University College London

Department of Physics & Astronomy

Exploring Quantum Computation Through the Lens of Classical Simulation

Author:
Padraic Calpin

Supervisor:
Prof. Dan Browne

Submitted in partial fulfilment for the degree of **Doctor of Philosophy**

September 4, 2019

I, Padraic Calpin, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Signature

September 4, 2019

Date

Abstract

It is widely believed that quantum computation has the potential to offer an exponential speedup over classical devices. However, there is currently no definitive proof of this separation in computational power.

This separation would in turn imply that quantum circuits cannot be efficiently simulated classically. However, it is well known that certain classes of quantum computations nonetheless admit an efficient classical description. Recent work has also argued that accurate classical simulation of quantum circuits would imply the collapse of the Polynomial Hierarchy, something which is commonly invoked in classical complexity theory as a no-go theorem. This suggests a route for studying this 'quantum advantage' through classical simulations.

This project looks at the problem of classically simulating quantum circuits through decompositions into stabilizer circuits. These are a restricted class of quantum computation which can be efficiently simulated classically. In this picture, the rank of the decomposition determines the temporal and spatial complexity of simulating the computation.

We approach the problem by considering classical simulations of stabilizer circuits, introducing two new representations with novel features compared to previous methods. We then examine techniques for building these so-called 'stabilizer rank' decompositions, both exact and approximate. Finally, we combine these two ingredients to introduce an improved method for classically simulating broad classes of circuits using the stabilizer rank method.

Impact Statement

This thesis is focused on classical simulation of quantum computing, an important tool both for understanding the theoretical separation between quantum and classical computations, and for developing quantum software in an era where access to actual hardware is still significantly limited.

The impact of this thesis is to significantly extend the notion of stabilizer rank, a classical description of quantum systems that has received considerable interest in the community. As part of this, we introduce a novel method for simulating quantum circuits. This method enables simulations of near-term quantum computations on much larger system sizes than previous publicly available tools. For example, we demonstrate simulations of the Quantum Approximate Optimization Algorithm on 50 qubits, using a personal computer.

We have developed open-source software implementations of our work, enabling researchers in academia and industry to apply our methods to simulate quantum computations. We have also integrated the work with IBM's Qiskit framework [1], making the benefits of our method immediately available to the large international community of quantum software developers already experimenting with the platform, including enthusiasts and educators.

Our work has been widely disseminated through the arXiv, has been cited 16 times, and was the third highest-rated paper for 2018 on Sci-Rate, an open-access community for discussion and recommendation of preprints[1]. It has since been published in Quantum, a high-quality open access journal run by members of the quantum information community, and subsequent papers have been published building on our results [2, 3].

As well as simulations of universal quantum circuits, we developed two novel methods for simulating a common class of quantum circuit called stabilizer circuits. Our software implementations show generally better performance than current popular

[1] `https://scirate.com/?date=2019-01-01&range=365`

methods, and are available to the community through open-source.[2]

Our results on the stabilizer rank also impact attempts in the community to understand non-stabilizer states as a 'resource' for quantum advantage over classical computations. The exact origins of quantum speedup are still not entirely understood, and our results on both exact and approximate stabilizer rank decompositions have consequences for the design of quantum algorithms and quantum computing architectures. For example, the quantity of stabilizer extent introduced in Chapter 3 has since been applied to the problem of synthesising quantum gates from a limited gate-set [4].

[2] `https://github.com/padraic-padraic/StabilizerSim`

Acknowledgements

To do...

Contents

List of Symbols

The following is a list of some common symbols and notations used in this thesis.

We also make us of common abbreviations for computational complexity classes, which we denote in bold-faced type e.g. **P**

n	Number of qubits in a given quantum circuit or computation.
ω	The 8th root of unity, $\omega = \frac{1+\mathrm{i}}{\sqrt{2}}$.
$\mathbb{Z}_n$	The space of all positive integers modulo n.
$\vec{\mathbf{x}}$	A vector quantity, typically a binary vector.
$\vec{\mathbf{e}}_i$	A binary vector that is all-zero except for entry x_i.
$\|\vec{\mathbf{x}}\rangle$	A computational basis state defined by the binary vector $\vec{\mathbf{x}}$.
$\|\psi\rangle$	An arbitrary quantum state.
$\|\phi\rangle, \|\varphi\rangle$	A stabilizer state.
U	An arbitrary unitary operator.
V	An arbitrary operator belonging to the Clifford group.
$\mathcal{P}_n$	The n-qubit Pauli group
$\mathcal{C}_j$	The set of unitaries belonging to level j of the Clifford hierarchy. Also a group for $j \leq 3$.
$p_U(\vec{\mathbf{x}})$	The probability associated with the basis state $\vec{\mathbf{x}}$ in the state $U\left\|\vec{\mathbf{0}}\right\rangle$.
χ	The stabilizer rank, the rank of a stabilizer state decomposition. Typically, this refers to an exact decomposition.
χ_ϵ	The rank of an aproxiamate stabilizer state decomposition with maximum imprecision ϵ.

$\xi(\psi)$ The stabilizer extent of a state $|\psi\rangle$.

$O(g(n))$ An asymptotic scaling bounded above by $kg(n)$ for some constant $k > 0$

$\Theta(g(n))$ An asymptotic scaling bounded above by $k_1 g(n)$ and below by $k_2 g(n)$ for some constants $k_{1,2} > 0$

$\Omega(g(n))$ An asymptotic scaling that grows larger than $kg(n)$ for some constant $k > 0$ and some value of n.

Chapter 1

Introduction

Over the past 10 years, quantum computation has rapidly transitioned from a field of research into a burgeoning industry, drawing attention from national governments [5, 6], and private enterprise [7, 8, 9] alike. This intense interest is driven by the expectation that quantum computers can perform exponentially faster than current, classical devices on certain computational tasks.

The kinds of problems that are expected to have such a 'quantum advantage' are not only limited to simulating quantum mechanical systems [10], with applications to chemistry and materials science [11], but also include a tasks as diverse as optimization [12], search [13], prime factorization [14], random walk algorithms [15], and linear algebra [16], and machine learning [17].

A natural consequence of this exponential separation in computational power is that any classical simulation of a quantum computer should in general require exponentially more time to complete. This might seem to preclude classical simulations as a means of studying the development of quantum computing.

In practice, however, classical simulations have proven to be a valuable tool in aiding the development of quantum computing. Firstly, as a practical tool; despite their expense, classical simulations of small systems can be used to study the performance of possible quantum hardware [18]. They also provide theorists and developers with explicit ways to verify quantum algorithms on limited numbers of qubits, and have the potential to support the newly emerging group of quantum software developers [1, 19, 20].

But classical simulations can also play an interesting role in the foundations of quantum information science. For example, by studying classical descriptions of quantum computing states and operations, we can identify cases with efficient classical rep-

resentations. As an immediate consequence, these models of quantum computing cannot outperform a classical device, and thus probing the boundaries of these 'restricted' models gives us a way to study the transition between classical and quantum computing.

This thesis presents a powerful method for classical simulations of quantum circuits, based on decompositions into circuits and states belonging to an efficiently simulable restricted model of quantum computation. The remaining chapters will outline the core components of this method. Here, we discuss the principles of quantum computing and some of the current understanding of the separation between quantum and classical paradigms.

1.1 Foundations of Quantum Computing

Quantum computing is has its origins in discussions on the connection between computation and physics. In particular, how can a computation be understood in terms of a physical system? Formally, we are interested in constructing a representation relation, such that inputs to the computational task can be encoded as physical parameters, and that a measurable property of the system can be decoded as a computational output [21]. An interesting example of this kind of physical computation is the MONIAC, a macroeconomic simulator realised using fluid dynamics [22].

Given this kind of correspondence, we can then ask what insights physics can give into computation, and vice-versa. For example, it can be shown that analogue computation has the potential to efficiently solve **NP**-hard problems [23], for which it is widely believed no efficient discrete algorithm exists ($\mathbf{P} \neq \mathbf{NP}$, c.f. Section 1.1.1). The caveat, however, is that such a device would require arbitrarily high precision, and subsequently according to the Berkenstein bound would need infinite energy to operate [24].

Alternatively, we can consider spin-glasses, a class of many-body Hamiltonian. It can be shown that computing the ground-state of a spin glass is **NP**-hard [25], and it is possible to construct spin-glass Hamiltonians such that their ground-state corresponds to the solution to optimization problems [26, 27]. However, spin-glasses are also metastable, with the property that they 'freeze' into higher energy config-

urations, which means it is difficult to bring the system to its ground state [28]. Adiabatic methods, where the spin-glass Hamiltonian is slowly turned on such that the system should stay in its ground-state configuration, can also be shown to require exponentially long anneal times. From these two examples, there thus seems to be an equivalence between the difficulty of the computational task, and the physical realisation.

The first proposals for a quantum computer are widely attributed to the first conference on Physics of Computation, held at MIT in 1982. At that conference, Paul Benioff presented a method for constructing a Turing machine using a quantum mechanical system of spins [29].

