INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

Optimization Approaches for Quantum Circuits using ZX-calculus

Korbinian Staudacher

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Sophia Grundner-Culemann Dr. Tobias Guggemos (DLR) Dr. Wolfgang Gehrke (UniBw M)
Abgabetermin:	30. September 2021

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

Optimization Approaches for Quantum Circuits using ZX-calculus

Korbinian Staudacher

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Sophia Grundner-Culemann Dr. Tobias Guggemos (DLR) Dr. Wolfgang Gehrke (UniBw M)
Abgabetermin:	30. September 2021

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 30. September 2021

.....
(Unterschrift des Kandidaten)

Abstract

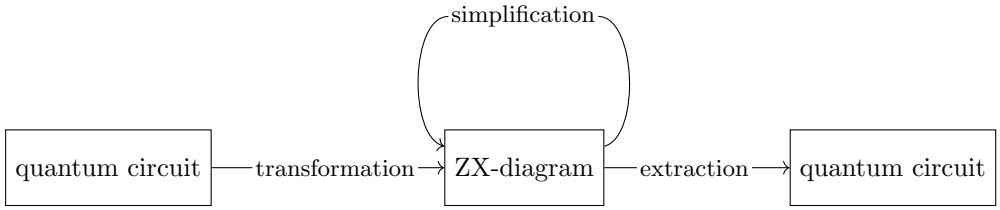
Quantum computing is a promising research field in modern computer science as quantum computers have the potential to solve some computationally hard problems much faster than classical computers. Instead of bits, quantum computers use two-state quantum mechanical systems called *qubits* which have some useful properties such as entanglement and superposition that have no bit equivalent. Similar to classical computers where bits are modified by Boolean gates on circuits, qubits can be modified by quantum-logical gates on quantum circuits.
While in theory there are many applications for quantum computers from a variety of different research areas, in practice most them cannot yet be realized on up to date quantum computers as the size of quantum circuits is strongly limited by hardware and physical constraints. Therefore, it is crucial to optimize quantum circuits as far as possible in terms of size and complexity.
This work focuses on quantum circuit optimization using the ZX-calculus, a recently developed graphical language designed to simplify reasoning about quantum systems. Quantum circuits can be optimized in an intuitive and efficient way by transforming them to equivalent ZX-diagrams and using rules of the ZX-calculus for diagram simplification.
However, some rules can modify ZX-diagrams in such a way that the re-extraction of a quantum circuit is no longer possible. Moreover, rules that simplify ZX-diagrams can still increase the size of the underlying circuits. Due to these problems, most of the existing ZX-calculus based optimization approaches become inefficient with increasing quantum circuit complexity.
In our work we develop different strategies to improve those approaches. In particular, we introduce heuristics to estimate the optimization benefit gained by certain ZX-rules. This allows treating ZX-diagram simplification as a classical search problem where heuristics can be applied to guide the sequence of rule applications towards minimal circuits. We implement different heuristic-based algorithms like greedy search, random search and simulated annealing in the open-source library PyZX where we test them against existing strategies on circuits of variant size. The results show that using heuristics in diagram simplification often leads to better overall optimization results, especially when optimizing large and complex quantum circuits. Our algorithms can be further improved in several aspects like runtime or consideration of hardware topology.

Contents

1 Introduction

Quantum computers have the potential to outperform classical computers since some properties of quantum systems like quantum entanglement can help solving hard problems in computer science much faster. A well-known application for quantum computers is the factorization of integers using Shor's algorithm which solves the problem in polynomial instead of superpolynomial time [27]. While this algorithm has achieved most public interest since it provides an attack on the widely-used RSA cryptosystem, quantum computers are expected to solve many other problems of different research fields more efficiently. In computer science, quantum search algorithms like Grover's algorithm can be used to speed up search problems significantly [11], in physics, quantum computers can be used as a tool for simulating quantum systems[6], and in chemistry, quantum computers are expected to calculate energy spectrums of molecular systems much faster [17].
At its core, quantum computers use two-state quantum mechanical systems called *qubits*. Compared to classical bits, qubits allow the exploitation of quantum effects like entanglement and superposition which have no classical equivalent. We can modify qubits using quantum gates on quantum circuits similar to classical computing where we can modify bits with Boolean gates.
In practice, realizing a stable quantum system suitable for quantum computations turns out to be very challenging as quantum gates are error-prone and qubits can interact with their environment which destroys the computation. As of 2021, the maximum number of qubits on a quantum computer is 65[1], so the computational power of quantum computers is still very limited.
To make the most of the resources of quantum computers, it is crucial to develop quantum circuits which are as efficient as possible. This has spawned a new field of research called quantum circuit optimization which focuses on finding algorithms for reducing size and complexity of quantum circuits. Optimizing quantum circuits in general is QMA-hard[2], which means most likely there exists no algorithm with polynomial runtime which returns an optimal solution for arbitrary quantum circuits. Nonetheless, we can construct polynomial algorithms which reduce the size of most circuits to some extent.
We focus on quantum circuit optimization using the ZX-calculus which is a recently developed graphical language designed to simplify reasoning about quantum systems. We can transform any quantum circuit to an equivalent representation in ZX-calculus called ZX-diagram and use rules of the ZX-calculus to simplify the diagram instead of the circuit as follows:

[1] https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/
[2] QMA is short for Quantum Merlin Arthur, a complexity class for quantum computers containing all problems from NP.



Since ZX-diagrams are not bound to the rigid structure of quantum circuits, there are optimizations which have no circuit analogue and ZX-calculus has been considered a promising tool for quantum circuit optimization [8].

Problem statement

While current ZX-calculus based approaches achieve very good results for a class of simple quantum circuits named *Clifford circuits*, most approaches fail to optimize more complex circuit classes effectively. This is often due to the fact that two-qubit quantum gates, i.e., quantum gates which act on two qubits at the same time, are not optimized very well. In fact, their number sometimes even increases, which is a serious drawback as they are usually one of the most expensive gates in terms of hardware cost.

Research question

Our research question regarding quantum circuit optimization in ZX-calculus is as follows:

Which strategies can be used to further improve ZX-calculus based quantum circuit optimization?

As main aspect we introduce heuristics for some ZX-rule applications used during the diagram simplification procedure which allow us to estimate the impact of those rules on the underlying quantum circuit in terms of two-qubit gates. Using heuristics we turn ZX-diagram simplification into a classical search problem where we can apply strategies like the greedy algorithm or the simulated annealing algorithm in order to find minimal ZX-diagrams. We develop heuristic-based algorithms and compare them to both the existing ZX-calculus based approaches and an up-to-date optimization algorithm which does not use ZX-calculus.

Methodology

In Chapter 2, we give a short overview of quantum computing and ZX-calculus. We explain the core mathematical foundations of qubits and quantum gates, how they can be composed to quantum circuits and which basic optimization steps are possible. Furthermore, we introduce the ZX-calculus and show how it can be used to represent quantum circuits. Chapter 3 focuses on the existing ZX-calculus based approaches for quantum circuit optimization. We explain the different ZX-rules used for diagram simplification and show how quantum circuits can be extracted from diagrams. Furthermore, we evaluate the existing algorithms in terms of two-qubit gate reduction which serves as a motivation for our own strategies. In Chapter 4 we develop some new simplification algorithms. We first show how the recently developed Euler-rule in ZX-calculus can be used to reduce arbitrary gates in theory before focusing on the main part of heuristic-based simplification algorithms. Chapter 5 evaluates our strategies against existing quantum optimization algorithms and Chapter 6 summarizes our results.

2 Background

In this chapter, we provide a brief introduction to the research areas relevant for our thesis. First, we outline the basic properties of quantum computing, then we explain quantum circuits and their optimization, and finally we give a short overview of the ZX-calculus.

2.1 Basics of quantum computing

At its core, quantum computing is the controlled manipulation of quantum systems in order to perform computations. While the details of quantum systems are outside the scope of this work, modern physics has established some fundamental postulates providing the foundation for a mathematical description of quantum systems. In the following section we discuss the most essential postulates and how they define the basics of quantum computing. We then focus on quantum circuits and how they are constructed from different quantum gates. A more detailed introduction to quantum computing can be found in [23].

2.1.1 Single qubit systems

The state of an isolated quantum system can be completely described by a unit state vector in a *Hilbert space*. State vectors are usually denoted as $|\psi\rangle$, where $|.\rangle$, the so called "ket" in Dirac notation, represents a complex column vector. Together with the "bra", denoted as $\langle\psi|$, which is a complex row vector, and matrix multiplication as inner product, we have a Hilbert space. A state space can have different spanning sets, which are sets of linear independent vectors $|\phi_1\rangle .. |\phi_n\rangle$ such that every vector in the state space can be written as a linear combination of these basis vectors. The most common representation of a qubit is as a state vector in $\mathbb{C}^2$

$$|\psi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \alpha\,|0\rangle + \beta\,|1\rangle\,, \alpha, \beta \in \mathbb{C}, \tag{2.1}$$

where α and β are called *amplitudes* and $\{|0\rangle\,, |1\rangle\}$ is the spanning set with

$$|0\rangle = \begin{pmatrix}1\\ 0\end{pmatrix}, |1\rangle = \begin{pmatrix}0\\ 1\end{pmatrix}. \tag{2.2}$$

This equation is also referred to as superposition, since $|\psi\rangle$ not only has two possible states 0 and 1, but infinitely many depending on its amplitudes. The only constraint, as $|\psi\rangle$ is a unit vector, is that the sum of the squared amplitudes equals 1:

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2.3}$$

For easier visualization, Equation 2.1 can be rewritten as

$$|\psi\rangle = cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi} sin\frac{\theta}{2}\,|1\rangle\,, \tag{2.4}$$

which allows representing a qubit on the three dimensional *Bloch sphere* as a unit vector defined by two rotations θ and φ around the X and Z axis (c.f. Figure 2.1). For consistency we express rotation angles in radians, i.e. as fractions of π, instead of degrees throughout the work, so $\theta, \varphi \in [0, 2\pi)$.

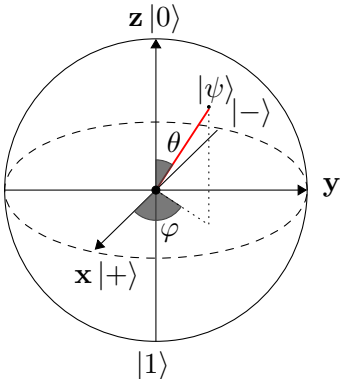


Figure 2.1: Single qubit $|\psi\rangle$ as red unit vector on Bloch sphere

Although $\{|0\rangle, |1\rangle\}$ is the most common spanning set of $\mathbb{C}^2$, we sometimes also use the so called *Hadamard base*:

$$|+\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, |-\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}. \tag{2.5}$$

However, apart from the different basis vectors the qubit equation looks the same:

$$|\psi\rangle = \begin{pmatrix} \alpha' \\ \beta' \end{pmatrix} = \alpha' |+\rangle + \beta' |-\rangle. \tag{2.6}$$

2.1.2 Single qubit gates

The evolution of an isolated quantum system is described by unitary transformations, which means a state vector $|\psi\rangle$ can be modified by multiplication with some unitary matrix U:

$$|\psi'\rangle = U |\psi\rangle. \tag{2.7}$$

An important property of unitary matrices is that they are invertible, and that the inverse is equal to the conjugate transpose, which is denoted by a dagger† symbol. This means for every unitary matrix U there exists an matrix $U^\dagger$ such that $UU^\dagger = I$. In quantum computing unitary transformations are interpreted as logical *gates* U acting on the qubit state $|\psi\rangle$, just like in classical computing gates like XOR, AND, etc. modify bits. Since every unitary matrix is invertible, every operation in quantum computing has to be reversible. In classical computing some this is not always the case: take for instance an operation which sets a bit to 1. Such non-invertible logical operations have no quantum gate equivalent.
The most basic gates are the so called *Pauli* gates, which correspond to the *Pauli* matrices:

$$-\boxed{I}- = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad -\boxed{X}- = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad -\boxed{Y}- = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad -\boxed{Z}- = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.8}$$

Apart from the identity gate I, these gates rotate a single qubit by an angle of π around the corresponding axis on the Bloch sphere. For instance, if we apply an X gate to a qubit in state $|0\rangle$ we get:

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle . \tag{2.9}$$

As can be seen in Figure 2.1, the transformation from $|0\rangle$ to $|1\rangle$ corresponds exactly to a rotation of π around the X-axis.
Pauli matrices are self-inverse, therefore applying the same Pauli gate twice in a row has no effect on the qubit state:

$$XX|0\rangle = X|1\rangle = |0\rangle . \tag{2.10}$$

On the Bloch sphere this is also clearly visible, since two identical Pauli gates correspond to a full rotation of 2π.
Some other important single qubit gates are the Hadamard gate

$$-\boxed{H}- = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.11}$$

and the Z-phase gate

$$-\boxed{Z_\alpha}- = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}, \alpha \in [0, 2\pi). \tag{2.12}$$

The Hadamard gate is crucial for quantum computing as it transforms any qubit in basis state to an equal superposition where both amplitudes are the same, e.g.

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = |+\rangle . \tag{2.13}$$

As the Hadamard gate is again self-inverse, an equal superposition can also be transformed back to a basis state:

$$H|+\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle . \tag{2.14}$$

On the Bloch sphere the Hadamard gate corresponds to a rotation of π around the Z-axis followed by $\pi/2$ around the Y-axis.
The Z-phase gate is a generalization of the Pauli Z gate allowing arbitrary rotations around the Z-axis parametrized by an angle α. Apart from the Pauli Z gate with $\alpha = \pi$, the most prominent are the S gate with $\alpha = \frac{\pi}{2}$ and the T gate with $\alpha = \frac{\pi}{4}$:

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \qquad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} . \tag{2.15}$$

Z-phase gates are in general not self-inverse, e.g.

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix} = S^\dagger . \tag{2.16}$$

2.1.3 Measurement

An essential problem in quantum computing is that in general we cannot observe a state vector and determine its current amplitudes. The only way to get information about the current state of a qubit is by performing a measurement relative to a specific basis. The measurement, however, destroys the superposition of a qubit, which means we can only get two mutually exclusive results depending on the measurement basis. If we measure a single qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ in basis $|0\rangle$ and $|1\rangle$, the probability of obtaining $|0\rangle$ is $|\alpha|^2$ and for $|1\rangle$ it is $|\beta|^2$.
For a qubit in equal superposition, e.g. $H|0\rangle$ in equation 2.13, measuring in the $\{|0\rangle, |1\rangle\}$ basis returns either $|0\rangle$ or $|1\rangle$ with a probability of 50%. If we measure $H|0\rangle$ in Hadamard base instead, the result is $|+\rangle$ with a probability of 100%. The measurement gate is denoted as

 (2.17)

and is the only non-unitary and non-reversible gate in quantum computing.

2.1.4 Multi qubit systems

The state vector of a composite quantum system is described by the *tensor product* $\otimes$ of the individual state vectors. As we only deal with complex vectors, the tensor product is effectively the *Kronecker product*. Applying the Kronecker product on two single qubits $|a\rangle$ and $|b\rangle$ produces:

$$|a\rangle \otimes |b\rangle = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \otimes \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_1 * b_1 \\ a_1 * b_2 \\ a_2 * b_1 \\ a_2 * b_2 \end{pmatrix}. \tag{2.18}$$

For simplification, $|a\rangle \otimes |b\rangle$ is often abbreviated as $|ab\rangle$. A two-qubit state space now has four basis vectors, e.g.:

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.19}$$

and we can represent the product $|ab\rangle$ as:

$$|\psi\rangle = a_1 b_1 |00\rangle + a_1 b_2 |01\rangle + a_2 b_1 |10\rangle + a_2 b_2 |11\rangle. \tag{2.20}$$

Each single qubit gate can also be extended to multiple qubits using the tensor product:

$$X^2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}. \tag{2.21}$$

Yet there are some special two-qubit gates which cannot be represented as a tensor product of single qubit gates. The most important ones are the $CNOT$ and the CZ gate:

$$CNOT = \quad = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.22}$$

$$CZ = \quad = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{2.23}$$

For qubits in state $|0\rangle$ or $|1\rangle$, the CNOT gate performs a classical XOR operation. While the upper qubit (control bit) remains unchanged, the lower qubit (target bit) gets flipped if the control bit is set to $|1\rangle$ else it remains the same. CZ gates act equal on two qubits except that the amplitude of the target bit gets flipped from positive to negative and vice versa if the control qubit is $|1\rangle$. CNOT and CZ gates are also self-inverse: Flipping the target bit twice is equal to not flipping the target bit at all.

2.1.5 Minimal and universal gate sets

In general we can decompose every unitary operation on multiple qubits into a combination of CNOTs and single qubit unitary operations. Furthermore, for single qubits we can also find universal gate sets from which we can generate every other unitary operation. An important set in this context is the *Clifford group*. For n qubits the elements of the Clifford group are defined as

$$C_n = \left\{ V \in U_{2^n} | VPV^\dagger = P' \right\}, \tag{2.24}$$

where P, P' are arbitrary (negated) Pauli matrices with dimension 2^n. Simply put, these are all matrices where its product with any Pauli matrix combined with its inverse results in a Pauli matrix again. For the single qubit case $n = 1$, the gates in the Clifford group are $\{S, H, I, X, Y, Z\}$ as well as their negatives $\{-S, -H, \ldots\}$, their inverses $\left\{S^\dagger, H^\dagger, \ldots\right\}$ and their negative inverses $\left\{-S^\dagger, -H^\dagger, \ldots\right\}$. However, the Clifford group is not universal. By visualizing the possible operations in C_1 on the Bloch sphere we can observe that every state resulting from any Clifford operation on the state $|0\rangle$ is a rotation of $\pi/2, \pi, 3\pi/2$ or 2π, so we end up on one of the X,Y or Z axes each time. Moreover, an important theorem by Gottesman and Knill shows that quantum circuits containing only Clifford gates can be efficiently simulated on a classical computer, so we would lose the advantage of faster calculations by restricting the possible gates to Clifford gates [1].
Yet it can be shown that we can approximate any single qubit unitary operation by using only H and T gates. According to the Solovay-Kitaev theorem, this can be done in an efficient way [16]. Therefore, the Clifford+T set, which we obtain by expanding C with the T gate (for $n = 1$: $C_1 \cup \left\{T, -T, T^\dagger, -T^\dagger\right\}$), is a universal single qubit gate set. Together with the CNOT gate we then have a universal gate set for an arbitrary number of qubits. Note that the S gate as well as the Pauli gates could be constructed from T and H gates, so a truly minimal gate set would only need three gates $\{T, H, CNOT\}$. However, for convenience it is common practice to take the complete Clifford+T+CNOT set as the basis for constructing quantum circuits.

2.1.6 Quantum circuits

We can now build quantum circuits in order to describe quantum systems and the operations applied on it. Each wire represents a qubit and gates on a wire are applied to the qubit from left to right. Since all gates are unitary, quantum circuits are unitary as well and represent

a $2^n \times 2^n$ unitary matrix where n is the number of qubits. If not specified, the qubits on the inputs are initialized with state $|0\rangle$. Figure 2.2 shows an implementation of the *Toffoli gate* or $CCNOT$ which flips a qubit (q_3) depending on two control bits (q_1, q_2). The essential

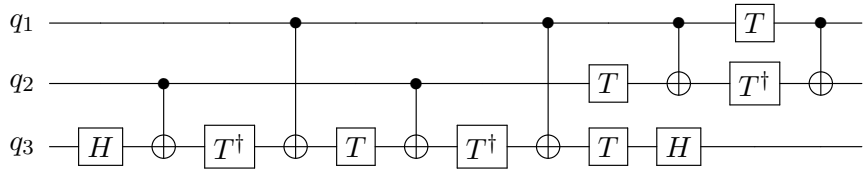


Figure 2.2: Implementation of the Toffoli gate using Clifford+T+CNOT gate set

part in this circuit are the rotations on q_3 by the CNOT, T and $T^\dagger$ gates. Note that if both upper wires are $|0\rangle$, q_3 is never flipped by a CNOT and the T and $T^\dagger$ gates have no effect on q_3 since together they correspond to two full rotations around the Z-axis. Also in the case that only one control qubit is $|0\rangle$, the rotations around the Z-axis sum up to 0. However, if both qubits are set the rotations sum up to π. Therefore, assuming the target qubit was initially in state $|0\rangle$, if q_1 and q_2 are set q_3 gets transformed to $|+\rangle$ via the first Hadamard gate and rotated to $|-\rangle$, otherwise it remains in $|+\rangle$. The last Hadamard gate then maps the final state of q_3 to $|0\rangle$ or $|1\rangle$.
Since the Toffoli gate is universal for classical computing, i.e., every Boolean function can be expressed as a combination of Toffoli gates, we can map any classical circuit to an equivalent quantum circuit. As the size of the circuit increases only by a constant factor, we can efficiently simulate every classical circuit on a quantum computer [12].

2.2 Quantum circuit optimization

As of 2021, quantum computers are still very limited in their power. The realization of quantum computations in practice is a difficult and expensive task which is prone to several errors. Moreover, there are severe restrictions on quantum circuits regarding their size. Thus, it is important to optimize quantum circuits as much as possible. We first outline the main hardware constraints of quantum computers and then show some basic strategies for optimizing quantum circuits.

2.2.1 Hardware limits of quantum circuits

Actual quantum computers in the real world are extremely challenging to build. It is both difficult to create an isolated quantum system that allows for qubit manipulation and measurement, and to maintain the isolated state long enough to perform computations. For the latter an essential metric is the *decoherence time* of a system which tells us how long we can keep a system quantum-mechanically coherent until it collapses, i.e., interacts with the environment. The decoherence time limits the total number of gates in a circuit since each operation on qubits takes a certain amount of time. For most physical implementations of quantum computers the decoherence time is in the range of micro- or nanoseconds and the maximum number of operations is between 10^3 and 10^6 [23, Figure 7.1].
In addition, quantum gates are error-prone, i.e., they may not always have the desired effect on the quantum system. Since different gates have different error probabilities, we can assign costs to each gate. A simple estimation of the gate cost can be obtained by the minimum

number of rotations necessary for its unitary operation. While Pauli gates only require a single rotation around the X-,Y- or Z-axis, the Hadamard gate requires at least two and the CNOT gate even five rotations [18]. In practice however, the cost of CNOT gates is even higher. Figure 2.3 compares the average gate error rates of Pauli X and CNOT gates of a state-of-the-art quantum computer. We can see that a CNOT gate is at least 10 times more error-prone than single Pauli qubit gates.

