Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Quantum Circuit Optimization for Circuit Cutting

Duc Anh Nguyen

Course of Study:	Informatik Master of Science
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	Marvin Bechtold, M.Sc., Dipl.-Ing. Alexander Mandl
Commenced:	December 6, 2024
Completed:	June 6, 2025

Abstract

In this work, we present an optimization framework aimed at reducing the sampling overhead associated with circuit cutting in quantum computing. We design a complete pipeline that integrates circuit rewriting, overhead evaluation via qiskit-addon-cutting, and heuristic search using simulated annealing and genetic algorithms. To enable more efficient circuit representations, we propose six rewriting techniques, including methods that exploit gate commutativity and ZX-Calculus-based transformations. The latter allows flexible rewrites through diagrammatic rules such as simplification, local complementation, and pivoting. Our experimental evaluation on benchmark circuits demonstrates that ZX-based strategies significantly reduce sampling overhead, while pure commutativity-based approaches show limited gains. To improve scalability, we introduce a windowed rewriting approach that targets random circuit sections, offering further performance benefits. Comparative results reveal that simulated annealing consistently finds lower-overhead solutions, whereas the genetic algorithm provides superior runtime performance through parallel evaluation. Our findings highlight the value of structured rewriting and search-based optimization in preparing circuits for circuit cutting.

Kurzfassung

In dieser Arbeit präsentieren wir ein Optimierungsverfahren zur Reduzierung des Sampling Overheads, der beim Circuit Cutting für Quantenschaltkreise ensteht. Wir entwickeln eine vollständigen Prozess, welches das Umschreiben von Quantenschaltkreisen, die Evaluierung des Overheads durch das qiskit-addon-cutting, sowie heuristische Suchverfahren wie Simulated Annealing und genetische Algorithmen integriert. Zur Erzeugung effizienterer Schaltkreisrepräsentationen stellen wir sechs Rewriting-Techniken vor, darunter Methoden, die die Kommutativität von Quantengatter ausnutzen, sowie Transformationen auf Basis des ZX-Calculus. Das ZX-Calculus ermöglicht durch diagrammatische Regeln wie Graphvereinfachung, Local Complementation und Pivoting flexible Umschreibungen von Quantenschaltkreisen. Unsere experimentelle Auswertung an Benchmark-Schaltkreisen zeigt, dass ZX-basierte Strategien den Sampling Overhead deutlich reduzieren, während rein kommutativitätsbasierte Ansätze nur begrenzte Verbesserungen vorweisen. Zur Verbesserung der Skalierbarkeit führen wir einen „Windowed Rewriting“-Ansatz ein, bei dem zufällige Abschnitte des Schaltkreises gezielt umgeschrieben werden, was weitere Optimierungen ermöglicht. Ein Vergleich der Suchverfahren zeigt, dass Simulated Annealing in der Regel Lösungen mit geringerem Overhead findet, während der genetische Algorithmus durch parallele Auswertung eine bessere Laufzeitleistung erzielt. Unsere Ergebnisse zeigen, dass strukturierte Umschreibungen und suchbasierte Optimierungsverfahren eine effektive Methode zur Vorbereitung von Quantenschaltkreisen auf das Circuit Cutting darstellen.

Contents

1 Introduction **17**

2 Background **19**
2.1 Qubits, Gates and Circuits 19
2.2 Circuit Cutting 22
2.3 ZX-Calculus 24
2.3.1 Graph Representation 24
2.3.2 Rules 26
2.3.3 Circuit Extraction 29

3 Methodology **31**
3.1 Rewriting Quantum Circuits 32
3.1.1 Cancel Commutative Gates 32
3.1.2 Swap Commutative Gates 33
3.1.3 ZX Simplifications 34
3.1.4 ZX Congruences 35
3.1.5 Random Section Rewrite 37
3.2 Optimization Techniques 39
3.2.1 Objective Function 39
3.2.2 Simulated Annealing 40
3.2.3 Genetic Algorithm 42

4 Experiments and Results **45**
4.1 Implementation 45
4.2 Isolated Rewriting 46
4.3 Combined Rewriting 47
4.4 Simulated Annealing vs Genetic Algorithm 49

5 Discussion **51**

6 Related Works **53**

7 Conclusion **55**

Bibliography **57**

A Appendix **61**

List of Figures

2.1 Example quantum circuit with 3 qubits. 22
2.2 Building a ZX-diagram from a given quantum circuit. 25
2.3 Basic ZX-rules. 26
2.4 ZX-diagram and its graph-like form. 29

3.1 Circuit optimization pipeline. 32
3.2 General genetic algorithm process for optimizing quantum circuits in the circuit cutting context. 42

4.1 Implemented circuit optimization pipeline. 46
4.2 Comparing simulated annealing and genetic algorithm in their effectiveness of reducing sampling overhead. 49
4.3 Runtime comparison between simulated annealing and genetic algorithm for circuits with different qubit counts. 50

A.1 VBE-Adder-3 circuit with cut placements suggested by qiskit-addon-cutting. . . . 61
A.2 Optimized VBE-Adder-3 circuit and suggested cut placements. 62

List of Tables

2.1 Different types of cuts for operations and corresponding sampling overheads. . . 23

4.1 Benchmarking circuits with rewriting techniques used in isolation. 47
4.2 Benchmarking circuits with all rewriting strategies used in combination. 48

List of Algorithms

3.1 Simulated annealing 40
3.2 Genetic algorithm 43

List of Abbreviations

GA genetic algorithm. 42

NISQ noisy intermediate-scale quantum. 17

QPD quasiprobability decomposition. 22

QPS qubits per subcircuit. 47

QPU quantum processing unit. 22

SA simulated annealing. 40

1 Introduction

Quantum computers operate in a fundamentally different way from classical computers by taking advantage of key principles of quantum mechanics, such as superposition and entanglement [NC10]. These properties enable quantum systems to process and represent information in ways that classical systems cannot, allowing quantum computers to tackle certain problems much more efficiently. Some of the most promising areas for quantum computing include cryptography, chemistry, optimization, and machine learning, where current computers struggle with the growing size and complexity of calculations [IHI+24].

Famous quantum algorithms such as Shor's algorithm for integer factorization [Sho97] and Grover's algorithm for unstructured database search [Gro96] have provided compelling theoretical evidence of these advantages. Shor's algorithm is particularly impactful as it provides a polynomial-time approach to factoring large integers, a task widely believed to be computationally infeasible for classical computers and poses a security threat to many current cryptographic protocols. Grover's algorithm, while offering a quadratic speedup, showcases the power of amplitude amplification in accelerating brute-force search problems. These algorithms serve as benchmarks for the immense promise of quantum computation.

Despite this theoretical potential, realizing practical quantum algorithms remains a difficult challenge. Quantum computing hardware is still in an early stage of development, and current devices face substantial limitations. The most advanced systems available today are noisy intermediate-scale quantum (NISQ) devices, which typically support only tens to a few hundred qubits [Pre18]. These systems are constrained by short coherence times, high gate error rates, limited qubit count, and an absence of fault-tolerant error correction [LB20]. As a result, executing deep and complex quantum circuits reliably is currently beyond reach.

A promising approach to overcome the limitations of current quantum hardware is circuit cutting [PHOW20; SPS25]. This method decomposes a large quantum circuit into smaller, more manageable subcircuits by introducing strategic cuts through wires and gates. These subcircuits can then be executed either sequentially on a single smaller NISQ device or in parallel across multiple quantum processors. Once executed, their results are classically combined to approximate the output of the original full circuit.

However, while circuit cutting enables computations beyond hardware limits, it incurs significant overhead, as the total number of required executions (also called shots) for each created subcircuit increases exponentially with each additional cut introduced [SPS25]. This is also known as the sampling overhead. The impact on overhead varies with the type of cut and the specific gates involved, making it important to determine the most effective cut configuration. Research has already addressed the challenge of identifying and reducing the sampling overhead in quantum circuits. Approaches typically focus on solving various partitioning problems and extracting efficient subcircuits, both of which aim to minimize the number of required cuts and thereby reduce the

associated sampling overhead [BPK23]. The open question, however, is whether it is possible to modify or optimize the input circuit itself to reveal new, advantageous cut positions that would further reduce the overhead.

This is the main motivation for the thesis. We formally define our research objective as

Optimizing Quantum Circuits for Circuit Cutting,

where the goal is to transform a given input circuit into an equivalent form that yields a reduced sampling overhead compared to the original, when circuit cutting is applied. To address this, we explore a range of circuit rewriting techniques, utilizing gate commutativity approaches and the ZX-Calculus framework and systematically evaluate their impact on circuit cutting overhead. Furthermore, we propose a comprehensive optimization pipeline tailored specifically for improving quantum circuits in the context of circuit cutting.

The remainder of this thesis is structured as follows: Chapter 2 provides the theoretical foundation by covering essential concepts of quantum computing, circuit cutting, and the ZX-Calculus framework. Chapter 3 presents the proposed rewriting techniques and optimization strategies in detail. We evaluate the effectiveness of our approach using selected benchmark circuits in Chapter 4 and discuss the results in Chapter 5. In Chapter 6, we relate our findings to existing research, and Chapter 7 concludes the thesis.

2 Background

This chapter introduces the foundational concepts of quantum computing. We begin by explaining the essential building blocks of quantum computation such as qubits, quantum gates, and circuits, followed by a detailed explanation of the circuit cutting technique. In addition, we cover the basics of ZX-Calculus, which serves as a key framework for the rewriting methods discussed in later chapters.

2.1 Qubits, Gates and Circuits

In this section, we adopt the notation and terminology used by Nielsen and Chuang [NC10] to describe the basic concepts of quantum computing. A qubit (quantum bit) is the fundamental unit of quantum information, analogous to the bit in classical computing. However, unlike a classical bit, which can only represent a ”0” or a ”1”, a qubit can exist in a superposition of both states simultaneously. Qubit states are generally described in the bra-ket notation (also known as Dirac notation), which is a standard and powerful way to represent quantum states and linear operators. Mathematically, a single qubit state can be represented as

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

where α and β are complex numbers representing the probability amplitudes of the qubit being in the $|0\rangle$ and $|1\rangle$ states with $|\alpha|^2 + |\beta|^2 = 1$. Furthermore $|0\rangle$ and $|1\rangle$ are basis states (computational basis) describing the vectors

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

The set $\{|0\rangle\,, |1\rangle\}$ is also referred to as the Z basis. Another frequently used basis is the X basis, given by $\{|+\rangle\,, |-\rangle\}$ with

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Multi-qubit states are defined by combining the states of individual qubits using the tensor product. Let us consider a system of the two qubits, with the first qubit being in state $|\psi_1\rangle = \alpha_1\,|0\rangle + \beta_1\,|1\rangle$ and the the second qubit being in state $|\psi_2\rangle = \alpha_2\,|0\rangle + \beta_2\,|1\rangle$. The state of the two-qubit system $|\Psi\rangle$ is given by the tensor product:

$$
\begin{aligned}
|\Psi\rangle &= |\psi_1\rangle \otimes |\psi_2\rangle = (\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes (|\psi_2\rangle = \alpha_2 |0\rangle + \beta_2 |1\rangle) \\
&= \alpha_1\alpha_2(|0\rangle \otimes |0\rangle) + \alpha_1\beta_2(|0\rangle \otimes |1\rangle) + \beta_1\alpha_2(|1\rangle \otimes |0\rangle) + \beta_1\beta_2(|1\rangle \otimes |1\rangle) \\
&= \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_2 |01\rangle + \beta_1\alpha_2 |10\rangle + \beta_1\beta_2 |11\rangle \\
&= \begin{pmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{pmatrix}
\end{aligned}
$$

While this state is described by an vector from $\mathbb{C}^4$, a n-qubit system is represented by a vector from $\mathbb{C}^{2^n}$.

Quantum gates are the building blocks of quantum circuits, just like classical logic gates (like AND, OR, NOT) are for classical computers. They are unitary transformations that manipulate the state of one or more qubits. Quantum gates operating on single qubits are described by 2×2 matrices, while multi-qubit gates acting on n qubits are unitary $2^n \times 2^n$ matrices.

Single-qubit Gates

The Hadamard gate is a fundamental gate, which imposes superposition on computational basis states. It is described by the matrix

$$
H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},
$$

which applied on the $|0\rangle$ and $|1\rangle$ states lead to

$$
\begin{aligned}
H |0\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\
H |1\rangle &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).
\end{aligned}
$$

The quantum *NOT* gate (or X gate) is represented by the following unitary matrix:

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

Given an state $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$, applying the X gate flips the computational basis state:

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

The Z gate

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

introduces a phase shift to the $|1\rangle$ state while leaving the $|0\rangle$ state unchanged:

$$Z|\psi\rangle = \alpha|0\rangle - \beta|1\rangle .$$

Transformation of qubit states can also be viewed as rotations around the bloch sphere [NC10]. The rotation gates

$$R_X(\theta) = e^{-i\theta X/2} = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$$
$$R_Z(\theta) = e^{-i\theta Z/2} = \begin{pmatrix} \cos(\theta/2) - i\sin(\theta/2) & 0 \\ 0 & \cos(\theta/2) + i\sin(\theta/2) \end{pmatrix}$$

will apply rotations with angle $\theta \in [0, 2\pi)$ on a given quantum state around the X and Z axis respectively.

Multi-qubit gates

Multi-qubit gates are unitary transformations on qubit states involving two or more qubits. Two important two-qubit gates are the Controlled-X (CNOT or also CX) gate and the Controlled-Z (CZ) gate:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$
$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Controlled means in this case if the first qubit (control qubit) is in state $|1\rangle$, then a X or Z gate will be applied on the second qubit (target qubit). Given a two-qubit state $|\psi\rangle = \alpha_1|00\rangle + \alpha_2|01\rangle + \alpha_3|10\rangle + \alpha_4|11\rangle$, applying a $CNOT$ or CZ gate will result in

$$CNOT\,|\psi\rangle = \alpha_1\,|00\rangle + \alpha_2\,|01\rangle + \alpha_3\,|11\rangle + \alpha_4\,|10\rangle\,,$$
$$CZ\,|\psi\rangle = \alpha_1\,|00\rangle + \alpha_2\,|01\rangle - \alpha_3\,|10\rangle - \alpha_4\,|11\rangle\,.$$

Quantum Circuits

Quantum circuits are constructed by combining these quantum gates in a specific sequence to implement a desired quantum algorithm. In order to visualize quantum circuits, people often use diagrams.

