Big(\boxed{} \Big)^{j_1\ldots j_n}_{i_1\ldots i_m} = \begin{cases} 1 & \text{if } i_1 = \ldots = i_m = j_1 = \ldots = j_n \\ 0 & \text{otherwise} \end{cases}$$

So to convert the diagram into a tensor network, we consider each H-box/Kronecker-box as a tensor and each Z-spider as an index. We can do this because a phaseless Z spider is essentially a copy-tensor that is not zero iff all of its indices are equal. Thus, in addition to labelling the inputs and outputs, we also label the Z-spiders with indices. We can represent this as follows:

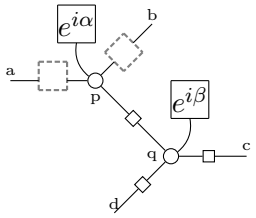


(4.6)

and the corresponding contraction is:

$$U_{abcd} = \sum_{pq} \delta^a_p \delta^b_p H_p(e^{i\alpha}) H^p_q(e^{i\pi}) H_q(e^{i\beta}) H^c_q(e^{i\pi}) H^d_q(e^{i\pi}) \tag{4.7}$$

$=$



(4.8)

We can use the above transformations to also convert graph-like and hypergraph-like diagrams into their corresponding tensor network representation. As hypergraph-like diagrams subsume

graph-like diagrams, we will present a scheme to convert hypergraph-like diagrams into tensor networks:

1. Start with a hypergraph-like diagram.
2. Pull phases out of every z-spider using 1-ary phase H-boxes.
3. Place Kronecker-boxes at every simple edge.
4. Convert every z-spider into a hyperindex and
5. Convert every H-box/Kronecker-box into a tensor.

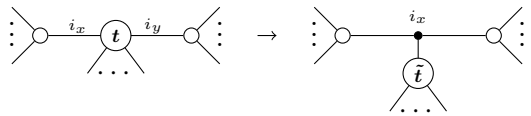
We call the graph so obtained a Hypergraph-like Tensor Network.

To further motivate the scheme, we present two of the local simplifications used in [45]. We show that there exists simplifications corresponding to the local simplifications using basic graphical-calculi-based rewrites.

Diagonal Reduction For a k-dimensional tensor, $t_{i_1,i_2\cdots i_k}$, with indices $i_1, i_2 \cdots i_k$, if for any pair$\{i_x, i_y\}$

$$t_{i_1,i_2\cdots i_k} = 0 \quad \forall \quad i_x \neq i_y \tag{4.9}$$

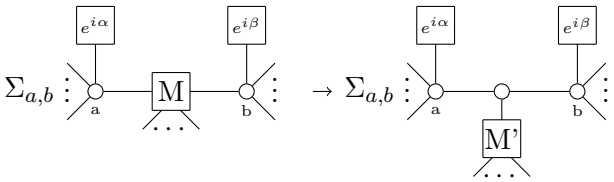
then the tensor is considered diagonal in the indices i_x and i_y. This enables us to replace the tensor t with a $k-1$ dimensional tensor $\tilde{t}$ such that $\tilde{t}_{\cdots i_x} = t_{\cdots i_x i_y}\delta^{i_x}_{i_y}$. We implement this in the tensor network by re-indexing i_y as i_x. This leads to i_x becoming a hyperedge in the tensor network.



(4.10)

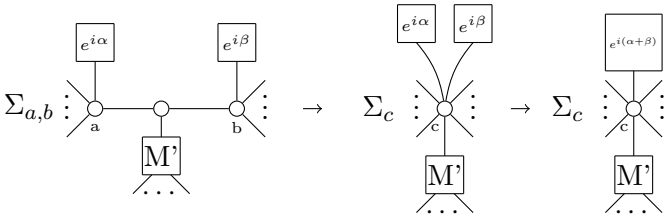
In case of hypergraph-like diagrams, the corresponding transformation can be achieved using simple rewrite rules of ZH calculus. Consider a tensor M which is diagonal in its two indices

a and b. This means that we can place a z-spider connecting the indices a, b and the now transformed tensor M'.



(4.11)

The z-spider can now fuse with indices a, b using spider fusion and the phase of the H-boxes connected to indices add up because of the multiply rule zhm.



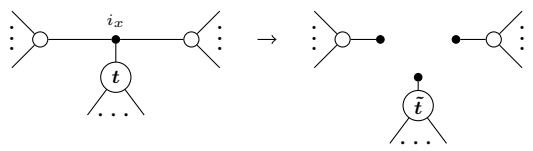
(4.12)

This essentially leaves us with the tensor M' being connected to other tensors with a hyperindex c just as in the case before.

Column Reduction For a k-dimensional tensor, $t_{i_1,i_2\cdots i_k}$, if there exists an index i_x and a column c such that

$$t_{i_1,i_2\cdots i_k} = 0 \quad \forall \quad i_x \neq c \tag{4.13}$$

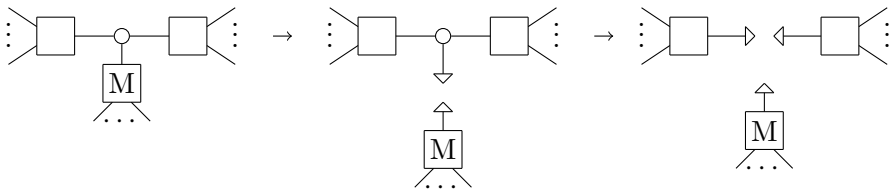
then the tensor is considered column in i_x and we can replace the tensor t with a $k-1$ dimensional tensor $\tilde{t}$ such that $\tilde{t} = t_{[i_x=c]}$. This can be implemented in the tensor network as



(4.14)

Now for a hypergraph-like tensor network, if we have a tensor M such that its column in

a hyperindex, we can sum over all possible values of the corresponding wire. This leads to creation of an effect tensor (corresponding to the value the tensor is column in), which copies through the z-spider representing the hyperindex as follows:



(4.15)

4.2 Evaluation Framework

In this section, we will present our framework of using ZX/ZH calculi and their associated simplification rules to enhance the efficiency of tensor network contractions. We will concern ourselves with the task of *strong simulation* i.e. computing the transition amplitude (ref. section 2.3.3) of quantum circuits in this dissertation. We will extend the framework developed by Gray and Kourtis [45] and use it to compute the transition amplitude of a circuit. In this chapter, we will evaluate our strategies on Sycamore supremacy circuits.

