# Efficient Heuristics for Classical Simulation of Quantum Circuits Using ZX-Calculus

Wira Azmoon Ahmad

Linacre College

University of Oxford

Submitted in partial completion of the

*Master of Science in Advanced Computer Science*

Trinity 2024

# Abstract

Quantum circuits require validation and testing to ensure that they perform as expected. Classical simulation is a powerful tool for this purpose, but it is limited by the exponential growth of complexity with the number of qubits. Nevertheless, using the ZX-calculus, a graphical language for quantum mechanics, we can attempt to reduce how exponential this growth is. Recent developments have allowed focus to be placed on complexity being dependent on the number of $T$-gates that are in the circuit. Given $t$ $T$-gates, if the complexity of classically simulating the circuit is $O(2^{\alpha t})$, the aim is to reduce the value of $\alpha$ as much as possible, where $0 < \alpha < 1$. This thesis presents a heuristic approach toward exploiting the structure of the ZX-diagrams used to represent the quantum circuits, in order to reduce $\alpha$. We conceptualise a formal definition of the heuristic method, applying it to existing approaches, and also develop new heuristics that take advantage of the structure made available in the ZX-diagrams. These heuristics perform well in practice, boosting the number of $T$-gates that can be simulated classically by up to three times as many on certain classes of circuits.

# Contents

# List of Figures

# 1 Introduction

## Contents

## 1.1 Motivation

Quantum computing, a field rooted in the principles of quantum mechanics, represents a revolutionary approach to computation. Unlike classical computing, which relies on binary operations performed on bits, quantum computing utilizes quantum bits, or qubits, capable of representing and processing information in ways that classical bits cannot. This unique capability of qubits to exist in superpositions and entangle with each other allows quantum computers to perform certain types of calculations exponentially faster than their classical counterparts. If a classical computer were limited by memory and runtime, a quantum computer could potentially bypass these scaling issues - dubbed the "quantum advantage".

If scalable quantum computers were to become a reality, they could solve some of the most challenging problems in computer science much more efficiently than classical computers. Classical computations play an essential role in assisting and accelerating the development of quantum computers, since classical simulation is a fundamental aspect of understanding and designing quantum hardware, often serving as the only means to validate these quantum systems [1]. Additionally, quantum algorithms can be implemented on classical simulators for testing and verification purposes while full-fledged quantum computers remain beyond reach (for now). These simulators emulate the behavior of quantum computers, allowing researchers to simulate processes as if running on actual quantum devices.

This thesis aims to present the speedup of classical simulation of quantum circuits in an almost entirely graphical approach. From the introduction of the qubits, to the representation of quantum circuits and linear maps, the underlying linear algebra that governs the operations of quantum computing is represented in a graphical manner. Using ZX-Calculus, a graphical calculus detailed in Section 1.3, quantum circuits can be represented as diagrams, allowing for different discoveries to be more easily made when analysing the structure of said diagrams.

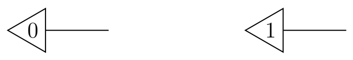
The thesis is organised as follows. First, from this chapter, we introduce the basic notation and concepts of quantum computing, with a focus on the ZX-Calculus. Next, Chapter 2 outlines the methods used for classical simulation via stabiliser decomposition using ZX-Calculus as covered in [2–4]. Chapter 3 formalises the definition of using heuristics to improve classical simulation, and connects these heuristics to the existing methods in the literature [4–6]. The classes of circuits benchmarked are introduced in Chapter 4, as well as the numerical results of using heuristics covered in Chapter 3. It also discusses the results and some of the insights gained from the experiments. Finally, Chapter 5 concludes the thesis with a summary of the results and future work.

The bulk of novel work in this thesis is in Chapter 3 and Chapter 4, where the heuristics are introduced and the results are presented.

## 1.2 Basic Notation

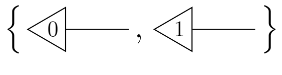
*This section, and the one after it, introduce the basic notation and concepts of quantum computing, along with the ZX-Calculus, and is a subset of the concepts covered by Coecke and Kissinger in Picturing Quantum Processes [7].*

In classical computing, a bit is the fundamental unit of information, and is denoted by a boolean value $b \in \{0, 1\}$. In quantum computing, the fundamental unit of information is the qubit, which we denote as $|b\rangle$, where $b \in \{0, 1\}$. Diagramatically, they are represented as follows:



We consider these the computational or Z-basis states.

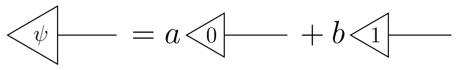
**Definition 1.2.1** (Z-basis). The *Z-basis* or computational basis is defined as



▲

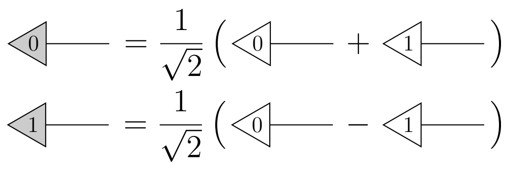
So far, the expressive power is still similar to classical computing. However, the power of quantum computing comes from the ability to exist in superpositions of these states. Concretely, the qubit can be represented in a linear combination of the basis states.

**Definition 1.2.2** (Pure state)**.** A *pure state* $|\psi\rangle$ is a state



where $a, b \in \mathbb{C}$ and $|a|^2 + |b|^2 = 1$. ▲

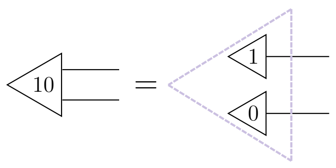
Then, the '+' or X-basis states can be defined.



(1.1)

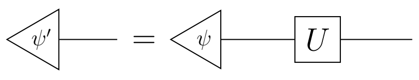
Along with the Z-basis states, these states form the building blocks of understanding quantum computing and the use of ZX-calculus for diagrammatic reasoning.

Intuitively, using the diagrammatic representation, we can consider combining diagrams in parallel. In the literature, this is the same as taking a *tensor product* $\otimes$. For example, we can have a 2-qubit state.



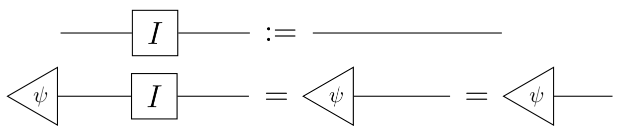
(1.2)

States can be transformed by quantum gates simply by *composing* them to obtain a new state. Given that composing the quantum gate $U$ to a state $|\psi\rangle$ gives a state $|\psi'\rangle$, we can just connect the wires.



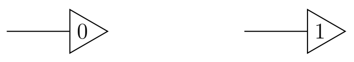
(1.3)

Intuitively, this means that the wire by itself should just be the identity gate, since composing it with any state should not change that state.



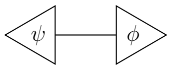
(1.4)

The dual of states are the effects. Here, we have the Z-basis effects.

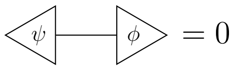


For a state $|\psi\rangle$ and an effect $\langle\phi|$, we can obtain a scalar inner product. This is essentially the *bra-ket* notation in diagrammatic form by just connecting the wires, which comes with a few other definitions.

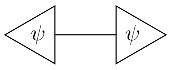
**Definition 1.2.3** (Inner product [7]). The *inner product* $\langle\phi|\psi\rangle$ of two states $|\psi\rangle$ and $|\phi\rangle$ is defined as



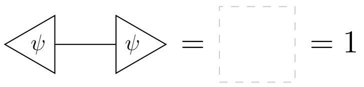
They are *orthogonal* if



The *squared-norm* of a state $|\psi\rangle$ is



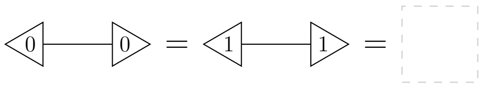
$|\psi\rangle$ is *normalised* if

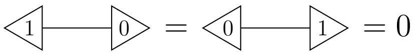


where the empty diagram represents the scalar 1. ▲

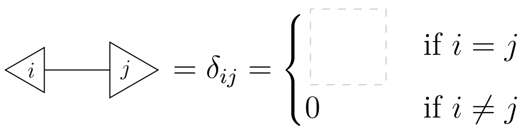
We can then define the inner product of the Z-basis states.

**Definition 1.2.4.** The inner product of the Z-basis states is





Succinctly, for $i, j \in \{0, 1\}$,

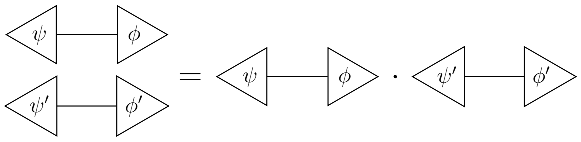


where $\delta_{ij}$ is the Kronecker delta. ▲

In fact, the succinct definition can be used to determine if a basis is an *orthonormal basis* (ONB). The X-basis is also an ONB.

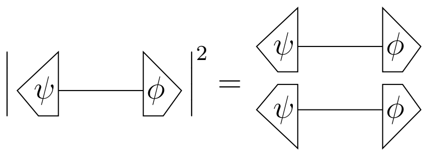
We are almost at the stage where we can obtain probabilities from the diagrams. The scalar inner product is not really a probability, that is, a number between 0 and 1, but instead, a number $z \in \mathbb{C}$.

First, we note that putting multiple scalars in a single diagram is just the same as multiplying them together.



(1.5)

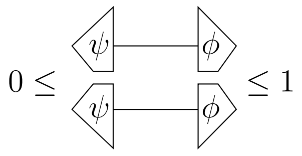
We also know that for a complex number and its conjugate, $z, \bar{z} \in \mathbb{C}, |z|^2 = z\bar{z}$ Translating that to the diagrammatic representation, we can obtain the following.



(1.6)

Note that assymetry is introduced to denote the conjugation. Remember that a complex number $z = a + bi \in \mathbb{C}$ has a conjugate $\bar{z} = a - bi \in \mathbb{C}$, so this corresponds to a vertical reflection in the diagram above.

Using the *Born rule*, if we make sure that $|\psi\rangle$ and $|\phi\rangle$ are normalised, we can obtain the following.



(1.7)

This can be interpreted as our desired probaility - the probability of measuring $|\psi\rangle$ in the state $|\phi\rangle$.

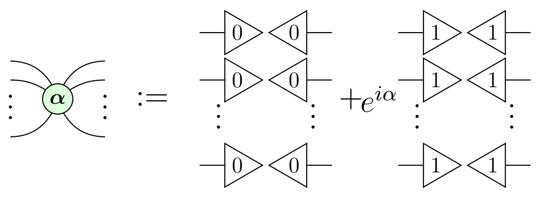
We now have all we need to introduce the ZX-calculus notation.

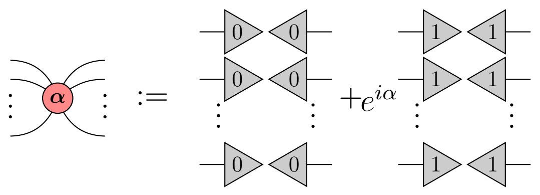
## 1.3 ZX-Calculus

So far, we have introduced the representation of quantum states, effects, and inner products in a diagrammatic form. However, there still isn't much we can do without a way to manipulate them. In the previous section, we mentioned that we can apply quantum gates to states to obtain new states. Here, we will introduce the ZX-Calculus and show the highly intuitive representation of some quantum gates such as the Hadamard and CNOT gates, based on the building blocks from the previous section. The ZX-Calculus is a graphical calculus for quantum computations, developed by Coecke and Duncan in 2008 [8]. All operations are expressed as *spiders*, connected by *edges* (or *wires*).

### 1.3.1 Spiders

**Definition 1.3.1** (Spiders)**.**

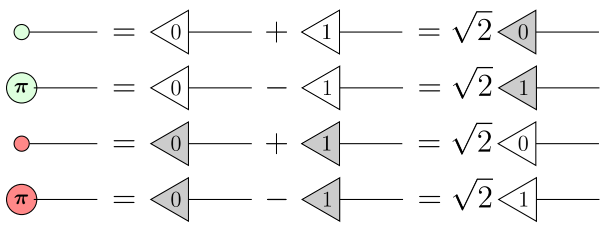




▲

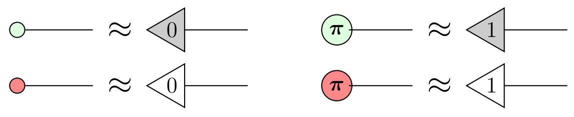
Here, we have green and red spiders, also known as Z and X spiders, made of the Z and X basis states respectively. Wires coming in from the left are *inputs*, and wires going out to the right are *outputs*. The $\alpha$ inside a spider node is also called the *phase* of the spider. For example, with $\alpha = 0, e^{i\alpha} =$ . It is convenient to denote spiders with $\alpha = 0$ as having no angle written inside the spider.

Immediately, we can see that the Z and X basis states can be formed from the definition of the spiders, up to the multiplicative constant $\sqrt{2}$. Note that $e^{i\pi} = -1$.



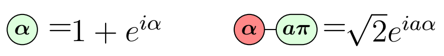
(1.8)

It turns out that in reasoning with spiders, we can ignore the multiplicative constants, since much of the manipulation of the diagrams is based on the structure of the diagrams themselves. The scalars are easy to determine using the definitions we have seen so far. By using $\approx$ to denote equality up to a non-zero factor, we have



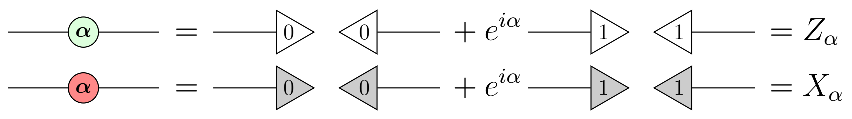
(1.9)

Scalars can also be represented using just spiders.



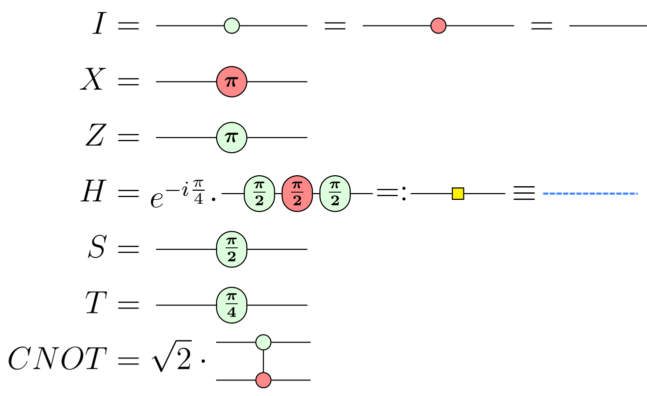
(1.10)

We can also see the single-qubit gates in the ZX-Calculus.



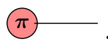
(1.11)

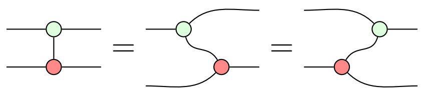
We can then define many of the common quantum gates in this way.



(1.12)

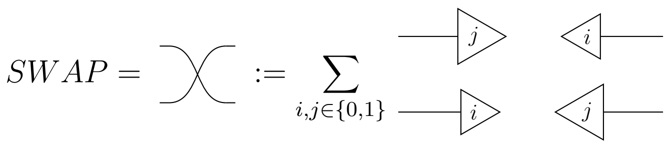
Here, we define a Pauli gate as any gate $P \in \{I, X, Y, Z\}$. There are 2 gates of special note here. The Hadamard gate $H$, denoted by the small yellow box, or as a blue dashed line, is a single-qubit gate that interchanges $Z$ and $X$ bases. We will see later in the ZX-rules that this is vital for the colour change (***cc***) rule, as well as why it is useful to denote it as the blue dashed line.

The CNOT gate is a two-qubit gate that acts on the target qubit if the control qubit is in the state $|1\rangle \approx$ . The vertical line is used simply because it does not matter which direction the line is going, where the following equality of the second and third forms can be verified.



(1.13)

Last but not least, we also introduce the SWAP gate, which as the name suggests, swaps two qubits.



(1.14)

We can now introduce the ZX-rules in the next section.

### 1.3.2 ZX-Rules

The ZX-Calculus is based on a set of rules that allow us to manipulate the diagrams. For $\alpha, \beta \in [0, 2\pi)$ and $a \in \{0, 1\}$, the rules are as follows in Figure 1.1.

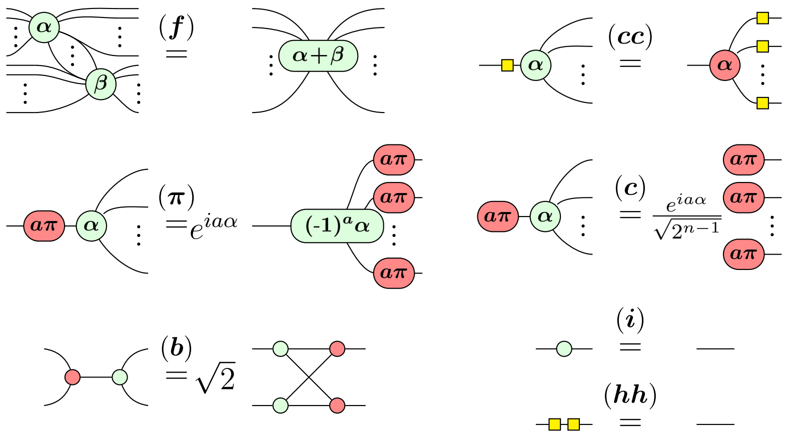
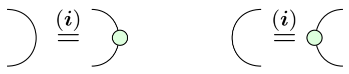


**Figure 1.1:** $ZX$-rules presented in [9]

The names of the rules in Figure 1.1 are spider $(\boldsymbol{f})$usion, colour change $(\boldsymbol{cc})$ rule, $(\boldsymbol{\pi})$-commutation, $(\boldsymbol{c})$opy rule, $(\boldsymbol{b})$ialgebra, $(\boldsymbol{i})$dentity rule, and $(\boldsymbol{hh})$adamard cancel rule. For $(\boldsymbol{c})$, $n$ refers to the number of outputs. By duality, the rules also hold if the colours are swapped.

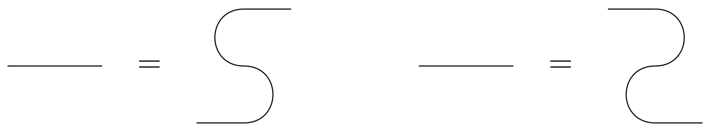
The power of ZX-diagrams comes from the fact that two diagrams are equivalent as long as the connectivity of the diagram is the same - that is, as long as order of inputs and outputs of the whole diagram is preserved. This does not depend on the position of the spiders or the direction of wires between them. This neat feature of ZX-diagrams is called "Only Connectivity Matters" (OCM).

We can then obtain caps and cups using 2-legged spiders together with $(\boldsymbol{i})$.



(1.15)

Caps and cups obey the *yanking equations.*



(1.16)

Before we move on to reasoning about the diagrams in a graph-theoretic manner, we first introduce the concept of a *Clifford+T diagram.*

**Definition 1.3.2** (Clifford diagram)**.** A *Clifford diagram* is a ZX-diagram where the phases are restricted to integer multiples of $\frac{\pi}{2}$. ▲

**Definition 1.3.3** (Clifford+T diagram)**.** A *Clifford+T diagram* is a ZX-diagram where the phases are restricted to integer multiples of $\frac{\pi}{4}$. ▲

It is clear from the definitions that a Clifford diagram is a subset of a Clifford+T diagram. The reason we introduce these concepts is that in trying to obtain our main goal of the thesis, it is sufficient to reason about Clifford+T diagrams, as a result of the following theorem.

**Theorem 1.3.4** (Approximate Universality [10])**.** Clifford+T diagrams are *approximately universal* for quantum computing - they can approximate any quantum circuit.

The proof of the above theorem involves the *Solovay-Kitaev theorem*, combining with the fact that any single-qubit phase gate can be approximated by a sequence of Hadamard and $T$ gates, with details found in [10].

## 1.4 Graph Reduction

*This section connects the ZX-diagrams to graphs, in a way which will continue to be used for the rest of the thesis. The content in this section will be covered by Kissinger and van de Wetering in the soon-to-be-published Picturing Quantum Software [11]. Most of the concepts were also covered in [12, 13]*

Because of OCM, ZX-diagrams up until now have very similar structure to graphs, where the spiders are the vertices and the wires are the edges. The only difference may be that the spiders are coloured, and that the Hadamard boxes are included. In this section, we solidify the connection between ZX-diagrams and graphs, and introduce an algorithm that will simplify ZX-diagrams, defining what it means for a graph to be simplified. Ultimately, simplifying ZX-diagrams should allow us to reduce the number of spiders in the diagram, which directly translates to a reduction in the number of gates in the quantum circuit. In turn, as we will see in the next chapter, this will lead to a speedup in the simulation of quantum circuits.

First, we define what it means for ZX-diagram to be *graph-like.*

**Definition 1.4.1** (Graph-like [11])**.** A ZX-diagram is *graph-like* when

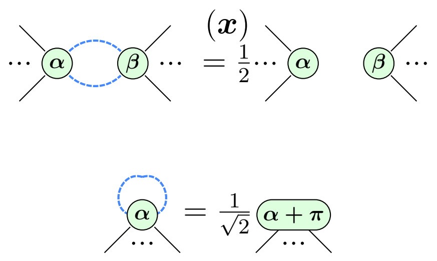
- Every spider is a Z-spider.
- Spiders are only connected via Hadamard edges.
- There is at most only one edge between each pair of spiders.
- There are no self-loops.
- Every input and output wire is connected to a Z-spider.

▲

Just from this definition, Hadamard edges are in fact the only edges in the graph-like ZX-diagram, leading to the convenience of having them as the blue dashed lines introduced earlier in (1.12). The following proposition can then be proven, using rewrite rules introduced previously in ZX-Rules.

**Proposition 1.4.2** ([11])**.** Every ZX-diagram can be converted to a graph-like ZX-diagram in polynomial time.

The proof [11, 12] makes use of the fact that Hadamard edges in parallel cancel, and that we can always remove a Hadamard self-loop. Parallel edges cancelling is an application of the *Hopf rule* ($\boldsymbol{x}$), derivable from the ZX-rules [1].



(1.17)

We can simplify the graph-like ZX-diagram further. Now, the following two rewrite rules, derivable from the original ZX-rules, are introduced.

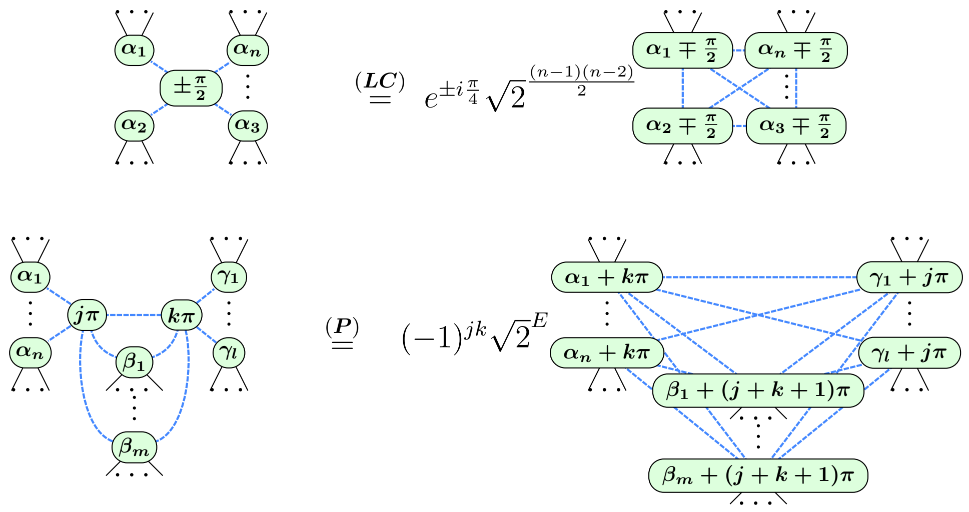


**Figure 1.2:** Derived $ZX$-rules [12], where $\alpha, \beta, \gamma \in [0, 2\pi)$ and $j, k \in \{0, 1\}$.

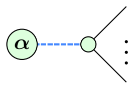
The rules are *local complementation* ($\boldsymbol{LC}$) and *pivoting* ($\boldsymbol{P}$) respectively. Here, $E = (n-1)m + (l-1)m + (n-1)(l-1)$.

[1]In fact, $(\boldsymbol{b}) \implies (\boldsymbol{x})$

Repeatedly simplifying and reducing diagrams in this manner will end up giving us patterns in the diagrams. The following definitions help us identify these patterns. First, we define interior and boundary spiders, and then phase gadgets.

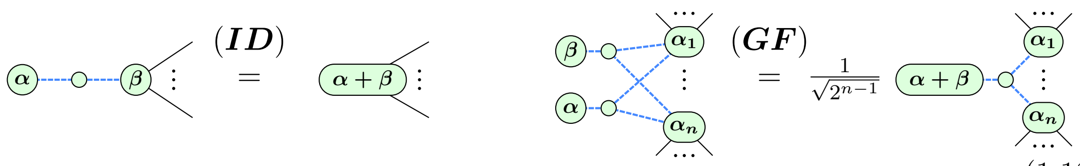
**Definition 1.4.3** ([13])**.** A *boundary spider* is a spider connected to an input or output. All other spiders are *internal spiders*. ▲

**Definition 1.4.4** (Phase gadget [13])**.** A *phase gadget* is an arity-1 spider with angle $\alpha$, connected via a Hadamard edge to a spider with no angle.



▲

The usefulness of phase gadgets is that they can be used to simplify the diagram further. Here, two more derived rules involve phase gadgets.



(1.18)

These help us define the reduced gadget form.

**Definition 1.4.5** (Reduced gadget form [13])**.** A graph-like ZX-diagram is in *reduced gadget form* when

- Every internal spider is a non-Clifford spider or part of a non-Clifford phase-gadget.
- Every phase-gadget has more than one target.
- No two phase-gadgets have the same set of targets.

▲

It turns out that there is an algorithm detailed below that is bounded in $O(n^3)$ where $n$ is the number of spiders, to convert any graph-like ZX-diagram to reduced gadget form [2]. Thus, much of the work in this thesis will focus on the reduced gadget form, as it tends to bring us closer to the goal of reducing overall $T$-count. However, there are exceptions to this, as we will see in the next chapter.

**Algorithm 1.4.6** (ZX-simplify [2])**.** Starting with a graph-like ZX-diagram, do the following:

1. Apply (***LC***) until all proper Clifford spiders are removed.
2. Apply (***P***) (and variations) until all interior $k\pi$-spiders are removed or transformed into phase-gadgets.
3. Remove all Clifford phase-gadgets using (***LC***) and (***P***).
4. Apply (***ID***) and (***GF***) wherever possible. If any matches are found, go back to step 1. Otherwise, the diagram is in reduced gadget form.