Multiple parallel horizontal lines or wires are used to represent each qubit in a system. Generally those wires are labeled on the left (e.g. $q_0, q_1, |0\rangle, |1\rangle$) to indicate the initial state or the qubit's identifier.

Quantum gates are represented as boxes or symbols drawn on the qubit wires. The type of gate is usually indicated by a letter or a specific symbol inside the box. Multi-qubit gates span over multiple qubit wires, while single-qubit operations are only applied on one line. The order of operations is read from left to right, meaning gates on the left are applied before the gates on the right. A concrete example is depicted in Fig. 2.1.

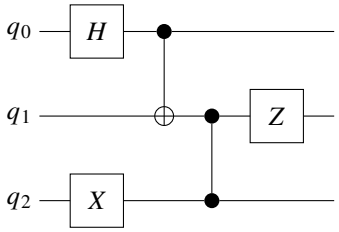


Figure 2.1: Example quantum circuit with 3 qubits. First a Hadamard gate and an X gate are applied to qubits q_0 and a q_2. Then a CNOT gate is applied to q_0 and q_1, with q_0 being the control and q_1 being the target qubit. Finally, a CZ gate is applied to q_1 and q_2.

2.2 Circuit Cutting

Circuit cutting is a technique used in quantum computing to split a large quantum circuit into smaller subcircuits that can be executed separately on quantum processing units (QPUs) with smaller qubit counts [PHOW20; SPS25].

Circuit cutting can be realized by using the quasiprobability simulation [BPS23]: Suppose we have an unitary operation $\mathcal{U}$ (i.e. a quantum gate) that cannot be physically executed on our quantum computer due to limiting qubit count constraints. Instead, we have access to a set of local operations $\mathcal{S}$ which can be run on the available hardware. The objective of quasiprobability simulation is to simulate the expected outcome of the desired operation $\mathcal{U}$ by only using such local operations in $\mathcal{S}$. This is accomplished by using quasiprobability decomposition (QPD).

The QPD for $\mathcal{U}$ is given by

Operation (Gate)	γ^2
CS, CSDG, CSX	$3 + 2\sqrt{2} \approx 5.828$
CX, CY, CZ, CH, ECR	3^2
iSWAP, DCX, SWAP	7^2
RXX, RYY, RZZ, RZX	$(1 + 2\lvert\sin(\theta)\rvert)^2$
CRX, CRY, CRZ, CPhase	$(1 + 2\lvert\sin(\theta/2)\rvert)^2$
Wire Cut	4^2

Table 2.1: Different types of cuts for operations and corresponding sampling overheads [BCE+24].

$$\mathcal{U} = \sum_i a_i \mathcal{F}_i, \tag{2.1}$$

where $\mathcal{F}_i \in \mathcal{S}$ are local operations and $a_i \in \mathbb{R}$ are corresponding coefficients. It comes with a cost tough: In order to reconstruct the proper expectation value from the original circuit the number of shots needed to achieve a certain accuracy grows exponentially. This is also known as the sampling overhead of a QPD and it is described by the γ-factor squared of an operation $\mathcal{U}$, denoted by $\gamma_S(\mathcal{U})^2$ [BPS23]. Formally, the γ-factor is defined as

$$\gamma_{\mathcal{S}}(\mathcal{U}) = \min \left\{ \sum_i^m |a_i| : \mathcal{U} = \sum_i^m a_i \mathcal{F}_i \text{ where } m \geq 1,\ \mathcal{F}_i \in \mathcal{S} \text{ and } a_i \in \mathbb{R} \right\}. \tag{2.2}$$

The γ-factor depends on what type of cuts are used in the decomposition. Currently, there are two types of possible cuts used in circuit cutting: wire cuts and gate cuts. A wire cut (also known as a time-like cut) is a technique used to split a quantum circuit by cutting along a single qubit's wire. This effectively divides the circuit into two parts at a specific point in time for that qubit. Gate cuts involve splitting a circuit by severing multi-qubit gates, such as a CNOT or CZ. Generally a mix of wire cuts and gate cuts is used when we perform circuit cutting. The wire cut has a sampling overhead $\gamma^2 = 4^2$, while gate cuts for CNOT or CZ have the overhead of $\gamma^2 = 3^2$. Table 2.1 shows different types of cuts and their respective overheads.

Assuming we intend to cut a circuit and have identified a suitable decomposition involving n non-local gates $U_1, \ldots, U_n$, we can compute the total sampling overhead by taking the product over all individual overheads introduced by each cut:

$$\text{total sampling overhead} = \prod_{i=1}^{n} \gamma(U_i)^2 \tag{2.3}$$

There are several methods for identifying optimal cut configurations that minimize overhead costs. Most approaches involve converting quantum circuits into corresponding directed graphs, which can then be processed using optimization solvers [BPK23; TTS+21]. While the internal workings of these solvers are not the primary focus of this work, we make use of their functionality through

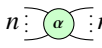
existing implementations provided by available libraries. As an example, qiskit-addon-cutting [BCE+24] provides a `find_cuts` function, which automatically cuts a quantum circuit. In later chapters, we call this function and calculate the resulting sampling overhead.

2.3 ZX-Calculus

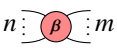
ZX-Calculus is a graphical language for reasoning about quantum circuits and quantum mechanics [Wet20]. It represents quantum computations using a diagrammatic notation, where quantum operations are expressed as interconnected nodes instead of traditional circuit elements like gates and qubits. It is particularly useful for simplifying, optimizing, and reasoning about quantum computations in a more intuitive way. For our use case, we leverage this framework to convert quantum circuits into flexible ZX-diagrams, apply various rule-based rewrites, and extract equivalent circuits that may be better suited for circuit cutting.

2.3.1 Graph Representation

Quantum circuits can be represented by ZX-diagrams, which are built upon two fundamental elements: Z-spiders (green nodes) and X-spiders (red nodes). These nodes describe operations in the Z and X bases, respectively, and can have any number of inputs and outputs, which do not need to match. They represent the following linear maps:



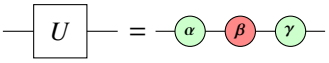
$$:= |0\rangle^{\otimes n} \langle 0|^{\otimes m} + e^{i\alpha} |1\rangle^{\otimes n} \langle 1|^{\otimes m}$$



$$:= |+\rangle^{\otimes n} \langle +|^{\otimes m} + e^{i\beta} |-\rangle^{\otimes n} \langle -|^{\otimes m}$$

Each spider can also carry a phase angle $\alpha \in [0, 2\pi)$, indicating a quantum rotation. A blank spider represents a spider with phase zero. Spiders are connected by wires, which represent the quantum circuit wires connecting classical gates. We can describe standard circuit single-qubit rotations around the X and Z axis with single input and output spiders:

Since any single-qubit operation can be decomposed into rotations around the X and Z axis [NC10], we can represent those operations by applying Z, X and Z-spiders in series:



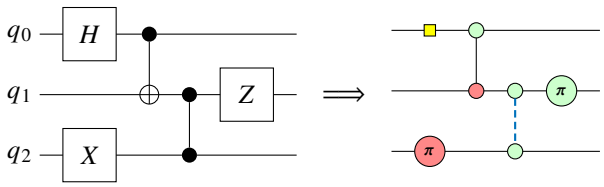
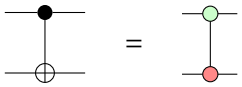


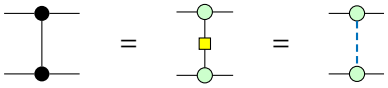
Figure 2.2: Building a ZX-diagram from a given quantum circuit.

The Hadamard gate is frequently used in ZX-diagrams. To represent this gate, we use either a simple yellow box with single input and output wires, or a blue dashed line. We also call this a Hadamard edge:

The CNOT circuit is represented by connecting a X-spider and Z-spider on two different wires:

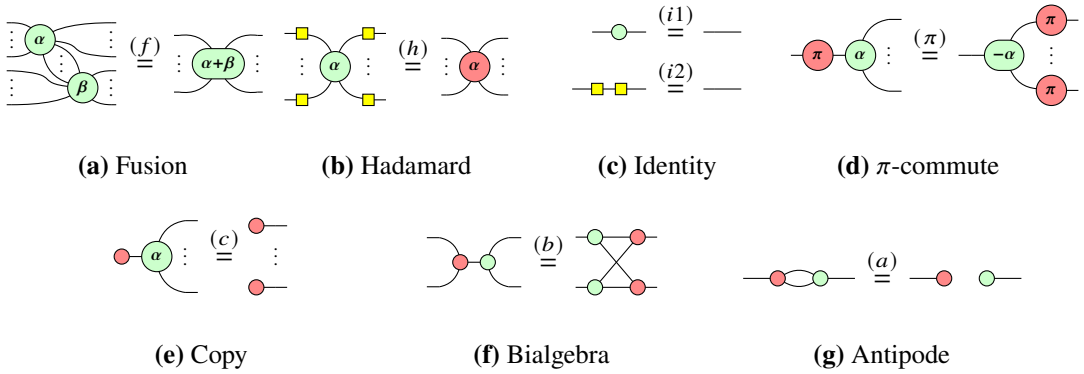


The CZ gate is constructed in a similar way but with a Hadamard edge as a connector:



These are the basic building blocks to translate standard quantum circuits into ZX-diagrams. An explicit example is shown in Figure 2.2. Because single-qubit rotations combined with the CNOT gate form a universal gate set [NC10], any quantum circuit can be represented as a ZX-diagram. However, it is important to note that a ZX-diagram may differ from the corresponding quantum circuit by a global phase [Wet20], which is irrelevant for our purposes.

2.3.2 Rules



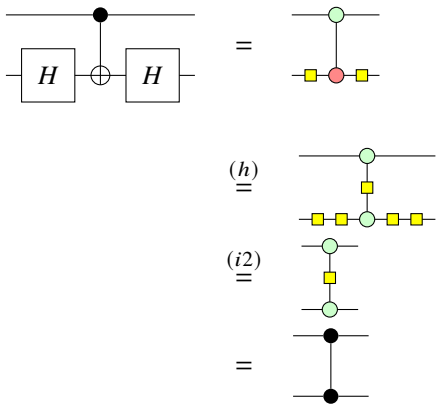
(a) Fusion **(b)** Hadamard **(c)** Identity **(d)** π-commute

(e) Copy **(f)** Bialgebra **(g)** Antipode

Figure 2.3: Basic ZX-rules. The color of all spiders can be interchanged.

ZX-Calculus provides a set of graphical rewrite rules that allow ZX-diagrams to be transformed into equivalent forms [Wet20]. These rewrite rules serve as the foundation for manipulating diagrams and include key operations such as spider fusion (Figure 2.3a), where two adjacent spiders of the same color (either Z or X) connected by a wire are merged into a single spider, with their phases summed. Other fundamental rules include the Hadamard rule (Figure 2.3b), which changes the color of a spider surrounded by Hadamard nodes on each edge, and the copy rule (Figure 2.3e), which expresses the duplication behavior of certain quantum operations. Additional rules, such as identity removal, Hopf-rule (also called Antipode rule), and π-rule, allow for further simplification and restructuring of diagrams (see Figure 2.3).

As an example, let us perform some valid operations on following circuit:



The two-qubit circuit with a CNOT and two Hadamards is transformed into a ZX-diagram. Then we apply the Hadamard rule to change the color of the X-spider by adding Hadamard nodes on each edge. Afterwards the sequential Hadamard nodes cancel each other out. The final obtained circuit is the CZ gate.

An important aspect of these transformations is that they are topological, meaning the semantics of a ZX-diagram are determined solely by how its components are connected, regardless of their spatial arrangement or layout in the drawing [Wet20]. ZX-diagrams can be reshaped, contracted, or rearranged visually, without altering the computation they represent.

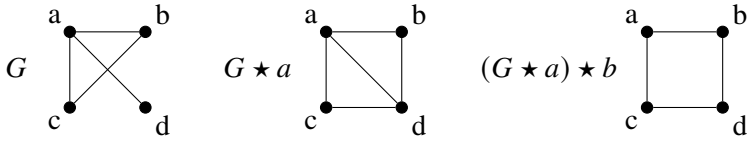
Local Complementation and Pivoting

Local Complementation and *Pivoting* are two graph-theoretic operations that act on graphs by locally transforming the connectivity between nodes [DKPW20; DP09]. We will use these methods in later chapters to transform ZX-diagrams and extract multiple equivalent quantum circuits.

Given a undirected graph $G = (V, E)$, a vertex $u \in V$ and its neighborhood $N(u) = \{v \in V \mid (v, u) \in E\}$: The local complementation of G about u, also written as $G \star u$, is performed by complementing the edge connectivity of all vertices in the neighborhood of u [DP09]. Formally, local complementation can be described as $G \star u = (V, E')$ with

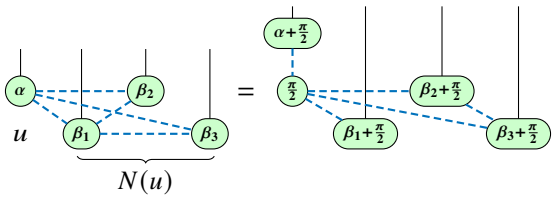
$$E' = (E \cup \{(v, w) \mid v, w \in N(u) \wedge (v, w) \notin E\}) \setminus \{(p, q) \mid p, q \in N(u) \wedge (p, q) \in E\}. \quad (2.4)$$

Let us define the graph $G = (\{a, b, c, d\}, \{(a, b), (a, c), (a, d), (b, c)\})$ and perform following local complementations:



The neighborhood of a is $\{b, c, d\}$ and we see that b is connected to c, while other neighbors are left unconnected. The local complementation $G \star a$ is constructed by removing the edge (b, c) and inserting the edges (b, d) and (c, d). Now we perform the local complementation $(G \star a) \star b$: The neighborhood $N(b)$ is $\{a, d\}$ and since those neighbors are connected, we remove the edge (a, d) from the current graph.