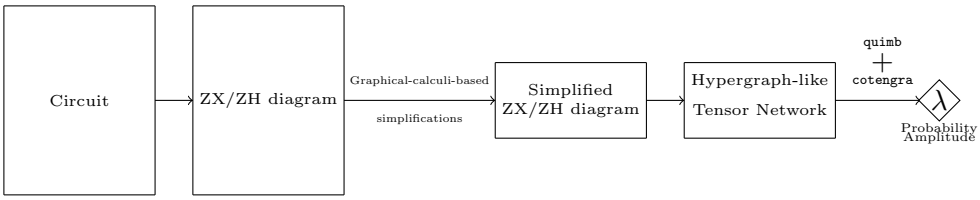
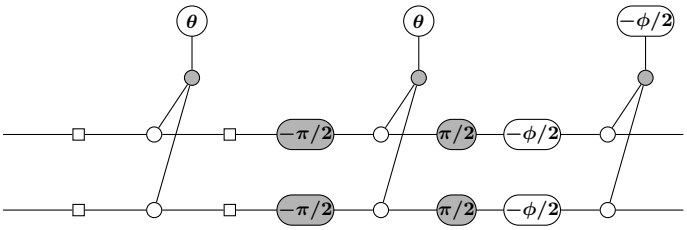


Figure 4.1: Modified pipeline for strong simulation of quantum circuits

We begin by creating a ZX diagram out of the circuit and applying the corresponding state and effect to create the transition amplitude of the circuit. The fSIM gates used in Sycamore circuits can be represented in ZX notation using phase gadgets. To this end, we first consider the decompositon of fSim gates as shown below:

$$fSim(\theta,\phi) = e^{-i\theta(X\otimes X)/2}e^{-i\theta(Y\otimes Y)/2}e^{-i\phi/4}e^{i\phi(Z\otimes I+I\otimes Z)/4}e^{-i\phi Z\otimes Z/4} \tag{4.16}$$

which leads to the following diagrammatic circuit:

$fSim(\theta,\phi) = e^{-i\phi/4}$ 

(4.17)

Moreover, similar to the experiments in [44] we will set θ to be $\pi/2$ for our experiments. This enables an iSWAP gate decomposition for the gate and was also used in [45] to simplify the circuit. All the results reported for `quimb` in this dissertation assume an iSWAP gate decomposition as well. Rest of the gates can easily be converted into the corresponding ZX-diagram following the transformations detailed in 3.1. Then we apply graphical-calculi-based simplifications to the diagram to simplify it. Finally we convert the diagram into a Hypergraph-like tensor network representation. This representation is then fed into `quimb` which applies local structural simplifications to reduce the complexity of the network. After this, the we use `cotengra` to perform a search of contraction tree for the particular tensor network which returns the found path and the corresponding time and memory requirements to perform the contraction. It uses congestion metrics of the tree embedding of contraction orders described in section 2.3.2 to estimate the time and memory requirements. We will compare different strategies using these estimates.

4.3 Basic Strategies

Table 4.1 summarizes the number of tensors and indices in the tensor network created for different sizes of Sycamore circuits. We restrict our attention to $m = 10$ cycles in this dissertation. In the table we compare the

cycles	Standard Tensor Network (#T , #I)	Hypergraph-like Tensor Network (#T , #I)
4	887 , 972	5403 , 3211
6	1251 , 1379	7947 , 4698
8	1615 , 1786	10499 , 6189
10	1979 , 2193	13035 , 7672

Table 4.1: Summary of the number of tensors (#T) and the number of indices (#I) in the Tensor Networks representing varying cycle depths of Sycamore circuits.

We see that initially, the Hypergraph-like Tensor Network representation has a lot more tensors and indices than the standard representation. This is possibly because the generators of ZX/ZH calculi are very small tensors whereas standard representation allows any gate to be represented using it's own tensor representation.

We now compare the performance of the existing strategies in `PyZX` (see. 3.4) in simplifying the tensor network against `quimb`. Table 4.2 compares the number of tensors and indices remaining in the tensor network after applying `zx_simp` and `zh_simp` to `quimb`'s contraction pipeline. We also summarize the FLOP counts of contractions for graphical-calculi-based strategies against `quimb` in Table 4.3.

cycles	`quimb` (#T , #I)	`zx_simp` (#T , #I)	`zh_simp` (#T , #I)
4	99 , 66	76 , 55	76 , 55
6	217 , 145	137 , 108	136 , 107
8	357 , 238	241 , 176	241 , 176
10	475 , 317	329 , 236	329 , 236

Table 4.2: Number of tensors (#T) and the number of indices (#I) in the Tensor Network produced after applying existing simplifications.

cycles	`quimb`	`zx_simp`	`zh_simp`
4	1.86E+04	2.42E+04	2.40E+04
6	9.78E+04	1.37E+05	1.37E+05
8	7.99E+08	8.15E+08	7.14E+08
10	1.07E+10	1.50E+10	1.54E+10

Table 4.3: Comparison of estimated FLOP counts of contraction for strategies.

We see that although the final tensor network we derive after performing `zx_simp`/`zh_simp`

are almost equivalent to the one we get after `quimb`, in terms of number of tensors and indices, there is a considerable difference in the FLOP counts for contraction. This is perhaps due to the fact that even though we started with a considerable large tensor network when starting with the ZX/ZH diagram based representation, we ended up with a tensor network almost the same size as derived using just `quimb`. This means that the tensor network simplified using `zx_simp`/`zh_simp` might be too dense to find good contraction paths.

4.3.1 Heuristic measure

To analyze how good a graph-like/hypergraph-like diagram is with respect to contraction, we would ideally want to compute contraction complexity of the underlying tensor network after each rule application in the algorithm. As this is too expensive, we propose a heuristic measure that is a proxy for the contraction complexity of the simplified tensor networks. As described, the proxy must be cheap to compute so that there is not a significant overhead for the simplification procedure but also should be accurate enough to help find good contraction paths. As we saw in section 2.3.2, vertex congestion of contraction trees serve as a good parameterization for contraction complexity. However, because congestion is defined as a global property over all possible contraction trees, it is not feasible to use every step of the simplification procedure. Thus, instead of computing the congestion over all possible vertices in the contraction tree, we will compute the average congestion of only the leaves of the contraction tree. We know that all the H-boxes in our hypergraph-like representation correspond to tensors in the tensor network which in turn corresponds to the leaves of the contraction tree. The congestion of a vertex of the contraction tree is the sum of weights of edges in the original tensor network routed through it. Because our tensor network is a hypergraph-like diagram, the weight of all edges is equal, and because we restrict our attention to the leaves of the contraction tree, the number of edges routed through an H-box is equal to the number of its next nearest neighbors. Using theorem 2.3.3 we know that the contraction complexity scales exponentially with vertex congestion and linearly with the number of tensors. To keep a similar dependence, we multiply the natural logarithm of the

number of H-boxes by the congestion of H-boxes defined above to get our proxy to contraction complexity.

4.4 Advanced strategies

Using the proposed heuristic measures, we wish to develop better simplification strategies for generating efficiently contractible tensor networks. Two recent master theses [60, 61] used classical optimization strategies to guide application of graphical simplification procedures for quantum circuit optimisation. Although they observed promising results, they found the search space to be restrictive due to the domain being the set of circuit-extractible diagrams. As we don't have the restriction of extracting circuits from our simplified diagrams, we hope to develop better optimization strategies. We now present some strategies using our heuristic measure as a guide for determining good tensor network candidates for contraction.