**Example 1.4.7.** The following diagram from [2] is in reduced gadget form. ▲

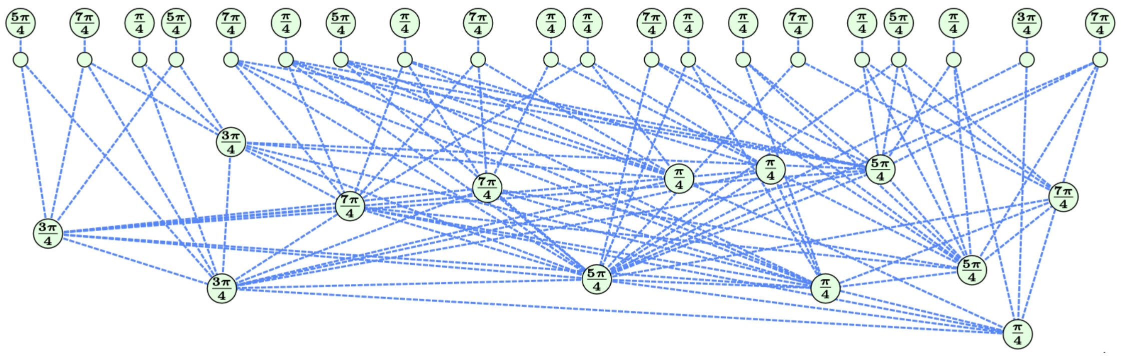


**Figure 1.3:** A ZX-diagram in reduced gadget form [2]

# 2 Background

## Contents

*This chapter introduced the concepts required for classical simulation, using stabiliser decompositions, magic states, cat states, and graph cuts. There is no novel work in this chapter, with all results being adapted from existing literature [2–4, 11, 14, 15]. We show a proof of Prop. 2.4.2 in ZX-Calculus, adapting from [14], and give a detailed analysis of the power of cuts in Graph Cuts.*

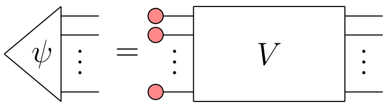
## 2.1 Introduction

As mentioned in the previous chapter, classical simulation of quantum circuits provides an essential tool for verification and validation, despite its inherent limitations. This chapter delves into the methods and advancements in classical simulation via stabiliser decomposition, discussing the role of magic states, cat states, and graph cuts, in extending the capabilities of these simulations. Classical simulation methods that store a complete description of an $n$-qubit quantum state as a complex vector of size $2^n$ are effective only for a small number of qubits, typically around 30. For example, Shor's factoring algorithm has been simulated with 31 qubits and approximately half a million gates [16]. However, for certain classes of quantum circuits, such as those that can be represented in a Clifford diagram, classical simulation can be significantly more efficient [17]. This is known as the *Gottesman-Knill theorem.*

**Theorem 2.1.1** (Gottesman-Knill Theorem)**.** A Clifford computation can be efficiently classically simulated.

In other words, a computation involving only *Clifford diagrams* can be efficiently classically simulated. This is extremely useful when considering that ZX-Calculus simplifies the process of identifying and manipulating *stabiliser states*.

**Definition 2.1.2** (Stabiliser state [15, 17]). An $n$-qubit state $|\psi\rangle$ is a *stabiliser state* if it has the form

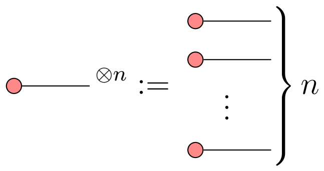


where $V$ is a Clifford diagram. ▲

It is clear that if a computation only involves stabiliser states, then it can be efficiently classically simulated. Stabiliser circuits are particularly notable for their applications in quantum error correction [17]. In the next few sections, we will see more clearly how these stabiliser states are used in classical simulation. Particularly, we also need to consider Clifford+T diagrams, and how we can still (inefficiently) simulate them using stabiliser states.

## 2.2 Strong Simulation

When simulating quantum circuits, we assume that the input state to our $n$-qubit circuits is of the form

 (2.1)

Given that we have a $n$-qubit circuit with the input state from (2.1), we would then want to obtain the probability of measuring a certain output state. For instance, we can plug in the outputs $\langle\vec{x}|$ for $\vec{x} = x_1, x_2, ..., x_n$ as spiders as in Figure 2.1.

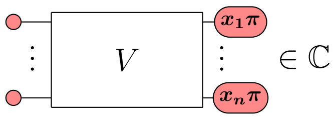
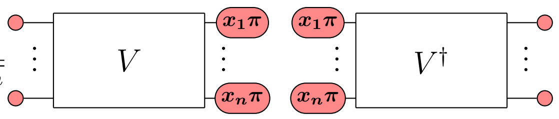


**Figure 2.1:** Scalar involving Clifford+T diagram $V$

Then using the Born rule, we can obtain the probability of measuring the output state [11].

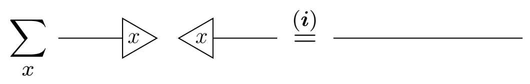
$$\Pr(x_1, x_2, ..., x_n) = \frac{1}{\sqrt{2^{4n}}} \tag{2.2}$$


$V^\dagger$ is used to denote the adjoint of $V$, which simply means the inputs and outputs are swapped, and the spider phases are negated. The scalar factors of $1/\sqrt{2}$ comes from those seen in (1.8). If $V$ is a Clifford diagram, then following from the Gottesman-Knill Theorem, the scalars can be computed efficiently.

Note that being able to obtain any *marginal* probability is called *strong simulation*. These can also be obtained similarly. Recall that using the law of total probability, we have for $k < n$,

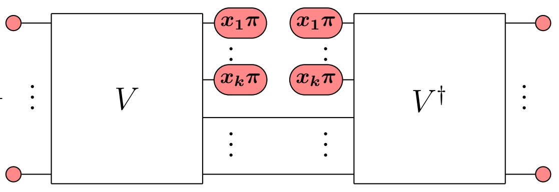
$$\Pr(x_1, x_2, ..., x_k) = \sum_{x_{k+1}, x_{k+2}, ..., x_n} \Pr(x_1, x_2, ..., x_k, x_{k+1}, ..., x_n) \tag{2.3}$$

Then, applying the (2.3) to (2.2), and using the $(\boldsymbol{i})$dentity rule, where



(2.4)

we can obtain the marginal probabilities.

$$\Pr(x_1, x_2, ..., x_k) = \frac{1}{2^{n+k}} \tag{2.5}$$


This technique to obtain the probabilities is sometimes called "doubling". A natural question that arises is whether the same can be done if $V$ is a Clifford+T diagram instead. The following sections will delve into the methods used to simulate such circuits.

### 2.2.1 Weak Simulation

Strong simulation requires the calculation of any marginal probability (2.5). Weak simulation, on the other hand, just requires a probabilistic algorithm that can produce the bitstrings $\vec{y} \in \{0,1\}^n$ with the distribution from (2.5) [1]. Strong simulation implies weak simulation - if we can calculate the actual marginal probabilities, we can sample from the distribution [11].

[1]The distribution can be 'suitably close'. See [11] Section 5.4.3

$$\begin{aligned}\Pr(x_1, x_2, ..., x_n) &= \prod_{i=1}^{n} \Pr(x_i | x_1, x_2, ..., x_{i-1}) \\ &= \prod_{i=1}^{n} \frac{\Pr(x_1, ..., x_i)}{\Pr(x_1, ..., x_{i-1})}\end{aligned} \tag{2.6}$$

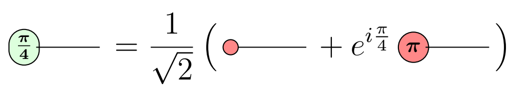
By iterating from $i = 1..n$, we can obtain the bitstring $\vec{y}$ that corresponds to an output state. In [4], this is called *qubit-per-qubit* simulation.

In [18], they found that weak simulation could be done without computing marginals, avoiding the doubling approach which can increase the $T$-count of the circuit. This is not covered in this thesis, but interested readers can refer to the *gate-per-gate* approach shown with ZX-Calculus [4].

What these show is that increasing the efficiency of strong simulation thus can imply increasing the efficiency of weak simulation. For results in this thesis, we will then focus on strong simulation.

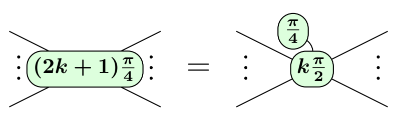
## 2.3 Stabiliser Decomposition

**Definition 2.3.1** (Magic state [15]). The *magic state* is



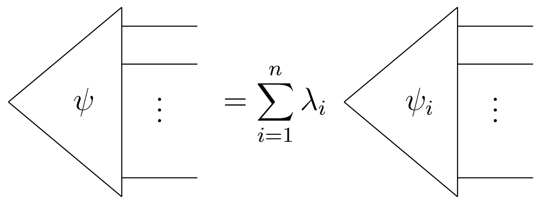
▲

The naïve approach to make use of this magic state decomposition is to then apply it to every single $T$-like spider in the Clifford+T diagram, where we define a $T$-like spider as a spider with a phase of $(2k+1)\frac{\pi}{4}$ for some $k \in \mathbb{Z}$. By un(***f***)using the magic state from these spiders as below, we can apply the Magic state [15] decomposition to obtain a sum of stabiliser states.

 (2.7)

This sum can be called the *stabiliser decomposition* of the circuit.

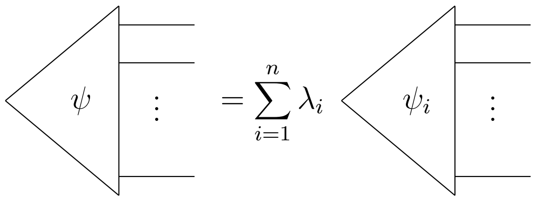
**Definition 2.3.2** (Stabiliser Decomposition [15]). The *stabiliser decomposition* of a state $|\psi\rangle$ is a sum of stabiliser states



where $\lambda_i \in \mathbb{C}$ and $|\psi_i\rangle$ are stabiliser states. ▲

More generally, a decomposition need not consist only of stabiliser states.

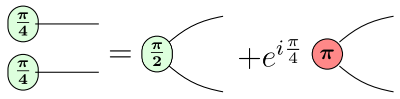
**Definition 2.3.3** (Decomposition)**.** The *decomposition* of a state $|\psi\rangle$ is a sum of states



where $\lambda_i \in \mathbb{C}$ and $|\psi_i\rangle$ are states. For any decomposition $d$, define its length to be $|d| = n$. ▲

Using the Magic state [15] decomposition, we can see that the value of $n = 2^t$ if we apply it to the $t$ $T$-like spiders in the circuit. We can call $t$ the *$T$-count* of the circuit.

However, there is an even better decomposition



(2.8)

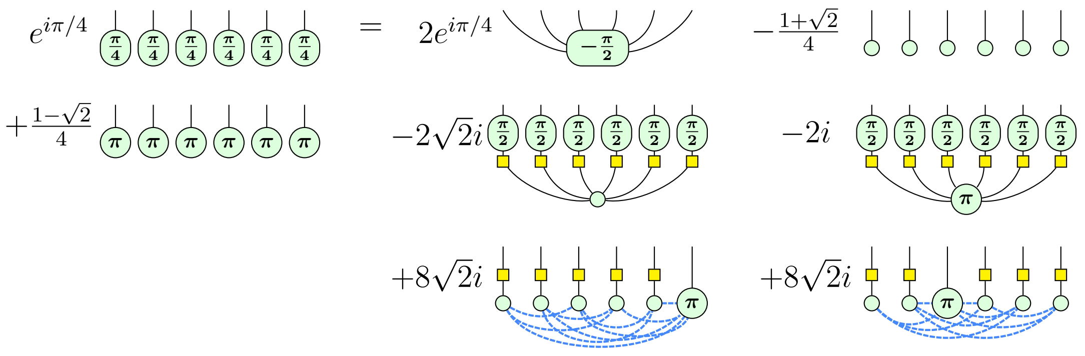
This allows us to decompose the circuit into a sum of stabiliser states with $n = 2^{\frac{t}{2}}$. Different methods of decomposing the circuit can lead to different values of $n$. Since $n$ is believed to be exponential in $t$ (or else we have $P = NP$) [19], we can take it that $n = O(2^{\alpha t})$ for some $0 < \alpha < 1$. We can then compare the efficiency of different methods by using this metric.

**Definition 2.3.4** ((In)efficiency [2, 3, 15, 20])**.** The *(in)efficiency* of a decomposition $D$ that reduces the $T$-count by $r$ using $p$ states is

$$\alpha(D) := \frac{\log_2(p)}{r} \tag{2.9}$$

where a lower value of $\alpha$ [2] indicates a more efficient decomposition. ▲

For example, the Bravyi-Smith-Smolin (BSS) decomposition [15] makes use of the following decomposition which has $\alpha \approx 0.468$.


$e^{i\pi/4}$
$\frac{\pi}{4}$ $\frac{\pi}{4}$ $\frac{\pi}{4}$ $\frac{\pi}{4}$ $\frac{\pi}{4}$ $\frac{\pi}{4}$
$=$
$2e^{i\pi/4}$
$-\frac{\pi}{2}$
$-\frac{1+\sqrt{2}}{4}$
$+\frac{1-\sqrt{2}}{4}$
$\pi$ $\pi$ $\pi$ $\pi$ $\pi$ $\pi$
$-2\sqrt{2}i$
$\frac{\pi}{2}$ $\frac{\pi}{2}$ $\frac{\pi}{2}$ $\frac{\pi}{2}$ $\frac{\pi}{2}$ $\frac{\pi}{2}$
$-2i$
$\frac{\pi}{2}$ $\frac{\pi}{2}$ $\frac{\pi}{2}$ $\frac{\pi}{2}$ $\frac{\pi}{2}$ $\frac{\pi}{2}$
$\pi$
$+8\sqrt{2}i$
$\pi$
$+8\sqrt{2}i$
$\pi$

(2.10)

[2]We omit $D$ when the context is clear

Using this metric, we can even compare the efficiency of different algorithms used to decompose the circuit. It also allows us to obtain the efficiency of different decompositions which do not use magic states at all. As we will see in the next few chapters, this ties in well when we consider the higher efficiency of *cat state* decomposition.

A closely related metric we can use is the *stabiliser rank*.

**Definition 2.3.5.** Let $D$ be the set of all stabiliser decompositions of a state $|\psi\rangle$. The *stabiliser rank* $\chi(|\psi\rangle)$ of a state $|\psi\rangle$ is

$$\chi(|\psi\rangle) := \min\{n \mid \exists(d_1, d_2, ..., d_n) \in D \text{ s.t. } |\psi\rangle = \sum_{i=1}^{n} d_i\}$$

▲

We can see that the relation between the stabiliser rank and its efficiency is

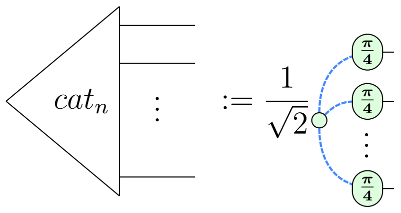
$$\chi(|\psi\rangle) = 2^{\alpha_\chi(|\psi\rangle)t} \tag{2.11}$$

with the $T$-count $t$ of the circuit.

The goal for any stabiliser decomposition is to obtain an efficiency as close to $\alpha_\chi$ as possible.

## 2.4 Cat States

**Definition 2.4.1** (Cat state [14]). A *cat state* is defined as

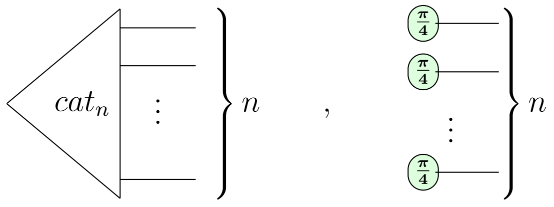


(2.12)

▲

Cat states are used because their decompositions have low value of $\alpha$, as well as the following proposition, which bounds the $\alpha$ of the corresponding magic state decomposition.

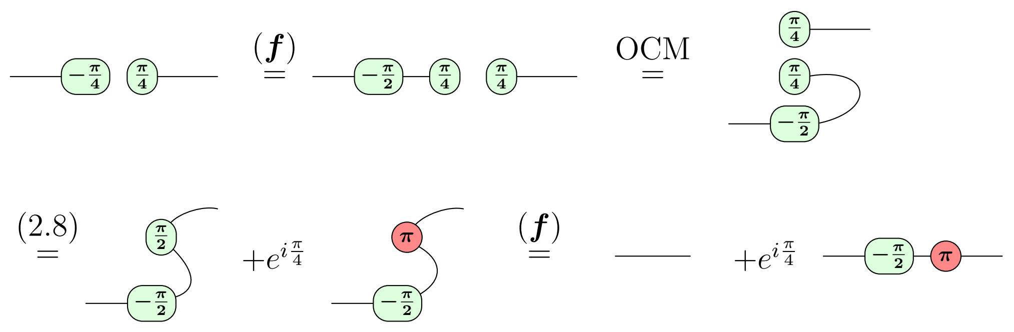
**Proposition 2.4.2.** Suppose $\alpha_1, \alpha_2$ are the efficiency metrics of the decompositions of



respectively. Then

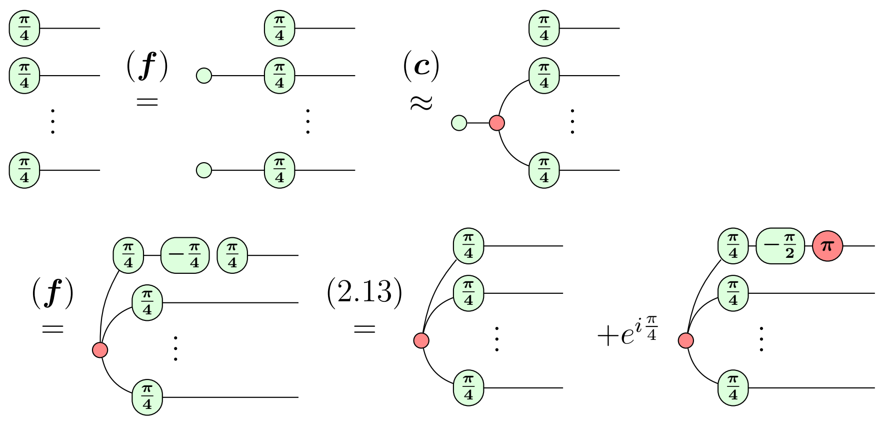
$$\alpha_1 \leq r \implies \alpha_2 \leq r + \frac{1}{n}$$

*Proof.* The proof is adapted from [14]. Suppose $\alpha_1 \leq r$. First, we show the following equality.



(2.13)

Then we can show the following.



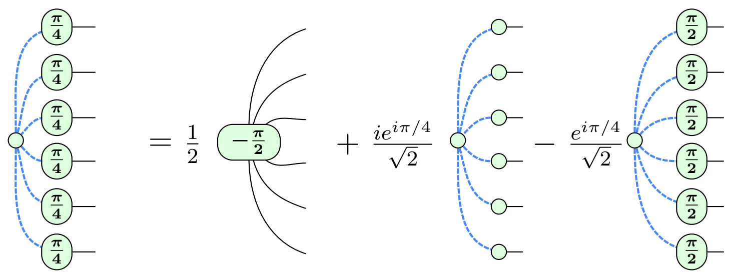
(2.14)

There are 2 copies of $|\mathrm{cat}_n\rangle$ which would each give at most $2^{n\alpha_1}$ stabiliser states, for a total of at most $2^{n\alpha_1+1}$ stabiliser states. Thus, we have

$$\alpha_2 \leq \frac{n\alpha_1 + 1}{n} \leq r + \frac{1}{n}$$

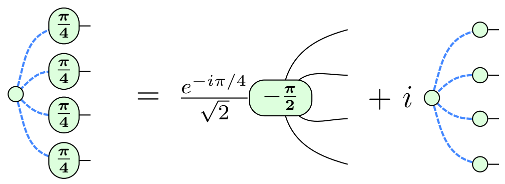
□

There are various cat state decompositions that have been found. For example, the decomposition for $|\mathrm{cat}_6\rangle$ is


$= \frac{1}{2}$
$+ \frac{ie^{i\pi/4}}{\sqrt{2}}$
$- \frac{e^{i\pi/4}}{\sqrt{2}}$

(2.15)

It has an efficiency of $\alpha \approx 0.264$. Even better, for $|\mathrm{cat}_4\rangle$, we have the decomposition



(2.16)

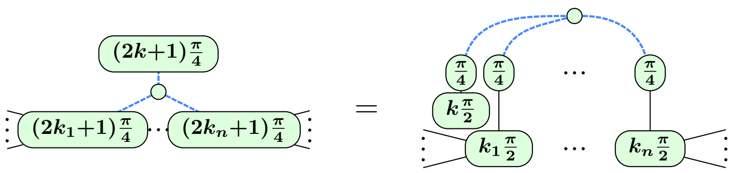
which has an efficiency of $\alpha = 0.25$.

Table 2.1 shows the best known efficiencies of the $|\mathrm{cat}_n\rangle$ decomposition for different values of $n$ [20].

| $n$ | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| terms | 2 | 2 | 3 | 3 | 6 | 6 | 9 | 9 | 27 | 27 | 27 | 27 |
| $\sim \alpha$ | 0.333 | 0.25 | 0.317 | 0.264 | 0.369 | 0.323 | 0.352 | 0.317 | 0.432 | 0.396 | 0.366 | 0.340 |

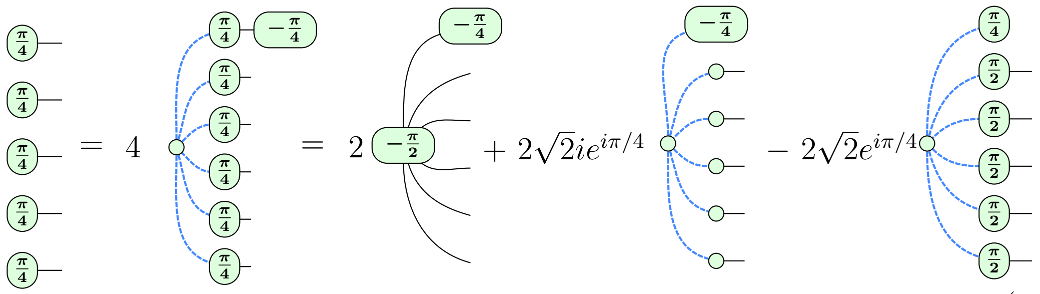
**Table 2.1:** $|\mathrm{cat}_n\rangle$ decomposition

Recall that if we are dealing with the Reduced gadget form [13], any internal spider that is Clifford will be part of a non-Clifford phase gadget, by definition. Further, no two Clifford spiders will be neighbours [3]. Then, we can apply the cat state decompositions to the cat states centered at the 0-spiders that are part of these phase gadgets in the reduced gadget form, if they have $\geq 3$ neighbours. This can be seen below, where $k, k_i \in \mathbb{Z}$ [3].



(2.17)

However, if the cat states are not present, we would need to fallback to a magic state decomposition. Here, the discovery of the (non-stabiliser) decomposition of 5-qubit magic states using the decomposition of $|\mathrm{cat}_6\rangle$ gives us a guaranteed way to decompose a circuit with a $T$-count $\geq 5$ [3].

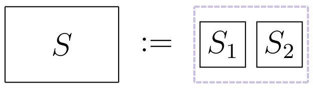


(2.18)

We then have a resulting $\alpha = \frac{\log_2(3)}{4} \approx 0.396$, which is already better than the BSS decomposition. In the next chapter, we will continue using the algorithm from [3] and improve upon the use of these cat and magic state decompositions.

## 2.5 Graph Cuts

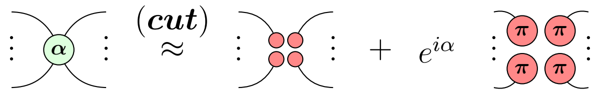
In a complementary approach to reducing the effective $\alpha$ of the stabiliser decomposition, graph cuts offer a different perspective [4]. Suppose we have 2 Clifford+T diagrams $S_1, S_2$ of $T$-counts $t_1, t_2$ that both have a stabiliser decomposition efficiency of $\alpha$.



(2.19)

If we combine the diagrams into a single diagram $S$, we could expect the stabiliser decomposition efficiency of $S$ to be $\alpha$, coming from having $2^{\alpha(t_1+t_2)}$ stabiliser states. However, since we can treat $S_1, S_2$ independently and multiply their stabiliser decompositions, the actual number of stabiliser states is instead $2^{\alpha t_1} + 2^{\alpha t_2} \leq 2^{\alpha t+1}$, where $t = \max(t_1, t_2)$. This gives a resulting $\alpha' \leq \alpha\frac{t+1}{t_1+t_2}$. In the ideal case, we have that $t_1 = t_2$, so $\alpha' \leq \frac{\alpha}{2} + \frac{1}{2t}$, a vast improvement especially for a high $T$-count. It is with this motivation that graph cuts are another promising approach.

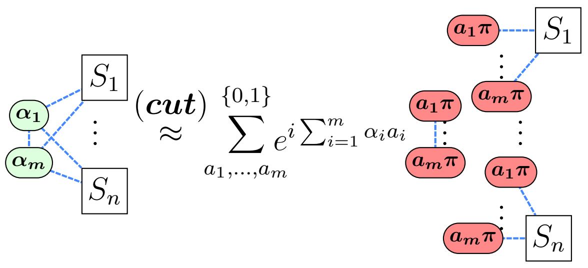
First, by definition, we can see that



(2.20)

Note that while this decomposition (***cut***) is not necessarily on a state, because of OCM, and the use of cups and caps, we can easily consider this to fit the definition of a Decomposition.

Then if we have a diagram with subdiagrams $S_1, ..., S_n$, we can see that the following holds.



$\alpha_1$
$\alpha_m$
$S_1$
$S_n$
(*cut*)
$\approx$
$\sum_{a_1,\ldots,a_m}^{\{0,1\}} e^{i\sum_{i=1}^{m}\alpha_i a_i}$
$a_1\pi$
$a_m\pi$

(2.21)

At the cost of $m$ cuts, and thus $2^m$ terms, the diagram is split into $\leq m^2 + n$ subdiagrams that can be treated independently. The $\leq m^2$ scalars in the form of spider-pairs are trivially scalars (1.10), and there are only $\leq n$ subdiagrams with $T$-counts $> 0$. If each subdiagram $S_i$ has the $T$-count of $t_i$, we have a sum of $2^m$ terms, each value which can be obtained with just $\leq m^2 + \sum_{i=1}^n 2^{\alpha t_i}$ stabilizer terms. This gives a total of $\leq 2^m(m^2 + \sum_{i=1}^n 2^{\alpha t_i})$ stabilizer terms. We can obtain the efficiency.

$$\begin{aligned} \alpha' &\leq \frac{\log_2(2^m(m^2 + \prod_{i=1}^n 2^{\alpha t_i}))}{\sum_{i=1}^n t_i} \\ &= \frac{m + \log_2(m^2 + \prod_{i=1}^n 2^{\alpha t_i})}{\sum_{i=1}^n t_i} \\ &\leq \frac{m + 2\log_2(m) + \sum_{i=1}^n \alpha t_i}{\sum_{i=1}^n t_i} \end{aligned} \tag{2.22}$$

In the ideal case where $t_i = t$ for all $i$, we can produce a crude upper bound

$$\begin{aligned} \alpha' &\leq \frac{\alpha}{n} + \frac{m + \log_2(m^2 n)}{nt} \\ &< \frac{\alpha}{n} + \frac{3m + n}{nt} \\ &= \frac{\alpha}{n} + \frac{3m}{nt} + \frac{1}{t} \end{aligned} \tag{2.23}$$

Keeping the ratio $m/n$ small is key to reducing the effective $\alpha$. Clearly, fewer cuts are better, with preferably the end result being many disconnected subdiagrams.

Another way graph cuts also reduce the effective $\alpha$ need not necessarily be by obtaining disconnected subdiagrams. On pseudo-structured circuits produced from CNOTs, phase gates, and Toffolis, graph cuts were used to take advantage of the structure to optimise for the use of spider $(\boldsymbol{f})$usion in cascading fashion, where a single cut can lead to a large reduction in the $T$-count [5]. There, the effective efficiency obtained was $0.1 \lesssim \alpha \lesssim 0.2$. More details on this will be discussed in Section 3.3.