We see that local complementation can produce multiple graphs with varying node connectivity, depending on the chosen node that we do this operation about. This method can also be applied to ZX-diagrams, but additional conditions need to be fulfilled [Kru22]: Given an graph-like ZX-diagram (definition is given in Section 2.3.3), we can perform a local complementation about a Z-spider u with phase α by applying a Z-spider with phase $\alpha + \pi/2$ to this node and connect them via a Hadamard edge. Additionally, the phase of u is updated to $\pi/2$ and an extra $\pi/2$ is added to every phase of the spiders in the neighborhood. With that we can complement the edges as described in Equation (2.4):



(lc)

Pivoting can be seen as applying multiple local complementations on a graph about two vertices [DKPW20]. The pivot of an undirected graph $G = (V, E)$ along an edge $(u, v) \in E$, denoted by $G \wedge (u, v)$, is defined as the series of local complementations $G \star u \star v \star u$. Note that the ordering of u and v does not matter since they both lead to the same final graph: $G \star u \star v \star u = G \star v \star u \star v$.

Instead of applying local complementations, we can also achieve pivoting by following method:

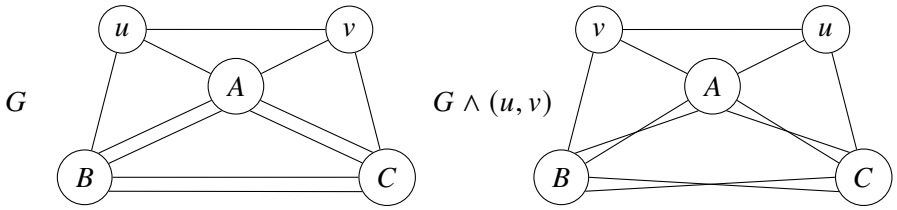
First, vertices u and v are switched in the graph. Then, three subsets related to the vertices u and v are defined as

(A) $N_G(u) \cap N_G(v)$: the common neighborhood of u and v,

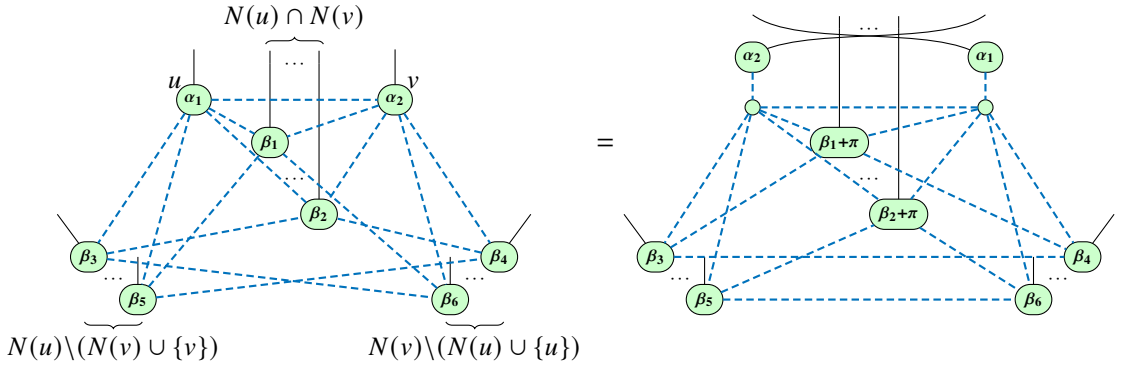
(B) $N_G(u) \backslash (N_G(v) \cup \{v\})$: the exclusive neighborhood of u,

(C) and $N_G(v) \backslash (N_G(u) \cup \{u\})$: the exclusive neighborhood of v.

The last step is to complement the edge connectivity between all nodes in those three subsets:



To perform a pivot operation on a ZX-diagram, a new Z-spider is created for each of the two original spiders that define the pivot edge, and these new spiders are connected by Hadamard edges. The input wires and original phases are transferred to these new Z-spiders, but switched. The phases of the original pivot spiders are then set to zero, and an additional phase of π is added to each spider in the common neighborhood [Kru22]:



(p)

2.3.3 Circuit Extraction

Compared to translating a circuit into an equivalent ZX-diagrams, converting a ZX-diagram into a quantum circuit is not a straightforward task: Due to the nature of ZX-calculus and the allowed operations, there are ZX-diagrams, which do not hold a valid quantum circuit representation [Wet20]. Algorithms exist that can automatically extract circuits from ZX-diagrams [DKPW20]. However, the current methods are not well optimized: In some cases, the number of gates in the extracted circuit exceeds that of the original, making them less suitable for efficient circuit cutting and evaluation in our context.

To perform graph-theoretic operations such as those required for circuit extraction, local complementation, and pivoting, it is sometimes necessary to convert ZX-diagrams into a more restrictive structure known as a *graph-like* diagram. A ZX-Diagram is graph-like if it fulfills following conditions:

1. All spiders are Z-spiders.
2. Z-spiders are only connected via Hadamard edges.
3. There are no parallel Hadamard edges or self-loops.
4. Every input or output is connected to a Z-spider via a normal edge or Hadamard edge.
5. Every Z-spider is connected to a most one input or output.

Every ZX-diagram can be transformed into an equivalent graph-like diagram by using specific rewriting rules [Wet20]. An example for a graph-like diagram is shown in Figure 2.4.

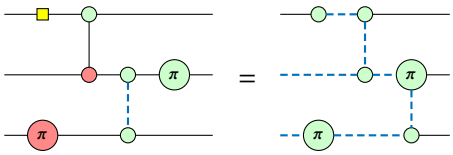


Figure 2.4: ZX-diagram and its graph-like form.

3 Methodology

Our starting point for the research objective is a quantum circuit, whose sampling overhead we want to decrease when circuit cutting is applied, under the constraint that we have a QPU with some fixed amount of available qubits. A central challenge in this context is the development of effective circuit modification schemes, which we refer to as rewriting techniques. These techniques are algorithms that take a quantum circuit as input, apply circuit transformations, and output a functionally equivalent circuit. However, simply applying a rewriting technique does not guarantee a reduction in sampling overhead. The benefit of a transformation remains unknown until the modified circuit is cut and the resulting sampling overhead is calculated. This calls for a more comprehensive optimization strategy that integrates rewriting and overhead evaluation into a feedback-driven process. A basic iterative optimization approach might involve the following steps:

1. Cut the original input circuit and calculate the sampling overhead.
2. Apply a rewriting technique to obtain a modified version of the circuit.
3. Cut the modified circuit and calculate the new sampling overhead.
4. Compare the new overhead with the original. If the overhead has decreased, accept the new circuit; otherwise, repeat step 2.

This approach is inspired by general quantum circuit optimization methods, where a variety of rewriting strategies are used to achieve goals such as reducing gate count, minimizing circuit depth, tailoring the circuit for specific QPUs, or enabling efficient compilation [KPJT25]. For instance, Krueger [Kru22] demonstrated how combining rewriting strategies with local search techniques could significantly reduce the number of two-qubit gates in a circuit.

Building on these ideas, we propose a complete optimization pipeline tailored to the circuit cutting context, as illustrated in Figure 3.1. Our pipeline begins with a quantum circuit and iteratively rewrites and evaluates it using various transformation techniques. Each rewritten version is assessed based on its resulting cut configuration and associated sampling overhead. The process continues for a fixed amount of iterations or until the sampling overhead cannot be reduced any further.

This chapter is structured as followed: In Section 3.1, we explore a range of techniques for rewriting quantum circuits into functionally equivalent forms, laying the groundwork for effective circuit transformations. Section 3.2 builds upon this by presenting optimization strategies designed to minimize sampling overhead resulting from circuit cutting.

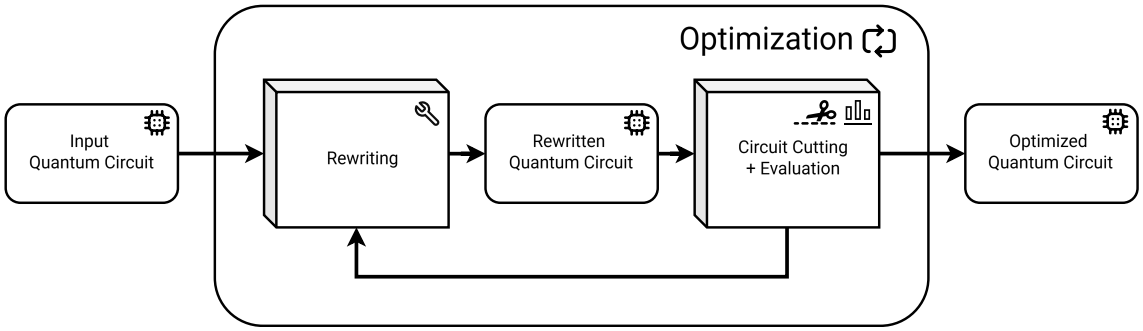


Figure 3.1: Circuit optimization pipeline. An quantum circuit is optimized for circuit cutting via an iterative, feedback-driven optimization scheme. It consists of a rewriting block that modifies the circuit, followed by an evaluation block that performs the circuit cutting and assesses the resulting sampling overhead.

3.1 Rewriting Quantum Circuits

In this section, we explore several strategies for rewriting quantum circuits, focusing on common techniques such as gate commutativity and gate cancellation. We examine how these transformations can influence the outcome of circuit cutting by potentially exposing better cut locations. In addition, we use concepts from ZX-Calculus to convert circuits into more flexible ZX-diagrams. These diagrams allow for powerful, rule-based rewriting. After applying selected transformations within the ZX framework, we translate the diagrams back into equivalent quantum circuits to evaluate their impact.

3.1.1 Cancel Commutative Gates

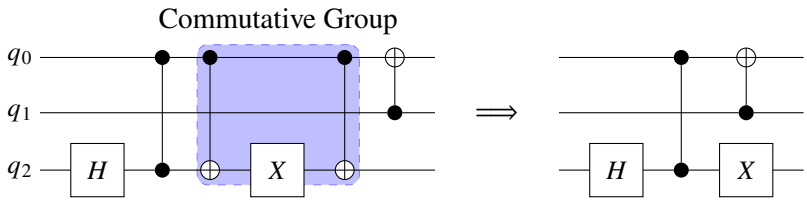
One of the simplest approaches in quantum circuit optimization is to eliminate redundant gates, thereby reducing both the circuit depth and overall gate count [NRS+18]. For instance, some quantum gates are self-adjoint, meaning their matrix representation is equal to its own conjugate transpose. In the context of unitary operations, this implies that applying such a gate twice cancels its effect, as it effectively acts as its own inverse.

Common examples of self-adjoint gates include the Pauli gates X and Z, the Hadamard gate H, and two-qubit gates like CNOT and CZ. Identifying and removing pairs of these gates when they appear sequentially in a circuit can simplify the overall structure.

We can further optimize circuits by leveraging commutativity, the property that certain gates can be rearranged without affecting the final outcome. By identifying groups of commuting gates, we can reorder them to bring canceling pairs of self-adjoint gates together and remove them.

These optimizations not only simplify the circuit but also reduce the number of gates eligible for cutting. In turn, this can lower the sampling overhead when applying gate cuts during circuit decomposition.

Consider following three qubit circuit with the commutative group consisting of two CNOTs and a single X gate:



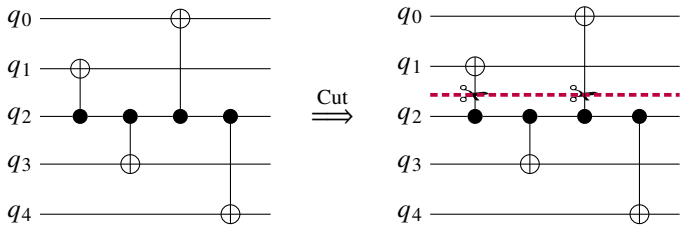
Since all gates in the commutative group can be reordered, we swap the X gate with one of the CNOT gates. This brings the two identical CNOT gates together, allowing them to cancel out. As a result, we obtain a new circuit that is two gates shorter.

Let us cut this circuit by using following qubit partitions: $\{q_0, q_1\}$ and $\{q_2\}$. In the original version, this configuration would require gate cuts on one CZ gate and two CNOT gates, leading to a total sampling overhead of $9^3 = 729$. After optimization, with the redundant CNOT gates removed, only one gate cut remains, reducing the sampling overhead significantly to just 9.

3.1.2 Swap Commutative Gates

The pattern and placement of gates can affect how easily the circuit can be partitioned and how many cuts must cross multi-qubit operations. To restructure a quantum circuit we can swap the order of two sequential commutative gates so that the cutting algorithm might be able to find different cut configurations. A straightforward implementation of this method can be described as follows: Select a random gate within the circuit along with its immediate neighboring gates that act on the same qubit, both before and after the selected gate. Check whether any of these neighboring gates commute with the selected gate. If a commuting pair is found, perform the swap and return the modified circuit. If no such commuting gates are identified, the procedure terminates and the original circuit is returned unchanged.