4.4.1 Greedy strategies

As described in section 3.4, existing strategies like `zx_simp` and `zh_simp` apply simplifications until no more rules are applicable. Each such step is often assured with strictly the number of interior z-spiders from the graph-like/hypergraph-like diagrams. However, it's impact on the number of tensors and therefore the contraction complexity is not straightforward. As a first proposal, we modify the existing algorithms to keep track of the heuristic measure after every rule application. If we find a diagram that has a better value with respect to the heuristic measure than the previous one, we store it else we continue. This way, we find the graph having the lowest value of the heuristic measure generated anytime in the simplification procedure. We propose greedy versions of the base algorithms `zx_simp` and `zh_simp` called `greedy_zx_simp` and `greedy_zh_simp`. We modify the `zh_simp` algorithm before transforming it into a greedy algorithm. We observed empirically that applying the *hyperpivot* rule in batches and then simplifying the resulting graph using parallel H-box simplification routine led to a better reduction in contraction complexity than applying the pair of rules one-by-one. We then proceed similarly to the `greedy_zx_simp` algorithm by

keeping track of the heuristic measure at each step and returning the best graph generated during the simplification procedure. A summary of the sizes of the tensor network is presented in Table 4.4 and the FLOP counts for contraction are presented in Table 4.5.

cycles	`greedy_zx_simp` (#T , #I)	`greedy_zh_simp` (#T , #I)
4	38 , 40	38 , 38
6	105 , 108	98 , 100
8	183 , 186	187 , 183
10	249 , 248	243 , 241

Table 4.4: Number of tensors (#T) and the number of indices (#I) in the Tensor Network produced after applying greedy-algorithms-based simplification.

cycles	`greedy_zx_simp`	`greedy_zh_simp`
4	1.77E+04	1.83E+04
6	8.78E+04	8.89E+04
8	4.29E+08	4.46E+08
10	1.52E+10	1.00E+10

Table 4.5: Comparison of estimated FLOP counts of contraction for greedy-algorithms-based strategies.

4.4.2 Simulated annealing

Simulated annealing is a metaheuristic to approximate global optimization in a large search space for an optimization problem. Simulated annealing takes inspiration from annealing transaction of metals where the potential energy of the mass is minimized as the metal cools gradually after being subjected to high heat. We frame our search as a minimization problem where the objective function $f(g_i)$ is our heuristic measure and the input set $\{g_1, g_2, \cdots, g_n\}$ denotes the search space of diagrams generated via our strategy. If $f(g_{i+1}) < f(g_i)$, then g_{i+1} is selected as the new candidate. Otherwise, it is selected with probability $exp(-\{f(g_{i+1}) - f(g_i)\}/T)$ where T is the current temperature observed during a cooling schedule. We use this stochastic search algorithm to find the best candidate graph for contraction.

For the ZX-based strategy, we begin by splitting the base`zx_simp` algorithm into subroutines where each subroutine applies a single rule for one iteration. For example we use `lcomp_iter` which applies local complementation on only one match instead of `lcomp_simp`.

This increases the search space and also encourages mixing of rule applications. Using proxy as the objective function, we search for the best candidate graphs for contractions. Similar to the ZX-based strategy, to create a greedy version of the `zh_simp` algorithm, we split it into multiple subroutines, each applying a single rule for simplification of the graph. We however, club hyperpivot rule along with the parallel H-box simplification routine as that led to better results. Table 4.6 and 4.7 summarize the sizes of the tensor network and the FLOP counts for contraction after applying the strategies respectively.

cycles	`sim_anneal_zx_simp` (#T , #I)	`sim_anneal_zh_simp` (#T , #I)
4	54 , 46	38 , 38
6	105 , 108	98 , 100
8	183 , 186	187 , 183
10	249 , 248	243 , 241

Table 4.6: Number of tensors (#T) and the number of indices (#I) in the Tensor Network produced after applying simulated-annealing-based simplification.

cycles	`sim_anneal_zx_simp`	`sim_anneal_zh_simp`
4	1.79E+04	1.94E+04
6	8.75E+04	8.93E+04
8	6.94E+08	4.41E+08
10	1.15E+10	1.03E+10

Table 4.7: Comparison of estimated FLOP counts of contraction for simulated-annealing-based strategies.

4.4.3 Genetic algorithms

A Genetic Algorithm is another metaheuristic inspired by natural selection and classified as an evolutionary algorithm. In a genetic algorithm, the candidate solutions can be thought of as chromosomes. The objective to maximize (correspondigly minimize) is specified via the fitness function and the search space is the set of candidate solutions. The algorithm is initialized with a random population of candidate solutions. After evaluating each candidate (chromosome) using the fitness function, the algorithm selects the best ones (survival of the fittest) to generate new candidates (off-springs) through the genetic operation called crossover. The algorithm then iterates through the entire process until the desired optimal solution is

achieved. For our algorithm, we chose hypergraph-like diagrams to be the chromosomes and our proxy to be the fitness function. Each simplification procedure applied on the hypergraph-like diagram is treated as a crossover step giving rise to new candidates.

Similar to simulated annealing algorithms, we split each simplification procedure of `zx_simp` into their respective iterative versions and use each such subroutine as a method for crossover. For the case of ZH simplifications, all simplifications subroutines in `zh_simp` were split into their iterative versions to be used as individual methods for crossover. Additionally, we club fourier hyperpivot with parallel H-box simplification into one crossover step. Table 4.8 and 4.9 summarize the sizes of the tensor network and the FLOP counts for contraction after applying the strategies respectively.

cycles	`genetic_zx_simp` (#T , #I)	`genetic_zh_simp` (#T , #I)
4	38 , 40	38 , 38
6	127 , 113	98 , 100
8	183 , 186	187 , 183
10	249 , 248	243 , 241

Table 4.8: Number of tensors (#T) and the number of indices (#I) in the Tensor Network produced after applying genetic-algorithms-based simplification.

cycles	`genetic_zx_simp`	`genetic_zh_simp`
4	1.83E+04	1.70E+04
6	1.03E+05	8.94E+04
8	7.20E+08	3.67E+08
10	1.23E+10	9.52E+10

Table 4.9: Comparison of estimated FLOP counts of contraction for genetic-algorithms-based strategies.