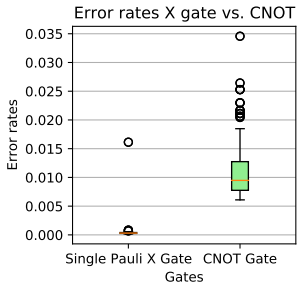


Figure 2.3: Average error rates of Pauli X and CNOT gate taken from the `ibmq_montreal` quantum computer during 23-26th of July 2021[1]

When estimating the overall cost of quantum circuits it makes sense to distinguish between the *single qubit gate count* and the *two-qubit gate count*, where the latter is weighed with a factor of 10 against the single qubit gate count.
However, this distinction neglects the fact that on hardware side non-Clifford gates are usually much more difficult to implement than Clifford gates. The best known way to implement non-Clifford gates requires a technique called *magic state distillation*[5]. For T gates is has been shown that the cost overhead of magic state distillation is about a constant factor higher than the cost overhead of CNOT gates ranging about 150 to 300 [24].
There are some mechanisms for automated error correction which can be built into a circuit. As they usually require additional helper qubits, we distinguish between logical qubits, which serve as a basis for operations on an abstract level, and physical qubits, which include all qubits involved in the system. However, most of these error correcting mechanisms cannot yet be implemented in practice. For instance, it is estimated that the surface code, an algorithm for correcting Clifford operations, requires at least over a 1000 times more physical qubits than logical ones [10]. As of 2021, the maximum number of qubits in a quantum computer is 65 [2], so using the suface code even on small circuits seems to be out of reach at the moment.
For some hardware realizations topology puts another limit on quantum circuits as two-qubit gates cannot be applied to any qubit pair. In Figure 2.4 an exemplary topology of an up to date quantum computer is shown. Here the qubits can only interact with their immediate neighbours and a CNOT gate from q_0 to q_2, for instance, is not possible. As shown in Figure 2.5, we could instead swap q_0 and q_1 using CNOT gates, apply a CNOT between q_1 and q_2, and swap q_0 and q_1 back in place. This variant, however, is far more expensive as it requires seven CNOT gates instead of just one.
All these points show that in quantum computing it is crucial to reduce circuit size as much as possible. Circuit size not only determines runtime and performance like in classical computing, but also whether a problem is executable at all and if we have a chance of

[1] https://quantum-computing.ibm.com/services?services=systems&system=ibmq_montreal
[2] https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/

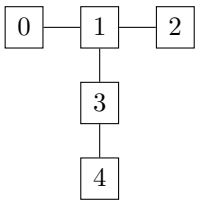


Figure 2.4: Topology of the `ibm_quito` quantum computer as of July 2021

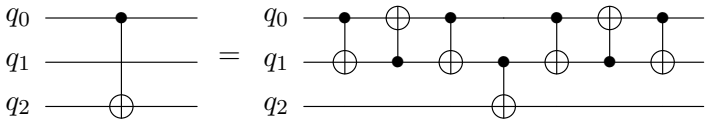


Figure 2.5: Decomposition of a CNOT gate from q_0 to q_2 into SWAP operations on q_0 and q_1 and CNOT on q_1 and q_2

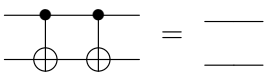
obtaining an error free computation. For Clifford+CNOT+T circuits, CNOT and T gates play a special role during optimization, as compared to Clifford gates they are the major contributors to the overall cost.

2.2.2 Optimization strategies

Quantum circuit optimization in general is QMA-hard, which means it is in a complexity class for quantum algorithms including the NP complexity class. So unless $P = NP$ we cannot find a general algorithm which returns a quantum circuit with minimal cost in polynomial time [22] [12]. Yet, we can use some strategies which reduce the total gate count in polynomial time, but do not always yield a circuit with minimal cost.

Gate cancellation

The simplest form of circuit optimization is gate cancellation. As we have already seen in the previous section, Pauli, Hadamard and CNOT gates are self-inverse and two of the same gate placed in a row cancel each other out:

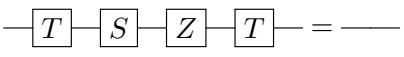


(2.25)

The same is possible for every gate where its inverse is placed directly after it

(2.26)

or, more generally put, every set of gates where the rotations sum up to 2π around an axis:

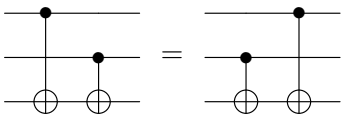


(2.27)

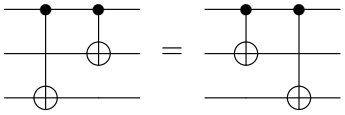
Here the rotations around the Z-axis sum up to $\pi/4 + \pi/2 + \pi + \pi/4 = 2\pi$ and we can cancel out all four gates.

Gate commutation

A second method which often works in combination with gate cancellation is gate commutation. For instance the CNOT gate commutes with itself:

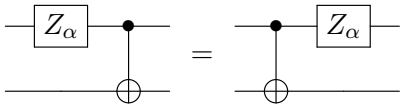


(2.28)

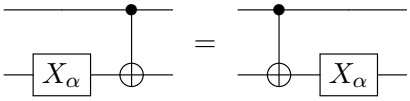


(2.29)

Moreover, the CNOT gate commutes with Z phase gates on the control wire and with X phase gates on the target wire:

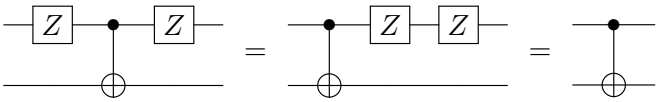


(2.30)



(2.31)

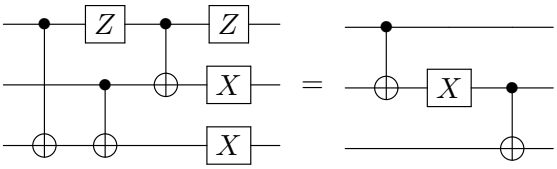
By combining gate commutation with gate cancellation we can optimize circuits as well:



(2.32)

Advanced optimizations

Using gate commutation and cancellation we can already reduce circuits significantly, but many optimization possibilities go far beyond these methods and may not be as intuitive. For instance, the following circuits represent the same unitary matrix [3]:



(2.33)

But apart from the two Z gates on the upper qubit which can be eliminated like in Equation 2.32, this requires more sophisticated optimization strategies. While this example also motivates using ZX-calculus for quantum circuit optimization we want two mention two more advanced concepts of optimization without ZX-calculus:

Template-based and *peephole optimizations*. In template-based optimization we have a known sets of gates called templates, which implement the identity operator. We then try to construct as many template matches as possible in a circuit by using commutation rules and eliminate them [20]. In peephole optimization, on the other hand, we have a set of known circuit optimizations for small circuits and we traverse a large circuit by replacing every occurrence of a known subcircuit by its optimized equivalent [25].

[3] This example was taken from a speech of Ross Duncan: https://www.youtube.com/watch?v=QbtSSqeYuFM#t=01h35m30s

2.3 ZX-calculus

The ZX-calculus is a graphical language for expressing linear maps on qubits. It has been invented as an alternative to the common Dirac-Notation to simplify reasoning about qubits. Relations in multi-qubit systems are often difficult to understand in Dirac notation, since the matrix size doubles per qubit and the complex number space quickly becomes confusing. ZX-calculus provides a way to represent quantum systems as 2-dimensional diagrams where nodes (*spiders*) and edges (*wires*) form an undirected graph. Since many important concepts in quantum mechanics follow very intuitively from this representation, it has been considered a promising tool for optimizing quantum circuits [8]. In the following section we give a short introduction to ZX-calculus. A more in-depth coverage of the topic can be found in [7] and [30].

2.3.1 Definitions

The basic elements of ZX-diagrams are spiders and wires. A spider can be either green (Z-Spider) or red (X-Spider) and has arbitrary many input and output wires:

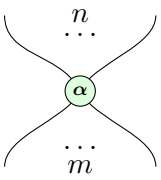


Figure 2.6: Z-Spider

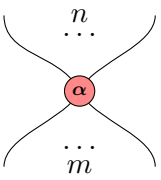


Figure 2.7: X-Spider

The variable α denotes an angle as a fraction modulo 2π and the "..." notation is translated as *0 or arbitrary many* wires, so the number of incoming and outgoing wires n, m does not have to be the same. We also distinguish between two types of wires: Normal and Hadamard wires. As the Hadamard wire itself can be decomposed into three spiders of phase $\frac{\pi}{2}$ connected by normal wires, the distinction is only for convenience.

Figure 2.8: Normal wire

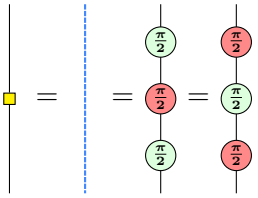
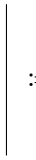


Figure 2.9: Hadamard wire

Furthermore, we have two types of compositions for any ZX-diagrams D_1, D_2:

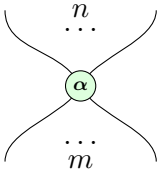
- **Spatial composition** $D_1 \otimes D_2$ consists of placing D_1 and D_2 side-by-side; D_2 to the right of D_1.
- **Sequential composition** $D_1 \circ D_2$ consists of placing D_1 on top of D_2 and connecting the outputs of D_1 to the inputs of D_2. This is only possible if the number of outputs of D_1 is equal to the number of inputs of D_2.

The *standard interpretation* associates a linear map in a Hilbert space to each ZX-diagram where the spatial composition corresponds to the Kronecker product and the sequential composition to matrix multiplication. Since diagrams are allowed to have different numbers of output and input wires, the standard interpretation of a diagram does not have to be unitary like in quantum circuits. For the basic elements the standard interpretation is defined as follows:

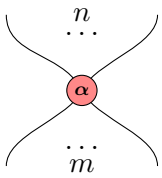


$$:= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$:= \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$



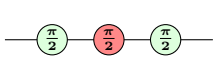
$$:= 2^m \left\{ \overbrace{\begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & e^{i\alpha} \end{pmatrix}}^{2^n} \right. = \overbrace{|0\dots0\rangle}^{m} \overbrace{\langle 0\dots0|}^{n} + e^{i\alpha} \overbrace{|1\dots1\rangle}^{m} \overbrace{\langle 1\dots1|}^{n}$$



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

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

Two diagrams are considered equal if their standard interpretation is the same up to a physically irrelevant global phase. For instance, the standard interpretation of the Hadamard decomposition is not exactly the same as the standard interpretation of the Hadamard wire:



$$:= \tfrac{1+i}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \neq \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} :=$$

However, since they only differ in the global phase, the measurement result remains the same regardless of which diagram we use for the Hadamard operation and we can treat the diagrams as equal.

2.3.2 Important spiders

The linear maps of Z and X spiders with only one input and output wire correspond to the matrices of the Z and X phase gate:


$$:= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}$$

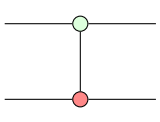
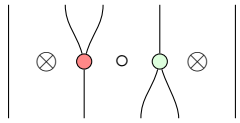
$$:= \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix} \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \tfrac{1}{2} \begin{pmatrix} 1+e^{i\alpha} & 1-e^{i\alpha} \\ 1-e^{i\alpha} & 1+e^{i\alpha} \end{pmatrix}$$

Although the X-Phase spider matrix may look unfamiliar, inserting π as angle α results exactly in the Z and X Pauli matrices:


$$:= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$:= \tfrac{1}{2} \begin{pmatrix} 1+e^{i\pi} & 1-e^{i\pi} \\ 1-e^{i\pi} & 1+e^{i\pi} \end{pmatrix} = \tfrac{1}{2} \begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

We can also construct a CNOT from a Z and X spider with angle 0:


$$=$$


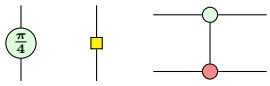
$$:= \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \right) \circ \left(\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right)$$

$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \circ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.34}$$

Apart from the global phase $\frac{1}{\sqrt{2}}$, this matrix corresponds to the CNOT matrix from Equation 2.22.

2.3.3 Universality

As mentioned in Section 2.1.5, we can decompose every unitary operation into a combination of CNOTs and single qubit unitaries. Since we are able to represent CNOT and arbitrary single qubit phase gates with Z and X spiders, the ZX-calculus is universal and we can build any quantum circuit in ZX-calculus. Moreover, we can build every non-unitary linear map on qubits as well, which makes ZX-diagrams a proper superset of quantum circuits. Similar to classical quantum computing we can construct the universal Clifford+T+CNOT set from the following spiders:

 (2.35)

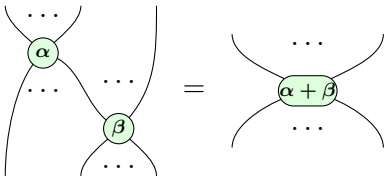
In general, we speak of Clifford diagrams/spiders if the phases of each spider are multiples of $\frac{\pi}{2}$ and of Clifford+T diagrams/spiders if all phases are multiples of $\frac{\pi}{4}$.

2.3.4 Rules

The ZX-calculus comes with a series of rules allowing to transform ZX-diagrams. The rules are sound, which means the underlying linear map of the standard interpretation is not changed. All rules can be applied in both directions and also hold with inverted colors (i.e. if the colors red and green are swapped). In addition for ZX-diagrams *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.

Fusion rule (f)

One of the most important rules is the spider fusion rule:



This rule allows us to merge two connected spiders of the same color into a single spider where the phases sum up. As α and β correspond to phases, we take the addition modulo 2π. The fusion rule also allows to split a spider into any number of other spiders of the same color where the sum of the phases needs to be equal to the original phase (modulo 2π). In Figure 2.10 some examples of the fusion rule are shown.

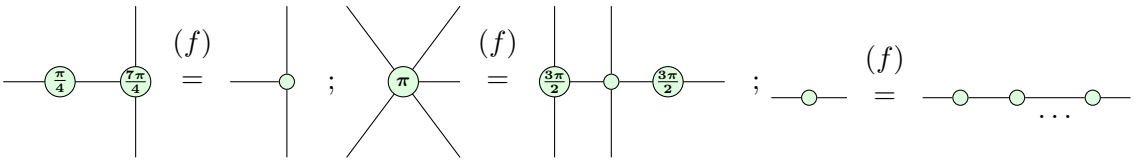
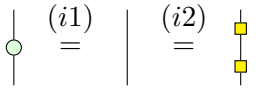


Figure 2.10: Examples of the spider fusion rule

Identity rules (i1, i2)

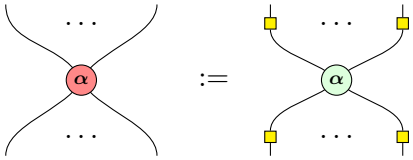
Furthermore, we can remove an empty spider with two connected wires, as well as two Hadamard wires in a row:



These rules are familiar from classical quantum computing, since a rotation around an axis by zero has no effect on a qubit and two Hadamard gates cancel each other out.

Hadamard rule (h)

The Hadamard rule allows us to swap the color of any spider by adding Hadamard wires to each input and output wire:

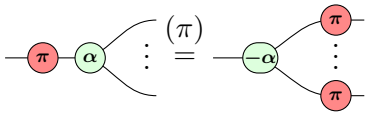


This also corresponds to the definition of the red spider in Section 2.3.1. If an input or output wire is already a Hadamard wire it gets flipped back to a normal wire due to $(i2)$.

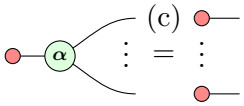
Π rule, Copy rule and Bialgebra rule

Further basic rules of the ZX-calculus are:

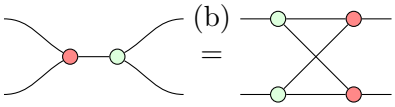
1. The π rule (π):



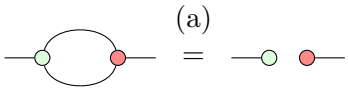
2. The copy rule (c):



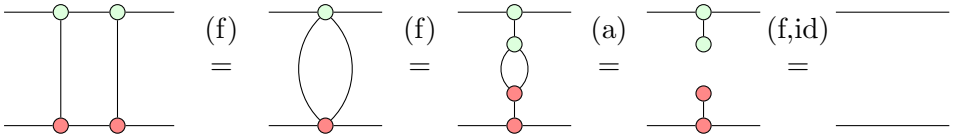
3. The bialgebra rule:



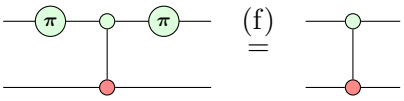
With this basic set of rules we can derive other rules or simplify quantum circuits in a purely graphical way. For instance, we can derive the antipode rule



from the basis set [9], which we can use to show the cancellation of two CNOTs in a row:



Other simplifications from the previous section, like the gate commutation and cancellation from Equation 2.32, become very simple due to spider fusion:



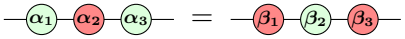
.

2.3.5 Completeness

In [3] it has been shown that this basic set of rules is sufficient to prove completeness for Clifford diagrams. Using the rules we can transform any ZX-diagram into every other ZX-diagram with the same standard interpretation. Since Clifford diagrams are not sufficient to represent all quantum circuits, much effort has been put into finding a rule set which is also complete for Clifford+T resp. all ZX-diagrams. However, all rule sets found so far

are significantly more complicated, as the new rules either require many spiders in a special arrangement, or have non-linear dependencies [13][29]. For instance, in [29] a complete rule set for all ZX-diagrams is presented which only adds a single rule called *Euler rule* to our basis rule set from the previous section:

$$(EU)$$



Although this rule looks simple, the phases $\beta_1, \beta_2, \beta_3$ are dependant on $\alpha_1, \alpha_2, \alpha_3$ in a non-linear way and require complex calculations. Nevertheless, this rule has some interesting characteristics which we will discuss in Section 4.1. For now, it is important to note that the calculus is *sound*, *complete* and *universal* so everything provable in Hilbert space using Dirac-Notation is also provable diagrammatically in ZX-calculus using spiders and wires.

3 Related work on ZX-diagram simplification

In the last years, ZX-calculus has been used for a variety of quantum circuit optimization approaches [14][15][9]. Using ZX-diagrams for optimization has some great advantages:
First, ZX-diagrams are not limited to the classical circuit structure. As we will see in this chapter, spiders and wires do not necessarily correspond to classical qubit gates and wires anymore and using the ZX-rules we can build abstract spider constructs which have no direct circuit equivalent.
Second, the ZX-rules allow simple reasoning about very complex circuit optimizations and we can deduce rules which are not applicable to normal circuits in the same form.
Third, we can abstract a circuit to the point where we can describe it using only two different generators: green and red spiders. As a consequence, in a ZX-diagram it does not matter whether spiders belong to Z, Hadamard or CNOT gates anymore. The only relevant features are color, angle and connectivity. Therefore, rules like the spider fusion or Hadamard rule are extremely powerful, as they are applicable to any spider at any time.
The basic procedure for optimizing quantum circuits via ZX-calculus is always as follows:

1. Transform the quantum circuit to a ZX-diagram.

2. Simplify the ZX-diagram by applying ZX-rules.

3. Extract a quantum circuit from the ZX-diagram.

When using ZX-diagrams for circuit optimization we speak of *simplification* of ZX-diagrams rather than optimization. While an optimization step reduces the number of gates in a quantum circuit, a simplification step only reduces the number of spiders or wires in a ZX-diagram, which does not always reduce the number of gates of the underlying quantum circuit.

Simplifying ZX-diagrams is by no means a trivial task. Rules can be applied in both directions and there is no general way to tell which rule application at which position minimizes the circuit the most. In some cases it makes sense to apply a rule in one direction or the other, sometimes it is better not to apply a rule at all even if we could. Finding the optimial circuit by calculating all possible rule applications and their outcomes is not feasible, since the unfusion of a single spider already has theoretically infinite possibilities. Therefore, simplifying ZX-diagrams seems to be an optimization problem with an infinitely large search space and in most cases we can only approximate the optimal solution.
In addition, it is also important to mention that step 3, i.e. the extraction of quantum circuits from ZX-diagrams, is also a non-trivial task. ZX-rules can modify diagrams in such a way that the circuit structure is lost. In theory, after step 2 we still have a unitary ZX-diagram whose linear map can be represented in a quantum circuit. In practice, though, as we will see in Section 3.3, there are some cases where we cannot extract such a circuit from a given ZX-diagram. This also plays a major role in ZX-diagram simplification, since we have to restrict the rule applications to those preserving the possibility of extraction. In this chapter we will give a general overview of the existing approaches for optimizing

quantum circuits using ZX-calculus. This field of research is very new, as the first paper on this topic appeared in 2018 [8], and there are still few papers on this topic. Most approaches presented in papers so far are already implemented in the PyZX library [1], an open source library programmed in Python for creating, visualizing and rewriting ZX-diagrams [14]. In this chapter we summarize and discuss the results of three papers [9][15][4]. First, we give an overview of the core diagram simplification strategies, then we look at the circuit extraction problem and last we discuss the actual PyZX implementation and analyze the effectivity of the simplification strategies for quantum circuit optimization.

3.1 Graph-based simplifications

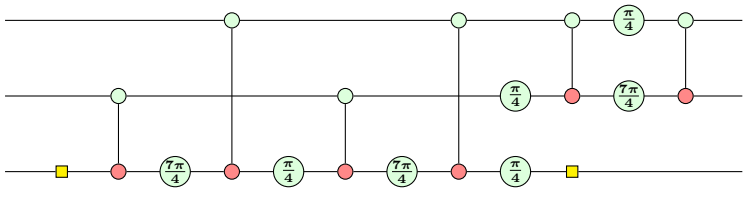
Instead of using ZX-rules in both directions, most simplification approaches only allow rules that eliminate at least one spider. This way, each simplification step makes the diagram smaller and the process is sure to terminate. Although the diagram may not be simplified in the best possible way, extracting a circuit from a diagram which contains fewer spiders usually results in fewer gates as well. The core of most simplification strategies are two rules named *local complementation* and *pivoting* coming from graph theory. However, these rules only work on diagrams which are in a normal form called *graph-like*. In this section we first introduce the graph-like normal form before showing the two graph-theoretic rewrite rules and their ZX-calculus equivalent. Last we present an algorithm for simplifying diagrams based on these rules.

3.1.1 Graph-like diagrams

ZX-diagrams are *graph-like* when:

1. All spiders are Z-spiders.
2. All connections between spiders are Hadamard wires.
3. There are no parallel wires or loops.
4. Every input and output is connected to a spider and every spider is connected to at most one input or output. These connections are the only non-Hadamard wires in a graph-like diagram.