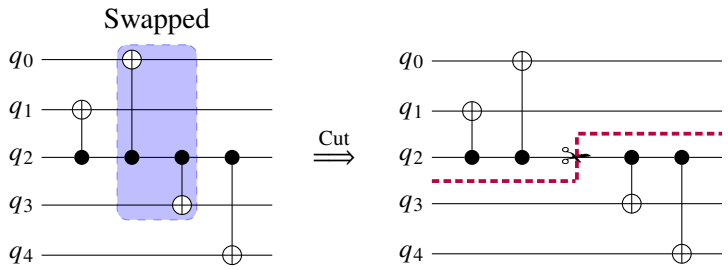
It should be noted that this technique will not be able to change circuits, containing only non-commuting gates. However, the sampling overhead could be reduced for the certain circuits like in the following example:



(3.1)

This five-qubit quantum circuit contains four CNOTs, each acting on different qubits. Now we want to run this circuit on another QPU, which only has three qubits available. For demonstrations sake, we say that the qubits are fixed and cannot be reordered for partitioning. If a cutting algorithm is applied, it would suggest us two gate cuts through two CNOTs, resulting in the sampling overhead of $3^2 \times 3^2 = 81$.

Now we swap the second and third CNOT and cut the circuit:



(3.2)

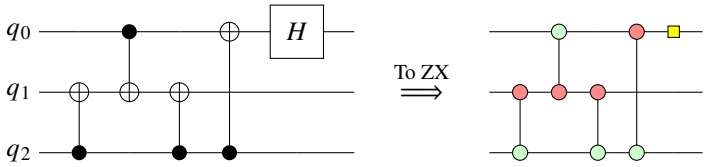
The two gate cuts are not necessary anymore for this specific circuit structure. The optimal cut would be a wire cut placed after the second gate on q_2, which results in the decreased sampling overhead of $4^2 = 16$.

3.1.3 ZX Simplifications

General circuit optimization strategies often rely on intermediate representations, such as directed acyclic graphs, which offer deeper structural insight and enable more flexible transformations [NRS+18]. In this work, we leverage the ZX-Calculus by transforming quantum circuits into ZX-diagrams, where we apply various rewrite rules aimed at producing equivalent circuits with reduced sampling overhead when circuit cutting is applied.

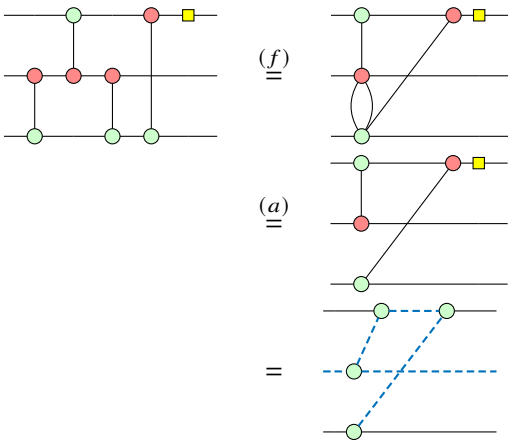
As previously discussed, simplifying a circuit can lead to fewer required cuts during the decomposition process. For this reason, it is also advantageous to reduce the overall complexity of the corresponding ZX-diagram. This simplification can be achieved by minimizing the number of spiders (nodes) and edges in the diagram, ultimately resulting in a potentially more compact and more efficient circuit representation.

Let us simplify a three qubit circuit with some CNOT gates and a Hadamard gate. The first step is to convert the circuit into a ZX-diagram:



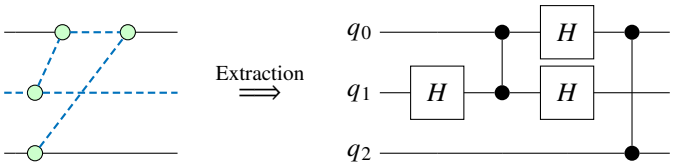
(3.3)

We then apply the fusion (f)-rule to merge the X-spiders and Z-spiders in the second and third row. Afterwards, the antipode (a)-rule removes the resulting double edge between the X-spider and Z-spider:



(3.4)

The final equality illustrates the transformation into a graph-like ZX-diagram, a necessary step for extracting valid quantum circuits. The resulting circuit contains fewer two-qubit gates than before:



(3.5)

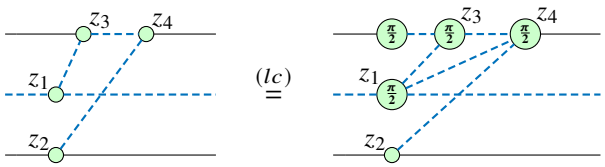
Note that by further rewriting the Hadamard and CZ gates as CNOTs, we can observe that the two CNOTs acting on the second and third qubits in the input circuit cancel each other out. This essentially achieves the same effect as the Commutative Cancellation method, but here it is accomplished through the ZX-Calculus framework.

3.1.4 ZX Congruences

After applying multiple simplification strategies onto a given ZX-diagram, the ZX-diagram will reach a state, in which no more simplification rules can be further applied to reduce the node and edge count. The idea is to start from this fully reduced state and apply non-simplification strategies such as local complementation and pivoting to generate structurally congruent ZX-diagrams (see Section 2.3.2). Both operations alter the connectivity between spiders, enabling the creation of multiple equivalent representations for a complex diagram. However, these transformations can also lead to an increase in structural complexity, potentially resulting in a more complex circuit. We aim to generate a larger set of circuits, as this increases the pool of candidates for evaluation and enhances the likelihood of identifying better solutions.

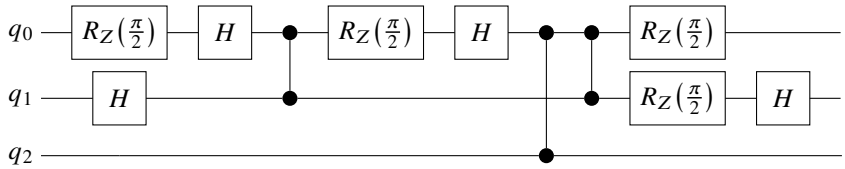
Example Local Complementation

We first start with the application of local complementation. Given the graph-like ZX-diagram from Equation (3.4), the Z-spiders are annotated from z_1 to z_4 and the local complementation is performed about the target spider z_3:



(3.6)

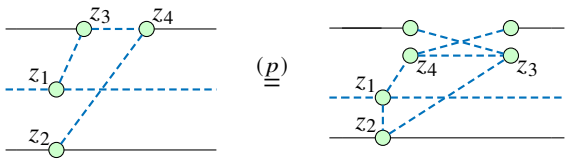
First, an extra Z-spider is added on the same row as z_3, connected via a Hadamard edge. The neighborhood is $N_{z_3} = \{z_1, z_4\}$ and the neighbors are not connected. This means we add an Hadamard edge between z_1 and z_4. Finally, a phase of $\pi/2$ is additionally added to the the target spider, the extra inserted spider and the neighboring spiders. Extracting a quantum circuit of this congruent diagram leads to following complex circuit:



(3.7)

Example Pivoting

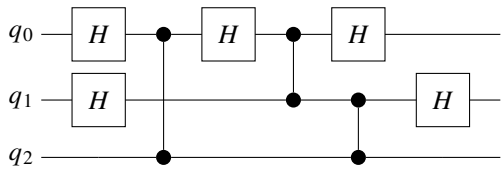
Again, we start from the ZX-diagram given in 3.4 and apply a pivot on the graph along edge (z_3, z_4):



(3.8)

The common neighborhood is $N(z_3) \cap N(z_4) = \{\}$. The exclusive neighborhoods are $N(z_3)\backslash(N(z_4) \cup \{z_4\}) = \{z_1\}$ and $N(z_4)\backslash(N(z_3) \cup \{z_3\}) = \{z_2\}$. Since there are no vertices in the common neighborhood, we only complement the edges between the exclusive neighborhoods. Specifically, an Hadamard edge is added between z_1 and z_2. Furthermore, the positions of z_3 and z_4 are swapped, and additional Z-spiders are introduced at these vertices, each connected via Hadamard edges.

The final extracted circuit is



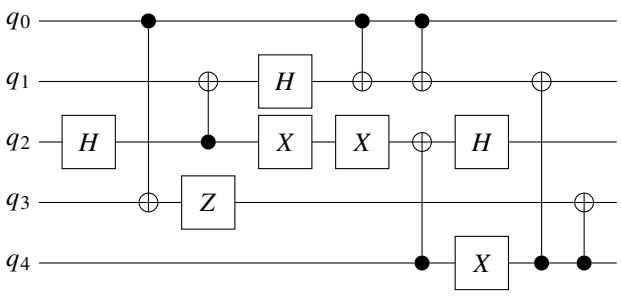
(3.9)

Although each circuit extracted from the different ZX rewriting strategies exhibits a distinct structure (see circuits 3.5, 3.7, and 3.9), they are all functionally equivalent and represent the same underlying quantum operation. While these examples do not immediately demonstrate benefits for circuit cutting, we will show in Chapter 4 that local complementation and pivoting can effectively reduce sampling overhead for certain circuits. One drawback of applying local complementation and pivoting to ZX-diagrams is the introduction of additional nodes, which increases the overall size of the diagram. To prevent excessive growth in complexity with each operation, it is recommended to apply simplification strategies after these transformations.

3.1.5 Random Section Rewrite

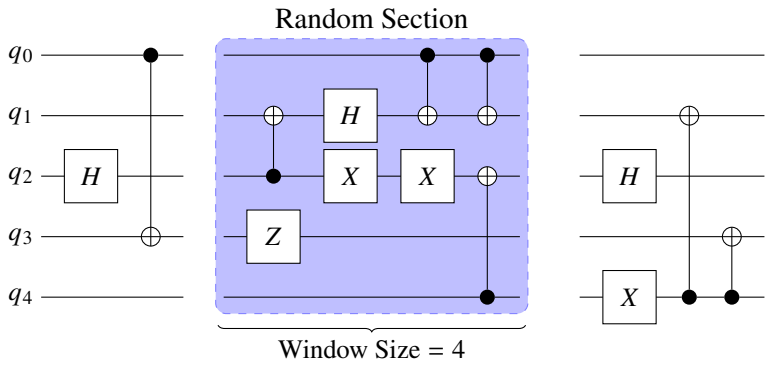
Instead of rewriting and modifying the entire circuit at once, this strategy focuses on optimizing smaller sections. The circuit is divided layer by layer, and a randomly selected portion is chosen for applying the optimization strategies discussed in previous sections. After optimization, the modified section is recombined with the left out parts of the circuit. This localized approach significantly improves runtime performance, as optimizing smaller segments of a large and complex circuit is much faster and more computationally efficient. In some cases, this method can even lead to improvements in sample overhead, as we will show in Chapter 4.

As a demonstration, we apply the `Random-Section-Rewrite` technique on following five-qubit circuit:



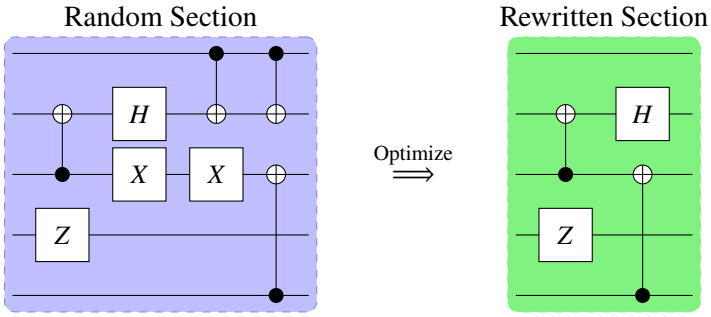
(3.10)

We begin by selecting a random portion of the input circuit and dividing the circuit into distinct sections accordingly.



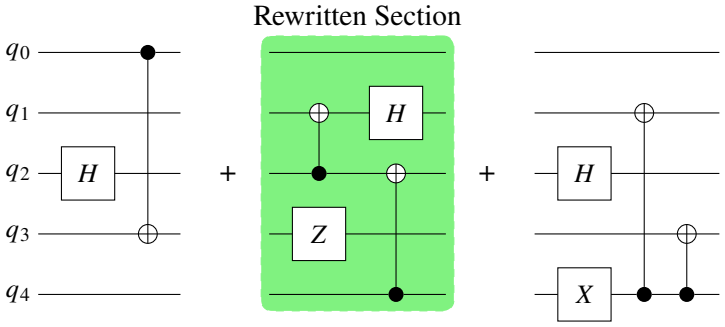
(3.11)

The window size determines the exact number of layers for the selected circuit segment, which in this case is set to four layers. The random section begins just after the second layer. Optimization strategies are then applied to this chosen section:



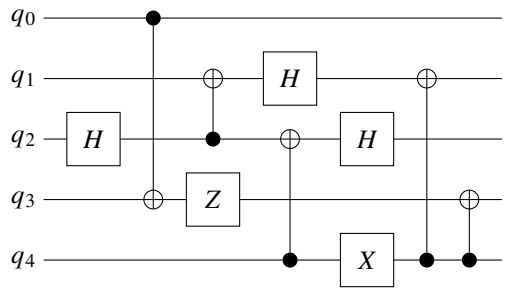
(3.12)

This subcircuit contains redundant X gates and CNOTs, which can be removed by using the `Cancel-Commutative-Gates` technique for example. In the final step, the beginning section and end section of the input circuit are recombined



(3.13)

and we obtain the whole rewritten and optimized circuit:



(3.14)

3.2 Optimization Techniques

In the previous section, we introduced several rewriting strategies that can potentially modify the circuit in a way that reduces the sampling overhead. However, we need a method to determine how to select the appropriate circuit rewrites. For this, we employ heuristic local search to minimize a specific objective function, namely the sampling overhead resulting from circuit cutting. By iteratively rewriting circuits and evaluating their effect on the objective function, the search process gradually identifies circuit variants with lower overhead. Techniques like genetic algorithms and simulated annealing serve as effective strategies within this framework, guiding the search toward configurations that yield lower cost with respect to the defined objective.

3.2.1 Objective Function

An objective function describes a cost that quantifies the quality or performance of a solution to a given problem. It serves as the metric that optimization algorithms aim to minimize or maximize, depending on the task. In our case, we aim to minimize the sampling overhead which turns the task into a minimization problem. The objective function can be freely chosen, but it should have some relations to the sampling overhead.