The same was also found for extremely structured circuits involving cat states, where a single decomposition reduced the $T$-count by 286, for an $\alpha \approx 0.0035$ [4], though this was an ideal case which is unlikely to occur in practice.

## 2.6 Subgraph Complement

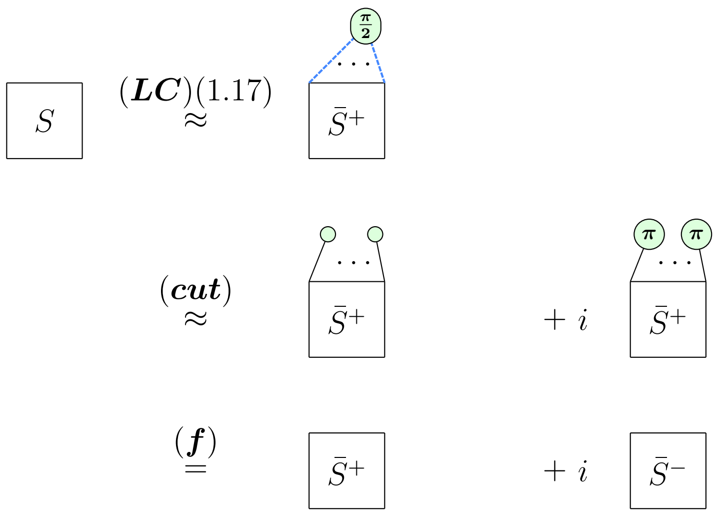
A further addition to the strength of graph cuts is the subgraph complement approach. For a graph-like ZX-diagram $(V, E)$, with $K$ being the set of edges for a complete graph on $V$, its *complement* is simply $(V, K\backslash E)$. We define a *subdiagram* to simply be a subgraph of the graph-like ZX-diagram. If we analyse $(\boldsymbol{LC})$ closely, and consider the Hopf rule $(\boldsymbol{x})$ making parallel edges cancel, there seems to be a way to obtain a subgraph complement with some manipulation of ZX-rules. In fact, we have the following result.

**Proposition 2.6.1** ([4])**.** Suppose we have a subdiagram $S$ of a graph-like ZX-diagram. Then

$$\boxed{S} \approx \boxed{\bar{S}^{+}} + i\boxed{\bar{S}^{-}} \tag{2.24}$$

where $\bar{S}^{+}, \bar{S}^{-}$ are the complements of $S$ where the spider phases differ from $S$ by $\pm\frac{\pi}{2}$ respectively.

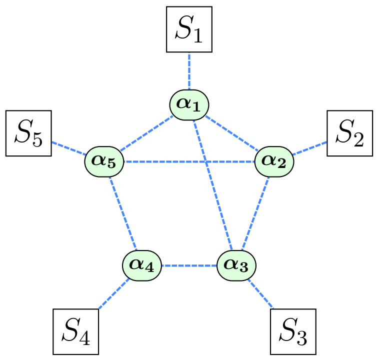
*Proof.* The proof is adapted from [4].


$S$
$\overset{(\boldsymbol{LC})(1.17)}{\approx}$
$\frac{\pi}{2}$
$\bar{S}^{+}$
$\overset{(\boldsymbol{cut})}{\approx}$
$\bar{S}^{+}$
$+\,i$
$\pi$
$\pi$
$\bar{S}^{+}$
$\overset{(\boldsymbol{f})}{=}$
$\bar{S}^{+}$
$+\,i$
$\bar{S}^{-}$

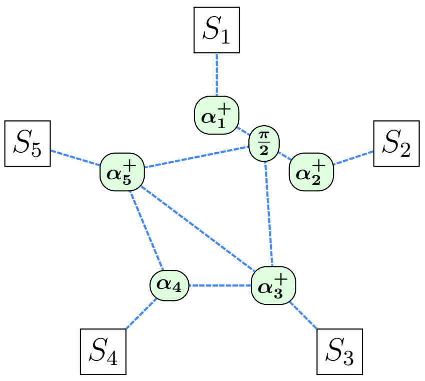
(2.25)

□

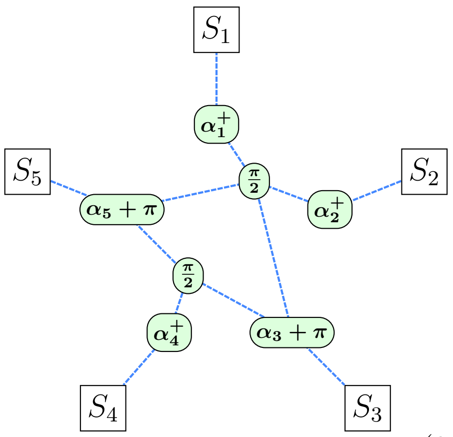
The power of this proposition can be demonstrated with the following example. Suppose we have the following graph-like ZX-diagram adapted from [21], where $S_i$ are subdiagrams.



$$\overset{(\boldsymbol{LC})}{\approx}$$



$$\overset{(\boldsymbol{LC})}{\approx}$$



(2.26)

Following from (2.21), we can apply 5 cuts to split the diagram into 5 disconnected subdiagrams, giving us $2^5$ terms. However, we can apply Prop. 2.6.1 twice to give us $2^2$ terms instead.

# 3 Methods

## Contents

*This chapter details heuristics used to speedup the stabiliser decomposition process. It involves the novel conceptualisation of a cut heuristic that is valid and useful, and details the implementation of the heuristic in practice. The heuristic is closely tied with the efficiency value $\alpha$, with its algorithmic implementation building off of work by Kissinger et al. [3]. The methods used by Sutcliffe and Kissinger, as well as Codsi, are also formalised using this idea of a cut heuristic [4, 5].*

*Novel heuristics used in experiments for the Results chapter are also detailed, and include those in Section 3.5, 3.6, 3.7, and 3.8. The updated algorithm is then presented in Section 3.9, and the complexity of each heuristic is discussed in Section 3.11.*

## 3.1 Introduction

The task of obtaining the stabiliser decomposition for classical simulation purposes has only been discussed at a high level so far. While we have the various methods of magic state, cat state decompositions, along with graph cut decompositions, we have not yet detailed the specific steps taken to obtain the stabiliser decompositions in the most efficient way known.

In modeling our task, we can think of the stabiliser decomposition almost as a search problem, where we are searching for the most efficient stabiliser decomposition.

We can visualize an example as in Figure 3.1, where $V$ is a Clifford+T diagram at the root. The directed edges represent the decomposition used from the parent to the child node, and the leaves $d_i$ are the possible stabiliser decompositions.

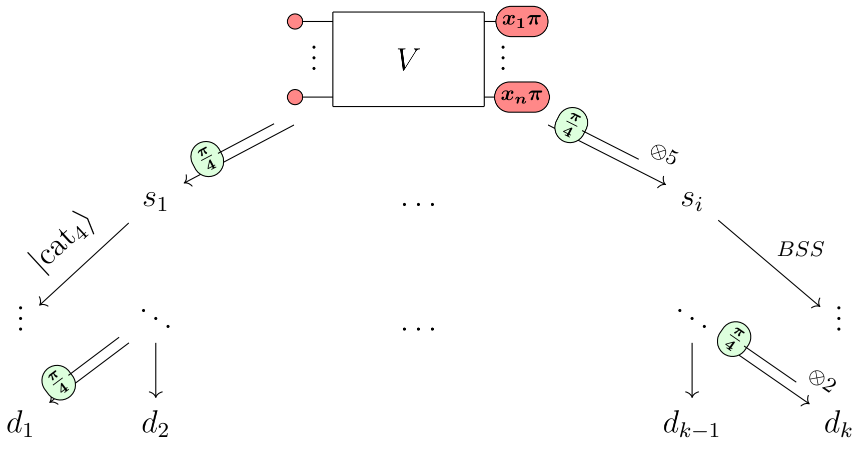


**Figure 3.1:** Example tree for a stabiliser decomposition

One question that arises is how we can efficiently traverse this tree. It is strongly believed that the stabiliser rank itself must increase exponentially with the $T$-count $t$, so we cannot have $\chi(V) = 2^{o(t)}$ [22], as cited in [14]. Thus, even if it was possible to know the most efficient decomposition, at each node, we would still face the exponential number of terms. In Figure 3.1, this means that not just that we have an exponential number of decompositions to consider, but also that every stabiliser decomposition $|d_i| = O(2^{\alpha t})$ would have an exponential number of terms.

**Algorithm 3.1.1.** To traverse the tree, we might take the following steps [2, 3].

1. Perform necessary spider $(\boldsymbol{f})$usions to reduce to graph-like form.
2. Apply a chosen decomposition ("choose an edge").
3. Simplify the resulting Clifford+T diagram using ZX-simplify. Some Clifford spiders may be removed, and some $T$-like spiders may be combined.
4. Repeat steps 1-3 until the $T$-count is 0.

The complexity from using this algorithm was found to be $O(N^3 + 2^{\alpha t}t^2)$, where $N$ is the number of gates in the original circuit [2]. Clearly, any circuit with a $T$-count $t$ that is not insignificant will mean that the complexity is dominated by $O(2^{\alpha t}t^2)$.

In this thesis, we will focus on the second step of the algorithm, where we choose the decomposition to apply at each node. There are a variety of ways to choose a decomposition, detailed in the next few sections.

## 3.2 Greedy Algorithm

The most obvious way to obtain an efficient decomposition is to, at each step, use the decomposition that has the smallest associated value of $\alpha$ as possible. This is essentially a greedy algorithm, and is the method used in [2, 3]. Intuitively, this should give us a decomposition with an $\alpha$ upper bounded by the worst decomposition available. More formally, we have the following lemma.

**Lemma 3.2.1.** Let $d$ be a stabiliser decomposition, and let $\alpha_1, ..., \alpha_n$ and $t_1, ..., t_n$ be the $\alpha$ values and $T$-count reductions of each of the $n$ constituent decompositions to obtain $d$ from the initial Clifford+T diagram. Then the efficiency $\alpha$ value of the decomposition $d$ is given by

$$\alpha(d) \leq \frac{\sum_{i=1}^{n} \alpha_i t_i}{\sum_{i=1}^{n} t_i} \tag{3.1}$$

*Proof.* Each decomposition gives $\leq 2^{\alpha_i t_i}$ terms, for a total of

$$\leq \prod_{i=1}^{n} 2^{\alpha_i t_i} = 2^{\sum_{i=1}^{n} \alpha_i t_i}$$

terms. Since the final $T$-count is 0, the initial $T$-count is $\sum_{i=1}^{n} t_i$. By definition, we have

$$\alpha \leq \frac{\sum_{i=1}^{n} \alpha_i t_i}{\sum_{i=1}^{n} t_i}$$

$\square$

This tells us that the final $\alpha$ value is essentially a weighted average of the $\alpha_i$ values, weighted by their $T$-count reductions. It is easy to see that if all decompositions have $\alpha_i \leq \alpha_{max}$, then (3.1) implies the overall $\alpha \leq \alpha_{max}$. The actual $\alpha$ value in practice is likely to be much lower than this upper bound, because of step 3 in Algorithm 3.1.1, where we simplify the diagram using ZX-simplify [2]. In [2], although the BSS decomposition has the $\alpha \approx 0.468$, the actual $\alpha$ value obtained in experiments was lower, at $\approx 0.4$ [5].

A step further than using the $\alpha$ value of a decomposition, is to consider the $\alpha$ value when including the simplification step. In [4], this is defined as the *effective* $\alpha_e$ value. If the associated number of $T$-like spiders removed is $r, r_e$, where $r \leq r_e$ before and after simplification respectively, then we have

$$\alpha_e = \frac{r}{r_e}\alpha \tag{3.2}$$

This relation can make a significant difference especially for low values of $r$. In the next section, we also consider the structure of the diagram to increase $r_e$.

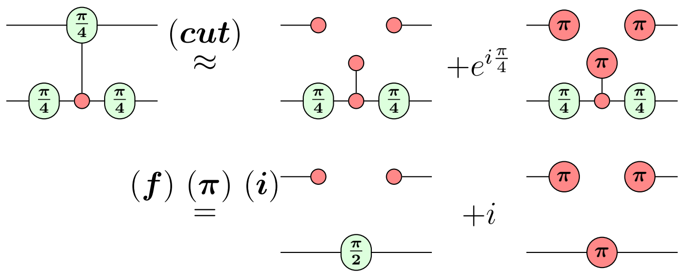
Recall also that the Algorithm 3.1.1 has $O(t^2)$ part of the complexity between each step of the decompisition. Given that we have $k$ decompositions to choose from at each node, if we employed a brute force approach to find the best stabiliser decomposition, and attempted to find the best decomposition from a node traversing down $d$ levels in the tree, it would take $O(k^{d+1}m^d t^2)$ time, where $m$ is the maximum number of terms coming from a decomposition, for the $k$ decompositions to be applied to all the $m$ terms after each level. For example, the above approach of obtaining $\alpha_e$ takes $d = 0$.

In theory, this could be extended to larger values of $d \geq 1$, but with $km \geq 2$, this would mean that the overall exponential complexity of $O(2^{\alpha t + d \log mk} kt^2)$ would be affected. This motivates the idea of using heuristics instead.

## 3.3 Cut Heuristic

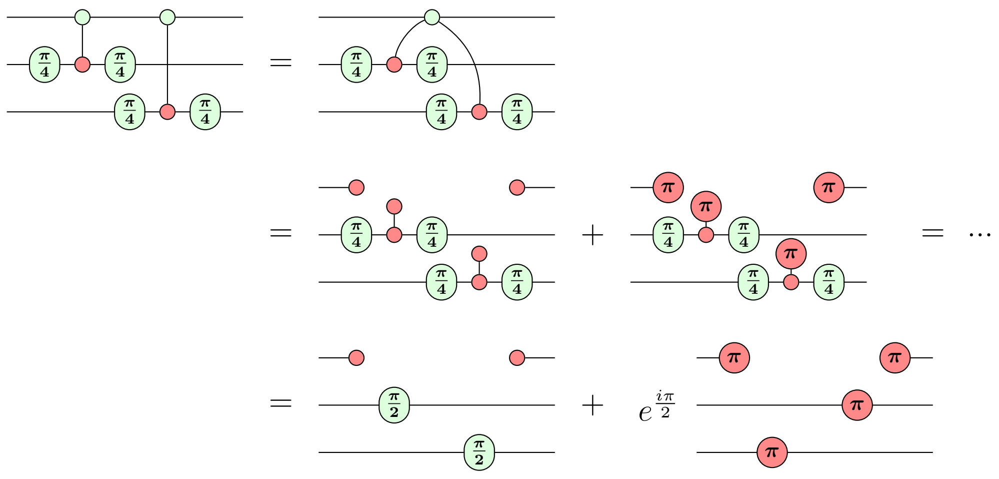
A disadvantage of the greedy algorithm is that it does not take into account the structure of the diagram. When we only consider the $\alpha$ value of the decomposition, we fail to take into account that $T$-like spiders can be fused in between decompositions where decompositions with seemingly higher $\alpha$ values may actually be more efficient.

In [5], the heuristic used considers the structure of CNOT gates placed in between $T$ gates. The following observation was made.



(3.3)

Using the graph $(\boldsymbol{cut})$, which without simplification has $\alpha = 1$ if the spider being cut is a $T$-like spider, we can instead obtain a decomposition with $\alpha \approx 0.333$, with simplication, for the reduction of 3 $T$-like spiders at the cost of 2 terms. In fact, this structure can be stacked in the following way.


(3.4)

Using these observations, a heuristic weighting function can be developed, where the main idea is to count the number of $T$-like spiders that can be removed, divided by how many cuts are needed to remove them.

**Definition 3.3.1.** Let $V$ be a Clifford+T diagram, and let $v$ be a vertex in $V$. Let $C$ be the set of sequence of graph cuts. For each $c = (v_c, ...) \in C$ where there are $|c| = n$ cuts in the sequence, let $t(c)$ be the number of $T$-like spiders removed by the sequence of cuts using (***cut***), including simplifications between each cut.

The *effective cut heuristic* $h_e : V \to \mathbb{R}^+$ for a sequence $c$ of length $n$ is defined as

$$h_e(v_c) = \frac{t(c)}{|c|} \tag{3.5}$$

A cut heuristic $h : V \to \mathbb{R}^+$ for sequences $c$ of length $n$ is *valid* if

$$\forall v \in V, \exists c = (v, ..., v_n) \in C, h(v) \leq h_e(v) \tag{3.6}$$

▲

Note that a useful cut heuristic is one where the value of $h(v)$ is close to $h_e(v)$, $\forall v \in V$. We have that a higher cut heuristic value implies a more efficient decomposition. In fact, the associated $\alpha_e$ value of the best sequence of (***cut***) decompositions $c$ is given by

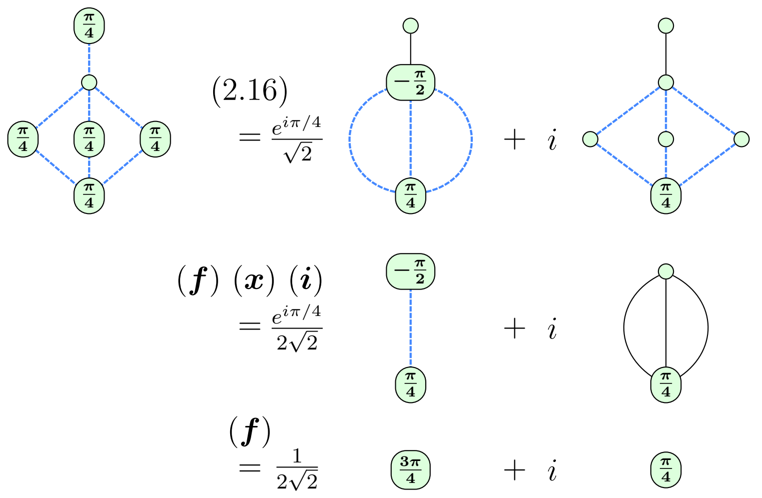
$$\alpha_e(c) = \frac{|c|}{t(c)} = \frac{1}{h_e(v_c)} \leq \frac{1}{h(v_c)} = \alpha_h(c) \tag{3.7}$$

where $\alpha_h(c)$ is the associated $\alpha$ value of the cut heuristic, and since the $n = \log_2(2^n)$ cuts contribute to $2^n$ terms.

If the sequence of cuts leads to a stabiliser decomposition, the effective cut heuristic is closely related to our $\alpha_\chi$ value, but does not take into account the other decompositions we have available. For some Clifford+T diagram $V$, let $C$ be the set of sequences of decompositions resulting in a stabiliser decompositions that only use graph cuts, and $D$ be the set of stabiliser decompositions. Let $f : C \to D$ be the function that maps a sequence of cuts to the decomposition obtained. Let $c^* = (v^*, ...) \in C$ be the most efficient decomposition using graph cuts such that

$$|f(c^*)| = \min\{|f(c)| \mid c \in C\} \tag{3.8}$$

Here, $\forall c = (v, ...) \in C, h_e(v^*) \geq h_e(v)$. If $c^*$ results in a decomposition where $|f(c^*)| = \chi(V)$, then $\alpha_\chi = \alpha_e(c^*)$ [1]. In general, however, $\alpha_\chi \leq \alpha_e(c^*)$. A simple example of this can be seen in the following diagram, where the decomposition makes use of $|\text{cat}_4\rangle$ (2.16). It starts with a $T$-count of 5.



(3.9)

Using only cuts would require 2 of them, which would give $2^2$ terms instead of 2 terms. We have

$$\alpha_\chi \leq \frac{1}{5} < \alpha_e(c^*) = \frac{2}{5} \tag{3.10}$$

## 3.4 CNOT Sandwich Heuristic

In [5], the cut heuristic used concentrated on sequences of cuts that were looking to eliminate CNOT gates sandwiched between $T$-like spiders gates. As a result, 71% of the time, they were able to obtain $c^*$ on small circuits. Ultimately, efficiency values of $0.1 \lesssim \alpha \lesssim 0.2$ were obtained in practice, considering semi-structured diagrams.

[1] Here, and in (3.7), we overload notation by equating $\alpha(f(c^*)) = \alpha(c^*)$

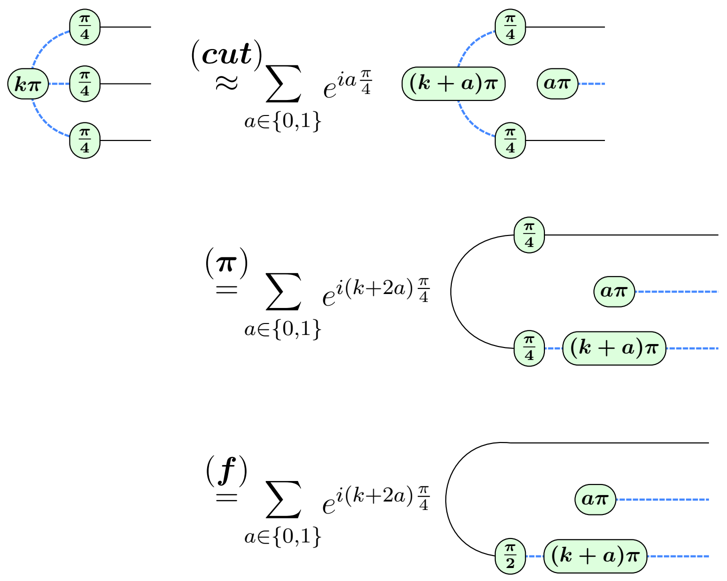
Using the cut heuristic means that there is some expert knowledge on the sequences of cuts to be made. With better consideration of sequences, we can continue to use the greedy approach on $\alpha$ values, with the hope of obtaining a more efficient decomposition. In the tiered approach of [5], a brief description is that increasing lengths of sequences were considered, where each sequence of cuts $c_i$ for tier $i$ led to computation of the cut heuristic $h_{i+1}(v)$, corresponding to sequences $c_{i+1} = k_i : c_i$ for tier $i+1$ for some sequence of vertices $k_i = (v, \ldots)$. Eventually, the algorithm terminates when there is some $l = i_{\max}$ where there are no sequences of vertices $k_l$ to compute $c_{l+1}$ at tier $l+1$. The result is the computation of $h_i$ values for tiers $i \leq l$, and the choice to cut $v$ where

$$v_{best} := \arg\max_v h_l(v) = \arg\min_v \alpha_{h_l}(v) \tag{3.11}$$

## 3.5 2-Legged Heuristic

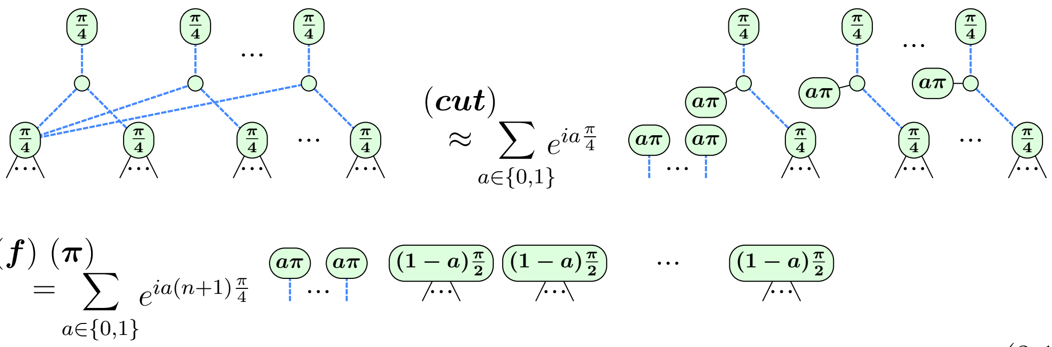
The cut heuristic is also useful outside of CNOT sandwiches. Particularly, the reduced gadget form contains structures that when cut, reduce the $T$-count greatly.

In [4], the following observation was made when applying the cut to a $|\text{cat}_3\rangle$.


$\frac{\pi}{4}$
$k\pi$
$\overset{(\boldsymbol{cut})}{\approx} \sum_{a\in\{0,1\}} e^{ia\frac{\pi}{4}}$
$(k+a)\pi$
$a\pi$
$\overset{(\boldsymbol{\pi})}{=} \sum_{a\in\{0,1\}} e^{i(k+2a)\frac{\pi}{4}}$
$\overset{(\boldsymbol{f})}{=} \sum_{a\in\{0,1\}} e^{i(k+2a)\frac{\pi}{4}}$
$\frac{\pi}{2}$

(3.12)

The associated efficiency is $\alpha = 1/3$ here, from removing 3 $T$-like spiders at the cost of 1 cut. This means that any $T$-like spider that is part of multiple $|\text{cat}_3\rangle$ can contribute to removing all the associated $T$-like spiders at once. A visualisation can be seen below (3.13). It can be noted that a $|\text{cat}_3\rangle$ and a two-legged phase gadget are interchangeable concepts in the reduced gadget form.



(cut)
$\approx \sum_{a\in\{0,1\}} e^{ia\frac{\pi}{4}}$
(f) (π)
$= \sum_{a\in\{0,1\}} e^{ia(n+1)\frac{\pi}{4}}$

(3.13)

In [4], the effective $\alpha_e$ value was used, after determining the number of $|\text{cat}_3\rangle$ each $T$-like spider was part of, in order to find the most suitable cut to compare with the other decompositions. As such, there was no calculation of the heuristic $\alpha_h$ value. We will see in subsection 4.1.2 as well, how this observation was related to the improved bounds on the complexity of simulating IQP circuits [6].

Here, we can define the heuristic as follows.

$$h(v) = 1 + 2 \cdot (\#\ |\text{cat}_3\rangle\ v \text{ is part of}) \tag{3.14}$$

since each neighbouring $|\text{cat}_3\rangle$ can contribute to the removal 2 $T$-like spiders.

**Proposition 3.5.1.** For any ZX-diagram in reduced gadget form, the heuristic $h(v)$ (3.14) is a valid cut heuristic for a single cut if $v$ is $T$-like.

$$\forall v \in V, v \text{ is T-like} \implies 0 < h(v) \leq \frac{t(c)}{|c|} = h_e(v)$$

*Proof.* If $v$ is part of a phase gadget, this is trivial, as $h(v) = 3$. Suppose $v$ has neighbours $w_1, ..., w_k$ that are 0-spiders that are part of $|\text{cat}_3\rangle$ states. Note that $w_1, ..., w_k$ are all part of phase gadgets, by Definition 1.4.5. Each $w_i$ then has 2 legs, one of which is connected to $v$. Since no two phase gadgets have the same exact legs, we must have for all $w_i, w_j, i \neq j$, that their other legs do not connect to the same vertex $u$. By cutting only $v$, we can remove at least $2k + 1$ $T$-like spiders corresponding to $k$ $|\text{cat}_3\rangle$ states, and vertex $v$. Then we must have

$$h_e(v) \geq 2k + 1 = h(v) \tag{3.15}$$

□

We can note that even if $k$ is small, the heuristic can be quite effective. For instance, with $k = 2, \alpha_h = 0.2$ which is already an improvement over all the $|\text{cat}_n\rangle$ decompositions in [2, 3], the best of which is $\alpha = 0.25$ for the $|\text{cat}_4\rangle$

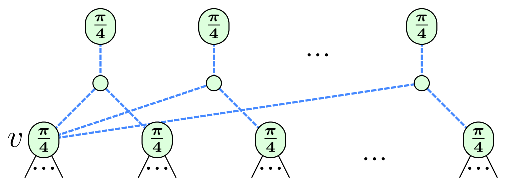


**Figure 3.2:** Diagram to showcase the heuristic (3.14)

decomposition. A $T$-like spider simply has to be connected to 2 $|\mathrm{cat}_3\rangle$ states to improve the decomposition.

