Vanishing 2-Qubit Gates with Non-Simplification ZX-Rules

Ryan Krueger
Christ Church College
University of Oxford

A thesis submitted for the degree of
Master of Science in Mathematical Sciences

Trinity 2021

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Acknowledgements

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Abstract

Traditional quantum circuit optimization is performed directly at the circuit level. Alternatively, a quantum circuit can be translated to a ZX-diagram which can be simplified using the rules of the ZX-calculus, after which a simplified circuit can be extracted. However, the best-known extraction procedures can drastically increase the number of 2-qubit gates. In this work, we take advantage of the fact that local changes in a ZX-diagram can drastically affect the complexity of the extracted circuit. We use a pair of congruences (i.e., non-simplification rewrite rules) based on the graph-theoretic notions of local complementation and pivoting to generate local variants of a simplified ZX-diagram. We explore the space of equivalent ZX-diagrams generated by these congruences using simulated annealing and genetic algorithms to obtain a simplified circuit with fewer 2-qubit gates. Our method can reliably outperform state-of-the-art optimization techniques for low-qubit (< 10) circuits and serves as a proof-of-concept for a new circuit optimization strategy in the ZX-calculus.

Contents

Chapter 1

Introduction

Quantum computing arose from the idea that the "bugs" of quantum mechanics (e.g., superposition) may serve as useful features for information processing. Since this framing was posed in the late 1970's, a rich theory of quantum information has developed that treats a *qubit*, a superposition of two classical bits, as the basic unit of information. Quantum information science is an exciting field of work as many quantum algorithms have been discovered that offer significant speedups from their classical counterparts (e.g., Shor's algorithm for integer factorization). However, like the formulation of the Turing machine decades before the invention of the transistor, efforts to engineer a real-world quantum computer that can implement these theories are in their infancy.

The current state-of-the-art are so-called noisy intermediate-scale quantum (NISQ) computers [7]. These are devices without enough qubits to spare for error correction (i.e., noisy) and with a relatively small qubit number (i.e., intermediate-scale). With ˜50 qubits, NISQ devices cannot run particularly resource-intensive quantum algorithms (e.g., Shor's algorithm at a meaningful scale) but do afford the occasional speedup over classical computers. To stress these capabilities, there is significant effort towards minimizing *quantum circuits*, the standard model for a quantum computation, to mitigate noise and decoherence. More formally, this field is known as *quantum circuit optimization*.

The ZX-calculus, a graphical formalism for quantum circuits rooted in category theory, provides a unique setting for circuit optimization. In this language, quantum circuits are represented as *ZX-diagrams* that are manipulated via a set of rewrite rules. ZX-diagrams provide a lower-level representation for quantum computations and can be deformed arbitrarily, enriching the space of possible simplifications. While there is no known procedure for recovering a quantum circuit from a general ZX-diagram, subsets of transformations have been identified that preserve circuit extractability.

These transformations have enabled a suite of circuit optimization procedures in the ZX-calculus (e.g., for reducing T-count) with various tradeoffs.

Still, circuit extraction from a ZX-diagram remains a bottleneck for optimization; the best known extraction procedures can introduce an unwieldy degree of complexity in the resulting circuit. For example, circuit extraction can introduce many CNOT gates which are particularly problematic on NISQ devices. Interestingly, however, closely related ZX-diagrams can induce drastically different circuits. Current optimization procedures employing the ZX-calculus do not exploit this fact (e.g., by searching local variants) but rather return a ZX-diagram (or its associated circuit) once no more graph simplifications can be applied.

In this thesis, we explore the role that such a search over local variants of ZX-diagrams could play in circuit optimization. More specifically, we apply various search procedures (i.e., simulated annealing and genetic algorithms) over the application of rewrite rules that do not necessarily simplify the graph itself but may reduce the complexity of the associated circuit. When seeding this search with a pre-simplified ZX-diagram (obtained via off-the-shelf methods), we find that FIXME.

Chapter 2

Background

To understand quantum circuit optimization in the ZX-calculus, one first must understand quantum circuits and the ZX-calculus. In this section we provide the requisite background in both of these, preceded by a brief discussion on qubits. We conclude with an overview of quantum circuit optimization in the ZX-calculus.

2.1 Qubits

A classical bit has a value of 0 or 1. A quantum bit, or *qubit*, encodes a quantum superposition of these two values and therefore more information than a classical bit. In the traditional bra-ket notation of quantum mechanics, a qubit is represented by a vector in $\mathbb{C}^2$,

$$|\psi\rangle = \lambda_1|0\rangle + \lambda_2|1\rangle = \begin{pmatrix}\lambda_1\\ \lambda_2\end{pmatrix}$$

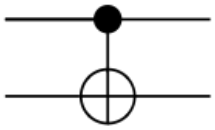
where

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

and λ_1 and λ_2 are the probability amplitudes of observing $|0\rangle$ and $|1\rangle$ upon measurement in this basis, respectively. Notably, qubits can be written with respect to any orthonormal basis. The set $\{|0\rangle, |1\rangle\}$ is known as the *computational* basis. Another common basis is the +/- basis, where

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$
$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

(a) The Hadamard gate.



(b) The CNOT gate. The top and bottom qubits are the control and target, respectively.

Figure 2.1: The circuit representation of two common quantum logic gates. Wires represent qubits and boxes represent quantum logic gates. Quantum circuits are read from left to right.

The computational and +/- bases are also known as the Z and X bases, respectively.

More generally, a system of n qubits is represented by the tensor product of the individual states. So, an n-qubit state corresponds to a 2^n-dimensional vector. For example, the two-qubit state with both qubits in $|0\rangle$ is

$$|0\rangle \otimes |0\rangle = |00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

In the ZX-calculus and this work, qubits are abstracted as wires in string diagrams. However, this definition is included both for completeness and to aid understanding of quantum circuits.

2.2 Quantum Circuits

A quantum circuit models a quantum computation as a sequence of discrete gates. In the traditional notation with qubits as vectors in $\mathbb{C}^2$, quantum gates correspond to unitary matrices. For example, the Hadamard gate, a quantum gate that acts on a single qubit and maps

$$|0\rangle \mapsto \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$
$$|1\rangle \mapsto \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

has the matrix form

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

Figure 2.1a depicts the circuit representation of the Hadamard gate. Note that $|+\rangle = H|0\rangle$ and $|-\rangle = H|1\rangle$. An important family of single-qubit gates are the *phase shift*

gates that modify the phase of the quantum state by some α. A phase shift gate maps

$$|0\rangle \mapsto |0\rangle$$
$$|1\rangle \mapsto e^{i\alpha}|1\rangle$$

and has the matrix form

$$R_\alpha = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{bmatrix}$$

Examples of phase gates are the T-gate ($\alpha = \frac{\pi}{4}$) and the S-gate ($\alpha = \frac{\pi}{2}$). Phase gates where α is a multiple of $\pi/2$ are called *Clifford* gates.

Quantum gates are not restricted to acting on single qubits. For example, the controlled-NOT (CNOT) gate acts on two qubits and flips the second (target) qubit only when the first (control) qubit is $|1\rangle$. The CNOT gate is typically depicted as shown in Figure 2.1b and has the following matrix form:

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

In general, an n-qubit gate is represented by a 2^n-dimensional unitary. However, attention is typically restricted to single- and 2-qubit gates as it is well known that the Clifford + T gate set, consisting of the H, T, S, and CNOT gates, is *universal* – any other operation can be represented by a finite sequence from this set. The *Clifford circuits* are those that can be generated by $\{H, S, CNOT\}$. As the T and S gates are instances of phase shift gates, $\{H, R_\alpha, CNOT\}$ is also universal though in practice any non-Clifford phase gate is implemented with T gates.

Quantum gates can be composed sequentially or in parallel. Consider two gates A and B. The effect of B applied in series after A can be described by a single gate (i.e., linear map) via matrix multiplication (i.e., $B \cdot A$). Alternatively, the tensor product is used to describe A and B in parallel (i.e., $B \otimes A$). For example, the simple circuit shown in Figure 2.3a can be described by the following linear map:

$$(\mathbb{I} \otimes CNOT) \cdot (\mathbb{I} \otimes H \otimes \mathbb{I}) \cdot (CNOT \otimes \mathbb{I}) \cdot (S \otimes \mathbb{I} \otimes \mathbb{I})$$

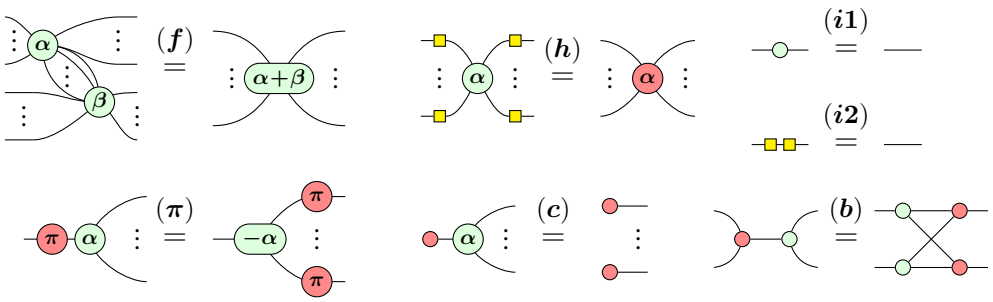
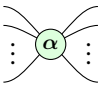


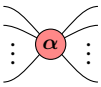
Figure 2.2: The rules of the ZX-calculus. All rules hold with the colors interchanged and for $\alpha, \beta \in [0, 2\pi)$.

2.3 ZX-Calculus

The ZX-calculus is a graphical language for representing and reasoning about quantum processes. Quantum processes are represented by *ZX-diagrams* which consist of *wires* and *spiders*. Spiders can have an arbitrary number of inputs and outputs and come in two flavors: Z spiders (shown in green) and X spiders (shown in red). As with quantum circuits, a ZX-diagram is interpreted from left to right.

ZX-diagrams abstract away the complexities of linear algebra and tensor products from traditional formalisms of quantum processes. Wires still represent qubits while spiders provide a general form for linear maps, where

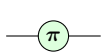
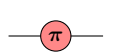

$:= |0...0\rangle\langle 0...0| + e^{i\alpha} |1...1\rangle\langle 1...1|$


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

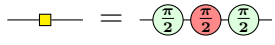
Two diagrams can be composed in serial by joining the outputs of the first to the inputs of the second, or in parallel by stacking the two diagrams.

From this definition, we can identify the diagrammatic represenations for several common components of quantum computation:

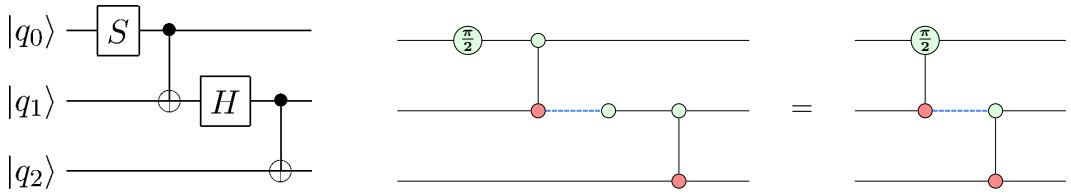

$= |0\rangle + |1\rangle \propto |+\rangle$ 
$= |+\rangle + |-\rangle \propto |0\rangle$


$= |0\rangle\langle 0| - |1\rangle\langle 1| = Z$ 
$= |+\rangle\langle +| - |-\rangle\langle -| = X$

where Z and X are the corresponding Pauli matrices. Note that spiders without any inputs can be regarded as qubit state preparations. The Hadamard gate is so pervasive that it merits shorthand notation; we use a yellow square to represent the Hadamard gate as shown below

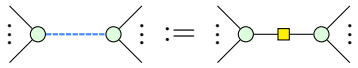
 (2.1)

and replace a Hadamard gate between two spiders with a blue dashed edge:

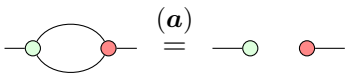


(a) Traditional notation (b) ZX-diagram

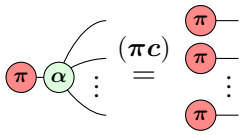
Figure 2.3: A simple 3-qubit quantum circuit shown in both the traditional notation and as a ZX-diagram.



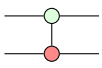
We can also reason about ZX-diagrams. The first rule for determining equality is that *only connectivity matters* (OCM). In other words, two ZX-diagrams are equal when one can be deformed into the other (e.g., via moving vertices around in the plane or bending wires) while maintaining connectivity and the order of the inputs and outputs. In addition to this OCM principle, the ZX-calculus includes a primary set of rewrite rules as shown in Figure 2.2. These rules only hold up to non-zero scalar factors, though these scalars are typically ignored as they correspond to negligible differences in global phase. Additional rules can be derived from this set, such as the antipode rule



and the π-copy rule



All quantum circuits can be translated to ZX-diagrams. This is a consequence of the fact that the following universal gate set can be easily represented in the ZX-calculus:

$CNOT =$  $R_\alpha =$  $H =$

In the CNOT diagram, the green and red spiders are the control and target qubits, respectively. As an example, Figure 2.3 depicts both the traditional and diagrammatic representations of a simple quantum circuit.

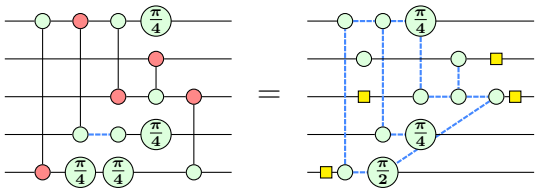


Figure 2.4: An example of a ZX-diagram and an equivalent, graph-like ZX-diagram.

2.4 Circuit Optimization in the ZX-Calculus

The goal of quantum circuit optimization is to simplify quantum circuits (e.g., via reducing the number of gates) to reduce noise and decoherence when run on modern quantum computers. More specifically, the T-count and 2-qubit-count are typically targeted; fault tolerant implementations of T gates generally require an order of magnitude more resources than Clifford gates [2], and the fidelity of single-qubit gates is demonstrably higher than that of 2-qubit gates [1]. It is standard to perform optimizations at the circuit-level. Examples of circuit-level optimizations are 3-qubit gate decomposition and cancelling back-to-back CNOT gates.

The ZX-calculus enables circuit optimization at the graph-level, a more flexible setting that is well-studied in its own right. Such an optimization involves converting a circuit to a ZX-diagram, simplifying the diagram, and converting the diagram back to a circuit. However, there is no known general-purpose procedure for recovering a quantum circuit from a generic ZX-diagram. This restricts the space of permissible simplifications to those that preserve whichever diagrammatic properties are required by the extraction procedure.

In 2020, Duncan et al. reported an extraction procedure that enabled current best practices [3]. Firstly, this procedure operates on *graph-like* ZX-diagrams. A ZX-diagram is graph-like when:

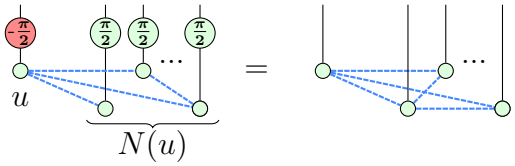
1. All spiders are Z-spiders.
2. Z-spiders are only connected via Hadamard edges
3. There are no parallel Hadamard edges or self-loops
4. Every input or output is connected to a Z-spider and every Z-spider is connected to at most one input or output

Every ZX-diagram is equal to a graph-like diagram (see Figure 2.4 for an example), and this form admits an underlying graph structure that permits graph-theoretic

analyses. Secondly, this procedure requires that the underlying graph of the input ZX-diagram satisfies a graph-theoretic invariant called *focused generalized flow* (focused gFlow). Given such an input ZX-diagram, extraction proceeds in such a way that each non-zero phase corresponds to one phase gate in the resulting circuit but each edge can correspond to multiple CNOTs. For this reason, local changes in a ZX-diagram (e.g., connectivity) can have significant effects on the complexity of the associated circuit. Importantly, this extraction procedure is not optimal; if a given circuit is converted to a ZX-diagram and the extraction procedure is immediately applied, the extracted circuit is often more complex than the original circuit. In particular, this procedure can drastically increase the number of 2-qubit gates. For more details on the extraction procedure or focused gFlow, see [3].

Alongside this extraction procedure, Duncan et al. also introduced an optimization procedure that relies on focused gFlow-preserving graph simplifications. More specifically, this procedure relies on two graph-theoretic transformations that each correspond to rewrite rules in the ZX-calculus: *local complementation* (LC) and *pivoting*. Given a graph G and some vertex u of G, the *local complementation* of G according to u, denoted $G \star u$, is the graph in which the connectivity of all pairs of neighbors of u is inverted. For example,

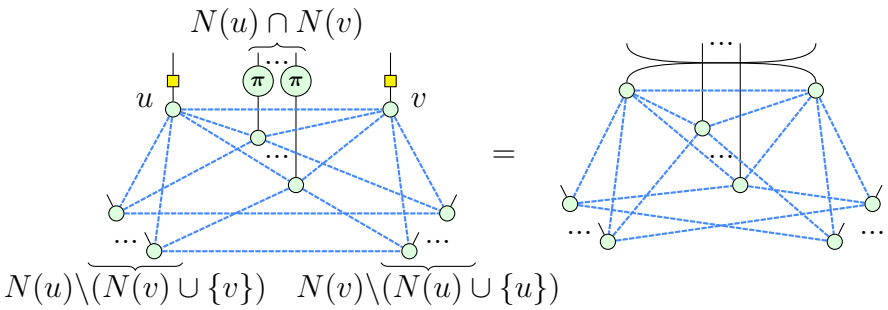
Given a ZX-diagram, local complementation can be induced in the underlying graph structure while maintaining equality by applying an $X_{-\pi/2}$ gate on (the spider corresponding to) u and a $Z_{\pi/2}$ gate to its neighbors [4]:



(2.2)

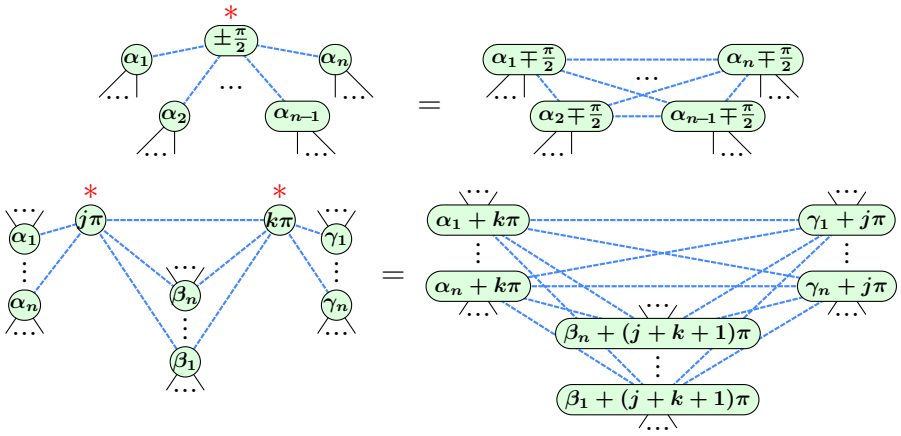
Relatedly, given two connected vertices u and v in G, the *pivot* of G along the edge uv is the graph $G \wedge uv := G \star u \star v \star u$. In practice, this consists of complementing the edges between three subsets of vertices: (1) the common neighborhood of u and v, (2) the exclusive neighborhood of u, and (3) the exclusive neighborhood of v:

Again, given a ZX-diagram, the rewrite rule reported in [5] introduces a pivot by applying Hadamard gates on u and v, and Z_π gates on their common neighborhood:



(2.3)

Each rewrite rule can be extended to a simplification by performing local complementation (resp. pivoting), removing the vertex (resp. the pair of vertices), and updating the phases:



Importantly, the existence of a focused gFlow is preserved in both cases. At a high-level, optimization can then be performed by applying these simplifications to fixpoint and extracting a circuit from the resulting ZX-diagram. Details on this optimization procedure and proofs of the simplification rules or their focused gFlow-preservation can be found in [3]. A variant of this procedure is implemented in the `full_reduce` method of the PyZX library.

One issue with `full_reduce` is its reliance on circuit extraction; it is not uncommon that the circuit extracted from the final, simplified ZX-diagram has more gates than the input circuit. Kissinger and van de Wetering introduced an alternative optimization method that uses *phase teleportation* to sidestep the issue of circuit extraction altogether [6]. Crucially, simplification of the ZX-diagram is performed symbollically. This symbolic simplification leverages the fact that several rewrite rules involving *phase-gadgets*, a particular ZX-diagrammatic motif, correspond to changes in the original phases. Therefore, non-Clifford phases can potentially cancel or combine with each other while leaving the original graphical structure of the ZX-diagram

intact. So, as simplification proceeds, application of these rules is simply tabulated to enable reconstruction of a final circuit with the same connectivity but a possible reduction in non-Clifford gates (and therefore T-count). Importantly, using phase teleportation rather than circuit extraction ensures that the number of 2-qubit and Hadamard gates is unchanged. By changing the angles of many non-Clifford phase gates to either 0 or multiples of $\pi/2$, this procedure can also make a circuit-level optimization procedure much more effective. This procedure is implemented in the `teleport_reduce` method of PyZX.

In both cases, it is standard to apply a set of circuit-level optimizations post-simplification. This set of optimizations is implemented in the `basic_optimization` routine in PyZX. Note that the circuit must first be extracted after `full_reduce` whereas these optimizations can be applied directly after `teleport_reduce`. In summary, the two standard circuit optimization pipelines in the ZX-calculus are `full_reduce` + `extract_circuit` + `basic_optimization` and `teleport_reduce` + `basic_optimization`.

Chapter 3

Methods

At a high level, the goal of any quantum circuit optimization is to reduce the complexity of the input circuit. A core assumption of circuit optimization in the ZX-calculus is that ZX-diagrams with fewer spiders typically correspond to simpler circuits. Earlier, however, we noted that local changes (e.g., connectivity) in a ZX-diagram can drastically affect the complexity of the associated circuit obtained via extraction. However, no existing optimization procedure over ZX-diagrams addresses this and searches this local space. The `full_reduce` method described in Section 2.4 is the quintessential example of this lost opportunity. After the two simplification rules can no longer be applied, the final circuit is taken to be that which is extracted from the resulting simplified ZX-diagram; however, there may be an equivalent ZX-diagram lurking nearby whose associated circuit is markedly less complex.

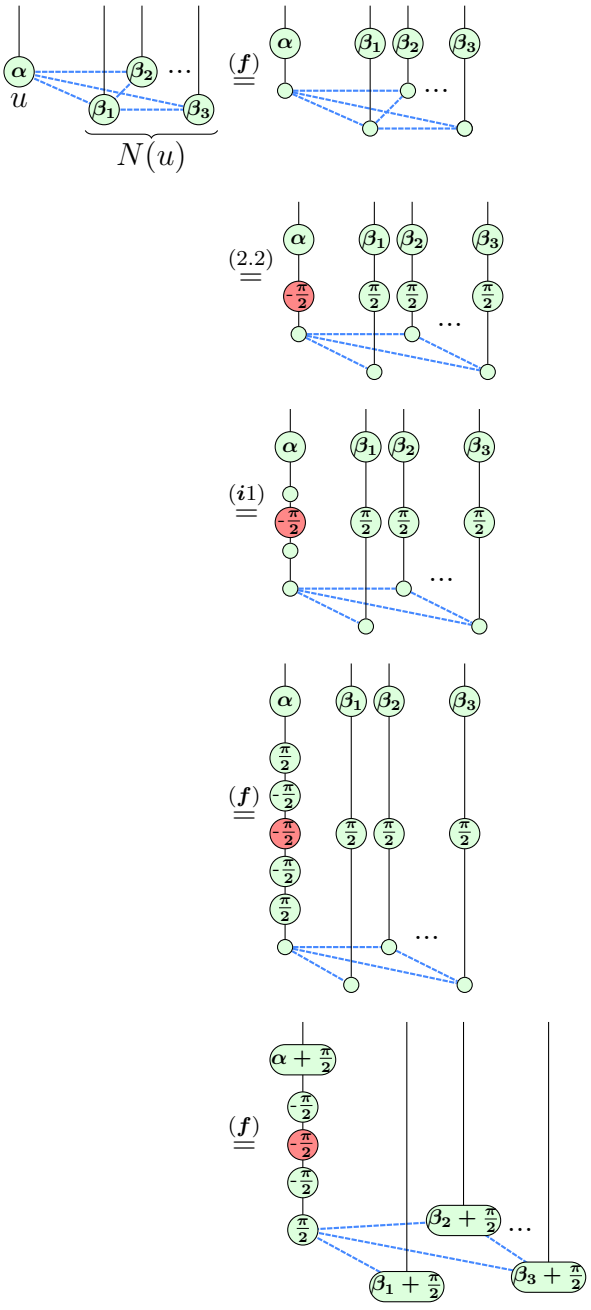
In this thesis, we explore the utility of searching this local space of equivalent ZX-diagrams. To do so, we first need rewrite rules that modify a ZX-diagram in a manner other than spider removal that may reduce circuit complexity. Indeed, a rule that introduces several additional spiders alongside changes in connectivity may in turn yield a less complex circuit. We refer to these rewrite rules as *congruences*. Given a set of congruences, we then need a procedure to search the space of equivalent ZX-diagrams generated by an input ZX-diagram and these rules. We can then devise strategies for incorporating this local search into existing methods for circuit optimization in the ZX-calculus.

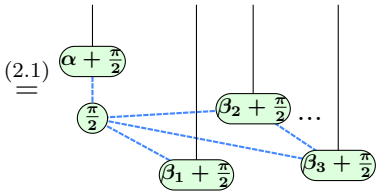
We present the methods of our work in this order. First, we generalize the original (non-simplification) variants of local complementation and pivoting to ZX-diagrams with arbitrary phases. We then describe two search procedures, simulated annealing (SA) and genetic algorithms (GA). We also define a measure of circuit complexity and discuss candidate objective functions to guide search. Lastly, we discuss how this search can be incorporated into existing optimization pipelines.

3.1 Congruences

The non-simplification versions of local complementation and pivoting presented in Section 2.4 embody the desired properties of congruences. They change the connectivity of the ZX-diagram without introducing an unwieldy number of additional gates, presenting an opportunity for a potential reduction in circuit complexity. However, Equations 2.2 and 2.3 only apply to spiders with zero phase and a single wire.

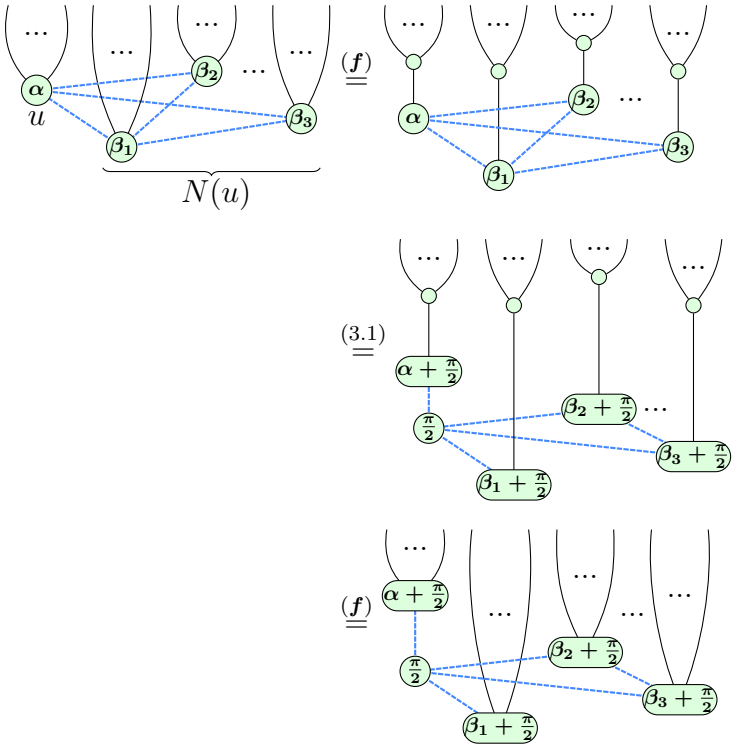
We can apply the rules of the ZX-calculus and Equation 2.2 to generalize local complementation to arbitrary phases ($\alpha_i, \beta_i \in [0, 2\pi)$):





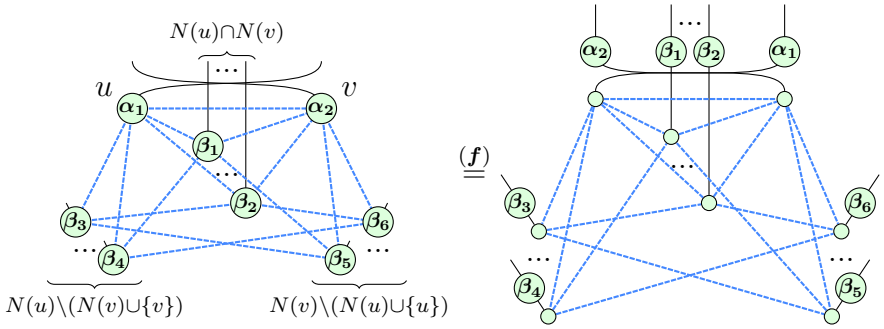
(3.1)

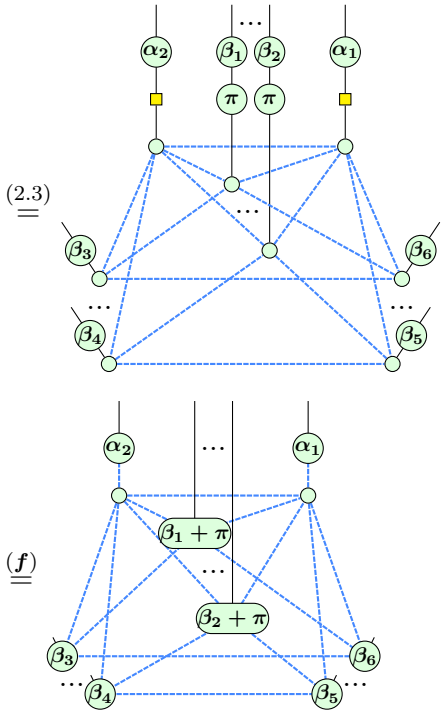
Equation 3.1 can be easily extended to apply for an arbitrary number of wires connected to each spider:



(C1)

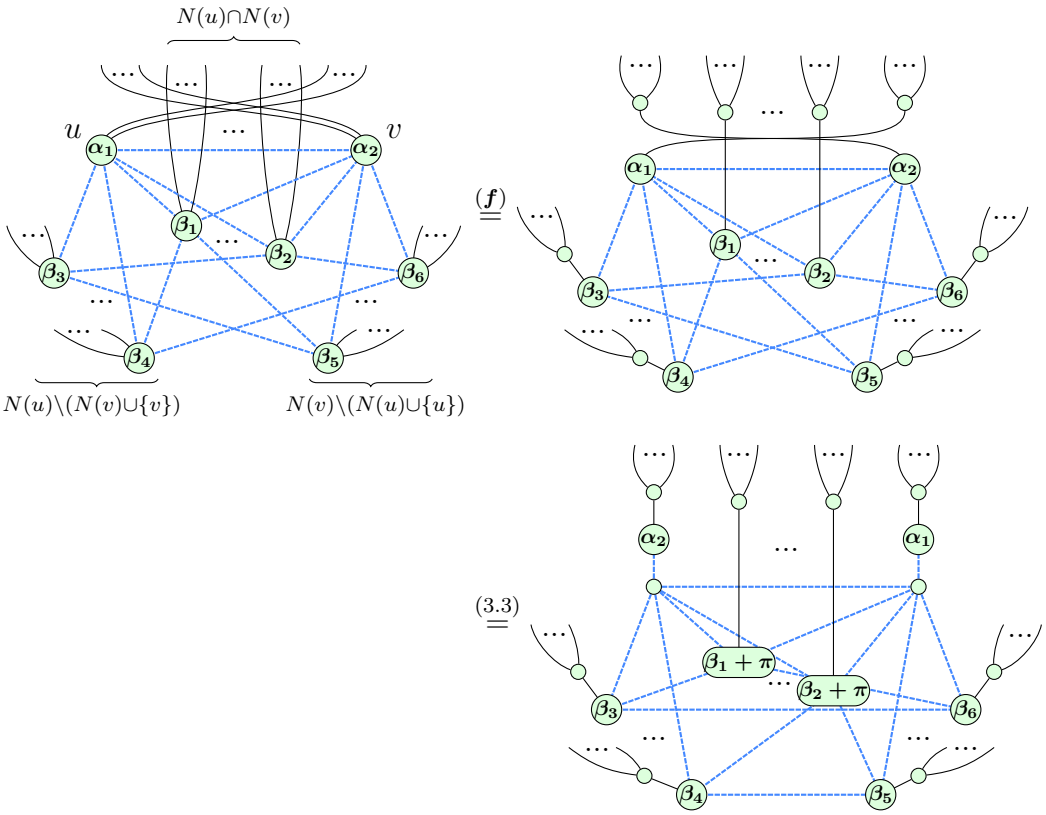
Similarly, we can generalize pivoting to arbitrary phases:

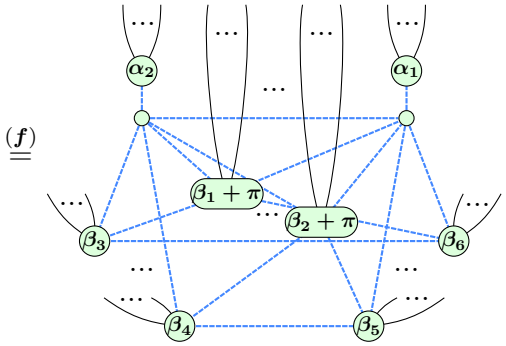




(3.3)

Again, we can extend this rewrite rule for arbitrary wiring:





(C2)

Equations C1 and C2 will serve as our primary congruences because they maintain a similar graph complexity (e.g., number of spiders and wires) but, due to their effects on connectivity, can produce circuits of varying complexities. Equation C1 can be applied to any spider with a degree of more than 1. Equation C2 can be applied to any pair of connected spiders.

3.2 Search Procedures

The space of ZX-diagrams generated by an input ZX-diagram and these congruences is combinatorial and therefore cannot be searched exhaustively. We can instead formulate this search as an optimization problem where the set of isomorphic ZX-diagrams are the states (reachable by congruence applications) that are evaluated by some quantitative measure of circuit complexity. Here we describe two search procedures as well as several candidate objective functions that either measure circuit complexity directly or use ZX-diagrammatic properties as a proxy.

3.2.1 Simulated Annealing

Simulated annealing is an optimization technique named after the annealing process in metallurgy in which a molten hot metal is cooled in a slow, controlled fashion in order for it to reach its most stable form. In SA, this notion of slow cooling is interpreted as a slow decrease in the probability of accepting worse solutions. In this way, the algorithm initially explores a broad region of the the search space and progressively narrows its scope.

More formally, the SA algorithm maintains a current state s and at each step randomly considers some neighboring state s^*. The algorithm then chooses whether or not to replace s with s^* via a Bernoulli random variable parameterized by the *acceptance probability*, denoted $P(s, s^*, T)$. The acceptance probability depends on (1) the *energy* (i.e., objective) *function*, and (2) the *temperature*. The energy function

Algorithm 1 Simulated annealing

function SimAnneal(s_0, T, c, k_{max})
    $s \leftarrow s_0$
    for $i \leftarrow 0$ to k_{max} **do**
        $s^* \leftarrow$ randomly sampled neighbor of s
        if $E(s^*) < E(s)$ **or** random$(0, 1) < exp(-\frac{E(s^*)-E(s)}{T})$ **then**
            $s \leftarrow s^*$
        $T \leftarrow T * c$
    return s

$E(s)$ assigns a score to each state (lower is better). If $E(s') < E(s)$, then s is always replaced with s^* (i.e., $P(s, s^*, T) = 1$). Otherwise, $P(s, s^*, T) = exp(-\frac{E(s^*)-E(s)}{T})$ where the temperature T controls the likelihood of moving to a higher energy state. Note that $P(s, s^*, T)$ is inversely proportional to $E(s^*) - E(s)$ and directly proportional to T. Informally, this means that s^* is more likely to be accepted if it is closer in energy to s and with a higher temperature. T is initialized to some positive value and progressively decreases to zero (default: $T = 25$). At each step, T is updated as $T = T * c$ where $c \in [0, 1)$ is the *cooling* parameter (default: $c = 0.005$). In this way, SA converges towards a greedy algorithm and is more likely to accept s^* when $E(s^*) > E(s)$ early in search when T is high. The algorithm terminates when a maximum number of steps k_{max} is reached (default: $k_{max} = 1000$). Algorithm 1 provides an overview of this procedure.

In this work, the state is a ZX-diagram and neighbors are sampled via the probabilistic application of rewrite rules. For example, to anneal using the rewrite rules described in Section 3.1, s^* would be sampled by first choosing one of Equation C1 or Equation C2 and then choosing a spider or pair of connected spiders to which the rule will be applied. Note that we use Equations C1 and C2 by default but in theory any set of rewrite rules can be used.

3.2.2 Genetic Algorithms

Genetic algorithms are a class of optimization procedures inspired by the process of natural selection. Rather than maintaining a single state (e.g., as in SA), a *population* of candidate solutions is *evolved*. This evolution is guided by a *fitness function* and operates via biologically inspired notions of *mutation*, *crossover*, and *selection*. In this work, we define fitness such that lower fitness is better to remain consistent with the energy function described in Section 3.2.1.

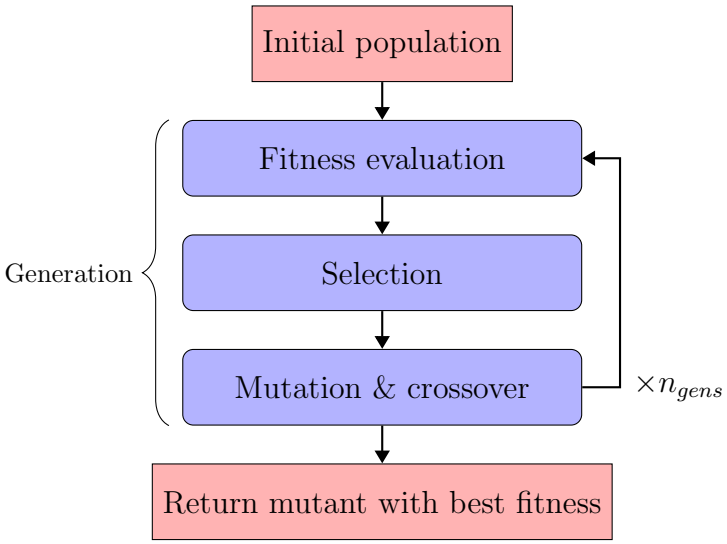


Figure 3.1: An overview of genetic algorithms.

A genetic algorithm operates in the following general fashion. First, a population of candidate solutions (i.e., *mutants*) is randomly generated. The population size is typically fixed to some $n_{mutants}$. This initial population then evolves in a series of *generations*. Each generation consists of two parts: (1) selection and (2) application of genetic operators (e.g., mutation and crossover). In the selection step, a score is assigned to each mutant via the fitness function and a new population is chosen given these scores. For example, a naive selection method would set the population to $n_{mutants}$ copies of the best-scoring mutant. Given this surviving population, the second step involves applying genetic operators to obtain a revised, diverse population. The two most common genetic operators are mutation and crossover. Analogous to mistakes being made during replication of a DNA sequence, a mutation operator involves randomly tweaking a single mutant. In the case where mutants are represented as bit strings, the canonical example of a mutation operator is a bit-flip. Alternatively, a crossover operator produces a new mutant given two or more existing mutants; this is analogous to reproduction and biological crossover. For example, given two mutants represented as strings, a new mutant can be generated by swapping the two segments defined by a random position. Evolution proceeds for n_{gens} generations and the algorithm returns the mutant with the highest fitness throughout search. This process is summarized in Figure 3.1.

The default in this work is to represent mutants as ZX-diagrams and to use rewrite rules as mutation operators. As is common in practical applications of genetic algorithms, we do not use any crossover operators though this is an attractive area for

future exploration given the graphical nature of ZX-diagrams. Lastly, we use *tournament selection* to determine the surviving mutants at each generation. In this selection method, k_{tourn} mutants are selected to compete in a "tournament" from which a winner is chosen based on fitness. This is repeated $n_{mutants}$ times with replacement to obtain a surviving population. We use a simple variant of tournament selection in which $k_{tourn} = 2$ and the winner is simply the mutant with better (i.e., lower) fitness.

3.2.3 Objective Functions

Both the energy function (used in SA) and the fitness function (used in GA) are instances of an *objective function*. An objective function measures the "goodness" of a particular solution to an optimization problem. For our purposes, we require an objective function that, given a ZX-diagram, provides a reasonable measure of the complexity of the circuit obtained via extraction. For consistency, all objective functions are defined in such a way that a low score is better (i.e., a low score indicates low complexity).

The most straightforward objective function involves extracting a circuit and measuring its complexity directly. A quantitative measure of circuit complexit is also useful for evaluating our optimization techniques. Given a circuit C in terms of single- and 2-qubit gates, we define its complexity $Comp(C)$ to be a weighted average of its single-qubit and 2-qubit gate counts:

$$Comp(C) = 10 * (\#\text{ 2-qubit gates}) + 1 * (\#\text{ single-qubit gates})$$

A scaling factor of 10 is chosen because 2-qubit gates are typically an order of magnitude more costly to implement than a single-qubit gate [2, 1]. Note that `basic_optimization` is always applied to the extracted circuit.

However, extracting a circuit from the ZX-diagram every time we want to measure its complexity is costly. For example, evolving a population of 50 mutants for 100 generations would require 5000 extractions. Instead, we could measure some property of the ZX-diagram that is a reasonable proxy for $Comp(C)$ (where C is the circuit obtained via extraction). The following are candidate properties to serve as such a proxy:

- The number of edges
- The density of the underlying graph

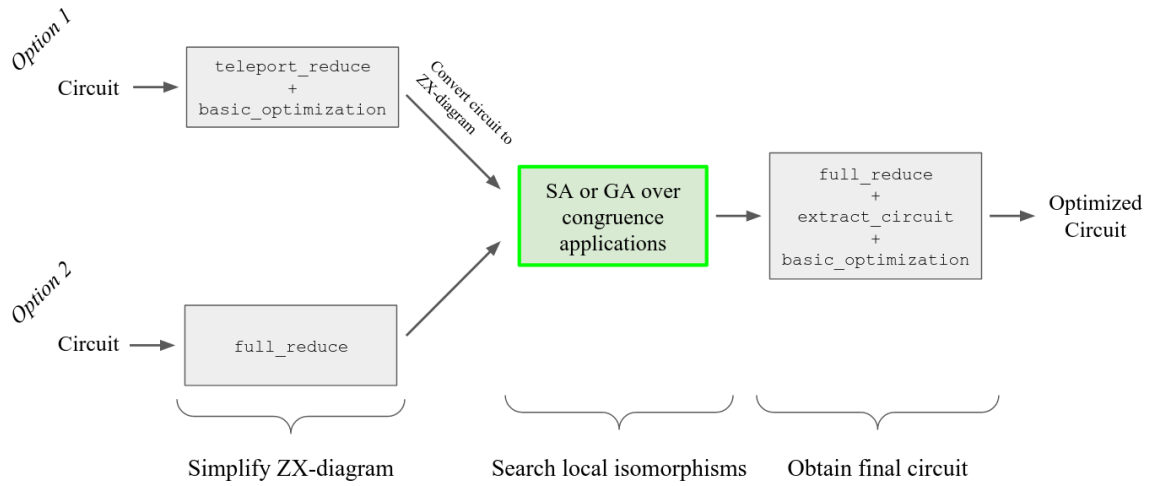


Figure 3.2: An overview of our primary optimization strategy. Given an input circuit, a simplified ZX-diagram is obtained via one of two off-the-shelf methods. Then, we search for an equivalent ZX-diagram with similar graph complexity but lower circuit complexity (highlighted in green). A circuit is extracted from the final ZX-diagram and standard circuit-level optimizations are performed.

- The centrality of the underlying graph

The ZX-diagram is assumed to be graph-like.

Measuring the ZX-diagram is clearly preferable but relies on the existence of a property that is sufficiently correlated to *Comp*. Note that none of the above objective functions, including *Comp*, are normalized and therefore scores can only be compared between isomorphic ZX-diagrams.

3.3 Circuit Optimization Strategy

Here we describe how we use these congruences and search procedures for quantum circuit optimization. At a high-level, our method involves first applying off-the-shelf methods to obtain a simplified ZX-diagram and subsequently searching the space of equivalent ZX-diagrams generated by Equations C1 and C2.

Our optimization strategy improves upon the two pipelines described in Section 2.4 by searching local variants of the simplified ZX-diagrams. Given an input circuit, we can obtain a simplified ZX-diagram to seed search in one of two ways. Firstly, we can apply `full_reduce`. Importantly, we do not apply the entire `full_reduce + extract_circuit + basic_optimization` pipeline and convert the simplified circuit to a ZX-diagram as the complexity cost incurred by

extraction will likely outweigh any simplifications achieved at the circuit-level. Alternatively, since `teleport_reduce` short-circuits circuit extraction, we can apply `teleport_reduce + basic_optimization` and convert the simplified circuit to a ZX-diagram. Given a simplified ZX-diagram, we can then apply SA or GA to search the space of ZX-diagrams generated by Equations C1 and C2 for an equivalent ZX-diagram whose extracted circuit is less complex. Note that search proceeds by first randomly selecting one of Equations C1 and C2 and then randomly selecting a subject (i.e., spider or pair of connected spiders) for the selected congruence. We denote the probability of selecting Equations C1 and C2 as p_{LC} and p_{pivot}, respectively (default: $p_{LC} = p_{pivot} = 0.5$). An overview of this strategy is depicted in Figure 3.2.

Since Equations C1 and C2 break the circuit structure of a graph, it is reasonable to apply `full_reduce` because a circuit has to be extracted regardless. Therefore, the default objective function scores a ZX-diagram by first applying `full_reduce`, extracting a circuit C and applying simple circuit-level optimizations with `basic_optimization`, and measuring its complexity $Comp(C)$. We can also apply `full_reduce` probabilistically via some free parameter p_{fr} throughout search (default: $p_{fr} = 0.1$). In SA, `full_reduce` is applied to the current state at each step with probability p_{fr} while in GA `full_reduce` is applied to each mutant in a given generation with probability p_{fr}.

There are several opportunities to refine this strategy. One example is applying Equations C1 and C2 with unequal probabilities. This could be beneficial if one congruence more effectively navigates the search space than the other. Another possible refinement is non-uniform sampling of spiders (resp. pairs of spiders) to which Equation C1 (resp. Equation C2) will be applied. The following are possible spider metrics to weight sampling for application of Equation C1:

- Degree
- Centrality measures. For example, the *load* centrality of a node measures the fraction of all shortest paths that pass through that node
- Degree of neighbors (sum or average)

Similarly, the following are possible metrics to weight sampling of pairs of nodes for Equation C2:

- Combined degree of the two spiders
- Degree of the union of all neighbors (sum or average)

- Edge centrality measures. For example, the *betweenness* centrality of an edge counts the number of the shortest paths that go through that edge

Given a measure μ on a set of possible candidates X for congruence application (i.e., spiders or pairs of connected spiders), we compute the probability of choosing some $x \in X$ as

$$P(x) = \frac{\mu(x)}{\sum_{x' \in X} \mu(x')}$$

Chapter 4

Results

Here we present the results of our optimization strategy. Before reporting on the performance of our strategy, we first conduct preliminary analyses to refine our method. All random circuits are generated using the `CNOT_HAD_PHASE_circuit` method in PyZX which constructs a circuit consisting of CNOT, HAD, and phase gates. Default parameters are used for the probability of each gate type.

4.1 Refinements

General opportunities for refinement are the following:

- Optimization function
- Probability of applying Equation C1 vs. Equation C2
- Sampling of subjects (i.e., spiders or pairs of connected spiders) for congruence application

For these analyses, we only use SA as it is more computationally efficient and we expect the behavior of these refinements to generalize to GA.

We also conduct procedure-specific refinements. We analyze how varying the number of iterations or mutants and generations affects optimization using SA and GA, respectively.

Optimization Function

First, we test if any property of a ZX-diagram could serve as a reliable proxy for the complexity of its associated circuit. To do so, we first generated 500 random circuits (10 qubits, 100 gates) and immediately converted them to ZX-diagrams. Given this

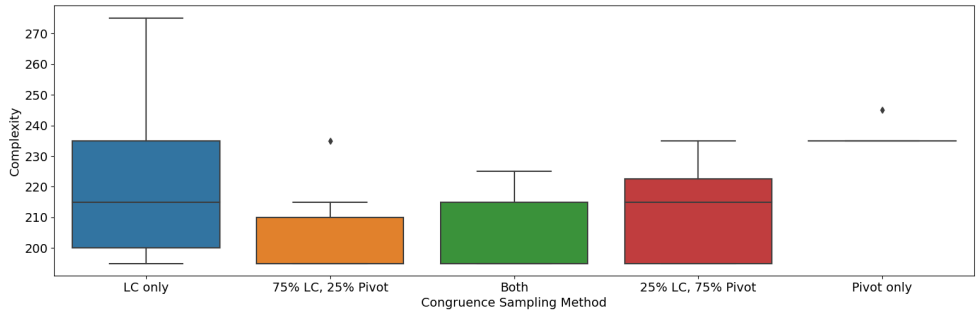


Figure 4.1: A representative comparison of congruence sampling methods for a 5 qubit, 50 gate circuit. All three sampling methods that include both LC and pivoting achieve the best simplification in the alotted number of steps. When using only local complementation (i.e., Equation C1), the same best-case complexity is achieved over the 10 trials but the average complexity is higher. Alternatively, using only pivoting does not achieve the same reduced circuit.

library of ZX-diagrams, we then measured the Pearson correlation coefficient between the complexity of the extracted circuit and each ZX-diagrammatic property discussed in Section 3.2.3. The following table summarizes these correlations:

	r	p-value
# Edges	0.097	0.029
Centrality	0.090	0.045
Density	-0.096	0.032

Pearson's correlation coefficient is denoted r and the 2-tailed p-value is provided. From these data, it is clear that no identified property of a ZX-diagram can serve as a reliable proxy for the complexity of its associated circuit. Therefore, the default scoring method discussed in Section 3.3 that relies on extraction is used for the remainder of our analyses.

Congruence Sampling

By default, we apply Equations C1 and C2 with equal probability ($p_{LC} = p_{pivot} = 0.5$). However, it may be the case that one congruence should be chosen more frequently than the other. To evaluate this, we generated 30 random circuits (5 qubits, 50 gates) and repeated search with a range of (p_{LC}, p_{pivot}) pairs. For a given circuit, search was performed 10 times for each pair and the complexities of the optimized circuits were plotted according to congruence sampling probabilities. In all cases, the sampling methods that include both LC and pivoting yielded the lowest average complexity across the 10 trials as well as the least complex circuit overall. There were

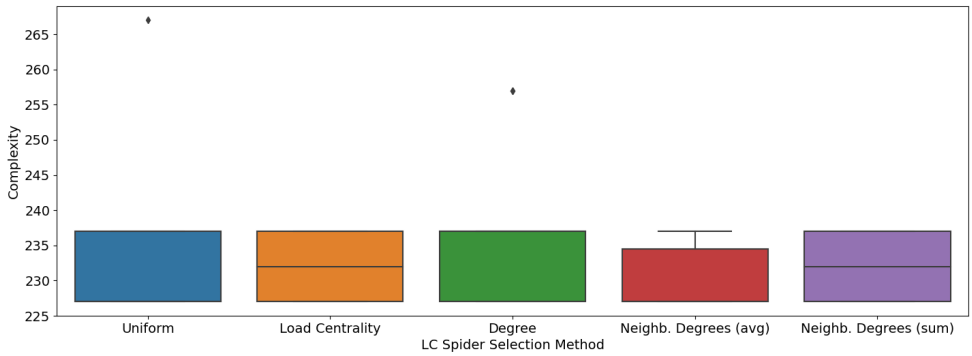


Figure 4.2: A representative example of optimization of a 5 qubit, 50 gate circuit with different metrics of sampling spiders for application of Equation C1. Optimization was performed 10 times for each sampling method using SA. No significant differences are observed in performance across the different weightings.

no significant differences between the three combined sampling methods (i.e., 50/50, 25/75, and 75/25) and the LC- ($p_{LC} = 1.0$, $p_{pivot} = 0.0$) or pivot-only ($p_{LC} = 0.0$, $p_{pivot} = 1.0$) sampling methods sometimes matched, but never outperformed these combined methods. Most notably, the least complex circuit identified using LC-only matched the minimum complexity 73.3% of the time while that identified using pivot-only was more complex than the best-case 86.7% of the time. A representative comparison for one random circuit is shown in Figure 4.1.

The observation that LC-only typically outperforms pivot-only is intuitive as one pivot is equivalent to three local complementations and therefore LC-only enables a more fine-grained search. While LC-only likely converges to the combined sampling methods in the limit, we retain pivoting in the action set as it appears to require a fewer number of iterations at no cost. In the remainder of our analyses, the default uniform sampling of Equations C1 and C2 (i.e., $p_{LC} = p_{pivot} = 0.5$) is used.

Congruence Subject Sampling

Similar to the probabilistic sampling of congruences to apply throughout search, we can experiment with methods of sampling subjects (i.e., spiders or pairs of connected spiders) for congruence application. By default, we sample from the set of all eligible subjects uniformly. However, we could alternatively weight this sampling in a way that improves search. Candidate metrics for Equations C1 and C2 are discussed in Section 3.3.

To test these alternative weightings, we employ a similar approach as for congruence sampling. First, we restrict the action space to that congruence for which we are

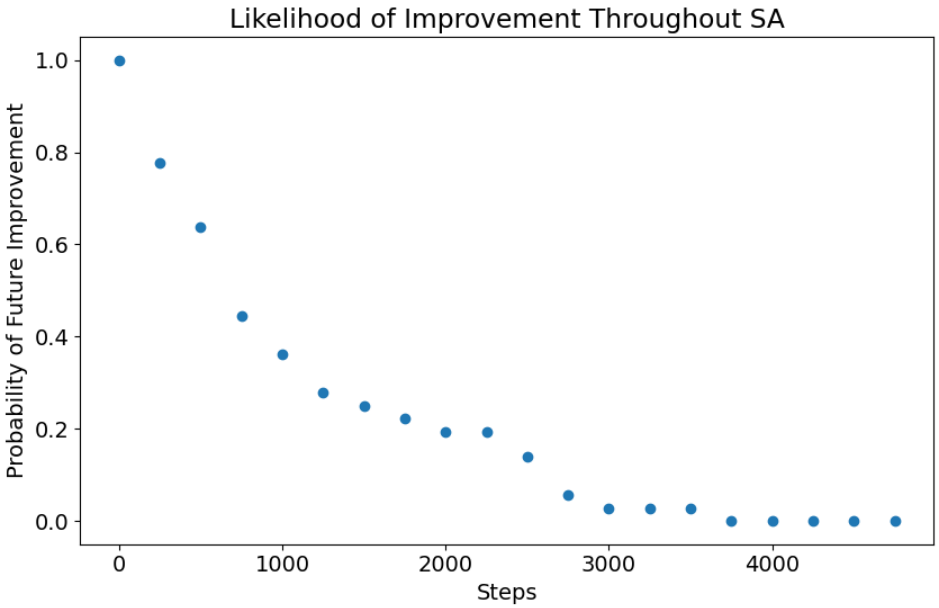


Figure 4.3: The likelihood of obtaining a simpler circuit at a given iteration in SA. For example, after 2000 steps, there is a 20% chance of obtaining an even simpler circuit as search progresses. These data were obtained using circuits with 4-10 qubits and 10-20 gates per qubit and do not necessarily generalize to larger circuits.

testing various weightings to isolate its effect. For example, if we are testing various weighting metrics to select a spider for application of Equation C1, we set $p_{LC} = 1.0$ and $p_{pivot} = 0.0$. We then proceed as before, evaluating the performance of SA on random circuits across 10 trials for each candidate weighting. We evaluate 20 random circuits (5 qubits, 50 gates) for each set of sampling procedures.

In both cases, no sampling method demonstrated reliable improvement over any other across the 20 trials. One representative example of this comparison for sampling spiders for Equation C1 is shown in Figure 4.2. Similar results were observed for sampling pairs of connected spiders for Equation C2. So, for the remainder of our analyses, we sample subjects for congruence application uniformly.

Number of SA Iterations

The maximum number of SA steps k_{max} should be high enough to permit the bulk of optimization while low enough to be computational feasible. We determine the optimal k_{max} by optimizing a set of random circuits and computing the probability that SA uncovers a complexity reduction after fixed intervals. First, we generated a set of 36 random circuits with 4-10 qubits (intervals of 2) and 10-20 gates per qubit (intervals of 5). For each circuit, we obtained a simplified ZX-diagram via

`teleport_reduce + basic_optimization`. We then optimized each simplified ZX-diagram using SA with $k_{max} = 5000$ and recorded the complexity of the best circuit every 250 steps. Lastly, we computed the probability that a less-complex circuit would be found as search progressed for each 250-step interval.

The results of this analysis are shown in Figure 4.3. We see that most improvement occurs in the beginning of search and that the chance of finding a simpler circuit after 2500 steps is less than 10%. For this reason, we fix $k_{max} = 2500$ for the remainder of our analyses unless otherwise noted.

4.2 Performance

Does well for small qubit numbers. (it is here that we can compare search-specific parameters).

Chapter 5

Discussion and Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Bibliography

[1] CJ Ballance, TP Harty, NM Linke, MA Sepiol, and DM Lucas. High-fidelity quantum logic gates using trapped-ion hyperfine qubits. *Physical review letters*, 117(6):060504, 2016.

[2] Earl T Campbell, Barbara M Terhal, and Christophe Vuillot. Roads towards fault-tolerant universal quantum computation. *Nature*, 549(7671):172–179, 2017.

[3] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John Van De Wetering. Graph-theoretic simplification of quantum circuits with the zx-calculus. *Quantum*, 4:279, 2020.

[4] Ross Duncan and Simon Perdrix. Graph states and the necessity of euler decomposition. In *Conference on Computability in Europe*, pages 167–177. Springer, 2009.

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

[6] Aleks Kissinger and John van de Wetering. Reducing t-count with the zx-calculus. *arXiv preprint arXiv:1903.10477*, 2019.

[7] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.