Discussion

We compare the estimated FLOP counts of contraction for the proposed strategies with the existing ones in Table 4.10. First, we note that the time for contraction scales exponentially with the depth of the circuit for all the strategies. All the new strategies proposed, outperform the baselines strategies `zx_simp` and `zh_simp` for the given task and most of them are also

strategy\cycles	4	6	8	10
`quimb`	1.86E+04	9.78E+04	7.99E+08	1.07E+10
`zx_simp`	2.42E+04	1.37E+05	8.15E+08	1.50E+10
`zh_simp`	2.40E+04	1.37E+05	7.14E+08	1.54E+10
`greedy_zx_simp`	1.77E+04	8.78E+04	4.29E+08	1.52E+10
`greedy_zh_simp`	1.83E+04	8.89E+04	4.46E+08	1.00E+10
`sim_anneal_zx_simp`	1.79E+04	**8.75E+04**	6.94E+08	1.15E+10
`sim_anneal_zh_simp`	1.94E+04	8.93E+04	4.41E+08	1.03E+10
`genetic_zx_simp`	1.83E+04	1.03E+05	7.20E+08	1.23E+10
`genetic_zh_simp`	**1.70E+04**	8.94E+04	**3.67E+08**	**9.52E+09**

Table 4.10: Comparison of estimated FLOP counts of contraction for all strategies. Strategies having FLOP counts higher than or equal to the baseline strategy `quimb` are highlighted in light grey while the strategy having the lowest estimated FLOP counts is highlighted in bold.

able to outperform the baseline tensor network contraction library `quimb`. Among the new strategies, genetic algorithm-based strategies performed exceptionally well,giving the best overall results when used with ZH-simplifications. This alludes to the observation that order of application of graphical-calculi-based simplifications can have significant impact on the contraction complexity of the resultant tensor network. This also speaks to the importance of using a well motivated heuristic measure for guiding the strategies. Congestion inspired heuristic measures perform well for the experiments performed and could give way to more carefull designed measures in future works. For the range of depths we have shown above, ZX-based strategies perform better for small depths, while ZH-based strategies have better scaling and perform better for larger depth circuits. We believe this is because ZH based simplification procedures have been carefully designed to deal with hypergraphs natively. Hypergraphs enable a more efficient search over contraction orders for example in the case of recursive *hypergraph* bipartitioning based algorithms by allowing more freedom in the search of graph-cuts [45]. Due to computational constraints, we limited our experiments to $m = 10$ cycles so it will be interesting to evaluate our strategies on bigger instances of the problem and analyze if the same relative scaling holds between the strategies.

Chapter 5

Evaluation

This chapter presents the results of evaluating our algorithms against existing ones on more classes of randomly generated circuits. We begin by describing the construction of different families of randomly generated circuits. We then outline the experimental setup we followed to evaluate the algorithms and present the results of testing our strategies on different circuit classes. We compare the performance of different algorithms, after which the chapter concludes with a discussion on the trends we observe.

5.1 Circuit Construction

To gain an understanding of the performance of our proposed algorithms on more general circuits, we chose to evaluate them on families of randomly generated circuits where we can control the number of gates, the number of qubits and the type of gates more granularly. To this end, we considered two classes of circuits described as follows :

- IQP
- Cliffords+T

We now briefly describe the construction of each of these classes of circuits

5.1.1 IQP

In addition to the Sycamore supremacy circuits, there have been several proposals for intermediate quantum computing models, which could be used to demonstrate quantum supremacy [42, 62, 63]. A prominent model among these, known as Instantaneous Quantum Polynomial-time (IQP) circuits, is a class of commuting quantum circuits which have been shown to be hard to simulate classically, assuming certain complexity-theoretic conjectures [64]. Circuits in this class are constructed by placing arbitrary diagonal gates between a column of Hadamard. An IQP unitary on n qubits can be written in the following form:

$$U_{IQP} = H^{\otimes n} D H^{\otimes n} \tag{5.1}$$

where D is an arbitrary Z diagonal unitary with a polynomial number of gates. We construct the diagonal unitary by placing a column of powers of T gate followed by a sequence of powers of $\frac{n^2-n}{2}$ many CS (controlled-S) gates. We considered IQP circuits with varying qubit counts ranging from $n = 8$ to $n = 20$.

5.1.2 Cliffords+T

Quantum gates are usually distinguished into two classes of primitive gates: Clifford gates and non-Clifford gates. Circuits consisting only of Clifford gates can be efficiently simulated [65] but doesn not represent a universal gate set. Achieving universality requires atleast one non-Clifford gate such as the T gate (see 2). Simulating Cliffords+T circuits often scales exponentially in the number of T gates present in the circuit [35, 66, 67]. Therefore we chose to evalue Cliffords+T circuits with varying T-counts. We used two subclasses of Cliffords+T circuits defined as follows:

- `CNOT_HAD_PHASE` : These circuits are constructed by using randomly placed CNOTs, Hadamard gates and T gates.

- `cliffordT` : These circuits consist of randomly placed CNOTs, HSH and S gates

comprising the Clifford fragment of the circuit along with randomly placed T gates.

We considered two size instance of Cliffords+T circuits, one smaller instance with 4 qubits and 256 gates and the other larger instance with 8 qubits with 512 gates. For each of these circuits, we vary the T-count by varying the probability of each gate being a T gate. We consider circuits with each gate being a T gate with probabilities 10%, 20%, 30% and 40%.

5.2 Experimental Setup

We begin by taking the transition amplitude of the given circuit and representing it as a ZX diagram. Then based on the algorithm applied it is either converted into a graph-like or a hypergraph-like diagram. The strategy we wish to evaluate is then applied to the diagram and we transform the simplified diagram into its Hypergraph-like Tensor Network as detailed in section 4.1. We give this representation into the library `quimb`, which applies further local simplifications and uses the library `cotengra` to perform a search over possible contraction orders. The estimated contraction costs of the best contraction order found is returned which we use to evaluate our algorithms. We perform each such experiment 5 times and average the contraction costs to get a better understanding of general behaviour of each algorithm.

5.3 Results and Discussion

We now discuss the results we obtained by estimating the contraction costs of different strategies on the previously mentioned classes of circuits.

5.3.1 IQP

Figure 5.1 summarizes the estimated contraction costs of different strategies with varying sizes of Random IQP circuits. Each subplot compares the ZX-based strategy against the corresponding ZH-based strategy and the baseline's i.e. `quimb`'s performance is included in all the subplots for reference.