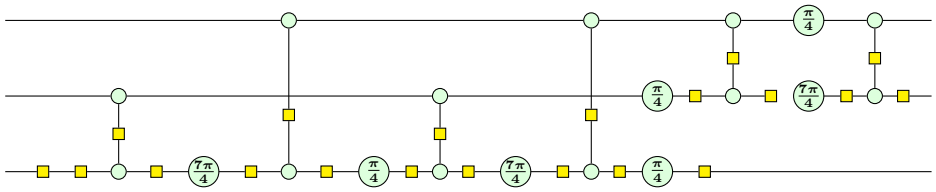
Every ZX-diagram can be transformed into an equal graph-like ZX-diagram using the rules from Section 2.3.4. We demonstrate this by using the Toffoli circuit from Figure 2.2, which has the following representation in ZX-calculus:



(3.1)

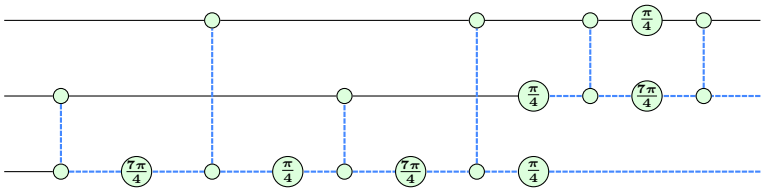
[1] https://github.com/Quantomatic/pyzx

First we turn every X-spider to a Z-spider using the Hadamard rule:



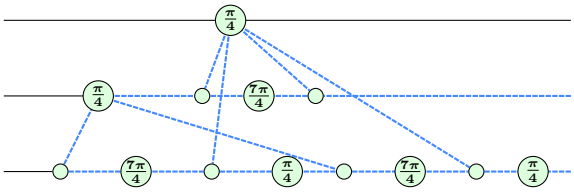
(3.2)

Then we use the identity rules to remove two Hadamard wires in a row and, for convenience, we switch to representing Hadamard wires as dashed blue lines:



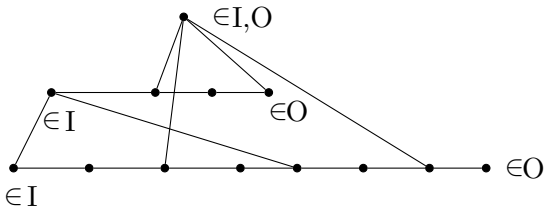
(3.3)

Last we maximally fuse every spider, thus eliminating every non-Hadamard wire between spiders:



(3.4)

Since there are no loops or parallel wires we are done, otherwise we could remove them using additional rules as shown in [9, p.7].
The advantage of graph-like diagrams is that we can capture the structure of a diagram in an *open graph*. An open graph is defined as the triple (G, I, O), where $G = (V, E)$ is an undirected graph with V as the set of vertices and E as the set of edges. The sets $I, O \in V$ represent the inputs and outputs of the graph. We distinguish between *internal vertices* $\{v \in V | v \notin \{I \cup O\}\}$ and *boundary vertices* $\{v \in V | v \in \{I \cup O\}\}$. In the open graph of a graph-like ZX-diagram, the spiders correspond to vertices and the Hadamard wires to edges. I and O are the subset of spiders corresponding to the inputs and outputs of the diagram. For instance, the open graph for 3.4 is:



(3.5)

3.1.2 Graph states

A special case of graph-like diagrams are graph-state diagrams where all spiders are required to have an angle of zero. These diagrams correspond to arbitrary combinations of Hadamard and CZ gates which have been studied in context of quantum error correction [28]. For graph states an important property was discovered known as *van Nest's theorem* which states that two graph states are equal up to a unitary Clifford matrix if and only if we

can transform the underlying graphs into another using sequences of *local complementation* steps [28]. As a consequence for graph-state diagrams, local complementation followed by updating the phases of some spiders according to the unitary Clifford matrix does not change the standard interpretation. Since the unfusion rule allows us to convert any subgraph of a graph-like diagram to a graph state, local complementation plays an important role in diagram simplificiation. We will first show how these transformations work on an abstract level before showing the corresponding rules in ZX-calculus.

3.1.3 Local complementation

The local complementation $\star$ of an undirected graph $G = (V, E)$ about a vertex u is defined as follows:

$$G \star u \coloneqq (V, E \,\Delta\, \{(a, b) \,|\, (a, u), (b, u) \in E, a \neq b\}), \tag{3.6}$$

where Δ is the symmetric set difference: $A \,\Delta\, B \coloneqq (A \cup B) \setminus (A \cap B)$. The resulting graph $G \star u$ is the same as G, except that for all neighbours $N(u)$ of u holds:

1. If two vertices $w, w' \in N(u)$ are connected in G, i.e $(w, w') \in E$, they are no longer connected in $G \star u$, i.e $(w, w') \notin E$.
2. If two vertices $w, w' \in N(u)$ are *not* connected in G, i.e $(w, w') \notin E$, they are connected in $G \star u$, i.e $(w, w') \in E$.

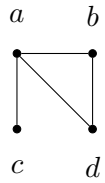
So two neighbours of u are connected in $G \star u$ if and only if they are not connected in G. For instance, consider the graph

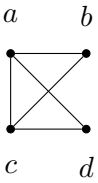
$$G = (V, E), V = \{a, b, c, d\}, E = \{(a, b), (a, c), (a, d), (b, d)\}. \tag{3.7}$$

Applying local complementation on a results in

$$G \star a = (V', E'), V' = V, E' = \{(a, b), (a, c), (a, d), (b, c), (c, d)\}, \tag{3.8}$$

or graphically:

$G =$ 

$(G \star a) =$ 

(3.9)

3.1.4 Pivoting

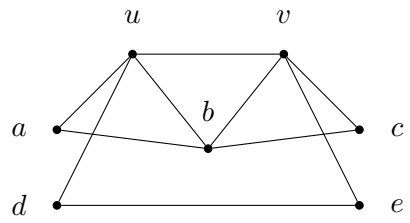
Pivoting is a rewrite around an edge $(u, v) \in E$ and is built up from three applications of local complementation:

$$G \wedge uv \coloneqq G \star u \star v \star u \tag{3.10}$$

For calculating the new connectivity of the graph we consider three disjoint sets:

- $A \coloneqq N(u) \cap N(v)$, i.e., all vertices connected to u and v.
- $B \coloneqq N(u) \backslash N(v)$, i.e., all vertices connected to u and not to v.
- $C \coloneqq N(v) \backslash N(u)$, i.e., all vertices connected to v and not to u.

In a pivoted graph $G \wedge uv$, all vertices between those sets are connected if and only if they are not connected in G. Connections between vertices of the same set are not modified. As an example consider the following graph

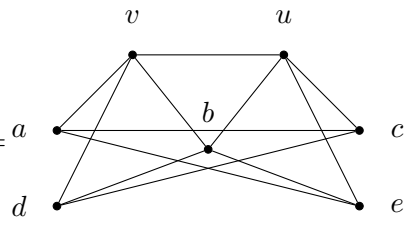
$G :=$ 

(3.11)

where we apply pivoting on the edge (u, v). The three sets as defined above are:

$$A := \{b\}, B := \{a, d\}, C := \{c, e\} \tag{3.12}$$

In the pivoted graph $G \wedge uv$ the connections between all three sets are flipped:

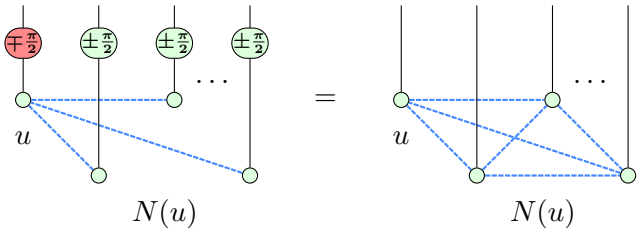
$G \wedge uv :=$ 

(3.13)

So for instance a and b are not connected in $G \wedge uv$ because they are connected in G and are in different sets ($a \in B, b \in A$), whereas a and d are not connected in $G \wedge uv$ because even though they are not connected in G they are in the same set ($a, d \in B$).

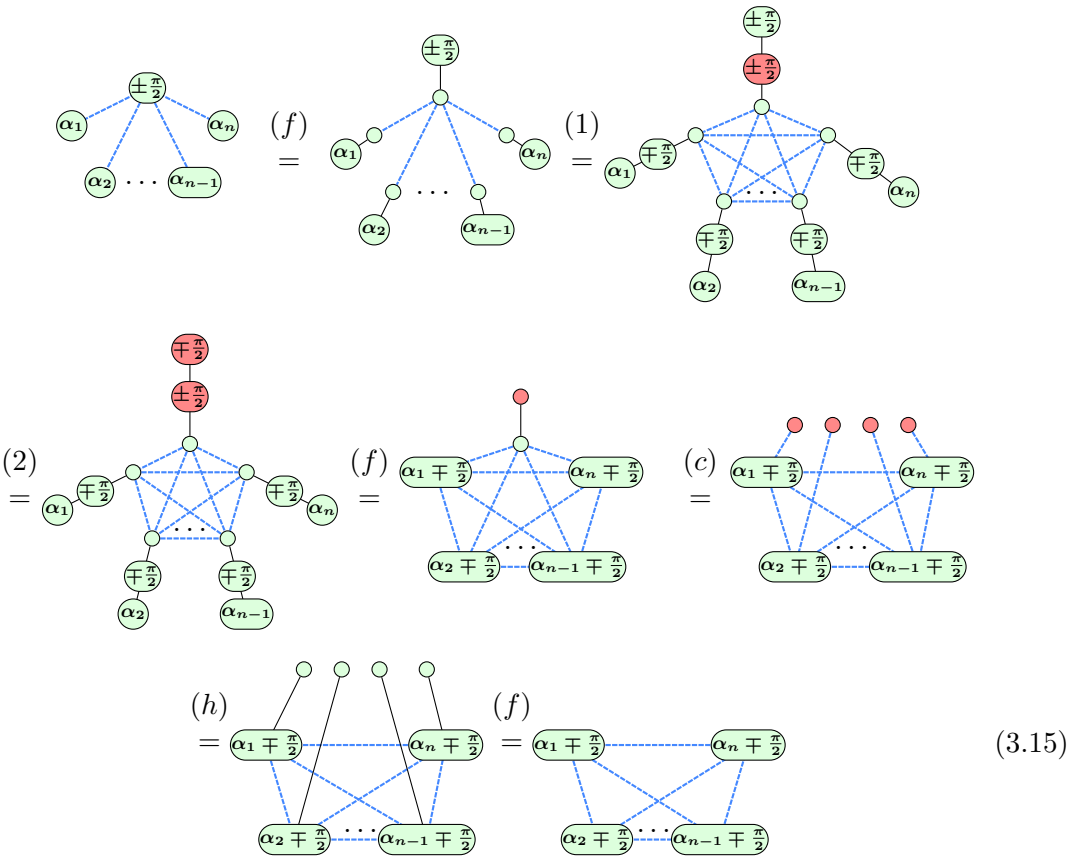
3.1.5 Local complementation and pivoting in ZX-diagrams

In ZX-calculus the following holds:



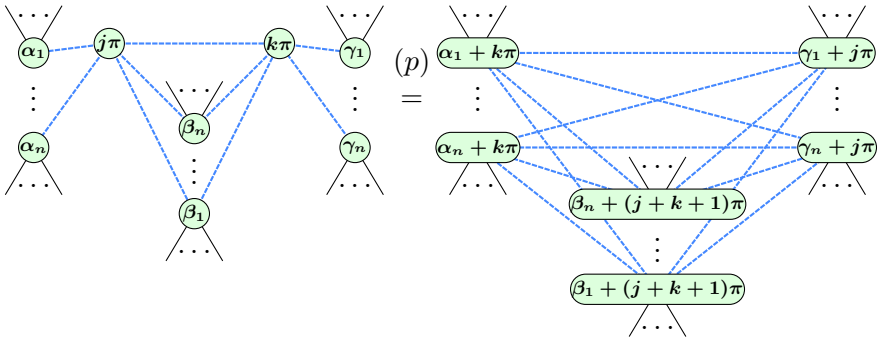
(3.14)

This means for a spider u and its neighbours $N(u)$, an adjacent red $\pi/2$ spider on u and adjacent green $\pi/2$ spiders on all neighbouring spiders are equal to a local complementation on u. As the "..." notation denotes 0 or arbitrary many wires, it is not relevant how many neighbours there are. 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. However, for spiders with phase $\pm\pi/2$, the application is special, since we can remove the spider u subsequently:



(3.15)

In (1) we use Equation 3.14 and in (2) the property that a $\pi/2$ spider with only one wire is equal to a $\pi/2$ spider with inverted color and phase [9, p.29].
For pivoting a similar rule can be derived:



(3.16)

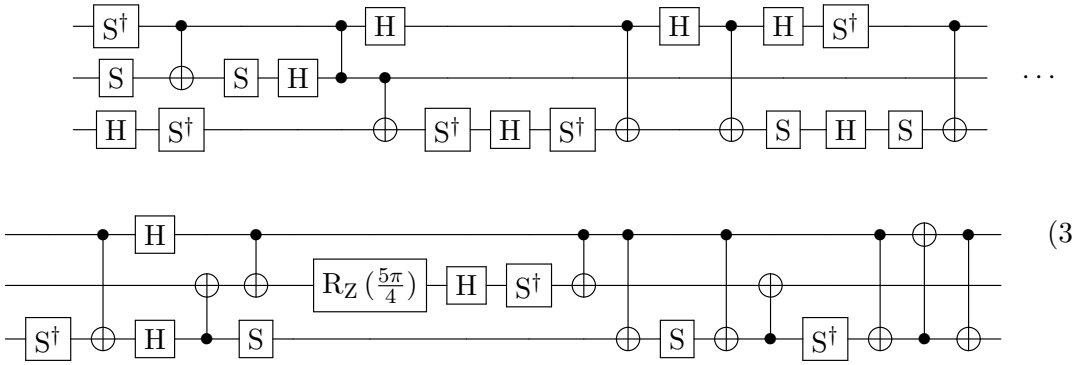
Here we can remove two adjacent spiders with a phase of 0 or π. So these two rules correspond to the graph theoretic definition of $G \star u$ and $G \wedge uv$ with additionally removing the initial spiders u resp. u, v.

3.1.6 Clifford simplification algorithm

As mentioned in Section 2.3.3, the spider types representing the Clifford subset of quantum gates are exactly all spiders with a phase of $k * \pi/2$, i.e. those spiders which can be removed using local complementation and pivoting. We can construct a simplification algorithm for eliminating Clifford spiders in ZX-diagrams which works as follows:

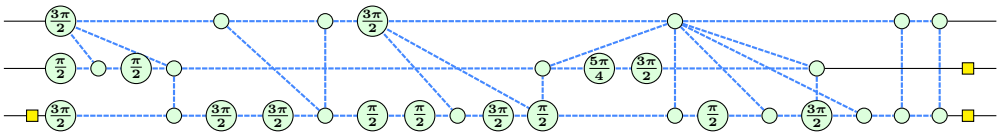
1. Transform the diagram to a graph-like diagram.
2. Eliminate empty spiders with two wires using the identiy rule and subsequently fuse the adjacent spiders in order to maintain a graph-like diagram.
3. Apply local complementation on every spider of phase $\pm\pi/2$ and pivoting on every pair of spiders of phase 0 or π as long as possible.
4. If step 3 modified the diagram go back to step 2, else we are done.

The simplified ZX-diagram then 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. As an example consider the following randomly generated quantum circuit:



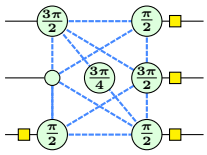
(3.17)

We can transform the circuit into a graph-like ZX-diagram



(3.18)

and apply the algorithm as described, which results in a much simpler diagram:



(3.19)

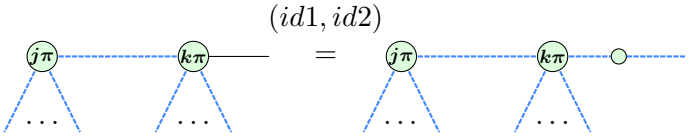
The algorithm indeed eliminated almost all Clifford spiders except those at the diagram boundaries and the only interior spider left is a non-Clifford spider with phase $3\pi/4$.

3.2 Advanced simplifications

Simplifying diagrams using the algorithm from 3.1.6 can already reduce diagrams drastically. However, there are two other strategies which can be applied additionally during the simplification process which we want to mention. First, we discuss the *pivot boundary simplification*, which aims to further reduce interior Clifford spiders with phase 0 or π, and second, we discuss the *phase gadget simplification*, which aims to reduce non-Clifford spiders as well.

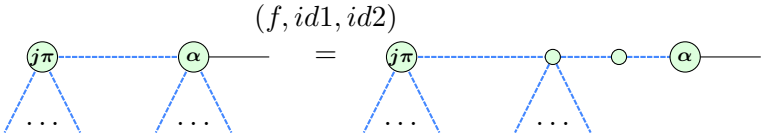
3.2.1 Pivot boundary simplification

The pivot rule can only be applied on two connected *interior* spiders with phase 0 or π. If the two spiders have the correct phase, but one of them is at the boundary of the diagram, i.e. has one non-Hadamard wire, the rule application is not possible. However, using the identity rules we can replace the non-Hadamard wire with an empty spider surrounded by two Hadamard wires:



(3.20)

Using this insertion which does not change standard interpretation, we transform the original boundary spider to an interior spider where we can apply the pivot rule. This can be generalized to remove a spider with phase 0 or π adjacent to a boundary spider with an *arbitrary* phase:

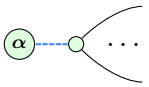


(3.21)

The problem with this method is that it does not decrease the number of spiders, since we add two spiders to the diagram and subsequently remove two spiders using the pivot rule. Applying this procedure again on the two new spiders yields the same diagram as the original, so we get stuck in an infinite loop and the simplification process does not terminate. In Section 3.4 we will show how this issue is resolved in the PyZX library.

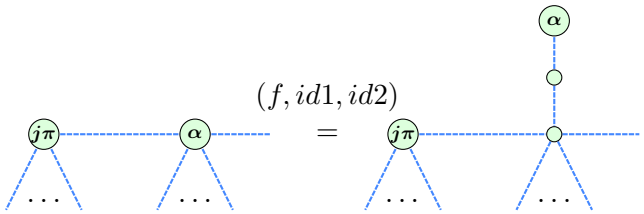
3.2.2 Phase gadget simplifications

A phase gadget is a parametrized spider with only one wire connected via Hadamard edge to a phaseless spider:



(3.22)

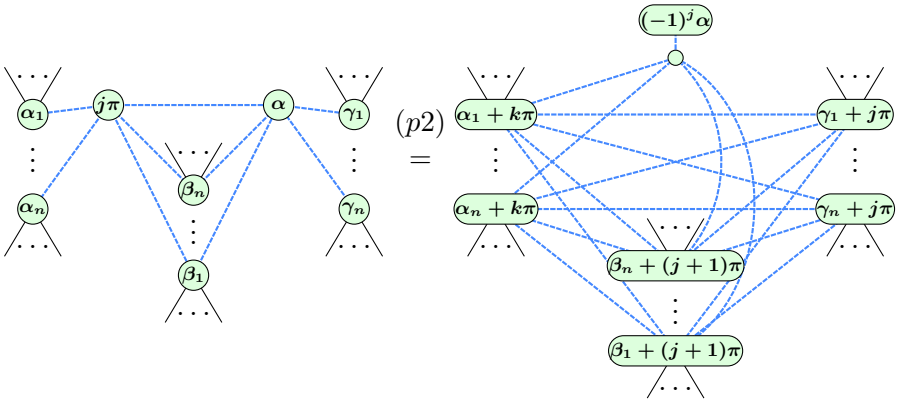
Phase gadgets have been studied in the context of diagram simplification [15], since they allow us to pull any "unwanted" phase of a spider into a phase gadget. For instance, if we have a single spider with phase 0 or π which is only connected to non-Clifford spiders, we modify one of these spiders as follows:



(3.23)

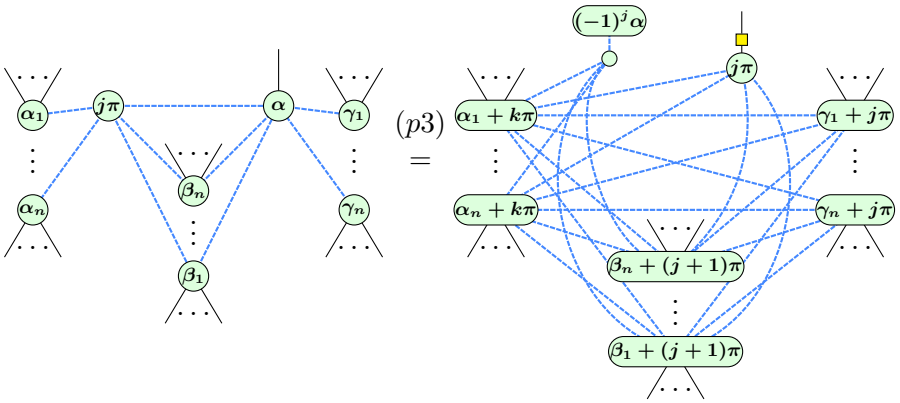
The non-Clifford spider is now a spider with phase 0 connected to a phase gadget with phase α and we can apply the pivot rule. Combining this step with the original pivoting rule we

get two new rules:



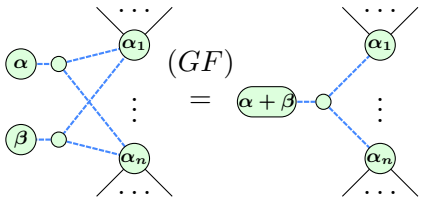
(3.24)

and



(3.25)

where the former is used for interior spiders and the latter is used if one spider is a boundary spider. Furthermore, two phase gadgets connected to the same set of spiders can be fused using the *gadget fusion* rule:



(3.26)

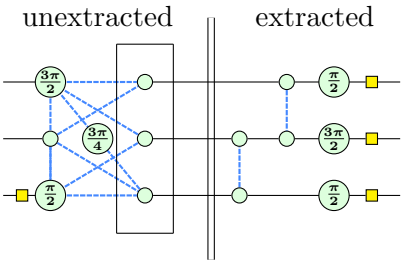
Adding 3.24, 3.25 and 3.26 to the simplification algorithm we can both remove every remaining interior Clifford spider and remove some non-Clifford spiders, since the gadget fusion may, for instance, add two $\pi/4$ spiders up to a $\pi/2$ Clifford spider. We can either use gadget fusion in the normal simplification algorithm, where phase gadgets are generated by the modified pivoting rule, or use gadget fusion outside of the simplification procedure on circuit-like diagrams. The latter is called *phase teleportation* and has the advantage that it eliminates some non-Clifford spiders but does not change the original circuit structure of the diagram, since no local complementation or pivoting is used in the procedure.

3.3 Circuit extraction

Converting graph-like ZX-diagrams back to quantum circuits is not always possible, because ZX-diagrams have a more open structure: While circuits always have a clear flow of time and we can easily see which gates operate on which qubits, for an interior spider we usually cannot distinguish whether it comes before or after another spider in the time flow and to which qubit it belongs. However, there is an extraction algorithm introduced in [9] which is able to extract a circuit provided the diagram meets some conditions. We first show how the extraction algorithm works using the simplified diagram from 3.19 as an example. Then we point out the limitations of this algorithm and show which conditions are required for a diagram in order to be extractable.