We define our objective function f_{cost} for a given circuit c as

$$f_{\text{cost}}(c) = \text{ total sampling overhead of } \texttt{CUT}(c), \tag{3.15}$$

where `CUT` is a function which cuts the given circuit c and returns information about the cut configuration. The total sampling overhead is then calculated by multiplying over the γ-factor of each needed cut as defined in Equation (2.3).

Unfortunately, evaluating this objective function is computationally expensive, as determining effective cut configurations for a circuit is in itself a complex optimization problem. This significantly influences the efficiency of the optimization strategies discussed in the following sections. To ease the computation cost, we could use a more simple objective function such as the gate count or depth for a given circuit. While this function is very easy to evaluate, the objective function is not strongly related to our task of reducing the sampling overhead, since a circuit with fewer gates or reduced

depth does not necessarily yield cut configurations with lower overhead. Therefore, formulating an objective function that accurately reflects the optimization target and has a low evaluation cost is essential for the effectiveness of the overall approach.

3.2.2 Simulated Annealing

Algorithm 3.1 Simulated annealing

function SIMANNEAL($c, k, T_0, T_{\text{cool}}, p_{\text{restart}}$)
  $c_{\text{best}} \leftarrow c$
  $s_{\text{best}} \leftarrow \texttt{score}(c_{\text{best}})$
  $s \leftarrow s_{\text{best}}$
  $T \leftarrow T_0$
  for $i \leftarrow 1$ **to** k **do**
    $c_{\text{new}} \leftarrow \texttt{rewrite}(c)$
    $s_{\text{new}} \leftarrow \texttt{score}(c_{\text{new}})$
    if $s_{\text{new}} < s$ **or** $\texttt{random}(0,1) < \exp\left(-\frac{\log_{10}(s_{\text{new}}) - \log_{10}(s)}{T}\right)$ **then** // acceptance condition
      $c \leftarrow c_{\text{new}}$
      $s \leftarrow s_{\text{new}}$
      if $s < s_{\text{best}}$ **then**
        $c_{\text{best}} \leftarrow c$
        $s_{\text{best}} \leftarrow s$
      end if
    else if $\texttt{random}(0,1) < p_{\text{restart}}$ **then** // restart scheme
      $c \leftarrow c_{\text{best}}$
    end if
    $T \leftarrow T - T_{\text{cool}}$ // cooling schedule
  end for
  return c_{best}
end function

Simulated annealing (SA) is a probabilistic optimization technique inspired by the physical process of annealing in metallurgy, where a material is heated and then slowly cooled to reduce defects and find a stable structure [KGV83]. The core idea behind the algorithm is to find an approximate global optimum of a given function in a large search space, especially when the function is non-convex, complex, or riddled with local optima.

The detailed implementation of SA for optimizing quantum circuit for our use case is given in Algorithm 3.1.

The algorithm begins with an input circuit c as the initial solution and an initial temperature T_0. Then, we modify the circuit for a fixed number of iterations k. At each iteration, the current circuit is modified by one randomly selected the rewriting technique discussed in Section 3.1 (in Chapter 4 we will exactly see how and why certain rewriting techniques are chosen). The new rewritten circuit is then evaluated by our objective function (Equation (3.15)), which measures the sampling overhead. If the new solution improves the objective, it is accepted unconditionally. However, if

the solution is worse, we may still accept it with a certain probability (line 9). This probability P is determined by the difference in scores from the current and previous circuit and the current temperature, following the Boltzmann distribution [Zha19]

$$P(s_{\text{new}}, s_{\text{prev}}) = \exp\left(-\frac{\log_{10}(s_{\text{new}}) - \log_{10}(s_{\text{prev}})}{T}\right),$$

where s_{new}, s_{old} denote the scores from the newly modified and previously evaluated circuits and T is temperature at the current iteration. Here, we use the logarithmic difference two compare the scores, because the sampling overhead grows exponentially and quickly reaches very large values. At the end of the iteration, the temperature cools down according to a predefined cooling schedule (line 19). As the temperature decreases over time, the algorithm becomes less likely to accept worse solutions, favoring exploitation over exploration.

The strength of SA lies in its ability to escape local minima early in the process by allowing uphill moves, and then gradually focusing on fine-tuning as the system cools. The cooling schedule, which dictates how fast the temperature drops, plays a crucial role in the algorithm's success. If the temperature decreases too quickly, the algorithm might get stuck in a local minimum. On the contrary, if it decreases too slowly, the process becomes inefficient and slow. So choosing the right hyper-parameters is essential.

Sometimes, local search algorithms incorporate a random restart mechanism [CJGM21], which resets the current solution to a previously discovered one at the end of an iteration with a given probability p_{restart} (see line 16). This strategy allows the search to explore alternative regions of the solution space, potentially leading to configurations with lower scores.

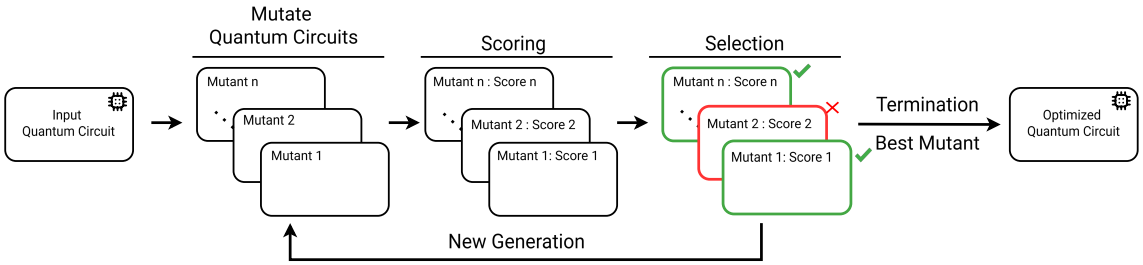


Figure 3.2: General genetic algorithm process for optimizing quantum circuits in the circuit cutting context. The procedure consists of three main stages: mutation, scoring, and selection. This cycle is repeated for a predefined number of generations. At the end of the process, the best performing mutant, corresponding to the circuit with the lowest observed sampling overhead, is selected and returned.

3.2.3 Genetic Algorithm

The genetic algorithm is a search and optimization method inspired by the principles of natural selection and genetics [Hol92]. It belongs to the class of evolutionary algorithms and operates on a population of candidate solutions, evolving them over successive generations to find high-quality solutions to complex problems. The central idea is to simulate the process of biological evolution, where the fittest individuals are more likely to pass on their traits to the next generation. In the context of genetic algorithms (GAs), the objective function is also typically referred to as the fitness function.

The implementation of GA for optimizing quantum circuits is provided in Algorithm 3.2 and a general process is depicted in Figure 3.2. The algorithm starts by initializing a population, composed of copies of the input circuit (line 6). Each member of the population is then mutated, meaning we apply randomly selected rewriting strategies on the circuits (line 9). In this setting, a rewritten circuit is referred to as a mutant. Every mutant is evaluated by the fitness function and assigned a score (fitness) based on its performance (line 11).

Next comes the selection phase, which sets up the population for the new generation (line 19). There are different ways to choose which mutants we should keep or discard. Selection methods used in this work are the tournament selection and truncation selection [Jeb13]. In the tournament selection, a set of randomly chosen mutants is constructed. Then those mutants are ranked by their fitness, and the best individual is passed to the next generation. This step is repeated multiple times until the new population is filled. The truncation selection is a simpler process, which ranks all mutants in the population according to their fitness at once. Then a predefined cut-off percentage retains only the top-performing individuals for the next generation.

After selection, the population undergoes another round of mutation, and the cycle continues. Over time, the population evolves as better solutions are selected and refined. The process continues for a predefined number of generations or until a satisfactory solution is found. The effectiveness of GAs largely depends on population size, the number of generations and the selection method.

GAs are particularly useful for this problem, since the search space is vast due to the nearly unlimited ways of representing quantum circuits.

Algorithm 3.2 Genetic algorithm

```
1:  function GENALG(c, n_mut, n_gens)
2:      c_best ← c
3:      s_best ← score(c_best)
4:      circuits ← []
5:      for i ← 1 to n_mut do
6:          circuits.add(c)                          // initial population
7:      end for
8:      for i ← 1 to n_gens do
9:          mutants ← mutate(circuits)               // mutate (rewrite) each circuit
10:         scores ← []
11:         for each c_mut ∈ mutants do              // parallelization can be used here
12:             s ← score(c_mut)
13:             scores.add(s)
14:             if s < s_best then
15:                 c_best ← c_mut
16:                 s_best ← s
17:             end if
18:         end for
19:         circuits ← select(mutants, scores)       // select the best mutants
20:     end for
21:     return c_best
22: end function
```

4 Experiments and Results

This chapter focuses on testing the proposed rewriting techniques and optimization methods discussed in the previous chapters. We start with Section 4.1, where we look at how these methods are specifically implemented using different libraries. Following that in Section 4.2 and Section 4.3 the rewriting techniques are evaluated against a set of benchmark circuits. Finally, in Section 4.4 the optimization methods SA and GA are compared against each other.

4.1 Implementation

All implementations are developed for the Python v3.10 platform and available on GitHub [Ngu25]. We utilize IBM's open-source library Qiskit v1.3.1 [JTK+24], which provides tools for constructing and working with quantum circuits. The `Swap-Commutative-Gates` and `Random-Section-Rewrite` techniques are implemented using this library, while the `Cancel-Commutative-Gates` method is directly available in Qiskit's `qiskit.transpiler.passes` package under the name `CommutativeCancellation`[1]. To apply ZX-calculus-based rewrites, we use the open-source library PyZX v0.8.0 [KW20a]. PyZX supports conversion functions between quantum circuits and ZX-diagrams, includes the `full_reduce` routine that automatically applies a range of simplification rules, and provides full implementations of local complementation and pivoting.

We implement the SA and GA search procedures according to Algorithm 3.1 and Algorithm 3.2. For the GA specifically, we implement a parallel evaluation stage, to reduce the overall runtime.

For evaluating circuits and determining cut configurations, we use qiskit-addon-cutting v0.9.0 [BCE+24], which facilitates automatic cut finding and sampling overhead calculations. One shortcoming of this library is that only circuits with single-qubit and two-qubit operations can be cut. For our tests, we break multi-qubit gates (≥ 3) down by using Qiskit's `transpile` function, which transforms those gates using a set of basis gates. Specifically, our chosen basis gates are $\{H, X, Z, CZ, CX, RX, RZ\}$, since they are straightforward to translate into ZX-diagrams, but any other universal basis gates set with one-qubit and two-qubit operations would have sufficed as well.

All methods and algorithms are run on an Intel Core i7-12700k processor at 3.6 GHz base clock.

An overview of the full optimization pipeline and its components is shown in Figure 4.1.

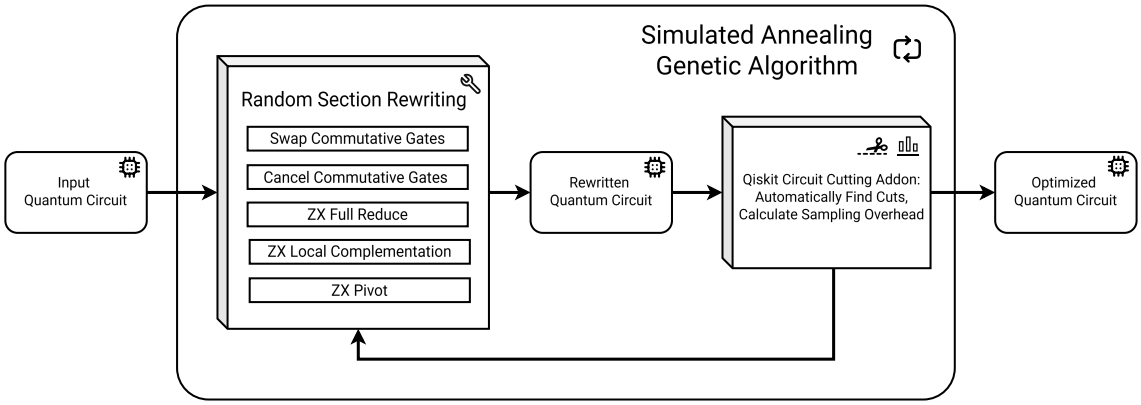


Figure 4.1: Implemented circuit optimization pipeline. An input circuit is optimized by either SA or GA. The rewrite block consists of our introduced rewriting techniques. The rewritten quantum circuit is then cut and evaluated by the qiskit-addon-cutting library. This process is repeated and we obtain a final optimized circuit.

4.2 Isolated Rewriting

To test the effectiveness of each rewriting strategy, we benchmarked 18 circuits[2] from [KW20b] and see whether the sampling overhead could be decreased for any of those circuits. The QPS determines the maximum number of qubits allowed for each created subcircuit when circuit cutting is applied. In this case, we set the QPS to half the size of the original circuit, rounded up. This corresponds to executing the subcircuits on a quantum computer that has only half the number of qubits required by the full circuit. We use SA with only one rewriting method in isolation to optimize over the sampling overhead. For the SA parameters, we set the number of iterations to $k = 200$, with an initial temperature of $T_0 = 10$, a cooling rate of $T_{\text{cool}} = 0.005$, and a reset probability of $p_{\text{reset}} = 0.05$. To better compare the improvements, we calculate the logarithmic difference

$$\Delta_{\log} = \log_{10}(\text{base overhead}) - \log_{10}(\text{optimized overhead}) \tag{4.1}$$

$$= \log_{10}\left(\frac{\text{base overhead}}{\text{optimized overhead}}\right). \tag{4.2}$$

It describes the ratio between the base overhead and the optimized overhead on the logarithmic scale. For example, $\Delta_{\log} = 5$ means that the sampling overhead decreased by an order of 10^5 times, while $\Delta_{\log} = 0$ hints at a no overhead change. The results are given in Table 4.1.

The tests have shown that the `ZX-Pivot` and `ZX-Local-Complementation` were the most effective strategies. `ZX-Pivot` could improve the overhead of 33.3% of the 18 circuits, while `ZX-Local-Complementation` improved 22.2% of circuits. Following that are the `ZX-Full-Reduce` and the `Swap-Commutative-Gates` methods, both improving 11.1% of circuits. The worst performer was the `Cancel-Commutative-Gate` strategy, which could not improve a single circuit.