More generally, it was also considered that cutting $T$-like spiders part of $|\mathrm{cat}_n\rangle$ states would convert them to $|\mathrm{cat}_{n-1}\rangle$ states. However, this was not considered in the heuristic of [4], as they focused on obtaining the single-step $\alpha_e$ value.

The heuristic considering $|\mathrm{cat}_n\rangle$ states could then be defined as follows.

$$h(v) = \sum_{i=1}^{k} \frac{2}{n(w_i) - 2} \tag{3.16}$$

where $w_i$ are neighbours of $v$ that are 0-spiders that are part of $\left|\mathrm{cat}_{n(w_i)}\right\rangle$ states, for $n : V \to \mathbb{N}$. The heuristic considers sequences of cuts for every $\left|\mathrm{cat}_{n(w_i)}\right\rangle$ that $v$ is part of, and attempts to add up each pair of $T$-like spiders that could be removed with $n(w_i) - 2$ cuts. Unfortunately, this heuristic is not a valid heuristic in general.

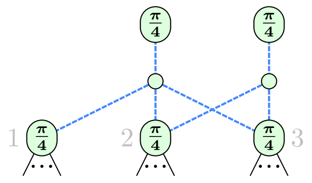


**Figure 3.3:** Diagram where the heuristic (3.16) is not valid

Here, the heuristic would give $h(v_2) = \frac{2}{1} + 2 = 3$, but cutting 2 vertices to remove 5 $T$-spiders would mean $h_e(v_2) = \frac{5}{2} < 3$.

We show that with an added condition, the heuristic is valid.

**Proposition 3.5.2.** Suppose $v$ has 0-spider neighbours $w_1, ..., w_k$ that are part of all $\left|\mathrm{cat}_{n(w_i)}\right\rangle$ states. Let $T(w_i)$ be the set of $T$-like spiders that is part of each $\left|\mathrm{cat}_{n(w_i)}\right\rangle$ state, so $|T(w_i)| = n(w_i)$. Let $r = \frac{\max\{n(w_i)\}_i - 2}{\min\{n(w_i)\}_i - 2}$. Then for the heuristic $h(v)$ (3.16), we have

$$\left|\bigcup_{i=1}^{k} T(w_i)\right| \geq 2k^2 r \implies \exists c = (v, ...) \in C, 0 < h(v) \leq \frac{t(c)}{|c|} = h_e(v) \tag{3.17}$$

In other words, if the number of distinct $T$-like spiders part of all the $\left|\text{cat}_{n(w_i)}\right\rangle$ states is at least $2k^2r$, then the heuristic $h(v)$ is a valid cut heuristic.

*Proof.* Note that $w_1, ..., w_k$ are all part of phase gadgets, by Definition 1.4.5. By cutting at most $\sum_{i=1}^{k}(n(w_i) - 2)$ $T$-like spiders as legs of the phase gadgets, we can remove at least $\left|\bigcup_{i=1}^{k} T(w_i)\right|$ $T$-like spiders. Then we must have

$$h_e(v) \geq \frac{\left|\bigcup_{i=1}^{k} T(w_i)\right|}{\sum_{i=1}^{k}(n(w_i) - 2)} \tag{3.18}$$

Then

$$\begin{aligned} h(v) &= \sum_{i=1}^{k} \frac{2}{n(w_i) - 2} \\ &\leq \sum_{i=1}^{k} \frac{2}{\min\{n(w_i)\}_i - 2} \\ &= \frac{2kr}{\max\{n(w_i)\}_i - 2} \\ &= \frac{2k^2r}{k(\max\{n(w_i)\}_i - 2)} \\ &\leq \frac{\left|\bigcup_{i=1}^{k} T(w_i)\right|}{\sum_{i=1}^{k}(n(w_i) - 2)} \\ &\leq h_e(v) \end{aligned} \tag{3.19}$$

□

The proposition condition is generally unlikely to be satisfied, and ultimately leads to the heuristic performing poorly in general. From the proof, we can gather that there is in fact a cut heuristic that is valid.

**Corollary 3.5.3.** The following heuristic is a valid cut heuristic.

$$h(v) = \frac{\left|\bigcup_{i=1}^{k} T(w_i)\right|}{\sum_{i=1}^{k}(n(w_i) - 2)} \leq h_e(v) \tag{3.20}$$

Unfortunately, although the heuristic is valid, it can vastly underestimate the effective heuristic value. This happens when there is a large overlap in $T$-like spiders connected to different phase gadgets. A simple example is as follows, where we have $|\text{cat}_n\rangle$ for $n = 3, 4, 5$.

$h_e(v_1) \geq \frac{7}{3}$ since only 3 cuts are needed to remove 7 $T$-spiders, but $h(v_1) = \frac{4}{3+2+1} = \frac{2}{3} < \frac{7}{3}$.

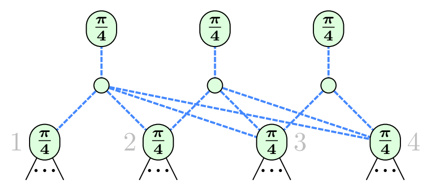
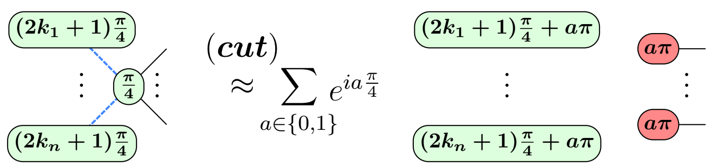


**Figure 3.4:** Diagram where the heuristic (3.20) underestimates the effective heuristic value

## 3.6 Lone Phase Heuristic

Another heuristic turns out to be somewhat common to measure is the lone phase heuristic. We can observe the following decomposition.


$(2k_1+1)\frac{\pi}{4}$
$\frac{\pi}{4}$
$(2k_n+1)\frac{\pi}{4}$
$(\boldsymbol{cut})$
$\approx \sum_{a\in\{0,1\}} e^{ia\frac{\pi}{4}}$
$(2k_1+1)\frac{\pi}{4}+a\pi$
$(2k_n+1)\frac{\pi}{4}+a\pi$
$a\pi$
$a\pi$

(3.21)

Since the legless spiders can easily be treated as scalars, a $T$-like spider connected to $n$ magic states $\frac{\pi}{4}$—— would allow the $T$-count to drop by $n$. We define the *lone phase heuristic* as follows.

$$h(v) = 1 + \#\ \tfrac{\pi}{4}\text{——}\ v \text{ is connected to} \tag{3.22}$$

It's clear to see that this is a valid cut heuristic.

What we have then is that we can combine the two "simple" heuristics (3.14, 3.22) so far. If $n$ is the number of $|\text{cat}_3\rangle$ states $v$ is part of, and $m$ is the number of magic states $\frac{\pi}{4}$—— $v$ is connected to, then we have the following heuristic.

$$h(v) = 1 + 2n + m \tag{3.23}$$

There is clearly no overlap between the two types of $T$-like spiders that each heuristic considers, so the heuristic is still valid. Surprisingly, just a simple combination of the two heuristics can be quite effective in practice, as we will see in the next chapter.

## 3.7 Paired Heuristics

The heuristics thus far concentrate on cutting single vertices, with the hope of further simplification aftewards. However, we can also consider heuristics that involve cutting pairs of vertices. Consider the following diagram.

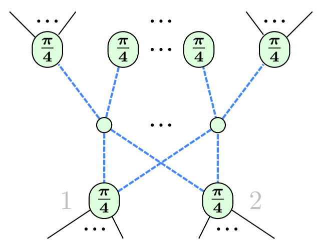
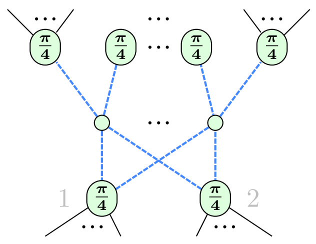
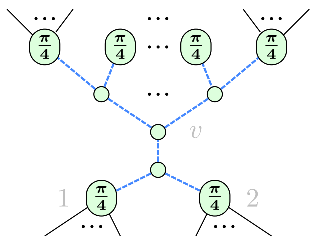


**Figure 3.5:** A pair of vertices connected to 3-legged phase gadgets

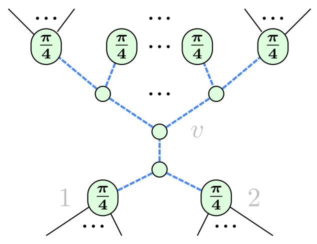
We see at the top that we have $n$ 3-legged phase gadgets (a.k.a. $|\mathrm{cat}_4\rangle$ states), fully connected to a pair of $T$-like spiders. At first glance, it might seem like gadget fusion (***GF***) can almost be applied, and that we need to remove the $n$ $T$-like spiders at the top to do so. However, this is not too optimal, since we at most remove $2n$ $T$-like spiders with $n$ cuts, giving an $\alpha = 0.5$. Instead, we can exploit the fully connected aspect of the pair of $T$-like spiders, by considering the (***P***)ivot rule again.



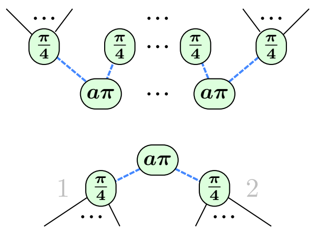
$$\overset{(\boldsymbol{P})}{\approx} \tag{3.24}$$



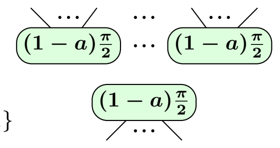
Then we can (***cut***) vertex $v$ to remove the $T$-like spiders.



$$\overset{(\boldsymbol{cut})}{\approx} \sum_{a\in\{0,1\}} \tag{3.25}$$

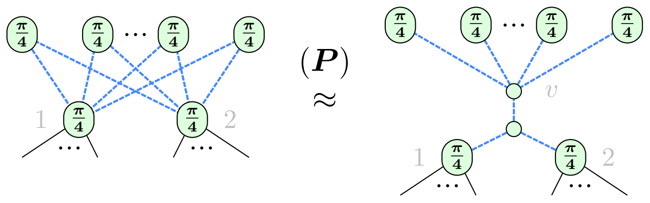


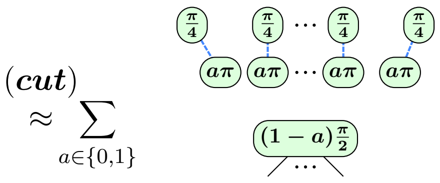
$$\overset{(\boldsymbol{f})}{=} \sum_{a\in\{0,1\}}$$



A total of $2n+2$ $T$-like spiders are removed with a single cut, giving an $\alpha = \frac{1}{2n+2}$.

We can go even further, if we draw inspiration from the Lone Phase Heuristic. If we have a pair of $T$-like spiders fully connected to $m$ 2-legged $T$-like spiders, we can again use the (***P***)ivot rule to our advantage.





(3.26)

We can summarise the paired heuristic as follows. Suppose $t_1, t_2$ are $T$-like spiders fully connected to $n$ 3-legged phase gadgets, and $m$ 2-legged $T$-like spiders respectively. Let $v$ be the vertex added after using the $(\boldsymbol{P})$ivot rule. Then we have two heuristics.

$$\begin{aligned} h_1(t_1,t_2) &:= h_1(v) = 2+2n \\ h_2(t_1,t_2) &:= h_2(v) = 2+m \end{aligned} \tag{3.27}$$

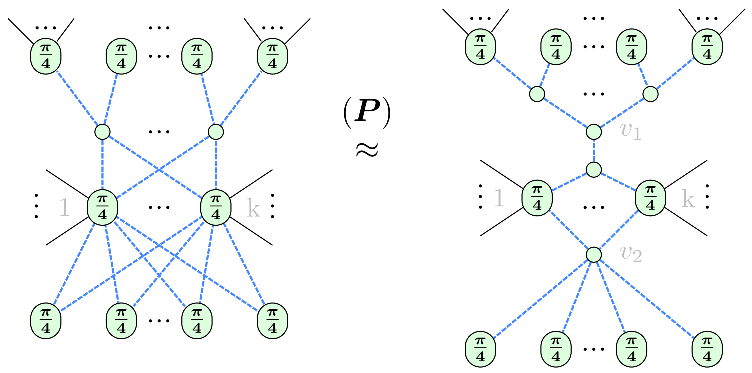
In this case, we don't necessarily merge the two heuristics since the pivot vertices are different. We could consider a combined heuristic, where the 2 pivots happen at the same time, so 2 cuts are needed to remove all the vertices.

$$\begin{aligned} h(v) &= \frac{2+2n+m}{2} \\ &< \frac{h_1(v)+h_2(v)}{2} \\ &\leq \max\{h_1(v), h_2(v)\} \end{aligned} \tag{3.28}$$

This is essentially the average of the two heuristics, which is necessarily smaller than the maximum between the two heuristics. Since we can process the heuristics sequentially, we can just consider them separately as before. By doing so, we maintain the validity of the heuristics with the same argument as before (3.23).

## 3.8 $k$-connected Heuristics

We can extend the idea of paired heuristics to $k$-connected heuristics. Suppose we have $k$ $T$-like spiders $\vec{t_k} = (t_1, ..., t_k)$ fully connected to $n$ $k+1$-legged phase gadgets, and $m$ $k$-legged $T$-like spiders. Using the $(\boldsymbol{P})$ivot rule, we can remove $m$ or $n$ $T$-like spiders with a single cut as before.



(3.29)

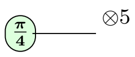
Then we have the following valid heuristics.

$$h_1(\vec{t_k}) = 2n$$
$$h_2(\vec{t_k}) = m \tag{3.30}$$

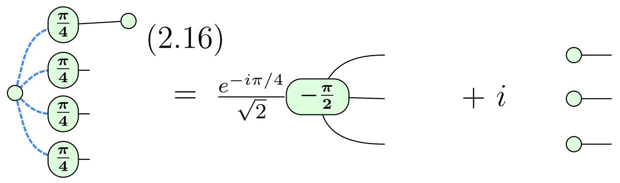
The drawback of this heuristic is that it can be relatively expensive to compute. We will see in the next Section 3.11 that the complexity of computing the heuristic is $O(kt^{k+1})$ given a $T$-count $t$. This results in an overall time complexity of $O(2^{\alpha t}t^{k+1})$, which practically affects runtime heavily for low $t$, even if $\alpha$ is reduced significantly. As such, we leave this general heuristic to be useful in much larger computations, and in numerical experiments of Chapter 4, we only consider the case of $k = 1, 2$, corresponding to (3.23, 3.27).

## 3.9 Greedy Algorithm with Cut Heuristic

A natural extension of the greedy algorithm is to also consider the cut heuristic on top of the cat decompositions.

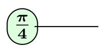
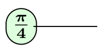
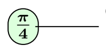
We have the $\alpha$ values from decompositions of $|\mathrm{cat}_4\rangle$, $|\mathrm{cat}_6\rangle$, $|\mathrm{cat}_5\rangle$, $|\mathrm{cat}_3\rangle$, and magic state  being $0.25 < 0.264 < 0.317 < 0.333 < 0.396$ respectively. Depending on the value of the best heuristic (3.23), $\max_v h(v)$, and the best available $|\mathrm{cat}_n\rangle$ decomposition in the diagram, we can continue to use the greedy approach. We have the simple Algorithm 1.

Even with a small values of $n, m$, the heuristic can be quite effective. Just having $n = m = 1$ can already give a $\alpha_h = 0.25$, matching the best $|\mathrm{cat}_4\rangle$ decomposition. Furthermore, this will cut the graph at a vertex, as compared to the $|\mathrm{cat}_4\rangle$ decomposition, which only locally partitions the second of its 2 terms, while its first term remains connected, seen below.



(3.31)

**Algorithm 1** Greedy Algorithm with Heuristic

```
1:  Input: $\alpha$ values for $|\text{cat}_4\rangle$, $|\text{cat}_6\rangle$, $|\text{cat}_5\rangle$, $|\text{cat}_3\rangle$, and $^{\otimes 5}$, and the graph $G$
2:  Output: Chosen state and corresponding decomposition
3:  Let $\alpha_4 = 0.25, \alpha_6 = 0.264, \alpha_5 = 0.317, \alpha_3 = 0.333, \alpha_{\otimes 5} = 0.396$
4:  Compute the best catstate $|\text{cat}_{best}\rangle$ in $G$ with the lowest $\alpha$ value
5:  Let $\alpha_{best}$ be the $\alpha$ value of $|\text{cat}_{best}\rangle$, or of $^{\otimes 5}$ if no $|\text{cat}_{best}\rangle$ is found
6:  Compute $\alpha_h$ on $G$ using a heuristic
7:  if $\alpha_h < \alpha_{best}$ then
8:      Use the decomposition corresponding to the heuristic
9:  else
10:     Use the decomposition corresponding to $|\text{cat}_{best}\rangle$ or $^{\otimes 5}$
11: end if
```

This heuristic also works in a diagram without any phase gadgets where $n = 0$, concentrating on the $T$-like spiders. In this scenario, having $m = 2$ will already give a better $\alpha_h = 1/3$ than the default 5-qubit magic decomposition at $\alpha \approx 0.396$.

## 3.10 Other Heuristics

Another heuristic considered, based off (3.20) takes into account the overlap of $T$-like spiders, and not overcount the number of cuts needed. Suppose the $T$-spider part of each phase gadget $w_i$ is labelled vertex $g_i$. Each phase gadget as part of a $|\text{cat}_n\rangle$ would only need a maximum of $n-2$ cuts, so our heuristic would look like

$$\frac{\left|\bigcup_{i=1}^{k} T(w_i)\right|}{1 + \left|\bigcup_{i=1}^{k} T(w_i) - \{g_i, v, u_i\}\right|} \tag{3.32}$$

where $\forall i, u_i \in T(w_i)$ is a chosen $T$-like spider on the leg of phase gadget $w_i$, that does not need to be cut. We would want to choose the $u_i$ that minimizes the denominator, so we have the following heuristic.

$$h(v) = \max_{\forall i, u_i \in T(w_i)} h(v) \tag{3.33}$$

where the denominator is minimized.

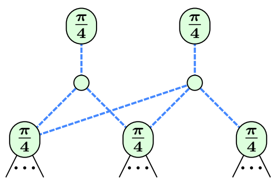
$$t_{\min} = 1 + \min_{\forall i, u_i \in T(w_i)} \left|\bigcup_{i=1}^{k} T(w_i) - \{g_i, v, u_i\}\right| \tag{3.34}$$

It is easy to see that this heuristic is valid, by the same argument as the previous Proposition 3.5.2. Solving this is akin to maximum bipartite matching. This is because to minimize the number of cuts, the number of $T$-like spiders matched with a phase gadget needs to be maximized, while only one $T$-like spider can be matched with each phase gadget. The matching can be found in polynomial time using any maximum bipartite matching algorithm like the Hopcroft-Karp algorithm [23]. Unfortunately, this heuristic is not as effective as the previous heuristic (3.14) just involving $|\text{cat}_3\rangle$ states.

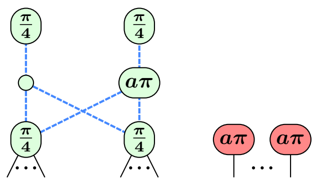
The phase gadget heuristic is also considered, where the $T$-like spiders connected to the legs of two phase gadgets $w_i, w_j$ are compared. Let $T(w_i), T(w_j)$ be the $T$-like spiders connected to the legs of $w_i, w_j$ respectively. Recall from ($\boldsymbol{GF}$), that gadgets with the same set of legs can be fused. If all the $T$-like spiders from the symmetric difference $T(w_i)\Delta T(w_j)$ are removed, where

$$A\Delta B = (A\cup B) - (A\cap B) \tag{3.35}$$

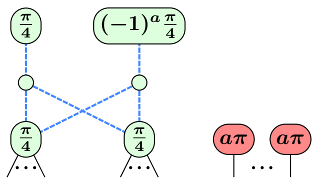
then the $T$-count can be reduced by 2 via gadget fusion ($\boldsymbol{GF}$), as seen below.



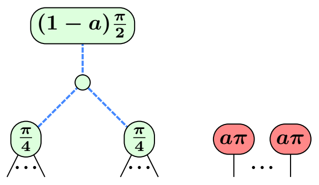
$$\overset{(\boldsymbol{cut})}{\approx} \sum_{a\in\{0,1\}} e^{ia\frac{\pi}{4}} \tag{3.36}$$



$$\overset{(\boldsymbol{f})}{=} \sum_{a\in\{0,1\}} e^{ia\frac{\pi}{4}}$$



$$\overset{(\boldsymbol{GF})}{=} \sum_{a\in\{0,1\}} e^{ia\frac{\pi}{4}}$$



We could then have the following heuristic, for the simplest case where

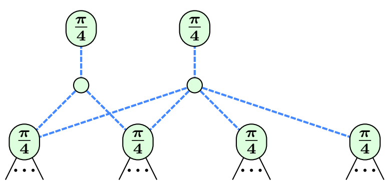
$$\exists i, j, T(w_i)\Delta T(w_j) = \{g_i, g_j, v\}$$

$$h(v) = 1 + 2\cdot |\{\{w_i, w_j\} \mid T(w_i)\Delta T(w_j) = \{g_i, g_j, v\}\}| \tag{3.37}$$

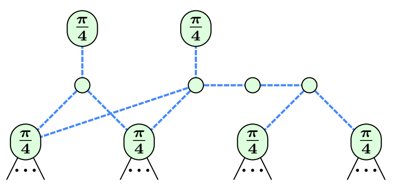
where $g_i, g_j$ are the $T$-like spiders part of the phase gadgets $w_i, w_j$ respectively.

This heuristic is valid, due to the requirement that $\{w_i, w_j\}$ must be distinct. As such, there is no overcounting of the number of $T$-like spiders that can be removed.

Similarly, the paired version of the heuristic can be considered, where there are 2 extra $T$-like spiders preventing gadget fusion.

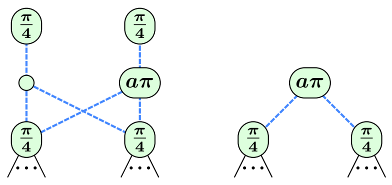


$$\overset{(\boldsymbol{f})}{=}$$

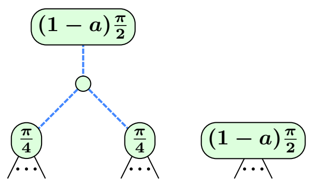


(3.38)

$$\overset{(\boldsymbol{cut})}{\approx} \sum_{a\in\{0,1\}}$$



$$\overset{(\boldsymbol{GF})\ (\boldsymbol{\pi})\ (\boldsymbol{f})}{\approx} \sum_{a\in\{0,1\}}$$



Alas, these phase gadget heuristics are not effective in practice, and earn their place in the "other heuristics" category. Scenarios for being close to gadget fusion are not common enough, and tend to take more cuts than necessary, by which time the cut heuristics (3.23, 3.27) would have already been more effective.

In Section 2.6, we introduce the subgraph complement cut. The heuristic that could be used here is simply to determine the number of edges in the subgraph excluding the phase gadgets, and to carry out the cut if the number of edges was above a certain threshold. For example, with $k$ vertices in the subgraph, we could cut if there were $> p \cdot \frac{k(k-1)}{2}$ edges for some $0.5 < p \leq 1$. We only tested with $p = 3/4$ and chose the subgraph of vertices not part of the phase gadgets, which did not seem to affect the results much.

Smartly choosing the right subgraph that are near cliques to execute the cut would be useful here, as demonstrated in Section 2.6, but finding near cliques is at least as hard as the maximum cliques problem, which itself is NP-hard [24]. However, there is work done on partitioning graphs that will be considered in future work [21].

## 3.11 Complexity Analysis

Here, we consider the complexity of computing the heuristics.

Suppose we have a reduced gadget form $V$ with $n$ phase gadgets and $m$ $T$-like spiders not part of phase gadgets. Let $d(v)$ be the degree of a vertex $v$.

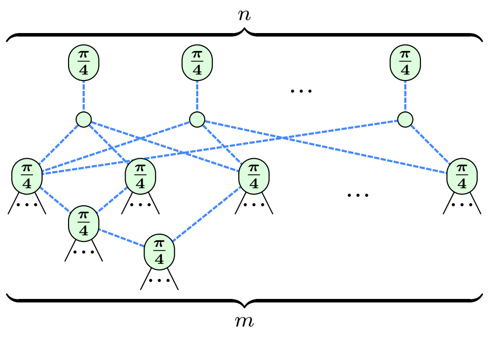


**Figure 3.6:** ZX-diagram in reduced gadget form, with $n$ phase gadgets and $m$ non-phase gadget $T$-like spiders

For the heuristic (3.14), it is easy to compute the heuristic in $O(mn)$ time. Checking which 0-spiders are part of $|\text{cat}_3\rangle$ states can be done in $O(n)$. Each $T$-like spider $v$ can be processed in $O(d(v)) = O(n)$ time, and checking for all of them takes $O(mn)$, for an overall of $O(n + mn) = O(mn)$ time.

The heuristic (3.16) has a similar complexity of $O(mn)$.

For the $|\text{cat}_n\rangle$ heuristic (3.20), a single $T$-like vertex $v$, and its 0-spider neighbours $w_1, ..., w_k$, it takes

$$O\left(d(v) + \sum_{i=1}^{k} d(w_i)\right) = O(n + mn) = O(mn) \tag{3.39}$$

time to compute the heuristic. For all $m$ $T$-like spiders, this takes $O(m^2 n)$ time.

The lone phase heuristic (3.22) can be computed in $O(m^2)$ time, since each $T$-like spider can be processed in $O(m)$ time, for all $m$ $T$-like spiders.

Combining the complexities for the general heuristic (3.23) will take $O(mn+m^2)$ time. With the existing greedy algorithm from [2, 3] that has complexity $O(2^{\alpha t} t^2)$, and since $m + n = t$, the complexity remains unchanged. Specifically, looking at the $O(t^2)$ operations in between decompositions,

$$O(t^2 + mn + m^2) = O(t^2) \tag{3.40}$$

Comparing to the method of [4] which requires computing $\alpha_e$ values (3.2) for each of the $p$ decompositions considered to select the best one, the number of operations between each decomposition there is $O(pt^2)$.

The paired heuristics are (3.27) are slightly more costly than the single heuristics, with a complexity of $O(m^2(n + m))$. Each pair of $T$-like spiders can have at most $O(n)$ phase gadget neighbours, and $O(m)$ $T$-like spider neighbours, and so can be processed in $O(n+m)$ time. With the $O(m^2)$ pairs of $T$-like spiders, the complexity is $O(m^2(n+m))$. This overhead cost comes into play since $O(m^2(n+m)) = O(t^3)$ in the worst case, compared to the original $O(t^2)$ complexity. With a higher $T$-count, and the reduction of $\alpha$ due to more efficient cuts, we will see that the paired heuristics can still be effective in practice.

However, in better cases, where $m \ll n \leq t$, we can have $O(t)$ complexity instead. Such cases can be described in the next chapter in Section 4.2 when we see different classes of circuits being affected differently by the overhead cost.

The $k$-connected heuristics (3.30) considers $O(m^k)$ $k$-connected $T$-like spiders, requiring the check of $O(k(m+n))$ edges, for a total of $O(m^k(m+n)k)$ time. In the worst case, this is $O(kt^{k+1})$, showing its impracticality for small $t$.