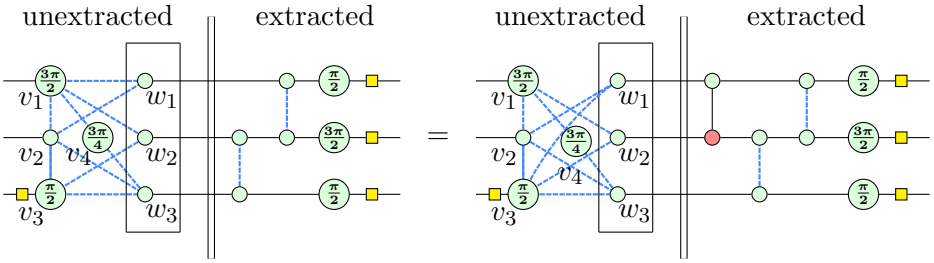
3.3.1 Basic extraction algorithm

The basic approach for extracting a circuit from a diagram is to split the diagram into an unextracted and an extracted part separated by a set of spiders named *frontier*. At the beginning the extracted part is empty and we initialize the frontier set with the outputs, of which we know each of them represents exactly one qubit. The process of extraction can be thought of as pulling the spiders through the frontier to the extracted part. Since we reconstruct time flow of a circuit, spiders left of the frontier are referred to as spiders in the past of the frontier and extracted spiders on the right of the frontier are referred to as being in the future. For spiders which only have two neighbours, i.e. one unextracted spider in the past and one extracted spider in the future, we can simply extract green spiders as R_z gates and Hadamard wires as Hadamard gates. Furthermore, we can extract connections between two frontier spiders as CZ gates. For the simplified diagram from 3.19, applying these basic extraction steps leads to the following diagram:



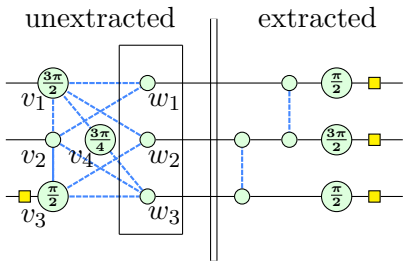
(3.27)

The doubled vertical line denotes the separation of the extracted and unextracted part of the diagram and the box around the rightmost unextracted spiders denotes the frontier set. However, as soon as the frontier only has spiders which are connected to multiple spiders in the past, we need to "add" rows using CNOT gates. In [9] it is shown that for two frontier spiders w and w', adding a CNOT with control wire w' and target wire w to the extracted part has the effect of adding the Hadamard wires to the past neighbours of w' to those of w. For instance:



(3.28)

Here we added the Hadamard wires to the past neighbours of w_2 to those of w_1 by adding a CNOT to the extracted part from w_1 to w_2. Previously w_1 was connected to v_1 and v_2, and w_2 was connected to v_1 and v_3. The new connections for w_1 sum up as $2 * v_1, v_2, v_3$. Since the two Hadamard wires between w_1 and v_1 cancel each other out like in the Antipode rule from Section 2.3.4, w_1 is only connected to v_2, v_3 at the end. The crucial part of the extraction procedure is to add rows via CNOTs until at least one spider in the frontier is only connected to a single spider in the past, which we can again extract as R_z gate. For this task it is useful to write the connections between frontier spiders and their unextracted neighbours in a *biadjacency* matrix. When annotating the spiders of diagram 3.27 as follows:



(3.29)

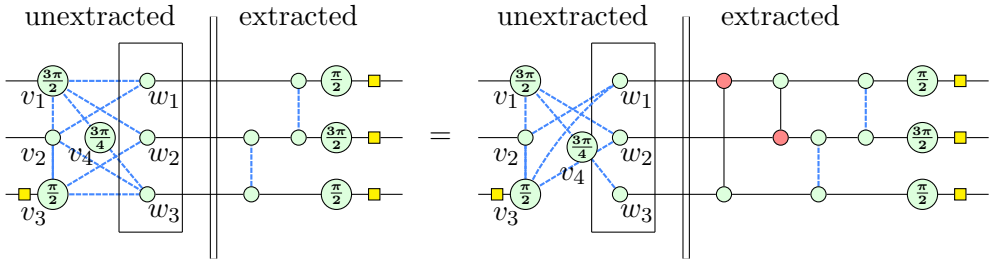
the biadjacency matrix is:

$$\begin{array}{c} \\ w_1 \\ w_2 \\ w_3 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \end{array}. \tag{3.30}$$

By using matrix row operations we can eliminate connections until at least one row contains only a single entry with 1:

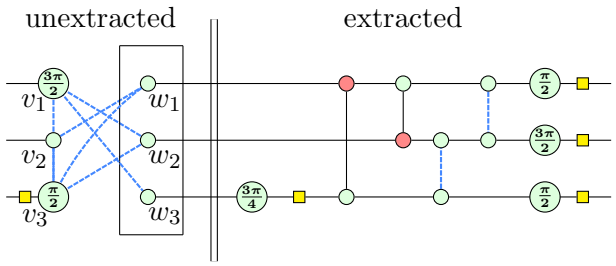
$$w_1 \mathrel{+}= w_2 = \begin{array}{c} \\ w_1 \\ w_2 \\ w_3 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \end{array} \Rightarrow w_3 \mathrel{+}= w_1 = \begin{array}{c} \\ w_1 \\ w_2 \\ w_3 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \tag{3.31}$$

Applying these row operations on the ZX-diagram puts two new CNOTs on the extracted circuit:



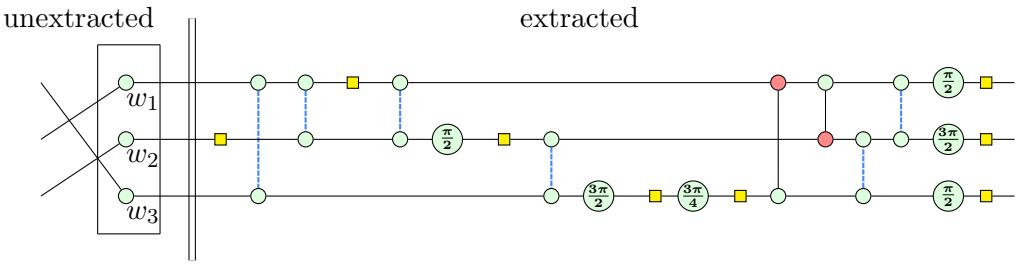
(3.32)

Now w_3 has only one neighbour in the past, i.e. the interior spider with phase $3\pi/4$, which can be pulled through the frontier as an R_z gate:



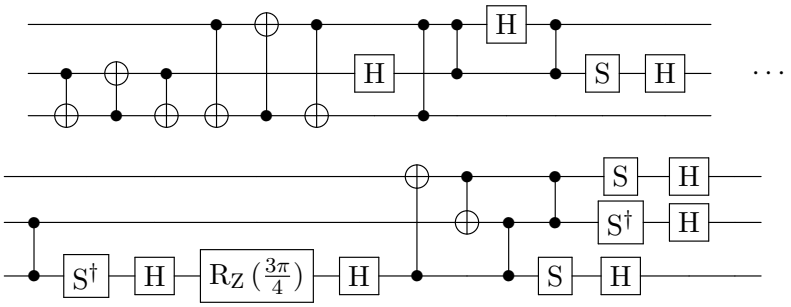
(3.33)

These steps are repeated until only input spiders are left in the frontier:



(3.34)

As a last step the input wires need to be swapped to match the correct output wires. This can be done with two SWAP gates on the first and third wire and on the third and second wire. The extracted part then is in a circuit like structure and corresponds to the following classical quantum circuit:

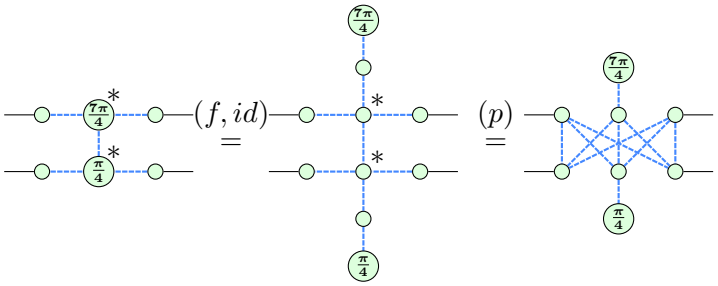


(3.35)

Compared to the original circuit from 3.1.6 this circuit contains nine fewer single qubit gates and two fewer two-qubit gates.

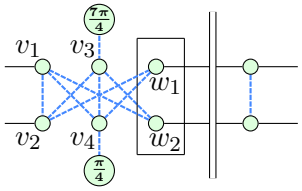
3.3.2 Extended algorithm with measurement planes

The basic algorithm works for all diagrams simplified with the algorithm from Section 3.1.6, however, when using phase gadgets for simplification the algorithm sometimes fails. Consider the following example:



(3.36)

Similar to Equation 3.23 we can "pull" the phases of the spiders marked with $*$ into phase gadgets in order to apply the pivot rule on them. When we start extracting the diagram we can still extract a CZ gate, but after that we are not able to proceed:

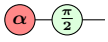


(3.37)

Both spiders in the frontier have two neighbours in the past and the biadjacency matrix is

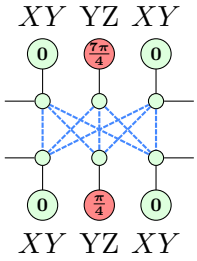
$$\begin{array}{c} \\ w_1 \\ w_2 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array}. \tag{3.38}$$

Finding a sequence of row operations after which one row contains only a single 1 is not possible for this matrix and the algorithm from the previous section would fail.
In [4] an extended algorithm for extraction has been presented originating from *measurement based quantum computation* or short *MBQC*. In MBQC, calculations are not performed with unitary gates, but by measuring a set of entangled qubits in a certain order. Measurements are usually parametrized by an angle between 0 and 2π and are restricted to either be done in the XY, XZ or YZ plane of the Bloch sphere. By measuring qubits in a specific order, the remaining unmeasured qubits can be modified in a way similar to classical quantum circuits [26]. In ZX-calculus the different measurement planes of MBQC-circuits correspond to different spider types:

measurement plane	XY	XZ	YZ
measurement effect			

(3.39)

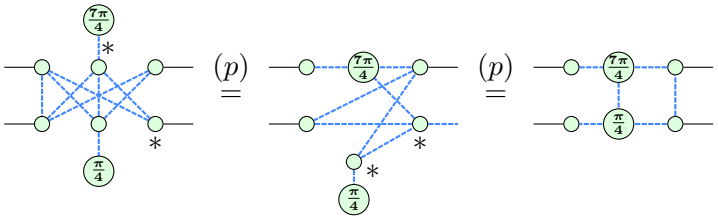
For graph-like ZX-diagrams the authors distinguish between normal spiders measured in XY plane, and phase gadgets measured in YZ plane. The unextractable graph from 3.37 would be written as follows in MBQC:



(3.40)

The spiders with phase $7\pi/4$ and $\pi/4$ are now *measurement effects* and do not count as individual spiders anymore. Instead, the MBQC graph only has six spiders left with an empty phase: Four of them measured in XY plane and two in YZ plane. The authors of [4] have developed an extraction algorithm which extracts a circuit containing spiders in all three measurement planes which works just like the basic extraction algorithm, except that spiders measured in XZ and YZ plane are treated differently. More precisely, a phase gadget, i.e., a spider in YZ plane, can be converted into a spider in XY plane by applying

a pivot on the root of the phase gadget and an arbitrary connected frontier spider. This converts the phase gadget back to a XY spider and the diagram can be extracted with the basic extraction procedure:



(3.41)

Similar, a spider in XZ plane can be converted into an XY spider by fusing the $\pi/2$ phase with the root spider and applying local complementation on it.

3.3.3 The gflow property

So far we have explained that diagrams are extractable if the biadjacency matrix between frontier spiders and their past neighbours can always be transformed via row operations into a matrix with at least one row containing a single 1 and vice versa. At graph theory level, there is another formulation equivalent to this property called *generalized flow* or *gflow*. Each diagram with gflow is extractable with the extraction algorithm from 3.3.2[4].

Definition of gflow

The gflow property is defined on *labelled open graphs*, which are like open graphs from Section 3.1.1 but extended with a function $\lambda(v) \in \{XY, XZ, YZ\}, v \in V$, which assigns a measurement plane to each vertex. A labelled open graph $(G, I, O, \lambda), G = (V, E)$ has gflow if there exists a map $g : \overline{O} \to \mathcal{P}(\overline{I})$, i.e. a map from all non-outputs to the power set of all non-inputs, and a partial order $\prec$ such that for all $v \in \overline{O}$:

(g1) If $w \in g(v)$ and $v \neq w$, then $v \prec w$.

(g2) If $w \in Odd(g(v))$ and $v \neq w$, then $v \prec w$.

(g3) If $\lambda(v) = XY$, then $v \notin g(v)$ and $v \in Odd(g(v))$.

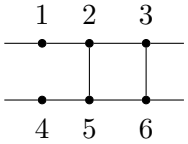
(g4) If $\lambda(v) = XZ$, then $v \in g(v)$ and $v \in Odd(g(v))$.

(g5) If $\lambda(v) = YZ$, then $v \in g(v)$ and $v \notin Odd(g(v))$.

$Odd(g(K))$ is defined as the *odd neighbourhood* of a set of vertices $K \subseteq V$:

$$Odd(K) = \{u \in V : |N(u) \cap K| \equiv 1 \bmod 2\}.$$

Simply put, the odd neighbourhood of a set of vertices K are all adjacent vertices which are connected to an odd number of vertices in K.
For a diagram which admits gflow there are usually different maps and orderings, but as long as we can find any, the diagram is extractable. For instance, the labelled open graph for (3.41)



$\lambda(1 \dots 6) = XY$ (3.42)

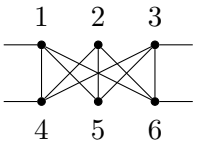
admits gflow with the map

v	1	2	4	5
$g(v)$	$\{2\}$	$\{3\}$	$\{5\}$	$\{6\}$
$Odd\,(g(v))$	$\{1,3,5\}$	$\{2,6\}$	$\{2,4,6\}$	$\{3,5\}$

and the ordering

$$1,4 \prec 2,5 \prec 3,6.$$

However, the labelled open graph for (3.40) only admits gflow if the middle vertices are measured in YZ plane:



$$\lambda(1,3,4,6) = XY \qquad \lambda(2,5) = YZ \tag{3.43}$$

Here the conditions are fulfilled with the following map:

v	1	2	4	5
$g(v)$	$\{5\}$	$\{2,3\}$	$\{2\}$	$\{5,6\}$
$Odd\,(g(v))$	$\{1,2,3\}$	$\{\}$	$\{4,5,6\}$	$\{\}$

and the ordering

$$1,4 \prec 2,5 \prec 3,6.$$

If all vertices were measured in XY plane instead, there would be no solution. This can be proven by contradiction:

Proof. By looking at the odd neighbourhoods of the graph (3.43) one can observe that for each possible combination of vertices, their odd neighbourhood is either $\{\}$, $\{1,2,3\}$, $\{4,5,6\}$ or $\{1,2,3,4,5,6\}$. Since, by condition $g3$, XY vertices have to occur in their own odd neighbourhood, we need to choose $g(1)$ in a way where at least 2 and 3 are in the odd neighbourhood of $g(1)$. From condition $g2$ follows that $1 \prec 2$ and $1 \prec 3$. However, we also need to choose $g(2)$ in a way where at least 1 and 3 are in the odd neighbourhood of $g(2)$. From condition $g2$ follows that $2 \prec 1$, which contradicts $1 \prec 2$. □

Gflow under rule application

Since having gflow is sufficient for successful circuit extraction, it is important to look at which ZX-rules preserve the gflow property and which do not. The simplification algorithm from Section 3.1.6 is based on five rules: fusion rule (f), Hadamard rule (h), the identity rules ($i1, i2$), local complementation (lc) and pivoting (p). Hadamard and identity rules do not modify any wire connections, so they do not change the underlying open graph and preserve the gflow property. In [4] it has been shown that the special case of spider fusion where two adjacent spiders are merged into one preserves the gflow property as well. It is important to note that there is no such result about spider unfusion, but since the basic algorithm does not use spider unfusion it does not matter here. Furthermore, it has been shown that local complementation and pivoting preserve gflow but the measurement planes of the neighbours can change. More precisely, applying local complementation on a vertex u flips all measurement planes of neighbouring vertices $N(u)$ from XZ to YZ and vice versa. Applying a pivot on two vertices u, v flips the measurement planes of XZ and YZ vertices

for all non-common neighbours of u and v, i.e. $N(u) \Delta N(v)$. However, neighbouring vertices in XY plane do not change their measurement plane in any case. The changing of the measurement planes will be discussed in the next chapter. For now it is important that, since the simplification algorithm preserves gflow, we can mathematically prove that every resulting ZX-diagram is extractable with the algorithm from 3.3.2.

3.4 PyZX Implementation

This section provides a short overview of the simplification strategies in the PyZX library[2]. PyZX is written in Python and is mainly designed to be used via Jupyter notebooks[3], although it also offers a command line interface. At its core, PyZX implements two basic classes: *Circuits* for representing quantum circuits, and *graphs* for representing ZX-diagrams. Circuits can be imported from various common Quantum circuit languages such as OpenQASM[4] and there are also algorithms implemented for optimizing circuits without using ZX-calculus. For instance, the `basic_optimization` algorithm optimizes circuits by using the techniques of Section 2.2.2. These algorithms are mostly used as a preprocessing step before converting circuits to graphs and using the simplification algorithms of this chapter as main optimization.

3.4.1 Simplification procedures

For graphs there are a variety of procedures based on ZX rewrite rules. Each procedure is split into a matcher method which calculates all matches, i.e., suitable non-intersecting spiders, and a rule application method which applies the rewrite rule on a list of matches, usually without checking suitability. For instance, the procedure `lcomp_simp` consists of the method `match_lcomp` which finds all spiders with phase $\pm\pi/2$, and the method `lcomp` which receives a list of those spiders and applies the local complementation rule on all of them. On a higher level these procedures are grouped into terminating simplification algorithms. There are three complete simplification algorithms, which are built on each other:

- `interior_clifford_simp`: This is an implementation of the algorithm from 3.1.6. After transformation to a graph-like diagram, the procedures `id_simp` ($i1, i2$), `spider_simp` (f), `pivot_simp` (p) and `lcomp_simp` (lc) are repeated until no changes have been made during a complete iteration.

- `clifford_simp`: This adds the pivot boundary simplification from Section 3.2.1. Both the `interior_clifford_simp` algorithm and the pivot boundary simplification are repeated until no changes have been made during an iteration. However, instead of using Equation 3.21, the pivot boundary simplification is implemented as a phase gadget as in Equation 3.25.

- `full_reduce`: This algorithm additionally applies the phase gadget simplifications 3.24 and 3.26 and is usually the most powerful simplification algorithm.

[2] https://github.com/Quantomatic/pyzx
[3] https://jupyter.org/
[4] https://github.com/QISKit/openqasm

Furthermore, there is also an algorithm named `teleport_reduce`, which works as described at the end of Section 3.2.2.
The extraction algorithm works as described in Section 3.3. The step of finding row operations for transforming the biadjacency matrix until at least one row contains only a single entry with 1 is done via *Gaussian elimination*. The gflow property ensures that by applying a full Gaussian elimination on a biadjacency matrix we get at least one row with a single 1.

3.4.2 Termination

An important point is how termination of the process is ensured. All procedures from `interior_clifford_simp` decrease the amount of spiders in every rule application. When there is no match left for any procedure, the algorithm terminates. However, the other two algorithms introduce phase gadgets via Equation 3.24 and 3.25. Since these rules do not reduce the number of spiders, it is possible that the algorithm reaches the same graph state multiple times and the algorithm does not terminate. PyZX prevents this by excluding phase gadgets from the matching methods. Since the implementation of spiders does not distinghuish between the three different measurement planes, this is done via counting the neighbours of spiders: If a spider has only one neighbour, this neighbour is considered a phase gadget and it is neither a candidate for local complementation nor pivoting.

3.5 Discussion of the PyZX simplifications

In order to test the effectivity of circuit optimization using the ZX-diagram simplification algorithms we used a method implemented in PyZX which generates random circuits. We can specify the number of qubits, the overall number of quantum gates, and the ratio of T gates to Clifford gates. Setting the T gate ratio to 0 generates a circuit which contains only Clifford gates and by increasing the ratio more and more non-Clifford gates with phase $k\pi/4, k \in \{1, 3, 5, 7\}$ will be present in the generated circuit. In this section we will first look at how the simplification strategies perform on pure Clifford circuits before looking at Clifford+T circuits.

3.5.1 Optimizing Clifford circuits

For testing the effectivity of the simplification algorithms on Clifford circuits we generated 60 random circuits split into three sets of 20 circuits with $10^2, 10^3$ and 10^4 gates. We compared the original circuits to the results of the `basic_optimization` algorithm for circuits and the `clifford_simp` algorithm for ZX-diagrams. Figure 3.1 shows the average gate count of the optimized circuits against the original gate count. Here the `clifford_simp` algorithm clearly outperforms the `basic_optimization` algorithm. Moreover, with increasing gate count the resulting circuits from `clifford_simp` have more or less the same gate count, which is always around 30 gates in total. We think this is because of the Gottesman-Knill theorem mentioned in 2.1.5. Since the Clifford gate set is not universal, the set of unitary matrices which can be represented by Clifford circuits is limited. Apparently each of these different matrices can be constructed using only a few gates. The use of local complementation and pivoting seems to be crucial for these good results, since the results for basic circuit optimization are far worse and do not stay constant with increased gate count.

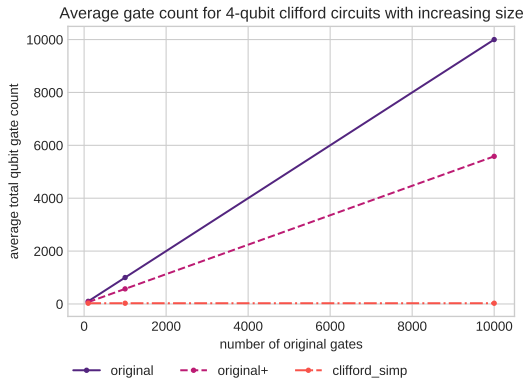
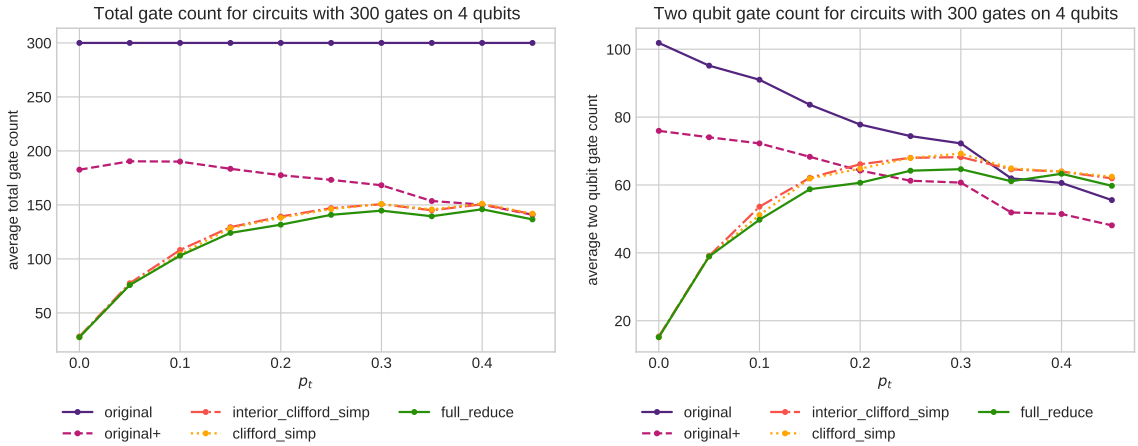


Figure 3.1: Average gate count of optimized 4-qubit Clifford circuits with increasing original gate count



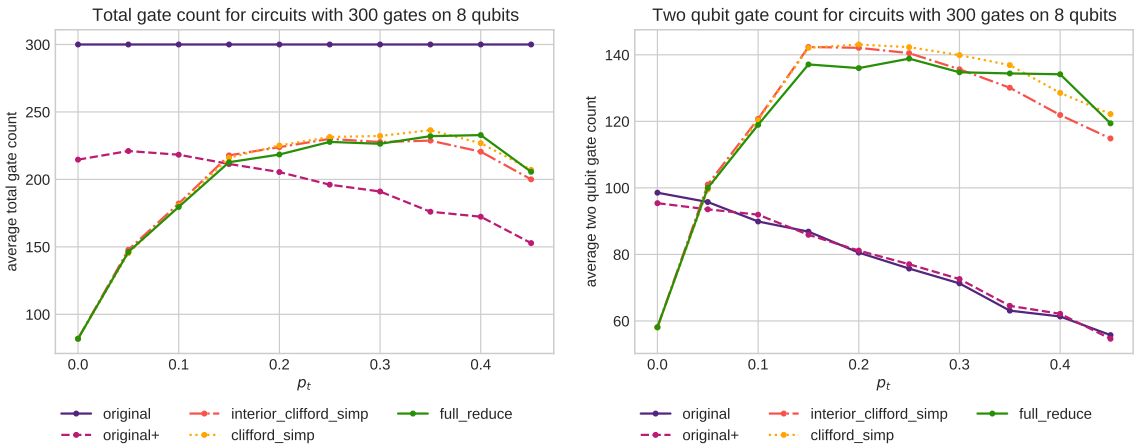
(a) Average total gate count

(b) Average two-qubit gate count

Figure 3.2: Average gate counts of 20 optimized 4-qubit circuits with an original gate count of 300 and increasing T gate probability p_t

3.5.2 Optimizing Clifford+T circuits

For testing the effectivity of the simplification algorithms for non-Clifford circuits we used four strategies: First, we optimized circuits without ZX-calculus by only applying the `basic_optimization` algorithm, then we optimized circuits by applying the `basic_optimization` algorithm followed by one of the `interior_clifford_simp`, `clifford_simp` or `full_reduce` algorithms plus circuit extraction. We tested each strategy on circuits with a fixed number of qubits and total gates for increasing T gate probabilities. For each T gate probability we generated 20 different circuits and calculated the average gate count after the optimization. As shown in Figure 3.2a, the methods using ZX-calculus outperform the basic circuit optimization strategy for circuits containing only a few number of T gates. While we get more or less the same results for `interior_clifford_simp` and `clifford_simp`, `full_reduce` is usually a bit better in optimization.
However, with increasing T gates, the strategies using ZX-calculus become ineffective. As



(a) Average total gate count

(b) Average two-qubit gate count

Figure 3.3: Average gate counts of 20 optimized 8-qubit circuits with an original gate count of 300 and increasing T gate probability p_t

shown in Figure 3.2b, when considering only the two-qubit gates, there are often even more gates in the optimized circuit than in the original circuit. If there are more than about 20% T gates in the original circuit, the `basic_optimization` algorithm without the ZX-diagram simplifications already reduces more two-qubit gates than the other strategies. This metric is especially important, since, as we mentioned in Section 2.2.1, two-qubit gates are more expensive than single qubit gates by a factor of 10. Applying the strategies on circuits with more qubits exacerbates the problem. Figure 3.3 shows the total and two-qubit gate count of optimized circuits with 8 qubits and an original gate count of 300.