[1] https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.CommutativeCancellation

[2] The original reference included additional circuits; however, some were excluded from our benchmark analysis, as the cutting algorithm was unable to find a solution due to the large circuit size.

				full circuit					random section rewrite				
circuit	#qubits	qps	base overhead	cancel	swap	f-reduce	lcomp	pivot	cancel	swap	f-reduce	lcomp	pivot
adder-8	24	12	5.2e+26	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
barenco-tof-10	19	10	3.4e+30	0.00	0.00	0.00	0.00	0.00	0.00	0.00	19.08	6.68	20.99
csla-mux-3	15	8	1.9e+15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
csum-mux-9	30	15	1.9e+15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.00
grover-5	9	5	6.4e+45	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.77
ham15-low	17	9	1.0e+104	0.00	25.76	0.00	0.00	7.63	0.00	20.04	15.27	43.90	2.86
ham15-med	17	9	4.1e+70	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.63	0.00
hwb6	7	4	2.2e+55	0.00	3.82	0.00	4.77	4.77	0.00	0.00	3.82	0.00	8.59
mod5-4	5	3	2.3e+13	0.00	0.00	3.82	8.34	7.63	0.00	0.00	0.95	1.91	3.82
mod-mult-55	9	5	8.0e+22	0.00	0.00	0.00	0.00	0.00	0.00	2.86	0.95	5.73	5.73
mod-red-21	11	6	4.2e+28	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.00	0.00
qcla-adder-10	36	18	256.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
qcla-com-7	24	12	2.8e+11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.00
qcla-mod-7	26	13	1.3e+81	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
qft-4	5	3	5.2e+26	0.00	0.00	0.95	8.59	7.63	0.00	0.00	10.50	6.68	9.54
rc-adder-6	14	7	3.9e+08	0.00	0.00	0.00	0.00	0.95	0.00	0.00	0.00	0.95	1.91
tof-10	19	10	1.9e+15	0.00	0.00	0.00	11.45	11.45	0.00	0.00	13.36	9.54	9.54
vbe-adder-3	10	5	6.6e+04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 4.1: Benchmarking circuits from [KW20b] with rewriting techniques used in isolation. The table shows each circuit with number of qubits, the allowed qubits per subcircuit (QPS) and the calculated base overhead when circuit cutting is applied. The used optimization method was SA($k = 200, T_0 = 10, T_{\text{cool}} = 0.005, p_{\text{reset}} = 0.05$). For each strategy, we calculate the logarithmic difference between the base and optimized overhead (see Equation (4.1)). This is done for each strategy applied on the full circuit and also on sections, using the `Random-Section-Rewrite` strategy. The green shaded values highlight positive improvements, where the sampling overhead could be reduced.

Applying `Random-Section-Rewrite` could improve even more circuits than before. `ZX-Pivot` and `ZX-Local-Complementation` now improved 50% and 55.5% of circuit respectively. The number of improved circuits rose significantly for the `ZX-Full-Reduce` with 44.4%. The `Swap-Commutative-Gates` and `Cancel-Commutative-Gates` methods both remained the same with a 11.1% and 0% improvement rate.

4.3 Combined Rewriting

Now we analyze what effect the rewriting strategies impose on the overhead when used in combination. For that we use SA with `Random-Section-Rewrite` as the rewriting method. `Random-Section-Rewrite` will perform the circuit modification by randomly applying one of the main rewriting techniques (`Swap-Commutative-Gates`, `Cancel-Commutative-Gates`, ...) each SA iteration. We have seen before that the ZX techniques are generally more effective. Therefore we set a probability distribution for the rewriting technique selection, boosting the probabilities of `ZX-Local-Complementation` and `ZX-Pivot`. The concrete probabilities are $p_{\text{swap}} = p_{\text{cancel}} = p_{\text{f-reduce}} = 0.1$ and $p_{\text{lcomp}} = p_{\text{pivot}} = 0.35$. The SA parameters are $k = 1000$, $T_0 = 10$, $T_{\text{cool}} = 0.005$, $p_{\text{reset}} = 0.05$ and the `Random-Section-Rewrite` uses a window size of 40% input circuit depth.

		original					optimized						
circuit	#qubits	#gates	#2q-ops	depth	#cuts	overhead	#gates	#2q-ops	depth	#cuts	overhead	$\Delta_{\log}$ overhead	runtime [s]
adder_8	24	885	385	242	28	5.2e+26	771	442	231	24	8.0e+22	3.82	217.1
barenco_tof_10	19	427	192	322	32	3.4e+30	382	220	258	10	3.5e+09	20.99	141.0
csla_mux_3	15	162	69	67	16	1.9e+15	162	69	67	16	1.9e+15	0.00	39.7
csum_mux_9	30	420	168	60	16	1.9e+15	420	168	60	16	1.9e+15	0.00	431.0
grover_5	9	749	288	505	48	6.4e+45	645	296	446	43	1.1e+41	4.77	247.2
ham15-low	17	425	236	271	109	1.0e+104	418	261	253	54	3.4e+51	52.48	125.1
ham15-med	17	1160	533	759	74	4.1e+70	986	491	619	67	8.6e+63	6.68	340.7
hwb6	7	250	115	163	58	2.2e+55	238	130	151	44	9.7e+41	13.36	47.7
mod5_4	5	60	28	48	14	2.3e+13	50	32	35	7	4.8e+06	6.68	10.6
mod_mult_55	9	117	48	51	24	8.0e+22	158	90	80	13	2.5e+12	10.50	49.5
mod_red_21	11	261	105	165	30	4.2e+28	261	115	159	29	4.7e+27	0.95	29.3
qcla_adder_10	36	482	213	75	2	256.0	482	213	75	2	256.0	0.00	294.7
qcla_com_7	24	406	174	85	12	2.8e+11	457	233	102	10	3.5e+09	1.91	213.2
qcla_mod_7	26	827	366	211	85	1.3e+81	986	528	247	76	3.3e+72	8.59	551.5
qft_4	5	107	44	72	28	5.2e+26	137	67	90	12	2.8e+11	15.27	34.7
rc_adder_6	14	187	81	103	9	3.9e+08	249	156	119	5	5.9e+04	3.82	83.1
tof_10	19	247	102	171	16	1.9e+15	269	152	147	2	81.0	13.36	125.4
vbe_adder_3	10	128	58	76	4	6.6e+04	135	74	69	4	2.1e+04	0.50	82.2

Table 4.2: Benchmarking circuits from [KW20b] with all rewriting strategies used in combination. We used the optimization method SA($k = 1000, T_0 = 10, T_{\text{cool}} = 0.005, p_{\text{reset}} = 0.05$) with the QPS set to half of the circuit qubit count. Additional information about the original circuit and optimized circuit are listed. Green shaded cells mark positive improvements and red cells mark negative improvements respective to the category. Grey shaded cells hint at no change.

Table 4.2 presents the results of applying all rewriting techniques in combination. Using this integrated approach, we successfully improved 15 out of 18 benchmark circuits, corresponding to an improvement rate of 83.3%. The sampling overhead decrease ranges from small log-differences of $\Delta_{\log} = 0.5$ (as seen in the vbe-adder-3 circuit) up to very large reductions, such as a log-difference of $\Delta_{\log} = 52.48$ for the ham15-low circuit.

In nearly all cases, where an improvement could be gained, the number of two-qubit operations for the optimized circuit increased, while the number of necessary cuts decreased. About half of the optimized circuits had less gates and a slightly shallower depth than before the rewrite. Also in this run we could not improve the csum-mux-9 circuit, while before using the rewriting strategies in isolation, that circuit could be improved up to a log-difference of $\Delta_{\log} = 0.95$ (compare Table 4.1 and Table 4.2).

Another interesting observation is for the optimized vbe-adder-3 circuit: The number of necessary cuts actually remained the same for the optimized circuit, but it still improved regarding the sampling overhead. The original cut circuit used four wire cuts which incurred an overhead of $16^4 = 65536$, while the optimized replaced two wire cuts with two CNOT gate cuts, introducing the reduced sampling overhead of $9^2 \times 16^2 = 20736$. The original and optimized cut placements are available in the appendix, shown in Figure A.1 and Figure A.2.

Lastly, no cases were found, where the sampling overhead could be reduced with an increasing number of cuts.

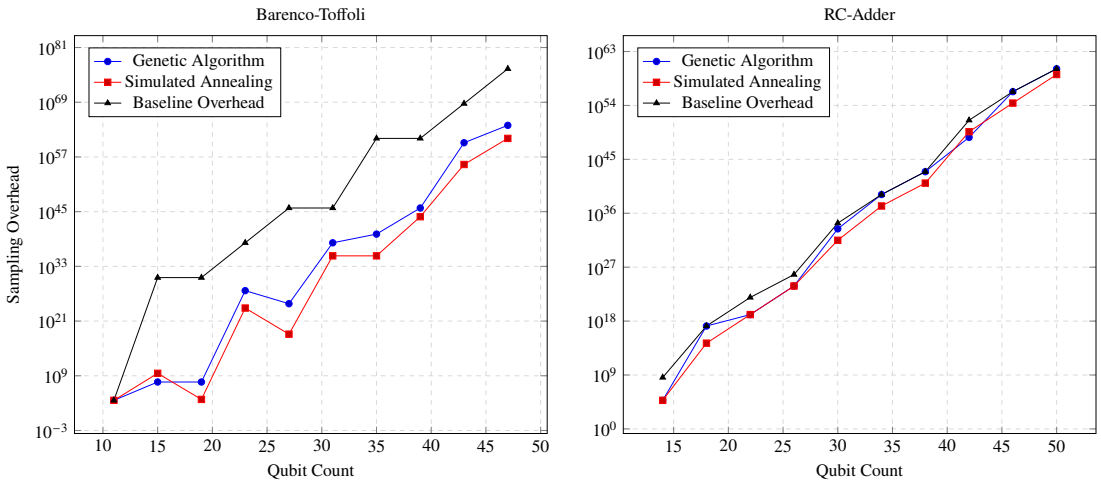


Figure 4.2: Comparing SA and GA in their effectiveness of reducing sampling overhead for circuits with different qubit counts. The QPS is set to 10 for the Barenco-Toffoli and to 7 for the Ripple-Carry-Adder. The following parameters were used for both circuit types: GA($n_{\text{mut}} = 10, n_{\text{gens}} = 30$) and SA($k = 300, T_0 = 10, T_{\text{cool}} = 0.005, p_{\text{reset}} = 0.05$). Each circuit was optimized 5 different times and the minimum sampling overhead was recorded.

4.4 Simulated Annealing vs Genetic Algorithm

In previous sections, we only used SA to optimize over the sampling overhead. In this section, we present a detailed comparison between the two implemented optimization methods, SA and GA, assessing their effectiveness in reducing sampling overhead and their average runtime performance. For this purpose, we evaluate both techniques on two benchmark circuits: the *Barenco-Toffoli* circuit [BBC+95] and the improved *Ripple-Carry Adder* [CDKM04]. These circuits were selected based on earlier experiments, where they showed notable improvements with regard to the sampling overhead after circuit modifications.

Effectiveness

To assess the effectiveness of reducing the sampling overhead for each optimization method, we generated 10 Barenco-Toffoli and 10 Ripple-Carry-Adder circuits, each with an increasing number of qubits. We optimized each circuits with both search procedures for circuit cutting with a QPS limit of 10 for the Barenco-Toffolis and 7 for the Ripple-Carry-Adders. Just like in Section 4.3, we use the `Random-Section-Rewrite` as the main circuit rewriting strategy, but this time with a fixed window size of 80. Each circuit was optimized five times using GA and SA, and the minimum resulting sampling overhead was recorded.

For GA, we used a population size of $n_{\text{mut}} = 10$ mutants over $n_{\text{gens}} = 30$ generations. The selection method was truncation selection, where only the top 50% of mutants were retained in each generation. In prior testing, truncation selection consistently outperformed tournament selection. For SA, we used $k = 300$ iterations per circuit, with a starting temperature of $T_0 = 10$, a cooling rate

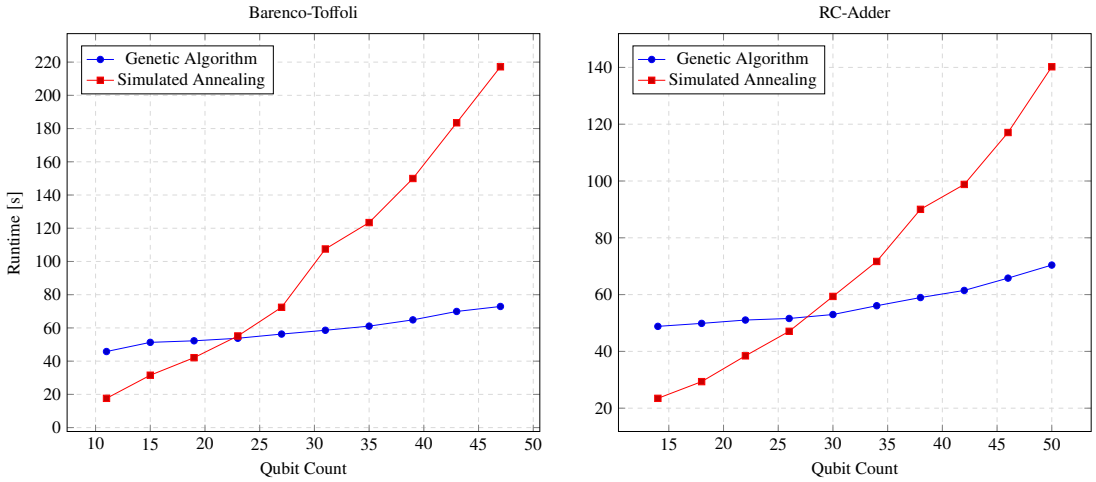


Figure 4.3: Runtime comparison between SA and GA for circuits with different qubit counts. The QPS for the Barenco-Toffoli is set to 10 and for the Ripple-Carry-Adder is set to 7. The following parameters were used for both circuit types: GA($n_{\text{mut}} = 10, n_{\text{gens}} = 30$) and SA($k = 300, T_0 = 10, T_{\text{cool}} = 0.005, p_{\text{reset}} = 0.05$). Each circuit was optimized 5 times and the average runtime was recorded.

of $T_{\text{cool}} = 0.005$, and a reset probability of $p_{\text{reset}} = 0.05$. This means that both algorithms evaluated 300 different circuit rewrites per run, allowing a fair comparison of their ability to minimize sampling overhead.