For the heuristic taking into account the overlap of $T$-like spiders (3.33), the Hopcroft-Karp algorithm has a complexity of $O(mn\sqrt{m+n})$ for each $T$-like spider, for a total of $O(m^2n\sqrt{m+n})$ time.

For the phase gadget heuristic (3.36), to compare two phase gadgets is $O(m)$, for a total of $O(mn)$ for all phase gadgets. Computation for all $m$ $T$-like spiders can be done at the same time, so the complexity is $O(mn)$.

The simple subgraph complement heuristic can be computed in $O(m^2)$ time.

# 4 Results

## Contents

*This chapter presents the classes of circuits used to benchmark our heuristics, which are not novel themselves, but are used to compare the effectiveness of our methods. There are some novel analyses or generalisations of the structures of some circuits, notably the Random IQP and the Modified Hidden Shift circuits. We also discuss the reasons behind effectiveness of heuristics to these classes of circuits.*

*In Section 4.2, we present the results of the benchmarking of our heuristics, with experimentation setup detailed.*

## 4.1 Benchmarking

To determine the effectiveness of our heuristic methods thus far, we need to benchmark them against known methods. In this chapter, we introduce the various benchmarking methods we use to evaluate our heuristics. This is based off of various previous work, particularly those using ZX-calculus [2–5]. We use QuiZX [25], with features added from [3], especially the $|\mathrm{cat}_n\rangle$ decompositions. QuiZX itself was a Rust port of PyZX, originally built in Python [26].

### 4.1.1 Random Clifford+T

Random Clifford+T circuits coming from *exponentiated Pauli unitaries* were used in the benchmark of the BSS decomposition [13], and subsequently continued use in the benchmark of the $|\mathrm{cat}_n\rangle$ decomposition [3]. The operators in this form naturally arise in the literature, including Clifford+T circuit representations [27], as cited in [13]. We introduce their definitions, pertaining to the ZX-calculus.

**Definition 4.1.1** (Unitary)**.** A unitary $U$ is a map of the form

$$-\boxed{U}-\boxed{U^\dagger}- \quad = \quad \text{———} \quad = \quad -\boxed{U^\dagger}-\boxed{U}- \tag{4.1}$$

▲

**Definition 4.1.2** (Pauli exponential [11])**.** A *Pauli exponential* is any unitary of the form

$$U := e^{\pm i\theta P_1 \otimes \cdots \otimes P_n} \tag{4.2}$$

where $P_i$ are Paulis, and $\theta \in \mathbb{R}$ is a phase, and the exponential is defined as

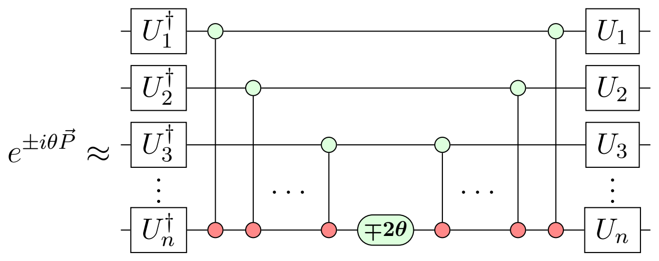
$$e^A := \sum_{k=0}^{\infty} \frac{A^k}{k!} \tag{4.3}$$

for any map $A$, where $A^k$ is

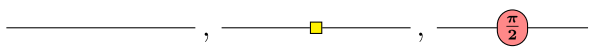
$$\text{——}\boxed{A^k}\text{——} \quad := \quad \underbrace{-\boxed{A}-\boxed{A}-\cdots-\boxed{A}-}_{k} \tag{4.4}$$

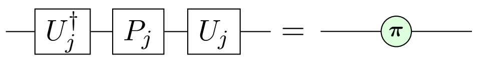
▲

**Proposition 4.1.3** ([11])**.** Let $\vec{P} = P_1 \otimes \cdots \otimes P_n$ be any Pauli string with no trivial entries, so $\forall i, P_i \in \{X, Y, Z\}$. Then the Pauli exponential $e^{i\theta\vec{P}}$ is a ZX diagram where



(4.5)

Here $U_j \in \{$  $\}$ such that



(4.6)

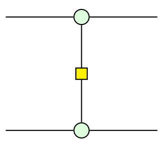
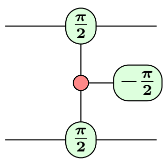
Choosing $\theta = (2k+1)\frac{\pi}{4}$ for $k \in \mathbb{Z}$, we can generate a random Clifford+T circuit with a fixed depth by composing these exponentiated Pauli unitaries. This fixes the $T$-count of the circuit, and can be used to benchmark decompositions.

### 4.1.2 Random IQP

Instantaneous Quantum Polynomial (IQP) circuits were used in the benchmark of [4]. These circuits were chosen as a possible model to demonstrate quantum supremacy [28]. They can likely be built in a quantum computer, while at the same time, they are a class of commuting quantum computations which are shown to be hard to simulate classically, even when adding in the presence of physically motivated constraints, such as sparsity and noise [28]. While it is noted that IQP circuits have been questioned in their ability to demonstrate quantum supremacy [4, 6], we show that the heuristic approach gives performance that match previous work.

First, we introduce the CZ-gate.

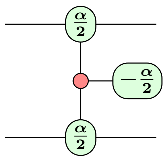
**Definition 4.1.4** (CZ-gate)**.** The CZ-gate is the controlled Z-gate, defined as

$CZ :=$  $=$  (4.7)

▲

More generally, we can define the controlled Z($\alpha$)-gate, where $\alpha = k\frac{\pi}{2}, k \in \mathbb{Z}$, so $CZ(\pi) = CZ$.

**Definition 4.1.5** ($CZ(\alpha)$-gate [7])**.** The controlled Z($\alpha$)-gate is the phase gadget,

$CZ(\alpha) :=$  (4.8)

▲

**Definition 4.1.6** (IQP circuit [6, 29])**.** An IQP circuit is a circuit composed of Hadamard gates, $CZ(\frac{k\pi}{2})$-gates, and $T^m$-gates, where $k, m \in \mathbb{Z}$. Thus, the following Figure 4.1 is a simplified IQP circuit. ▲

Intuitively, the commutative nature of the phase gadgets can be seen directly in the ZX-calculus. We also see the presence of many phase gadgets which only have 2 legs, which lends itself directly to the strength of our heuristic (3.23), considering the number of $|\text{cat}_3\rangle$ states (3.14).

In [6], dense IQP circuits were found to be simulable in $O(\frac{\log^2 n}{n}2^n)$ time, where $n$ is the number of qubits, where dense IQP circuits are those with $O(n^2)$ phase gadgets, and thus $t = O(n^2)$ $T$-count. This is faster than $O(2^n/poly(n))$ for any

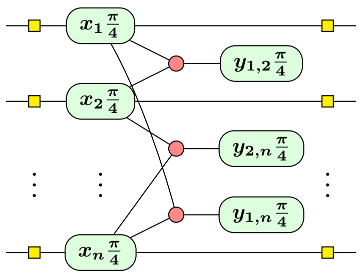


**Figure 4.1:** An IQP circuit as a ZX-diagram, for $x_i, y_{i,j} \in \mathbb{Z}$, as defined in [6]

*poly*$(n)$, compared to $O(n^4 2^{O(n^2)})$ of just using stabiliser decomposition. They concluded that current hardware is probably not enough to demonstrate quantum supremacy using IQP circuits. We note, however, that the heuristic approach can easily give us a better upper bound than than the general one.

**Lemma 4.1.7.** Given a $n$-qubit IQP circuit, the heuristic approach to stabiliser decomposition will give $O(2^n)$ terms.

*Proof.* Note that as stated in [6], we can decompose the IQP circuit into $O(2^n)$ terms just by cutting the $T$-like spiders corresponding to the $n$ qubits. It suffices to show that the heuristic will pick the correct cut decompositions, or result in an equivalent number of terms.

First, note that we can set the algorithm 3.1.1 to skip the ZX-simplify [2] step between each cut. Consider a $T$-like spider connected to a leg of a phase gadget. Note that there are only $n$ of such spiders. We know it will always be connected to $k \geq 1$ phase gadgets (i.e. the $x_i$ $T$-like spiders in Figure 4.1). Also, the $T$-like spiders in the phase gadgets themselves are only connected to a single phase gadget by definition (i.e. the $y_{i,j}$ $T$-like spiders in Figure 4.1). If $k = 1$ for both legs of the phase gadget, it doesn't matter which $T$-like spider we pick, as they would be equivalent cuts. Otherwise, if $k > 1$ for some leg, the heuristic will always pick that $T$-like spider, and since its $\alpha_h \leq 0.2 < 0.25$ is better than any of the $|\text{cat}_n\rangle$ decompositions, it will be picked over them in the greedy algorithm.

If we are left with $T$-like spiders where all are connected to $k = 1$ phase gadgets, then since the only cat state present is the $|\text{cat}_3\rangle$, its decomposition of 2 terms will give an equivalent number of terms as the cut decomposition. Thus, the heuristic will always pick the correct cuts, or use the $|\text{cat}_3\rangle$ decomposition, resulting in an equivalent number of terms. □

While this is not as good as the $O(\frac{\log^2 n}{n} 2^n)$ time of the dense IQP circuits, it gives a much closer upper bound than the general bound of $O(n^4 2^{O(n^2)})$, and is relatively easy to implement in practice. In fact, we can generalise this even further.

**Proposition 4.1.8.** Suppose we have ZX-diagram in reduced gadget form, which has $n$ $T$-like spiders not part of phase gadgets, and $\tilde{O}(n^k)$ [1] phase gadgets, for $k \in \mathbb{R}_+$. Then the stabiliser decomposition can be completed in $O(2^n)$ time.

*Proof.* Simply cut the $n$ $T$-like spiders to get $O(2^n)$ terms. Since the phase gadgets will now be legless, the diagram is easy to simulate as scalars. □

We have that for dense IQP circuits, $k = 2$.

The resulting efficiency of just using cuts is then

$$\alpha = \frac{n}{n + \tilde{O}(n^k)} = \frac{1}{1 + \tilde{O}(n^{k-1})} \tag{4.9}$$

It can be noted that for $k \leq 1$, the cut approach will not really be useful, since $n^{k-1} \leq 1$ so its $\alpha \geq 0.5$ as $n \to \infty$. However, for $k > 1$, we see that $\alpha < 0.5$ as $n \to \infty$ is inversely correlated with $n^{k-1} > 1$, and so it might be useful to use a different metric that is not based on the $T$-count. Instead, as observed in [6], we see that the growth of the number of terms is more like $O(2^{\beta n})$ for some $0 < \beta \leq 1$. For dense IQP circuits, they obtained $\beta \approx 0.93$. For sparse IQP circuits where $k = 1$ (so there were $O(n \log n)$ phase gadgets), they obtained $0.5 < \beta < 0.93$. Compare this to using $\alpha$ where we have the growth being

$$O(2^{\alpha \cdot \tilde{O}(n^k)}) = O\left(2^{\frac{\tilde{O}(n^k)}{\tilde{O}(n^{k-1})}}\right) = O(2^{O(n)}) \tag{4.10}$$

which ends up being equivalent.

**Definition 4.1.9** ($\beta$ efficiency)**.** Suppose we have ZX-diagram in reduced gadget form, which has $n$ $T$-like spiders not part of phase gadgets, and $\tilde{O}(n^k)$ phase gadgets, for $k \in \mathbb{R}_{>1}$. Let $p$ be the number of terms in the stabiliser decomposition $D$. We define the $\beta$ *efficiency* as

$$\beta(D) = \frac{p}{n} \tag{4.11}$$

▲

We note that the cut heuristic as defined in (3.23) works very well due to the presence of many $|\text{cat}_3\rangle$. In the general case where this may not be true, the conceptualisation of different heuristics could be developed to exploit any classes of circuits that have a structure such that its reduced gadget form allows for Proposition 4.1.8 to be applied.

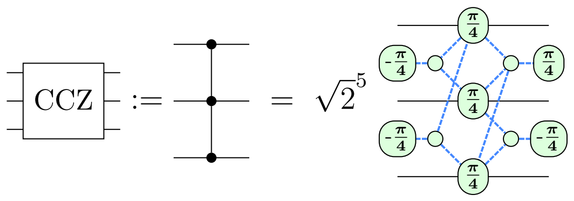
[1]$\tilde{O}(f(n)) = O(f(n) \log^p f(n))$ for some $p$ is a convenient shorthand ignoring polylogarithmic factors

### 4.1.3 Modified Hidden Shift

Hidden shift circuits have been used in various benchmarks since being introduced in [15], starting from the benchmark of the BSS decomposition, and continuing to the benchmark of the $|\text{cat}_n\rangle$ decomposition [2, 3, 19, 30]. However, it was noted that improvements to QuiZX trivialised the decomposition of these circuits, with a conjecture of the class of circuits being simulable in polynomial time [4]. Thus, in [31], they introduced the *modified hidden shift* circuits.

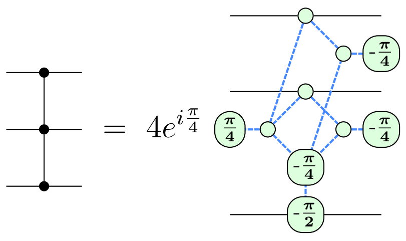
First, we note that the $T$-count of the hidden shift circuits only comes from the $T$-like spiders appearing in CCZ gates in the circuits. We introduce the definition of the CCZ gate, shown as presented in [2].

**Definition 4.1.10** (CCZ gate [2]). The CCZ gate is the controlled-CZ-gate, defined as



(4.12)

It arises from simplifying the 'textbook' presentation of CCZ or Toffoli in terms of CNOT and T gates (see e.g. [10], Section 4.3). The 4-spider version is the following:

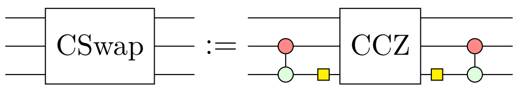


(4.13)

▲

While the structure of the original hidden shift circuits made use of CCZ gates, the modified hidden shift circuits used controlled swap gates instead. The simple addition of the CNOT and Hadamard gates to form the new controlled swap gate made the circuits non-trivial again.

**Definition 4.1.11** (Controlled swap gate). The controlled swap (Friedkin) gate is defined as



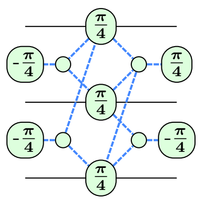
(4.14)

▲

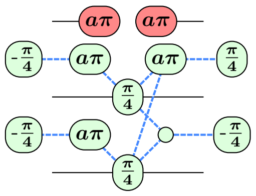
We also have the following useful observation about the CCZ gate.

**Lemma 4.1.12.** The CCZ gate can be decomposed into 2 stabiliser terms.

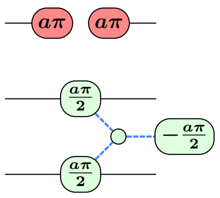
*Proof.*



$$(\boldsymbol{cut})\ (\boldsymbol{f}) \approx \sum_{a\in\{0,1\}} e^{ia\frac{\pi}{4}}$$

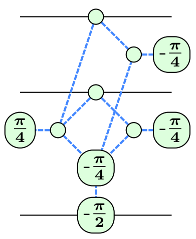


$$(\boldsymbol{\pi})\ (\boldsymbol{GF}) = \sum_{a\in\{0,1\}} e^{ia\frac{\pi}{4}} \tag{4.15}$$

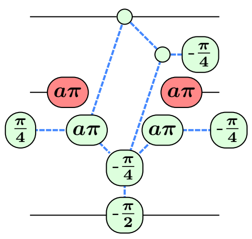


□

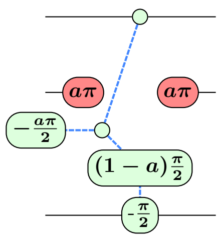
While at first glance, this seems to give a decent $\alpha = 1/7 \approx 0.143$, we notice from (4.13) that CCZ gates can be expressed with just 4 $T$-like spiders. In fact, the 2 terms can be derived from the 4 $T$-spider version.



$$(\boldsymbol{cut})\ (\boldsymbol{f}) \approx \sum_{a\in\{0,1\}}$$



$$(\boldsymbol{f})\ (\boldsymbol{\pi})\ (\boldsymbol{GF}) = \sum_{a\in\{0,1\}} \tag{4.16}$$



It is then fairer to say that this cut decomposition has $\alpha = 0.25$.

The structure of the CCZ gate can be visually inspected to see that 2 and 3-legged phase gadgets will be quite common in the circuits. The effectiveness of the heuristic in exploiting this structure can be seen later in their results in subsection 4.2.3.

### 4.1.4 Random Clifford+T with CCZ

Last but not least, we consider the combination of the random Clifford+T circuits with CCZ gates, as was benchmarked in [31]. We use the similar distribution of gate sets with 5% each of CCZ and T gates, with the rest of distribution uniform over the Clifford gates (Hadamard, CNOT, CZ, S).

Similar to the modified hidden shift circuits, the CCZ gates will contribute to the effectiveness of the heuristic.

## 4.2 Experiments

We now present the results of the benchmarking of our heuristics. For all experiments, we have a timeout of 90 seconds per circuit simulated. All $T$-counts shown are the $T$-count *after the first simplification* using ZX-simplify [2]. For initial $T$-counts before simplification, we will place these in the appendix. We also filter for circuits that have a $T$-count $> 10$ after initial simplification using ZX-simplify [2], since simulating low $T$-count can already be done very quickly. All $\log_2$ values are taken in base 2, so graphs showing a log scale with a best fit line are showing an exponential fit $y = c2^{mx}$. When comparing with the default `quizx` implementation, it uses the greedy approach on $|\text{cat}_n\rangle$ states [3]. We show the results for `single`, corresponding to (3.23), and `single+paired`, corresponding to using both `single` and (3.27). For each circuit, we simulate the scalar from Figure 2.1. Specifically, we randomly plug an output state $\langle\vec{x}|$ where $\vec{x} \in \{0, 1\}^n$ and decompose from there.

We note that the default `quizx` implementation does not in fact make use of the Graph Cuts in their implementation, so disconnected components are not treated independently when decomposing. However, for our comparisons, we implement this by default and all comparisons make use of this. We add the relevant code used in the appendix Section 6.1, including the code additions and edits made in `decompose.rs`.

### 4.2.1 Random Clifford+T

For these circuits, we can choose the number of qubits $q$, and the depth $d$. We stick with the standard minimum and maximum weights of 2 and 4 as in [3]. We take $q = 8, 20, 50, 100$ and $10 \leq d \leq 103$, where we increment $d$ by 3 each time, and take the results of 50 random circuits for each $(q, d)$ pair.

In Figure 4.2, we can see that the $\alpha$ values are generally lower for the heuristic, than the default `quizx` implementation.

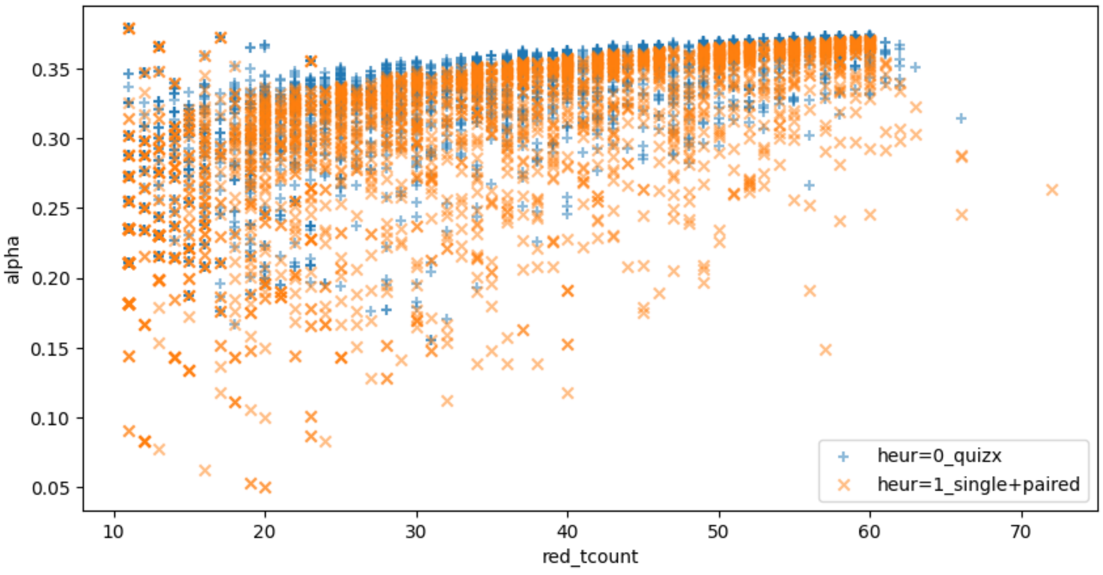


**Figure 4.2:** $\alpha$ values (lower better) for Clifford+T circuits

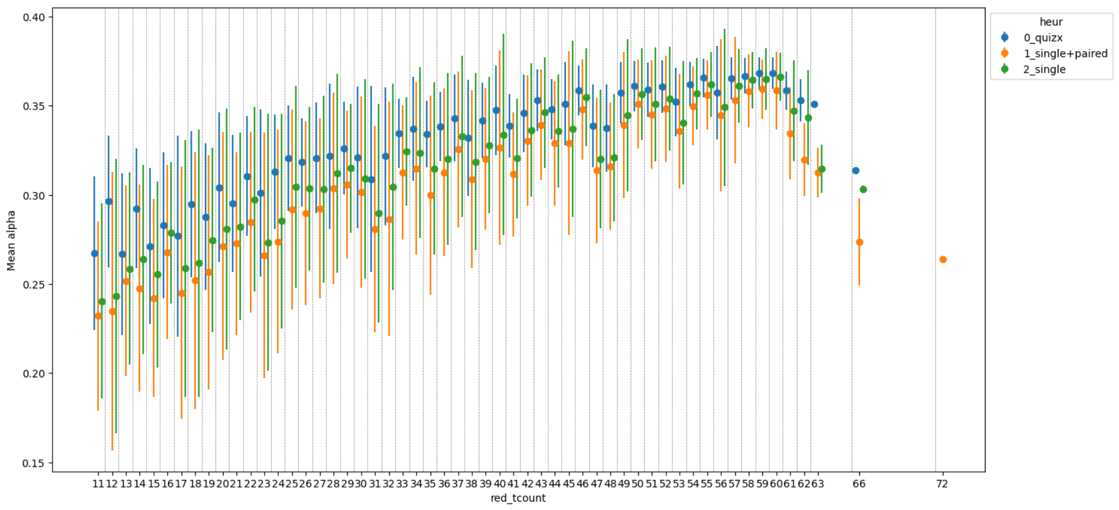


**Figure 4.3:** Mean $\alpha$ values (lower better) with standard deviation bars against different $T$-counts for Clifford+T circuits

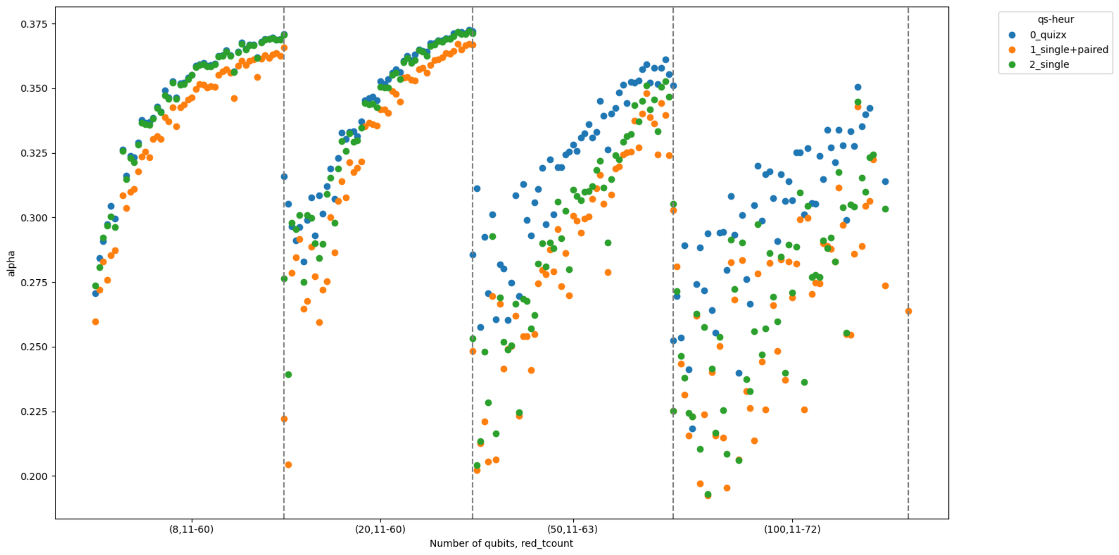


**Figure 4.4:** Mean $\alpha$ values (lower better), separated by number of qubits, for Clifford+T circuits

Figure 4.3 shows the mean $\alpha$ values for each $T$-count, where there are less experiments for higher $T$-counts, especially those with $t > 63$, as most of these correspond to $q = 100$, which have a larger variation in the reduced $T$-count after initial simplification. This is much clearer to be seen in Figure 4.4, which separates the mean $\alpha$ values by number of qubits.