The circuits optimized using ZX-diagrams soon have twice as many two-qubit gates than the original circuits and most of the time the `basic_optimization` algorithm performs better even in terms of total gate count.

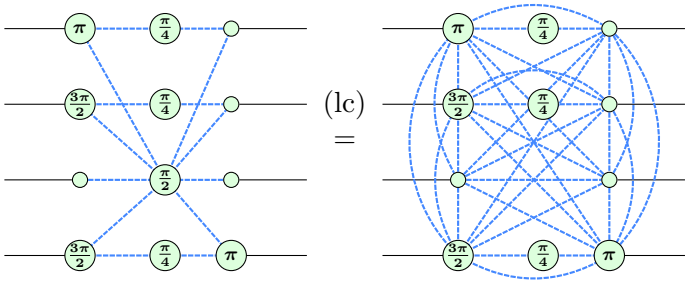
3.5.3 Factors for bad two-qubit gate reduction

The problem of bad two-qubit gate reduction seems to be essential for using ZX-calculus in quantum circuit optimization. In order to understand why this problem arises, we need to look at the extraction algorithm. Any unextracted simplified diagram for an m-qubit circuit has $2 * m$ boundary spiders and n interior spiders. The maximum number of Hadamard wires between spiders is

$$\frac{(2m+n)(2m+n-1)}{2} \tag{3.44}$$

The important point is to understand what happens to these wires during the extraction algorithm: We know that $m+n$ Hadamard wires connect spiders on the same qubit and are extracted as Hadamard gates. However, the remaining wires are either extracted as CZ gates or they get canceled out by using row operations on the biadjacency matrix which puts CNOT gates on the extracted part. In the worst case each wire is extracted as CZ gate otherwise the row operations can eliminate at least some wires. As far as we observed, the row operations can reduce wires by 50 % in the optimal case but proving this conjecture is outside the scope of this work.

From these observations we can draw two conclusions for improving ZX-calculus based optimization: Either the extraction algorithm has to be modified such that Hadamard wires are extracted more effective or the simplification algorithms optimize diagrams to contain as few Hadamard wires as possible. For the former it could be interesting to focus the row operations on biadjacency matrices not on finding *any* row with a single 1 but on finding as many rows with a single 1 as possible. This could lead to more extracted CNOTs in the first place but connections between frontier spiders, which always get extracted as CZ gates, would decrease. We think the amount of saved CZ gates could be larger than the amount of additional CNOT gates but we did not test this assumption. The main focus of our work is on the latter option. The current simplification algorithms optimize diagrams towards the lowest number of spiders no matter how many connections between the remaining spiders are left. Eliminating every possible spider often leads to more expensive circuits than keeping some spiders. Consider the following example where the last simplification step of a diagram consists in applying the local complementation rule on the third middle spider:



(3.45)

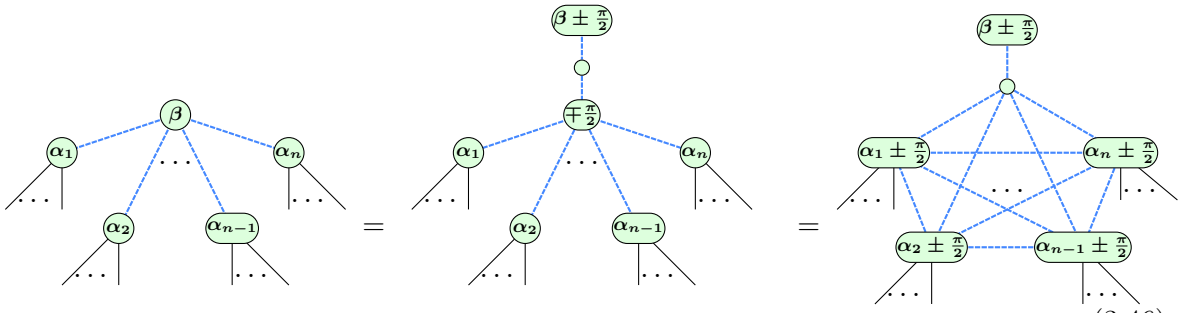
The first diagram has a total of 14 Hadamard wires, of which $4 + 4 = 8$ are extracted as Hadamard gates since they connect spiders on the same qubit. The remaining six wires are extracted either via the row operations or as CZ gates and indeed the PyZX extraction algorithm yields a circuit with a total gate count of 21 and two-qubit gate count of 6. The second diagram, however, has 34 Hadamard wires, of which $4 + 3 = 7$ are extracted as Hadamard gates. Extracting the diagram yields a circuit with a total gate count of 45 and a two-qubit gate count of 21 which is more than three times as many two-qubit gates than in the first extraction. So even though we can apply local complementation here to eliminate a spider it is not useful for circuit optimization.

3.5.4 Using local complementation and pivoting for reducing Hadamard wires

Since local complementation and pivoting are the only rules used in the simplification algorithms which can increase connections between spiders, our main idea was to investigate under which circumstances the application of these rules is useful. In a recent master thesis [5], genetic algorithms and simulated annealing are used to randomly apply local complementation and pivoting transformations after the `full_reduce` simplification. This approach exploits the fact that these rules can be applied to spiders with arbitrary phase with the

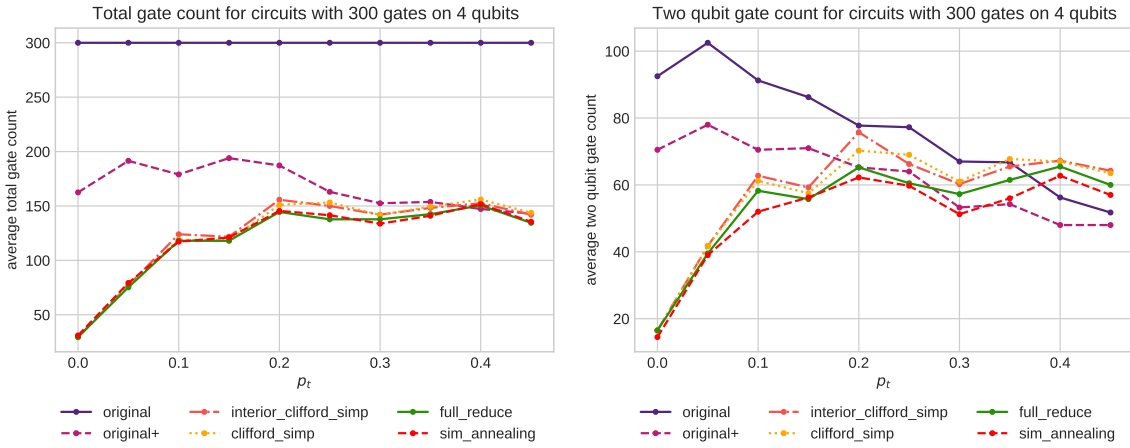
[5] https://www.youtube.com/watch?v=tUIcqXKEFhk

cost of introducing a new phase gadget, i.e.



(3.46)

The basic algorithm applies the rules to any spider or pair of spiders and checks whether the extraction algorithm yields a better result than before. Since the source code of this approach is already available in PyZX, we tested it against the existing algorithms. In Figure 3.4 we show how the simulated annealing algorithm applied on diagrams simplified with the `full_reduce` algorithm compares to the normal algorithms. The total gate count remains more or less the same, but the two-qubit gate count is already reduced significantly.



(a) Average total gate count

(b) Average two-qubit gate count

Figure 3.4: Average gate counts of 20 optimized 4-qubit circuits with an original gate count of 300 and increasing T gate probability p_t

However, this approach has some drawbacks: First, it can only be used as a post-processing step since no spiders get eliminated and we always need to simplify the diagrams in the first place using algorithms as `full_reduce`. As a consequence, all spiders with Clifford phases are already eliminated before this approach can take place so rule applications as in Equation 3.45 which drastically increase two-qubit count still occur. Second, depending on the iterations the approach is very slow in terms of runtime. In general, the gaussian elimination is the dominating part of all optimization algorithms, so the extraction algorithm usually takes much longer than diagram simplification[4, p. p.45]. Since this approach needs to extract a circuit after each rule application for scoring the result, the runtime increases very quickly: On a laptop computer running the simulated annealing algorithm with 100 iterations on diagrams containing 100 vertices takes 30 seconds, whereas running it on diagrams containing 300 vertices already takes more than 3 minutes.

4 Enhancing ZX-diagram simplification

In this chapter we present some of our own strategies for optimizing quantum circuits using ZX-calculus. Our strategies are drawn from the conclusions of the previous section: As soon as there are non-Clifford gates present in circuits, diagram simplification and extraction often lead to an increased CNOT and CZ count, which is particularly bad since they some of the most expensive gates in quantum circuits. For improving diagram simplification we can either try to reduce non-Clifford gates as much as possible or we can focus on two-qubit gate reduction. In this chapter we will first present a theoretical approach for eliminating non-Clifford gates using the Euler rule from Section 2.3.5. Since this approach is difficult to implement in practice and leads to spider phases which cannot be represented as fractions of π anymore, we decided not to focus on this approach but instead on reducing two-qubit gates in the remaining part of the chapter.
So far all existing approaches eliminate spiders as much as possible without considering how many wires are left. We place our focus on eliminating wires as much as possible while keeping some spiders even though they could be removed. For this task we use *cost functions* as a means of guiding the simplification process. Since local complementation and pivoting are the only rules creating new wires, we introduce cost functions for these rules which measure how many wires are being saved or newly generated.
However, we do not exactly know how the generation and elimination of Hadamard wires affect circuit extraction since some Hadamard wires are canceled out during the extraction algorithm. The two-qubit gate count of the extracted circuit either decreases by eliminating a Hadamard wire or it stays the same. We could get an exact cost similar to the approach from Section 3.5.4 by calculating the extracted circuit after each rule application but since this is very costly we consider the extraction algorithm as a *black box* in our work. Therefore, we only *estimate* how much two-qubit gates are saved or newly generated by a rule application and we will refer to the cost functions as *heuristics*. These heuristics then serve as basis for various simplification strategies.

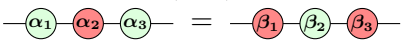
In this chapter we first present the elimination strategy based on the Euler rule before introducing the heuristic-based approaches. We then discuss some of the main issues we encountered during the design of those approaches and how we circumvent them. Since many issues arise from using phase gadgets, we then present an alternative method which keeps the effect of pulling "unwanted" phases out of spiders but does not rely on phase gadgets.

4.1 Reducing 2-ary spiders with the Euler rule

As first strategy, we will show how we can use the Euler rule to reduce 2-ary spiders, i.e.,spiders with only two neighbours. We mentioned the Euler rule already in 2.3.5 since it can be used to prove completeness of the ZX-calculus, but omitted the calculation of the

phases. Here we show the complete rule as denoted in [29]:

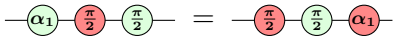
$$(EU)$$



$$\begin{aligned}
x^+ &:= \frac{\alpha_1 + \alpha_3}{2} \\
x^- &:= x^+ - \alpha_3 \\
z &:= cos\left(\frac{\alpha_2}{2}\right) cos\left(x^+\right) + i\, sin\left(\frac{\alpha_2}{2}\right) cos\left(x^-\right) \\
z' &:= cos\left(\frac{\alpha_2}{2}\right) sin\left(x^+\right) - i\, sin\left(\frac{\alpha_2}{2}\right) sin\left(x^-\right) \\
\beta_1 &:= arg(z) + arg(z') \\
\beta_2 &:= 2\, arg\left(i + \left|\frac{z}{z'}\right|\right) \\
\beta_3 &:= arg(z) - arg(z')
\end{aligned} \tag{4.1}$$

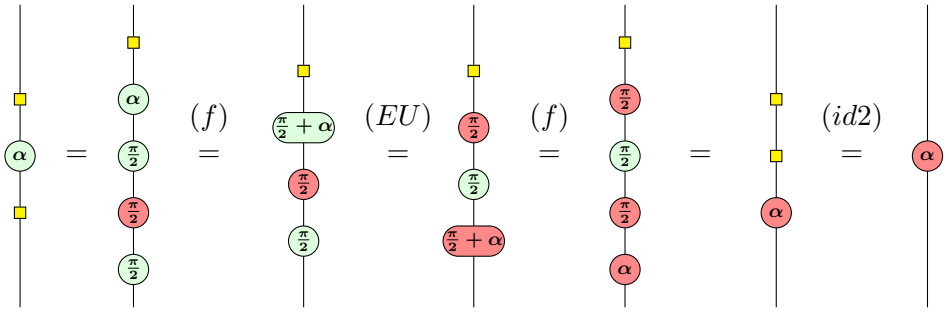
Although the graphical rule looks very simple, the calculation of the phases $\beta_1, \beta_2, \beta_3$ is not very intuitive. Nevertheless, the rule has some interesting properties.
First, this rule is related to the *Euler angles*: Every possible rotation in a three dimensional space (i.e. like in the Bloch sphere) can be achieved by three rotations around two axes: XYX, YXY, XZX, ZXZ, YZY or ZYZ. The Euler rule in ZX-calculus transforms an XZX rotation to its ZXZ equivalent and vice versa. So although the calculation of the angles is not very straightforward, the rule expresses a very fundamental property of qubits.
Second, the rule seems to be related to the Hadamard rule 2.3.4 which allows us to change the color of a spider by interchanging Hadamard wires and normal wires. Using the Euler rule we can "tunnel" the phase of a spider to the other side of two $\frac{\pi}{2}$ spiders as follows:

$$(EU)$$



(4.2)

At least for spiders with two wires the Hadamard rule is entirely contained in the Euler rule:

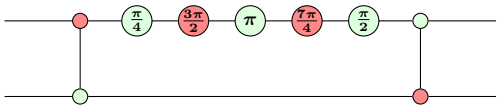


(4.3)

4.1.1 Simplification algorithm

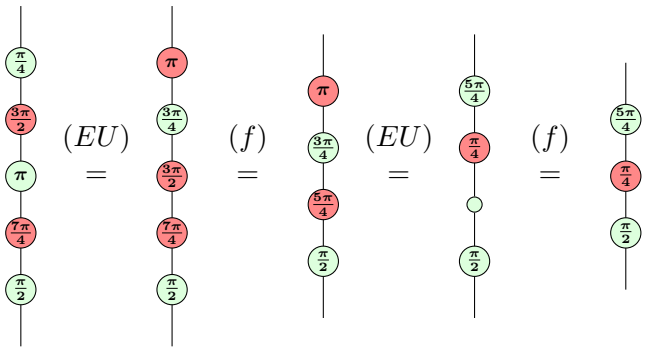
For simplifying diagrams we focus on the property that every sequence of 2-ary XZX spiders can be transformed to ZXZ spiders and vice versa. When transforming quantum circuits to ZX-diagrams, the diagrams consist of sequences of 2-ary spiders separated by some 3-ary spiders representing CNOTs or CZs. Using the Euler rule we can reduce these 2-ary spider

sequences between 3-ary spiders to at most two spiders in a row. Consider the following example, where we have five 2-ary spiders in a row:



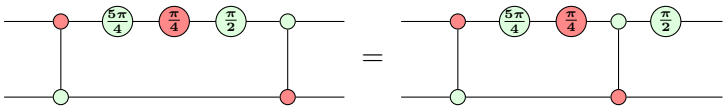
(4.4)

We can now do the following to reduce the amount of 2-ary spiders on the topmost wire to three:



(4.5)

This can be done with every sequence of 2-ary spiders and corresponds to the well known fact in classical quantum computing that every unitary operation on a single qubit can be achieved using at most three rotation gates [23]. However, we can reduce the amount of 2-ary spiders between 3-ary spiders even more by using the Euler rule to change the color of the rightmost or the leftmost spider to the color of the adjacent 3-ary spider and use the fusion rule for pulling this spider to the other side as follows:



(4.6)

If we have more than two spiders at the other side of the CNOT after this step, we can again use fusion and Euler rule to reduce the number of spiders to two. This process can be repeated until we reach the end of the qubit wire.

4.1.2 Drawbacks of the Euler rule

For most input phases $\alpha_1 - \alpha_3$, the Euler rule returns quite complicated phases for $\beta_1 - \beta_3$, for instance

$$\alpha_1 = \frac{\pi}{4}, \alpha_2 = \frac{\pi}{2}, \alpha_3 = \frac{\pi}{4} \Rightarrow \beta_1 = tan^{-1}\left(\sqrt{2}\right), \beta_2 = \frac{\pi}{3}, \beta_3 = tan^{-1}\left(\sqrt{2}\right). \tag{4.7}$$

A large part of the simplicity of ZX-calculus is based on having only fractions of π as spider phases which can be added or substracted very easily. Usually each rule application can be calculated without the help of a calculator. However, when we use the Euler rule a lot of this simplicity is lost since we can obtain phases which we can no longer represent as a fraction of π. In order to avoid rounding errors when adding irrational phases we would need to include a computer algebra system into PyZX. Moreover, while this approach could reduce some non-Clifford gates, the two-qubit gates are not reduced at all. Therefore, we decided to describe this approach only in theory and focus instead on improving the two-qubit count which is covered in the remaining part of this chapter.

4.2 Heuristics

In this section we introduce the basic heuristics for optimizing Hadamard wires with local complementation and pivoting. The main idea is to calculate how many wires are created by a rule application and compare them against the existing number of wires.

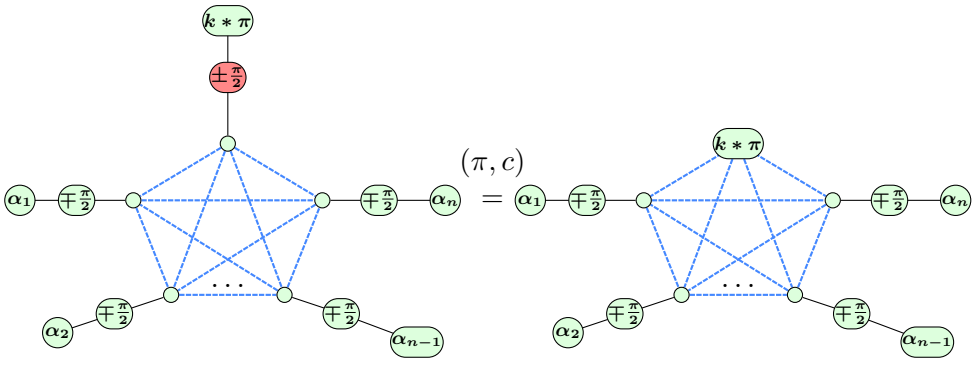
4.2.1 Local complementation heuristic

In order to find a suitable heuristic for local complementation, we first examine how the edges behave in the graph theoretic case, before looking at the corresponding ZX-rules:

Proposition. *Let $G = (V, E)$ be an open graph and $u \in V$ an arbitrary vertex with neighbours $N(u) \in V$. Furthermore, let $m = |N(u)|$ denote the number of neighbours, and n the number of edges between the neighbours, i.e., $n = |\{(a, b) \,|a, b \in N(u)\}|$. For $G \star u$, m remains the same, but n changes to $n' = \triangle_{m-1} - n$, where $\triangle_{m-1}$ is the triangular number $\frac{m(m-1)}{2}$.*

For our purposes we choose the heuristic function LCH to return the number of *saved* Hadamard wires, i.e., we substract the new number of wires from the original number of wires. The resulting equation $(m + n) - (m + (\triangle_{m-1} - n))$ can be simplified to $2n - \triangle_{m-1}$. In ZX-diagrams the application of local complementation on a spider u depends on its phase $\varphi(u)$:

- If $\varphi(u) = \pm\pi/2$ we can remove it as shown in Equation 3.15. In this case we add m to the heuristic, since all connections between u and $N(u)$ are eliminated.
- If $\varphi(u)$ is a non-Clifford phase we need to pull this phase in a phase gadget like in 3.46. Therefore, all connections between u and $N(u)$ remain but we need to add one additional wire for the gadget.
- If $\varphi(u)$ is 0 or π, we discovered that a phase gadget is not needed. Using the steps from 3.15 the π spider can be copied through the red spider in step 3:



(4.8)

Considering the three cases, our heuristic for local complementation LCH is as follows:

$$LCH(u) = \begin{cases} 2n - \triangle_{m-1} + m & \varphi(u) = \pm\frac{\pi}{2} \\ 2n - \triangle_{m-1} & \varphi(u) = k * \pi, k \in \mathbb{Z} \\ 2n - \triangle_{m-1} - 1 & otherwise \end{cases} \tag{4.9}$$

As an example, in Equation 3.45 calculating LCH for the middle spider with phase $\pi/2$ results in

$$2 * 0 - \frac{8 * 7}{2} + 8 = -20,$$

so after applying local complementation the diagram has 20 more Hadamard wires than before.