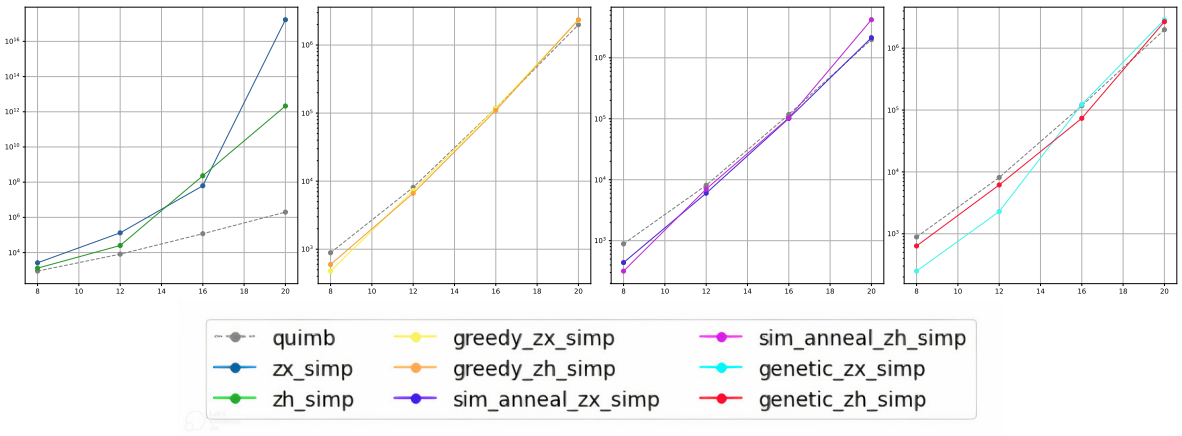


Figure 5.1: Estimated contraction costs of different strategies for `IQP` circuits plotted on log scale with increasing qubit counts.

We observe that the cost of contraction scales exponentially with the number of qubits for all the strategies.On comparing `quimb` against existing graphical calculi based strategies `zx_simp` and `zh_simp`, we see that the existing strategies are outperfromed by `quimb`. This is similar to our observation with Sycamore circuits where the existing graphical-calculi-based strategies led to dense tensor networks which in turn affected the contraction costs. We note that our proposed strategies, which use heuristic measures to guide application of simplification procedures, perform better than the existing ones and have costs comparable to the baseline. Greedy strategies show no clear difference between ZX-based strategies and ZH-based strategies. For simulated annealing based algorithms however, we not that ZH-based strategies perform better for smaller circuits while ZX-based strategies perform better for larger circuits and an opposite trend can be observed for genetic-algorithm based strategies. Overall, we observe that genetic-algorithm based strategies perform the best and have similar scaling as compared to the baseline `quimb`.

5.3.2 Cliffords+T

CNOT_HAD_PHASE

Figure 5.2 and 5.3 summarize the estimated contraction costs of different strategies with varying circuit parameters.

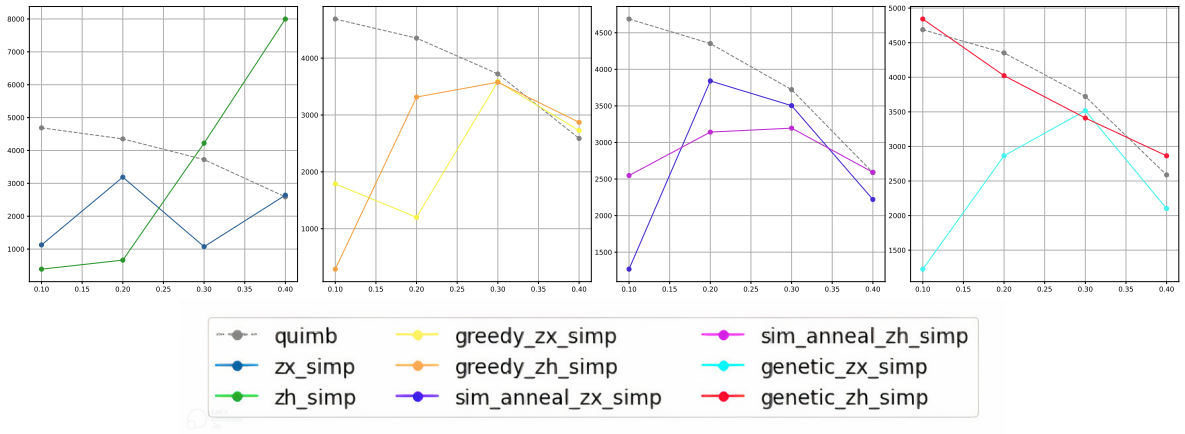


Figure 5.2: Estimated contraction costs of different strategies for `CNOT_HAD_PHASE` circuits on **4 qubits** and **256 gates** plotted against increasing T-gate probability.

For the smaller 4 qubit 256 gatecount circuit, we first observe that the baseline algorithm `quimb` has a decrease in contraction costs with increasing T count. This is possibly because increasing the T count leads to decrease in the CNOT counts which in turn makes the circuits less dense. As `quimb` is concerned with the geometry of the tensor networks, and mostly agnostic to the values of the tensors inside, it is expected to find better contraction orders with sparser tensor networks. Moving to the comparison of `quimb` against existing graphical calculi based strategies `zx_simp` and `zh_simp`, we do not observe a clear advantage. While `zx_simp` performs well for smaller T-counts, it's advantage over `quimb` in the case of larger T-count circuits is not clear. For `zh_simp` we see that is also performs well from smaller T-count circuits but does worse as the T-count goes up. Among the proposed strategies, we observe that almost all perform better than the baseline. In most cases, ZX-based strategies seem to outperform ZH-based strategies. For higher T count i.e. with circuits having T-gate probability $\geq 30\%$, we see that the performance of proposed strategies is close to baseline. This alludes to the fact that graphical calculi based strategies perform best when the circuits are mostly Clifford as there is more applicability of simplifications for cliffords.

We now move to the larger 8 qubit 512 gatecount circuits where we see a similar trend of improvement in the estimated contraction costs for `quimb` with increasing T-count. The baseline strategies `zx_simp` and `zh_simp` perform poorly for these large circuits whereas the

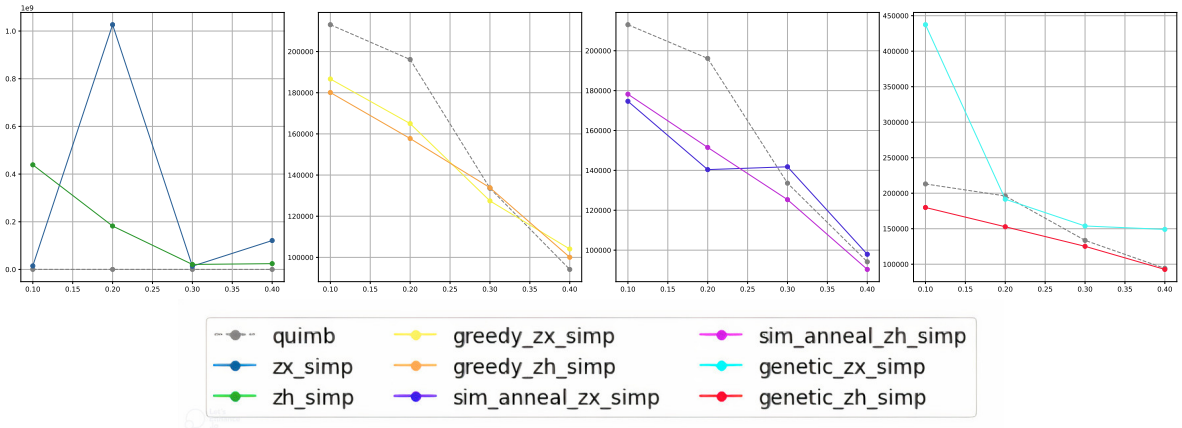


Figure 5.3: Estimated contraction costs of different strategies for `CNOT_HAD_PHASE` circuits on **8 qubits** and **512 gates** plotted against increasing T-gate probability.

newly proposed strategies seem to mostly beat the baseline. While ZX-based strategies performed better for smaller circuits, we observe that ZH-based strategies perform better for larger circuits. We also note that the contraction costs for all the proposed strategies decrease with an increase in T-count and similar to the previous case, circuits having T-gate probability $\geq 30\%$ have contraction costs very close to baseline.

cliffordT

Figure 5.4 and 5.5 summarize the estimated contraction costs of different strategies with varying circuit parameters.