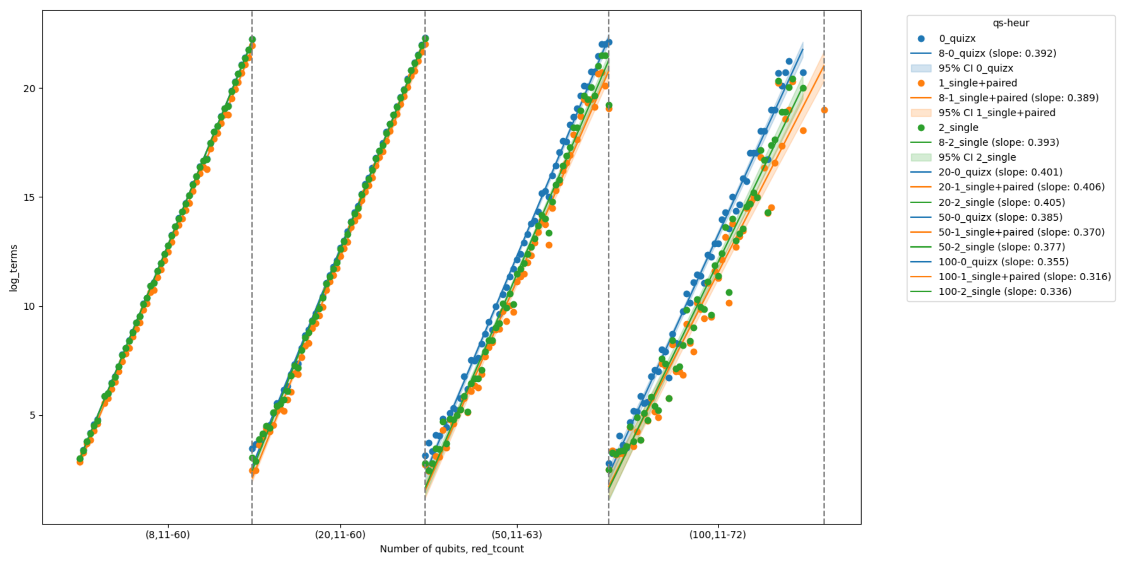


**Figure 4.5:** Mean $\log_2$ terms (lower better) by $T$-counts, separated by number of qubits, for Clifford+T circuits

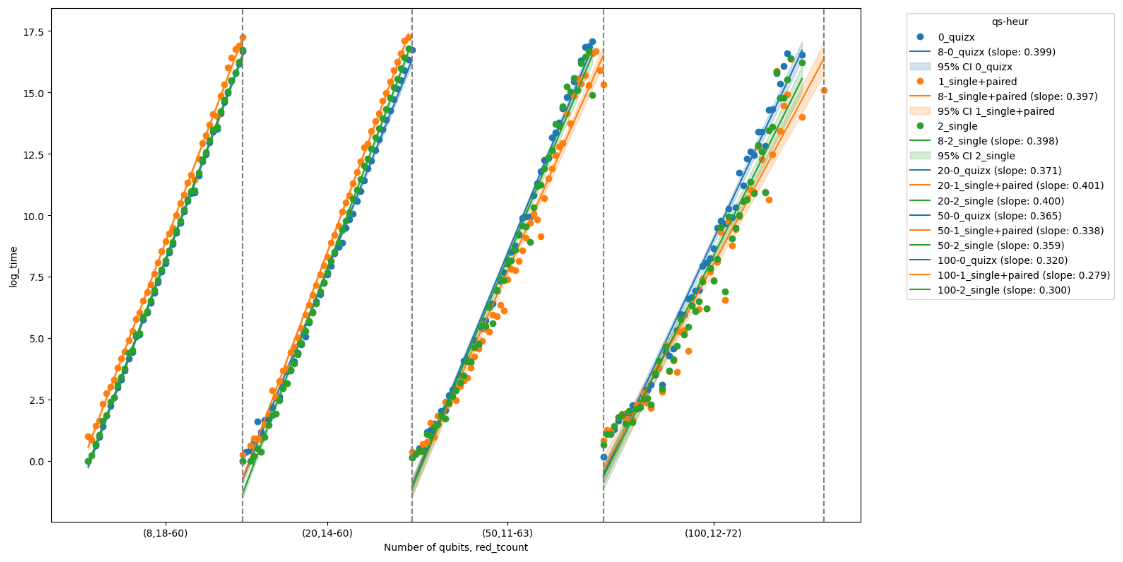


**Figure 4.6:** Mean $\log_2$ time (lower better) by $T$-counts, separated by number of qubits, for Clifford+T circuits

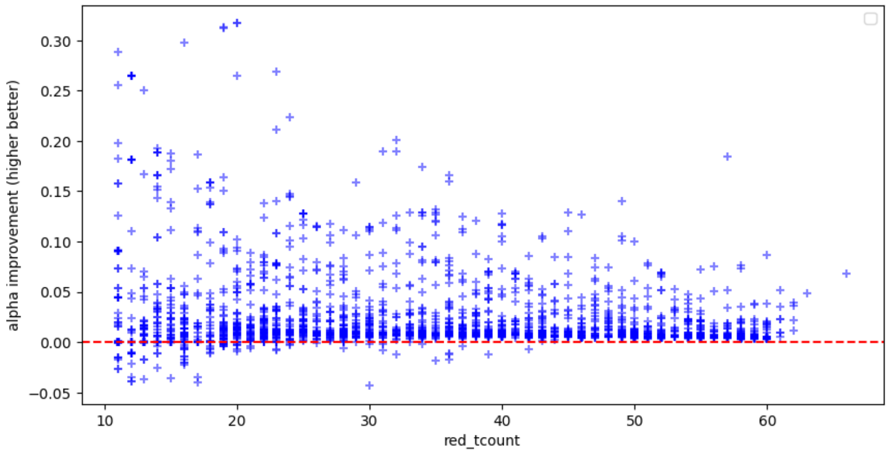


**Figure 4.7:** Improvement in $\alpha$ values (`single+paired` vs. `quizx`) for Clifford+T circuits

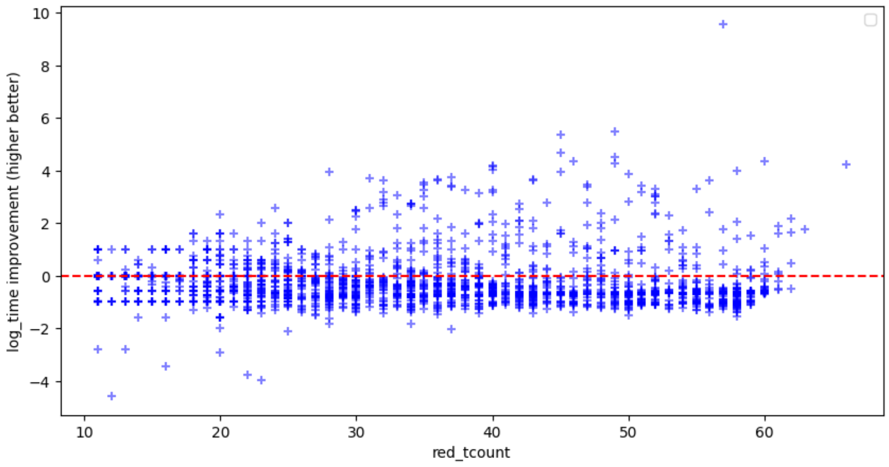


**Figure 4.8:** Improvement in $\log_2$ time for Clifford+T circuits

We find that although almost all $\alpha$ values show an improvement in Figure 4.7, we find a significant proportion of the same circuits giving worse values for the $\log_2$ time values in Figure 4.8. Since the time complexity of the decomposition is $O(t^2 2^{\alpha t})$, its slope, in theory, should be close to value of $\alpha$. The worse values in time could be attributed to the overhead caused by computing the heuristic between each step of the decomposition. We see in the $\log_2$ terms and time graphs in Figure 4.5, 4.6, that the slopes are essentially similar for $q \leq 20$, and only diverge for $q \geq 50$. On the other hand, Figure 4.4 shows that $\alpha_h$ is generally lower no matter the value of $q$. We show in Lemma 4.2.1 that as the $T$-count $t$ increases, this overhead becomes less significant if our efficiency $\alpha_h < \alpha$ is decisively lowered.

**Lemma 4.2.1.** Suppose we have a circuit in reduced gadget form with a $T$-count $t$. Given $\alpha_h < \alpha$ using the heuristics (3.23, 3.27) compared against the default `quizx` implementation, the log time improvement is $(\alpha - \alpha_h)t - \log t$. The overhead of computing the heuristic can be given by $\log t$.

*Proof.* Without the heuristic, we have that the time taken is $O(t^2 2^{\alpha t})$. With the heuristic, we have that the time taken is $O(t^3 2^{\alpha_h t})$. We assume that the constant factors are comparable. Taking the log of the ratio of the 2 terms, the log time improvement is

$$\begin{aligned} \log\left(\frac{t^2 2^{\alpha t}}{t^3 2^{\alpha_h t}}\right) &= -\log t + \log 2^{\alpha t} - \log 2^{\alpha_h t} \\ &= -\log t + \alpha t - \alpha_h t \\ &= (\alpha - \alpha_h)t - \log t \end{aligned} \tag{4.17}$$

□

We can see that the overhead can be significant for low $t$ where $(\alpha - \alpha_h)t < \log t$, but since $\log t = o((\alpha - \alpha_h)t)^2$ for any $\alpha > \alpha_h$, the overhead becomes less significant as $t$ increases. For the results shown here, we are limited to showing low $T$-counts as the time taken to run experiments is not feasible for higher $T$-counts. In Figure 4.9, 4.10, we see that the number of completed simulations before timeout is lower for the heuristic for $q \leq 50$, but $q = 100$ has a higher number of completed simulations. We expect that if we run this experiment with a much longer timeout, then the heuristic will show its superiority in terms of time to match that of its improved $\alpha$. We can quantify the expected $T$-counts for which this occurs by solving for $(\alpha - \alpha_h)t > \log t$, shown in the appendix Section 6.2.

[2] using little-o notation

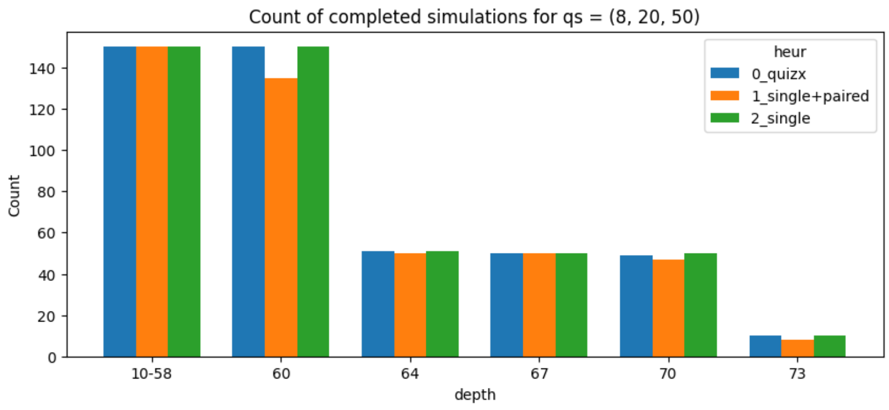


**Figure 4.9:** Number of completed simulations before timeout for $q = 8, 20, 50$ for Clifford+T circuits

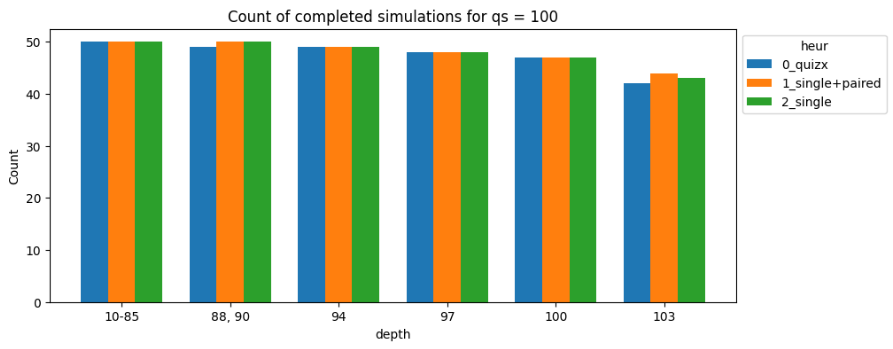


**Figure 4.10:** Number of completed simulations before timeout for $q = 100$ for Clifford+T circuits

### 4.2.2 Random IQP

For these circuits, we can choose the number of qubits $q$. We take $10 \leq q \leq 27$ and the results of 100 random circuits for each $q$. Random IQP circuits will be dense, with the $T$-count $t = O(q^2)$.

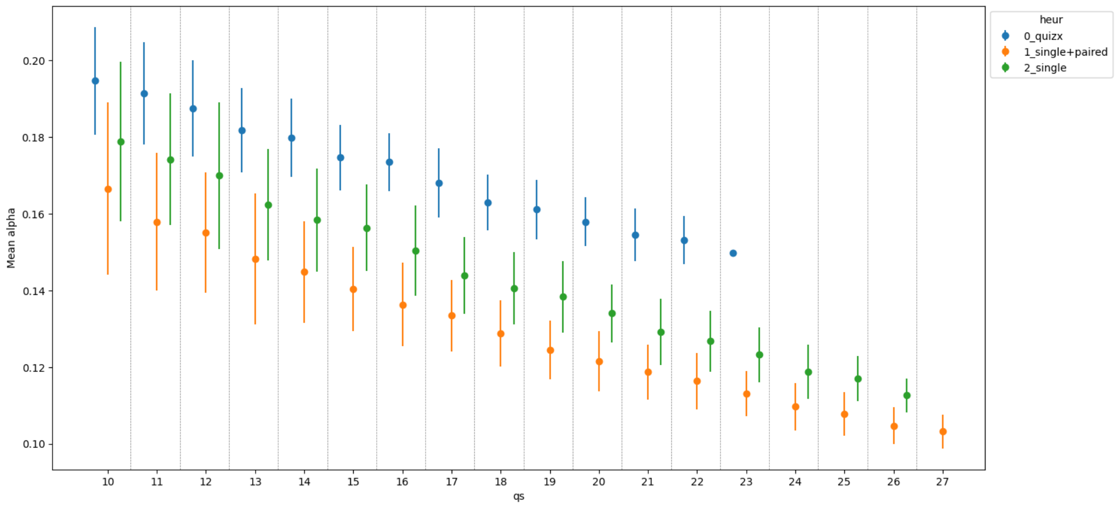


**Figure 4.11:** Mean $\alpha$ values (lower better) by number of qubits for IQP circuits

Here in Figure 4.11 we can see that there is a clear inverse relationship between the number of qubits and the $\alpha$ values. Just by observing the trend in the data points, there is a strong likelihood that set of data obeys the relationship $\alpha = c/q$ for some $c$ (since $\alpha > 0$). As such, the $\beta$ values as defined in Def. 4.1.9 are more useful for these circuits, to better compare the various decomposition methods without being directly correlated to the number of qubits or $T$-count. These are shown below in Figure 4.12.

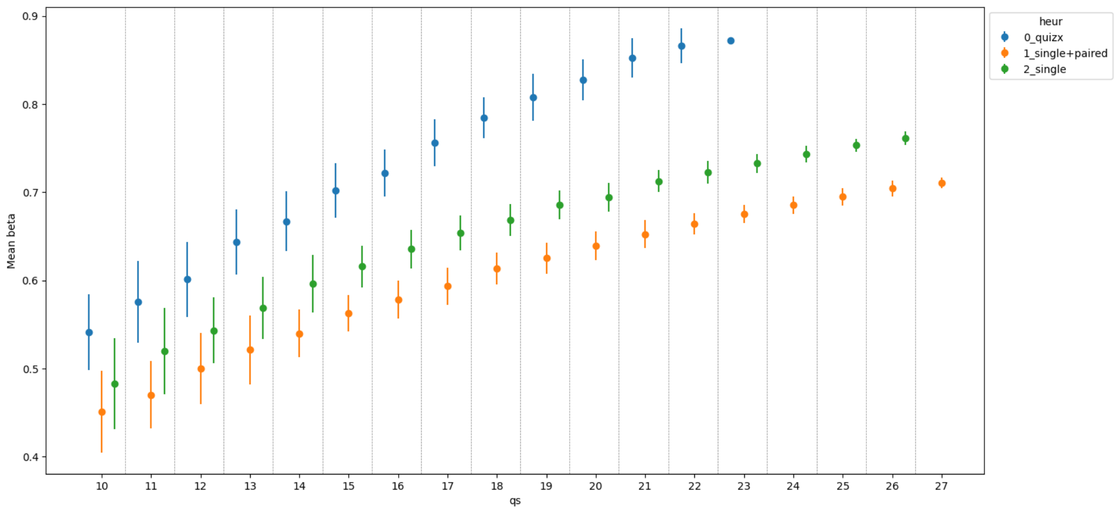


**Figure 4.12:** Mean $\beta$ values (lower better) by number of qubits for IQP circuits

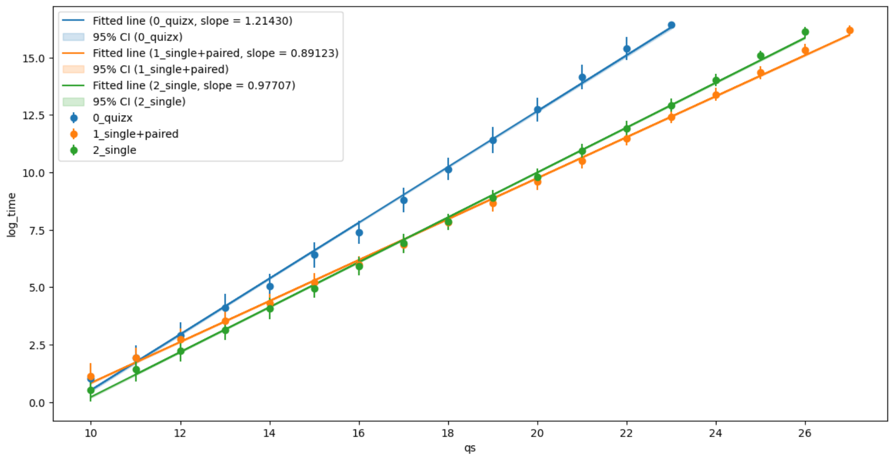


**Figure 4.13:** Mean $\log_2$ time (lower better) by number of qubits for IQP circuits

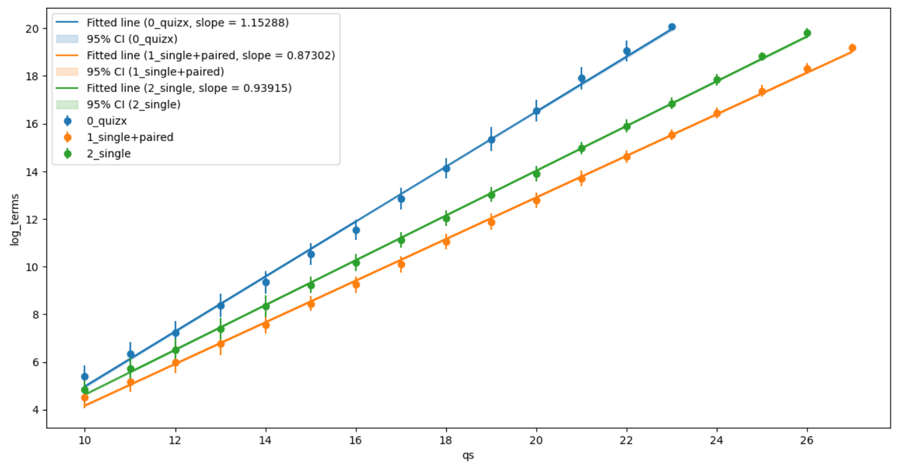


**Figure 4.14:** Mean $\log_2$ terms (lower better) by number of qubits for IQP circuits

We find that in Figure 4.13, 4.14, that the slopes have similar results to that of [6]. In fact, considering that the circuits are random with each qubit sharing a phase gate with $O(n)$ other qubits, we have that our circuits are dense. We can then compare with the value of $\beta \approx 0.93$ obtained in [6], and see that the slope of $\beta \approx 0.891$ is an improvement. It can be noted that comparing using time taken may be inconsistent across different machines, so we show the log terms as a better alternative for comparison for any future work, where $\beta \approx 0.873$ for `single+paired` and $\beta \approx 0.939$ for `single`.

The improvement comparison shown as follows in Figure 4.15, 4.16 is with only using the single cut heuristic, since that is the state of the art for IQP circuits [4, 6].

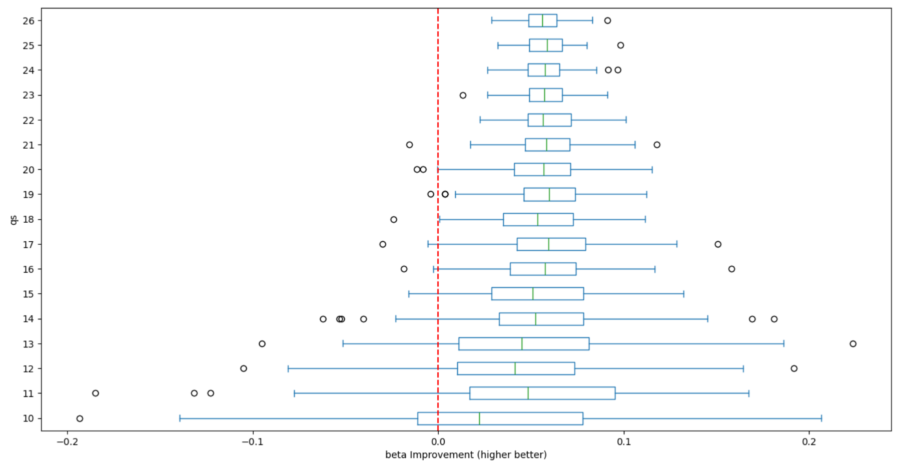


**Figure 4.15:** Improvement in $\beta$ values (`single+paired` vs. `single`) for IQP circuits

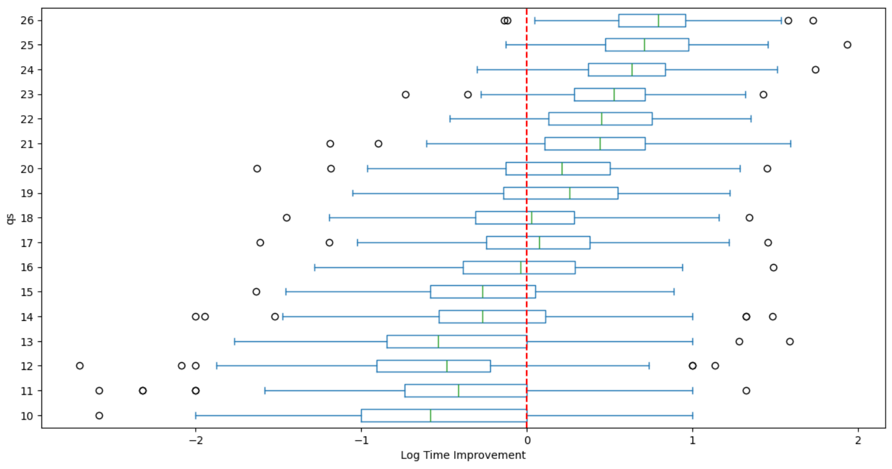


**Figure 4.16:** Improvement in $\log_2$ time (`single+paired` vs. `single`) for IQP circuits

The number of completed simulations before timeout in Figure 4.17, also shows the heuristic `single+paired` is able to simulate up to $q = 27$, whereas the default `quizx` implementation is only able to simulate up to $q = 22$, and `single` up to $q = 26$.

We see a clear improvement in using the `single+paired` heuristic in IQP circuits, with its performance even matching the theoretical state-of-the-art results of [6]. We have shown that for these class of circuits, the $\beta$ efficiency proves to be a useful metric when comparing any future improvements in decomposition methods.

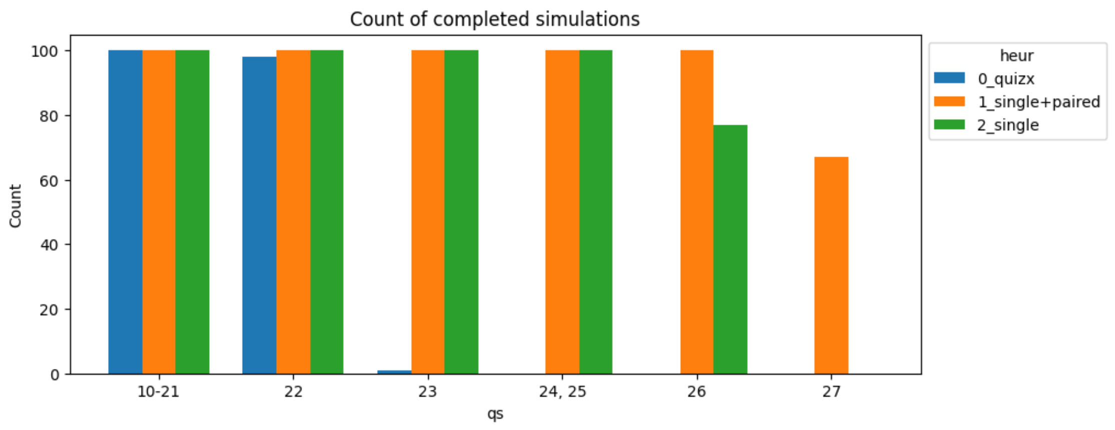


**Figure 4.17:** Number of completed simulations before timeout for IQP circuits

### 4.2.3 Modified Hidden Shift

For these circuits, we can choose the number of qubits $q$, and the number of CCZ gates $n$. We take $q = 20, 50, 100$ and $2 \leq n \leq 60$ at steps of 10 for $n \geq 10$. For each $(q, n)$ pair, we take the results of 100 random circuits.

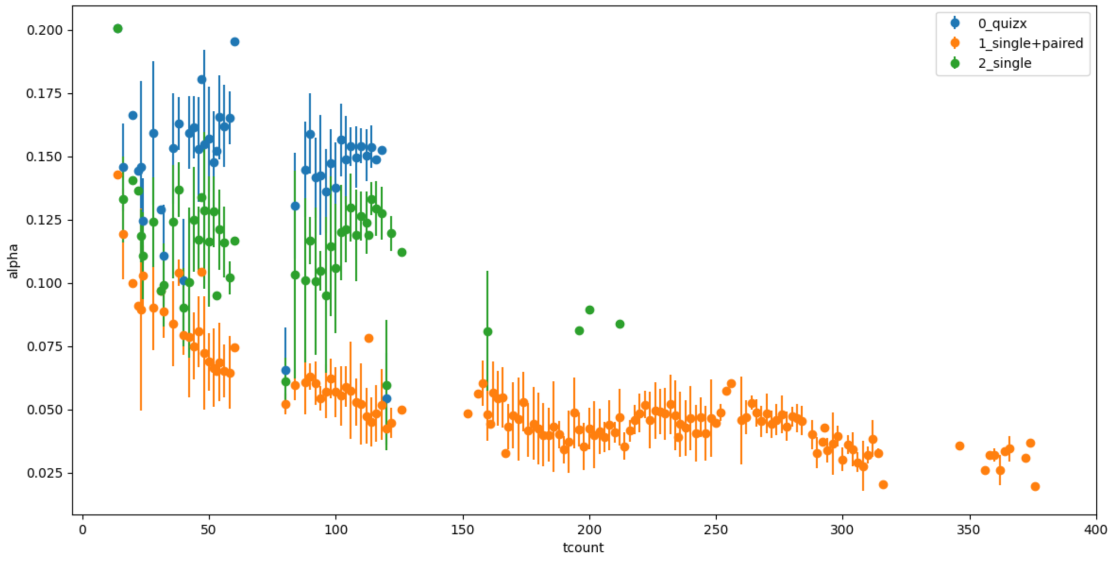


**Figure 4.18:** Mean $\alpha$ values (lower better) by $T$-count for modified hidden shift circuits

Across the whole range of $T$-counts after initial simplification, we see in Figure 4.18 that the heuristic is able to give lower $\alpha$ values, compared to both the `quizx` and `single` implementations. Because of the steps of 10 for the CCZ gates $n$, we see that there are gaps in Figure 4.18, 4.19. Nevertheless, the trend is more important to see.