Figure 4.2 shows the overhead of the circuits after applying both methods, and the base overhead is given for the original circuit. In the case for Barenco-Toffoli circuits, both SA and GA achieved reductions in sampling overhead in 9 out of 10 circuits, with SA generally finding slightly better solutions. For example, the highest reduction was found by SA for the 19-qubit Barenco-Toffoli circuit from an initial sampling overhead of 3.4×10^{30} down to 6561, denoting a log-difference of $\Delta_{\log} = 26.7$. For the Ripple-Carry Adder, SA successfully reduced overhead in all 10 circuits, whereas GA only succeeded in 5 cases. However, the overall overhead decrease for both techniques were smaller in this benchmark circuit. The highest decrease was found by both SA and GA for the 14-qubit Ripple-Carry-Adder with a log-difference of $\Delta_{\log} = 3.81$.

Runtimes

Figure 4.3 illustrates the average runtime for each circuit. Execution times ranged from 17 seconds for the smallest Barenco-Toffoli circuit to 217 seconds for the largest. The Ripple-Carry-Adder circuits followed a similar trend, with runtimes ranging from 23 to 140 seconds. For Barenco-Toffoli circuits with ≤ 23 qubits and Ripple-Carry-Adder circuits with ≤ 26 qubits, SA had a lower average runtime than GA. However, for circuits with ≥ 26 qubits, the GA outperformed SA significantly. Specifically, GA was 64% faster on the largest Barenco-Toffoli circuit and 50% faster on the largest Ripple-Carry Adder. These results indicate that SA scales less efficiently with circuit size, while GA maintains a relatively modest growth in runtime as the number of qubits increases.

5 Discussion

First, we tested the rewriting strategies `Cancel-Commutative-Gates`, `Swap-Commutative-Gates`, `ZX-Full-Reduce`, `ZX-Local-Complementation` and `ZX-Pivoting` in isolation. Among these, the ZX-based rewrites performed the best. `Cancel-Commutative-Gates` might have failed to optimize any circuits because most circuits are already well-designed, and finding redundant gates is relatively rare. When we switched to the `Random-Section-Rewrite` technique, we observed an increased number of circuits whose overhead could be further reduced. One possible reason is that `Random-Section-Rewrite` enables more equivalent circuit configurations, which expands the search space for improved solutions. For example, applying `ZX-Full-Reduce` to the entire circuit yields an optimized circuit. Running `ZX-Full-Reduce` again on the same modified circuit does not change it, as it is already in a reduced state. However, splitting the circuit into smaller sections and applying `ZX-Full-Reduce` individually allows for more structural variation, as each section is different.

Regarding the heuristic search procedures, SA was generally more effective than GA, given the same number of circuit evaluations. This may be due to the differing ways in which both methods traverse the solution space. SA performs rewrites on a single circuit for a predefined number of iterations, resulting in a deep traversal path. In contrast, GA explores the space more broadly through the inclusion of multiple mutants, though the limited number of generations restricts the overall depth. In our setup, the final optimized circuit from GA undergoes at most 30 rewrites, while SA evaluates and rewrites up to 300 times.

Two primary factors contribute to the overall runtime: the circuit rewriting steps and the circuit cutting evaluation. The rewriting techniques, particularly the ZX-based ones, significantly impact runtime due to the cost of converting a circuit into a ZX diagram, modifying it, and extracting a valid circuit back. The evaluation step probably contributes the most to runtime. In our optimization process, many different circuits must be evaluated and cut. The search for optimal cut placements is complex and time-consuming. GA mitigates this issue through parallelization. During the scoring step, all mutants are evaluated simultaneously, rather than sequentially. Better hardware could further increase parallel evaluation, reducing total runtime. SA, on the other hand, cannot benefit from parallelization because it is inherently sequential. Each SA iteration relies on the previous solution to determine the next step.

An interesting observation is that SA outperforms GA in runtime for smaller circuits. This may be due to our specific implementation of GA, which involves many circuit manipulations, list operations, and copy steps. Additionally, the selection stage in each generation introduces non-negligible computational effort compared to the simpler SA implementation.

A potential solution to further reduce runtime would be to replace the qiskit-addon-cutting with a faster cutting strategy. Since our approach is modular, the circuit cutting component can be substituted with other frameworks that offer improved performance.

6 Related Works

Reducing sampling overhead is a critical step toward making circuit cutting practical for real-world quantum computing applications. Several algorithms have been proposed to identify optimal cut configurations for a given quantum circuit. For example, Brandhofer et al. [BPK23] introduced a method combining wire and gate cuts, the insertion of ancilla qubits, and classical communication to achieve optimal partitioning [BPS23].

Another direction focuses on optimizing the postprocessing stage. ShotQC [CCJ24] proposes two techniques, shot distribution and cut parameterization, to reduce the total number of circuit executions required. Shot distribution uses an adaptive Monte Carlo approach to allocate more quantum resources to subcircuits that contribute most to the output variance, effectively lowering the statistical error. Cut parameterization further reduces variance by introducing additional degrees of freedom in the postprocessing step.

CiFold [KLW+24] addresses resource optimization by identifying and folding repeated patterns at the qubit level. By representing quantum circuits as graphs and applying parallel folding and modular partitioning, CiFold enables the extraction of more efficient subcircuits.

If we divide the circuit cutting process into three stages, namely pre-cut, cut, and post-cut, then these methods primarily focus on the cut and post-cut phases. In contrast, our approach operates during the pre-cut phase, where the input circuit is prepared for subsequent cutting processes. Our optimization framework is complementary to these techniques and can be integrated with them by adapting the objective function accordingly, which could potentially lead to improved results when used in combination.

Related work in distributed quantum circuit execution also addresses similar challenges. Dadkhah et al. [DZH21] focus on minimizing quantum teleportation operations by exploiting gate commutativity and substituting parts of the circuit with equivalent structures. Their method also employs a genetic algorithm to determine optimal qubit partitioning. This approach shares strong similarities with ours, differing mainly in the objective function used.

7 Conclusion

In this work, we aimed to optimize quantum circuits with respect to sampling overhead incurred during circuit cutting. To address this, we developed a comprehensive optimization pipeline that rewrites circuits, applies circuit cutting, evaluates the resulting sampling overhead using qiskit-addon-cutting, and searches for improved circuit representations using simulated annealing and genetic algorithms. As part of this pipeline, we proposed six different circuit rewriting techniques to generate equivalent but potentially more efficient circuits.

Several of these techniques exploit the commutativity property of quantum gates to reorder operations and cancel redundant gates when possible. Additionally, we leveraged the ZX-Calculus framework to convert circuits into ZX-diagrams, which offer more flexibility for transformation using a wider set of rewriting rules. These include both general simplification strategies and more advanced operations such as local complementation and pivoting, which enable significant structural changes while preserving circuit equivalence.

We evaluated our optimization pipeline on a set of benchmark circuits and observed that ZX-based rewriting methods were effective in reducing sampling overhead in most cases. In contrast, swapping commutative gates offered limited benefit, and cancellation-based methods only worked when redundant gates were present, which was rarely the case. To further enhance scalability, we introduced a windowed optimization technique, where only randomly selected sections of the circuit are rewritten, yielding additional improvements in certain circuits.

Finally, we compared the performance of simulated annealing and genetic algorithms using the Barenco-Toffoli and Ripple-Carry-Adder circuits. Our results show that simulated annealing typically discovers lower-overhead solutions, while the genetic algorithm achieves faster runtimes by taking advantage of parallel circuit evaluation.

Several directions remain open for extending this work. First, additional rewriting strategies could be explored, particularly those that more directly influence the behavior of the circuit cutting module or exploit alternative transformation rules. Future research could also incorporate other optimization algorithms that may outperform the current methods, particularly in terms of solution quality or runtime efficiency. Furthermore, more systematic exploration of hyperparameter tuning for simulated annealing, genetic algorithms and our `Random-Section-Rewrite` technique could yield even better solutions.

Bibliography

[BBC+95] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, H. Weinfurter. "Elementary gates for quantum computation". In: *Physical Review A* 52.5 (Nov. 1995), pp. 3457–3467. ISSN: 1094-1622. DOI: 10.1103/physreva.52.3457. URL: http://dx.doi.org/10.1103/PhysRevA.52.3457 (cit. on p. 49).

[BCE+24] A. M. Brańczyk, A. Carrera Vazquez, D. J. Egger, B. Fuller, J. Gacon, J. R. Garrison, J. R. Glick, C. Johnson, S. Joshi, E. Pednault, C. D. Pemmaraju, P. Rivero, I. Shehzad, S. Woerner. *Qiskit addon: circuit cutting*. https://github.com/Qiskit/qiskit-addon-cutting. 2024. DOI: 10.5281/zenodo.7987997 (cit. on pp. 23, 24, 45, 61).

[BPK23] S. Brandhofer, I. Polian, K. Krsulich. *Optimal Partitioning of Quantum Circuits using Gate Cuts and Wire Cuts*. 2023. arXiv: 2308.09567 [quant-ph]. URL: https://arxiv.org/abs/2308.09567 (cit. on pp. 18, 23, 53).

[BPS23] L. Brenner, C. Piveteau, D. Sutter. *Optimal wire cutting with classical communication*. 2023. arXiv: 2302.03366 [quant-ph]. URL: https://arxiv.org/abs/2302.03366 (cit. on pp. 22, 23, 53).

[CCJ24] P.-H. Chen, D.-W. Chiou, J.-H. R. Jiang. *Enhanced Quantum Circuit Cutting Framework for Sampling Overhead Reduction*. 2024. arXiv: 2412.17704 [quant-ph]. URL: https://arxiv.org/abs/2412.17704 (cit. on p. 53).

[CDKM04] S. A. Cuccaro, T. G. Draper, S. A. Kutin, D. P. Moulton. *A new quantum ripple-carry addition circuit*. 2004. arXiv: quant-ph/0410184 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/0410184 (cit. on p. 49).

[CJGM21] X. Chou, U. Junior Mele, L. M. Gambardella, R. Montemanni. "Re-Initialising Solutions in a Random Restart Local Search for the Probabilistic Orienteering Problem". In: *Proceedings of the 2021 8th International Conference on Industrial Engineering and Applications (Europe)*. ICIEA 2021-Europe. Barcelona, Spain: Association for Computing Machinery, 2021, pp. 153–158. ISBN: 9781450389921. DOI: 10.1145/3463858.3463895. URL: https://doi.org/10.1145/3463858.3463895 (cit. on p. 41).

[DKPW20] R. Duncan, A. Kissinger, S. Perdrix, J. van de Wetering. "Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus". In: *Quantum* 4 (June 2020), p. 279. ISSN: 2521-327X. DOI: 10.22331/q-2020-06-04-279. URL: https://doi.org/10.22331/q-2020-06-04-279 (cit. on pp. 27–29).

[DP09] R. Duncan, S. Perdrix. *Mathematical Theory and Computational Practice: 5th Conference on Computability in Europe, CiE 2009, Heidelberg, Germany, July 19-24, 2009. Proceedings*. Springer Berlin Heidelberg, 2009. ISBN: 9783642030734. DOI: 10.1007/978-3-642-03073-4. URL: http://dx.doi.org/10.1007/978-3-642-03073-4 (cit. on p. 27).

[DZH21] D. Dadkhah, M. Zomorodi, S. E. Hosseini. “A New Approach for Optimization of Distributed Quantum Circuits”. In: *International Journal of Theoretical Physics* 60.9 (Sept. 2021), pp. 3271–3285. ISSN: 1572-9575. DOI: 10.1007/s10773-021-04904-y. URL: https://doi.org/10.1007/s10773-021-04904-y (cit. on p. 53).

[Gro96] L. K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. arXiv: quant-ph/9605043 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/9605043 (cit. on p. 17).

[Hol92] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, Apr. 1992. ISBN: 9780262275552. DOI: 10.7551/mitpress/1090.001.0001. URL: https://doi.org/10.7551/mitpress/1090.001.0001 (cit. on p. 42).

[IHI+24] T. Ichikawa, H. Hakoshima, K. Inui, K. Ito, R. Matsuda, K. Mitarai, K. Miyamoto, W. Mizukami, K. Mizuta, T. Mori, Y. Nakano, A. Nakayama, K. N. Okada, T. Sugimoto, S. Takahira, N. Takemori, S. Tsukano, H. Ueda, R. Watanabe, Y. Yoshida, K. Fujii. “Current numbers of qubits and their uses”. In: *Nature Reviews Physics* 6.6 (May 2024), pp. 345–347. ISSN: 2522-5820. DOI: 10.1038/s42254-024-00725-0. URL: http://dx.doi.org/10.1038/s42254-024-00725-0 (cit. on p. 17).

[Jeb13] K. Jebari. “Selection Methods for Genetic Algorithms”. In: *International Journal of Emerging Sciences* 3 (Dec. 2013), pp. 333–344. URL: https://www.researchgate.net/publication/259461147_Selection_Methods_for_Genetic_Algorithms (cit. on p. 42).

[JTK+24] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, J. M. Gambetta. *Quantum computing with Qiskit*. 2024. DOI: 10.48550/arXiv.2405.08810. arXiv: 2405.08810 [quant-ph] (cit. on p. 45).