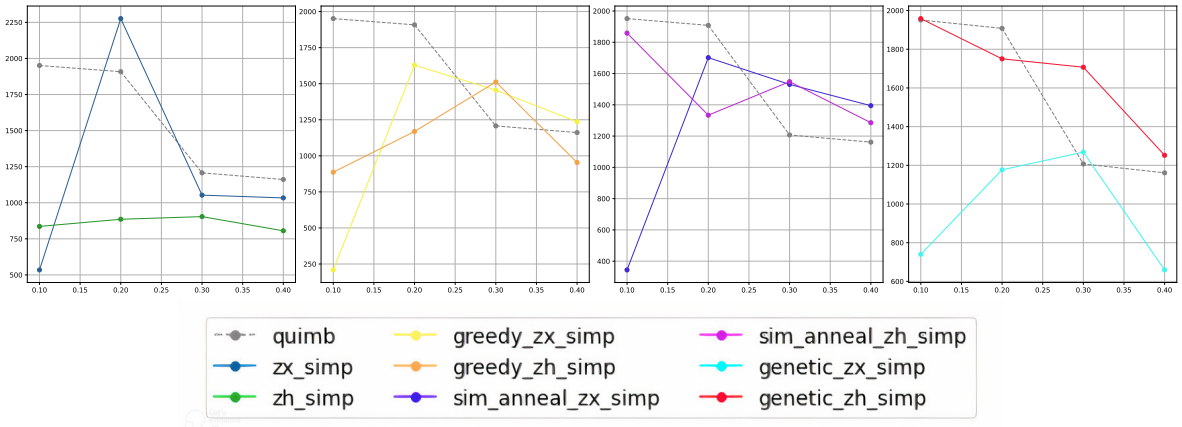


Figure 5.4: Estimated contraction costs of different strategies for `clifford_T` circuits on **4 qubits** and **256 gates** plotted against increasing T-gate probability.

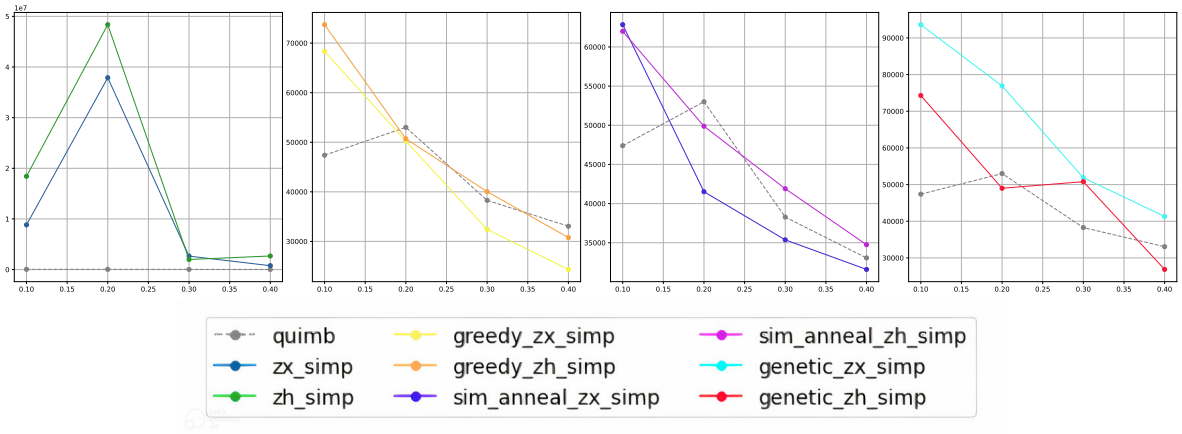


Figure 5.5: Estimated contraction costs of different strategies for `clifford_T` circuits on **8 qubits** and **512 gates** plotted against increasing T-gate probability.

Similar to `CNOT_HAD_PHASE` circuits, we see that the contraction costs for `quimb` decreases with increase in T-count of the circuits. For the smaller circuits, we again observe that the performance of `zx_simp` and `zh_simp` is similar to `quimb`. Most of the proposed strategies perform better than the baseline and in general ZX-based strategies perform better than ZH-based strategies. For larger circuits, existing graphical-calculi-based strategies perform much worse while the proposed strategies are still mostly better than the baseline. For greedy strategies and simulated annealing based strategies, ZX-based strategies still perform better while ZH-based subroutines are better for genetic algorithms based strategy.

Chapter 6

Conclusion and Future Work

In this dissertation, we investigated the use of quantum graphical calculi in enhancing tensor-network-based quantum circuit simulations. We proposed a framework for using graphical simplifications to simplify quantum circuits before feeding them into a tensor-network contraction pipeline. We evaluated existing graphical simplification procedures in our framework and found that they often resulted in dense tensor networks and thus adversely affected the complexity of simulations. To help guide simplification procedures, we proposed a heuristic measure to estimate contraction costs of underlying graphical representations. We also proposed three new graphical simplification strategies based on the heuristic measure and evaluated them against existing strategies.

To evaluate our framework, we considered the Sycamore supremacy circuits used to demonstrate quantum supremacy by Google in 2019. We set the tensor network contraction framework without any graphical-calculi-based simplifications as the baseline and compared its performance against the pipeline enhanced with existing and the newly proposed graphical-calculi-based strategies. On Sycamore circuits, we found that the proposed strategies perform much better than existing strategies in reducing the contraction complexity of tensor networks. Moreover, we found at least one proposed strategy to

outperform the baseline for all the considered circuit depths. We also observed that ZX-based strategies performed better for smaller circuit instances, but ZH-based strategies performed much better as the circuit depths grew.