4.2.2 Pivot heuristic

In order to find a suitable heuristic for pivoting, we again first examine how the edges behave in the graph theoretic case. When pivoting around two connected vertices u, v, the connections are flipped between the disjoint sets

- $A \coloneqq N(u) \cap N(v) \backslash \{u, v\}$,
- $B \coloneqq N(u) \backslash N(v) \backslash \{v\}$,
- $C \coloneqq N(v) \backslash N(u) \backslash \{u\}$

and not between every neighbour of u and v. So instead of triangular numbers, we use the following sum to calculate the maximum number of new connections:

$$C_{max} = |A| * |B| + |A| * |C| + |B| * |C|.$$

Let m_u and m_v denote the number of neighbours of u and v, i.e.

$$m_u = |\{w \in V | w \in N(u)\}|, m_v = |\{w \in V | w \in N(v)\}|,$$

and n the number of edges between neighbours of different sets

$$n = |\{(a, b) | a \in A, b \in B \cup C \vee a \in B, b \in A \cup C \vee a \in C, b \in A \cup B\}|.$$

For $G \wedge uv$, the sum of m_u and m_v remains the same, but n changes to $C_{max} - n$. Again we choose the heuristic function to return the number of saved wires, which leads to the following equation:

$$(m_u + m_v + n - 1) - (m_u + m_v + (C_{max} - n) - 1). \tag{4.10}$$

Note that the -1 is necessary, because otherwise the wire between u and v would be counted twice. We can simplify this equation to

$$2n - C_{max}. \tag{4.11}$$

In ZX-diagrams we distinguish three cases depending on the phases $\varphi(u), \varphi(v)$ of u and v:

- If both spiders have a phase of 0 or π, all connections between $\{u, v\}$ and $N(u) \cup N(v)$ are eliminated. Therefore, we add m_u and m_v to Equation 4.11:

$$2n - C_{max} + m_u + m_v - 1.$$

- If v becomes a phase gadget and u gets eliminated, all neighbours of u get connected to v, i.e. $m_u - 1$, and we have an additional wire for the phase gadget (c.f. 3.24). The resulting equation is

$$\begin{aligned} & 2n - C_{max} + m_u + m_v - 1 - (m_u - 1) - 1 \\ = \; & 2n - C_{max} + m_v - 1. \end{aligned}$$

  The same equation holds with interchanged u and v, i.e., if u becomes a phase gadget and v gets eliminated.

- If both spiders become phase gadgets all neighbours of u get connected to v and all neighbours of v get connected to u. Furthermore, u gets connected to v again and we have two more wires for the phase gadgets:

$$\begin{aligned} & 2n - C_{max} + m_u + m_v - 1 - (m_u - 1) - (m_v - 1) - 1 - 2 \\ = \; & 2n - C_{max} - 2. \end{aligned}$$

The complete pivoting heuristic PH then is as follows:

$$PH(u,v) = \begin{cases} 2n - C_{max} + m_u + m_v - 1 & \varphi(u) = j * \pi, \varphi(v) = k * \pi, \; j,k \in \mathbb{Z} \\ 2n - C_{max} + m_v - 1 & \varphi(u) = j * \pi, \varphi(v) \neq k * \pi, \; j,k \in \mathbb{Z} \\ 2n - C_{max} + m_u - 1 & \varphi(u) \neq j * \pi, \varphi(v) = k * \pi, \; j,k \in \mathbb{Z} \\ 2n - C_{max} - 2 & otherwise \end{cases} \tag{4.12}$$

4.3 Basic strategies

Scoring rule applications with LCH and PH allows a variety of different simplification strategies. In general we use the Clifford simplification algorithm from 3.1.6, but instead of applying local complementation and pivoting as long as possible in step 3, we apply these rules based on a matching and a selection function. The selection function first aggregates all possible rule applications returned by the matching function and evaluates them using the heuristics. Depending on the strategy, one or no rule application is then selected and applied to the diagram. We repeat this step until no rule gets selected, then we go to step 4 and start a new iteration. In theory, when using phase gadgets, the matching function can return every spider for local complementation and every pair of spiders for pivoting. However, allowing every match can lead to non-termination of the simplification algorithm. In this section we first discuss some basic selection functions, then we show how certain matches can cause non-termination and how we circumvented this in our implementation.

4.3.1 Selection functions

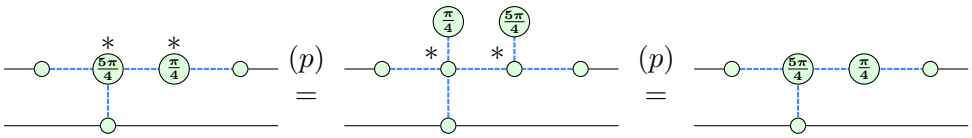
The selection function receives all matches and calculates the gain for each rule application, i.e., the number of saved Hadamard wires, using LCH and PH. For our selection functions we can specify three parameters: First, we implemented the possibility to specify a lower bound for the gain. For instance, setting the lower bound to -5 would result in only applying those rules which do not increase the amount of Hadamard wires in the diagram by more than five wires. Second, we can specify whether rule applications are allowed on boundary spiders and third, whether rule applications are allowed which generate phase gadgets. In PyZX we implemented three different selection functions:

- **Random selection**: This function randomly applies a rule with gain $\geq$ the lower bound. If we do not specify a lower bound, this strategy corresponds to the current PyZX simplification process.
- **Greedy selection**: This function always applies the best rule application, i.e., the application which saves the most Hadamard wires, until we either reach the lower bound, or no rule can be applied anymore.
- **Metaheuristic selection**: This function uses a metaheuristic to dynamically explore the search space. In each step we randomly choose a rule application from the matching set. If the gain is > 0, i.e. some Hadamard wires are saved, we apply the rule in any case, otherwise we only apply the rule up to a certain probability.

As a metaheuristic selection function we implemented the *simulated annealing* algorithm. For a given number of iterations n, a starting temperature t and a cooling factor α, this algorithm chooses a random rule application n times. If the heuristic of the chosen rule application Δ_c is positive and Hadamard wires are saved, the rule is applied to the diagram in any case. Otherwise the rule is only applied with a probability of $e^{\Delta_c/t}$. After each iteration the temperature decreases by a factor of α, i.e. $t_{n-1} = t_n * \alpha$. This has the effect that rule applications which increase the amount of Hadamard wires are more likely to be applied at the beginning of the simplification algorithm since $e^{\Delta_c/t}$ will be closer to 1. Towards the end $e^{\Delta_c/t}$ becomes almost 0 and rules with a negative heuristic will most likely not be applied anymore.

4.3.2 Termination

Each algorithm terminates when allowing only rule applications with a heuristic result > 0. Since every step reduces at least one Hadamard wire, the algorithms terminate when there is no rule left that decreases the amount of Hadamard wires. Furthermore, the algorithms also terminate if we allow negative gains, but restrict the matches to interior spiders where the rule application does not generate new spiders. This is the case for the standard local complementation on a spider with phase $\pm\pi/2$ and pivoting on a pair of spiders with phase 0 or π. Since every step eliminates at least one spider, the algorithm terminates when there is no interior spider left with phase $\pm\pi/2$ and no pair of interior spiders with phase 0 or π. However, allowing rule applications on spiders of arbitrary phases with gain ≤ 0 may result in a non-terminating algorithm. Consider the following example:



(4.13)

Here, pivoting on the spiders marked with $*$ has a gain of -2 and generates two new phase gadgets. As second step we apply pivoting again on the newly generated gadgets, which has a gain of $+2$. As this results in the original diagram, the simplification algorithm could repeat these two steps forever and would not terminate. We have implemented a way to circumvent this problem by storing all spiders at the beginning of the simplification algorithm and only allow rule applications with positive heuristic result except for rule applications on those previously stored spiders which may also have a heuristic result of ≤ 0. Since each

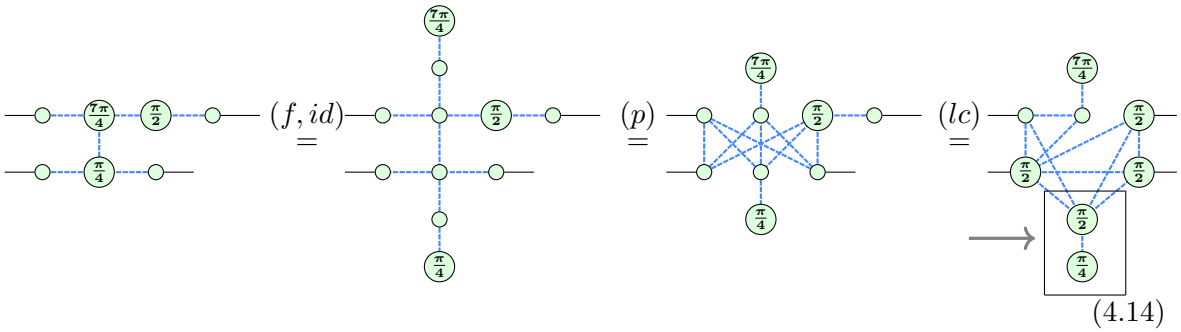
step then decreases either the amount of wires, spiders or original spiders, the algorithm terminates as well. In our example from 4.13, the pivoting would be applied once on the non-Clifford spiders, since they are present at the beginning of the simplification algorithm, but after that, pivoting would not be applied again, since the non-Clifford spiders are now newly generated spiders.

4.4 Measurement planes in PyZX

One of the main issues during the implementation of our strategies was that PyZX does not distinguish between different measurement planes. Even though the normal simplification algorithms use phase gadgets, i.e. spiders in YZ plane, they are only distinguished from other spiders by looking at the connectivity: If a spider u has only one neighbour v, v is considered as the root of a phase gadget measured in YZ plane and u is the corresponding measurement effect (c.f. 3.39). So far, rule application on measurement effects is undefined, so u is excluded from every matching function, i.e. even if such a spider has a phase of $\pm\pi/2$, no local complementation is applied to it during the simplification algorithm. During the extraction algorithm all spiders v are removed using pivoting as shown in 3.41. However, spiders in YZ or XZ plane which do not look like a phase gadget cannot be recognized as such in PyZX. In the existing simplification algorithms, this is no problem, as the only spiders measured in a non XY plane are spiders in phase gadgets. Yet, our simplification algorithms generate more diverse spider types, which cause problems when extracting diagrams. In this section we show which spider types are generated and how we prevent diagrams from becoming unextractable in PyZX.

4.4.1 XZ spiders in simplified diagrams

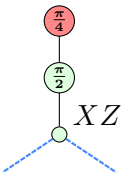
After a diagram has been simplified with one of the existing algorithms it contains only XY spiders and YZ spiders in the form of phase gadgets. Our algorithms generate diagrams with spiders in all three measurement planes. Consider the following example, which is based on the diagram from Section 3.3.2:



(4.14)

In the last step we have created something which is neither a spider in XY nor in YZ plane. By unfusing the spider with phase $\pi/2$ and changing the color of the spider with phase $\pi/4$

we can see that this construct corresponds to a measurement in XZ plane (cf. 3.39):



(4.15)

Although this spider type can also occur during the normal PyXZ simplification algorithms `clifford_simp` and `full_reduce`, eventually all XZ spiders are removed via local complementation. However, since applying local complementation to every spider with phase $\pm\pi/2$ may increase the number of Hadamard wires, our strategies do not apply local complementation on some spiders and the simplified diagram may contain spiders in XZ plane. The extraction algorithm from [4] is able to extract XZ spiders via local complementation, but this feature is missing in the PyZX extraction algorithm, because it is not necessary for the normal algorithms. In order to extract spiders in XZ plane, we distinguish phase gadgets by the phase of its root spider. If a spider has only one neighbour, this neighbour is in YZ plane if it has a phase of 0 or π and it is in XZ plane if it has a phase of $\pm\pi/2$. We then modified the extraction algorithm by inserting the following algorithm at the beginning:

1. Search a spider that is connected to exactly one neighbour whose phase is $\pm\pi/2$.

2. If nothing is found we are done, else apply local complementation on this neighbour and go back to step 1.

4.4.2 Rule application on XZ and YZ spiders

As shown in [4], local complementation and pivoting change the measurement plane of the vertices they are applied on. More precisely, the planes change as follows for local complementation on a vertex u:

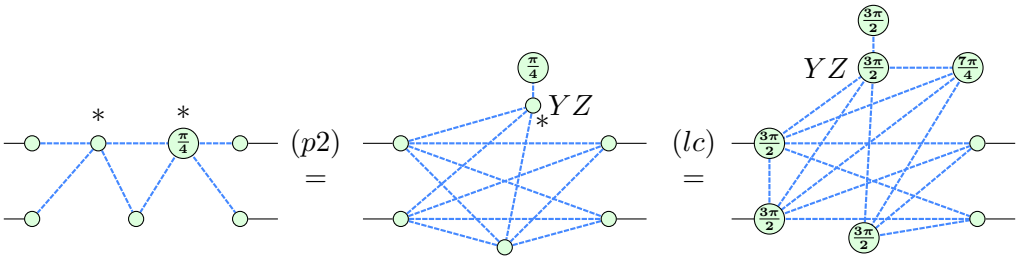
$$\lambda'(u) = \begin{cases} XY & \lambda(u) = XZ \\ YZ & \lambda(u) = YZ \\ XZ & \lambda(u) = XY \end{cases}, \tag{4.16}$$

and for pivoting on two vertices u, v:

$$\lambda'(w) = \begin{cases} XY & \lambda(w) = YZ \\ YZ & \lambda(w) = XY \\ XZ & \lambda(w) = XZ \end{cases}, w \in \{u, v\}. \tag{4.17}$$

Usually, the changing of measurement planes causes no problems in PyZX, since spiders in XY plane get either removed or transformed to a phase gadget when applying local complementation or pivoting. However, allowing rule applications on spiders which are already in XZ or YZ plane may lead to spiders where we cannot determine the measurement

plane based on the connectivity. For instance:

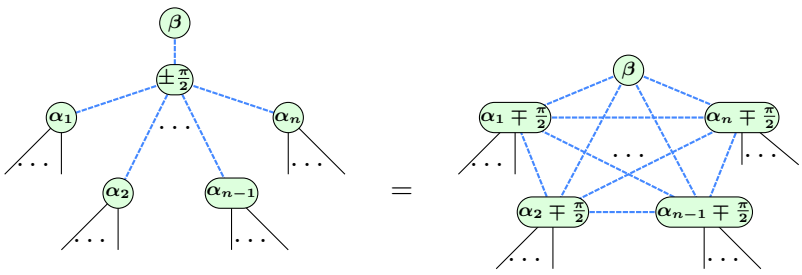


(4.18)

Here we applied Equation 3.24 on the two marked spiders, which generates a spider measured in YZ plane. Since the top spider with phase $\pi/4$ has only one neighbour, the extraction algorithm can identify the YZ spider and remove it. However, in the second step we applied 3.46 on the YZ spider. According to 4.16, the remaining spider is still measured in YZ plane. However, it has a phase of $-\pi/2$, so PyZX recognizes this spider as being measured in XZ plane. The extraction algorithm now tries to eliminate the spider with local complementation instead of pivoting, which leads to an unextractable diagram. Unfortunately, this is also the case for Equation 4.8, since the remaining spider is measured in YZ plane, but cannot be identified as such. To overcome those issues we choose to exclude all spiders which we can identify as spiders in XZ and YZ planes from our matching functions and apply Equation 3.46 instead of 4.8 on spiders with phase 0 or π. In theory, we do not need this restriction, but rewriting the PyZX library for supporting measurement planes had been out of the scope of this work.

4.5 Using neighbour unfusion instead of phase gadgets

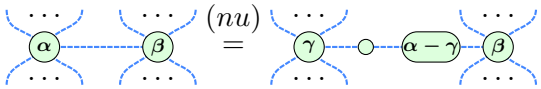
It seems that allowing spiders of YZ and XZ planes causes a lot of problems for our diagram simplification algorithms. In addition to the restrictions of the previous section, spiders in XZ and YZ plane are modified via local complementation and pivoting during the extraction algorithm, which affects the number of Hadamard wires *after* the simplification. These rule applications are usually especially expensive, since the measurement effect spider gets connected to every neighbour of the XZ resp. YZ spider, i.e. in the worst case for XZ spiders:



(4.19)

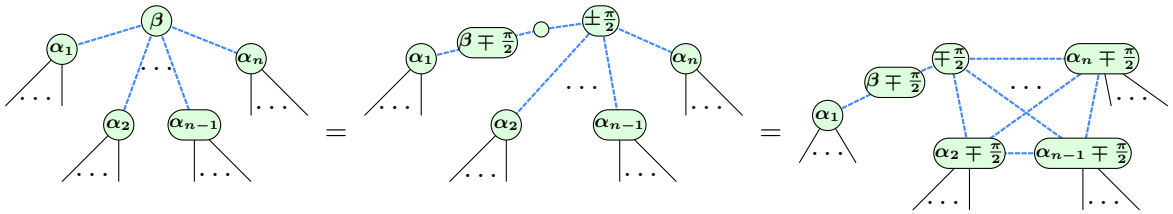
Therefore, a simplified diagram containing some YZ and XZ spiders may result in a more expensive circuit than another simplified diagram with more Hadamard wires but fewer YZ and XZ spiders.
To overcome this obstacle, we came up with a different approach which keeps most advantages of phase gadgets but does not generate any spider in XZ or YZ plane. The basic

concept looks as follows:



(4.20)

This rule, which we will refer to as *neighbour unfusion*, follows directly from the spider fusion and identity rules. As long as a spider with a phase α is connected to a neighbour, we can change its phase to an arbitrary phase γ by inserting an empty spider and a spider with phase $\alpha - \gamma$ between the spider and its neighbour. We can use neighbour unfusion for applying local complementation with spider removal to any spider with arbitrary phase since we can always change the phase of the spider to $\pm\frac{\pi}{2}$ as follows:

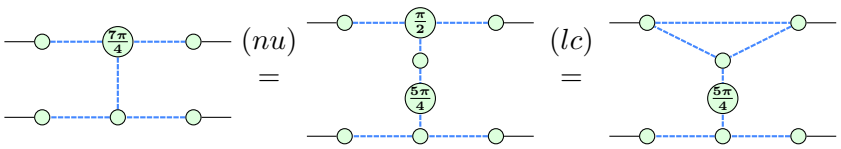


(4.21)

So compared to Equation 3.46, we do not complement all neighbours $a_1 - a_n$ but only $a_2 - a_n$, but otherwise have the same effect without introducing a phase gadget. This also works for pivoting, since each spider of u, v has at least one neighbour $w \notin \{u, v\}$ which can be used for neighbour unfusion.

4.5.1 Gflow under neighbour unfusion

As mentioned in 3.3.3, there are no proofs on whether general spider unfusion preserves the gflow property of a diagram yet. With our constraint to allow only spiders in XY measurement plane, spider unfusion and thus neighbour unfusion *destroys* the gflow property in some cases. Consider the following example:



(4.22)

It can be proven in a way similar to 3.40 that we cannot construct a gflow map if all spiders are measured in XY plane. On the other hand, the diagram admits gflow if some spiders are measured in YZ plane. We do not know whether there are some rule applications of neighbour unfusion which break the gflow completely, but since our goal is to restrict simplification to diagrams containing only XY spiders, we tried to find a function which tells us whether neighbour unfusion preserves gflow if all spiders are measured in XY plane. Our basic observation was that if two connected spiders are extracted on the same qubit, we can insert as many XY spiders as we want between them, because they are all extracted as R_z gates. If we insert spiders between two connected spiders which are extracted on different qubits though, the diagram sometimes becomes unextractable, although we do not know yet whether this is due to YZ and XZ spiders or due to general violation of the gflow property. So the question is if we can find a function which tells us whether two spiders are extracted

on the same qubit, because then we can apply neighbour unfusion with XY spiders between them. We think that this can be achieved using the gflow property itself, more precisely the map $g(v)$, which assigns a correction set to each non-output vertex v of a labelled open graph. For any vertex v measured in XY plane $v \in Odd(g(v))$ by condition $g(3)$, therefore we can always find a vertex $w \in g(v)$ which is connected to v. If furthermore w is the only element in $g(v)$, our hypothesis is that the extraction algorithm always extracts v and w to the same qubit. On the basis of this hypothesis we built another simplification algorithm which works like the basic strategies from Section 4.3, but using neighbour unfusion instead of phase gadgets. When applying local complementation and pivoting on spiders with non-Clifford phases, we unfuse the phase of a spider v to a neighbour w iff one of the following two conditions is met:

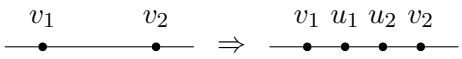
1. $w \in g(v) \wedge |g(v)| = 1$
2. $v \in g(w) \wedge |g(w)| = 1$

4.5.2 Gflow calculation

In the normal simplification algorithms the gflow map does not need to be calculated. Since all rule applications preserve gflow, the extraction algorithm works and it is sufficient to know that the map exists in theory. However, in our neighbour unfusion algorithm we need the gflow map in order to find suitable neighbours. PyZX implements an algorithm for calculating gflow for XY spiders, which is taken from [21]. This algorithm calculates the *maximally delayed gflow* for an open graph, which is minimal in the depth of the partial ordering $\prec$. Like the extraction algorithm, the gflow algorithm in PyZX uses gaussian elimination on biadjacency matrices which is computationally very expensive. Although we were able to speed up the performance of the gflow algorithm significantly by moving the gaussian elimination one step out of a for loop to a higher level, computing the gflow map at each simplification step is too expensive. For large diagrams with ~ 10000 spiders, even a single gflow calculation takes $5 - 10$ minutes on a laptop computer and recalculating it after each simplification step soon leads to total runtimes of days or even months. Thus, we came up with the idea of calculating the complete gflow map only at the beginning of our simplification and modify the map of all vertices affected by the rule application after each step. In [4, Chapter 3.1], it is shown how to calculate a gflow preserving map after local complementation and pivoting. For neighbour unfusion between two vertices v_1 and v_2, where $v_2 \in g(v_1)$ we modify the gflow map as follows:

$$\begin{aligned} g'(v_1) &= \{u_1\} \\ g'(u_1) &= \{u_2\} \\ g'(u_2) &= \{v_2\} \end{aligned} \tag{4.23}$$

Here u_1 and u_2 are the inserted vertices between v_1 and v_2:



Using these modifications after rule application we can speed up the simplification algorithm a lot and we are able to simplify the aforementioned diagrams with ~ 10000 spiders in less than one hour.

As we will see in the next chapter, modifying the gflow instead of recalculating it after each step sometimes leads to unextractable diagrams. Since we could not find an error in our implementation, we think that either our hypothesis of gflow preserving neighbour unfusion is wrong, or that it only holds for maximally delayed gflows, because when recalculating the gflow after each step the simplification algorithm never leads to unextractable diagrams. However, an exact proof or refutation of our hypothesis is out of scope of this work. To our knowledge, the evolution of gflow under spider unfusion is an area of research that is currently being investigated as part of a PhD thesis.

5 Evaluation

In this chapter we compare the algorithms from the last chapter to the existing algorithms of the PyZX library and to an up to date quantum circuit optimization algorithm which is not based on ZX-calculus. We run the algorithms with different parameters on both randomly generated circuits and existing benchmark circuits and evaluate which algorithms optimize circuits the most. We will first define the basic metrics for scoring circuits and outline which strategies we used to evaluate the PyZX algorithms, then we show how our algorithms perform on randomly generated circuits and last we compare the algorithms on benchmark circuits.

5.1 Circuit metrics

As mentioned in 2.2.1, the real cost of quantum gates varies depending on hardware implementation. Usually in Clifford+T+CNOT circuits, T gates are considered the most expensive gates followed by two-qubit gates and last Clifford gates [22]. Since the T gate it is the most expensive gate, in recent years a lot of effort has been put into developing algorithms for reducing T gate count [2][22][15]. Yet, our algorithms focus on two-qubit gate reduction and while they may eliminate some Clifford gates, the amount of T gates stays the same. Thus, while the T gate count remains an important factor for comparing circuits our main metrics are the two-qubit gates count and the total gate count. In summary we use four different metrics for evaluating circuit cost:

- The total gate count C_{QG} which includes all gates of a quantum circuit.
- The two-qubit gate count C_{2QG} which counts all two-qubit gates in a circuit.
- The T gate count C_{TG} which counts all T gates.
- The weighed gate count C_w which counts all gates of a quantum circuit but weighs two-qubit gates with a factor of 10 compared to single qubit gates.