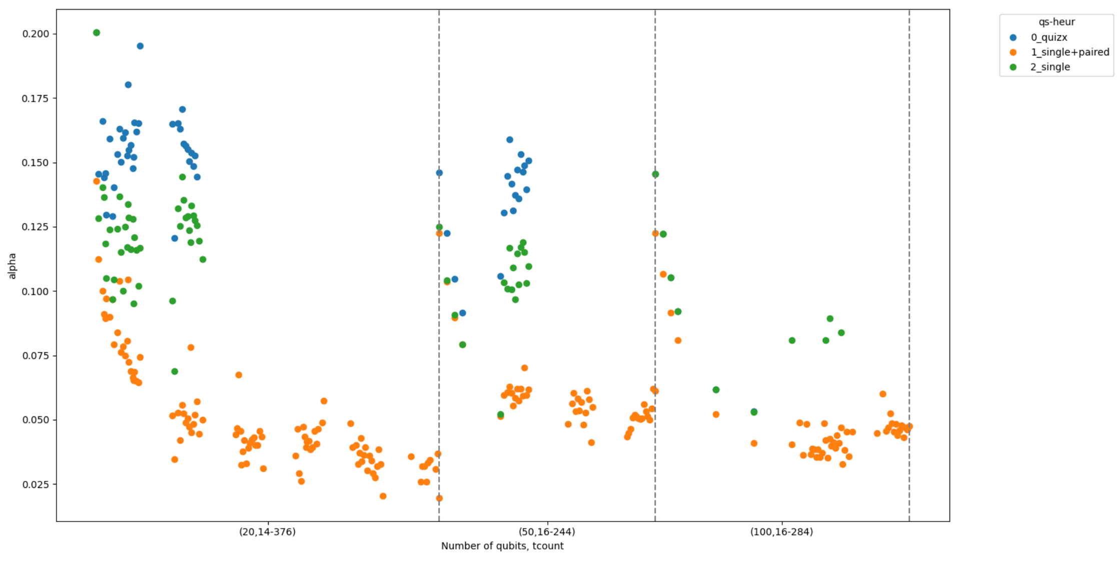


**Figure 4.19:** Mean $\alpha$ values (lower better) by $T$-count, separated by number of qubits, of modified hidden shift circuits

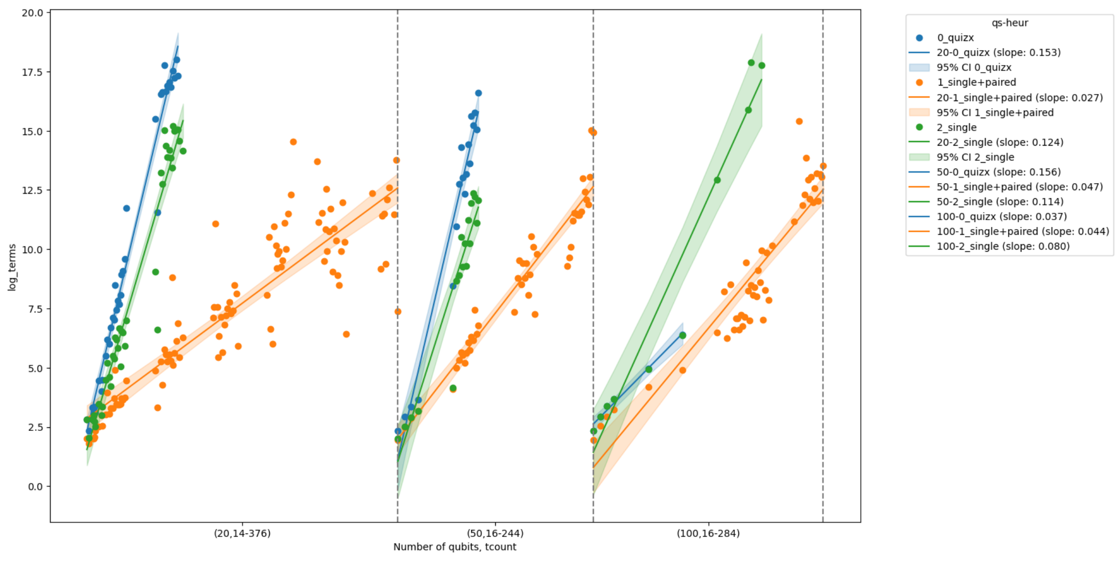


**Figure 4.20:** Mean $\log_2$ terms (lower better), separated by number of qubits, of modified hidden shift circuits

We see great improvements for $\log_2$ terms and time graphs in Figure 4.20, 4.21. For $q = 20$, the slope in the $\log_2$ terms graph gives $\alpha \approx 0.027$ for the heuristic, compared to $\alpha \approx 0.153, 0.124$ for `quizx` and `single` respectively. For $q = 50$, the slope gives $\alpha \approx 0.047$ for the heuristic, compared to $\alpha \approx 0.156, 0.114$ for `quizx` and `single` respectively.

Note that for $q = 100$, there is not enough data to show a best fit line for `quizx` and `single` since most of the simulations could not complete before the

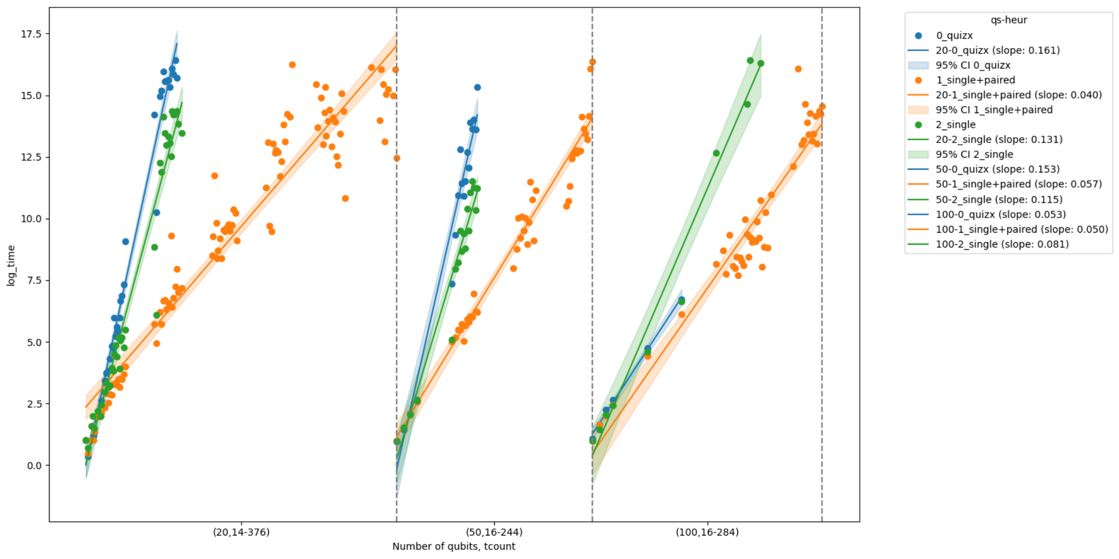


**Figure 4.21:** Mean $\log_2$ time (lower better), separated by number of qubits, of modified hidden shift circuits

timeout. As such, it is difficult to draw conclusive slope comparison here, though it does show the superiority of the heuristic, as we see in the count of completed simulations in Figure 4.24.

We also see clear improvement in both $\alpha$ and $\log_2$ time values in Figure 4.22, 4.23.

Although it is not shown here, we also found that for $q = 6$, the slope of the best fit line for `single+paired` is $\approx 0$, meaning that the heuristic is able to find the optimal decomposition with a constant number of terms with respect to the $T$-count, while `quizx` and `single` were not able to do so. See the appendix Section 6.2 for these results.

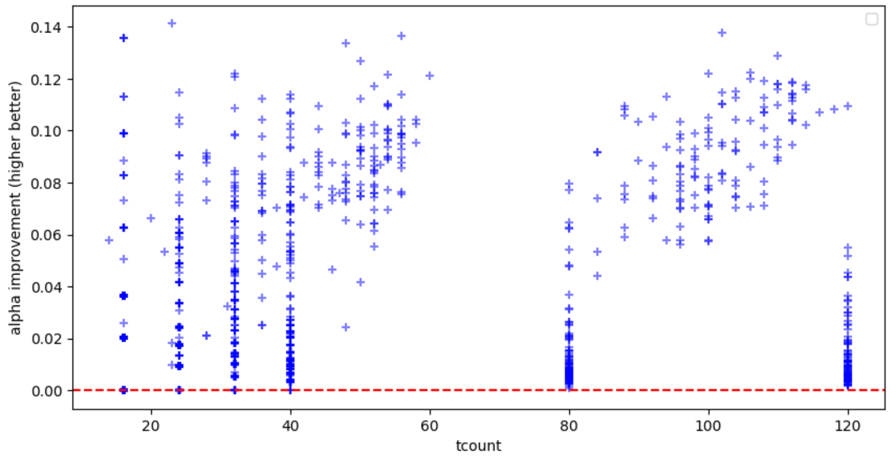


**Figure 4.22:** Improvement in $\alpha$ (`single+paired` vs. `quizx`) for modified hidden shift circuits

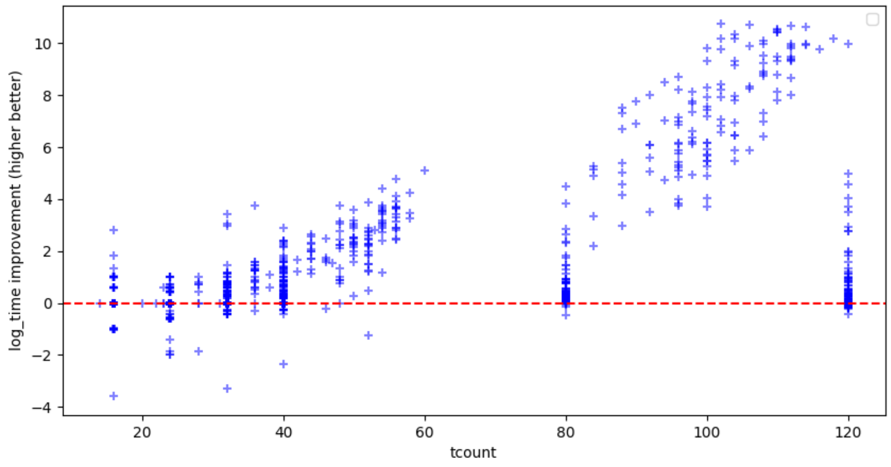


**Figure 4.23:** Improvement in $\log_2$ time (`single+paired` vs. `quizx`) for modified hidden shift circuits

We have a signficiant improvement in using the `single+paired` heuristic in both $\alpha$ and log time for the modified hidden shift circuits, with a much higher possible $CCZ$ depth simulated at $n = 60$ compared to $n = 40, 30$ for the `quizx` and `single` implementations respectively.

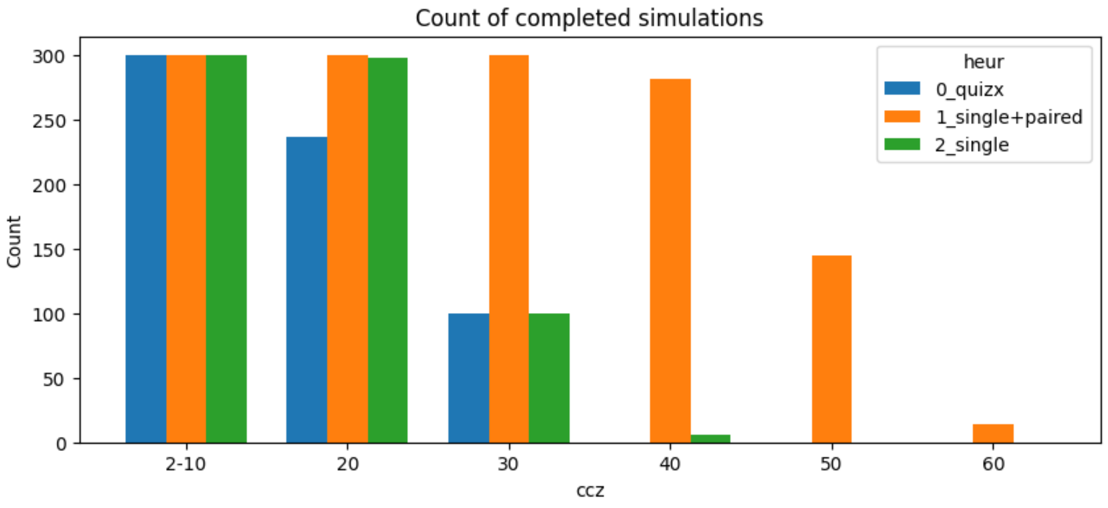


**Figure 4.24:** Number of completed simulations before timeout for modified hidden shift circuits

### 4.2.4 Random Clifford+T with CCZ

For these circuits, we can choose the number of qubits $q$, and the depth $d$. We take $q = 8, 19, 20, 50$ and $100 \leq d \leq 500$, where we increment $d$ by 30 each time (and include the multiples of 100). For $q = 50$, we take until $100 \leq d \leq 800$. For $q = 100$, we take $1050 \leq d \leq 1400$, since the structures of the circuits cause the comparable $T$-counts to be at a higher depth, and lower depths generally having $T$-counts of 0 after initial simplification. We take the results of 100 random circuits for each $(q, d)$ pair.

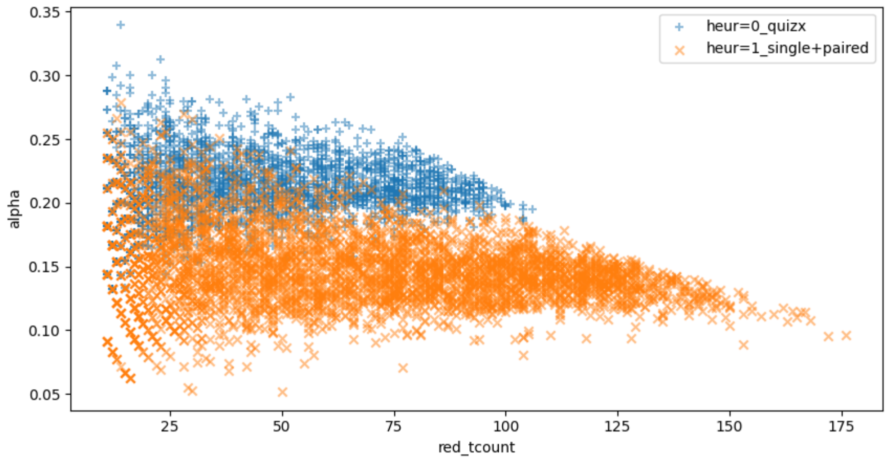


**Figure 4.25:** $\alpha$ values (lower better) for Clifford+T circuits with CCZ

Both Figure 4.25, 4.26 show that the heuristic is able to give lower $\alpha$ values than both the `quizx` and `single` implementations.

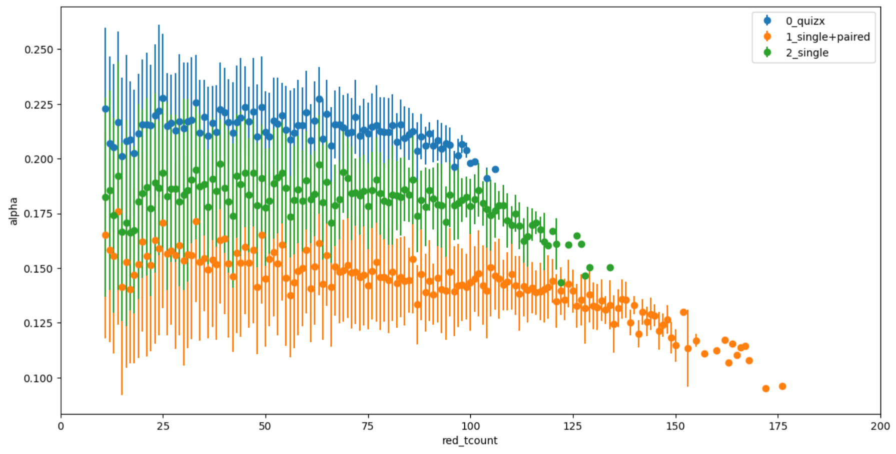


**Figure 4.26:** Mean $\alpha$ values (lower better) by $T$-count for Clifford+T circuits with CCZ

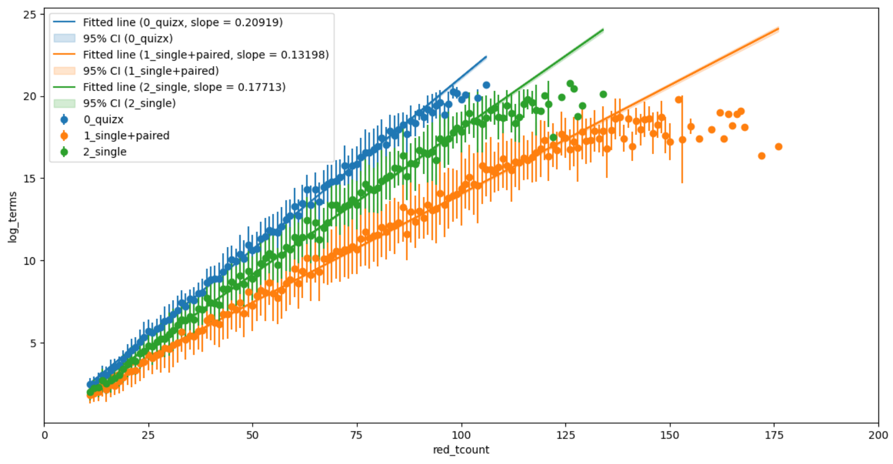


**Figure 4.27:** Mean $\log_2$ terms (lower better) by $T$-count for Clifford+T circuits with CCZ

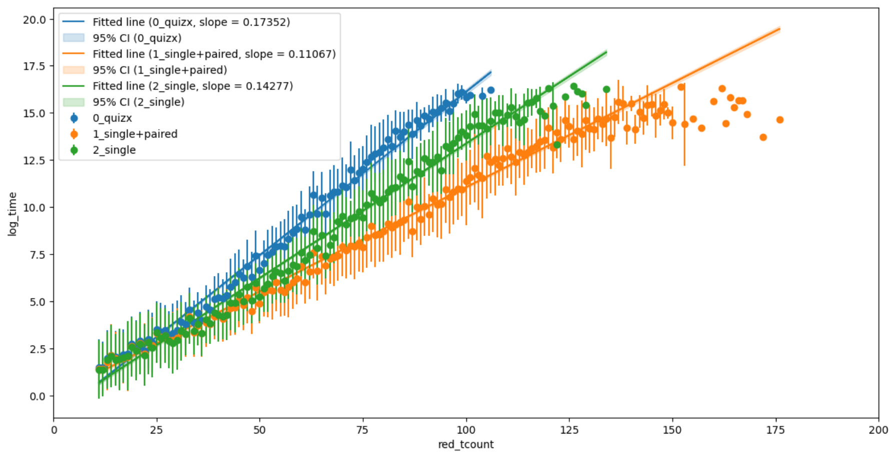


**Figure 4.28:** Mean $\log_2$ time (lower better) by $T$-count for Clifford+T circuits with CCZ

The $\log_2$ terms and time graphs in Figure 4.27, 4.28, show lower values and lower slopes obtaining $\alpha \approx 0.132$ from the $\log_2$ terms graph, compared to $\alpha \approx 0.209, 0.177$ for the `quizx` and `single` implementations respectively.

Across the board, we see a clear dominance of the `single+paired` heuristic in the Clifford+T circuits with CCZ gates. Both $\alpha$ and $\log_2$ time show a significant improvement in Figure 4.29, 4.30, with the number of completed simulations before timeout being significantly higher in Figure 4.31, 4.32, 4.33.

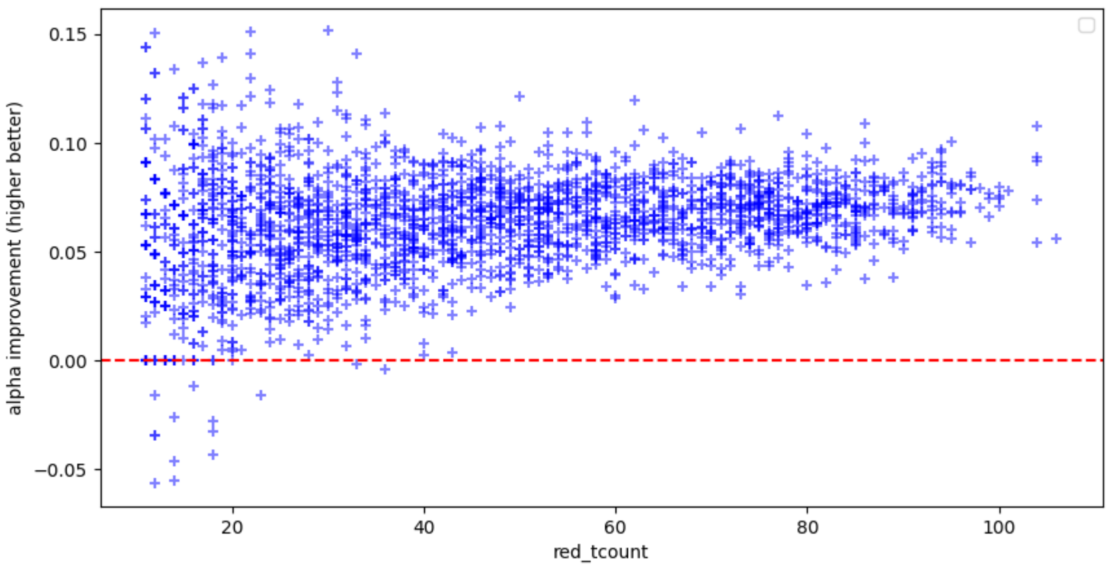


**Figure 4.29:** Improvement in $\alpha$ values (`single+paired` vs. `quizx`) for Clifford+T circuits with CCZ

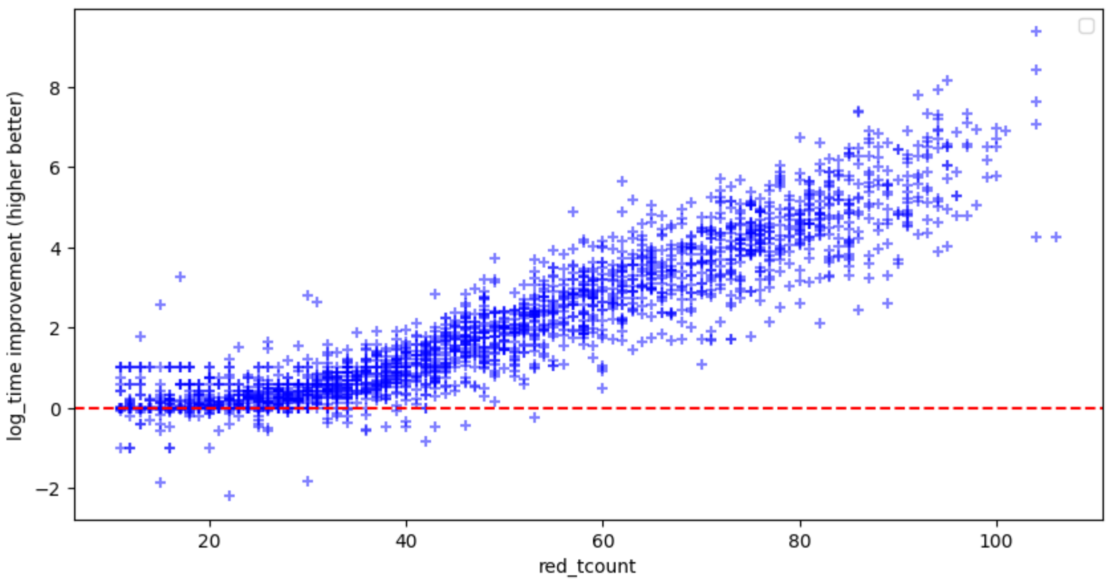


**Figure 4.30:** Improvement in $\log_2$ time (`single+paired` vs. `quizx`) for Clifford+T circuits with CCZ

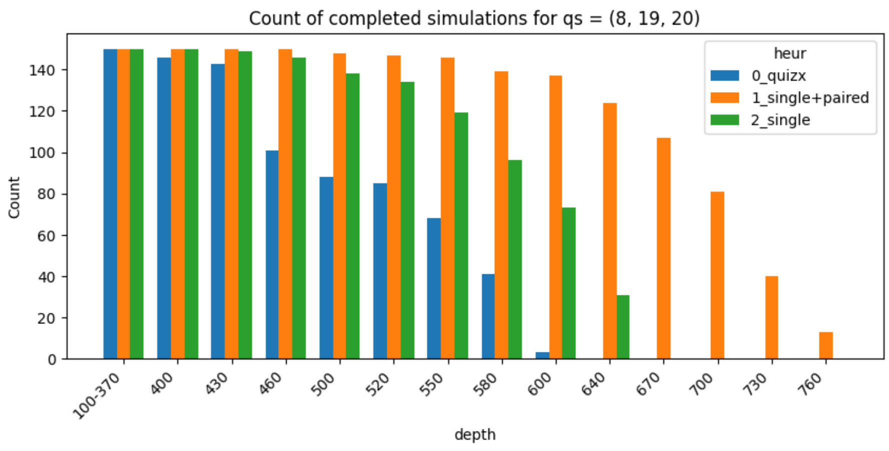


**Figure 4.31:** Number of completed simulations before timeout for $q = 8, 19, 20$ for Clifford+T circuits with CCZ

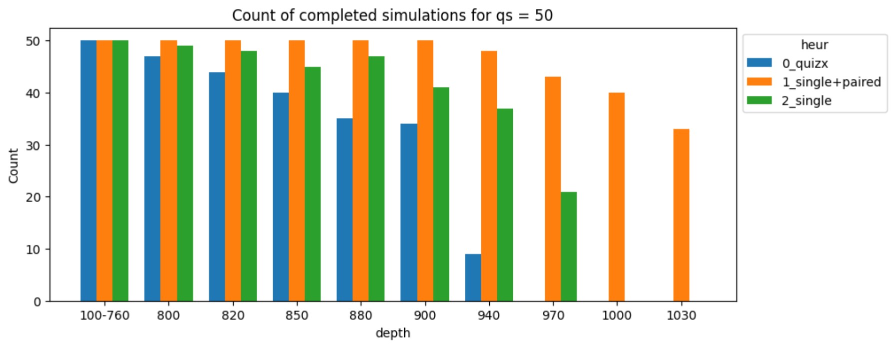


**Figure 4.32:** Number of completed simulations before timeout for $q = 50$ for Clifford+T circuits with CCZ

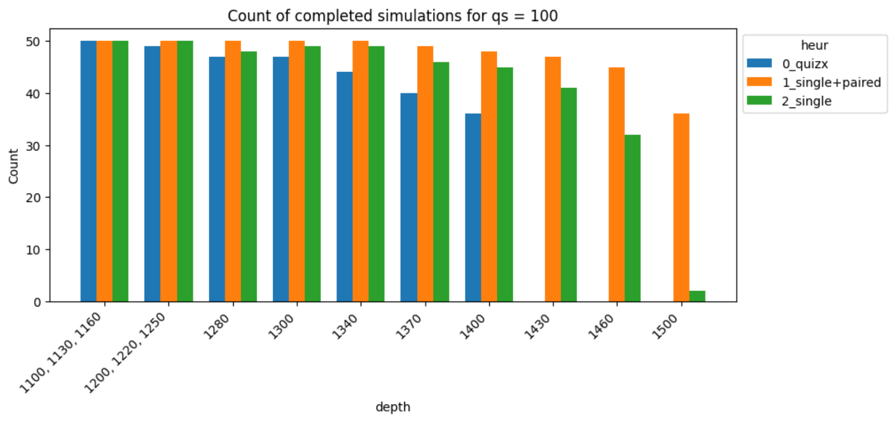


**Figure 4.33:** Number of completed simulations before timeout for $q = 100$ for Clifford+T circuits with CCZ

# 5
# Conclusion

In this thesis, we started in Chapter 1 and 2 by exploring the existing approaches to the problem of classical simulation with the ZX-Calculus. We have seen that the existing stabiliser decomposition algorithms are powerful tools, as demonstrated in [3]. We also saw that heuristic approaches can be used to improve the performance of these algorithms, as demonstrated in [4, 5]. However, as was challenged in [5], and alluded to in [4], obtaining lower $\alpha$ decompositions need not be just by looking at the immediate $T$-count reduction from a decomposition, but also by exploiting the structure of the ZX-diagrams after carrying out the decomposition. This involves looking at how the $T$-like spiders can fuse after using the (***cut***) decomposition, based on certain local patterns.

Thus, in Chapter 3 we formalised the heuristic method and applied the formalism to the existing approaches. We also developed new heuristics that take advantage of structure made available in the ZX-simplify [2] algorithm of [12]. We presented the results of these heuristics in Chapter 4 and showed that the `paired+single` heuristic can greatly improve the performance of the existing stabiliser decomposition algorithms.

We also explored various classes of circuits and how some of their structures lend very well to the heuristics developed. We managed to match the performance of [6] for IQP circuits, and showed a clear improvement for circuits with a high number of CCZ gates. For modified hidden shift circuits, we obtained an improvement of 3 times the $T$-count that could be simulated within the time limit.