In addition to Sycamore circuits, we evaluated our strategies on Random IQP circuits and two classes of random Cliffords+T circuits. For Random IQP circuits, we observed similar behavior to Sycamore circuits, with the proposed strategies outperforming the existing strategies. The proposed strategies beat the baseline often but were very close. A similar trend was observed concerning comparing ZX- and ZH-based strategies, with ZX-based strategies performing better for smaller circuit instances and ZH-based strategies performing better for larger circuit instances. For Random Cliffords+T circuits, we considered fixed circuit sizes but varying T-counts. Existing graphical-calculi-based strategies performed similarly to the baseline for smaller circuit instances but performed much worse for the larger circuits. The proposed strategies consistently outperformed the baseline, but their advantage over the baseline decreased with increasing T-counts. This was in accordance with the fact that graphical-calculi-based strategies can find a lot more simplifications when the circuits are mostly Clifford and show a decrease in their advantage over the baseline as the T-counts increase.

Although we observed that our proposed strategies often beat the baseline and the existing strategies, we were limited by computational resources and could only benchmark a limited set of circuits. For example we benchmarked Sycamore circuits up to a cycle depth of 10 but the proposed depth for an experiment demonstrating quantum supremacy was 20. It would thus be interesting to benchmark bigger circuits and analyze the performance of the proposed strategies in future work. It will also be useful to evaluate our strategies on more circuit classes and check if the observations we made still hold for other classes and bigger instances of the same circuit class.

Our framework used the same heuristic measure for all the strategies and, therefore, might not be able to make a fair comparison across different strategies. It would be interesting to develop

strategy-specific heuristic measures and draw comparisons between strategies. Also, because we observed that genetic-algorithms-based strategies performed the best overall, evaluating more evolutionary-search-based strategies is a promising research direction.

In this dissertation, we limited our scope to performing structural simplifications on tensor networks using graphical calculus. An extension to our framework could be analyzing how to improve the contraction pathfinders using graphical calculi. An idea worth exploring is to use the graphical-calculi-based representation of quantum circuits to guide hypergraph partitioning of the tensor network, an algorithm that can often find the most efficient contraction orders.

Bibliography

[1] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, oct 1997.

[2] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[3] Seth Lloyd. Quantum algorithm for solving linear systems of equations. In *APS March Meeting Abstracts*, volume 2010, pages D4–002, 2010.

[4] Roger Penrose. Applications of negative dimensional tensors. In *Combinatorial Mathematics and its Applications (Proc. Conf., Oxford, 1969)*, pages 221–244. Academic Press, London, 1971.

[5] J Ignacio Cirac and Frank Verstraete. Renormalization and tensor product states in spin chains and lattices. *Journal of physics a: mathematical and theoretical*, 42(50):504004, 2009.

[6] Simone Montangero, Montangero, and Evenson. *Introduction to Tensor Network Methods*. Springer, 2018.

[7] Shi-Ju Ran, Emanuele Tirrito, Cheng Peng, Xi Chen, Luca Tagliacozzo, Gang Su, and Maciej Lewenstein. *Tensor network contractions: methods and applications to quantum many-body systems*. Springer Nature, 2020.

[8] Guifre Vidal, José Ignacio Latorre, Enrique Rico, and Alexei Kitaev. Entanglement in quantum critical phenomena. *Physical review letters*, 90(22):227902, 2003.

[9] Jens Eisert, Marcus Cramer, and Martin B Plenio. Colloquium: Area laws for the entanglement entropy. *Reviews of modern physics*, 82(1):277, 2010.

[10] Garnet Kin-Lic Chan and Sandeep Sharma. The density matrix renormalization group in quantum chemistry. *Annual review of physical chemistry*, 62(1):465–481, 2011.

[11] Sebastian Keller, Michele Dolfi, Matthias Troyer, and Markus Reiher. An efficient matrix product operator representation of the quantum chemical hamiltonian. *The Journal of chemical physics*, 143(24):244118, 2015.

[12] Szilárd Szalay, Max Pfeffer, Valentin Murg, Gergely Barcza, Frank Verstraete, Reinhold Schneider, and Örs Legeza. Tensor product methods and entanglement optimization for ab initio quantum chemistry. *International Journal of Quantum Chemistry*, 115(19):1342–1391, 2015.

[13] Garnet Kin-Lic Chan, Anna Keselman, Naoki Nakatani, Zhendong Li, and Steven R White. Matrix product operators, matrix product states, and ab initio density matrix renormalization group algorithms. *The Journal of chemical physics*, 145(1):014102, 2016.

[14] Huanchen Zhai and Garnet Kin-Lic Chan. Low communication high performance ab initio density matrix renormalization group algorithms. *The Journal of Chemical Physics*, 154(22):224116, 2021.

[15] Brian Swingle. Entanglement renormalization and holography. *Physical Review D*, 86(6):065007, 2012.

[16] Masamichi Miyaji, Tokiro Numasawa, Noburo Shiba, Tadashi Takayanagi, and Kento Watanabe. Continuous multiscale entanglement renormalization ansatz as holographic surface-state correspondence. *Physical review letters*, 115(17):171602, 2015.

[17] F Pastawski, B Yoshida, D Harlow, and J Preskill. Holographic quantum error-correcting codes: Toy models for the bulk/boundary correspondence.-preprint. 2015.

[18] Patrick Hayden, Sepehr Nezami, Xiao-Liang Qi, Nathaniel Thomas, Michael Walter, and Zhao Yang. Holographic duality from random tensor networks. *Journal of High Energy Physics*, 2016(11):1–56, 2016.

[19] Bartlomiej Czech, Lampros Lamprou, Samuel McCandlish, and James Sully. Tensor networks from kinematic space. *Journal of High Energy Physics*, 2016(7):1–38, 2016.

[20] Glen Evenbly. Hyperinvariant tensor networks and holography. *Physical review letters*, 119(14):141602, 2017.

[21] Edwin Stoudenmire and David J Schwab. Supervised learning with tensor networks. *Advances in Neural Information Processing Systems*, 29, 2016.

[22] John Martyn, Guifre Vidal, Chase Roberts, and Stefan Leichenauer. Entanglement and tensor networks for supervised image classification. *arXiv preprint arXiv:2007.06082*, 2020.

[23] Song Cheng, Lei Wang, and Pan Zhang. Supervised learning with projected entangled pair states. *Physical Review B*, 103(12):125117, 2021.

[24] Jing Liu, Sujie Li, Jiang Zhang, and Pan Zhang. Tensor networks for unsupervised machine learning. *arXiv preprint arXiv:2106.12974*, 2021.

[25] Yuhan Liu, Wen-Jun Li, Xiao Zhang, Maciej Lewenstein, Gang Su, and Shi-Ju Ran. Entanglement-based feature extraction by tensor network machine learning. *Frontiers in Applied Mathematics and Statistics*, 7:716044, 2021.

[26] E. Schuyler Fried, Nicolas P. D. Sawaya, Yudong Cao, Ian D. Kivlichan, Jhonathan Romero, and Alá n Aspuru-Guzik. qTorch: The quantum tensor contraction handler. *PLOS ONE*, 13(12):e0208510, dec 2018.

[27] Benjamin Villalonga, Sergio Boixo, Bron Nelson, Christopher Henze, Eleanor Rieffel, Rupak Biswas, and Salvatore Mandrà. A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware. *npj Quantum Information*, 5(1), oct 2019.

[28] Roman Schutski, Danil Lykov, and Ivan Oseledets. Adaptive algorithm for quantum circuit simulation. *Physical Review A*, 101(4):042335, 2020.

[29] Feng Pan and Pan Zhang. Simulation of quantum circuits using the big-batch tensor network method. *Physical Review Letters*, 128(3):030501, 2022.

[30] Maksim Levental. Tensor networks for simulating quantum circuits on fpgas. *arXiv preprint arXiv:2108.06831*, 2021.

[31] Trevor Vincent, Lee J O'Riordan, Mikhail Andrenkov, Jack Brown, Nathan Killoran, Haoyu Qi, and Ish Dhand. Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction. *Quantum*, 6:709, 2022.

[32] Bob Coecke and Ross Duncan. Interacting quantum observables. In *International Colloquium on Automata, Languages, and Programming*, pages 298–310. Springer, 2008.

[33] Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, apr 2011.

[34] Miriam Backens and Aleks Kissinger. ZH: A complete graphical calculus for quantum computations involving classical non-linearity. *Electronic Proceedings in Theoretical Computer Science*, 287:23–42, jan 2019.

[35] Aleks Kissinger and John van de Wetering. Reducing the number of non-clifford gates in quantum circuits. *Physical Review A*, 102(2), aug 2020.

[36] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic simplification of quantum circuits with the ZX-calculus. *Quantum*, 4:279, jun 2020.

[37] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[38] P Oscar Boykin, Tal Mor, Matthew Pulver, Vwani Roychowdhury, and Farrokh Vatan. On universal and fault-tolerant quantum computing: a novel basis and a new constructive proof of universality for shor's basis. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 486–494. IEEE, 1999.

[39] Scott Aaronson and Lijie Chen. Complexity-theoretic foundations of quantum supremacy experiments. *arXiv preprint arXiv:1612.05903*, 2016.

[40] Alexander Zlokapa, Sergio Boixo, and Daniel Lidar. Boundaries of quantum supremacy via random circuit sampling. *arXiv preprint arXiv:2005.02464*, 2020.

[41] Scott Aaronson and Sam Gunn. On the classical hardness of spoofing linear cross-entropy benchmarking. *arXiv preprint arXiv:1910.12085*, 2019.

[42] Adam Bouland, Bill Fefferman, Chinmay Nirkhe, and Umesh Vazirani. On the complexity and verification of quantum random circuit sampling. *Nature Physics*, 15(2):159–163, 2019.

[43] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J Bremner, John M Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595–600, 2018.

[44] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, oct 2019.

[45] Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, mar 2021.

[46] Cupjin Huang, Fang Zhang, Michael Newman, Junjie Cai, Xun Gao, Zhengxiong Tian, Junyin Wu, Haihong Xu, Huanjun Yu, Bo Yuan, et al. Classical simulation of quantum supremacy circuits. *arXiv preprint arXiv:2005.06787*, 2020.

[47] Charles Neill, Pedran Roushan, K Kechedzhi, Sergio Boixo, Sergei V Isakov, V Smelyanskiy, A Megrant, B Chiaro, A Dunsworth, K Arya, et al. A blueprint for demonstrating quantum supremacy with superconducting qubits. *Science*, 360(6385):195–199, 2018.

[48] Jacob D. Biamonte, Jason Morton, and Jacob Turner. Tensor network contractions for #SAT. *Journal of Statistical Physics*, 160(5):1389–1404, jun 2015.

[49] Bryan O'Gorman. Parameterization of tensor network contraction. 2019.

[50] John van de Wetering. Zx-calculus for the working quantum computer scientist. *arXiv preprint arXiv:2012.13966*, 2020.

[51] Aleks Kissinger and John van de Wetering. PyZX: Large scale automated diagrammatic reasoning. *Electronic Proceedings in Theoretical Computer Science*, 318:229–241, may 2020.

[52] Bob Coecke and Aleks Kissinger. Picturing quantum processes. In *International Conference on Theory and Application of Diagrams*, pages 28–31. Springer, 2018.

[53] John van de Wetering and Sal Wolffs. Completeness of the phase-free zh-calculus. *arXiv preprint arXiv:1904.07545*, 2019.

[54] Yaoyun Shi. Both toffoli and controlled-not need little help to do universal quantum computation. *arXiv preprint quant-ph/0205115*, 2002.

[55] Dorit Aharonov. A simple proof that toffoli and hadamard are quantum universal. *arXiv preprint quant-ph/0301040*, 2003.

[56] Jeroen Dehaene and Bart De Moor. Clifford group, stabilizer states, and linear and quadratic operations over gf (2). *Physical Review A*, 68(4):042318, 2003.

[57] Ross Duncan and Simon Perdrix. Pivoting makes the zx-calculus complete for real stabilizers. *arXiv preprint arXiv:1307.7048*, 2013.

[58] Louis Lemonnier, John van de Wetering, and Aleks Kissinger. Hypergraph simplification: Linking the path-sum approach to the ZH-calculus. *Electronic Proceedings in Theoretical Computer Science*, 340:188–212, sep 2021.

[59] Stach Kuijpers, John van de Wetering, and Aleks Kissinger. Graphical fourier theory and the cost of quantum addition. *arXiv preprint arXiv:1904.07551*, 2019.

[60] Ryan krueger. Vanishing 2-qubit gates with on simplification ZX-rules. `https://www.youtube.com/watch?v=tUIcqXKEFhk`, 2021. [Online; accessed 30-August-2021].

[61] Korbinian Staudacher. Optimization Approaches for Quantum Circuits using ZX-calculus . `https://www.mnm-team.org/pub/Diplomarbeiten/stau21/`, 2021. [Online; accessed 30-August-2021].

[62] Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 333–342, 2011.

[63] James King, Sheir Yarkoni, Jack Raymond, Isil Ozfidan, Andrew D King, Mayssam Mohammadi Nevisi, Jeremy P Hilton, and Catherine C McGeoch. Quantum annealing amid local ruggedness and global frustration. *Journal of the Physical Society of Japan*, 88(6):061007, 2019.

[64] Michael J Bremner, Ashley Montanaro, and Dan J Shepherd. Achieving quantum supremacy with sparse and noisy commuting quantum computations. *Quantum*, 1:8, 2017.

[65] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), nov 2004.

[66] Sergey Bravyi and David Gosset. Improved classical simulation of quantum circuits dominated by clifford gates. *Physical review letters*, 116(25):250501, 2016.

[67] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. Simulation of quantum circuits by low-rank stabilizer decompositions. *Quantum*, 3:181, 2019.