5.2 PyZX-based algorithms

Throughout this chapter we use seven different algorithms from the PyZX library including the four new algorithms from the last chapter. Although they were already discussed in the previous chapter we consider it useful to list them here again. From the existing PyZX library as of June 2021 we chose the following algorithms:

- `basic_optimization`
  This algorithm does not use ZX-calculus but gate cancellation and commutation to optimize circuits. While this is usually only a preprocessing step before optimizing circuits with ZX-calculus, we will see that for some circuits it still performs better than ZX-calculus based approaches which is why we include it in our evaluation.

- `full_reduce`
  This algorithm is the standard method for ZX-diagram simplification and uses the algorithm from 3.1.6 and all rules listed in 3.2

- `simulated_annealing_post`
  This algorithm uses the approach from Section 3.5.4, i.e., after running the `full_reduce` algorithm, simulated annealing is used to find a minimal circuit using local complementation and pivoting on randomly selected spiders as a post-processing step.

These algorithms are compared against four heuristic-based new algorithms from the last chapter:

- `random_simplification`
  This algorithm uses a random selection function for selecting a rule and can be parametrized with a lower bound for the result of LCH and PH and whether rule applications which generate phase gadgets are allowed.

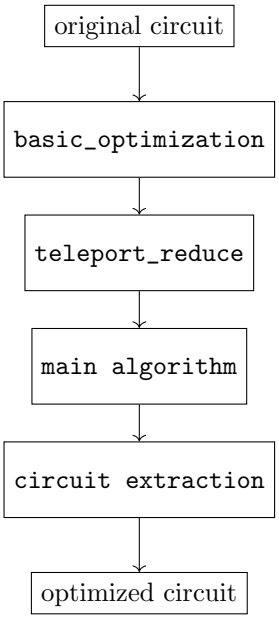
- `greedy_simplification`
  This algorithm always selects the best possible step in terms of Hadamard wire reduction, i.e., the rule application where LCH or PH are maximal. It can be parametrized like the `random_simplification`.

- `simulated_annealing`
  This algorithm uses simulated annealing to apply random rule applications where the probability of applying rules which increase Hadamard wire count gets smaller during runtime. It is parametrized by an initial temperature t_s and a cooling factor α.

- `neighbour_unfusion`
  This algorithm works like the `greedy_simplification` algorithm but includes the neighbour unfusion from Section 4.5. Since no phase gadgets are created we can only parametrize this algorithm with a lower bound for LCH and PH. As mentioned in 4.5.2 in this algorithm we need to recalculate gflow after each simplification step which leads to long runtimes. As we will discuss in Section 5.3.4 we cannot apply this approach on very large circuits in practice since the total runtime would increase too much.

Since our optimization algorithms do not eliminate T gates we always apply the `teleport reduce` algorithm before the actual diagram simplification. Since `teleport reduce` *only* eliminates T gates and otherwise leaves the circuit structure unchanged, it serves as a basis for comparing all ZX-calculus algorithms against each other because after this step every algorithm only eliminates Clifford and CNOT gates. Furthermore, we use the `basic_optimization` as a preprocessing step. In summary, for evaluating ZX-calculus based algorithm we use the following pipeline to compare the original circuits to the optimized

ones:



5.3 Optimization of randomized circuits

As a first approach we evaluate all PyZX based algorithms on circuits using a random circuit generator where we can specify the number of qubits, total gates, two-qubit gates and T gates. Although these circuits do not implement a meaningful problem, this serves as a theoretical test environment for evaluating how our algorithms perform on different circuit types.

5.3.1 Framework

The random circuits were generated using a combination of the following parameters:

- A total gate count of either 500 or 1000 gates.
- A number of qubits of either 4 or 8.
- A T Gate probability of 10,20,30, or 40%.
- A two-qubit gate count of 30%.

We choose the different number of total gates and number of qubits for evaluating what happens to our optimizations when doubling either one or both of the parameters but due to the long runtime of the `neighbour_unfusion` algorithm we only generate small circuits. The different T gate probabilities are motivated by real world circuits. From our observations most quantum circuits do have at least some T-gates, but circuits with more than 50% T gates are very unlikely. As we will see, the number of qubits and the T gate probability have the most impact on the performance of the optimization algorithms so we choose to set the two-qubit gate count to a fixed number in order to reduce the dimensionality of the different parameters. Since the variable placement of gates in randomly generated circuits leads to differing results of the simplification algorithms, we generate 10 circuits for each parameter

combination and take the average gate count as a more expressive result. Furthermore, we use the following parameters for the heuristic-based algorithms:

1. A lower bound for LCH and PH of either 1,−10,−20 or −30. Using the first bound we allow only rule applications which *decrease* the amount of Hadamard wires, using the other bounds we allow also rules which *increase* the amount of Hadamard wires up to the specified number. This bound can be applied to all strategies except `simulated_annealing`.

2. A flag whether rules with phase gadget generation are allowed or not for `random_simplification` and `greedy_simplification` which is either $False$ or $True$.

3. An initial temperature for `simulated_annealing` set to 0.5,1, or 2 times the amount of gates before simplification

4. A cooling factor for `simulated_annealing` of either 0.9, 0.95, or 0.99.

5.3.2 Evaluating different strategy parameters

As a first step we evaluated which parameter settings are useful on which types of circuits. As it can be seen in Figure 5.1 we first compared the `random_simplification`, `greedy_simplification` and `neighbour_unfusion` algorithms against various circuits using either 1,−10,−20 or −30 as lower bounds for the heuristic.

While allowing only rules which do not increase Hadamard wire count seems to be the best strategy for circuits with 4 qubits and a low T gate count, for circuits with more T gates or qubits it is better to allow some rule applications which increase Hadamard wire count. However, results vary on which negative lower bound is optimal. While for the 4 qubit circuits with high T gate probability and for 8 qubit circuits with low T gate probability it seems best to set the lower bound to a number ≤ -20, the best lower bound for 8 qubit circuits with high T gate probability is -10.
Another comparison can be made for the random and the greedy simplification on whether to allow phase gadgets or not during optimization. For this task we collected the average of the simplification algorithms using phase gadgets on different lower bounds and the average of the simplification algorithms not using phase gadgets. As shown in Figure 5.2, disallowing phase gadgets almost always leads to better results. This seems to support our hypothesis that the resolving of phase gadgets during circuit extraction, like in Equation 4.19 increases the two-qubit gate count significantly. Usually diagrams simplified with phase gadgets contain equal or less Hadamard wires than their equivalent diagrams simplified without phase gadgets so the only possible explanation is that the Hadamard wires increase during circuit extraction.
As last parameter we evaluate the choosing of the initial temperature t_s for the simulated annealing algorithm and the cooling factor α. For the evaluation we assumed that a static initial temperature will not accomodate the different circuit sizes. Therefore, we set the initial temperature to a multiple of the circuit size which is either 0.5, 1 or 2. As for α, the higher the cooling factor is, the more iterations the algorithm will run through. We assumed that the cooling factor should in general not be too low, because otherwise the algorithm would soon behave like the `random_simplification` with lower bound 1 so we used 0.9, 0.95 and 0.99 as cooling factors. As it can be seen in Figure 5.3, the cooling factor is much more important for good optimization results than the number of iterations. Moreover, while for small circuits increasing the factor α seems to yield better optimization results, for larger

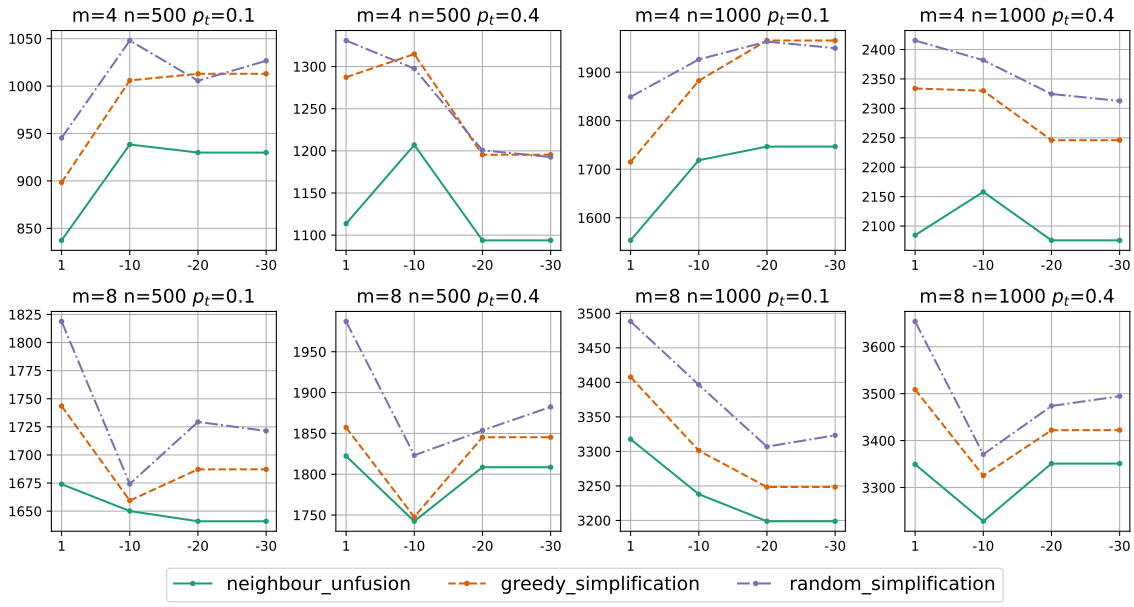


Figure 5.1: Weighed gate count for m qubit circuits with a gate depth of n and T gate probability p_t optimized with the `neighbour_unfusion`, `random_simplification` and `greedy_simplification` algorithm under varying lower heuristic bounds. The X-axes denote the lower bound, the Y-axes the weighed gate count C_w.

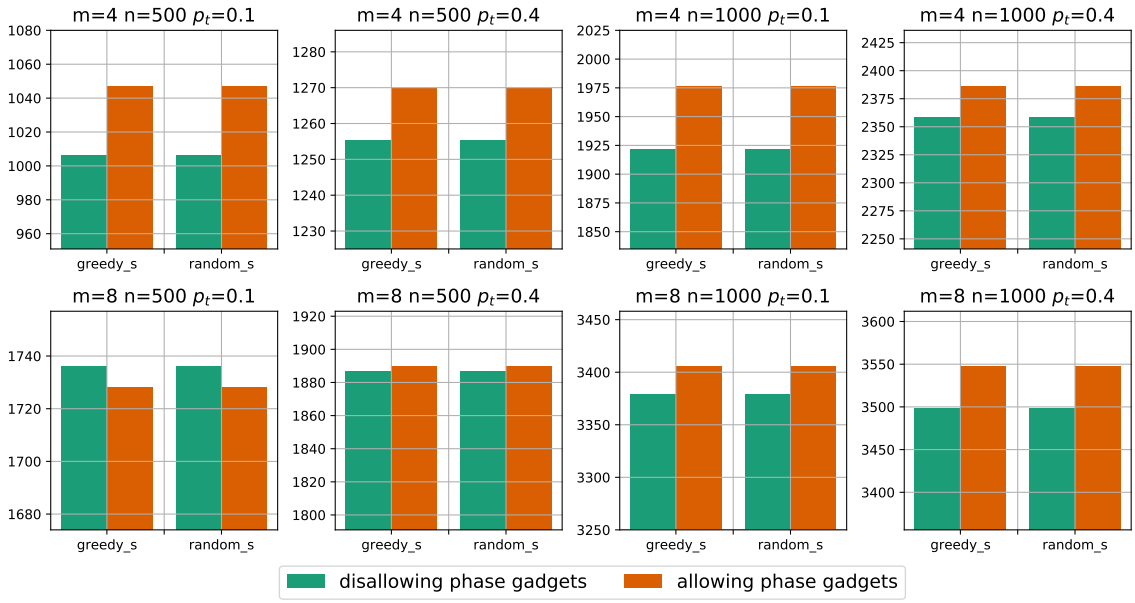


Figure 5.2: Weighed gate count for m qubit circuits with a gate depth of n and T gate probability p_t optimized with the `greedy_simplification` and `random_simplification` either allowing or disallowing rule applications which phase gadgets. The X-axes denote the algorithms abbreviated to `greedy_s` and `random_s`, the Y-axes the weighed gate count C_w.

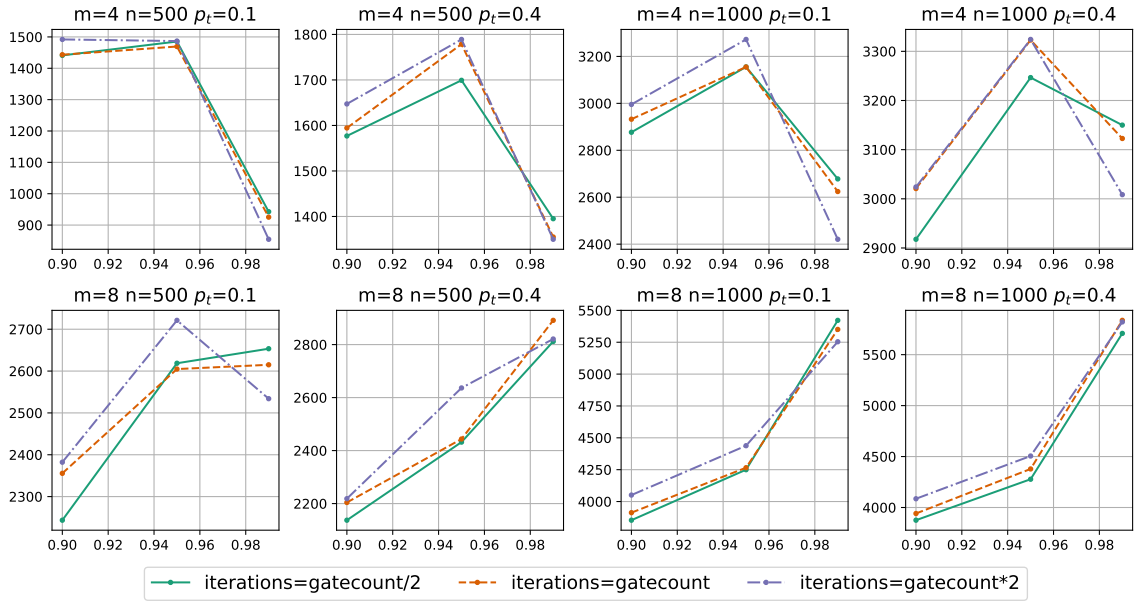


Figure 5.3: Weighed gate count for m qubit circuits with a gate depth of n and T gate probability p_t optimized with `simulated_annealing` under varying starting temperatures and cooling factors α. The X-axis denotes the cooling factor, the Y-axis the weighed gate count C_w.

circuits it is often best to have a low cooling factor with less iterations. Our first guess had been that for larger circuits the number of iterations is far to little but testing the algorithm with a factor of 5 or 10 more iterations than circuit depth showed no difference. Therefore, we think that for large circuits a high factor α leads to a very slow annealing process and many particularly "bad" rules in terms of Hadamard wire increase are applied before the algorithm terminates, which subsequently leads to a high circuit cost.

5.3.3 Optimization results

Considering the parameter selection we can now both compare our algorithms against each other and to the existing PyZX algorithms. For 4 qubit circuits we choose to disallow phase gadgets and to set the lower heuristic bound to 1, for 8 qubit circuits we set the lower heuristic bound to -10. For the `simulated_annealing` algorithm we set the initial temperature to the number of gates in the circuit and the cooling factor α to 0.99. In Figure 5.4 we first compare the new approaches against each other. We can see a clear ranking between the different algorithms regardless of circuit size and number of qubits: The `neighbour_unfusion` algorithm optimizes circuits the best, followed by the `greedy_simplification` and the `random_simplification` and last the `simulated_annealing` algorithm. With increasing T gate count the algorithms do not optimize circuits as well as for a low T gate count. Increasing the number of qubits has a similar effect: While for 4 qubit circuits the costs get nearly halved by the `neighbour_unfusion` algorithm, for 8 qubit circuits the algorithms reduce the cost only marginally. However, the total amount of gates does not seem to have an impact on the effectivity of the algorithms. Compared to the other algorithms, the simulated annealing algorithm does not perform very well. As mentioned, we suspect that this may be to the fact

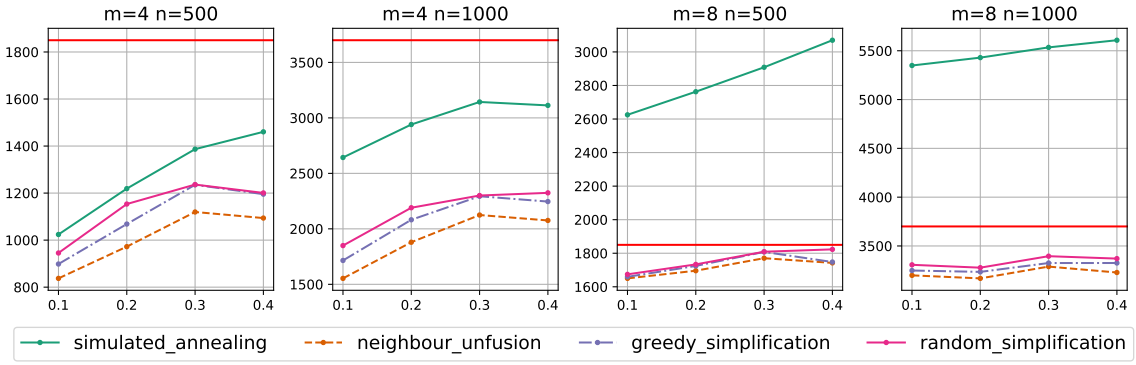


Figure 5.4: Weighed gate count for m qubit circuits with a gate depth of n and increasing T gate probability p_t optimized with different algorithms. The X-axis denotes the T gate probability, the Y-axis the weighed gate count C_w. The red line denotes the weighed gate count of the original circuit.

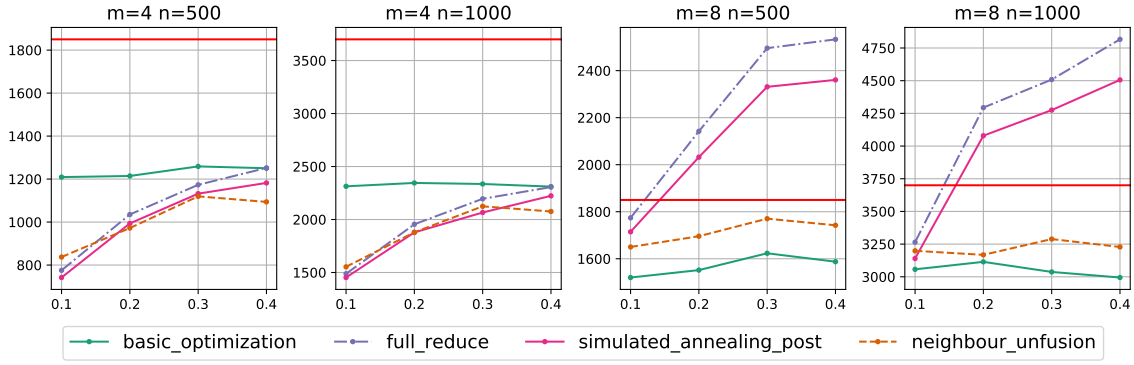


Figure 5.5: Weighed gate count for m qubit circuits with a gate depth of n and increasing T gate probability p_t optimized with different algorithms. The X-axis denotes the T gate probability, the Y-axis the weighed gate count C_w. The red line denotes the weighed gate count of the original circuit.

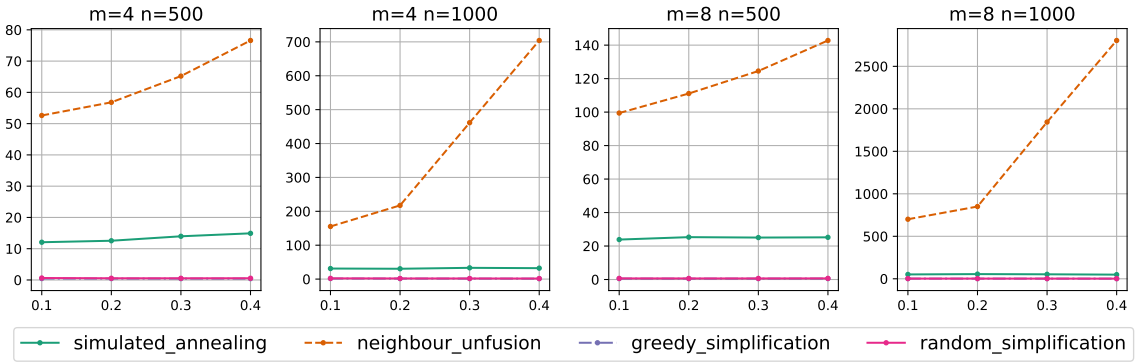


Figure 5.6: Average runtime of our algorithms for m qubit circuits with a gate depth of n and increasing T gate probability p_t. The X-axes denote the T gate probability, the Y-axes the time in seconds.

that the simulated annealing algorithm is the only algorithm which allows rule applications with arbitrary negative results for LCH and PH. However, the performance may improve when changing the number of iterations or the cooling factor.
Since the `neighbour_unfusion` algorithm outperforms the other algorithms in all cases, in Figure 5.5 we only compare this algorithm against the existing PyZX algorithms for clarity.

While for circuits with low T gate count, the `full_reduce` and the `simulated_annealing` post-processing algorithm sometimes yield better results, the `neighbour_unfusion` algorithm performs significantly better on 8 qubit circuits with T gate probability $> 10\%$. In this setting it is the only ZX-calculus based algorithm which does not *increase* circuit cost when compared to the original circuit. However, while the `basic_optimization` algorithm is the worst algorithm for 4 qubit circuits, it is the best algorithm for 8 qubit circuits. This suggests that also the new ZX-calculus based approaches perform worse when applied on larger circuits.

5.3.4 Runtime Evaluation

For evaluating the runtime of our algorithms we ran them with the same settings as in the previous section single threaded on a laptop computer with an Intel(R) Core(TM) M-5Y10c CPU @ 0.80GHz CPU unit. As we can see in in Figure 5.6, our algorithms have very different runtimes.

While the runtime of the greedy and the random algorithm is constantly in the range of a few seconds, the runtime of the `simulated_annealing` algorithm increases up to a minute and the runtime of the `neighbour_unfusion` algorithm increases up to almost an hour for 8 qubit circuits with 1000 gates. Moreover, while the runtime of the other algorithms seems to be only determined by the number of gates and the number of qubits, the runtime of the `neighbour_unfusion` algorithm also depends much on T gate probability. This is due to the calculation of the maximally delayed gflow needed for neighbour unfusion. Since the runtime of the gflow algorithm is dominated by the gaussian elimination on biadjacency matrices, the runtime increases when there are more connections, i.e., Hadamard wires, in a diagram. This is especially the case when we cannot remove many spiders with the local complementation and pivoting rule, i.e., when there are many T gates in a circuit. As already mentioned in Section 4.5.2 we can speed up this process by only modifying some relevant vertices after rule application instead of recalculating gflow but then our criteria for choosing a neighbour for unfusion sometimes fail and we end up in unextractable diagrams.

5.4 Evaluation on Benchmark circuits

In this section we test our algorithms against circuits that implement actual computing problems. First we use the QASMBench library for evaluating the algorithms on quantum circuits from various scientific research areas such as chemistry, machine learning and cryptography [19]. This serves as a general test environment to show how the algorithms behave on real quantum circuits. Then we compare our optimization algorithms to the Tpar benchmark circuits from [2]. This is a set of benchmark circuits which have been used by a number of papers in recent years to show the effectiveness of various quantum circuit optimization algorithms including a T gate reduction approach with ZX-calculus based phase teleportation [15]. We compare the algorithms against this approach and another up to date optimization algorithm presented in [22].