[KDP+24] S. Kan, Z. Du, M. Palma, S. A. Stein, C. Liu, W. Wei, J. Chen, A. Li, Y. Mao. *Scalable Circuit Cutting and Scheduling in a Resource-constrained and Distributed Quantum System*. 2024. arXiv: 2405.04514 [quant-ph]. URL: https://arxiv.org/abs/2405.04514.

[KGV83] S. Kirkpatrick, C. Gelatt, M. Vecchi. “Optimization by Simulated Annealing”. In: *Science (New York, N.Y.)* 220 (June 1983), pp. 671–80. DOI: 10.1126/science.220.4598.671 (cit. on p. 40).

[KLW+24] S. Kan, Y. Li, H. Wang, S. Mouradian, Y. Mao. *Circuit Folding: Modular and Qubit-Level Workload Management in Quantum-Classical Systems*. 2024. arXiv: 2412.18705 [quant-ph]. URL: https://arxiv.org/abs/2412.18705 (cit. on p. 53).

[KPJT25] K. Karuppasamy, V. Puram, S. Johnson, J. P. Thomas. “A Comprehensive Review of Quantum Circuit Optimization: Current Trends and Future Directions”. In: *Quantum Reports* 7.1 (Jan. 2025), p. 2. ISSN: 2624-960X. DOI: 10.3390/quantum7010002. URL: http://dx.doi.org/10.3390/quantum7010002 (cit. on p. 31).

[Kru22] R. Krueger. *Vanishing 2-Qubit Gates with Non-Simplification ZX-Rules*. 2022. arXiv: 2209.06874 [quant-ph]. URL: https://arxiv.org/abs/2209.06874 (cit. on pp. 27, 28, 31).

[KW20a] A. Kissinger, J. van de Wetering. "PyZX: Large Scale Automated Diagrammatic Reasoning". In: *Electronic Proceedings in Theoretical Computer Science* 318 (May 2020), pp. 229–241. ISSN: 2075-2180. DOI: 10.4204/eptcs.318.14. URL: http://dx.doi.org/10.4204/EPTCS.318.14 (cit. on p. 45).

[KW20b] A. Kissinger, J. van de Wetering. "Reducing the number of non-Clifford gates in quantum circuits". In: *Physical Review A* 102.2 (Aug. 2020). ISSN: 2469-9934. DOI: 10.1103/physreva.102.022406. URL: http://dx.doi.org/10.1103/PhysRevA.102.022406 (cit. on pp. 46–48).

[LB20] F. Leymann, J. Barzen. "The bitter truth about gate-based quantum algorithms in the NISQ era". In: *Quantum Science and Technology* 5.4 (Sept. 2020), p. 044007. ISSN: 2058-9565. DOI: 10.1088/2058-9565/abae7d. URL: http://dx.doi.org/10.1088/2058-9565/abae7d (cit. on p. 17).

[NC10] M. A. Nielsen, I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. URL: https://doi.org/10.1017/CBO9780511976667 (cit. on pp. 17, 19, 21, 24, 25).

[Ngu25] D. A. Nguyen. *Quantum Circuit Optimization for Circuit Cutting*. https://github.com/ngydc/qcofcc-public. 2025 (cit. on p. 45).

[NRS+18] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, D. Maslov. "Automated optimization of large quantum circuits with continuous parameters". In: *npj Quantum Information* 4.1 (May 2018). ISSN: 2056-6387. DOI: 10.1038/s41534-018-0072-4. URL: http://dx.doi.org/10.1038/s41534-018-0072-4 (cit. on pp. 32, 34).

[PHOW20] T. Peng, A. W. Harrow, M. Ozols, X. Wu. "Simulating Large Quantum Circuits on a Small Quantum Computer". In: *Physical Review Letters* 125.15 (Oct. 2020). ISSN: 1079-7114. DOI: 10.1103/physrevlett.125.150504. URL: http://dx.doi.org/10.1103/PhysRevLett.125.150504 (cit. on pp. 17, 22).

[Pre18] J. Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. URL: http://dx.doi.org/10.22331/q-2018-08-06-79 (cit. on p. 17).

[Sho97] P. W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. URL: http://dx.doi.org/10.1137/S0097539795293172 (cit. on p. 17).

[SPS25] L. Schmitt, C. Piveteau, D. Sutter. "Cutting circuits with multiple two-qubit unitaries". In: *Quantum* 9 (Feb. 2025), p. 1634. ISSN: 2521-327X. DOI: 10.22331/q-2025-02-18-1634. URL: http://dx.doi.org/10.22331/q-2025-02-18-1634 (cit. on pp. 17, 22).

[TTS+21] W. Tang, T. Tomesh, M. Suchara, J. Larson, M. Martonosi. "CutQC: using small Quantum computers for large Quantum circuit evaluations". In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '21. ACM, Apr. 2021. DOI: 10.1145/3445814.3446758. URL: http://dx.doi.org/10.1145/3445814.3446758 (cit. on p. 23).

[VBE96] V. Vedral, A. Barenco, A. Ekert. "Quantum networks for elementary arithmetic operations". In: *Physical Review A* 54.1 (July 1996), pp. 147–153. ISSN: 1094-1622. DOI: 10.1103/physreva.54.147. URL: http://dx.doi.org/10.1103/PhysRevA.54.147 (cit. on p. 61).

[Wet20] J. van de Wetering. *ZX-calculus for the working quantum computer scientist*. 2020. arXiv: 2012.13966 [quant-ph]. URL: https://arxiv.org/abs/2012.13966 (cit. on pp. 24–27, 29).

[Zha19] B. Zhang. *Coconuts and Islanders: A Statistics-First Guide to the Boltzmann Distribution*. 2019. arXiv: 1904.04669 [cond-mat.stat-mech]. URL: https://arxiv.org/abs/1904.04669 (cit. on p. 41).

All links were last followed on June 5, 2025.

A Appendix

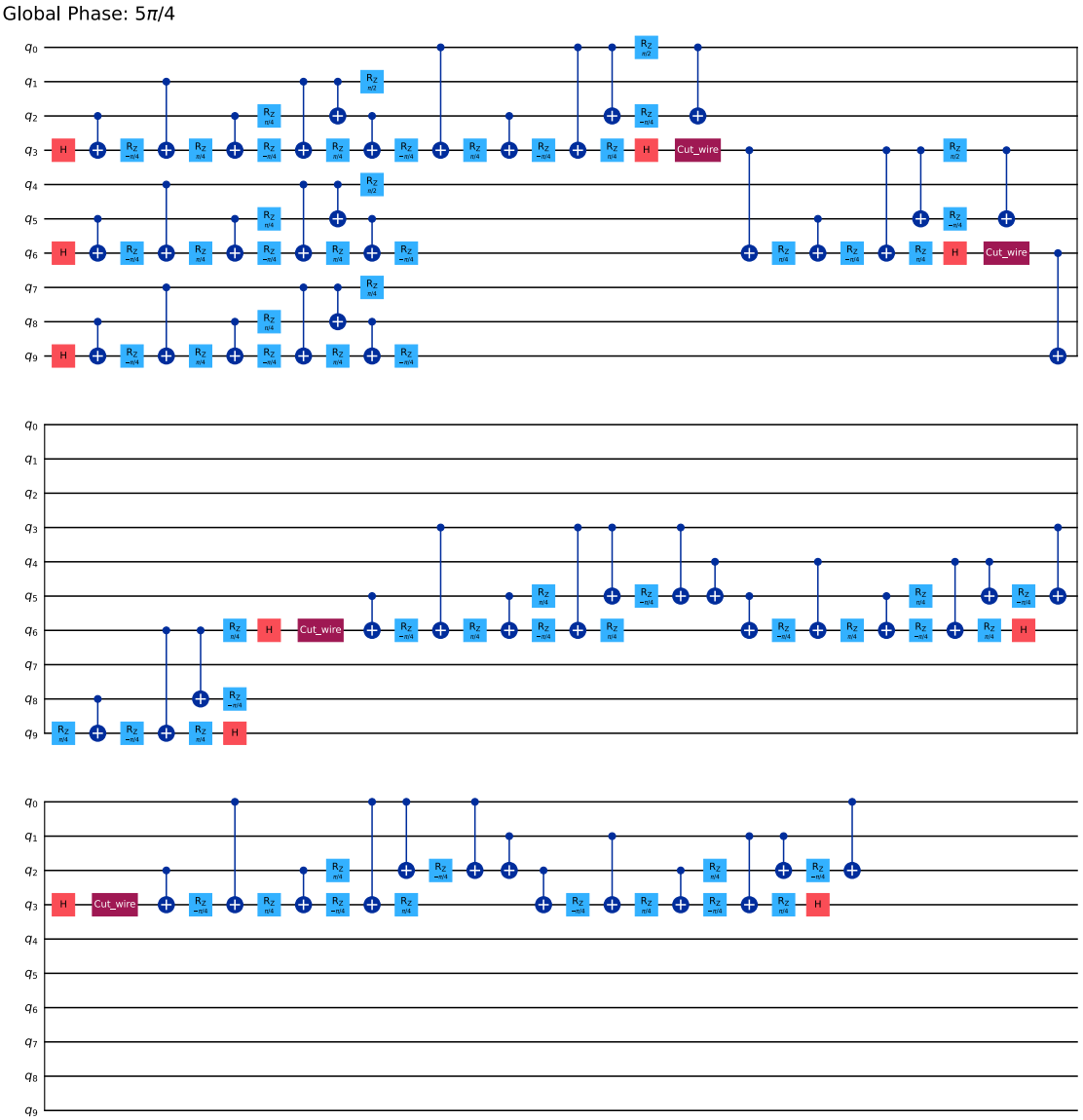


Figure A.1: VBE-Adder-3 circuit [VBE96] with cut placements suggested by qiskit-addon-cutting [BCE+24]. The original circuit is transpiled using the basis gates $\{H, X, Z, CX, CZ, RX, RZ\}$. Four wire cuts are placed (indicated by the darker red gates in the circuit), which results in a total sampling overhead of 65536.

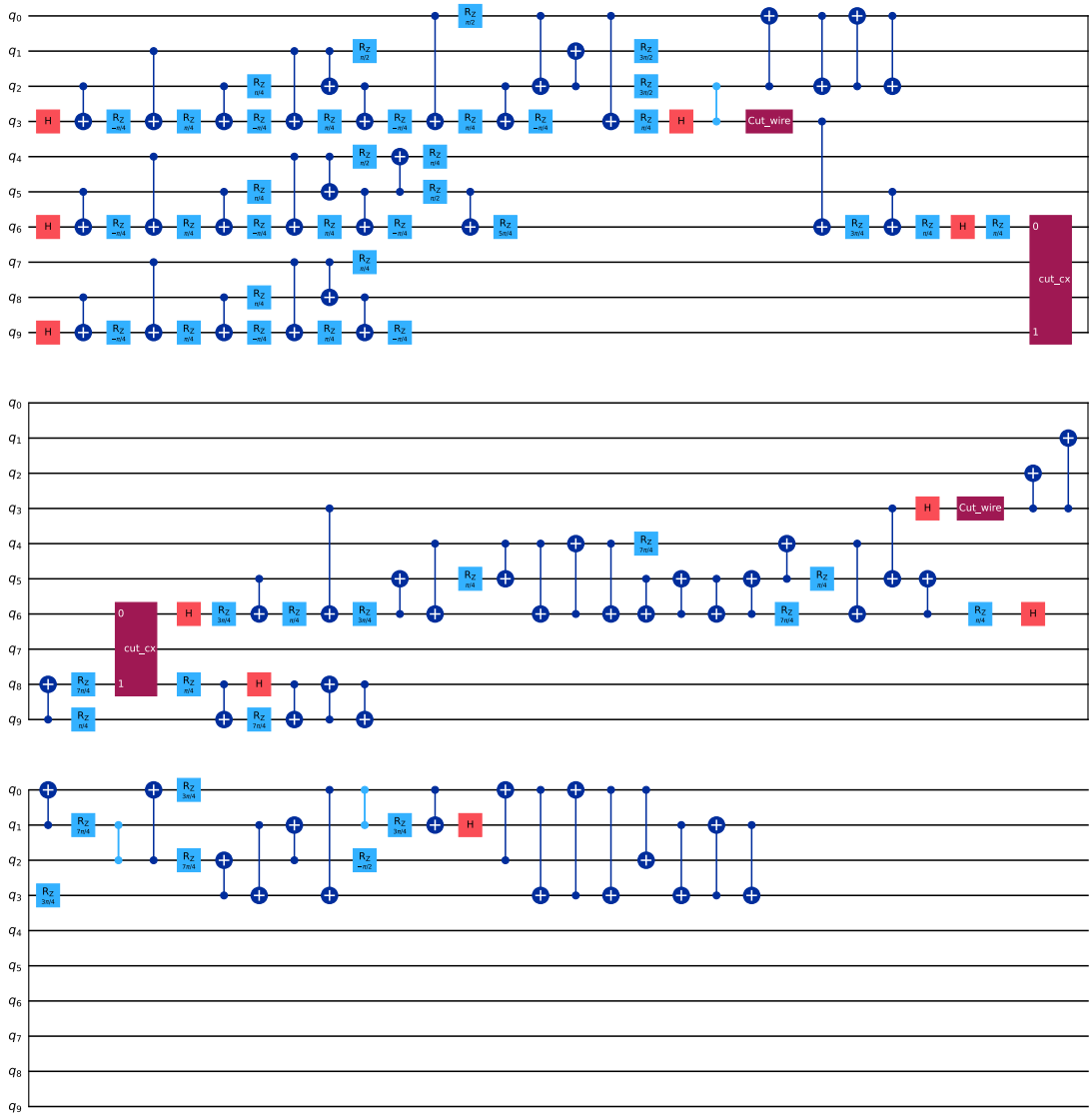


Figure A.2: Optimized VBE-Adder-3 circuit and suggested cut placements. Optimized with simulated annealing and `Random-Section-Rewrite` technique. The qiskit-addon-cutting found new cut configurations: Two wire cuts were replaced by two CX gate cuts, driving the total sampling overhead down to 20736.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature