Cutting-Edge Graphical Stabiliser Decompositions for Classical Simulation of Quantum Circuits

Julien Codsi
St Edmund Hall
University of Oxford

A thesis submitted for the degree of
MSc in Mathematics and Foundations of Computer Science
Trinity 2022

Acknowledgements

I would like to thank *Tuomas Laakkonen* for the countless fruitful discussion and help with the implementation of the algorithm. His help has greatly improved the content of this work. I would also like to thank *John van de Wetering* for his supervision. Finally, I would like to thank my family and my girlfriend *Thalie St-Jacques* for their support and encouragement.

Abstract

The aim of this thesis is to study and improve classical simulation of quantum circuits with stabiliser decompositions by using the ZX-calculus. We propose more efficient decompositions for a large family of states in addition to integrating graphs cuts to make use of potential low connectivity of ZX-diagrams. We also discovered different heuristics to speedup the simulations. Finally, we implemented our method and compared it with the state-of-the-art on different families of circuits to asses the improvements achieved by our approach.

Contents

List of Figures

Chapter 1

Introduction to Quantum Computing and the ZX-Calculus

Over the last hundred years, computers have completely revolutionised the world and continue to do so today. Yet for a vast number of problems of interest, we do not have the computational capabilities to solve them in a reasonable time. With the end of Moore's Law, we cannot simply rely on hardware improvements to solve ever larger problems. It is in this context that quantum computing poses itself as a potential solution. This new model of computing makes use of the quantum nature of the microscopic world to achieve greater efficiency than one would expect from a classical computer. This idea, generally attributed to Richard Feynman [13], has now been the subject of a continuous global effort for just over forty years.

Two components make quantum computing so powerful, namely superposition and entanglement.

Superposition is the property of an object to be in several states at once. For example, an electron in an atom can be in several disjoint energy levels simultaneously. This electron is not between these levels, but rather in each of them a little. In the context of computing, we can create an analogue to the classical bit, the qubit, which can be 0 and 1 simultaneously. By composing n qubits together, it is possible to have a superposition state of the 2^n possible n-bits strings. This makes it possible, for example, to compute a function f on all strings of length n by calling it only once. Essentially, it is this phenomenon that makes quantum computing seem to have an exponential advantage over classical computing. Unfortunately, it's not all that simple, to get information out of a qubit, you have to measure it. One of the most important and mysterious components of quantum physics is that any measurement of a quantum system changes it. More precisely, it collapses its wave function. In the case of our qubits and f, a measurement will return the value of the function only of a

string (and even of a random string). The amount of information that can be obtained from a quantum system is fundamentally limited. On the other hand, this limited information can depend on the entire state, allowing us to obtain global information more quickly. The name of the game is to be able to extract the information we want from a state. For example, if you have an array containing a single non-zero element, and you want to know which one, conventionally, you would have to try them all (or at least half of them on average) to accomplish this task. On the other hand, with a quantum computer, it is possible to do this in $O(\sqrt{n})$ with Grover's algorithm.

The second component of quantum computing that has no analogue in classical computing is entanglement. Entanglement is the property of a system that cannot be understood by looking only at its parts. It is possible to correlate the value of several qubits so that they all give the same result when measured while keeping the result of each measurement random. In a classical context, this would be the equivalent of having several people flip a coin and being able to guarantee that they all get the same result. If we repeat this experiment several times, we will have that each person gets heads and tails about 50 per cent of the time, but these marginal distributions are not enough to understand the whole system. The sum is more complex than its parts. This almost gives the impression that the coins can communicate with each other, but the correlation of entangled qubits remains even if they are so far apart that the laws of physics forbid any rapid communication between them (because information cannot travel faster than the speed of light). Moreover, it is possible to show that no information can be sent using these correlations. One might think that the results of the measurements are determined in advance and are hidden in the qubits, but Bell's theorem shows that this cannot explain the phenomenon[1]. Quantum computing tries to use these strange properties to get a (potentially exponential) advantage over classical computers.

1.1 Basic notation

A useful property of quantum mechanics is that it is linear. This linearity allows (pure) quantum states to be represented by vectors in $\mathbb{C}^2$. For example, the quantum

[1] A common misconception is that Bell's theorem shows that our universe is *non-local*. This is not the case. It only shows that no hidden variables theory can explain quantum phenomena. For example, superdeterminism is a local theory that could explain quantum behaviours. Other local realist theories have been found with weaker assumptions (see, for example, [28])

analogue of a bit at zero is $|0\rangle$ and the analogue of a bit at 1 is $|1\rangle$.

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

More generally, we have the definition of *pure states*.

Definition 1 (Pure States)**.** *A pure state is a vector*

$$|\ \rangle = \qquad = \ |0\rangle + \ |1\rangle$$

where $\ ; \ \in \mathbb{C}$ *with* $\ ^2 + \ ^2 = 1$

The condition on and is a normalisation condition guarantying that $\langle|\ \rangle = 1$. Two pure states that are very common are

$$|+\rangle = \frac{1}{\sqrt{2}} \begin{matrix} 1 \\ 1 \end{matrix} \qquad |\ \rangle = \frac{1}{\sqrt{2}} \begin{matrix} 1 \\ 1 \end{matrix}$$

They represent a uniform superposition of $|0\rangle$ and $|1\rangle$. In addition to pure states, there are also mixed states which can be represented by matrices. We will, however, not need these so we omit their formal definition. To form larger spaces, we can combine states through the use of tensor products. For one qubit states, we have a special and less cumbersome notation $|x\rangle \ |y\rangle = |xy\rangle$.

Now that we know what are the states of quantum computing, we will explain how to transform states into others. Quantum gates are unitary operators that allow us to do exactly that. They are described by unitary matrices. To apply a gate U to a state $|\ \rangle$, we can use the usual matrix/vector multiplication $U|\ \rangle$. Just like states, we can combine gates with a tensor product to build larger gates that can be applied to larger states. We can also chain multiple gates together $U_1 U_2 \cdots U_k|\ \rangle$. There is also a more general class of transformation that is called isometries. We will not need those in this dissertation so we omit their definition. Some usual gates are listed in table 1.1.

Finally, we have measurements. The simplest form of measurement is what is called *projective measurements*. This type of measurement tells us how far one state is from another. To project a state $|\ \rangle$ to a state $|\ \rangle$ we can use the scalar product $\langle\ |\ \rangle$ which is what we call a *bra-ket*. We can also compute the probability of obtaining a string $x_1;\cdots;x_n$ while measuring a state $|\ \rangle$ by computing

$$|\langle x_1 \cdots x_n|\ \rangle|^2$$

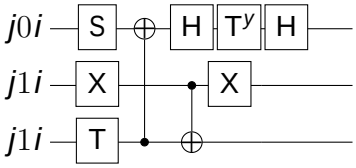
$$I = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \qquad X = \text{NOT} = \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} \qquad Z = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

$$Y = \begin{matrix} 0 & i \\ i & 0 \end{matrix} \qquad H = \frac{1}{\sqrt{2}} \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} \qquad S = \begin{matrix} 1 & 0 \\ 0 & i \end{matrix}$$

$$T = \begin{matrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{matrix} \qquad \text{CNOT} = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix}$$

Table 1.1: Some common quantum gates

This is what we call the *Born rule*. We can regroup all of these concepts in the model of *quantum circuit*. In this model, a gate is represented by a box with wires coming in and out. Those wires represent the inputs and outputs of a gate. To compose gates we can simply chain them with wires from left to right. For example, we can build the following circuit



The left of a circuit is the input of a circuit while the right is the output. Here the vertical wire is a notation shorthand for a very common gate called the CNOT gate.

1.2 ZX-Calculus

The quantum circuit model has long been considered the model of choice for representing quantum computation. This model has several advantages over, for example, quantum Turing machines. It offers a strong visual intuition for understanding the workings of the calculations made with it. By knowing some useful identities such as the commutation of certain gates, it is even possible to manipulate a quantum circuit in a limited way. Although this visual intuition is very convenient, it is sometimes necessary to express everything in equations containing a lot of Kronecker products and matrix products. In a way, these equations concern objects that are two-dimensional (the circuits), which we are forced to project into one dimension. It would be much nicer if we could always work directly with diagrams. This is where the ZX-calculus shines. ZX-calculus is a graphical language first introduced by Bob Coecke and Ross Duncan in 2008 [8], that allows one to reason about quantum computations without having to manipulate anything other than ZX-diagrams, which are, in a way, a

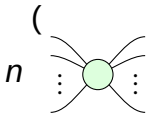
generalisation of quantum circuits (in fact, they can be used to represent any linear operator). This approach turns out to give a much more combinatorial intuition to quantum computing. This new viewpoint has been successfully used in a wide range of applications, from understanding indeterminate causal structures, through measurement-based computing, to classical quantum circuit simulation (the subject of this dissertation). In the next section, we give a brief introduction to the ZX-calculus inspired by [33].

1.2.1 ZX-diagrams

We now introduce the notion of ZX-diagram as a generalisation of quantum circuits. ZX-diagrams are built out of four generators; Hadamard boxes, green and red vertices and bent wires. Like quantum gates, these represent matrices. The number of incoming and outgoing wires gives us the dimensions of the matrix in question. The matrix product and the Kronecker product are represented in the same way as in a quantum circuit. That is, a wire between two matrices represents a matrix product and the vertical juxtaposition of two matrices represents the Kronecker product. If more than one wire connects two matrices, it can be seen as a tensor contraction or, more intuitively, that many of the qubits leaving one gate enter another gate. The formal definition of these generators.

1.2.1.1 Spiders

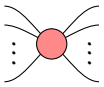
Definition 2 (Green Spider)**.**



$$= |0\rangle^{\otimes m}\langle 0|^{\otimes n} + e^{i\alpha}\,|1\rangle^{\otimes m}\langle 1|^{\otimes n} \tag{1.1}$$

Here the dots are used to define an infinite family with an arbitrary number of potentially different inputs and outputs. For obvious reasons, this type of vertex is called a green spider. Good intuition for these spiders, especially when $\alpha = 0$, is that a spider is a generalisation of the Kronecker delta. Indeed, if all the qubits that enter a spider are in the same state on the standard computational basis, a certain number of copies of this pure state are produced. On the other hand, if this is not the case, then nothing (or more precisely 0) is obtained. We can define red spiders in a very similar way.

Definition 3 (Red Spider)**.**

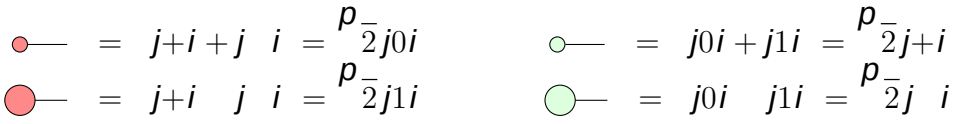


$$= j+ \quad +\hbar+ \quad +j+e^{i} \; j \qquad \hbar \qquad j \tag{1.2}$$

Here we dropped the m and n, as it makes the writing more cumbersome. For reasons we will not discuss here, green and red spiders are also often called Z-spiders and X-spiders respectively.

Note: When $= 0$, it is convenient not to write the angle in the spider. For example, this greatly simplifies the notation for circuits composed of only CNOTs.

Slightly abusing Dirac notation, let's define "empty" bras and kets as $\hbar j := 1$ and $ji := 1$. We can now define spiders with no input wire or no output wire. We then have that



$$= j+i+j \;\; i \; = \frac{p}{2} j0i \qquad = j0i+j1i \; = \frac{p}{2} j+i$$

$$= j+i \;\; j \;\; i \; = \frac{p}{2} j1i \qquad = j0i \;\; j1i \; = \frac{p}{2} j \;\; i$$

We find the usual basic states modulo a multiplicative constant. Note that it is customary to ignore multiplicative constants (as long as they are non-zero) in the same way that it is customary to forget the global phase in quantum circuits. The reason for this is that it is often unnecessary and it is very easy to find this constant at the end of our calculations (for example by using circuit unitarity). Another reason is also that the ZX-calculus has several software implementations that automatically calculate these constants. Therefore, we will use to denote "equal up to a multiplicative constant". With this notation, we have

$$j0i \qquad j+i \tag{1.3}$$

$$j1i \qquad j \;\; i \tag{1.4}$$

This also allows us to use a shorthand of notation to talk about the basis states.

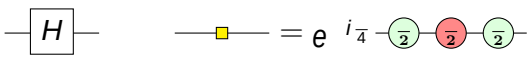
$$jki$$

1.2.1.2 Hadamard Box

It is possible, using the Euler rotation decomposition theorem, to prove that any unitary gate on a qubit can be represented by a series of red and green spiders[2]. Thus, it is possible to express the Hadamard gate with spiders. Nevertheless, this gate is so common that it is worthy of its own notation.

[2] This is because the rotations of the *Bloch sphere* with respect to the X and Z axis can be expressed in terms of spiders.

Definition 4. *(Hadamard box or H-box)*



This box corresponds to the usual matrix definition of a Hadamard gate. Hadamard boxes are represented by a square instead of a circle because, unlike spiders, we restrict them to having only one input and one output wire[3].

1.2.1.3 Cups, Caps and Swaps

The last family of generators are bent wires.

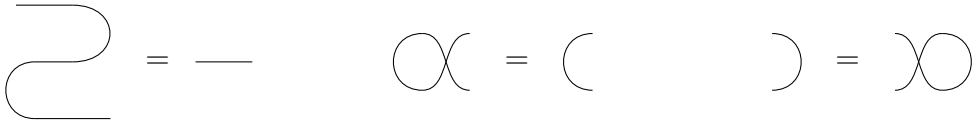
$$\text{CUP} = \quad = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{CAP} = \quad = \quad 1 \quad 0 \quad 0 \quad 1$$

$$\text{SWAP} = \quad =$$

Next, we will see why those are represented with wires.

1.2.1.4 Symmetries and OCM

Cups, caps and swaps obey what we call the *yanking equations*

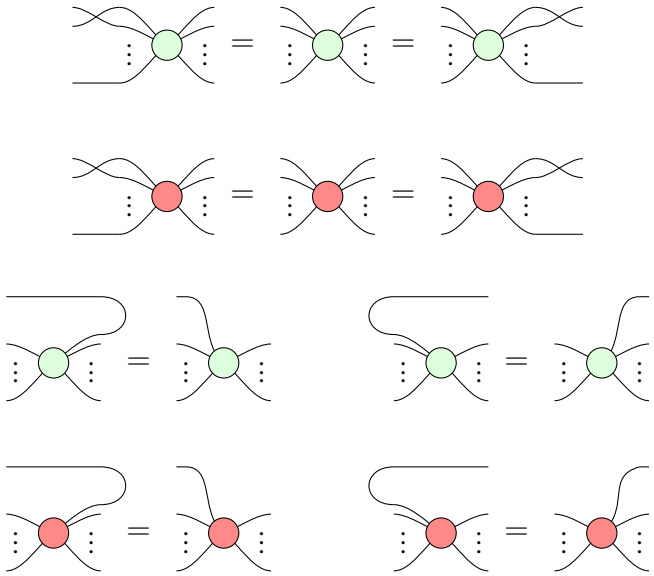


(1.5)

These equations imply that we can deform wires at will without changing the value of the underlying matrix as long as they connect to the same spider and in the same

[3] There are types of calculus, such as ZH-calculus, which do not have this restriction.

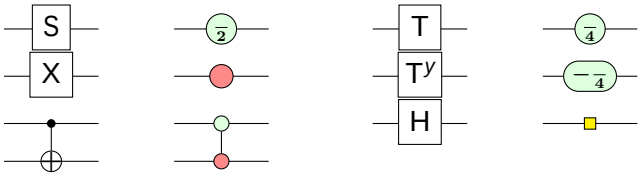
order. Moreover, we have similar wire bending equations for spiders.



These equations imply that spiders are what we call *flexsymetric* meaning that the ordering or direction of the wires doesn't matter as long as they are connected to the spider. Together with the yanking equations, these imply that ZX-diagrams can be seen as graphs and are topologically deformable! More formally, if we define an isomorphism between two diagrams as a function that preserves adjacency relations, spider colours, spider angles, inputs and outputs of the whole diagram and their order, then two isomorphic diagrams represent the same matrix! This fact is known as "Only Connectivity Matters" or OCM. It implies that we can consider ZX-diagrams as a special type of graph[4] and use a range of techniques from the fields of combinatorics and graph theory!

1.2.1.5 Generalisation of Quantum Circuit

At the beginning of this chapter, we mentioned that ZX-diagrams were a generalisation of quantum circuits and we now know enough about ZX-diagrams to see why this is the case. For most common gates, there is a simple ZX equivalent.

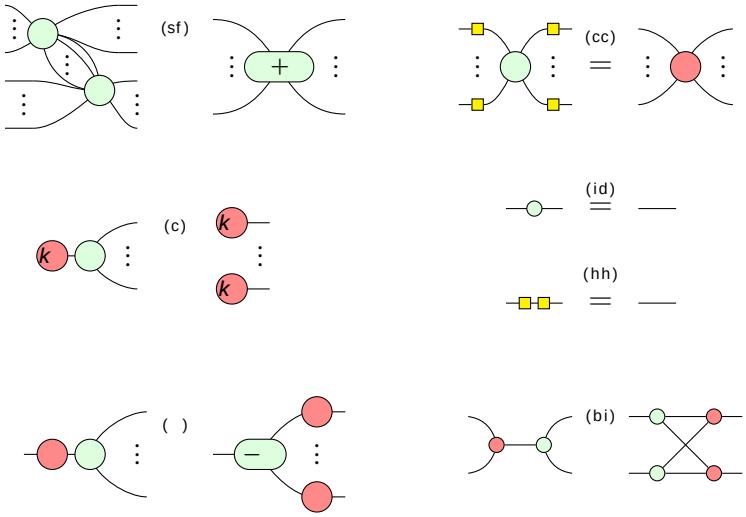


[4] In fact, we should use multigraphs instead, because nothing prevents two spiders from having more than one wire in common, but we can always get rid of multiple edges and even loops using ZX-calculus. We can therefore limit ourselves to the case of simple graphs

We invite curious readers to verify those identities by computing the matrices directly. This means that we can easily transform quantum circuits into ZX-diagrams. Thus, ZX-diagrams are at least as general as quantum circuits but are even more expressive (they can represent any linear maps) and have nicer symmetries as we have just seen.

1.3 ZX-Calculus

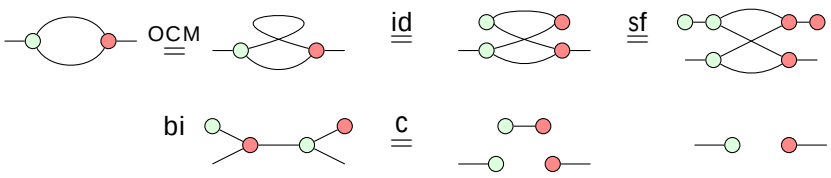
We have now defined ZX-diagrams and how they represent linear operators. Beyond being just a convenient notation, we want to be able to do proofs using only these. To do this, we will need rules that allow us to manipulate ZX-diagrams.



These seven rules and their colour-symmetric versions (when colours are swapped) form the ZX-calculus. In order, these rules are called, spider fusion, colour change, -copy, identity removal, Hadamard cancel, -commute and the bialgebra rule. To showcase how these rules can be used, we will prove a known circuit identity. Do to so, we will need a lemma called the *Hopf rule*.

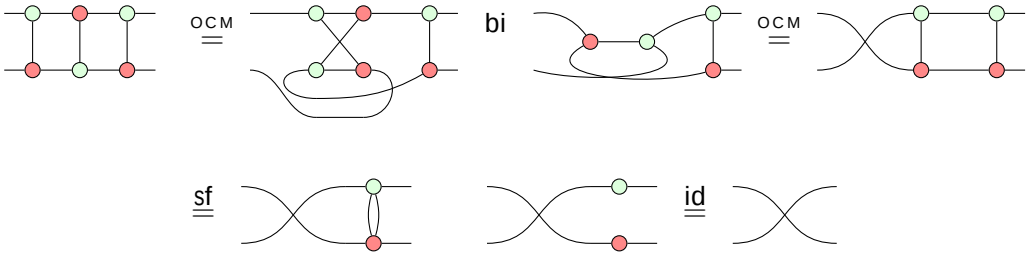
Lemma 1 (Hopf rule)**.**

Proof.

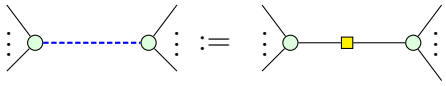


□

We can now show that three alternate CNOTs are equivalent to a SWAP.



Note : Since H-boxes can only have one input and one output and they are self-inverse a shorthand of notation often used is to represent them as a type of edges called *H-edges*.



1.4 Fragments

It often happens that one wants to discuss a restricted class of quantum circuits. For example, one may be interested in circuits composed only of CNOTs, Clifford gates[5] or Clifford+T. It turns out that these last classes have a simple representation with ZX diagrams. Indeed, these correspond to *fragments* of the ZX-calculus.

Definition 5 (Fragment)**.** *A fragment of the ZX-calculus is a set of ZX-diagrams induced by a subset of the generators which is closed under the rules of the ZX-calculus.*

CNOT circuits

CNOT circuits are representable using only phase-free spiders. Indeed, none of the rules of the ZX-calculus introduces an angle into a diagram that did not previously have one.

Clifford circuits

Similarly, Clifford circuits can be represented using only spiders whose phases are restricted to multiples of $\frac{1}{2}$. To see this, we note that the rules of ZX-calculus only add or negate phases. Consequently, the fact that the set $\{k\frac{1}{2}\}_{k=0}^{3}$ is closed on these two operations allows us to conclude this is a proper fragment.

Clifford+T circuits

For the same reasons, Clifford+T circuits can be represented using only spiders whose phases are restricted to multiples of $\frac{1}{4}$. This fragment is particularly interesting,

[5] Clifford gates are CNOT, H, NOT and S.

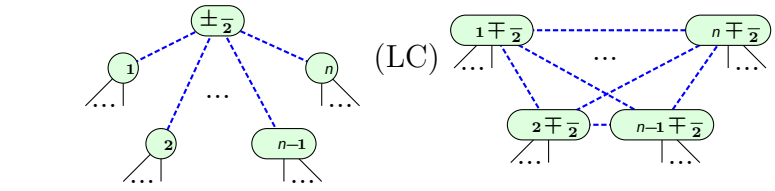
because, thanks to the Solovay-Kitaev theorem[6], it is known to be approximately universal for quantum computing. We will focus on this fragment for the rest of this thesis and all diagrams from now on will be assumed to be of this form.

1.5 Simplifications Algorithm

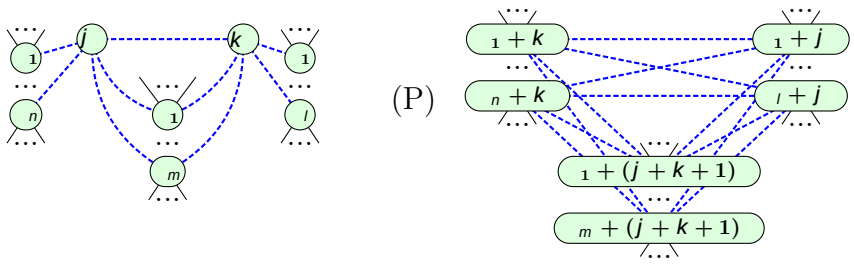
With the ZX-calculus, it is possible to design diagram simplification routines that optimise certain metrics. We briefly present the procedure introduced in [10] and improved in [18] that we will use in this thesis. Firstly, by using spider fusions, Hadamard cancels, colour changes and identity removals, it is possible to transform any diagram into a *graph-like* diagram.

Definition 6 (Graph-Like ZX-diagram)**.** *A graph-like ZX-diagram is a diagram where all spiders are green and all edges except outputs and inputs wire are H-edges.*

In addition, it is also possible to prove that the two very useful following equations hold.



and



Which we call *local complementation* and *pivoting*. With these two operations, it is possible to simplify a ZX-diagram by removing a great deal of Clifford spiders. In particular, if a diagram is initially only made of Clifford spider, it is possible to trivialise it by repetitively using these operations. Doing so offer an alternative proof of the Gottesman-Knill theorem [14] which states that any Clifford circuit is classically simulatable in polynomial time[7]. In a Clifford+T diagram, applying these repetitively produces an equivalent diagram in the *reduced form*.

[6] A proof of which can be found in [26].

[7] see [9] for a fully diagrammatic proof of this statement

Definition 7. *(Reduced Form) A ZX-diagram is said to be in the reduced form if*

- *No two Clifford spiders are adjacent*
- *No internal*[8] *Clifford spider has a degree smaller than 3*
- *No internal spider phases are odd multiples of* $\frac{}{2}$

In the following chapters, we will say that a diagram is simplified if it is put in reduced form through the use of local complementations, pivots, spider fusions, Hadamard cancels and identity removals.

[8] An internal spider is a spider not connected to the inputs or outputs of a diagram.

Chapter 2

Simulation Through Stabiliser Decomposition

One task that is proving important to be able to assert quantum supremacy is to understand the power of (classical) quantum computer simulation. Indeed, in 2019 Google [2] declared to have reached this historical milestone by performing a calculation with a quantum computer that they estimated no classical computer could do in less than 10,000 years. Since then, new simulation techniques have been developed and it is now known that the Google experiment can be reproduced classically in less than a day on modern supercomputers [16][27]. In addition to allowing us to draw the line of supremacy, the study of quantum simulation helps us to understand what makes quantum computers so powerful. One might initially believe that it is the entanglement that gives quantum advantage. However, the Gottesman-Knill theorem shows that this is not the case, because the class of Clifford circuits contains maximally entangled states, but can still be simulated in polynomial time.

In this chapter, we will focus on what is called *strong simulation*. This is the task of computing the probability of obtaining an output knowing the input of a circuit. We note that after having chosen an input and output vector, it is enough to compute the resulting amplitude to then calculate a probability using the Born rule. This task differs from *weak simulation* in which the goal is to obtain a sample from the output distribution of a circuit given an input (more on this task in chapter 9). In both cases, we can without loss of generality focus on the specific case where the input of a circuit is $|0^n\rangle$.

Several techniques exist to simulate quantum computations using classical computers. We can, for example, directly make matrix products. This has a complexity $\tilde{O}(2^n)$[1] for any circuit with polynomially many bounded size gates. An approach

[1] O is a standard notation that ignores polynomial factors of exponential functions.

that often proves to be much more efficient (time-wise and memory-wise) is to do a stabiliser decomposition.

The goal of this technique is to represent the output $j \quad i$ of the circuit as a linear combination of Clifford states[2] $j \quad _1 i ; \ldots ; j \quad _k i$.

$$j \quad i = \sum_{i=1}^{k} a_i \, j \quad _i i$$

We can then compute the probability of any output x by computing

$$hxj \quad i = \sum_{i=1}^{k} a_i \, hxj \quad _i i$$

Since all of $j \quad _i i$ are Clifford, it is possible to compute each $hxj \quad _i i$ in polynomial time. Such a simulation can be done in $\tilde{O}(k)$. It is therefore important to find sums in which the value of k is minimized.

Definition 8 (Stabiliser Rank)**.** *The stabiliser Rank of a state $j \quad i$, written $( )$ is the smallest k for which it is possible to express $j \quad i$ as a linear combination of k Clifford states.*

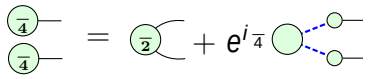
It is strongly believed that, in the worst case, $( )$ grows exponentially with the size of the circuit. Otherwise, a quantum computer would only have a subexponential advantage over a classical computer. However, extensive work has been done to give an upper bound on the stabiliser rank of states from a specific family of circuits.

Fortunately, there is an elegant adaptation of these ideas to the ZX-calculus. To explain it, let us restrict ourselves to the Clifford+T class of circuits (and thus to the fragment of the ZX-calculus induced by the multiples of $\frac{}{4}$). In this class, the only real obstacles to simulating a circuit efficiently are T gates or, more precisely, spiders with angles that are multiples of $\frac{}{4}$ without being multiples of $\frac{}{2}$. Indeed, without such gates, the Gottesman-Knill theorem or, more specifically, the simplification routine from [18] assures us that it is possible to simulate the resulting ZX-diagram in polynomial time. If we have a diagram that contains only one of these T gates, then using the definition of a T state,

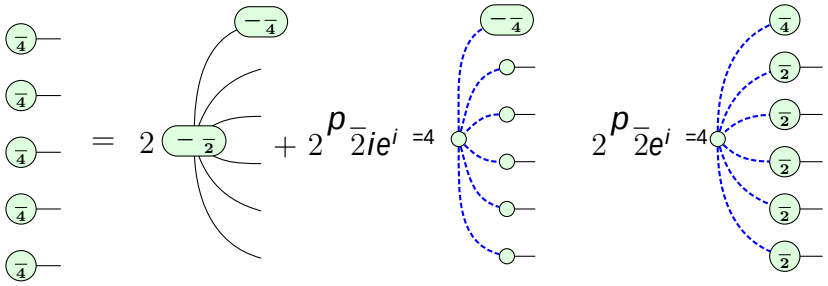
we can replace the T state in the diagram with a sum of two ZX-diagrams without T gates. These two diagrams will then be easy to simulate. To make sure that we get

[2] A Cli ord state is a state that can be obtained using a Cli ord circuit.

our spider with a "bad" angle in this form, we can use the spider fusion rule to unfuse a T state from any T spider. This technique is somewhat equivalent to a technique from the quantum literature called magic state injection. If a diagram contains more than one T spider, then you can repeat this procedure for each of these spiders leading to decomposition into 2^t terms where t is the number of t gates. It is possible to slow down this exponential growth by using more efficient decompositions than the one mentioned above. For example, it is possible to decompose 2 T states into only two terms.



This gives us a decomposition in $2^{\ t}$ where $= 0.5$. The most efficient decomposition of (magic) T states known to date, in terms of , is that of [20] which transforms five T states into three diagrams in which only one T gate remains.



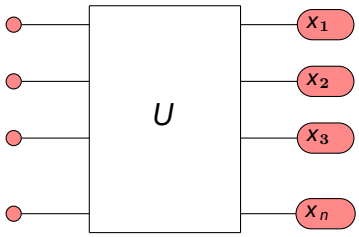
Indeed, this decomposition removes 4 T gates in 3 terms, giving us a 0.396.

Another idea from [19] that proved to be very beneficial is to try to do the most ZX simplification possible in between two replacements. This works because if two T gates ever get fused, they transform into a Clifford and since Cliffords are closed under the ZX-calculus, no new T gates can be created. Even though this led to only a polynomial improvement in the worst-case time complexity (if no T gate gets simplified during this process), in practice, these gave massive speedups compared to previous techniques.

Another approach that is also able to greatly speed up the whole process is to look for other patterns than T state injections. Even if such patterns aren't as generic and cannot be used to remove every T gate, they might have better decompositions so it is worth looking for them as pattern matching can be done quickly. One particular example of such patterns is cat states which are quite common, can be found in linear time, and, for cat states of a certain size, can bring the complexity down to $= 0.25$. We will discuss cat states in more detail in the next chapters.

We note that our technique is slightly different from a direct stabiliser decomposition because instead of decomposing $U\,|0^n\rangle$, we are decomposing $\langle x|\,U\,|0^n\rangle$ which is

equivalent to finding the value of



which is slightly simpler as we can use the output to obtain some ZX simplifications.

2.1 Time Complexity of Partial Decomposition

In the previous section, we encountered a partial decomposition. Since all terms of the decomposition removed the same number of T gates, the time complexity analysis of this decomposition was straightforward. In general, this is not always the case and it is necessary to have a way to capture this non-homogeneity so that we can compare our decompositions. To do this, we would like to be able to calculate the complexity of an algorithm that would be able to always use a given decomposition. Let us consider a decomposition that takes a ZX-diagram with t T gates and returns k terms. Let $a_1;:::a_k$ be the number of T gates removed in each of these terms. Without loss of generality, let $a_1 \quad a_2;:::; \quad a_k$. We want to find the total number of terms generated using only this decomposition. Thus, we want to understand the asymptotic behaviour of

$$F(t) = \sum_{i=1}^{k} F(t \quad a_i)$$

One way to bound this function uses the minimal a_i for every term.

$$F(t) \sum_{1}^{k} F(t \quad a_k) = kF(t \quad a_k)$$

This gives us a bound of

$$F(t) \quad 2^{\ t} \text{ where } \frac{\lg(k)}{a_k}$$

Moreover, it turns out that we can find exactly the value . Indeed, if we take the characteristic polynomial of our recurrence, we obtain

$$x^{a_k} \sum_{i=1}^{k} x^{a_i + a_k}$$

Since there is only one sign change in the coefficients, we can use Descartes' sign rule[3]. This implies that this polynomial has only one positive root r and that,

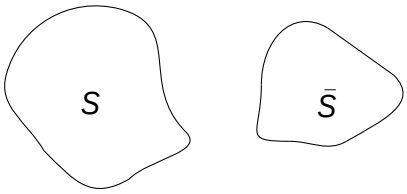
[3]First proven in Descartes' revolutionary work, La Geometrie.

therefore, $F(t) = O(r^t)$. In other words, we have that $\quad = \lg(r)$. This value of r can be found quickly, by for example Newton¡s method, as we know that $1 \quad r \quad 2$ for all useful decompositions. This trick to obtain the asymptotics of F is a special case of *branching numbers* (see [11] for a more in-depth discussion).

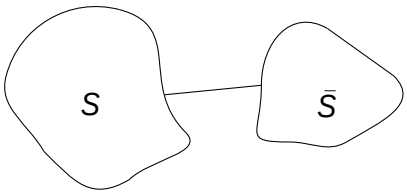
Chapter 3

Graph Cuts Decompositions

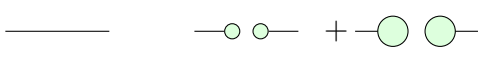
An interesting situation occurs when, after a decomposition, one obtains a ZX-diagram D which splits into two disconnected components S and $\bar{S}$.



In this case, we can simply compute the amplitude after simulating S and multiply it with the one of $\bar{S}$ to get that of D. To simplify the calculations, let's say that S and $\bar{S}$ contain the same number of T spiders. Using this divide and conquer approach, we can compute the amplitude we are looking for by simulating only $2 \cdot 2^{\frac{t}{2}} = 2^{\frac{t}{2}+1}$ terms which is much better! This situation is unfortunately not very common, but it is possible to make this kind of argument work in many similar situations. For example, in the same situation, but with an edge between S and $\bar{S}$, one could think that the reasoning above does not work anymore.

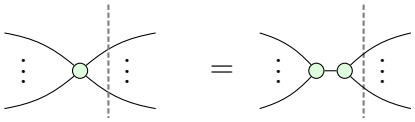


Fortunately, all is not lost. By using the following identity resolution

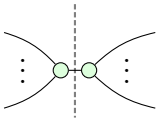


we can cut the problematic edge at the cost of two extra terms. The total decomposition is then done in $2 \cdot 2^{\frac{t}{2}+1}$ terms which is only slightly worse. In general, if

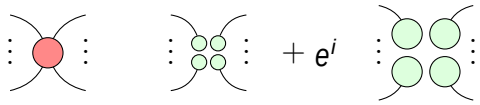
we can separate a graph by a cut of c edges, then we can simulate a diagram by computing $2^c \ 2^{\frac{t}{2}+1}$ terms. As long as the required cut is not too large, this gives a significant advantage. Thus, finding small cuts that separate a graph into two can help in quantum circuit simulation. It turns out that it is even possible to be a little more economical than cutting edges. If we want to cut several edges adjacent to a single vertex, then we can use the following transformation:



one can thus obtain the same partition of vertices by making the following cut



which is done by cutting only one edge. An even better decomposition is to simply use the definition of a spider



Therefore, one can afford not just to cut edges, but to cut vertices which are always more efficient to disconnect graphs. This is the case since we can reproduce an edge cut by cutting an endpoint of every edge in the cut. These types of cuts are often called *vertex separators*.

Definition 9 (Vertex Separator)**.** *Let $G(V; E)$ be a connected graph. We call $C \quad V$ a vertex separator if the graph induced by $V \ n \, C$ is disconnected.*

We will also use vertex cut or simply cut somewhat interchangeably with vertex separator.

A legitimate question is then: can we expect there to be many good vertex separators while simulating real circuits? It turns out that on a fairly common class of circuits, it is possible to demonstrate the efficiency of this approach.

Theorem 2 (Adapted from [21])**.** *Any problem on n qubits that can be solved with $o(n^2)$ quantum gates in a linear nearest neighbour architecture can be solved classically in sub-exponential time.*

Proof. Notice that an underlying ZX-diagram of a circuit where wires can only interact with their neighbour needs to be planar. By the planar separator theorem first proven by Lipton & Tarjan in [23], for any planar graph, there exists a vertex separator of size $O(\sqrt{|V|})$ that partitions the graph into two disconnected subgraphs of at most $\frac{2}{3}|V|$ vertices. Moreover, these cuts can be found in linear time. This naturally gives us a recursive algorithm where we simply use these cuts until the graph is fully disconnected. The time taken by such an algorithm can be bounded by

$$T(|V|) \quad 2^{O(\sqrt{|V|})+1}T \quad \frac{2|V|}{3} \quad + O(|V|)$$

Taking logarithm on both sides gives,

$$\begin{aligned}
\log(T(|V|)) \quad & \log \quad 2^{O(\sqrt{|V|})+1}T \quad \frac{2|V|}{3} \quad + O(|V|) \\
& \log \quad 2^{O(\sqrt{|V|})+1}T \quad \frac{2|V|}{3} \quad + \log\left(O(|V|)\right) \\
& = O \quad \sqrt{|V|} \quad + 1 + \log \quad T \quad \frac{2|V|}{3} \quad + \log\left(O(|V|)\right) \\
& = \log \quad T \quad \frac{2|V|}{3} \quad + O \quad \sqrt{|V|}
\end{aligned}$$

Setting $F(|V|) = \log\left(T(|V|)\right))$ and using the master theorem let us conclude that

$$T(|V|) \; 2 \; 2^{O(\sqrt{|V|})}$$

Thus, since $|V|$ is sub-quadratic, this technique can simulate a circuit in sub-exponential time. □

This argument can be adapted to work with any family of circuits for which there exists an analogue of the planar separator theorem for their underlying ZX-diagrams such as k nearest neighbours architectures or finite-ranged two-qubit gates circuits (see [34]).

The effectiveness of this approach on common circuit classes motivates the search for good separators in the general case. For arbitrary ZX diagrams, we will only consider greedily finding separators, meaning finding the best cut possible at each step of the algorithm. This has two advantages. Firstly, it gives a natural way to incorporate this technique with other ZX stabiliser decomposition techniques as we

will see in chapter 7. Secondly, it makes the problem more tractable. Precisely, our goal is to find a cut set C that optimises the set function

$$f(C) = 2^{|C|} \sum_{S_i \in \Pi_C} 2^{T(S_i)}$$

Where Π_C is the partition into disconnected components induced by the removal of C and T is the T-gate counting function. We call this the *efficient vertex separator problem* (EVSP).

Separating graphs into smaller and somewhat balanced subgraphs while minimising the size of the *interface* between them is a technique widely used to design divide-and-conquer algorithms. Sadly, optimising the size of the interface under different metrics such as the number of edges crossing the partition or the total weight of those edges for some weight functions tends to give rise to NP-Hard problems [32].

In the next section, we will discuss the hardness of the efficient separator problem which will motivate our search for good heuristics presented in the remaining sections of this chapter.

3.1 Hardness

An approach to find good cuts is to try to directly minimise the cost function.

$$f(C) = 2^{|C|} \sum_{S_i \in \Pi_C} 2^{T(S_i)}$$

Most theorems about minimal vertex cuts do not apply to this situation as we need to take into consideration the size of each component. For example, any algorithm solely based on minimal separators through Menger's theorem[1] is doomed to fail as it only optimises the size of the cut. To the best of our knowledge, no good method exists to directly optimise such a function and an exhaustive search would take time of $O(2^V)$. Conversely, no results exist to prove the hardness of such an optimisation directly. However, we can try to approximate it with a function with nicer properties. Firstly, since that, in most instances, the best cut won't be perfectly balanced and since we have exponential growth, we can expect that one of the terms in our cost function will dominate. This leads us to the following approximation.

$$f(C) \quad 2^{|C|} 2^{\max_{S_i \in \Pi_C} T(S_i)}$$

[1] A version of the max- ow min-cut theorem.

Which is equivalent to optimising

$$F(C) = |C| + \max_{S_i \in C} T(S_i) \tag{3.1}$$

This problem is very similar to the (weighted) *Most Balanced Minimum Vertex Cut - Largest Component problem (MBMVC-LC).*

MBMVC-LC

Input: A graph $G = (V, E)$, two vertices x, y, a cut size M and a weight function w.

Goal: Find the vertex separator $C \subseteq V \setminus \{x, y\}$ of size M that minimises $\max_{S_i \in C} w(S_i)$ such that x and y aren't in the same connected component.

This problem studied in [1] has been shown to be NP-hard even in the unweighted version ($w(S_i) = |S_i|$). Moreover, the best approximation that we are aware of is a 2-approximation[2] (also from [1]). We notice that this problem is equivalent to EVSP where we impose a constraint on the size of the cut and the separation of two vertices x and y. Therefore, a solution to EVSP also provides a solution for about half of the MBMVC-LC instances for the obtained cut size. It is, however, not trivial to obtain a reduction in the sense of computational complexity of this problem as we have no control over the cut size in EVSP. If all the cuts obtained by an EVSP oracle are bounded by a constant M, it only allows us to solve MBMVC-LC instances where the size of the cut is smaller than M. This is not an NP-hard problem as it can be solved in polynomial time by brute force. On the other hand, by letting α exceed 1, one can add weight to the balance constraint which must eventually force the size of the cut to grow larger than any fixed constant.

This is a good indication that, assuming that $P \neq NP$ this problem has no solution in polynomial time or at least no simple one. We will, therefore, work on heuristics to obtain cuts that are good in practice.

3.2 Heuristics

We now present different approaches to finding good cuts in arbitrary ZX diagrams by relating the problem to similar graph problems where good approximation algorithms are known. We will first focus on finding small separators that cut graphs into roughly equal parts. This potentially avoids very cheap cuts that cut out a small part of the graph and artificially worsen the optimality of the cuts. This issue is inherent in the

[2] A k-approximation is an approximation at most k times larger than the optimal solution.

use of balanced graph partitioning problems and we will present another way to find small efficient vertex separators that avoids it in the next section.

3.2.1 Balanced Vertex Separators

A problem closely related to finding efficient vertex separators is the one of *Balanced Vertex Separators*.

Balanced Vertex Separators
Input: A graph $G = (V; E)$, 2 $[0; 1)$.
Goal: Find the smallest vertex separator C V such that

$$\max_{S_i 2 \ c} jS_i j \qquad jV j$$

A solution for this problem would be a good candidate for the EVSP because it tries to minimize jCj while also giving us a bound on each term of the sum. Unfortunately, this problem is NP-Hard [24]. However when 2=3 $<$ 1, quasi-linear time approximation algorithms exist that approximate the optimal value within an essentially quadratic multiplicative factor with respect to the size of the optimal value (see [12] for more details of those kinds of schemes). Since this quadratic error might matter to us, we could also use another algorithm with a worse time complexity but a better guarantee. Sadly, no efficient enough libraries are available to try this approach. In practice, recent algorithms have been able to obtain a good approximation for graphs of around 300 vertices in several minutes [29] which is way too slow to use recursively in our application.

3.2.2 (Hyper)graph Partitioning

Most of the work done in recent years on the problem of cutting graphs into roughly equal parts has focused not on vertex cuts, but on edge cuts. Thus, we will study these variants and how they can be used to find good vertex cuts. The edge variant of the balanced vertex separators problem is called the *graph partitioning problem*. In this problem, we want to partition a graph into k parts such that none of the parts is too big. Formally,

Graph Partitioning
Input: A graph $G = (V; E)$, $2 \quad k \quad jVj]$ and $"$.
Goal: Find the minimal edge cut $EC \quad E$ such that

$$\max_{S_i 2 \ E C} jS_i j \quad (1 + ")\frac{jVj}{k}$$

Even though there exist a large body of literature, good approximation algorithms and efficient libraries for this problem and its generalisation to hypergraphs[3], we cannot directly use them in our situation as they minimise the number of edges to remove and not the number of vertices. To overcome this issue, we will present two different approaches in the next subsections.

3.2.2.1 Extracting vertex separators

It is often the case that the smallest edges cut can be done by cutting a good (and even smaller) vertex separator. Therefore, if given a set of edges to cut C_e, we could efficiently extract an optimal set of vertices C_v performing that cut, and we would obtain a good method to generate vertex cuts. Fortunately, this can be done very quickly.

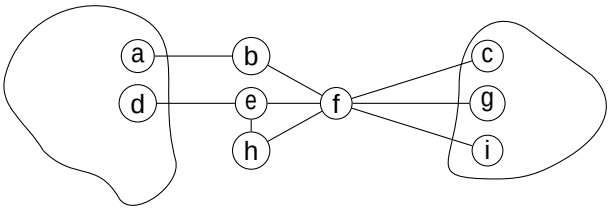
Lemma 3. *It is possible to extract a minimal vertex cut from an edge cut in* $O(jC_e j^{3=2})$

Proof. Since it is useless to cut vertices that are not adjacent to at least an edge in C_e, we can only consider the bipartite subgraph induced by C_e. Finding a vertex cut of the original graph is equivalent to finding a vertex cover of this subgraph. By Kőnig's theorem this is equivalent to finding a maximal matching which can be found in $O(jC_e j^{3=2})$ by the Hopcroft–Karp algorithm. □

By considering a weighted version of the graph partitioning problem, we can even give a weight of 1 for every T gate and 0 for everything else giving us an algorithm that tries to split the T gates, instead of vertices, somewhat equally. The issue with this method is that it results in vertex separators being only locally optimal as using small edge cuts to find vertex cuts is only a heuristic. For example, in the following

[3]Even if this problem is also NP-Hard.

graph



the optimal edge cut would be $\{ab, de\}$ and its induced vertex separator would need to have two vertices while $\{f\}$ is a better vertex cut.

3.2.2.2 Hypergraph Partitioning

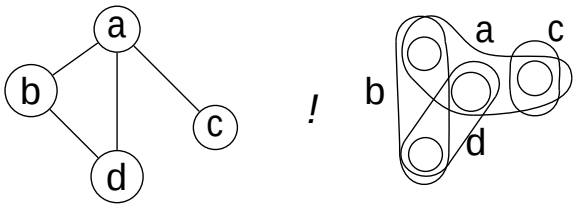
Another approach is to transform our ZX-diagrams in its *line hypergraph*.

Definition 10 (Line Hypergraph)**.** *A line hypergraph* $H = (V', E')$ *of a graph* $G = (V, E)$ *is defined by*

$$V' = E$$
$$E' = \{N_E(v) \ \forall v \in V\}$$

where N_E *means the edge neighbourhood which is simply all the edges adjacent to* v*.*

For example,



Using this transformation, we can use hypergraph partitioning algorithms to obtain vertex cuts on our ZX-diagrams. We can then use ε as a hyperparameter and optimise it through benchmarks. This line hypergraph transformation followed by a hypergraph partitioning similar to a technique used in [21] to obtain contraction trees in tensor networks. In our implementation, we used the KaHyPar library [31] to obtain the cuts. This algorithm is efficient enough to be used recursively[4] without being a bottleneck in the speed of the overall implementation and giving good quality cuts as we will see in chapter 7.

However, there is one main drawback of this approach. Namely, optimising the equal distribution of the vertices in the hypergraph is equivalent to finding a partition

[4] Therefore, an exponential amount of time

in the ZX-diagram where each half has the same number of Hadamard edges and not the number of T gates. If we make the reasonable assumption that edges are somewhat uniformly distributed and that T gates are also uniformly distributed among the gates, then this is not a problem. This issue could also be mitigated by only giving weight to edges adjacent to T gates. Furthermore, the weight can be proportional to the inverse of the degree of the T gates so all T gates have a similar impact on the cost function.

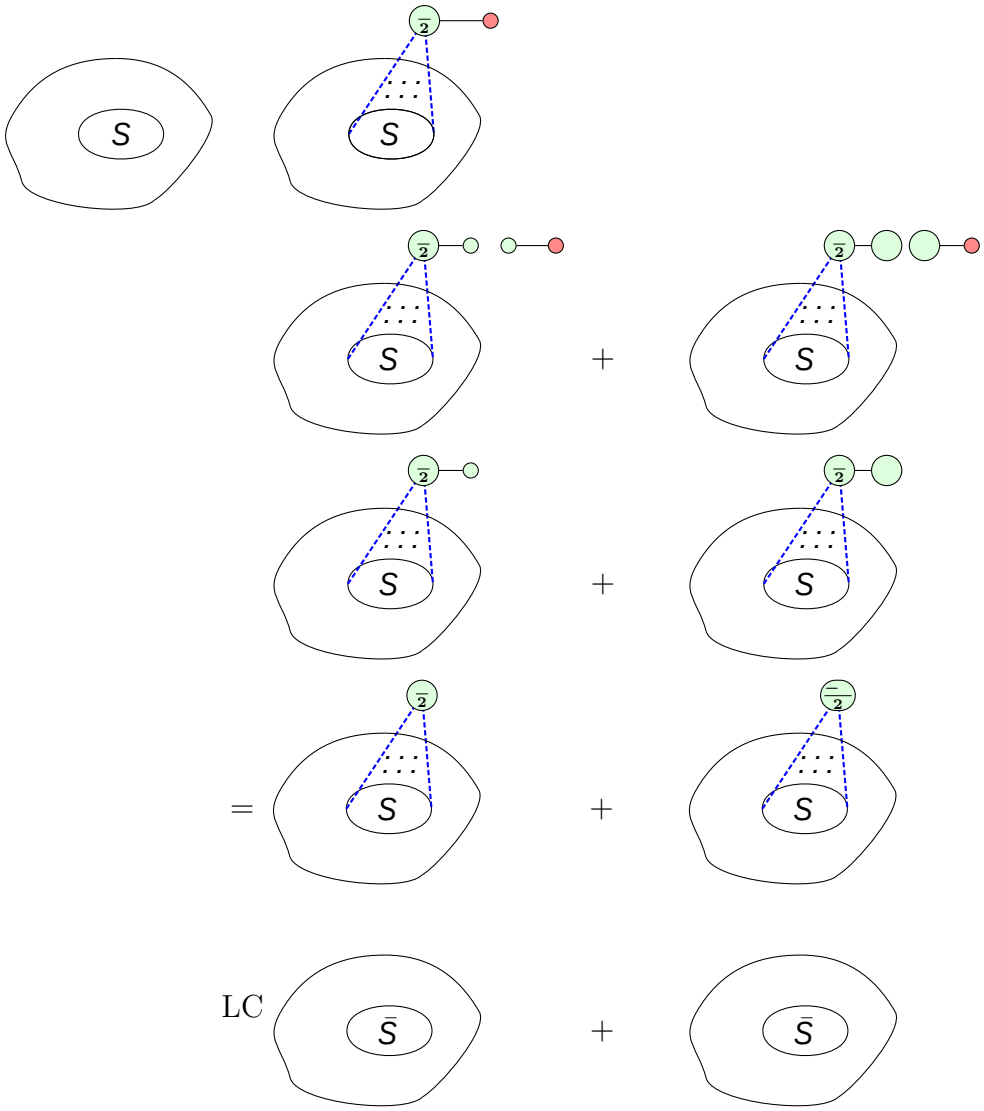
3.2.3 Small Separators

In this section, we will try to address the shortcomings of the previous methods. All of the methods above don't find small vertex cuts that cut the ZX-diagrams into two parts of hugely different sizes. This won't allow us to find some very efficient cuts that would, for example, cut out a chain of length 20 by removing only a single vertex which is equivalent to 0:05.

One way to find such a cut is to try to remove each vertex and count the number of resulting connected components using a depth-first search. Since it is only necessary to do this operation a linear number of times, the cost of this search remains polynomial. This can be transformed into a more efficient algorithm by first looking for an articulation point in the graph which can be done in $O(jVj + jEj)$. For larger cuts, we can use the min-cut/max-flow theorem to find vertex separators. To do so, we must consider each pair of vertices x and y of our graph and find the smallest cut separating them using the Ford-Fulkerson algorithm. It is not necessary to repeat this operation jVj^2 times because the same separator optimally separates x' and y' for each pair of vertices contained in the respective related components of x and y after having separated them. Moreover, it is possible to stop this algorithm prematurely when the cut found is too large. If we impose a small finite bound on the size of the cut, we can find all these minimal cuts in $O(jVj^3jEj)$. One problem with this approach is that if the cut size exceeds the degree of x or y then trivial cuts are found even if better cuts exist. It is possible to prevent this kind of effect and it is on this kind of technique that several optimisation methods of the balanced vertex separator problem are based. The main drawback of this approach is that it tries to only minimise the size of the cut and not the balanced. However, considering different initial vertices gives some diversity in the solutions found.

3.3 Subgraph Complement Cuts

At the beginning of this chapter, we generalised edge cuts to vertex cuts. There is another way to generalise edge cuts by making cuts by induced subgraph completion. Indeed it is possible to perform the complementation of any induced subgraph at the cost of two terms. Let S be an induced subdiagram, then



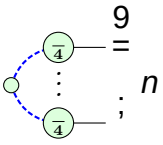
A special case of this operation is to remove cliques. Since an edge is a 2-clique, this generalises edge cuts. As we only noticed this identity shortly before the submission of this work, we postpone the analysis of this technique to future work. However, another implication of this operation is that we can complement the entire ZX-diagram for the cost of two terms which could lead to more ZX simplifications and bound the edge density of the diagram.

Chapter 4

Finding New Decompositions

A considerable amount of effort has been put into discovering new and more efficient decompositions for several types of states. A particularly interesting class is the one of *cat states* named after Schrödinger's cat.

Definition 11 (Cat State)**.** *A cat state is a state formed by a Clifford spider only attached to T gates.*

$$|cat_n\rangle := \frac{1}{\sqrt{2}}$$ 

It is through the study of this type of state that the algorithm with the smallest provable complexity for the stabilizer decomposition was discovered. Indeed, in [20], it was shown that one can transform a $|\text{cat}_6\rangle$ decomposition into a partial decomposition of the injection of five T magic states. In doing so, they showed that it is possible to simulate an arbitrary diagram in $O(2^{\ t})$ diagram where 0:396. Moreover, several cat state decompositions with much better have been discovered. A by-product of the ZX simplification procedure used in between every round of decomposition is that no two Clifford spiders can be adjacent since that would lead to a pivot (which would remove the two spiders). Moreover, this procedure removes every Clifford spider with an angle other than 0 or with the local complementation operation. Finally, with the use of the copy rule, it is possible to get rid of any angle on a Clifford spider. Thus, at every step of the algorithm, every Clifford spider is the centre of a cat state. Even if these are not guaranteed to be present in an arbitrary diagram, they are quite common and searching for them and using their more efficient decomposition turns out to be a high-yielding method.

To the best of our knowledge, we list the best-known cat state decompositions from [4].

$\|\text{cat}_n\rangle$	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
$_n$	0.333	0.25	0.317	0.264	0.369	0.323	0.352	0.317	0.432	0.366	0.366	0.34

Table 4.1: Decomposition for small cat states

In the general case, the best-known construction offers a decomposition of $|\text{cat}_{4l+2}\rangle$ into 3^l terms. From this, we can infer the other cases with the inequality

$$(|\text{cat}_n\rangle) \qquad (|\text{cat}_{n+1}\rangle)$$

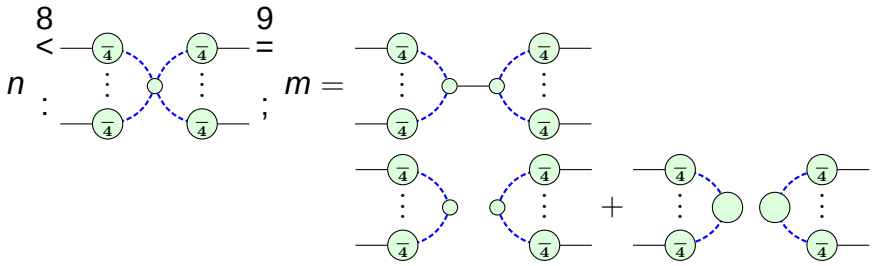
Here and for the rest of this chapter, we use $_n$ to denote the of $|\text{cat}_n\rangle$. In the next section, we will introduce a simple technique that allows combining cat state decompositions to obtain new decompositions which will let us improve the decomposition for some cat states.

4.1 Combining cat states decompositons

We introduce two operations that allow us to express a large cat state using several small cat states.

4.1.1 Cuts

Through a resolution of the identity and spider fusion, we can transform a $|\text{cat}_{n+m}\rangle$ into a $|\text{cat}_n\rangle$ and a $|\text{cat}_m\rangle$ at the cost of two terms.



We can then push through the phases from the second term to obtain cats. Therefore if we have a decomposition of $|\text{cat}_n\rangle$ into x terms and a decomposition of $|\text{cat}_m\rangle$ into y terms we have decomposition of $|\text{cat}_{n+m}\rangle$ into $2xy$ terms. This implies that

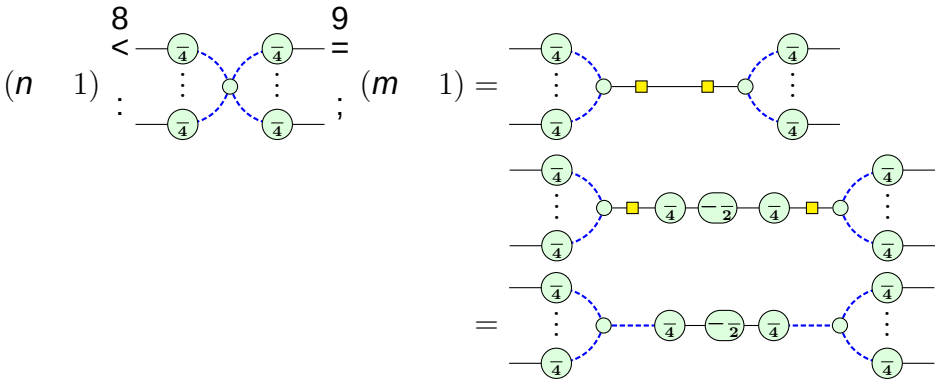
$$_{n+m} \quad \frac{\lg(2xy)}{n+m} = \frac{1+\lg(x)+\lg(y)}{n+m}$$

Even if this allows us to find new decompositions for cat states, it, unfortunately, does not allow us to find a decomposition better than its constituents. This is because,

$$\begin{aligned}\frac{1+\lg(x)+\lg(y)}{n+m} &> \frac{\lg(x)+\lg(y)}{n+m}\\ &\geq \min\left(\frac{\lg(x)}{n};\frac{\lg(y)}{m}\right)\\ &= \min\left(f_n;\ f_m\right)\end{aligned}$$

4.1.2 Splits

Another way to decompose a cat state into smaller cat states is to use the *splits* operation.



Here on the left, we have a $|\mathrm{cat}_n\rangle$ while on the right we have a $|\mathrm{cat}_m\rangle$. Weirdly enough artificially adding T gates this way helps us find better decomposition for some cat states. This construction is similar to one from [4].

If we have a decomposition of $|\mathrm{cat}_n\rangle$ into x terms and decomposition of $|\mathrm{cat}_m\rangle$ into y terms we gain again simply compute the of this decomposition

$$(n+m-2) = \frac{\lg(xy)}{n+m-2}$$

We can again compare the of this new decomposition with the of its constituents.

$$\begin{aligned}\frac{\lg(xy)}{n+m-2} &\geq \frac{\lg(x)+\lg(y)}{n+m-2}\\ &> \frac{\lg(x)+\lg(y)}{n+m}\\ &\geq \min\left(\frac{\lg(x)}{n};\frac{\lg(y)}{m}\right)\\ &= \min\left(f_n;\ f_m\right)\end{aligned}$$

4.1.3 Combining both

Even if, as n goes to infinity, using these two operations yields $_n$ that gets worse[1] these constructions can lead to better decompositions for specific cat states.

To do this, we will optimally combine these two operations. This is can be done easily because the optimality principle applies[2] so we can proceed by dynamic programming. We then obtain an algorithm which obtains the best possible result for $|\mathrm{cat}_n\rangle$ in $O(n^2)$.[3]

Running this algorithm gives us improvements for half of the instances larger than 10 as can be seen in figure 4.1.

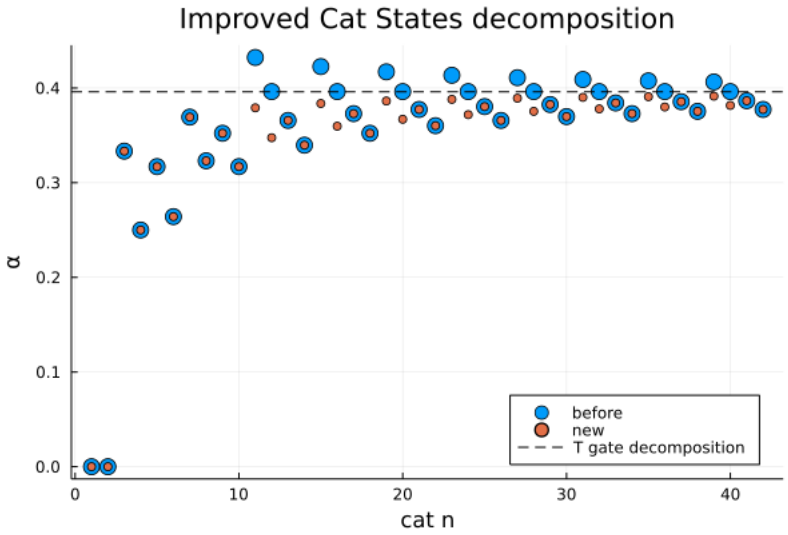


Figure 4.1: Single copy cat state decomposition

For example, splitting a $|\mathrm{cat}_{12}\rangle$ into a $|\mathrm{cat}_4\rangle$ and a $|\mathrm{cat}_{10}\rangle$ gives a decomposition into 18 terms which improved on the previous best which was 27. Even if the gains are only modest, the technique we have created can be used in other contexts, as we will see in the next sections.

4.2 Star States

Since the cat state decompositions that have been found are quite good, it is interesting to see if they can be used for similar states. To this end, we introduce what

[1] In fact, we have that $_n \overset{n\to\infty}{?} 0.396\ldots$ just like in [4].

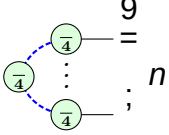
[2] Since an optimal decomposition must be made of optimal sub-instances.

[3] This is not very signi cant, because this algorithm does not need to be executed each time we want to do simulation but it is nice to know that this scales nicely and could easily be rerun if someone ever nds a new operation.

we call a *star state*.

Definition 12 (star state)**.** *A star state is a T gate surrounded by other T gates.*

$$|Star_n\rangle := \left\{ \begin{array}{l} \\ \\ \end{array} \right. \quad ; \ n$$

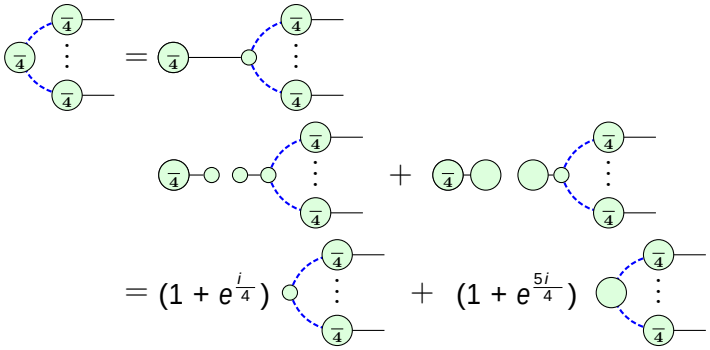


These states are in some way dual to the cat states. This is the case because if a (non-trivial) Clifford+T diagram contains no cat states, it must necessarily contain star states and vice versa. Thus, finding good decompositions for these states complements in a surprisingly efficient way the algorithm of [20] which had to resort to the injection of magic states as soon as there were no cat states left.

We will note $\quad_n$ the $\quad$ of $|\text{star}_n\rangle$. As for the cat states, we will present different operations allowing us to generate star state decompositions from smaller ones.

4.2.1 Catification

The simplest technique to obtain a decomposition of a star state is to use the spider fusion and then a wire cut to transform a star state into two cat states.



If we have a decomposition of $|\text{cat}_n\rangle$ into x terms then we have decomposition of $|Star_n\rangle$ into $2x$ terms. This gives us

$$_n = \frac{\lg(2x)}{n+1}$$

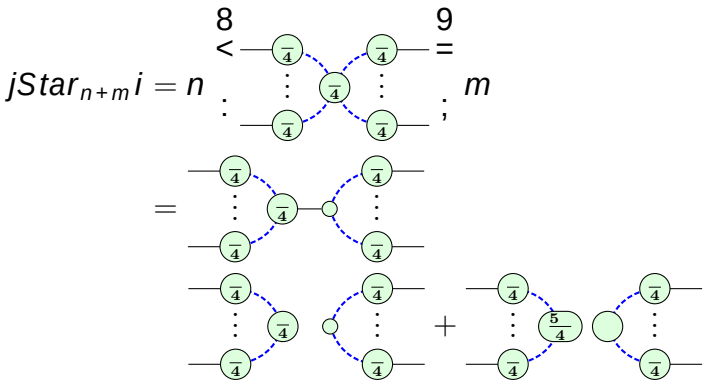
Which we can compare to $\quad_n$

$$= \frac{\lg(x)+1}{n+1}$$
$$\frac{\lg(x)}{n}$$
$$= \quad_n$$

Note : In the last inequality, we used the fact that $\lg(x) \quad n$. This is the case because even with the decomposition from the definition (the one that gives us the biggest x for a given n), we have that $\lg(x) = \lg(2^n) = n$.

4.2.2 Cuts (Star version)

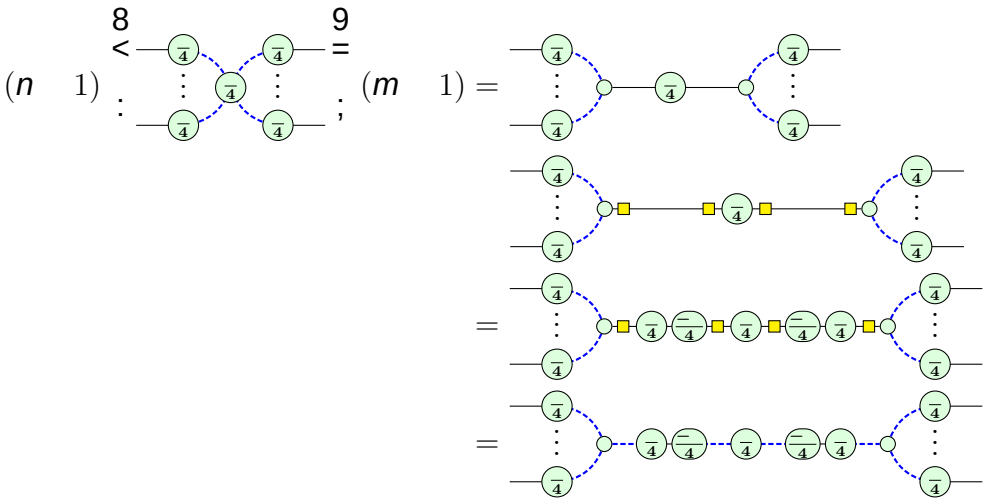
Much like the cats, we can decompose a big star state into two. However, this time around, doing so gives a star and a cat state.



So with a decomposition of $|star_n\rangle$ into x terms and a decomposition of $|cat_m\rangle$ into y terms, we have a decomposition of $|star_{n+m}\rangle$ into $2xy$ terms.

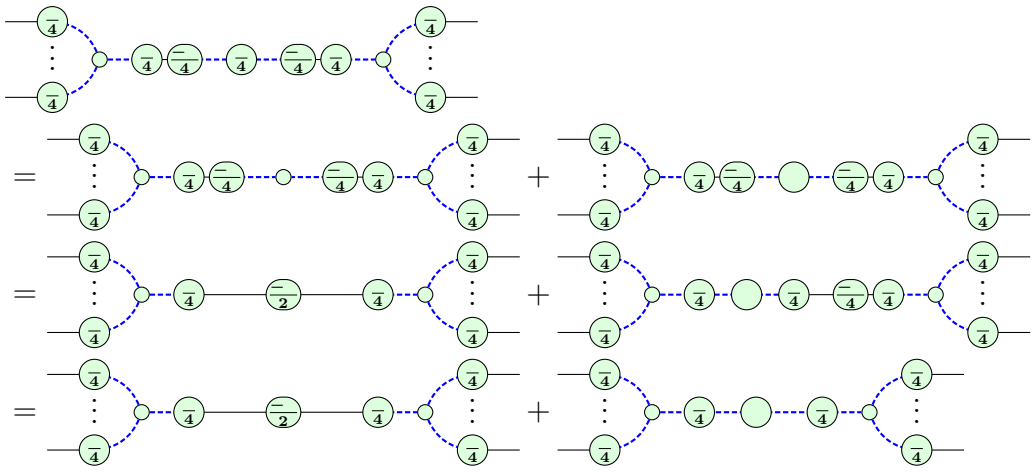
4.2.3 Splits (Star version)

Here again, we can find an analogue of the split operation for star states.



We could stop here and use it as a partial decomposition (where some T gates remain) but we can also use the fact that the remaining T gates form a $|Star_2\rangle$ which can be

decomposed into two terms through catification.



So with a decomposition of $|\mathrm{cat}_n\rangle$ into x terms and a decomposition of $|\mathrm{cat}_m\rangle$ into y terms, we have a decomposition of $|\mathrm{star}_{n+m-2}\rangle$ into $2xy$ terms.

4.2.4 Results

Using these transformations leads us to find good decomposition for some stars. For example, catifiying $|\mathrm{star}_2\rangle$ and $|\mathrm{star}_6\rangle$ gives us decomposition in respectively 2 and 6 terms or, in other words 0.333 and 0.369. We leave the full list of obtained this way in the appendix as we will improve upon these in the next section.

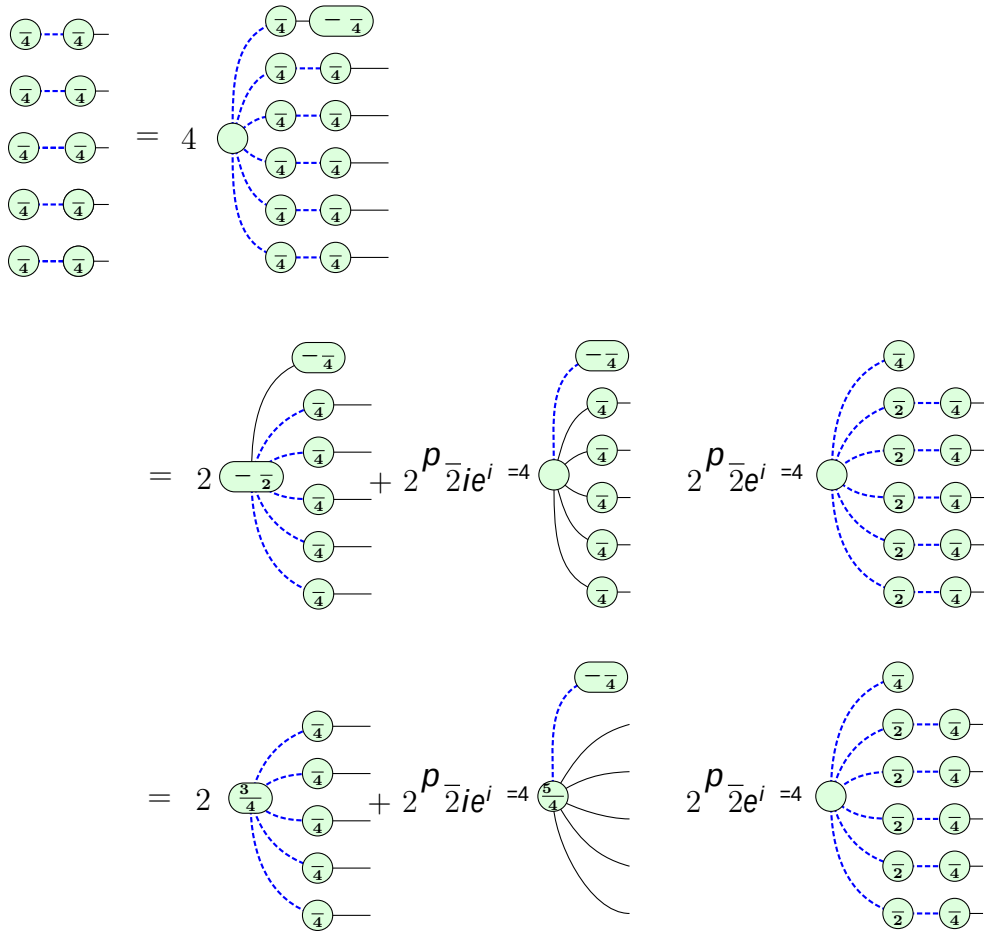
4.3 Multiple Copies of Cats and Stars

In the previous section, we looked at decompositions of $|\mathrm{cat}_n\rangle$ and $|\mathrm{star}_n\rangle$. It turn out that it is fruitful to also look at decompositions of $|\mathrm{cat}_n\rangle^{\otimes k}$ and $|\mathrm{star}_n\rangle^{\otimes k}$. One trivial bound is that

$$|\mathrm{cat}_n\rangle^{\otimes k} \qquad (|\mathrm{cat}_n\rangle)^k \quad \text{and} \quad |\mathrm{star}_n\rangle^{\otimes k} \qquad (|\mathrm{star}_n\rangle)^k$$

This is because the stabilizer rank is sub-multiplicative which can be seen in this case by simply applying a decomposition of $|\Psi\rangle^{\otimes k}$ times to obtain a decomposition for $|\Psi\rangle^{\otimes k}$. One example where $|\mathrm{star}_n\rangle^{\otimes k} < (|\mathrm{star}_n\rangle)^k$ is with $k = 5$ and $n = 1$. This is because we can use the decomposition of $|\mathrm{cat}_6\rangle$ in a similar way to how the

decomposition of $|T\rangle^{\otimes 5}$ was found in [20].



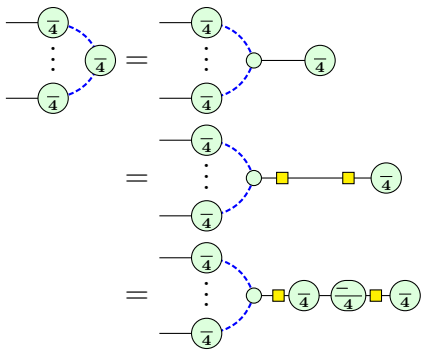
If we stop here, we obtain a partial decomposition that removes 4, 8 and 4 T gates in each term respectively. Using the calculation for partial decomposition explained in the previous chapter, we can compute that 0.3179. However, we can go further by noticing that the last term can be transformed into a $|\text{cat}_6\rangle$ by using five local complementations. We can then apply the decomposition of $|\text{cat}_6\rangle$ into three terms to obtain a partial decomposition for which 0.2999. In fact, this strategy can be used as long as two $|\text{star}_1\rangle$ are present. Indeed we can use this idea to decompose $|\text{cat}_1\rangle^{\otimes 2} \otimes |T\rangle^{\otimes 3}$ leading to 0.347.

Similarly, a decomposition of $|\text{star}_1\rangle^{\otimes 6}$ into 8 terms and a decomposition of $|\text{cat}_3\rangle^{\otimes 2}$ into 3 terms were recently be found by Tuomas Laakkonen [22] leading to 0.25 and 0.264 respectively[4]. These new decompositions can be used to find other ones for other cats and stars with the same algorithm explained before. To use these newly found decompositions to their full extent, we add two more operations to our search.

[4]Since these decompositions are not published anywhere, we include them in the appendix.

4.3.1 $|\text{star}_1\rangle$ Split

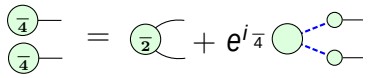
A special case of the star split operation can be done in a more efficient way when $m = 1$.



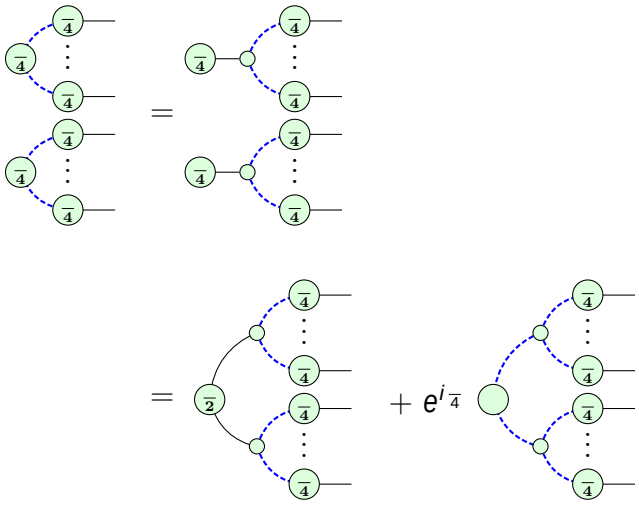
This has both the benefit of introducing fewer T gates than the general split operation and leaving a $|\text{star}_1\rangle$ behind which as we know has a good decomposition (when multiple copies are present). Therefore, if we do this transformation on 6 different instances, we are guaranteed to be able to follow it by a $|\text{star}_1\rangle^{\otimes 6}$ decomposition.

4.3.2 Star Fusion

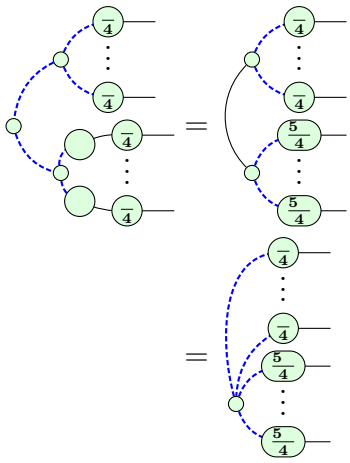
Using the $|T\rangle^{\otimes 2}$ decomposition introduced in chapter 3,



it is possible to partially fuse two stars into a cat at the cost of two terms.



We will focus on the second term which is where the fusing of the stars happens.



Using this technique with, for example, two $|\text{star}_3\rangle$ we obtain a partial decomposition with 0.365. We will discuss how this operation can be generalised to other types of states in the last section of this chapter.

4.3.3 Results

With the technique mentioned above applied to cats, we obtain improved decompositions even for small cat states as can be seen in figure 4.2.

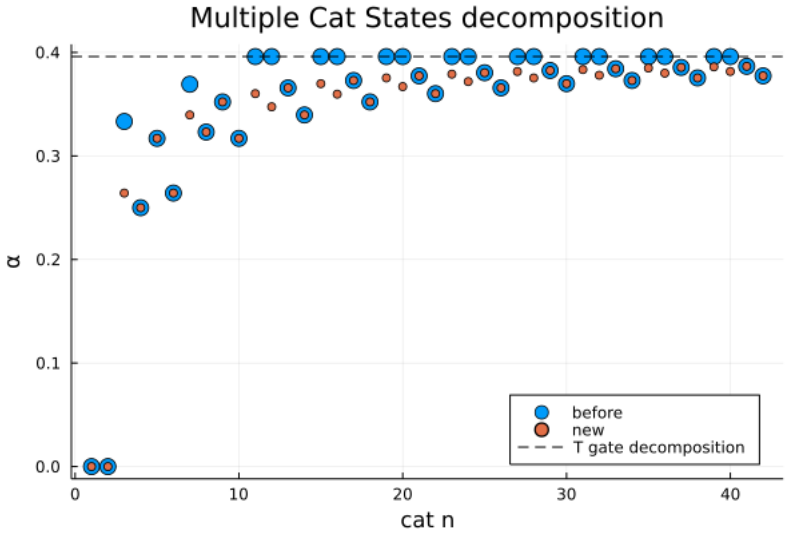


Figure 4.2: Multiple copies cat state decompositions

Note : Since we assume that we have $|\text{star}_n\rangle^{\otimes k}$ for k large enough to work, we can always resort to using the T magic states injection which is why no can be higher than 0.396.

Similarly, we obtain improvements in the decompositions of numerous star states. Results for star states are presented in figure 4.3. We emphasise that for these states,

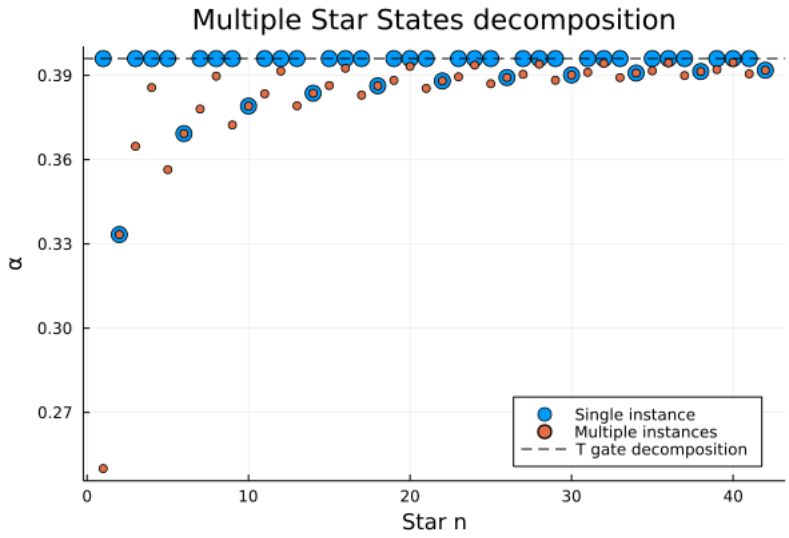


Figure 4.3: Multiple copies stars state decompositions

no better decomposition than the one using magic state injection (0:396) was previously known.

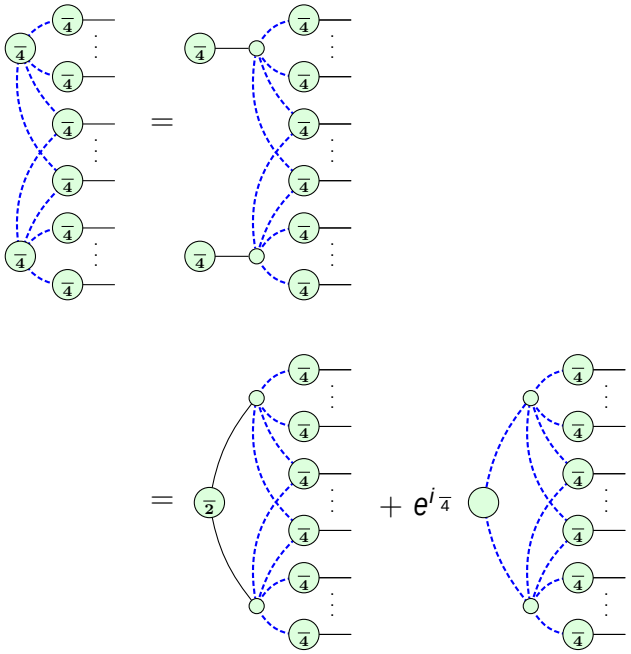
Those results imply that, for instance, if we could simplify the whole diagrams only using multiple copies of cat states and stars states of degree less than 38 and 7 respectively then we could simplify the whole diagrams with an bounded by 0:3856 (the for $|\text{star}_4\rangle^{\otimes k}$). Another interesting aspect of these results is that they hint at the fact that highly connected diagrams are harder to simulate in general (as they do not have any small cat or star state) which was also the worst case of the techniques presented in the previous chapter. We will try to address this issue in the next section.

4.4 Decomposition of Large Stars

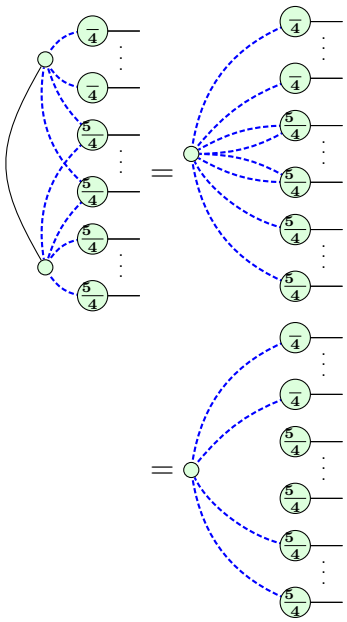
Earlier in this chapter, we introduced star fusion to find good decompositions of $|\text{star}_n\rangle^{\otimes k}$. It is easy to see that the same derivation holds when we fuse stars of different sizes. For example, we found a decomposition of $|\text{star}_4\rangle^{\otimes k} \quad |\text{star}_{10}\rangle^{\otimes k}$ with 0:378 which is better than both our decompositions for $|\text{star}_4\rangle^{\otimes k}$ and $|\text{star}_{10}\rangle^{\otimes k}$. Even when dealing with single instances combining a $|\text{star}_1\rangle$ and a $|\text{star}_5\rangle$ gives a partial decomposition for which 0:365 which is an improvement over the decom-

positions of $j\text{star}_1 i$ and $j\text{star}_5 i$ considered separately. All these decompositions are actually a special case of a more general statement.

In a variety of cases in practice, stars share T gates and aren't disjoint. Let us observe what stars fusion gives us when this happens.



We will again focus on the second term which is where the fusing of the stars happens.



We see that the degree of the resulting cat only depends on the symmetric difference of the stars and not on their initial degree. Depending on the size of the symmetric difference, this decomposition can be very beneficial. We found that whenever

the difference is even, we were able to beat magic state injection using this strategy. In table 4.2, we list the first ten sizes where this method leads to better .

Symmetric di erence	2	4	6	8	10	12	14	16	18	20
	0:347	0:381	0:365	0:386	0:373	0:388	0:378	0:390	0:381	0:391

Table 4.2: List of the size of symmetric difference which improves over magic state injection

This approach allows us to fix a weakness that is common to all of the previous techniques introduced to speed up our algorithm, namely that they do not work well on highly connected ZX-diagrams. This is the case because high connectivity rules out the existence of small separators and only gives us high degree cats and stars states for which our decompositions are only marginally better than magic state injection. However, when a diagram is highly connected, some stars must necessarily share several T gates which makes it possible to use high-degree star states to obtain a good decomposition. They are also some other benefits from using this decomposition that are not captured by the value of which makes this technique work surprisingly well in practice. We will discuss those in the next chapter.

Chapter 5

Where to Apply Decompositions

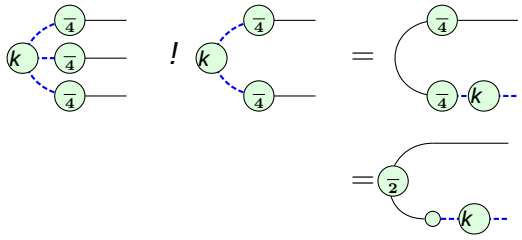
In the previous chapters, we neglected one important aspect to assess the quality of decompositions. Although we found bounds on the number of T gates guaranteed to be removed by a decomposition, we did not discuss how much they help to find further simplification of the diagrams through the rules of ZX-calculus and how they might help to find instances of small stars and cats in the next iterations. If a decomposition isn't theoretically as efficient as another one but leads to more cancellation then that decomposition should be preferred. For example, this was the case in [19] where the seven T gates construction of a Toffoli gate was preferred to the four T gates one. It is impossible to conclude that one decomposition is always better than another and should be prioritized all the time. In fact, when there are several possible instances of a decomposition within a ZX-diagram, it happens that some of them are vastly superior. This is because, in addition to the T gates removed directly by a replacement, many others can also be removed by the subsequent simplification of the diagrams depending on the surroundings of the instance. The difference between a good and a bad instance can sometimes have a major impact on the efficiency of the simulation. For example, we saw earlier if a magic state injection of 5 T gates happens to choose the centre of $|\mathrm{star}_1\rangle$ then that leads to a $|\mathrm{star}_1\rangle^{\otimes 5}$ decomposition with an 0:2999. A natural question then is: how do we choose which instance of a decomposition to use?

A fairly simple technique, that was previously tried without success, is to try several possibilities and take the best one. The problem is that several decompositions can be applied in a very large number of ways most of which don't lead to much further simplification if any. For example, the decomposition that takes any five T gates can be applied $\binom{5}{t}$ times. Thus, finding good heuristics for applying decompositions to the best possible locations turns out to be a problem of interest.

5.1 Where to apply the trivial decomposition?

Given the somewhat erratic behaviour of ZX-diagrams through several rounds of simplification and decomposition, we will focus on the number of cancellations after a single round of simplification. With this simplification, we have found a simple heuristic for applying the trivial (and theoretically least efficient) decomposition that surprisingly turns out to be, in practice, frequently more efficient than any other decomposition.

The only simplifications one can initially obtain after this decomposition come when the removed T gate was part of a $|\text{cat}_3\rangle$ or when it was the only gate stopping a *gadget-fusion*. Since the second scenario is somewhat uncommon, we will try to maximise the effect of the first one. The $|\text{cat}_3\rangle$ cancellation happen because removing a T gate in this way transforms a $|\text{cat}_n\rangle$ into a $|\text{cat}_{n\ 1}\rangle$ and $|\text{cat}_2\rangle$ are Clifford. Indeed, one can use the commutation rule of $|\text{cat}_3\rangle$ and the spider merge to get rid of the other two T gates.



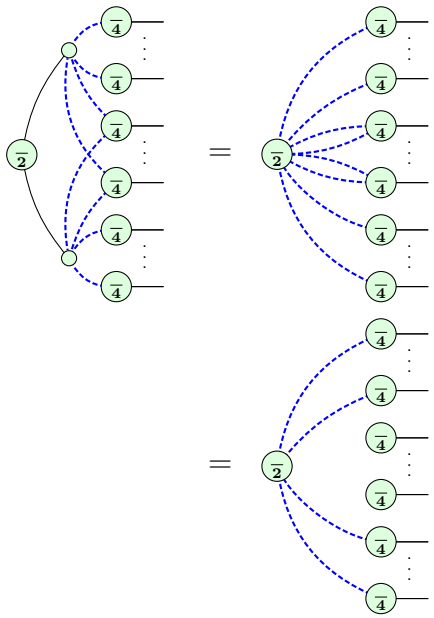
Therefore, after removing a single T gate, we can decrease the degree of all adjacent cat states, in addition, to completely simplifying all adjacent $|\text{cat}_3\rangle$. In addition to this, there are several cases where other rules can be applied such as Hopf's rule (which can lower the degree of Cats) as well as the pivot which offers other opportunities for simplification. This means that after this first round of simplifications, it is very likely that other $|\text{cat}_2\rangle$ will be formed. These $|\text{cat}_2\rangle$ will in turn be simplified creating new simplification opportunities and new $|\text{cat}_2\rangle$ and so on. This rippling effect makes the decomposition of some T gates can reduce the T-count significantly. There is no limit to the number of T gates that can be removed with this decomposition and we found instances from ZX-diagrams used for SAT counting where a single decomposition removed 286 T gates which is equivalent to 0:0035.

Despite the potential to find very good decompositions, the cost of trying all the T gates at each step of the algorithm is prohibitive because of the time required to do the ZX-simplification. Therefore, it is desirable to find a good heuristic to obtain a good decomposition quickly. One simple idea is to maximise the effect of the first round of simplification by selecting the T gate that is in the largest number of $|\text{cat}_3\rangle$. Intuitively, maximizing the size of the first wave should also create bigger ripples.

This heuristic is fast to compute, as it can be done by only considering each edge at most once. So we can compute the whole thing in $O(jEj)$.

5.2 Where to apply star fusion?

In section 4.4, we explained how star fusion can be used to find efficient partial decomposition even when no small stars and cats are left in the diagram. Since the resulting of these are not so great compared to some of the cats, these are in some sense the "last resort" before falling back to magic state injection. Thus, these are mostly used in the context where the diagram is densely connected and no better decomposition where found. Even if we cannot guarantee the removal of further T gates, it is still possible to apply star fusion in a way that *simplifies* the ZX-diagram the most. Specifically, we can try to use it in a way that removes as many edges as possible. Previously, we only looked at what happens in the half of the partial decomposition where we were creating a star. Let's now look at the second term.



Here again, the Hopf rule will detach the shared neighbourhood of the stars in addition to creating an opportunity for the ZX simplification to apply local complementation. So the number of edges removed is at least twice the size of the shared neighbourhood in both terms. Therefore, a good heuristic to remove the most edges is to maximize the size of the shared neighbourhood as it.

We also note that the simplification algorithm will remove the centre spider through local complementation. It is not clear if this will help or not as the local complementation may add some edges. This should not have a significant impact

as the size of the symmetric difference of the neighbourhood is bounded and since this pivoting only happens in one of the terms.

Removing the most edges is a sensible thing to do as it lowers the degrees of the stars and cats as well as creates better opportunities to find good vertex separators. As we will see in chapter 7, the edges density of the graph has a great impact on the speed of the algorithm so this heuristic gets us further from it.

Chapter 6

Combining All the Decompositions

Previously used methods in the literature imposed a priority order between several decompositions based on their theoretical . As we have seen in the previous chapter, this kind of approach neglects the structure of the underlying diagram and misses several possibilities of applying very good decompositions. Moreover, this type of approach rarely uses certain decompositions and thus does not take advantage of the full extent of the efforts that have been made to find a wide variety of decompositions. Another disadvantage of this kind of approach is that there is no natural way to incorporate graph cuts. On the other hand, this kind of approach allows obtaining good theoretical guarantees on the efficiency of the algorithm, which is not the case if one uses theoretically worse decompositions. We, therefore, propose an approach that attempts to use decompositions more adapted to the structure of the diagram and that incorporates graph cuts while maintaining bounds on the time complexity.

6.1 Our algorithm

We start by finding good instances of a set of decompositions. To do so, we use heuristics as discussed in the previous section or simply the first instance found if such heuristics do not exist. We then attempt to apply each of these decompositions and compute their efficiency. The metric used is essentially a version of which is better suited for disconnected graphs and takes into account ZX simplifications.

As opposed to the *theoritical* guaranteed by the decomposition, we can compute the *effective* . The calculation for the *effective* is the same as the one for the theoretical one but this time using the real number of T gates cancelled (after simplification). Therefore for non partial decomposition,

$$_{e} = \frac{\lg(\#\text{ terms})}{\text{T gates removed}}$$

We can also adapt the discussion about the efficiency of partial decomposition to incorporate cancellation after simplification. Naturally, we have that $\quad_e \quad$ for every decomposition. Although this metric works for most decompositions, it fails to represent the true efficiency of graph cuts because it fails to take into account the divide-and-conquer nature of the problem when multiple disconnected components are involved. This is the same reason why stabiliser rank also fails to take into account disconnected graphs. The only bound we get for two disconnected state $j\Phi i$ and $j\Psi i$ is that

$$(j\Phi i \quad j\Psi i) \qquad (j\Phi i) \quad (j\Psi i)$$

which doesn't tell the whole story as we can compute $hx_{1;:::;k}j\Phi i \quad hx_{k+1;:::;n}j\Psi i$ in time that scales linearly (and not multiplicatively) with

$$(j\Phi i) + \quad (j\Psi i)$$

which is the whole point of doing graph cuts in the first place. To solves this issue, we can, for the sake of computing the complexity of a cut, see every connected component as a term in a partial decomposition. Thus, if a diagram has k connected components after a cut of size C, and each component i has a_i less T gates then the initial diagram, we can compute the of a cut as the only positive root of

$$x^{a_{max}} \quad 2^C \sum_{i=1}^{k} x^{a_i + a_{max}}$$

Using this metric helps to keep the theoretical guarantee that we had without using cuts. Indeed, since our set of decompositions includes the best theoretical decompositions, we are guaranteed that the theoretical bounds also hold for our algorithm. Furthermore, by not restricting ourselves to a predefined priority order, it turns out that our algorithm only rarely uses the theoretically best decomposition. Since finding each instance and computing their efficiency takes a polynomial time, this preprocessing has little impact on the total exponential efficiency of the algorithm. On the other hand, we limit ourselves to a set of decompositions that are not too large and that tend to work well in practice (while including the best theoretical decomposition) to keep the cost of this preprocessing step to a minimum. We also mention that graph cuts take significantly more time to compute as it is the only decomposition that takes more than quadratic time to find.

Chapter 7

Numerical Experiments

7.1 Implementation

Our implementation is written in Rust and is available at `https://github.com/Codsilla/ZX-stabiliser-simulator`. It includes all the decompositions of cat and star states up to degree 10, the decompositions of $|\text{star}_1\rangle^{\otimes 5}$ and $|\text{cat}_3\rangle^{\otimes 2}$, the merging of stars according to their symmetric difference up to 10 and the graph cuts found via hypergraphs and vertex cuts extraction. As we decompose the diagrams in a depth-first fashion, the memory consumption of our algorithm is negligible as it grows linearly with the number of T gates considered.

For the representation and simplifications of the diagrams, we used Quizx (a rust port of PyZX [17]). We also uses Kahypar [31] and Kahip [30] to find cuts in (hyper)graphs.

7.2 Benchmarks

We tried our algorithm on multiple families of circuits to assess the efficiency of our approach and to compare it with the state-of-the-art from [20] implemented in the Quizx library. For every size of circuit considered we computed 10 different instances. We also imposed a time limit to the algorithm from [20] to 45 min. We ran all circuits on a dedicated computation server (24-core Intel Xeon E5-2667, 2.90GHz). We mention that the Quizx algorithm doesn't seem to utilise the full power of multi-threading which somewhat skews the results. However, this does not impact the scaling of both algorithms which is what matters. We also mention that our algorithm falls back on Quizx's algorithm for T-counts under 20 to save the overhead of having to compute all our decompositions for small diagrams.

7.2.1 Random IQP

Instantaneous quantum polynomial-time (IQP) is a family of circuits built out of diagonal gates[1] and two layers of Hadamard gates.

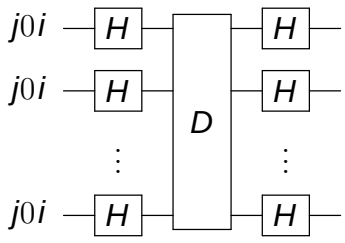


Figure 7.1: Shape of a generic IQP circuit

These circuits are instantaneous in the sense that every gate in the diagonal section commute so there is no order in which gate should be done and, in physical implementations, can be seen as happening at the same time. These circuits have been studied for their possible uses to demonstrate quantum supremacy [25] because they have proprieties that make them possible to implement in the near term future while still being unlikely to be classically simulated in polynomial time. It has been shown that in the worst case, those circuits are hard to simulate unless the polynomial hierarchy collapses to the third level [6]. Moreover, it has been shown that even restricting this class further by imposing a sparsity condition on the circuit maintains this worst-case hardness [7]. It has also been shown in [6] that under mild assumptions, IQP circuits are also hard to simulate in the average case. These hardness results still hold when restricted to Clifford+T circuits. Within this fragment, the diagonal part of the circuit can be simplified to a layer of power of T gates followed by a sequence of powers of controlled S gates.

To generate random IQP circuits, we randomly generated both the power of the T gates and the number of controlled S gates between each qubit.

7.2.2 Random Clifford+T

Random Clifford+T circuits arising from exponentiated Pauli unitaries is a class of circuits previously used to benchmark the efficiency of stabiliser decomposition algorithms [19]. These circuits are interesting as they contain barely any small cat and star states. Moreover, they are generally densely connected meaning that no

[1] A gate is said to be diagonal if its matrix representation in the computational basis is diagonal.

efficient cut can be found. Therefore, these circuits will test how much star fusion helps in this kind of instance.

We decided to reuse the same generation method and circuit size from this article to compare our algorithm fairly. Readers interested in the generation of the random instances should refer to [19] (section 4.1).

7.2.3 Other Families

In the literature, other families of circuits have also been considered for benchmarks. One such family is the *hidden-shift family* considered in [3], [4], [19] and [20]. Surprisingly, recent improvements to the simplification routine from Quizx completely trivialised the simulation of this family. Indeed, we haven't been able to find a single instance where any T gate was left after an initial round of simplification. Thus, no stabiliser decomposition was needed to simulate these circuits and we were able to find the hidden shift of all the circuits previously considered in way less than a second. We conjecture that this class of circuit can always be simulated in polynomial time (since the simplification algorithm runs in $O(n^3)$).

We also tried to simulate sets of circuits coming from real circuits found on `https://github.com/Quantomatic/quizx/tree/master/circuits`. This dataset includes adders, multipliers, Grover circuits, etc. Again, for these circuits, we found that simulations were completely trivialised by ZX simplifications.

7.3 Results

As mentioned in [19], the number of T gates removed by the first simplification has a great impact on the number of the total runtime of the algorithm. Therefore, we also include results comparing time and the remaining T-count after the initial simplification.

7.3.1 IQPs

For random IQP circuits, we noticed that small cat states were almost always present at every step of the decomposition which was beneficial for our approach. Using linear regression, it is possible to estimate the value of these two algorithms. Before simplification, our algorithm has an $0{:}0252$ while the previous algorithm had $0{:}0449$. If we take the T-count after the first simplification then, we obtain $0{:}0737$ whereas the previous algorithm $0{:}1295$. We also compare the time

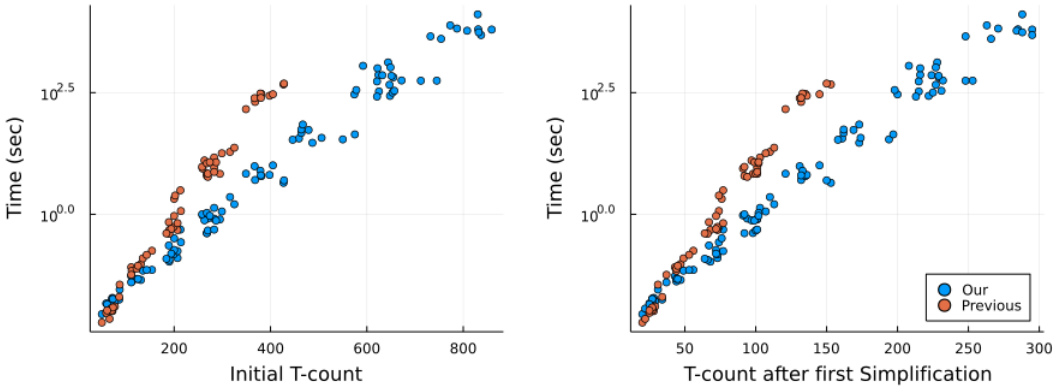


Figure 7.2: Strong simulation of IQP circuits

taken with the number of qubits by taking the mean of all the instances of every size considered.

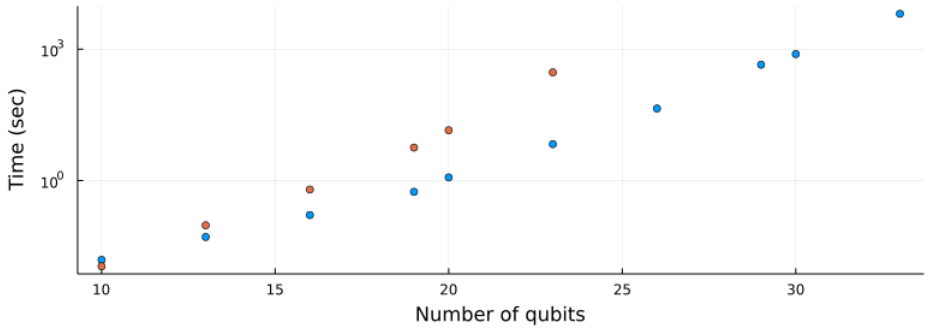


Figure 7.3: Time for strong simulations of IQP circuits compared to the number of qubits

In addition to the improved exponential growth, we obtained an improvement of multiple orders of magnitudes for bigger instances. We also mention that we estimate that our algorithm takes time that grows in $O(2^{0:825n})$ with the number of qubits which is much better than matrix-vector multiplication-based algorithms while also avoiding the consumption of a large amount of memory.

7.3.2 Random Clifford+T

For those circuits, small cats and stars were rarely present and almost no ZX simplification ever happened. Thus, the previous algorithm had to fall back to using magic state injection for most T gates. However, ours used a lot of star fusion which sparsified the ZX-diagrams. We have added dashed lines that grow exactly like $= 0:396$

as a baseline for comparison (the height of these lines was chosen arbitrary, only their rates of growth matter).

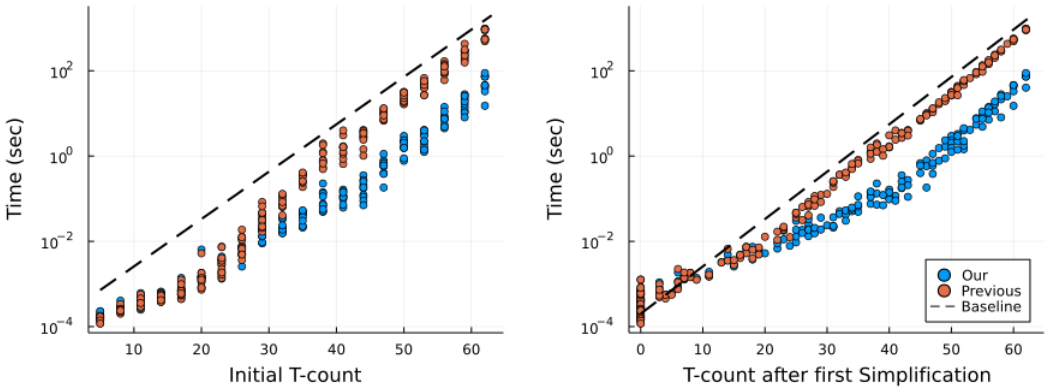


Figure 7.4: Strong simulation of random Clifford+T circuits

We see that the previous algorithm does scale similarly to the baseline while ours seems to do slightly better. We estimate that our algorithm has an 0:315. However, we note that the behaviour of our algorithm seems to not have fully settled and more data would be necessary to get a reliable measurement. Here again, we aggregate the data by taking the mean of circuits with the same depth.

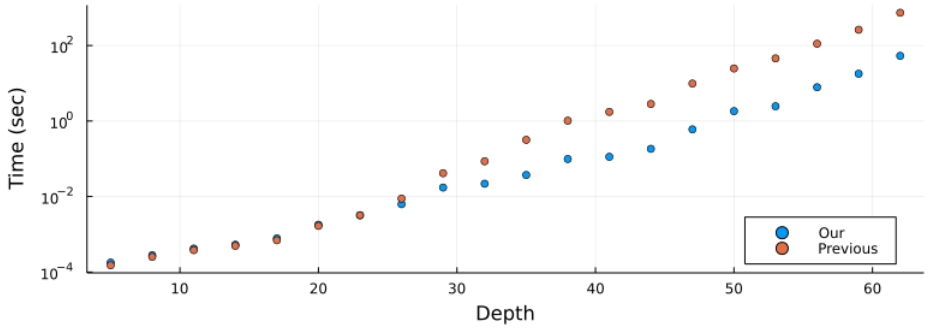


Figure 7.5: Time for strong simulations of Clifford+T circuits compared to the number of qubits

We observed significant improvement over the previous technique. Indeed we obtain up to a 13 fold improvement in our largest instance.

Chapter 8

Weak Simulation Techniques

Throughout the last chapters, we have discussed how to accelerate the strong simulation of a quantum circuit i.e. the calculation of the probability of an output state. However, there is another type of simulation that we have not discussed. Weak simulation is the task of producing a sample of the output distribution of a quantum circuit. Obviously, having an algorithm for weak simulation allows one to have an approximate algorithm for strong simulation. Indeed, it is enough to generate a large amount of sample to estimate the probability of a given output.

With a little more ingenuity, it is possible to use a strong simulation algorithm to do weak simulations. Although it may seem paradoxical to use an intuitively more difficult problem to solve a simpler one, weak simulation techniques based on strong simulation are among the fastest approaches known for this problem. One simple (and inefficient) way to do it is to compute the full output distribution by computing the probability of each possible output with a strong simulation. Since there are an exponential amount of possible outputs, this is completely impractical. There are two known ways to reduce weak simulation to strong simulation efficiently. One that has been used extensively in the past is the *qubit-per-qubit* method while the second which has been discovered only this year by [5] is the *gate-per-gate* method.

8.1 Qubit-per-Qubit

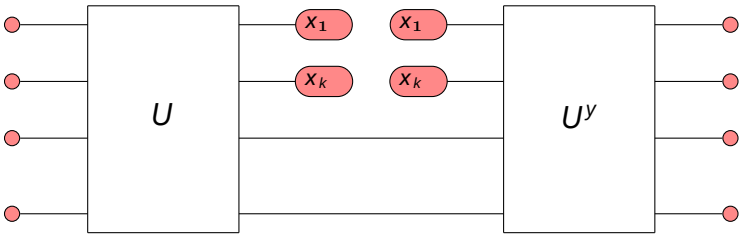
This technique is based on the chain rule from probability theory. Let $Q = Q_1, \ldots, Q_n$ be the result of each qubit of a measure of a circuit. Then the probability of obtaining an outcome $x = x_1, \ldots x_n$ is

$$\begin{aligned} P(Q = x) &= P(Q_1 = x_1 \wedge \ldots \wedge Q_n = x_n) \\ &= \prod_{k=1}^{n} P\left(Q_k = x_k \,\middle|\, \bigwedge_{j=1}^{k-1} Q_j = x_j\right) \end{aligned}$$

More than giving us another way to compute probabilities, this equation also gives us a way to sample a circuit. Indeed, if we were able to compute marginal probabilities on qubits then we could simply generate randomly the first qubit according to its distribution. We could then compute the marginal distribution on the second qubit, generate it and so on until we would have generated a complete sample. Doing so necessarily gives us a sample with the right distribution since it is exactly following the application of the chain rule. Thus, using strong simulation to compute marginals offers us a way to do weak simulations.

Fortunately, it is possible to do this operation. Indeed, we have that

$$\mathsf{P}\left(Q_1 ::: Q_k = x_1 ::: x_k\right) =$$



So we can compute the marginal distribution on Q_k by using the definition of conditional probability

$$\mathsf{P}\left(Q_k = x_k \,\middle|\, \bigwedge_{j=1}^{k\setminus 1} Q_j = x_j\right) = \frac{\mathsf{P}\left(Q_1 ::: Q_k = x_1 ::: x_k\right)}{\mathsf{P}\left(Q_1 ::: Q_{k\ 1} = x_1 ::: x_{k\ 1}\right)}$$

This technique is known as *CPM-construction* or simply *doubling* (for a proof that this technique works see [9]).

One of the main drawbacks is that this technique, as its name suggests, doubles the number of gates and specifically doubles the T-count of a diagram. This is bad news since in the worst case, our algorithm has an $\tilde{O}(2^{2\ t})$ time complexity, squaring the complexity of the strong simulation. This effect is mitigated as a lot of cancellation happens between U and U^y especially when a lot of wires are connecting them. For example, in an IQP circuit, it is possible to cancel all the gates on pairs of qubits that are both marginalised over.

8.2 Gate-per-Gate

In a recent paper [5], it has been shown that it is possible to avoid having to compute marginals (and therefore avoid doubling) to do weak simulations with a strong simulation algorithm. In this section, we will explain their method.

The idea is to create a sample of a subset of the circuit and make that sample evolves by adding gates one at a time until the whole circuit is considered. To see

how this works more concretely, we notice two facts. Firstly, it is trivial to sample an empty circuit. The sample $j0^n i$ is the only one possible. Secondly, if we add a gate to a circuit, it can only change the output distribution relative to the qubit that entered the new gate. More precisely, let U be a circuit, g a gate acting on the bounded set of qubits Q which we want to add to U making the new circuit $U^{\emptyset}$. Moreover let $Q_1;:::;Q_n$ be the (random) results of a measurement of $U j0^n i$ and $Q_1^{\emptyset};:::;Q_n^{\emptyset}$ be the result of a measurement of $U^{\emptyset}$, then

$$\mathsf{P}\left(\bigwedge_{i\in Q} Q_i = x_j\right) = \mathsf{P}\left(\bigwedge_{i\in Q} Q_i^{\emptyset} = x_j\right) \qquad \forall x$$

The reason for this is the non-signalling of quantum mechanics. If this was not the case, we could design an experiment, where we would take all of the qubits in Q far away from the rest of the qubits but send a signal instantaneously to someone in possession of the rest of the qubits by applying g or not. Since signalling is not permitted, this equation must hold. This equation implies that if we ignore the qubit in Q, then a sample from the output distribution of U is also a sample from the output distribution of $U^{\emptyset}$. Therefore, if we take a sample from the output of U, we only need to adjust the qubits in Q to obtain a sample from $U^{\emptyset}$. Since jQj is bounded (in most applications by two) we can use the brute force strategy to compute the probability distribution of the 2^{jQj} outputs where the qubits that are not in Q are the same as before. We can then sample from this distribution to create our sample from $U^{\emptyset}$. With this approach, we have to do up to $2^k m$ strong simulations where k is the size of the biggest gate in the circuit and m is the number of gates in the circuit. Thus, this whole procedure can be done in $\tilde{O}(2^k m 2^{-t})$. Since k can be assumed to be bounded by a small constant, we obtain $\tilde{O}(m 2^{-t})$. Furthermore, on every polynomial-size circuit, this reduces to $\tilde{O}(2^{-t})$ which is much better than the $\tilde{O}(2^{2^{-t}})$ from the qubit per qubit approach. However, the polynomial factors are much greater than before as we have to use a strong simulation routine multiple times for every gate.

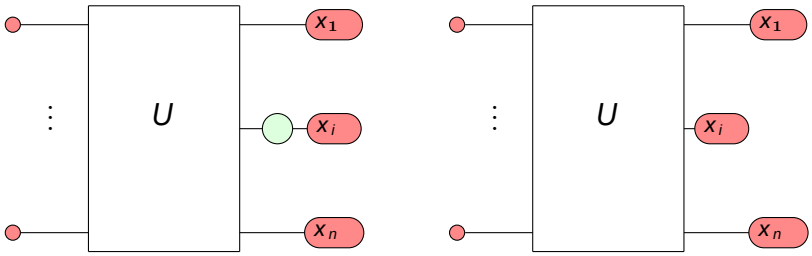
8.2.1 Speeding up the algorithm

Let us focus on the gate set Clifford+T as this is the set used in the circuits for our benchmarks. With this set, we have $k = 2$ that is achieved only when a CNOT gate is applied. However, the CNOT gate is deterministic on basis states which is a fact we can exploit to our advantage. This means that if we have a sample $q_1;:::q_n$ from a circuit U and we want to add a CNOT to the qubit i and j, we know that

$$\mathsf{P}\left(Q_1 ::: Q_k = q_1 ::: q_n\right) = \mathsf{P}\left(Q_1^{\emptyset} ::: Q_k^{\emptyset} = q_1 ::: q_i ::: q_j \quad q_i ::: q_n\right)$$

In other words, we could have sampled the circuit U' by taking a sample from U and classically applying a CNOT. Therefore, in this case, a strong simulation is not necessary. The above argument works whenever the gates we want to apply are deterministic on basis states. This implies that we also do not need to use strong simulation when the gate added is a NOT gate or even a Toffoli gate.

We can, in fact, even avoid having to use strong simulation whenever we want to add a T gate or more generally a Z-rotation. This is because adding a rotation gate doesn't change the output distribution. With ZX-calculus this can be seen as an application of the -copy rule.



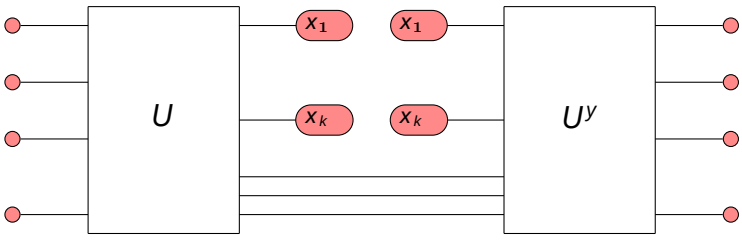
This means that the only time where strong simulation is necessary is when adding Hadamard gates which are single qubit gates.

We mention that this method scale especially well on IQP circuits where the number of Hadamard gates is $2n$ (in which the first n can trivially be simulated[1]).

8.3 Relation to graph cuts

We briefly mention how graph cuts can be useful for each of the weak simulation techniques presented.

Firstly, in the qubit-per-qubit method, we know that the cases with the least initial cancellation happen when we are computing marginal over the last qubits. This is because very few of the qubits are connected to the adjoint circuit removing a lot of the opportunity for cancellation.



At the same time, when only a few wires connect both circuits, there is necessarily a good vertex separator. It suffices to cut the remaining wire to separate the graph

[1]This is the case since, after the rst layer of Hadamard, we have a uniform output distribution.

into two equal parts. Thus, vertex cuts alleviate the worse case of the qubit per qubit simulation.

Secondly, for the gate-per-gate approach, a clever ordering of the gates can try to maintain the underlying ZX-diagram sparse for the longest time possible. For example, by taking gates affecting sets of wire far apart first, we can try to keep the graph weakly connected for the first few iterations of the algorithm which helps to find good cuts. However, if the whole circuit gives rise to a dense diagram, adding gates will eventually increase the connectivity of the diagram.

8.4 More Numerical Results

To assess the quality of both approaches, we tested both on several circuits from the same families as before. These simulations were run on the same dedicated computation server (24-core Intel Xeon E5-2667, 2.90GH).

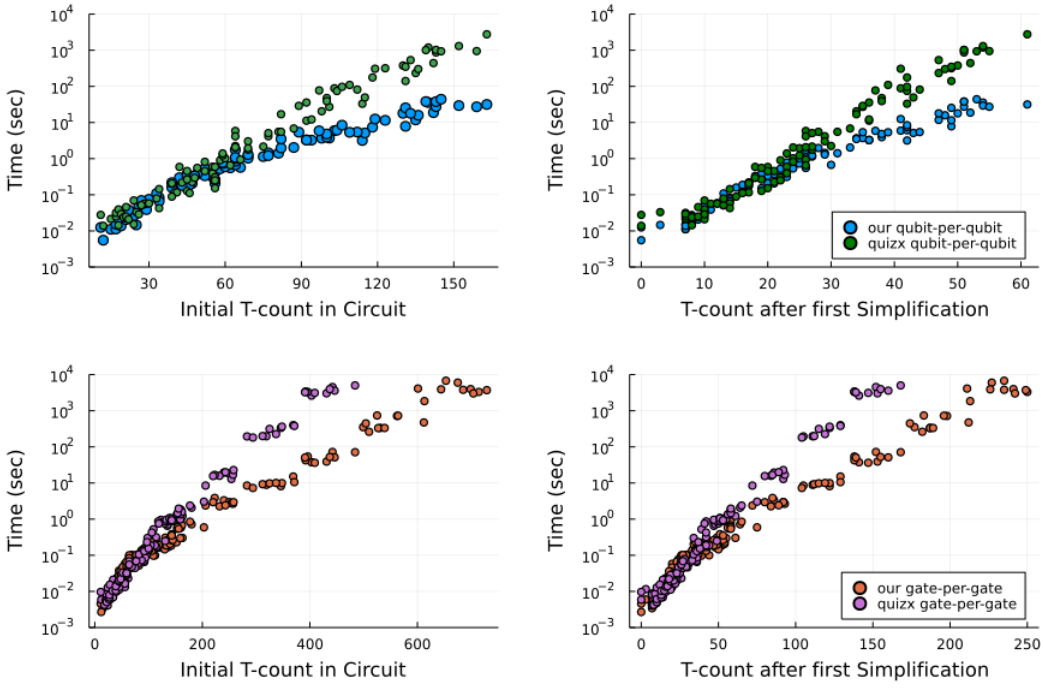


Figure 8.1: Weak simulation of random IQP

We see that gate-per-gate simulations are much more efficient than the qubit-per-qubit ones on IQP circuits. Indeed, we were able to sample circuits around four times as large within the time limit with this approach. This is not surprising as these circuits have very few Hadamard gates. In fact, both methods require $2n$ strong

simulations to obtain a sample. We can therefore see how not having to use doubling is beneficial in this type of circuit.

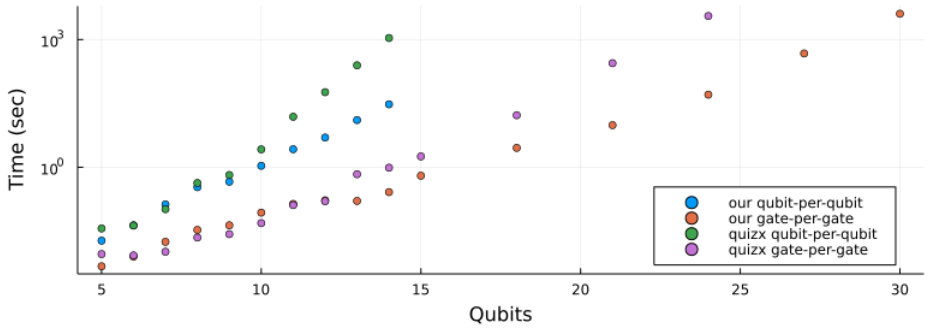


Figure 8.2: Weak simulation of random IQP compared to the number of qubits

In addition, by extrapolating the time taken by our gate-per-gate algorithm, we estimate that sampling 53 qubits circuits[2] should take around 50 to 60 years on our setup. Thus, we estimate that on large computer clusters, it would be possible to sample such a circuit in a few hours. Moreover, we estimate that, with such clusters, it would be possible to sample 65 qubits IQP circuits in less than 6 months. Sampling of random IQP circuits have previously been proposed as a task for quantum supremacy experiments [15, 7], but our results call into question the usefulness of IQP circuits in this context.

Similarly to IQP circuits, for random Clifford+T circuits, we see in figure8.4 that the gate-per-gate approach seems to be much more efficient as we managed to simulate circuits twice as big within the same time limit. This is interesting as those circuits contain more Hadamard gates which increase the polynomial factor in the time complexity. However, we believe that there are families where the polynomial factors will have a much greater impact as some families have a higher Hadamard gate density. Therefore, further testing would be required to say if the gate-per-gate approach is always more efficient for moderate-size circuits.

[2] The size of Google supremacy experiment.

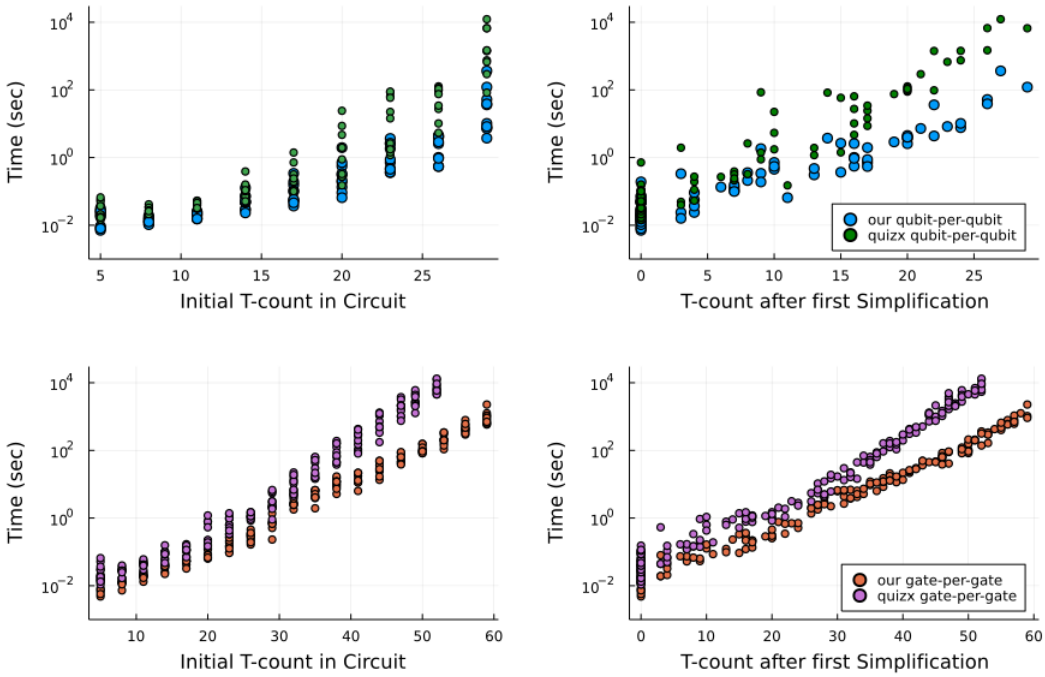


Figure 8.3: Weak simulation of random Clifford+T

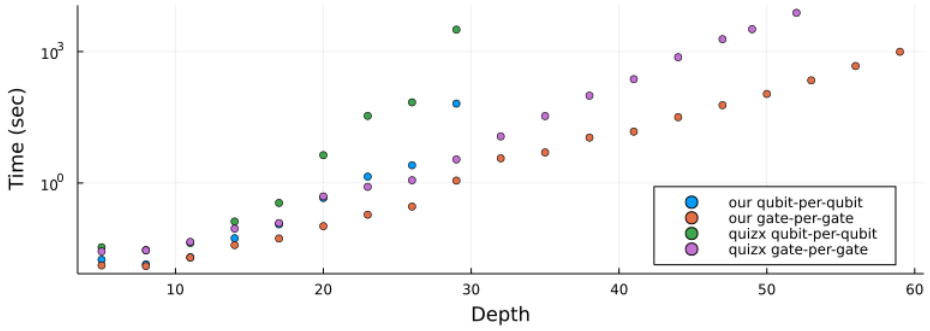


Figure 8.4: Weak simulation of random Clifford+T compared to the depth

Chapter 9

Conclusions

In this work, we have explored different approaches to improve strong quantum simulation techniques based on stabiliser decompositions. We have also explored two different methods for doing weak simulations. We have succeeded in significantly improving the current techniques to the point where we have questioned the usefulness of a class of quantum circuits for quantum supremacy experiments. This result adds to recent results from tensor contraction techniques indicating that the quantum supremacy milestone still requires progress on the experimental side to be reached. We also mention that the partial decomposition techniques developed in this work open up a new world of improvement opportunities. We expect that several better such decompositions exist which could eventually drastically speed up our algorithms. In addition, the subgraph complementation operation presented in section 3.3 has a lot of potential and requires further work.

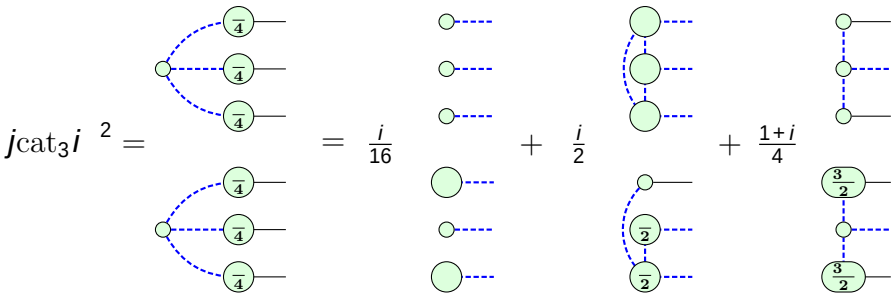
Appendix A

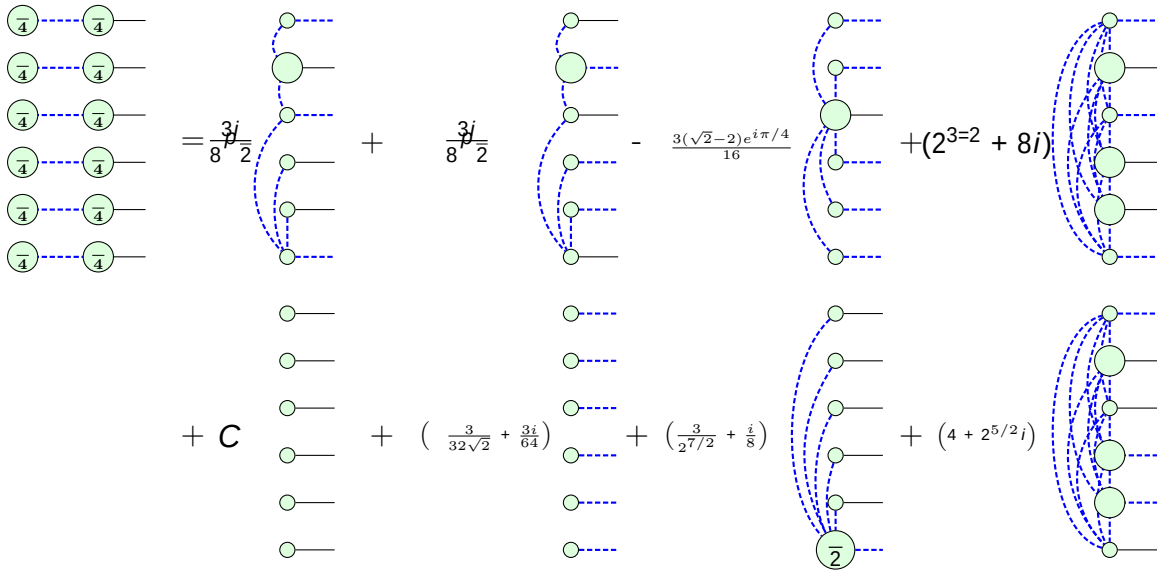
List of best decompositions

We here list the best-known decompositions for cat and star states up to size 40. An implementation of the algorithm used to find most of these decompositions is accessible at `https://github.com/Codsilla/ZX-stabiliser-simulator`.

A.1 Laakkonen's new decompositons

We here states two new decompositions for $|\text{cat}_3\rangle^{\otimes 2}$ and $|\text{star}_1\rangle^{\otimes 6}$ found numerically by Laakkonen [22].





Where C $0{:}017520 + 0{:}035826i$. The last two terms are fully connected graphs.

A.2 Single Copy of a Cat State

$\|\text{cat}_n\rangle$	Terms	$_n$	Method
1	1	0	Direct calculations
2	1	0	Direct calculations
3	2	0.3333	Direct calculations
4	2	0.2500	Direct calculations
5	3	0.3170	Direct calculations
6	3	0.2642	Direct calculations
7	6	0.3693	split into $\|\text{cat}_3\rangle$ and $\|\text{cat}_6\rangle$
8	6	0.3231	split into $\|\text{cat}_4\rangle$ and $\|\text{cat}_6\rangle$
9	9	0.3522	split into $\|\text{cat}_5\rangle$ and $\|\text{cat}_6\rangle$
10	9	0.3170	split into $\|\text{cat}_6\rangle$ and $\|\text{cat}_6\rangle$
11	18	0.3791	split into $\|\text{cat}_3\rangle$ and $\|\text{cat}_{10}\rangle$
12	18	0.3475	split into $\|\text{cat}_4\rangle$ and $\|\text{cat}_{10}\rangle$
13	27	0.3658	split into $\|\text{cat}_5\rangle$ and $\|\text{cat}_{10}\rangle$
14	27	0.3396	split into $\|\text{cat}_6\rangle$ and $\|\text{cat}_{10}\rangle$
15	54	0.3837	split into $\|\text{cat}_3\rangle$ and $\|\text{cat}_{14}\rangle$
16	54	0.3597	split into $\|\text{cat}_4\rangle$ and $\|\text{cat}_{14}\rangle$
17	81	0.3729	split into $\|\text{cat}_5\rangle$ and $\|\text{cat}_{14}\rangle$
18	81	0.3522	split into $\|\text{cat}_6\rangle$ and $\|\text{cat}_{14}\rangle$
19	162	0.3863	split into $\|\text{cat}_3\rangle$ and $\|\text{cat}_{18}\rangle$
20	162	0.3670	split into $\|\text{cat}_4\rangle$ and $\|\text{cat}_{18}\rangle$
21	243	0.3774	split into $\|\text{cat}_5\rangle$ and $\|\text{cat}_{18}\rangle$
22	243	0.3602	split into $\|\text{cat}_6\rangle$ and $\|\text{cat}_{18}\rangle$
23	486	0.3880	split into $\|\text{cat}_3\rangle$ and $\|\text{cat}_{22}\rangle$
24	486	0.3719	split into $\|\text{cat}_4\rangle$ and $\|\text{cat}_{22}\rangle$
25	729	0.3804	split into $\|\text{cat}_5\rangle$ and $\|\text{cat}_{22}\rangle$
26	729	0.3658	split into $\|\text{cat}_6\rangle$ and $\|\text{cat}_{22}\rangle$
27	1458	0.3893	split into $\|\text{cat}_3\rangle$ and $\|\text{cat}_{26}\rangle$
28	1458	0.3753	split into $\|\text{cat}_4\rangle$ and $\|\text{cat}_{26}\rangle$
29	2187	0.3826	split into $\|\text{cat}_5\rangle$ and $\|\text{cat}_{26}\rangle$
30	2187	0.3698	split into $\|\text{cat}_6\rangle$ and $\|\text{cat}_{26}\rangle$
31	4374	0.3902	split into $\|\text{cat}_3\rangle$ and $\|\text{cat}_{30}\rangle$
32	4374	0.3780	split into $\|\text{cat}_4\rangle$ and $\|\text{cat}_{30}\rangle$
33	6561	0.3842	split into $\|\text{cat}_5\rangle$and $\|\text{cat}_{30}\rangle$
34	6561	0.3729	split into $\|\text{cat}_6\rangle$ and $\|\text{cat}_{30}\rangle$
35	13122	0.3908	split into $\|\text{cat}_3\rangle$ and $\|\text{cat}_{34}\rangle$
36	13122	0.3800	split into $\|\text{cat}_4\rangle$ and $\|\text{cat}_{34}\rangle$
37	19683	0.3855	split into $\|\text{cat}_5\rangle$and $\|\text{cat}_{34}\rangle$
38	19683	0.3754	split into $\|\text{cat}_6\rangle$ and $\|\text{cat}_{34}\rangle$
39	39366	0.3914	split into $\|\text{cat}_3\rangle$ and $\|\text{cat}_{38}\rangle$
40	39366	0.3816	split into $\|\text{cat}_4\rangle$and $\|\text{cat}_{38}\rangle$

A.3 Single Copy of a Star State

We note that it would probably be possible to obtain better asymptotics for the decomposition with significantly bigger than 0.396 by using a combination of magic state injection of different size. We did not do it as only the decomposition with < 0.369 should ever be used.

$\|\text{star}_n\rangle$	Terms	$\_{n}$	Method
1	2	0.5	cut
2	2	0.3333	Catification
3	4	0.5	Catification into $\|\text{cat}_3\rangle$
4	4	0.4	Catification into $\|\text{cat}_4\rangle$
5	6	0.4308	Catification into $\|\text{cat}_5\rangle$
6	6	0.3693	Catification into $\|\text{cat}_6\rangle$
7	9	0.3962	magic state injection
8	12	0.3983	Catification into $\|\text{cat}_8\rangle$
9	18	0.417	Catification into $\|\text{cat}_9\rangle$
10	18	0.3791	Catification into $\|\text{cat}_{10}\rangle$
11	27	0.3962	magic state injection
12	36	0.3977	Catification into $\|\text{cat}_{12}\rangle$
13	54	0.4111	Catification into $\|\text{cat}_{13}\rangle$
14	54	0.3837	Catification into $\|\text{cat}_{14}\rangle$
15	81	0.3962	magic state injection
16	108	0.3973	Catification into $\|\text{cat}_{16}\rangle$
17	162	0.4078	Catification into $\|\text{cat}_{17}\rangle$
18	162	0.3863	Catification into $\|\text{cat}_{18}\rangle$
19	243	0.3962	magic state injection
20	324	0.3971	Catification into $\|\text{cat}_{20}\rangle$
21	486	0.4057	Catification into $\|\text{cat}_{21}\rangle$
22	486	0.388	Catification into $\|\text{cat}_{22}\rangle$
23	729	0.3962	magic state injection
24	972	0.397	Catification into $\|\text{cat}_{24}\rangle$
25	1458	0.4042	Catification into $\|\text{cat}_{25}\rangle$
26	1458	0.3893	Catification into $\|\text{cat}_{26}\rangle$
27	2187	0.3962	magic state injection
28	2916	0.3969	Catification into $\|\text{cat}_{28}\rangle$
29	4374	0.4032	Catification into $\|\text{cat}_{29}\rangle$
30	4374	0.3902	Catification into $\|\text{cat}_{30}\rangle$
31	6561	0.3962	magic state injection
32	8748	0.3968	Catification into $\|\text{cat}_{32}\rangle$
33	13122	0.4023	Catification into $\|\text{cat}_{33}\rangle$
34	13122	0.3908	Catification into $\|\text{cat}_{34}\rangle$
35	19683	0.3962	magic state injection
36	26244	0.3967	Catification into $\|\text{cat}_{36}\rangle$
37	39366	0.4017	Catification into $\|\text{cat}_{37}\rangle$
38	39366	0.3914	Catification into $\|\text{cat}_{38}\rangle$
39	59049	0.3962	magic state injection
40	78732	0.3967	Catification into $\|\text{cat}_{40}\rangle$

A.4 Multiple Copies of Cats

$\lvert\text{cat}_n\rangle$	Copies	Terms		Method
1	1	1	0.0000	Direct calculations
2	1	1	0.0000	Direct calculations
3	2	3	0.2642	Direct calculations
4	1	2	0.2500	Direct calculations
5	1	3	0.3170	Direct calculations
6	1	3	0.2642	Direct calculations
7	2	27	0.3396	split into $\lvert\text{cat}_3\rangle$ and $\lvert\text{cat}_6\rangle$
8	1	6	0.3231	split into $\lvert\text{cat}_4\rangle$ and $\lvert\text{cat}_6\rangle$
9	1	9	0.3522	split into $\lvert\text{cat}_5\rangle$ and $\lvert\text{cat}_6\rangle$
10	1	9	0.3170	split into $\lvert\text{cat}_6\rangle$ and $\lvert\text{cat}_6\rangle$
11	2	243	0.3602	split into $\lvert\text{cat}_3\rangle$ and $\lvert\text{cat}_{10}\rangle$
12	1	18	0.3475	split into $\lvert\text{cat}_4\rangle$ and $\lvert\text{cat}_{10}\rangle$
13	1	27	0.3658	split into $\lvert\text{cat}_5\rangle$ and $\lvert\text{cat}_{10}\rangle$
14	1	27	0.3396	split into $\lvert\text{cat}_6\rangle$ and $\lvert\text{cat}_{10}\rangle$
15	2	2187	0.3698	split into $\lvert\text{cat}_3\rangle$ and $\lvert\text{cat}_{14}\rangle$
16	1	54	0.3597	split into $\lvert\text{cat}_4\rangle$ and $\lvert\text{cat}_{14}\rangle$
17	1	81	0.3729	split into $\lvert\text{cat}_5\rangle$ and $\lvert\text{cat}_{14}\rangle$
18	1	81	0.3522	split into $\lvert\text{cat}_6\rangle$ and $\lvert\text{cat}_{14}\rangle$
19	2	19683	0.3754	split into $\lvert\text{cat}_3\rangle$ and $\lvert\text{cat}_{18}\rangle$
20	1	162	0.3670	split into $\lvert\text{cat}_4\rangle$ and $\lvert\text{cat}_{18}\rangle$
21	1	243	0.3774	split into $\lvert\text{cat}_5\rangle$ and $\lvert\text{cat}_{18}\rangle$
22	1	243	0.3602	split into $\lvert\text{cat}_6\rangle$ and $\lvert\text{cat}_{18}\rangle$
23	2	177147	0.3790	split into $\lvert\text{cat}_3\rangle$ and $\lvert\text{cat}_{22}\rangle$
24	1	486	0.3719	split into $\lvert\text{cat}_4\rangle$ and $\lvert\text{cat}_{22}\rangle$
25	1	729	0.3804	split into $\lvert\text{cat}_5\rangle$ and $\lvert\text{cat}_{22}\rangle$
26	1	729	0.3658	split into $\lvert\text{cat}_6\rangle$ and $\lvert\text{cat}_{22}\rangle$
27	2	1594323	0.3816	split into $\lvert\text{cat}_3\rangle$ and $\lvert\text{cat}_{26}\rangle$
28	1	1458	0.3753	split into $\lvert\text{cat}_4\rangle$ and $\lvert\text{cat}_{26}\rangle$
29	1	2187	0.3826	split into $\lvert\text{cat}_5\rangle$ and $\lvert\text{cat}_{26}\rangle$
30	1	2187	0.3698	split into $\lvert\text{cat}_6\rangle$ and $\lvert\text{cat}_{26}\rangle$
31	2	14348907	0.3835	split into $\lvert\text{cat}_3\rangle$ and $\lvert\text{cat}_{30}\rangle$
32	1	4374	0.3780	split into $\lvert\text{cat}_4\rangle$ and $\lvert\text{cat}_{30}\rangle$
33	1	6561	0.3842	split into $\lvert\text{cat}_5\rangle$ and $\lvert\text{cat}_{30}\rangle$
34	1	6561	0.3729	split into $\lvert\text{cat}_6\rangle$ and $\lvert\text{cat}_{30}\rangle$
35	2	129140163	0.3849	split into $\lvert\text{cat}_3\rangle$ and $\lvert\text{cat}_{34}\rangle$
36	1	13122	0.3800	split into $\lvert\text{cat}_4\rangle$ and $\lvert\text{cat}_{34}\rangle$
37	1	19683	0.3855	split into $\lvert\text{cat}_5\rangle$ and $\lvert\text{cat}_{34}\rangle$
38	1	19683	0.3754	split into $\lvert\text{cat}_6\rangle$ and $\lvert\text{cat}_{34}\rangle$
39	2	1162261467	0.3861	split into $\lvert\text{cat}_3\rangle$ and $\lvert\text{cat}_{38}\rangle$
40	1	39366	0.3816	split into $\lvert\text{cat}_4\rangle$ and $\lvert\text{cat}_{38}\rangle$

A.5 Multiple Copies of Stars

$\jmath\text{star}_n\imath$	Copies	Terms		Method
1	6	8	0:2500	Direct-calculation
2	1	2	0:3333	Catification
3	4	57	0:3648	Star fusion with two copies of $\jmath\text{star}_3\imath$
4	2	14	0:3857	Star fusion with two copies of $\jmath\text{star}_4\imath$
5	6	7290	0:3564	efficient split into $\jmath\text{star}_1\imath$ and $\jmath\text{cat}_6\imath$
6	1	6	0:3693	Catification into $\jmath\text{cat}_6\imath$
7	4	4382	0:3780	Star fusion with two copies of $\jmath\text{star}_7\imath$
8	2	129	0:3897	Star fusion with two copies of $\jmath\text{star}_8\imath$
9	6	5314410	0:3724	efficient split into $\jmath\text{star}_1\imath$ and $\jmath\text{cat}_{10}\imath$
10	1	18	0:3791	Catification into $\jmath\text{cat}_{10}\imath$
11	4	347320	0:3835	Star fusion with two copies of $\jmath\text{star}_{11}\imath$
12	2	1160	0:3915	Star fusion with two copies of $\jmath\text{star}_{12}\imath$
13	6	3874204890	0:3792	efficient split into $\jmath\text{star}_1\imath$ and $\jmath\text{cat}_{14}\imath$
14	1	54	0:3837	Catification into $\jmath\text{cat}_{14}\imath$
15	4	27810049	0:3864	Star fusion with two copies of $\jmath\text{star}_{15}\imath$
16	2	10422	0:3926	Star fusion with two copies of $\jmath\text{star}_{16}\imath$
17	6	2824295364810	0:3830	efficient split into $\jmath\text{star}_1\imath$ and $\jmath\text{cat}_{18}\imath$
18	1	162	0:3863	Catification into $\jmath\text{cat}_{18}\imath$
19	4	2236501135	0:3882	Star fusion with two copies of $\jmath\text{star}_{19}\imath$
20	2	93677	0:3932	Star fusion with two copies of $\jmath\text{star}_{20}\imath$
21	6	2058911320946490	0:3854	efficient split into $\jmath\text{star}_1\imath$ and $\jmath\text{cat}_{22}\imath$
22	1	486	0:3880	Catification into $\jmath\text{cat}_{22}\imath$
23	4	180272119162	0:3895	Star fusion with two copies of $\jmath\text{star}_{23}\imath$
24	2	842358	0:3937	Star fusion with two copies of $\jmath\text{star}_{24}\imath$
25	6	1500946352969991210	0:3871	efficient split into $\jmath\text{star}_1\imath$ and $\jmath\text{cat}_{26}\imath$
26	1	1458:0	0:3893	Catification into $\jmath\text{cat}_{26}\imath$
27	4	14550292599195	0:3904	Star fusion with two copies of $\jmath\text{star}_{27}\imath$
28	2	7576349	0:3940	Star fusion with two copies of $\jmath\text{star}_{28}\imath$
29	6	$1{:}094e+21$	0:3883	efficient split into $\jmath\text{star}_1\imath$ and $\jmath\text{cat}_{30}\imath$
30	1	4374	0:3902	Catification into $\jmath\text{cat}_{30}\imath$
31	4	1175404270123638	0:3911	Star fusion with two copies of $\jmath\text{star}_{31}\imath$
32	2	68153528	0:3943	Star fusion with two copies of $\jmath\text{star}_{32}\imath$
33	6	$7{:}977e+23$	0:3892	efficient split into $\jmath\text{star}_1\imath$ and $\jmath\text{cat}_{34}\imath$
34	1	13122	0:3908	Catification into $\jmath\text{cat}_{34}\imath$
35	4	95006833833682880	0:3917	Star fusion with two copies of $\jmath\text{star}_{35}\imath$
36	2	613142596	0:3945	Star fusion with two copies of $\jmath\text{star}_{36}\imath$
37	6	$5{:}815e+26$	0:3900	efficient split into $\jmath\text{star}_1\imath$ and $\jmath\text{cat}_{38}\imath$
38	1	39366	0:3914	Catification into $\jmath\text{cat}_{38}\imath$
39	4	7682471058401394418	0:3921	Star fusion with two copies of $\jmath\text{star}_{39}\imath$
40	2	5516538960	0:3946	Star fusion with two copies of $\jmath\text{star}_{40}\imath$

Bibliography

[1] Most balanced minimum cuts. *Discrete Applied Mathematics*, 158(4):261–276, February 2010. Publisher: North-Holland.

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

[3] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. Simulation of quantum circuits by low-rank stabilizer decompositions. *Quantum*, 3:181, September 2019. arXiv:1808.00128 [quant-ph].

[4] Sergey Bravyi and David Gosset. Improved Classical Simulation of Quantum Circuits Dominated by Clifford Gates. *Physical Review Letters*, 116(25):250501, June 2016. Publisher: American Physical Society.

[5] Sergey Bravyi, David Gosset, and Yinchen Liu. How to simulate quantum measurement without computing marginals. *Physical Review Letters*, 128(22):220503, June 2022. arXiv:2112.08499 [quant-ph].

[6] Michael J. Bremner, Ashley Montanaro, and Dan J. Shepherd. Average-Case Complexity Versus Approximate Simulation of Commuting Quantum Computations. *Physical Review Letters*, 117(8):080501, August 2016. Publisher: American Physical Society.

[7] Michael J. Bremner, Ashley Montanaro, and Dan J. Shepherd. Achieving quantum supremacy with sparse and noisy commuting quantum computations. *Quantum*, 1:8, April 2017. arXiv:1610.01808 [quant-ph].

[8] Bob Coecke and Ross Duncan. A graphical calculus for quantum observables. page 4.

[9] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, Cambridge, 2017.

[10] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum*, 4:279, June 2020. arXiv:1902.03178 [quant-ph].

[11] David Eppstein. Quasiconvex Analysis of Backtracking Algorithms. *arXiv e-prints*, page cs/0304018, April 2003. ADS Bibcode: 2003cs.....4018E.

[12] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved Approximation Algorithms for Minimum Weight Vertex Separators. *SIAM Journal on Computing*, 38(2):629–657, January 2008. Publisher: Society for Industrial and Applied Mathematics.

[13] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, June 1982.

[14] Daniel Gottesman. The Heisenberg Representation of Quantum Computers, July 1998. arXiv:quant-ph/9807006.

[15] Aram W. Harrow and Ashley Montanaro. Quantum Computational Supremacy. *Nature*, 549(7671):203–209, September 2017. arXiv:1809.07442 [quant-ph].

[16] Cupjin Huang, Fang Zhang, Michael Newman, Junjie Cai, Xun Gao, Zhengxiong Tian, Junyin Wu, Haihong Xu, Huanjun Yu, Bo Yuan, Mario Szegedy, Yaoyun Shi, and Jianxin Chen. Classical Simulation of Quantum Supremacy Circuits, May 2020. arXiv:2005.06787 [quant-ph].

[17] Aleks Kissinger and John van de Wetering. PyZX: Large Scale Automated Diagrammatic Reasoning. *Electronic Proceedings in Theoretical Computer Science*, 318:229–241, May 2020. arXiv:1904.04735 [quant-ph].

[18] Aleks Kissinger and John van de Wetering. Reducing the number of non-Clifford gates in quantum circuits. *Physical Review A*, 102(2):022406, August 2020. Publisher: American Physical Society.

[19] Aleks Kissinger and John van de Wetering. Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions, September 2021. arXiv:2109.01076 [quant-ph].

[20] Aleks Kissinger, John van de Wetering, and Renaud Vilmart. Classical simulation of quantum circuits with partial and graphical stabiliser decompositions, February 2022. arXiv:2202.09202 [quant-ph].

[21] Stefanos Kourtis, Claudio Chamon, Eduardo Mucciolo, and Andrei Ruckenstein. Fast counting with tensor networks. *SciPost Physics*, 7(5):060, November 2019.

[22] Tuomas Laakkonen. New stabiliser decompositions of cats states. Private Communication, 2022.

[23] Richard J. Lipton and Robert Endre Tarjan. A Separator Theorem for Planar Graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, April 1979. Publisher: Society for Industrial and Applied Mathematics.

[24] Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, February 2006.

[25] Ramis Movassagh. Quantum supremacy and random circuits, November 2020. arXiv:1909.06210 [cond-mat, physics:hep-th, physics:math-ph, physics:quant-ph].

[26] Michael A. Nielsen and Isaac L. Chuang. Quantum Computation and Quantum Information: 10th Anniversary Edition, December 2010. ISBN: 9780511976667 Publisher: Cambridge University Press.

[27] Feng Pan, Keyang Chen, and Pan Zhang. Solving the Sampling Problem of the Sycamore Quantum Circuits. *Physical Review Letters*, 129(9):090502, August 2022. Publisher: American Physical Society.

[28] Paul Raymond-Robichaud. A local-realistic model for quantum theory. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2250):20200897, June 2021. Publisher: Royal Society.

[29] Fanz Rendl and Renata Sotirov. The min-cut and vertex separator problem. *Computational Optimization and Applications*, 69(1):159–187, January 2018.

[30] Peter Sanders and Christian Schulz. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Experimental Algorithms*, Lecture Notes in Computer Science, pages 164–175, Berlin, Heidelberg, 2013. Springer.

[31] Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. k-way Hypergraph Partitioning via n-Level Recursive Bisection. In *2016 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, Proceedings, pages 53–67. Society for Industrial and Applied Mathematics, December 2015.

[32] David B. Shmoys. Cut Problems And Their Application To Divide-And-Conquer, 1996.

[33] John van de Wetering. ZX-calculus for the working quantum computer scientist, December 2020. arXiv:2012.13966 [quant-ph].

[34] Thorsten B. Wahl and Sergii Strelchuk. Simulating quantum circuits using efficient tensor network contraction algorithms with subexponential upper bound, August 2022. arXiv:2208.01498 [cond-mat, physics:hep-th, physics:quant-ph].