Circuit	Pre-Optimization			basic optimization			full reduce			sim annealing post			Our best Optimization			
	C_{QG}	C_{2QG}	C_w	C_{QG}	C_{2QG}	C_w	C_{QG}	C_{2QG}	C_w	C_{QG}	C_{2QG}	C_w	C_{QG}	C_{2QG}	C_w	Algorithm
basis-trotter$_4$	2598	582	7836	712	268	3124	574	190	2284	**569**	185	**2234**	641	**183**	2288	neighbour_unfusion
iswap$_2$	9	**2**	**27**	**7**	4	43	9	4	45	9	4	45	7	4	43	random_simplification
qaoa$_3$	**15**	6	69	21	6	75	21	**4**	**57**	21	**4**	**57**	21	**4**	**57**	simulated_annealing
variational$_4$	54	16	198	**40**	**12**	**148**	70	38	412	66	32	354	62	14	188	random_simplification
adder$_4$	23	**10**	113	**22**	**10**	**112**	49	21	238	49	21	238	41	15	176	random_simplification
dnn$_2$	514	42	892	168	28	420	171	29	432	171	31	450	**149**	**19**	**320**	simulated_annealing
bell$_4$	77	7	140	**20**	**5**	**65**	29	7	92	28	6	82	29	**5**	74	neighbour_unfusion
error-correctiond3$_5$	113	49	554	**19**	11	118	34	20	214	34	20	214	21	**10**	**111**	simulated_annealing
basis-test$_4$	110	46	524	74	27	317	70	22	268	64	**18**	226	**60**	**18**	**222**	neighbour_unfusion
cat-state$_4$	**4**	**3**	**31**	**4**	**3**	**31**	10	3	37	10	3	37	10	3	37	neighbour_unfusion
toffoli$_3$	**18**	**6**	**72**	**18**	**6**	**72**	36	9	117	36	9	117	34	6	88	random_simplification
qec-en$_5$	25	**10**	**115**	**17**	14	143	23	14	149	25	16	169	21	12	129	greedy_simplification
qrng$_4$	**4**	**0**	**4**	**4**	**0**	**4**	**4**	**0**	**4**	-	-	-	**4**	**0**	**4**	simulated_annealing
teleportation$_3$	**8**	**2**	**26**	**8**	**2**	**26**	10	2	28	10	2	28	10	2	28	simulated_annealing
wstate$_3$	34	9	115	**24**	**6**	**78**	38	14	164	40	14	166	34	**6**	88	neighbour_unfusion
linearsolver$_3$	27	**4**	63	**14**	**4**	**50**	17	**4**	53	17	**4**	53	17	**4**	53	random_simplification
basis-change$_3$	125	**10**	215	86	**10**	**176**	80	30	350	**76**	26	310	93	**10**	183	neighbour_unfusion
hs4$_4$	28	**4**	**64**	12	8	84	17	6	71	17	6	71	**10**	8	82	greedy_simplification
vqe-uccsd$_4$	332	88	1124	138	80	858	94	50	544	**86**	42	464	89	**33**	**386**	neighbour_unfusion
lpn$_5$	11	**2**	29	**3**	**2**	**21**	7	**2**	25	7	**2**	25	7	**2**	25	simulated_annealing
deutsch$_2$	5	**1**	14	**4**	**1**	**13**	6	**1**	15	6	**1**	15	6	**1**	15	simulated_annealing
grover$_2$	16	**2**	**34**	**6**	4	42	10	4	46	10	4	46	8	4	44	neighbour_unfusion
fredkin$_3$	**19**	**8**	**91**	**19**	**8**	**91**	42	17	195	41	14	167	38	9	119	neighbour_unfusion
quantumwalks$_2$	43	**3**	70	**30**	**3**	**57**	36	8	108	36	6	90	35	**3**	62	neighbour_unfusion

Table 5.1: Comparison of the optimization results from the *small* circuit class of the QASMBench library. The best results in their category are highlighted.

5.4.1 Results for the QASMBench circuits

The QASMBench circuit library is grouped into three circuit classes: small, medium and large sized circuits. While circuit depths for small circuits are usually in the range of 10-1000 gates, some circuits in the medium and especially in the large section contain more than 10000 gates. Due to the slow runtime of the `neighbour_unfusion` algorithm we excluded those circuits from our evaluation. This benchmark serves as a comparison between all PyZX algorithms. For each circuit we compare the `basic_optimization`, `full_reduce` and `simulated_annealing_post` algorithm against the best result of one of our own heuristic-based algorithms. Table 5.1 shows the results for the small circuit class and Table 5.2 shows the results for the middle circuit class. Note that we did not include the C_{TG} metric for counting the T gates since all of the circuits had been put through the `teleport_reduce` algorithm before the actual simplification and the T gate count remains the same. While for small circuits the original circuit or the circuit resulting from the `basic_optimization` have the best results in terms of weighed circuit cost, the circuits in the medium class are almost all best optimized by the `neighbour_unfusion` algorithm. This is most likely because many small sized circuits are already optimal in terms of gates and the optimization algorithms cannot improve them. However, this is not the case for medium sized circuits where our algorithms can improve the circuits a lot. Moreover, while the existing ZX-calculus based algorithms `full_reduce` and `simulated_annealing_post` yield the best results for the vqe-uccsd$_6$ circuit, in all other cases they are outperformed by our heuristic-based algorithms which indicates that the they are more suitable to optimize real world quantum circuits.

5.4.2 Tpar Benchmark

As a last step we compare our algorithms using the Tpar benchmark [2]. This set of circuits consists mostly of arithmetical problems such as adders, multipliers or different applications of the Toffoli gate. It first has been used to demonstrate the effectiveness of a T gate

Circuit	Pre-Optimization			`basic optimization`			`full reduce`			`sim annealing post`			Our best Optimization			
	C_{QG}	C_{2QG}	C_w	C_{QG}	C_{2QG}	C_w	C_{QG}	C_{2QG}	C_w	C_{QG}	C_{2QG}	C_w	C_{QG}	C_{2QG}	C_w	Algorithm
vqe-uccsd$_6$	3362	1052	12830	1464	938	9906	**420**	202	2238	421	**199**	**2212**	579	239	2730	`neighbour_unfusion`
bv$_{14}$	41	**13**	158	**28**	**13**	**145**	54	**13**	171	54	**13**	171	43	**13**	160	`greedy_simplification`
adder$_{10}$	**142**	65	727	143	61	692	208	97	1081	199	82	937	196	**51**	**655**	`neighbour_unfusion`
multiplier$_{15}$	574	246	2788	476	222	2474	470	247	2693	427	204	2263	**426**	**131**	**1605**	`random_simplification`
ising$_{10}$	480	90	1290	**340**	90	1150	445	163	1912	409	125	1534	360	**72**	**1008**	`neighbour_unfusion`
dnn$_8$	2320	192	4048	720	128	1872	752	152	2120	749	149	2090	**717**	**101**	**1626**	`neighbour_unfusion`
qft$_{15}$	540	210	2430	**449**	210	2339	685	309	3466	656	280	3176	530	**144**	**1826**	`neighbour_unfusion`
seca$_{11}$	182	84	938	**138**	61	687	215	112	1223	196	98	1078	185	**50**	**635**	`neighbour_unfusion`
qaoa$_6$	594	54	1080	**122**	**36**	**446**	159	57	672	160	60	700	146	**36**	470	`neighbour_unfusion`
sat$_{11}$	679	252	2947	580	252	2848	591	248	2823	585	242	2763	**546**	**160**	**1986**	`neighbour_unfusion`
multiply$_{13}$	98	**40**	458	**95**	**40**	**455**	222	103	1149	209	94	1055	171	**40**	531	`neighbour_unfusion`
simon$_6$	44	14	170	34	14	160	**15**	**4**	**51**	**15**	**4**	**51**	**15**	**4**	**51**	`simulated_annealing`
bb84$_8$	27	**0**	27	**9**	**0**	**9**	15	**0**	15	15	**0**	15	11	**0**	11	`greedy_simplification`

Table 5.2: Comparison of the optimization results from the *medium* circuit class of the QASMBench library. The best results in their category are highlighted.

optimization algorithm named Tpar. Since then the benchmark has been used to evaluate several other T gate optimization algorithms against each other including a ZX-calculus based approach using a combination of the `basic_optimization` and `teleport_reduce` algorithms [15]. We compare our algorithms to this ZX-calculus based approach and an optimization algorithm presented in [22] which outperforms the original Tpar algorithm and, to our knowledge, is one of the best up-to-date algorithms for reducing both two-qubit gate count and T gate count. It is worth mentioning that the results of the T gate counts in [15] were further reduced using the TODD algorithm. However, we observed that this algorithm increases two-qubit gate count significantly in most cases. Since we especially want to evaluate two-qubit gate count, we compare our results against the results from [15] without the application of the TODD algorithm. Table 5.3 compares our algorithms to the results from [22] and [15]. For each circuit we only compare the best result from any of the PyZX algorithms defined in 5.2 against the two benchmark results. While the T gates are optimized approximately the same, in most cases the optimization algorithm from [22] outperforms the ZX-calculus based algorithms in terms of total gate count and two-qubit gate count. Still, there are two circuits where our optimization algorithms generates the best result: the VBE-Adder$_3$ and the Mod-Mult$_{55}$ circuit. Moreover, the simulated annealing post-processing algorithm generates the best result for the Mod 5_4 circuit. For all other circuits the algorithm from [22] generates the best results but most of the time our algorithms come close those results especially for the two-qubit gate count. Compared to the best known ZX-calculus strategy from [15] the results the heuristic-based algorithms almost always have a lower total and two-qubit gate count. This indicates that while there is still room for improvement for ZX-calculus based approaches to reach the optimization degree of the leading optimization algorithms, our initial idea of using heuristics for reducing Hadmard wires instead of spiders seems to be a promising approach for ZX-calculus based quantum circuit optimization.

Circuit	Pre-Optimization			Ref [22] Post-Optimization			Ref [15] Post-Optimization			Best PyZX Post-Optimization			
	C_{QG}	C_{2QG}	C_{TG}	C_{QG}	C_{2QG}	C_{TG}	C_{QG}	C_{2QG}	C_{TG}	C_{QG}	C_{2QG}	C_{TG}	Algorithm
Mod 5_4	63	28	28	51	28	16	46	27	**8**	**29**	**15**	**8**	`simulated_annealing_post`
VBE-Adder$_3$	150	70	70	89	50	**24**	101	54	**24**	**87**	**42**	**24**	`neighbour_unfusion`
CSLA-MUX$_3$	170	80	70	**155**	**70**	64	156	75	**62**	**155**	74	**62**	`neighbour_unfusion`
CSUM-MUX$_3$	420	168	196	**266**	**140**	**84**	308	168	**84**	303	150	**84**	`neighbour_unfusion`
QCLA-Com$_7$	443	186	203	**284**	**132**	**95**	316	146	**95**	297	140	**95**	`random_simplification`
QCLA-Mod$_7$	884	382	413	**624**	**292**	**235**	717	324	237	709	312	237	`neighbour_unfusion`
QCLA-Adder$_{10}$	521	233	238	**399**	**183**	**162**	437	201	**162**	433	200	**162**	`neighbour_unfusion`
Adder$_8$	900	409	399	606	**291**	215	690	351	**173**	**592**	300	**173**	`neighbour_unfusion`
RC-Adder$_6$	200	93	77	**140**	**71**	**47**	155	**71**	**47**	159	**71**	**47**	`simulated_annealing`
Mod-Red$_{21}$	278	105	119	**180**	**77**	**73**	217	93	**73**	196	85	**73**	`neighbour_unfusion`
Mod-Mult$_{55}$	119	48	49	91	**40**	**35**	91	42	**35**	**90**	**40**	**35**	`greedy_simplification`
Toff-Barenco$_3$	58	24	28	**40**	**18**	**16**	50	22	**16**	53	20	**16**	`simulated_annealing_post`
Toff-NC$_3$	45	18	21	**35**	**14**	**15**	40	16	**15**	36	15	**15**	`neighbour_unfusion`
Toff-Barenco$_4$	114	48	56	**72**	**34**	**28**	95	44	**28**	90	39	**28**	`simulated_annealing_post`
Toff-NC$_4$	75	30	35	**55**	**22**	**23**	65	26	**23**	57	24	**23**	`neighbour_unfusion`
Toff-Barenco$_5$	170	72	84	**104**	**50**	**40**	140	66	**40**	121	54	**40**	`simulated_annealing_post`
Toff-NC$_5$	105	42	49	**75**	**30**	**31**	90	36	**31**	78	33	**31**	`neighbour_unfusion`
Toff-Barenco$_{10}$	450	192	224	**264**	**130**	**100**	365	176	**100**	333	156	**100**	`neighbour_unfusion`
Toff-NC$_{10}$	255	102	119	**175**	**70**	**71**	215	86	**71**	183	78	**71**	`neighbour_unfusion`
GF(2^4)-Mult	225	99	112	**187**	**99**	**68**	193	**99**	**68**	195	101	**68**	`random_simplification`
GF(2^5)-Mult	347	154	175	**296**	**154**	**115**	304	**154**	**115**	306	156	**115**	`greedy_simplification`
GF(2^6)-Mult	495	221	252	**403**	**221**	**150**	422	**221**	**150**	422	**221**	**150**	`neighbour_unfusion`
GF(2^7)-Mult	669	300	343	**555**	**300**	**217**	573	**300**	**217**	573	**300**	**217**	`neighbour_unfusion`
GF(2^8)-Mult	883	405	448	**712**	**405**	**264**	745	**405**	**264**	745	**405**	**264**	`neighbour_unfusion`

Table 5.3: Comparison of the optimization results from [22] against the PyZX based strategy from [15] and the best result from any PyZX strategy including our own approaches. The best results in each of the categories *Total gate count*, *two-qubit gate count* and *T gate count* are highlighted.

6 Conclusion and Future work

Optimizing quantum circuits is a main challenge in quantum computing as hardware and physical constraints of quantum computers put severe limits on the size of quantum circuits. ZX-calculus offers a graphical approach to optimize quantum circuits using ZX-diagrams which, due to their more open structure, allow optimizations that have no equivalent in quantum circuits. Since existing ZX-calculus based approaches have several drawbacks like not reducing two-qubit gates significantly in many cases, we focus on the following research question:

Which strategies can be used to further improve ZX-calculus based quantum circuit optimization?

As main idea we introduced heuristics in the ZX-diagram simplification algorithms which allow to estimate how rule applications change the number of two-qubit gates of the underlying quantum circuit changes by counting the so called *Hadamard wires* in ZX-diagrams. This turns ZX-diagram simplification into a classical search problem and we design various search algorithms which use the heuristics for guiding the simplification process towards minimal circuits. As a first step we implement algorithms for greedy search, random search and a search algorithm based on simulated annealing. While the simulated annealing algorithm shows no significant improvement, the other two simplification algorithms outperform the existing ZX-calculus algorithms most of the time, especially for larger non-Clifford circuits. The main obstacle for our simplification algorithms are the use of phase gadgets, i.e. spiders in YZ plane, in combination with the extraction algorithm necessary for converting ZX-diagrams back into quantum circuits. Although phase gadgets extend the possibilities of diagram simplification, we have to introduce restrictions on rule applications in order to keep diagrams extractable. Moreover, as the resolving of YZ spiders during the extraction algorithm generates new Hadamard wires, our heuristic-based approaches are not able to develop their full power because these new Hadamard wires cannot be taken into account during the simplification process.
Therefore, we develop a new simplification rule called *neighbour unfusion* as an alternative to phase gadgets which keeps all the extended simplification possibilities from using phase gadgets but does not generate spiders in other measurement planes. As a consequence we neither have to restrict rule applications nor does the extraction algorithm generate new Hadamard wires. The greedy algorithm based on neighbour unfusion shows the most promising results in terms of circuit optimization so far. For most quantum circuits this algorithm yields the best results of all ZX-calculus based approaches and although most of the time the current state-of-the-art algorithms not based on the ZX-calculus still outperform the neighbour unfusion algorithm, the results for two-qubit gates do not differ much.
However, a major drawback of the neighbour unfusion algorithm is its high runtime. This is due to the fact that before each optimization step the *maximally delayed gflow* has to be calculated which is necessary to find pairs of connected spiders operating on the same qubit. The complete source code of our heuristic-based algorithms can be found in the following

repository https://github.com/mnm-team/pyzx-heuristics which is based on a fork from the PyZX library as of June 2021.

Future work

In the following we want to mention some aspects which we consider promising topics for future work on our approaches and on general quantum circuit optimization using ZX-calculus. It can be further explored if and how far the Euler-rule is suitable for diagram simplification and the heuristic-based algorithms allow a variety of improvements such as lookahead simplification, topology-aware simplification and finding more applications of the neighbour unfusion rule.

Further evaluation of the Euler-rule

Although using the Euler-rule in order to eliminate 2-ary spiders may lead to spiders with complicated phases, it could serve as a means of eliminating non-Clifford spiders in some cases. For this task a mathematical library has to be included in the PyZX library to efficiently calculate the resulting angles of the Euler-rule. It would also be interesting if we can derive further simple rules from the Euler-rule which can be used for different simplification approches.

Lookahead simplification

As a general improvement for the greedy algorithms some lookahead strategy could be implemented which does not only evaluate which single rule application is best for optimizing the underlying quantum circuit but which combinations of rules up to a specified depth are best for optimization. We observed that rule applications which increase Hadamard wire count often lead to new possible applications which decrease Hadamard wire count even more. Therefore, it would be useful to rank the first rule application better which could be covered to some extent by lookahead strategies.

Using gflow for topology optimization

Our hypothesis for the neighbour unfusion algorithm states that using the maximally delayed gflow we can find pairs of connected spiders which get extracted to the same qubit. We think it is worth investigating if there is a general way to determine the qubit on which a spider operates using the gflow. If this is possible in an efficient way, we can optimize ZX-diagrams in terms of quantum circuit topology which means we can construct algorithms for quantum circuits where two-qubit gates are not allowed on any pair of qubits. We can either prohibit rule applications which generate connections between spiders whose qubits cannot be connected directly or weigh them with a negative factor making it unlikely for these rules to get selected.

Improvement of the neighbour unfusion algorithm

Using neighbour unfusion for diagram simplification shows the best results so far but there are possible improvements regarding runtime and effectivity. First, it can be investigated whether neighbour unfusion between connected spiders which do not get extracted to the

same qubit also preserves gflow. So far we see that in many cases the diagram becomes unextractable using the PyZX library but it remains unclear whether this is due to the missing support for spiders in XZ and YZ plane in PyZX or whether neighbour unfusion does break the gflow property in some cases. In both cases it is worth investigating on which other neighbour types spiders can be unfused while staying in XY plane as this leads to more possibilities for diagram simplification and therefore most likely to better overall results in quantum circuit optimization. Second, the runtime of the algorithm would be improved if there is another way of finding spider pairs corresponding to the same qubits without calculating the maximally delayed gflow.

List of Figures

Bibliography

[1] Scott Aaronson and Daniel Gottesman. “Improved simulation of stabilizer circuits”. In: *Physical Review A* 70.5 (2004), p. 052328.

[2] Matthew Amy, Dmitri Maslov, and Michele Mosca. “Polynomial-time T-depth optimization of Clifford+ T circuits via matroid partitioning”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.10 (2014), pp. 1476–1489.

[3] Miriam Backens. “The ZX-calculus is complete for stabilizer quantum mechanics”. In: *New Journal of Physics* 16.9 (2014), p. 093021.

[4] Miriam Backens et al. “There and back again: A circuit extraction tale”. In: *Quantum* 5 (2021), p. 421.

[5] Sergey Bravyi and Alexei Kitaev. “Universal quantum computation with ideal Clifford gates and noisy ancillas”. In: *Physical Review A* 71.2 (2005), p. 022316.

[6] Katherine L Brown, William J Munro, and Vivien M Kendon. “Using quantum computers for quantum simulation”. In: *Entropy* 12.11 (2010), pp. 2268–2307.

[7] Bob Coecke and Aleks Kissinger. “Picturing quantum processes”. In: *International Conference on Theory and Application of Diagrams*. Springer. 2018, pp. 28–31.

[8] Bob Coecke et al. “Kindergarden quantum mechanics graduates (... or how I learned to stop gluing LEGO together and love the ZX-calculus)”. In: *arXiv preprint arXiv:2102.10984* (2021).

[9] Ross Duncan et al. “Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus”. In: *Quantum* 4 (2020), p. 279.

[10] Austin G Fowler et al. “Surface codes: Towards practical large-scale quantum computation”. In: *Physical Review A* 86.3 (2012), p. 032324.

[11] Lov K Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[12] Matthias Homeister. *Quantum Computing verstehen: Grundlagen-Anwendungen-Perspektiven*. Springer-Verlag, 2015.

[13] Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. “A complete axiomatisation of the ZX-calculus for Clifford+ T quantum mechanics”. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 2018, pp. 559–568.

[14] Aleks Kissinger and John van de Wetering. “Pyzx: Large scale automated diagrammatic reasoning”. In: *arXiv preprint arXiv:1904.04735* (2019).

[15] Aleks Kissinger and John van de Wetering. “Reducing T-count with the ZX-calculus”. In: *arXiv preprint arXiv:1903.10477* (2019).

[16] Aleksei Yur'evich Kitaev. "Quantum computations: algorithms and error correction". In: *Uspekhi Matematicheskikh Nauk* 52.6 (1997), pp. 53–112.

[17] Benjamin P Lanyon et al. "Towards quantum chemistry on a quantum computer". In: *Nature chemistry* 2.2 (2010), pp. 106–111.

[18] Soonchil Lee et al. "The Cost of Quantum Gate Primitives." In: *Journal of Multiple-Valued Logic & Soft Computing* 12 (2006).

[19] Ang Li et al. "QASMBench: A Low-level QASM Benchmark Suite for NISQ Evaluation and Simulation". In: *arXiv preprint arXiv:2005.13018* (2021).

[20] Dmitri Maslov et al. "Quantum circuit simplification and level compaction". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.3 (2008), pp. 436–444.

[21] Mehdi Mhalla and Simon Perdrix. "Finding optimal flows efficiently". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2008, pp. 857–868.

[22] Yunseong Nam et al. "Automated optimization of large quantum circuits with continuous parameters". In: *npj Quantum Information* 4.1 (2018), pp. 1–12.

[23] Michael A Nielsen and Isaac L Chuang. "Quantum Computation and Quantum Information". In: *Quantum Computation and Quantum Information* (2010).

[24] Joe O'Gorman and Earl T Campbell. "Quantum computation with realistic magic-state factories". In: *Physical Review A* 95.3 (2017), p. 032338.

[25] Aditya K Prasad et al. "Data structures and algorithms for simplifying reversible circuits". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 2.4 (2006), pp. 277–293.

[26] Robert Raussendorf and Hans J Briegel. "A one-way quantum computer". In: *Physical Review Letters* 86.22 (2001), p. 5188.

[27] Peter W Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[28] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. "Efficient algorithm to recognize the local Clifford equivalence of graph states". In: *Physical Review A* 70.3 (2004), p. 034302.

[29] Renaud Vilmart. "A near-minimal axiomatisation of zx-calculus for pure qubit quantum mechanics". In: *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE. 2019, pp. 1–10.

[30] John van de Wetering. "ZX-calculus for the working quantum computer scientist". In: *arXiv preprint arXiv:2012.13966* (2020).