We showed that $\alpha$ is generally reduced for all classes of circuits tested, though this does not necessarily translate to a reduction in time. This happens with random Clifford+T circuits coming from Pauli exponentials, where overhead of computing the heuristics outweighed the saved terms for the range of $T$-counts tested. However, we have also shown that this overhead is insignificant for circuits with a high $T$-count, though this remains to be tested for larger circuits and more compute power to match.

We therefore conclude that the heuristic method to stabiliser decomposition is not only limited to that of the CNOT sandwich [5], or the trivial (***cut***) decomposition on 2-legged phase gadgets [4], but can also be extended to a wider range of ZX-diagram structures. In this thesis, this concentrated on the structure created by the reduced gadget form, as well as the (***P***)ivot rewrite rule, but potentially more of such structures could be found.

## 5.1 Future Work

As mentioned in Section 3.10, the Subgraph Complement cut is a potential area for future work, perhaps making use of the future graph partioning work in [21].

More compute power could also test the potential of the $k$-connected Heuristics on larger circuits, and also to determine the size of the circuit for which the overhead of the heuristics is outweighed by the saved terms.

Further analysis could also be done on Complexity Analysis of computing the heuristics, and whether there could also be a heuristic to not use or only partially use ZX-simplify [2] between each step of a decomposition. This could allow the overhead of using heuristics to be minimised, while still providing the $\alpha$-reduction benefits. Particularly, there are few ways this could be done:

- Coming up with a heuristic based version of ZX-simplify [2] that knows when *not* to simplify certain vertices to aid the (***cut***) decomposition.
- Parallelising the heuristic computation for each vertex in the ZX-diagram to avoid the $poly(n)$ time overhead.
- Developing much cheaper meta-heuristics that can decide when to use the heuristics.
- Bounding how often the heuristics are used or computed in the overall decomposition.

Another direction for heuristics is also to also have them for the $|\text{cat}_n\rangle$ decompositions. These would be more complex due to different resulting patterns in the resulting terms of a decomposition, but could be very beneficial. The ideal heuristic would be able to give the best decomposition not just using (***cut***), but also from any known decomposition.

# 6 Appendix

## 6.1 Rust Code

### 6.1.1 Split Components

**Listing 6.1:** Decompose with split components

```rust
pub fn decomp_split_comps(&mut self, g: &G, g_comps: &Vec<G>, depth:
    usize) -> &mut Self {
    let mut nterms_comps = 0;
    let mut scalar_comps = g.scalar().clone();
    for h in g_comps {
        let mut d = Decomposer::new(h);
        d.use_cats(self.use_cats);
        d.split_comps(self.split_comps);
        d.use_heur(self.use_heur);
        d.with_full_simp();

        let depth = if depth > self.max_depth {
            depth - self.max_depth
        } else {
            0
        };

        let d = d.decomp_parallel(depth);

        nterms_comps += d.nterms;
        scalar_comps *= d.scalar;
    }
    self.nterms += nterms_comps;
    self.scalar = &self.scalar + scalar_comps;

    self
}
```

### 6.1.2 Heuristic Decomposition

**Listing 6.2:** Algorithm code in `decomp_top` method of `decompose.rs`

```
if self.use_cats {
    let cat_nodes = Decomposer::cat_ts(&g);
    let nts = cat_nodes.iter().fold(0, |acc, &x| {
        if g.phase(x).denom() == &4 {
            acc + 1
        } else {
            acc
        }
    });

    if self.use_heur {
        let (vs, eff_a) = Decomposer::cut_v(&g);
        let vs_pg = vec![];
        let (vs_pair, eff_a_pair) = if self.use_paired_heur
{
            Decomposer::cut_v_pair(&g)
        } else {
            (vec![], -1.0)
        };

        let mut should_cut_v = eff_a > 0.0;
        let mut should_cut_v_pair = eff_a_pair > 0.0;
        let should_cut_pg = !vs_pg.is_empty();
        let mut bet_eff_a = f64::MAX;
        let mut bet_vs = vec![];

        if should_cut_v && eff_a < bet_eff_a {
            bet_eff_a = eff_a;
            bet_vs = vs;
        }
        if should_cut_v_pair && eff_a_pair < bet_eff_a {
            bet_eff_a = eff_a_pair;
            bet_vs = vs_pair;
            should_cut_v = false;
        }
        if should_cut_pg && 0.25 < bet_eff_a {
            bet_eff_a = 0.25;
            bet_vs = vs_pg;
            should_cut_v = false;
            should_cut_v_pair = false;
        }
```

```
            if bet_eff_a > 0.0
                && ((nts == 4 && bet_eff_a < 0.25)
                    || (nts == 6 && bet_eff_a < 0.264)
                    || (nts == 5 && bet_eff_a < 0.316)
                    || (nts == 3 && bet_eff_a < 0.333)
                    || ((nts > 6 || nts < 3) && bet_eff_a < 0.4)
)
            {
                if should_cut_v {
                    return self.push_decomp(
                        &[Decomposer::replace_t0, Decomposer::
replace_t1],
                        depth + 1,
                        &g,
                        &bet_vs,
                    );
                } else if should_cut_v_pair {
                    return self.push_decomp(
                        &[Decomposer::replace_tpair0, Decomposer
::replace_tpair1],
                        depth + 1,
                        &g,
                        &bet_vs,
                    );
                } else if should_cut_pg {
                    return self.push_decomp(
                        &[Decomposer::cut_pg_0, Decomposer::
cut_pg_0],
                        depth + 1,
                        &g,
                        &bet_vs,
                    );
                }
            }
        }

        if !cat_nodes.is_empty() {
            // println!("using cat!");
            return self.push_cat_decomp(depth + 1, &g, &
cat_nodes);
        }

        let ts = Decomposer::first_ts(&g);
        if ts.len() >= 5 {
            return self.push_magic5_from_cat_decomp(depth + 1,
&g, &ts[..5]);
            }
        }
        let ts = if self.random_t {
            Decomposer::random_ts(&g, &mut thread_rng())
        } else {
            Decomposer::first_ts(&g)
        };
        self.decomp_ts(depth, g, &ts);
        self
    }
```

**Listing 6.3:** Implemented heuristic code in `decompose.rs`

```
// Cut paired v
pub fn cut_v_pair(g: &G) -> (Vec<V>, f64) {

    let mut best_n = 0usize;
    let mut best_vs = vec![];
    let mut eff_a = -1.0;
    for v in g.vertices() {
        if g.phase(v).denom() != &4 {
            continue;
        }
        let v_neigh0 = g
            .neighbor_vec(v)
            .iter()
            .cloned()
            .filter(|&w| g.phase(w).denom() == &1 && g.
neighbor_vec(w).len() == 4)
            .collect_vec();
        let v_neight = g
            .neighbor_vec(v)
            .iter()
            .cloned()
            .filter(|&w| g.phase(w).denom() == &4 && g.
neighbor_vec(w).len() == 2)
            .collect_vec();


        let v_neigh0_set = v_neigh0.iter().cloned().collect::<
HashSet<_>>();
        let v_neight_set = v_neight.iter().cloned().collect::<
HashSet<_>>();

        for w in g.vertices() {
            if g.phase(w).denom() != &4 || w == v {
                continue;
            }
            let w_neigh0 = g
                .neighbor_vec(w)
                .iter()
                .cloned()
                .filter(|&w| g.phase(w).denom() == &1 && g.
neighbor_vec(w).len() == 4)
                .collect_vec();
            let w_neight = g
                .neighbor_vec(w)
```

```
                 .iter()
                 .cloned()
                 .filter(|&w| g.phase(w).denom() == &4 && g.
neighbor_vec(w).len() == 2)
                 .collect_vec();

             let w_neigh0_set = w_neigh0.iter().cloned().collect
::<HashSet<_>>();
             let w_neight_set = w_neight.iter().cloned().collect
::<HashSet<_>>();

             let common0 = v_neigh0_set
                 .intersection(&w_neigh0_set)
                 .cloned()
                 .collect_vec();
             let commont = v_neight_set
                 .intersection(&w_neight_set)
                 .cloned()
                 .collect_vec();

             let n0 = 2 * common0.len();
             let nt = commont.len();

             if n0 <= best_n && nt <= best_n {
                 continue;
             }

             if n0 >= nt {
                 best_n = n0;
                 best_vs = common0;
                 best_vs.append(vec![v, w].as_mut());
                 eff_a = 1.0 / (n0 + 2) as f64;
             } else {
                 best_n = nt;
                 best_vs = commont;
                 best_vs.append(vec![v, w].as_mut());
                 eff_a = 1.0 / (nt + 2) as f64;
             }
         }
     }

     (best_vs, eff_a)
 }

 pub fn cut_v(g: &G) -> (Vec<V>, f64) {
        let mut vertices_with_denom_1 = HashMap::new();
        let mut vertices_with_denom_4 = HashMap::new();
        let mut weights = HashMap::new();
        let mut weights5 = HashMap::new();

        for v in g.vertices() {
            if *g.phase(v).denom() == 1 {
                let neighbours = g.neighbor_vec(v);
                let filtered_neighbours = neighbours
                    .iter()
                    .filter(|&w| g.neighbor_vec(*w).len() > 1)
                    .cloned()
                    .collect::<HashSet<_>>();
                vertices_with_denom_1.insert(v, filtered_neighbours)
;
            } else if *g.phase(v).denom() == 4 {
                // ignore the ones in the gadgets
                if g.neighbor_vec(v).len() < 2 {
                    continue;
                }
                let filtered_neighbours = g
                    .neighbor_vec(v)
                    .into_iter()
                    .filter(|&w| *g.phase(w).denom() == 1)
                    .collect::<HashSet<_>>();
                vertices_with_denom_4.insert(v, filtered_neighbours)
;

                weights.insert(v, 0.0);
                weights5.insert(v, 0.0);
            }
        }

        // Add weight for T-spiders in a pair
        for v in g.vertices() {
            let v_neigh = g.neighbor_vec(v);
            let n = v_neigh.len();
            if *g.phase(v).denom() == 1 {
                if n == 3 {
                    // cat3 heuristic
                    for w in v_neigh.clone() {
                        weights.entry(w).and_modify(|e| *e += 2.0);
                    }
                    // cat3 vertices should not be part of phase
gadget cuts
                    continue;
                } else if n == 5 {
                    // cat5 heuristic
                    for w in v_neigh.clone() {
                        weights5.entry(w).and_modify(|e| *e += 1.0);
                    }
                }
            } else {
                // Removing itself from the cut
                weights.entry(v).and_modify(|e| *e += 1.0);
                if n > 1 {
                    continue;
                }
                // Lone phase heuristic
                for w in v_neigh {
                    weights.entry(w).and_modify(|e| *e += 1.0);
                }
            }
        }


        for (k, _v) in weights.clone() {
            let ncut5 = *weights5.get(&k).unwrap();
            weights.entry(k).and_modify(|e| *e = f64::max(*e, (*e +
    4.0 * ncut5) / (ncut5 + 1.0)));
        }

        let max_weight = weights.iter().max_by(|a, b| a.1.
    partial_cmp(b.1).unwrap());
        let mut nbs = Vec::new();
        let mut frac = -1.0;
        if let Some((max_key, max_val)) = max_weight {
            if *max_val <= 1.0 {
                return (nbs, frac);
            }
            nbs = vec![*max_key];
            frac = 1.0 / max_val.clone() as f64;
        }

        (nbs, frac)
    }
    fn reverse_pivot(g: &mut G, vs0: &[V], vs1: &[V]) -> Vec<V> {
        let x = vs0.len() as i32;
        let y = vs1.len() as i32;
        g.scalar_mut().mul_sqrt2_pow(-(x - 1) * (y - 1));
        let v0 = g.add_vertex(VType::Z);
        let v1 = g.add_vertex(VType::Z);

        // Revert the edges between the neighbors of v0 and v1
        for &n0 in vs0 {
            for &n1 in vs1 {
                g.remove_edge(n0, n1);
            }
        }

        // Restore the original neighbors of v0 and v1
        for &n0 in vs0 {
            g.add_edge_smart(v0, n0, EType::H);
        }
        for &n1 in vs1 {
            g.add_edge_smart(v1, n1, EType::H);
        }

        g.add_edge_smart(v0, v1, EType::H);

        vec![v0, v1]
    }

    fn replace_tpair0(g: &G, verts: &[V]) -> G {
        let mut g = g.clone();

        let n = verts.len();

        let vs0 = Decomposer::reverse_pivot(&mut g, &verts[..n-2], &
   verts[n-2..]);

        Decomposer::replace_p0(&g, &vs0[..1])
    }

    fn replace_tpair1(g: &G, verts: &[V]) -> G {
        let mut g = g.clone();

        let n = verts.len();

        let vs0 = Decomposer::reverse_pivot(&mut g, &verts[..n-2], &
   verts[n-2..]);

        Decomposer::replace_p1(&g, &vs0[..1])
    }
```

## 6.2 Experimental Results

Solving for $(\alpha - \alpha_h)t > \log t$, for various values of $\Delta\alpha = \alpha - \alpha_h$ and $t$, we see the values of $t$ for which the inequality holds in Table 6.1. Of course, this does not take into account varying constant factors in the $O$ notation, and the possibility of the actual overheads being closer to $O(t)$ or even $O(t^a)$ for some $0 < a < 3$ rather than $O(t^2)$ or $O(t^3)$. Nevertheless, it showcases that as long as $\Delta\alpha > 0$, it will eventually outperform for a large enough $t$.

| $\Delta\alpha$ | 0.01 | 0.02 | 0.05 | 0.1 |
|---|---|---|---|---|
| $t >$ | 997 | 440 | 144 | 59 |

**Table 6.1:** Values of $t$ for which $(\alpha - \alpha_h)t > \log t$ holds

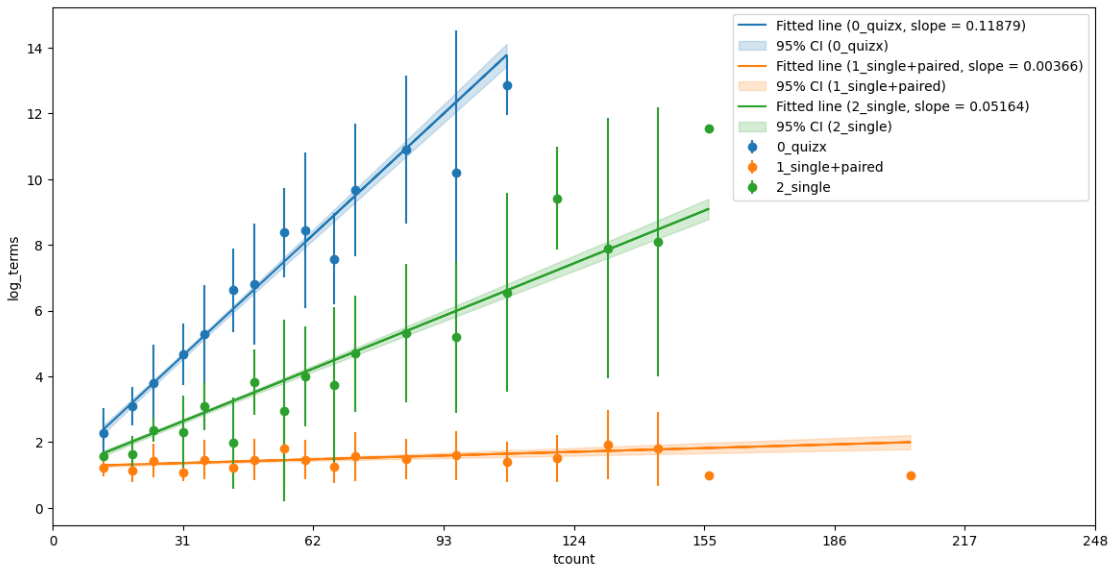


**Figure 6.1:** Mean log terms (lower better) for 6-qubit modified hidden shift circuits

The slope being close to 0 in Figure 6.1 points to a constant number of terms $O(1)$ with respect to the $T$-count.

The log time graph supports this showing an $O(\log t)$ relationship, showing that the heuristic is $poly(t)$ in time complexity.

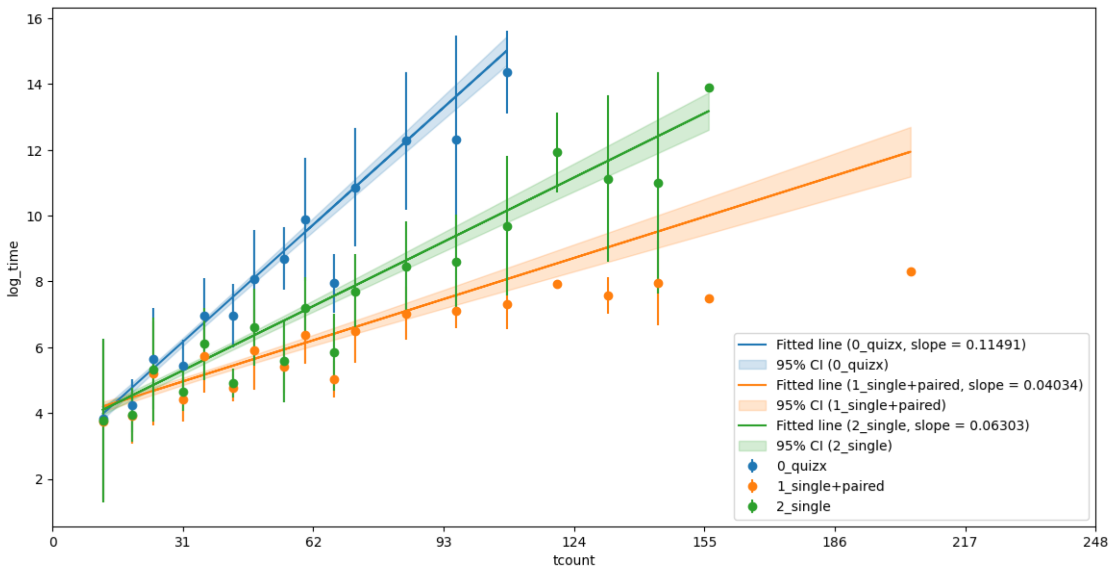


**Figure 6.2:** Mean log time (lower better) for 6-qubit modified hidden shift circuits

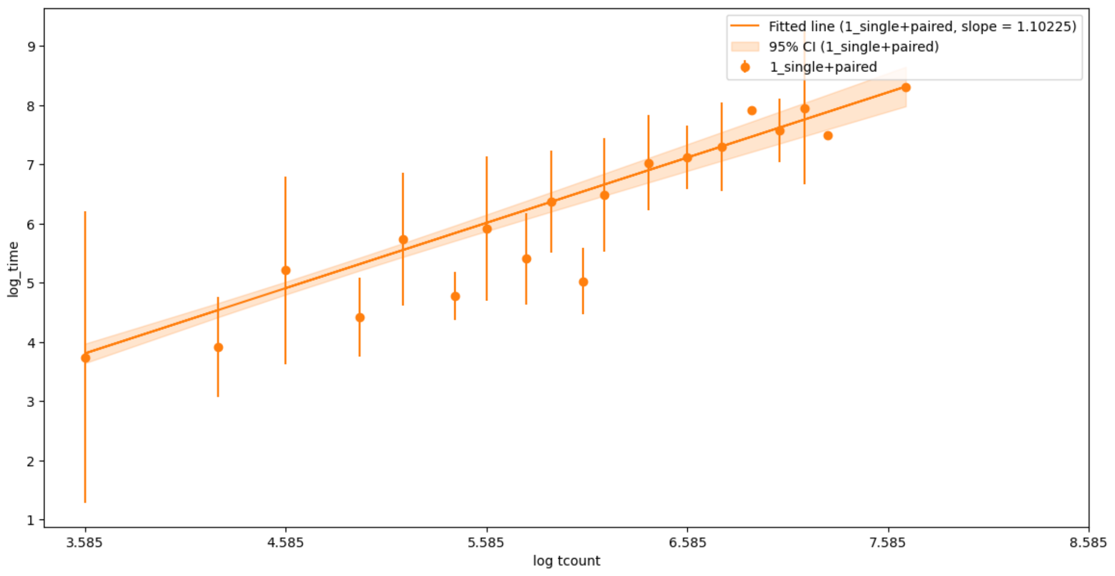


**Figure 6.3:** Mean log time (lower better) against log $T$-count for `single+paired` on 6-qubit modified hidden shift circuits

# References

[1] Xiaosi Xu, Simon Benjamin, Jinzhao Sun, Xiao Yuan, and Pan Zhang. *A Herculean task: Classical simulation of quantum computers.* 2023. arXiv: `2302.08880`.

[2] Aleks Kissinger and John van de Wetering. “Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions”. In: *Quantum Science and Technology* 7.4 (July 2022), page 044001. URL: `http://dx.doi.org/10.1088/2058-9565/ac5d20`.

[3] Aleks Kissinger, John van de Wetering, and Renaud Vilmart. “Classical Simulation of Quantum Circuits with Partial and Graphical Stabiliser Decompositions”. en. In: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TQC.2022.5`.

[4] Julien Codsi. “Cutting-Edge Graphical Stabiliser Decompositions for Classical Simulation of Quantum Circuits”. Master’s thesis. University of Oxford, 2022.

[5] Matthew Sutcliffe and Aleks Kissinger. “Procedurally Optimised ZX-Diagram Cutting for Efficient T-Decomposition in Classical Simulation”. In: *Quantum Physics and Logic 2024*. Edited by Alejandro Díaz-Caro and Vladimir Zamdzhiev. 2024. URL: `https://arxiv.org/abs/2403.10964`.

[6] Julien Codsi and John van de Wetering. *Classically Simulating Quantum Supremacy IQP Circuits through a Random Graph Approach.* 2023. arXiv: `2212.08609 [quant-ph]`. URL: `https://arxiv.org/abs/2212.08609`.

[7] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes.* Cambridge University Press, 2017. URL: `https://books.google.com.sa/books?id=I9gcDgAAQBAJ`.

[8] Bob Coecke and Ross Duncan. “Interacting quantum observables: categorical algebra and diagrammatics”. In: *New Journal of Physics* 13.4 (Apr. 2011), page 043016. URL: `http://dx.doi.org/10.1088/1367-2630/13/4/043016`.

[9] Aleks Kissinger and John van de Wetering. “Universal MBQC with generalised parity-phase interactions and Pauli measurements”. In: *Quantum* 3 (Apr. 2019), page 134. URL: `http://dx.doi.org/10.22331/q-2019-04-26-134`.

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

[11] Aleks Kissinger and John van de Wetering. *Picturing Quantum Software.* 2024.

[12] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. “Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus”. In: *Quantum* 4 (June 2020), page 279. URL: `https://doi.org/10.22331/q-2020-06-04-279`.

[13] Aleks Kissinger and John van de Wetering. “Reducing the number of non-Clifford gates in quantum circuits”. In: *Phys. Rev. A* 102 (2 Aug. 2020), page 022406. URL: `https://link.aps.org/doi/10.1103/PhysRevA.102.022406`.

[14] Hammam Qassim, Hakop Pashayan, and David Gosset. “Improved upper bounds on the stabilizer rank of magic states”. In: *Quantum* 5 (Dec. 2021), page 606. URL: `http://dx.doi.org/10.22331/q-2021-12-20-606`.

[15] Sergey Bravyi and David Gosset. “Improved Classical Simulation of Quantum Circuits Dominated by Clifford Gates”. In: *Physical Review Letters* 116.25 (June 2016). URL: `http://dx.doi.org/10.1103/PhysRevLett.116.250501`.

[16] Dave Wecker and Krysta M. Svore. *LIQUi|>: A Software Design Architecture and Domain-Specific Language for Quantum Computing.* 2014. arXiv: `1402.4467 [quant-ph]`. URL: `https://arxiv.org/abs/1402.4467`.

[17] Daniel Gottesman. *The Heisenberg Representation of Quantum Computers.* 1998. arXiv: quant-ph/9807006 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/9807006.

[18] Sergey Bravyi, David Gosset, and Yinchen Liu. "How to Simulate Quantum Measurement without Computing Marginals". In: *Physical Review Letters* 128.22 (June 2022). URL: http://dx.doi.org/10.1103/PhysRevLett.128.220503.

[19] Sergey Bravyi et al. "Simulation of quantum circuits by low-rank stabilizer decompositions". In: *Quantum* 3 (Sept. 2019), page 181. URL: http://dx.doi.org/10.22331/q-2019-09-02-181.

[20] Sergey Bravyi, Graeme Smith, and John A. Smolin. "Trading Classical and Quantum Computational Resources". In: *Physical Review X* 6.2 (June 2016). URL: http://dx.doi.org/10.1103/PhysRevX.6.021043.

[21] Matthew Sutcliffe. *Smarter k-Partitioning of ZX-diagrams for Improved Quantum Circuit Simulation.* Presentation at Oxford Quantum Group Workshop. 2024.

[22] Tomoyuki Morimae and Suguru Tamaki. "Fine-grained quantum computational supremacy". In: *Quantum Information and Computation* 19.13-14 (2019), pages 1089–1115.

[23] John E. Hopcroft and Richard M. Karp. "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs". In: *SIAM Journal on Computing* 2.4 (1973), pages 225–231.

[24] Balaram Behera, Edin Husić, Shweta Jain, Tim Roughgarden, and C. Seshadhri. "FPT Algorithms for Finding Near-Cliques in c-Closed Graphs". In: *13th Innovations in Theoretical Computer Science Conference (ITCS 2022).* Edited by Mark Braverman. Volume 215. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 17:1–17:24. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITCS.2022.17.

[25] Aleks Kissinger, John van de Wetering, and Renaud Vilmart. *QuiZX: a quick Rust port of PyZX.* https://github.com/zxcalc/quizx. 2021.

[26] Aleks Kissinger and John van de Wetering. "PyZX: Large Scale Automated Diagrammatic Reasoning". In: Proceedings 16th International Conference on *Quantum Physics and Logic,* Chapman University, Orange, CA, USA., 10-14 June 2019. Edited by Bob Coecke and Matthew Leifer. Volume 318. Electronic Proceedings in Theoretical Computer Science. Open Publishing Association, 2020, pages 229–241.

[27] David Gosset, Vadym Kliuchnikov, Michele Mosca, and Vincent Russo. "An algorithm for the T-count". In: *Quantum Info. Comput.* 14.15–16 (Nov. 2014), pages 1261–1276.

[28] Michael J. Bremner, Ashley Montanaro, and Dan J. Shepherd. "Achieving quantum supremacy with sparse and noisy commuting quantum computations". In: *Quantum* 1 (Apr. 2017), page 8. URL: http://dx.doi.org/10.22331/q-2017-04-25-8.

[29] Michael J. Bremner, Ashley Montanaro, and Dan J. Shepherd. "Average-Case Complexity Versus Approximate Simulation of Commuting Quantum Computations". In: *Physical Review Letters* 117.8 (Aug. 2016). URL: http://dx.doi.org/10.1103/PhysRevLett.117.080501.

[30] Filipa C. R. Peres and Ernesto F. Galvão. “Quantum circuit compilation and hybrid computation using Pauli-based computation”. In: *Quantum* 7 (Oct. 2023), page 1126. URL: `http://dx.doi.org/10.22331/q-2023-10-03-1126`.

[31] Mark Koch, Richie Yeung, and Quanlong Wang. *Speedy Contraction of ZX Diagrams with Triangles via Stabiliser Decompositions*. 2023. arXiv: `2307.01803 [quant-ph]`. URL: `https://arxiv.org/abs/2307.01803`.