A Turing machine is an abstract model of computation, introduced to aid the study for classical computation [30]. In this model, the computer is made up of a 'tape', discrete cells capable of storing a single value each, and a 'head', capable of reading or writing a value to the tape, or moving left and right. At each time-step, the head performs either a movement, read, or write operation, or else terminates the computation, according to its internal programme. Importantly, the Church-Turing thesis states that *any* computable function can be computed in finite time by a Turing machine. Equivalently stated, a Turing machine is a 'universal' model of computation, and any system capable of realising a Turing machine is said to be 'Turing Complete' [30].

In Benioff's model, each spin encodes a single cell of the tape, and the programme would be realised by spin-spin interactions. However, while Benioff's proposal used quantum dynamics, and he argued that its direct realization on a physical system should make it highly efficient [29], the computational model was strictly classical.

In his keynote remarks at the same conference, Richard Feynman discussed the possibility of simulating quantum mechanical systems with classical computers. He argued that any classical simulation of a quantum mechanical system should require resources that scale exponentially in the size of the system, and also discussed a variant of the sign problem, arguing that this made classical simulation intractable [31]. Instead, he considered the possibility of building computers distinct from the Turing

machine, out of purely quantum mechanical pieces, such that they could be used to realise Hamiltonians homomporphic to the process we want to simulate. These quantum computers, he proposed, could also be built out of lattices of spin-$\frac{1}{2}$ systems, and as quantum mechanical objects they should in some sense be able to efficiently encode quantum physical systems [31].

The idea of a quantum computation device built using two-level quantum systems was formalised by Deutsch, who introduced a more general model of quantum computing called a Quantum Turing Machine (QTM) [32]. In a QTM, the cells of the tape and the processor head are now built up of two-level quantum mechanical systems, which were later dubbed qubits in subsequent work by Schumacher on quantum information theory [33]. At each time-step, the head and tape interact under one of a finite set of unitary operations, which are capable of generating a group dense in the group of all unitary operations on the Hilbert-space of 2^n dimensional quantum systems [32].

Deutsch also proves several powerful features of the QTM model. Firstly, noting the correspondence between reversible classical dynamics, reversible computation [34], and unitary evolution, he points out that the QTM is itself Turing complete, and thus universal for classical computing [32]. He also proves that the QTM is capable of simulating any finite dimensional quantum system, and thus incorporates the notion of a quantum simulator discussed by Feynman [32].

Having defined this notion of a universal quantum computer, Deutsch also introduces an extension of the Church-Turing thesis that relates computation more closely to physical systems. Today known as the Church-Turing-Deutsch thesis, it states

> Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means.

The QTM, as defined, satisfies this thesis, as it is capable of simulating finite quantum mechanical systems. However, the Turing machine fails this criteria for both classical and quantum physics, as continuously valued systems cannot be efficiently encoded in binary arithmetic [32].

1.1.1 Complexity Theory and Quantum Advantage

The Church-Turing-Deutsch thesis, inspired by Feynman's intuition, thus gives the first example of a gap in the computational power of quantum and classical devices. In this section, we will discuss this potential separation using computational complexity theory, and define more precisely the notion of a task being computationally 'hard'.

Computational complexity theory is the study of computation by quantifying how the required number of operations, called 'temporal complexity', and the amount of memory, called 'spatial complexity', for computing a given task scales asymptotically as a function of the input parameters, typically the 'size' of the problem input [30]. As an abstract model of computing, we can relate the temporal and spatial complexity to the Turing machine; in this case, the number of steps taken by the head, and the number of cells on the tape.

Common Complexity Classes

Complexity theory is generally interested in grouping problems into 'classes', based on their asymptotic complexity, and studying the relationships between different classes. Provable membership of a class sets upper and lower-bounds on the performance of algorithms [30].

Typically, in complexity theory, a task is considered 'efficient' if its runtime is polynomial in the input size n. Given a deterministic algorithm to solve a problem efficiently, that problem belongs to the corresponding complexity class **P**. Otherwise, algorithms with super-polynomial scaling are considered 'inefficient'. While this separation seems coarse, as an algorithm with scaling $O\left(a^{0.1n}\right)$ would scale more efficiently than a method that scales as $O(n^{1000})$, in practice problems in **P** perform better than those with exponential scaling [30].

An additional important concept in complexity theory are 'efficient' mappings, also called reductions in some contexts. A mapping if efficient if it can be computed in at most a polynomial amount of time This concept allows us to perform computing without needing to use Turing machines. For example, a programming language can be shown to be Turing complete by using it to simulate a Turing machine. We can

also allow for algorithms that employ probabilistic strategies. In the Turing machine model, this corresponds to allowing the program to pick a move at random based on some probability distribution. **PP**, for 'Probabilistic Polynomial', is the class of problems for which a probabilistic algorithm exists that fails with probability $p_f < \frac{1}{2}$. By running this algorithm repeatedly, this failure probability p^m becomes arbitrarily small. However, this can in practice require a large number of repetitions if for example $p = \frac{1}{2} - \frac{1}{2^n}$ [35]. Thus, the related class **BPP**, Bounded Probabilistic Polynomial, is defined as problems where the failure probability $p < \frac{1}{3}$, such that the failure probability is exponentially decreasing in the number of repetitions [30]. It is immediately apparent that $\mathbf{P} \subseteq \mathbf{BPP}$, by setting $p_f = 0$, and that $\mathbf{BPP} \subseteq \mathbf{PP}$.

Another common class considered is **NP**, that class of problems for which a candidate solution can be checked in polynomial time, using a piece of additional information called a 'witness' or 'proof'. A common example is finding the factors of a number, which can be checked by multiplying the factors together. Importantly, however, there is not necessarily an efficient polynomial algorithm to find the solutions.

The class takes its name from the set of problems that can be efficiently solved by a 'non-deterministic' Turing machine. These can be conceptualised as similar to a probabilistic Turing machine, but where the machine chooses the 'best' path at each branching point such that it always arrives at a solution.

It follows that $\mathbf{P} \subseteq \mathbf{NP}$, as any solution for a **P** problem can be efficiently checked by running the algorithm and comparing the solutions. Interestingly, if we allow for post-selection in a probabilistic computation, then the resulting complexity class $\mathbf{PostBPP} \supseteq \mathbf{NP}$.This can be see from our heuristic description of the nondeterministic Turing machine, were we post-select on our probabilistic machine making the 'best' choice. It can in fact also be shown that $\mathbf{NP} \subseteq \mathbf{PP}$ [36].

The class **NP** contains many 'interesting' problems for which an efficient classical algorithm is not known to exist [30]. In fact, many of these problems, including optimization tasks like the Traveling Salesman Problem, are called **NP**-complete. A problem P is said to be 'complete' for a given complexity class if it is a member

of that class, and if it satisfies an additional criterion called 'hardness'. P is said to be **C**-hard if there exists an efficient reduction from all problems in **C** to P. Given an efficient algorithm to solve P, this the hardness property implies we can use the algorithm as a subroutine to efficiently solve all problems in **C** [30].

We can also define spatial equivalents of complexity classes. **PSPACE** is the class of all problems that can be solved requiring at most a polynomial amount of memory. Intuitively, this can also be thought of as the class of all problems that can be efficiently specified. We can also define exponential versions of the temporal and spatial complexity classes, **EXP** and **EXPSPACE**, which leads to the following inclusion relation:

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP} \subseteq \mathbf{EXPSPACE}.$$

Finally, here we briefly introduce two open questions in computational complexity that are nonetheless widely assumed to be false [30].

$$\mathbf{P} \neq \mathbf{NP}. \tag{1.1}$$

$$\text{The Polynomial Hierarchy } \mathbf{PH} \text{ does not collapse.} \tag{1.2}$$

Both of these results are typically invoked in 'no-go' arguments when reasoning about computational complexity. $\mathbf{P} \neq \mathbf{NP}$ can also be stated as asserting that no polynomial time algorithm exists for **NP**-complete problems. Scott Aaronson has discussed a number of arguments, from the physical to the philosophical, as to why we expect this to be the case [24, 37]. Intuitively, he describes $\mathbf{P} = \mathbf{NP}$ as implying 'thereś no fundamental gap between finding a solution, and recognising a solution once it is found' [37].

The Polynomial hierarchy is a recursively defined, infinite hierarchy of complexity classes that generalise **P** and **NP**. A collapse of the Polynomial hierarchy would imply that the hierarchy is finite. This statement is closely related to class #**P**, which captures the complexity of problems where we are asked to count the number of valid solutions. Requiring the Polynomial Hierarchy to be infinite can be understood as expecting #**P** problems to still be hard even if we are able to approximately count

solutions [38].

Quantum Computational Complexity

From the existence of the QTM, we can also define analogues of these complexity classes for quantum computation. However in practice, quantum algorithms are instead defined as families of quantum circuits. We focus on the circuit model, which is more immediately applicable to quantum hardware, as it can be shown that any computation that can be solved efficiently by a QTM can also be solved by a quantum circuit with depth at most polynomial in the number of qubits [39].

The class considered 'efficient' for quantum algorithms is **BQP**, Bounded Quantum Polynomial; problems for which a polynomial time quantum algorithm exists with a bounded failure probability $p_f < \frac{1}{3}$ [30].

Following Deutsch's observation that a QTM is capable of reversible classical computation, it can be shown that $\mathbf{P} \subseteq \mathbf{BQP}$ [40]. Interestingly, in the same paper also proved that a quantum computation on T steps requires just $O(\log T)$ precision. Thus, if $\mathbf{P} \neq \mathbf{BQP}$, this super-classical advantage doesn't require arbitrary precision, as in the analogue computing case [40].

The quantum class considered analogous to **NP** is **QMA**, where there exists a polynomial time quantum algorithm that can verify a solution that is encoded as a quantum state, with failure probability $p_f < \frac{1}{3}$ [41]. Examples of problems complete for **QMA** include deciding if a Hamiltonian built of just local interactions has a spectral gap [42].

When compared to classical classes, it can be shown that $\mathbf{BQP} \subseteq \mathbf{QMA} \subseteq \mathbf{PP}$. However, it is believed that $\mathbf{QMA} \neq \mathbf{PP}$, as otherwise $\mathbf{PH} \subseteq \mathbf{PP}$ [43]. The addition of post-selection, as in the classical case, also significantly boosts the capabilities of a quantum computer, and it in fact can be shown that the corresponding class $\mathbf{PostBQP} = \mathbf{PP}$ [35].

QMA is also closely related to the classical class **MA**, a proabilistic version of **NP** where we instead have a classical algorithm that can verify a solution with high probability given a classical proof. We can also consider a 'semi-classical' class **QCMA**, where we allow for a quantum verifier with just classical inputs, and it can

be shown that $\mathbf{MA} \subseteq \mathbf{QCMA} \subseteq \mathbf{QMA}$ [44].

In 1994, Shor famously demonstrated an efficient quantum algorithm for prime factorization [14]. Prime factoring is demonstrably in **NP**, as we can efficiently check a solution by multiplying factors. Importantly, there is currently no known efficient classical method for prime factorization. However, it has also not been definitively proven that factorization $\notin \mathbf{P}$ [30]. Shor's algorithm thus strongly suggests that $\mathbf{P} \neq \mathbf{BQP}$, but this separation has not been definitively proven.

There is, however, strong evidence that $\mathbf{NP} \not\subseteq \mathbf{BQP}$. In particular, if we consider a quantum algorithm with access to an oracle for verifying **NP** problems, then it can be shown the runtime of must scale as $O\left(2^{\frac{n}{2}}\right)$. This means the quantum speedup is at most quadratic, compared to many classical methods for **NP** problems which scale as $O\left(2^n\right)$ [45]. Such an 'oracle' can be efficiently implemented using a quantum circuit, based on the corresponding reversible classical circuit for the verifier. It was shown by Grover that this bound on the complexity is tight, using a quantum algorithm now commonly referred to as Grover search [13].

Overall, these results give some insight into the boundaries of **BQP**, and the kinds of problems that quantum computers can solve efficiently. Some of these relationships are represented schematically in Figure 1.1.

1.2 Classical Simulations of Quantum Computation

The previous section discussed formal notions of quantum computation, and examined its relation to classical computation through the lens of complexity theory. In this section we will introduce classical simulations of quantum computation. We will begin by introducing more precise notions of classical simulation, before focusing on different classical descriptions of quantum systems. This is by no means an exhaustive survey, but introduces some common paradigms for classical simulation. Finally, we will discuss results on the hardness of classical simulation, and briefly introduce the notion of a 'quantum supremacy' test.

1.2.1 Definitions of Classical Simulations

The general structure of quantum computations involves preparing an initial quantum state $\left|\vec{\mathbf{0}}\right\rangle$, applying a unitary evolution U, and then applying a measurement

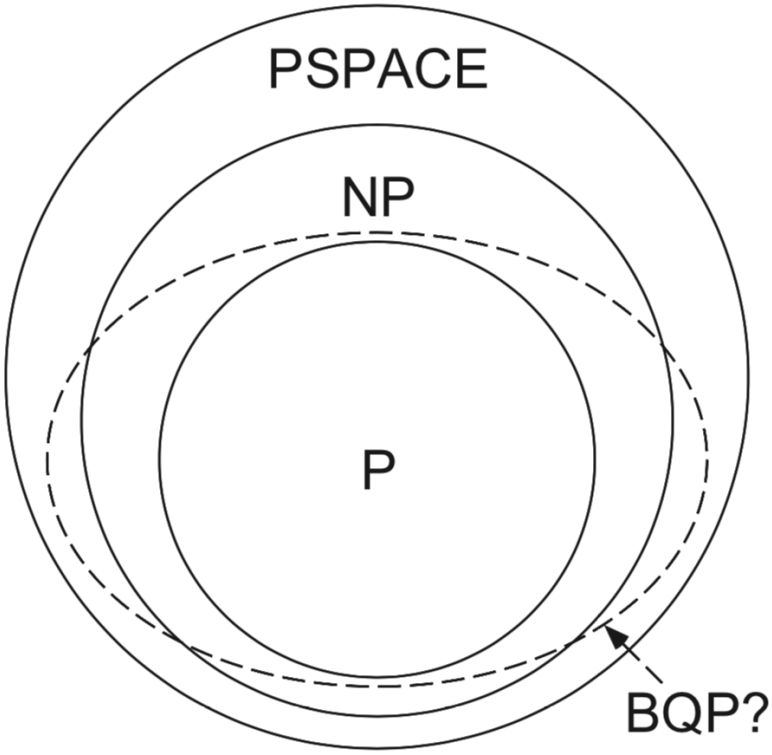


Figure 1.1: A Venn-diagram, illustrating the relationship of **BQP** to other classical complexity classes. Figure taken from [30].

or otherwise estimating an observable. Given this, an obvious definition of classical simulation is to compute the probabilities of different observables on the final state $|\psi\rangle = U\left|\vec{\mathbf{0}}\right\rangle$. This task is typically called 'strong' classical simulation [46].

Definition 1.1 (Strong Classical Simulation)**.** A strong classical simulator is any classical algorithm $\mathcal{A}$ that takes as input a description of a circuit U, and a description of the output observable s, and returns the probability of that output $p_U(s)$ [46].

Here, s could be the probability of some n-qubit computational state, or a marginal probability obtained by measuring some subset of the qubits. We denote as $\mathbb{S}$ the set of observables we are interested in simulating, and $\mathbb{P}_{\mathbb{S}}$ is the distribution of those events.

However, this task is distinct from what we are typically asking a quantum computer to do. When running a quantum algorithm, we are instead preparing the final state $|\psi\rangle$, and then sampling from its output distribution. These samples are then post-

processed to estimate other observables. Thus, we introduce the notion of a weak classical simulation.

Definition 1.2 (Weak Classical Simulation)**.** A weak classical simulator is a classical algorithm $\mathcal{A}$ capable of taking a classical description of a circuit U, and returning samples from its output distribution $\mathbb{P}_{\mathbb{S}}$ [46].

Access to a weak classical simulator should in effect be equivalent to having access to a quantum computer itself [47].

In practice quantum computations are not perfectly accurate, due to the influence of physical noise and control errors. We can correspondingly relax our definitions of classical simulation, to allow for a degree of approximation. An approximate strong simulation computes a given probability to within some specified precision ϵ, and an approximate weak simulation allows us to sample from a distribution $\hat{\mathbb{P}}_{\mathbb{S}}$ that approximates $\mathbb{P}_{\mathbb{S}}$.

There are several different definitions of precision used in approximate classical simulation. For strong simulation, there are three common definitions:

Definition 1.3 (Approximate Strong Simulation)**.** An approximate strong simulator computes estimates of output probabilities $\hat{p}(s)$ to within a given error ϵ, that is either

1. **Additive**: $|p(s)-\hat{p}(s)| \leq \epsilon \; \forall s \in \mathbb{S}$ [47]

2. **Multiplicative**: $\frac{1}{\epsilon}p(s) \leq \hat{p}(s) \leq \epsilon p(s) \; \forall s \in \mathbb{S}$ [48]

3. **Relative**: $(1-\epsilon)\,p(s) \leq \hat{p}(s) \leq (1+\epsilon)\,p(s) \; \forall s \in \mathbb{S}$ [49, 48].

Multiplicative and relative precision are slightly stronger requirements than additive error, as can be seen by considering the case where $p(s) \to 0$. In the literature, 'relative' simulation is also sometimes referred to as 'multiplicative' simulation, as the condition can be rewritten as [47]

$$|p(s)-\hat{p}(s)| \leq \epsilon p(s)\,.$$

For weak simulation, the commonly used notion of precision is variously referred to as ℓ_1-precision [50], additive precision [51] or simply ϵ-precision [47].

Definition 1.4 (Approximate Weak Simulation)**.** An approximate weak simulator samples from an output distribution $\hat{\mathbb{P}}$ such that

$$\left\|\mathbb{P}_{\mathbb{S}} - \hat{\mathbb{P}}_{\mathbb{S}}\right\|_1 = \sum_{s\in\mathbb{S}} |p(s) - \hat{p}(s)| \leq \epsilon.$$

The one-norm is used as it is directly proportional to the total variational distance between the two distributions.

Sometimes, a classical algorithm is capable of generating an approximate simulation $\hat{q}$ with some precision $f(\epsilon)$, with a non-zero probability of failure such that

$$\Pr\left[|q - \hat{q}| > f(\epsilon)\right] \leq \delta.$$

This is referred to as an (ϵ, δ)-precision approximation [47]. If this precision δ is bounded, then by m repeat rounds the probability of failure can be reduced to δ^m, as in the case for the complexity class **BPP**.

Given these notions then, we can define an efficient (ϵ, δ)-approximate classical simulation of an n qubit system as one with a complexity that scales as $\text{poly}\left(n, \epsilon^{-1}, \log\delta^{-1}\right)$ [47].

1.2.2 Classical Descriptions of Quantum Systems

What could be called the 'textbook' description of a quantum computation is the state-vector representation, a vector $|\psi\rangle \in \mathbb{C}^{2^n}$ that encodes the wave-function of the n-qubit system [30]. In this picture, the state is updated by applying $2^n \times 2^n$ unitary matrices. Thus, simulating a quantum computation in this picture might seem to require exponential spatial and temporal resources.

However, as quantum circuits are typically built out of local interactions, usually 1 and 2 qubit gates, it is possible build block decompositions of both the state-vector and unitary matrices. While a general state-vector requires 2^n complex numbers to completely specify, a tensor-product of smaller states requires just $O(n)$ [52].

A quantum computation can be described as p-blocked if at all time-points it can be decomposed into a tensor product of subsystems each acting on at most p qubits, where $1 \leq p \leq n$ [53]. It can be shown that the complexity of both strong and weak classical simulation on a p-blocked state-vector has a running time that scales as $O(2^{p+1}\text{poly}(m))$, where m is the number of blocks [53]. If p grows at most logarithmically in the system size, then this simulation is efficient. For a fixed p, then the runtime also scales efficiently as the system size grows. Otherwise, if p grows unboundedly in the duration of the computation, then simulation requires exponential resources. Interestingly, it can be shown that for example in Shor's algorithm, $p = n$ [52].

The Quantum Multiple Decision Diagram (QMDD) is an alternative encoding of the state-vector that similarly exploits redundancies arising from terms with equal amplitudes [54]. They combine this with a tree-based data-structure, where each leaf of the tree corresponds to a distinct amplitude, labelled by the outcomes to which it corresponds. In general, an arbitrary QMDD will have 2^n leaves, but the authors show that in practice QMDDs can be significantly smaller than this during common routines including the Quantum Fourier Transform. This can be exploited to develop a classical simulation method where updates scale as $O(n|v|)$, where $|v|$ is the number of leaves in the tree [55]. If this number grows at most polynomially in the system size, the simulation is efficient.

The cases where $|v|$ grows at most polynomially in the system size corresponds to the definiton of computationally tractable states, classes of states that admit efficient strong and weak simulation in the computational basis [56]. Certain sparse circuits acting on computationally tractable state in turn produce a computationally tractable output state, and thus can be efficiently simulated. This definition can also be expanded to approximate simulation, where we approximate its output distribution $\mathbb{P}$ to within additive error ϵ, with some other distribution $\hat{\mathbb{P}}$ that is computationally tractable [57].

An alternative representation of a quantum circuit, that is more distinct from the state-vector picture, is the use of tensor networks to simulate quantum circuits. These methods were first introduced in the context of simulating dynamics of many-

body quantum systems [58].

Tensor networks are undirected graphs, where tensors represent input and output states, and quantum gates, and edges represent qubit wires [59]. Tensors are combined or contracted by summing over shared indices, with a runtime that scales as the product of the dimensions of the indices to be contracted. For fixed input and output states x and y, contracting the entire network results in a rank-0 tensor or scalar value which corresponds to the amplitude $\langle \vec{\mathbf{x}}|U|\vec{\mathbf{y}}\rangle$ [59]. It can be shown that the complexity of simulating a tensor network scales exponentially with the 'tree-width', a property of the underlying graph of the network [59]. In general, tensor networks are known to be efficient when the entanglement of the system is bounded [58]. It can also be shown that for circuits built out of one and two-qubit gates, the tree-width scales linearly with both the number of qubits and the circuit depth [59].

Another alternative classical description of quantum circuits commonly discussed in the literature are quasiprobability or 'Wigner function' representations [60]. In this picture, the state is encoded in a quasiprobability distribution on a phase-space, where each point is described by a set of mutually anti-commuting operators [60]. The phase-space can be continuous, as is used in the study of quantum optics [30], or discrete, as is usually considered in the context of quantum computing [61].

It can be shown that, if the Wigner function representation is strictly positive, then the system is efficiently simulable classically [62, 63]. This is analogous to the same sign-problem discussed by Feynman in his 1982 address [31]. Positive Wigner function simulations use a random walk across the phase-space, with transition probabilities given by Wigner function expansions of each gate U. If the Wigner function is negative, then it is possible to define an alternative sampling strategy that can be used to simulate the computation, but at the expense of an increase in the number of samples, which scales exponentially 'negativity' of the system, the one-norm of its quasiprobability expansion [64]. The negativity is by definition 1 if the system is non-negative, and > 1 otherwise.

1.2.3 Efficiently Simulable Quantum Systems

In the previous section, we introduced several different representations of quantum computations used in classical simulations. In each case, there are also certain circumstances under which the simulation is efficient. Indeed, in their paper on simulation of p-blocked computations, it was noted by Jozsa & Linden that in general for any classical description $\mathcal{D}$, then there exists a corresponding property $\text{prop}(\mathcal{D})$ that is required for the simulation to be intractable classically [53]. For example, in both the p-blocked picture and the tensor network picture, the required feature $\text{prop}(\mathcal{D})$ is that the entanglement in the system grow unboundedly during the computation [53, 58].

Perhaps the most famous example of this requirement is related to the Gottesman-Knill theorem [65, 53].

> **Gottesman-Knill Theorem:** A quantum circuit built out of only
>
> - Preparation of states in the computational basis
> - Clifford and Pauli gates
> - Measurement in the computational basis
>
> can be efficiently simulated on a classical computer.

We will discuss the Gottesman-Knill theorem and how these circuits, typically called stabilizer circuits, can be simulated in Chapter 2. This theorem has gained particular attention in the field for several reasons. Firstly, it sets up a clear correspondence between classical simulability and universal quantum computing. The gate-set of stabilizer circuits is not universal for quantum computation. This means these circuits cannot be used to build up arbitrary unitary operations [30]. In fact, stabilizer circuits are also not universal for classical computation [66]. However, introducing a single gate outside of this set is sufficient to generate a group dense in the special unitary group, and thus 'promote' the gate-set to universality.

Stabilizer circuits are also closely related to the study of quantum error-correcting codes, where logical gates 'native' to he code are typically Pauli and Clifford gates

corresponding to stabilizer circuits [30]. Non-Clifford gates are then introduced through a protocol called state-injection, involving 'magic' states that cannot be generated by stabilizer circuits [67, 68].

Interestingly, it has also been shown that there is a correspondence between stabilizer circuits, and 'contextuality', a generalised notion of locality [69]. Contextuality was first discussed in the 1960s, where it was argued that non-contextuality is a unique feature of quantum systems [70, 71]. For example, in Spekkens's Toy Model, a classical model of quantum systems based on a restriction of available information, features like non-locality and entanglement can be efficiently described, but non-contextuality cannot [71]. Similarly, it was shown by Howard et al. that in odd-prime dimensional quantum systems, stabilizer circuits are entirely contextual, and the onset of non-contextuality is connected to the ability to distill high-fidelity magics states [69]. The case is slightly more complicated for quantum computing with qubits, as qubits show state-independent contextuality, but recent work has extended this correspondence between contextuality and qubit magic states [72].

Resources for Quantum Computation

The Gottesman-Knill theorem makes it clear that non-stabilizer resources are required for universal quantum computation, and as stated there is a correspondence between this required property and classical simulability. In fact, in general, the property $\text{prop}(\mathcal{D})$ can be interpreted as 'required' for quantum advantage, from the corresponding breakdown of efficient classical simulation. Jozsa & Linden argue that rather than any one 'true' $\text{prop}(\mathcal{D})$, it is likely that quantum computation requires all of these properties to be true.

For example, as discussed in the p-blocked case, this suggests that entanglement is required for quantum advantage. However, given that entangled stabilizer circuits exist, entanglement alone is clearly not sufficient. Similarly however, it is easy to construct non-stabilizer circuits that are either p-blocked, or sparse such that the output is computationally tractable.

The study of these potential resources, not only for computation but also for non-classical phenomena more generally, are called quantum resource theories. In general, a quantum resource theory is composed of a set of states that are considered

'free', a set of states that have an associated 'cost', and a set of allowed or 'free' operations [73]. We are then interested in asking what is the associated resource cost of certain protocols, and how does the resource behave under free operations?

Resource theories have been applied to a broad range of phenomena, including; non-stabilizer states [74]; magic states [75], more specifically; and asymmetry [76], which relates to both computational tractability [56] and to generalizations of Gottesman-Knill called normalizer circuits [77]. Of particular interest a resource theories where the set of free states is convex, as these also admit convex cost functions that can be more easily analysed [78]. All of the examples given in this paragraph fall into the category of convex resource theories [79].

There is also typically a correspondence between free states and operations in resource theories and their classical simulability. Indeed, many of the resource theories discussed above explicitly admit classical simulation through a discrete Wigner function representation. For example, it can be proven that any mixed state on n qubits with a positive Wigner function representation can be described as a convex mixture of stabilizer states [61]. In these pictures, the resource cost of a given state is directly related to the computational cost of classical simulation [75, 80, 81, 82].

As an example of the power of resource theories, we can consider the Robustness of Magic (RoM), introduced in [75]. This measure is equal to the negativity of a stabilizer state decomposition of a state. The RoM is largest for pure, non-stabilizer states, which lie outside the convex hull of the stabilizer states. However, as the state we are considering becomes more and more mixed, for example as a result of environment noise, it is pushed closer to the set of free states. Below some threshold, the state has RoM $\mathcal{R}(\rho) = 1$, and can be efficiently simulated. This corresponds neatly with the notion of magic state distillation, where we are trying to 'distill' pure magic states using Clifford circuits. We are consuming multiple copies, to produce a magic state with increased robustness, and which is subsequently harder to classically simulate. It is also congruous with the existence of a noise threshold below which we cannot distill [68]; below this threshold, the 'noisy' magic states are 'free' states, and thus we cannot increase their RoM using just free operations.

1.2.4 Simulation and Quantum Advantage

Given some of the results quoted in the previous sections, we might imagine that a sufficiently optimized classical simulation could exist to simulate quantum circuits. Here, we will review results from complexity theory which suggest that even an approximate efficient classical simulation of arbitrary quantum computation should not be possible.

Hardness of Strong Simulation

It can be shown that exact strong simulation would imply the existence of efficient classical algorithms to solve problems in $\#\mathbf{P}$, using a correspondence based on quantum circuits. From a famous computational result in complexity theory called Toda's theorem, it can be shown that the complexity class of problems solvable by a $\mathbf{P}$ algorithm with access to an oracle for $\#\mathbf{P}$, typically denoted $\mathbf{P}^{\#\mathbf{P}}$, in fact contains the entire $\mathbf{PH}$ [83]. Thus, the existence of an efficient classical algorithm for $\#\mathbf{P}$ problems would imply even strong consequences than $\mathbf{P} = \mathbf{NP}$.

The proof relies on the ability to construct quantum circuits C out of H and Toffoli gates [84], or alternatively out of H, Z, CZ and CCZ gates [85], such that computing the probability amplitude

$$\left\langle \vec{\mathbf{0}} \middle| C \middle| \vec{\mathbf{0}} \right\rangle \propto \mathrm{gap}\,(f)\,,$$

where $f : \{0,1\}^n \to \{0,1\}$ is an n-variable binary polynomial, and

$$\mathrm{gap}\,(f) = \#\vec{\mathbf{x}} : f\,(\vec{\mathbf{x}}) = 1 - \#\vec{\mathbf{x}} : f\,(\vec{\mathbf{x}}) = 0.$$

This problem is known to be $\#\mathbf{P}$ hard in general, if f has degree ≥ 3 [85].

Interestingly, this example also illustrates a case where studying quantum algorithms casts light on classical complexity theory. It was believed but not conclusively shown that $\mathrm{gap}\,(f)$ could be efficiently computed for polynomials with degree ≤ 2. A quantum circuit to compute the gap of such a polynomial could be built using just H, Z, and CZ gates, and is thus a stabilizer circuit. This means the amplitude $\left\langle \vec{\mathbf{0}} \middle| C \middle| \vec{\mathbf{0}} \right\rangle$ can in turn be computed efficiently, resolving the open problem [85]. This example also further illuminates the relationship between universal quantum ciruits and classical computation; circuits built using H, Z, CZ and CCZ gates are known

to be universal for quantum computing, and in turn are likely not to be strongly simulable.

There is in fact evidence that approximate strong simulation of quantum computations would also imply the collapse of the **PH**. For example, there exists a **BQP** algorithm for approximately evaluating the Jones polynomial [86], an important problem in the study of topological quantum field theories, to within additive error. This algorithm is based on repeatedly running a quantum circuit a polynomial number of times to obtain an estimate of a particular amplitude, and thus a strong classical simulation could be used to compute the Jones polynomial exactly. However, it can be shown that even approximating the Jones polynomial to within multiplicative error ϵ is a problem in #**P** [87], and thus no approximate strong classical simulation can exist unless the **PH** collapses. Interestingly, it is believed that approximately evaluating the Jones polynomial is a **BQP**-complete problem, and thus this result would suggest that **BQP** problems cannot be efficiently strongly simulated classically.

A core lemma in the proof of [87] is that given some family of quantum circuits C such that, with the addition of post-selection, the class of problems solvable by C circuits $\mathbf{postC} = \mathbf{PostBQP}$, then an efficient strong simulation of the output distribution of C would collapse the polynomial hierarchy. For the Jones polynomial, this follows as the problem is **BQP**-complete. Interestingly however, this result can also be used to imply that approximate strong simulation is hard even for certain types of quantum computation that are 'weaker' i.e. they are strictly not universal [88]. For example, Instantaneous Quantum Polynomial (IQP) circuits, circuits with depth $\mathrm{poly}(n)$, built out commuting gates like Z, CZ and CCZ, have the property that they are universal under post-selection [50]. Thus, strong simulation of IQP circuits implies the collapse of the **PH** [89].

An alternative strategy involves finding some correspondence between the quantum computation and another problem known to be #**P** hard. For example, the output distribution of IQP circuits can be shown to be equivalent to partition functions of Ising model Hamiltonians [90], and thus from a characterisation of these partition functions IQP circuits are #**P** hard to simulate in general [89]. This is also the

strategy employed by Aaronson & Arkhipov, when showing that a quantum optics task called Boson Sampling is equivalent to computing the permanent of a matrix to within additive error [88], a known #**P**-hard problem [91].

Extending Results to the Weak Simulation Case

While the results above apply to strong classical simulation, as discussed weak simulation can be considered as a more appropriate definition of classical simulation. It was also shown by Van den Nest that it is possible to construct quantum circuits whose strong simulation can be proven to be in #**P**, but which nonetheless admit an efficient weak simulation [46]. Thus, a bound on strong simulation is not sufficient to rule out classical simulation of quantum circuits.

Initial evidence for the hardness of weak sampling was given in [50], where the authors showed a weak simulation of IQP circuits with multiplicative error would imply the collapse of **PH**. Here, multiplicative error means that we have some approximate distribution $\hat{\mathbb{P}}$, such that every term is itself a multiplicative approximate of the true probability. This is strong requirement for approximate classical simulation.

Subsequent work has shown that in fact, it is possible to lift a complexity theoretic bound on strong simulation with multiplicative error, to obtain an equivalent bound on weak simulation with additive error, given a proof that the circuit families satisfies a pair of conjectures called 'anti-concentration' and 'average-case hardness' [48]. Recall that we are considering a family of quantum circuits C, which are hard to approximately strongly simulate, either as they are universal under postselection, or through correspondence to some other problem which is known to be #**P**-hard in the worst case.

The output distribution of these circuits is said to anticoncentrate if there exist two positive real numbers α and β, such that

$$\mathrm{Pr}_{U\sim\mathbb{P}_C}\left(\left|\left\langle\vec{\mathbf{x}}\middle|U\middle|\vec{\mathbf{0}}\right\rangle\right|^2 \geq \frac{\alpha}{N}\right) > \beta, \tag{1.3}$$

where N is the dimension of the system, typically 2^n for n qubits [48]. This inequality states that for some random circuit U drawn from the family C, the probability of an arbitrarily chosen entry being greater than uniform is greater than β. Intuitively,

anticoncentration requires that there is high probability the output distribution of U is reasonably close to uniform [92]. This ensures we are unlikely to find a circuit U with an exactly or approximately sparse output distribution, that would be subsequently amenable to classical simulation [46, 57].

Average-case hardness requires that, given a problem that is known to be #**P** hard to approximately compute in the worst-case, then it is also #**P** hard to simulate on some constant fraction $c > 0$ of the problem instance [93]. This takes us from a family of problems that can in principle be hard to simulate, to one where there are many instances known to be hard to simulate [92]. For example, some Ising Hamiltonians, and thus some instances of IQP circuits, are known to be classically simulable [90]. Nonetheless, IQP circuits can be shown to satisfy average-case hardness [93].

These two properties can be combined with a result from classical complexity theory called the Stockmeyer Counting Algorithm [94]. Taken together, average-case hardness and anticoncentration imply that a classical simulator capable of sampling from the output distribution with a worst-case additive error, is in fact capable of sampling with an average-case multiplicative error [93]. Subsequently, using the Stockmeyer algorithm, this weak simulator can then be used to obtain an estimate of an output probability with multiplicative error. This completes the reduction from weak simulation with additive error, to strong simulation with multiplicative error, and thus shows that weak simulation is also #**P**-hard [93].

Subsequent work has strengthened the case for the hardness of weak simulation. Consider having access to a classical algorithm capable of computing certain output probabilities with additive (ϵ, δ)-precision, efficiently — namely, with a runtime that scales as $\text{poly}\left(n, \epsilon^{-1}, \log \delta^{-1}\right)$. Such a device is called a poly-box [47]. It can be shown that even with such a capability, the output distribution cannot be efficiently weakly simulated with additive error as long as the output distribution $\mathbb{P}$ is not approximately sparse. Namely, there does not exist some distribution $\hat{\mathbb{P}}$ such that $\left\|\mathbb{P} - \hat{\mathbb{P}}\right\| \leq \epsilon$, and which has $t = \text{poly}\left(\epsilon^{-1}\right)$ non-zero amplitudes. Namely, even with such a strong classical simulation device, the distribution cannot be weakly simulated if it is sufficiently dense.

Quantum 'Supremacy'

While not necessarily practically applicable, these kind of random problems that are believed to be hard to simulate clasically under the assumption the **PH** does not collapse form the basis of an effort in the quantum computing community to demonstrate so-called 'quantum supremacy' — the ability for a quantum computer to successfully run an algorithm super-polynomially faster than a classical computer [95]. [1]

Random circuit problems are especially interesting compared to more direct tasks such as Shor's algorithm, because they require significantly fewer qubits to implement, and in some cases do not even require a universal architecture [85]. They are also more interesting than 'analogue' quantum simulators, quantum systems that realise model Hamiltonians, as we have much stronger guarantees on their computational hardness [85]. Nonetheless, there has been recent effort in the community towards constructing quantum simulators with provable hardness [48, 96, 97, 98].

A quantum supremacy experiment based on these kind of random sampling problems with be made up of two key parts: sampling, and verification. The first task, also referred to as 'Heavy Output Generation', is to generate a large number of samples from the output distribution of a given quantum circuit, drawn at random from a family of circuits [99]. This is done simultaneously using both a quantum computer, and an appropriate classical simulation method, presumably using High Performance Computing (HPC) resources.

The next step is to use an additional classical method to verify the output of both the quantum and classical samples. This step is in general even more computationally intensive than the sampling step, and is a significant caveat in current quantum supremacy proposals compared to running an **NP** problem such as factorization, which can be efficiently checked [92]. After verification, the runtime and resources required for both the quantum and classical methods are compared.

An important caveat in realising quantum supremacy experiments, however, is their

[1] I want to acknowledge here the very real concerns that have been raised about the use of the word supremacy, given the historical and contemporary political significance of the word. I use the term in this thesis as, despite much discussion, it has become the *de facto* name for this phenomenon.

susceptibility to noise. As previously discussed, sufficiently noisy quantum computations can in fact be efficiently simulated classically. It is thus important that quantum supremacy protocols are reasonably robust, as quantum error correction is out of reach for contemporary quantum computers [100].

This caveat also acts in tandem with continued development of classical simulation methods. For example, recent effort in simulation of noisy boson sampling problems has pushed the threshold for quantum supremacy to around $30-40$ photons [101], compared to current experimental records of about 6.

Because noisy quantum circuits can typically be more easily simulated, an alternative proposal for quantum supremacy experiments changes the order of the classical sampling and verification steps. The idea is to use a classical verifier capable of estimating noise in the system [102], such that this noise parameter can be used as input to the classical simulation to enable a fairer comparison.

Chapter 2

Methods for Simulating Stabilizer Circuits

2.1 Introduction

In Section 1.2.3, we briefly introduced the notion of stabilizer circuits as a class of efficiently simulable quantum computations. In this chapter, we revisit stabilizer circuits in detail, with a focus on different classical data structures for encoding stabilizer states and the corresponding algorithms for simulations.

Several informal definitions of stabilizer circuits have been used in the quantum computing literature [65, 66, 46, 81]. However, what each definition has in common is that they consider abelian subgroups $\mathcal{S} \subseteq \mathcal{P}_n$. These groups $\mathcal{S}$ are also called a stabilizer groups. Operations in the circuits have the property that they either leave these groups unchanged, or map them to new groups $\mathcal{S}' \subseteq \mathcal{P}_n$.

We focus exclusively on stabilizer circuits acting on pure states $|\phi\rangle$ called stabilizer states, which are entirely characterized by their associated stabilizer group as

$$s\,|\phi\rangle = |\phi\rangle \;\forall s \in \mathcal{S} \tag{2.1}$$

For an n-qubit state, the group $\mathcal{S}$ has 2^n elements [65]. As $\mathcal{S}$ is also abelian, this means it can itself be efficiently described by a set of n independent generators

$$\mathcal{S} = \langle g_1, g_2, \ldots, g_n \rangle \; : g_i \in \mathcal{S}, \tag{2.2}$$

which are commonly referred to as the 'stabilizers' of the state $|\phi\rangle$. We also note

that this definition allows us to write

$$|\phi\rangle\langle\phi| = \frac{1}{2^n}\sum_{s\in\mathcal{S}} s = \frac{1}{2^n}\prod_{i=1}^{n}(\mathbb{I}+g_i) \tag{2.3}$$

As stabilizer circuits map stabilizer states to other stabilizer states, this means they must be built up of unitary operations which map Pauli operators to other Pauli operators under conjugation. This set is commonly denoted as either $\mathcal{C}$ of $\mathcal{C}_2$, and alternatively referred to the 'second level of the Clifford hierarchy':

$$\mathcal{C}_2 \equiv \{V : VPV^\dagger \in \mathcal{P}_n \; \forall P \in \mathcal{P}_n\} \tag{2.4}$$

$$\mathcal{C}_j \equiv \{U : UPU^\dagger \in \mathcal{C}_{j-1} \; \forall P \in \mathcal{P}_n\} \tag{2.5}$$

where in Equation. 2.5 we have also introduced the (recursive) definition for level j of the Clifford hierarchy. For concreteness, we define level 1 of the hierarchy as $\mathcal{C}_1 = \mathcal{P}_n$.

From this definition, applying a Clifford unitary V updates the stabilizer group as

$$V\mathcal{S}V^\dagger = \langle V g_i V^\dagger\rangle = \langle g_i'\rangle = \mathcal{S}' \tag{2.6}$$

We also allow stabilizer circuits to contain non-unitary operations, in the form of measurements in the Pauli basis [65].

Simulating stabilizer circuits

From the above definitions, we can see that simulating a stabilizer circuit on n qubits corresponds to updating the n stabilizer generators for each unitary and measurement we apply. As the number of generators grows linearly in the number of qubits, if these updates can be computed in time $O(\text{poly}(n))$ then it follows the circuits can be efficiently simulated clasically.

The first proof of this was given by Gottesman in [65], by showing through examples that stabilizer updates can be quickly computed for the CNOT, H and S gates, and for single qubit Pauli measurements. The n qubit Clifford group can be entirely

generated from these gates,

$$\mathcal{C}_2 = \langle \mathrm{CNOT}_{i,j}, H_i, S_i : i,j \in \{1,2,\ldots,n\} \rangle, \tag{2.7}$$

and thus any Clifford operation can also be efficiently simulated. This result is typically referred to as the 'Gottesman-Knill' theorem.

A more formal proof follows from the work of Dehaene & de-Moor, who showed that the action of Clifford unitaries on Pauli operators corresponds to multiplication of $(2n+1)\times(2n+1)$ symplectic binary matrices with $(2n+1)$-bit binary vectors [103]. The dimension of these elements also grows just linearly in the number of qubits, and as matrix multiplication requires time $O(n^{2.37})$ it follows that we can update the stabilizers in $O(mn^{2.73})$ for m Clifford gates.

This work was extended by Aaronson & Gottesman, who introduced an efficient data structure for stabilizer groups, and algorithms for their updates under Clifford gates and Pauli measurement [66]. This method avoids the need for matrix multiplications, instead providing direct update rules allowing stabilizer circuits to be simulated in $O(n^2)$.

Since 2004, there have been several papers looking at different data structures and algorithms for simulating stabilizer circuits of the type we consider here. For example, a method based on encoding stabilizer states as graphs [104], refinements of the Aaronson & Gottesman encoding [105], and an encoding using affine spaces and phase polynomials [46, 49].

In the rest of this section, we will discuss different aspects of simulating stabilizer circuits, focusing on updating stabilizer states under gates and measurements, computing stabilizer inner products, and the connections between stabilizer circuits and states.

2.1.1 Tableau Encodings of Stabilizer States

The method in [66] is based on a classical data structure the authors call the 'stabilizer tableau', a collection of Pauli matrices that define the stabilizer group, encoded

using the binary symplectic representation of [103]

$$P = (-1)^{\epsilon}\,\mathrm{i}^{\delta}\bigotimes_{i=1}^{n} x_i z_i \tag{2.8}$$

where the Pauli matrix at qubit i is defined by two binary bits such that

$$x_i z_i = \begin{cases} I & x_i = z_i = 0 \\ X & x_i = 1, z_i = 0 \\ Z & x_i = 0, z_i = 1 \\ Y & x_i = z_i = 1 \end{cases} \tag{2.9}$$

Together with the δ and ϵ phases, a generic Pauli operator can be encoded in $2n+2$ bits; two bits to encode the phase, and two n-bit binary strings $\vec{\mathbf{x}}, \vec{\mathbf{z}} \in \mathbb{Z}_2^n$ to encode the Pauli acting on each qubit, commonly referred to as 'x-bits' and 'z-bits' respectively. In this picture, multiplication of Pauli operators corresponds to addition of $\vec{\mathbf{x}}$ and $\vec{\mathbf{z}}$ bits modulo 2, with some additional, efficiently computable function for correcting the phase [103].

$$PQ = i^{\delta_{pq}} - 1^{\epsilon_{pq}} \bigotimes_{i=1}^{n} x'_i z'_i \tag{2.10}$$

$$x'_i = x_{pi} \oplus x_{qi} \tag{2.11}$$

$$z'_i = z_{pi} \oplus x_{qi} \tag{2.12}$$

where $\delta_{pq} = \delta_p \oplus \delta_q$, $\epsilon_{qr} = f(\vec{\mathbf{x}}_p, \vec{\mathbf{z}}_p, \vec{\mathbf{x}}_q, \vec{\mathbf{z}}_q)$.

In stabilizer groups, we can restrict ourselves to considering Pauli operators with only real phase. This is because if $iP \in \mathcal{S}$, then $(iP)^2 = -I \in \mathcal{S}$. But, this implies that $-I\,|\phi\rangle = |\phi\rangle$, which can only be satisfied by the null vector.

While only n generators S_i are needed to characterize the stabilizer group $\mathcal{S}$, the tableau also includes an additional $2n$ operators called 'destabilizers' $D_i \in \mathcal{P}_n$. Together, these $2n$ operators generate all 4^n elements of $\mathcal{P}_n$. Including this additional information speeds up the task of simulating stabilizer circuits with this representation.

There are many possible choices of destabilizer, but the tableau chooses operators such that [66]

$$\begin{aligned}[D_i, D_j] &= 0 \,\forall i, j \in \{1, \ldots, n\} \\ [D_i, S_j] &= 0 \iff i \neq j \\ \{D_i, S_i\} &= 0\end{aligned}$$

Altogether, the full tableau has spatial complexity $4n^2 + 2n$. These are sometimes referred to as 'Aaronson-Gottesman 'tableaux or 'CHP' tableaux, after the software implementation by Aaronson [106].

$$\begin{array}{c|cccc|cccc|c} \mathcal{D}_1 & x_{1,1} & \cdots & x_{1,n} & & z_{1,n} & \cdots & z_{1,n} & & r_1 \\ \vdots & \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots & & \vdots \\ \mathcal{D}_n & x_{n,1} & \cdots & x_{n,n} & & z_{n,1} & \cdots & z_{n,n} & & r_n \\ \hline \mathcal{S}_1 & x_{n+1,n} & \cdots & x_{n+1,n} & & z_{n+1,1} & \cdots & z_{n+1,n} & & r_{n+1} \\ \vdots & \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots & & \vdots \\ \mathcal{S}_n & x_{2n,1} & \cdots & x_{2n,n} & & z_{2n,1} & \cdots & z_{2n,n} & & r_{2n} \end{array} \tag{2.13}$$

Figure 2.1: Example of a 'CHP' tableau, where the first n rows are the Destabilizers and the next n rows are the stabilizers. The $2n+1$th column gives that phase -1^{r_i} for each operator.

Simulating Gates

Gate updates for each individual operator in the tableau can be computed constant time. For example, the Hadamard transforms single qubit Pauli matrices under conjugation as

$$HPH^\dagger = \begin{cases} I & P = I \\ Z & P = X \\ X & P = Z \\ -Y & P = Y \end{cases} \tag{2.14}$$

In the symplectic form, we then have to update the ith Pauli operator as

$$x_i' z_i' = (x_i \oplus p)(z_i \oplus p) \;:\; p = x_i \oplus z_i \tag{2.15}$$

and the phase as

$$\delta' = \delta \oplus (x_i \wedge z_i) \tag{2.16}$$

Similar update rules exist for the CNOT and S gates, which together generate the n qubit Clifford group. As there are $O(n)$ operators in the tableau, and each update is constant time, gate updates overall take $O(2n)$ [66]. This is in contrast to the $O(n^{2.37})$ complexity of [103]

Simulating Measurements

The addition of the destabilizer information is used to speed up the simulation of Pauli measurements on Stabilizer states. Measuring some operator P on a stabilizer state will always produce either a deterministic outcome, or an equiprobable random outcome [65].

If the outcome is deterministic, then $\pm P$ is in the stabilizer group, and the outcome is $+1$ or -1 respectively. Using the stabilizer generators, this allows us to write

$$[P, S_i] = 0 \; \forall S_i \in \mathcal{S} \implies \prod_i c_i S_i = \pm P. \tag{2.17}$$

for binary coefficients c_i.

Checking if the outcome is deterministic takes $O(n^2)$ time in general, using the symplectic inner product to check the commutation relations [103]. However, checking which measurement outcome occurs involves computing the coefficients c_i. In the symplectic form, this can be rewritten as

$$A\vec{\mathbf{c}} = P$$

where $\vec{\mathbf{c}}$ is a binary vector, A is a matrix with each stabilizer as a column vector, P is the operator to measure, and we have dropped the phase. Solving this would require inverting the matrix A, and take time $O(n^3)$.

Aaronson & Gottesman show that for single qubit measurements, including destabilizer information instead allows us to compute the c_i and the resulting measurement outcome in $O(n^2)$. As this is a single qubit measurement, they also show that the commutivity relation requires checking only individual bits of the stabilizer vectors,

also reducing that step to $O(n)$ time.

For random measurements, from Equation. 2.17, $\exists S_i : \{S_i, P\} = 0$, and it suffices to replace this stabilizer with P, and update the other elements of the group as $S'_j = PS_j$ iff $\{S_j, P\} = 0$ [65, 66].

'Canonical' Tableaux

There are multiple possible choices of generators for each stabilizer group/state. For example, the stabilizer group for the Bell state $|\phi^+\rangle = \frac{1}{2}(|00\rangle + |11\rangle)$ can be written as

$$\mathcal{S} = \{II, XX, -YY, ZZ\} = \langle XX, -YY\rangle = \langle XX, ZZ\rangle = \langle -YY, ZZ\rangle. \tag{2.18}$$

In simulation, tableau are fixed by choice of a convention. For example, it is possible to arrive at a 'canonical' set of stabilizer generators using an algorithm which strongly resembles Gaussian elimination [105]. This method rearranges the stabilizer rows of the tableau by multiplying and swapping generators, such that the overall stabilizer group is left unchanged. Computing this canonical form requires time $O(n^3)$ [105].

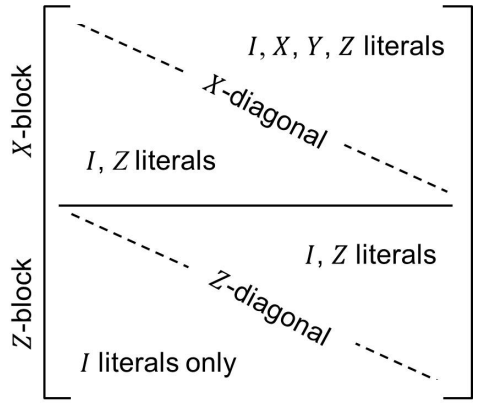


Figure 2.2: Representation of the canonical or 'row-reduced' set of stabilizer generators. Figure taken from [105].

These tableau can then be updated using the same methods as in [66], though this will in general not preserve the canonical form. Each Clifford gate will change one or two columns of the tableau, and thus an additional $O(n)$ row multiplications are required to restore it to canonical form, taking total time $O(n^2)$ per gate [107].

Importantly this canonical tableau can also be used to compute deterministic measurement outcomes in time $O(n)$, and so this method can simulate measurement outcomes more efficiently at the cost of more expensive gate updates [107].

In contrast, Aaronson & Gottesman fix the stabilizer tableau through an initial state, $\left|\vec{\mathbf{0}}\right\rangle$. The full tableau for this state looks like the identity matrix, with an additional zero-column for the phases. The tableau of a given state $|\phi\rangle$ is then built-up gate by gate using a stabilizer circuit $V : |\phi\rangle = V\left|\vec{\mathbf{0}}\right\rangle$.

2.1.2 Connecting Stabilizer States and Circuits

The convention for the 'CHP' stabilizer tableaux mentioned above, and the definition of stabilizer circuits given in Section 2.1, show that stabilizer states can also be defined by a stabilizer circuit and an initial state.

In [66], the authors derive examples of these 'canonical circuits', and show that its possible for any stabilizer state to be synthesised by a unique circuit acting on the $\left|\vec{\mathbf{0}}\right\rangle$ state

$$|\phi\rangle = V\left|\vec{\mathbf{0}}\right\rangle = H\,C\,S\,C\,S\,C\,H\,S\,C\,S\left|\vec{\mathbf{0}}\right\rangle \tag{2.19}$$

where each letter denotes a layer made up of only Hadamard (H), CNOT (C) or S gates. The proof is based on a sequence of operations reducing an arbitrary tableau to the identity matrix, each step of which corresponds to applying layers of a given Clifford gate [66]. As a corollary, the total number of gates in the canonical circuit for an n-qubit stabilizer state scales as $O(n\log(n))$ [66], based on previous work on synthesising CNOT circuits with the $O(n\log(n))$ gates [108], and that each H and P layer can act on at most n-qubits.

A simpler canonical form was derived in 2008, which allows a stabilizer circuit to be written as

$$|\phi\rangle = S\,CZ\,X\,C\,H\left|\vec{\mathbf{0}}\right\rangle \tag{2.20}$$

where the CZ and X layers are made up of Controlled-Z gates and Pauli X gates, respectively [46]. This circuit follows from the work of [103], who showed that any

stabilizer state can be written as

$$|\phi\rangle = \frac{1}{\sqrt{2^k}} \sum_{x \in \mathcal{K}} i^{f(\vec{\mathbf{x}})} |\vec{\mathbf{x}}\rangle . \tag{2.21}$$

In this equation, $\mathcal{K} \subseteq \mathbb{Z}_2^n$ is an affine subspace of dimension k, and $f(\vec{\mathbf{x}})$ is a binary function evaluated mod 4. Thus, a stabilizer state is always a uniform superposition of computational basis strings, with individual phases $\pm$i, ± 1. The affine space $\mathcal{K}$ has the form

$$\mathcal{K} = \{G\vec{\mathbf{u}} + \vec{\mathbf{h}}\}$$

for k-bit binary vectors u, an $n \times k$ binary matrix G, and an n-bit binary 'shift-vector' h.

Van den Nest notes that this representation can be directly translated into a stabilizer circuit; we begin by applying H to the first k qubits to initialize the state $\sum_{\vec{\mathbf{u}}} |\vec{\mathbf{u}}\rangle \otimes \left|0^{\otimes n-k}\right\rangle$. We then apply CNOTs to prepare $\sum_{\vec{\mathbf{u}}} |G\vec{\mathbf{u}}\rangle$, and finally Pauli Xs to prepare $\sum_{\vec{\mathbf{u}}} \left|G\vec{\mathbf{u}} \oplus \vec{\mathbf{h}}\right\rangle$ [46].

The phases can be further decomposed into two linear and quadratic binary functions $l, q : \mathbb{Z}_2^n \to \mathbb{Z}_2$, such that $i^{q(\vec{\mathbf{x}})} = i^{l(\vec{\mathbf{x}})}(-1)^{q(\vec{\mathbf{x}})}$. The linear terms correspond to single qubit phase gates, which can be generated by the S gate, and the quadratic terms to two-qubit phase gates, generated by the CZ [46]. Thus,

$$|\phi\rangle = \sum_{\vec{\mathbf{x}} \in \mathcal{K}} i^{l(\vec{\mathbf{x}})}(-1)^{q(\vec{\mathbf{x}})} |\vec{\mathbf{x}}\rangle = S\ CZ\ X\ C\ H\ |0\rangle \tag{2.22}$$

While [46] showed that these simpler canonical circuits exist, an algorithm to compute them first introduced in 2012 [105]. This method allowed such a circuit to be read off from the 'canonical' set of stabilizer generators introduced in Section 2.1.1.

2.1.3 Computing Inner Products

The final task we might consider in simulating stabilizer circuits is the problem of computing probability amplitudes $P(\vec{\mathbf{x}}) = |\langle \vec{\mathbf{x}} | \phi \rangle|^2$. As computational states are also stabilizer states, this corresponds more broadly to computing inner products between stabilizer states.

From the affine space form in Equation. 2.21, we can see that

$$\langle\varphi|\phi\rangle = \frac{1}{\sqrt{2^{k+k'}}} \sum_{\vec{\mathbf{x}}\in\mathcal{K}\cap\mathcal{K}'} i^{f(\vec{\mathbf{x}})-f'(\vec{\mathbf{x}})} \tag{2.23}$$

and the problem of computing the inner product corresponds to computing the magnitude of an 'exponential sum' of phase differences ($\pm$i, ±1) for each string $\vec{\mathbf{x}}$ in the intersection of the two affine spaces [49]. As each term in the sum has amplitude $\frac{1}{\sqrt{2}}$ and as terms $\pm\mathrm{i}, \pm1$ can cancel, we can see that

$$|\sum_{\vec{\mathbf{x}}} i^{f(\vec{\mathbf{x}})-f'(\vec{\mathbf{x}})}| = \begin{cases} 0 \\ 2^{s/2} : s \in \{0,1,\ldots,n\} \end{cases}$$

This sum can be solved in $O(n^3)$ time, using an algorithm developed by Sergey Bravyi [49, 109, 110]. An algorithm for computing this intersection was also described in [49], which we discuss further in Section 2.2.3.

Alternatively, the inner product can also be computed using the stabilizer generators directly. Consider two states $|\phi\rangle, |\varphi\rangle$ with respective generators G_i, H_i. If $\exists i,j : G_i = -H_j$, the states are orthogonal and the inner product is 0. Otherwise, the inner product is given by 2^{-s}, where $\vec{\mathbf{s}}$ the number of generators $G_i \notin \{H_i\}$.

While there are multiple choices of stabilizer generators, we note that inner products are invariant under unitary operations U as

$$\langle\varphi|\phi\rangle = \langle\varphi|U^\dagger U|\phi\rangle\,.$$

Thus, given the canonical circuit $V : |\varphi\rangle = V\left|\vec{\mathbf{0}}\right\rangle$

$$\langle\varphi|\phi\rangle = \langle\varphi|V^\dagger V|\phi\rangle = \left\langle\vec{\mathbf{0}}\right|V\left|\phi\right\rangle.$$

Each stabilizer G'_i of $\left|\vec{\mathbf{0}}\right\rangle$ has a single Pauli Z operator acting on qubit i. By

simplifying the stabilizer H'_i of $V|\phi\rangle$ using Gaussian elimination, then we have

$$\left|\left\langle\vec{\mathbf{0}}\middle|V\middle|\phi\right\rangle\right| = \begin{cases} 0 & \exists H'_i = \bigotimes_i Z_i \\ 2^{-s} & \exists H'_i : \{H'_i, G'_i\} = 0 \end{cases} \tag{2.24}$$

where $\vec{\mathbf{s}}$ is the number of stabilizers that anticommute with the corresponding stabilizer G'_i [66]. The second case arises as if $\{H'_i, G'_i\} = 0$, then H'_i acts as either Pauli X or Y on qubit i. Thus, the qubit is in state $|\pm 1\rangle$ or $|\pm \mathrm{i}\rangle$, and $\langle 0|\pm i, 1\rangle = \frac{1}{\sqrt{2}}$. Because this method involves computing the canonical circuit and then applying Gaussian elimination, it runs in time $O(n^3)$.

The first implementation of this algorithm was given in [105], where the authors first use their canonical form to construct a 'basis circuit' $B : |\varphi\rangle = B\left|\vec{\mathbf{b}}\right\rangle$ for some computational state $\left|\vec{\mathbf{b}}\right\rangle$, and then compute $\left\langle\vec{\mathbf{b}}\middle|B\middle|\phi\right\rangle$ using the same method outlined above [105].

2.2 Results

The main result of this chapter is to introduce two new classical representations of stabilizer states developed in collaboration with Sergey Bravyi [109]. We will discuss their algorithmic complexity, and implementation in software. We will also briefly discuss the implementation of a classical data-structure based on affine spaces, introduced in [49].

Finally, we present data evaluating the performance of all three methods. For the affine space representation, we benchmark against existing implementations in MATLAB [49]. For the two novel representations, we present data comparing their performance to two pieces of existing stabilizer circuit simulation software [66, 104].

2.2.1 Novel Representations of Stabilizer States

Existing classical simulators have two important limitations. One is that they focus only on implementations of single qubit Pauli measurements made in the Z basis. Multi-qubit measurements, or measurements in different bases, need to be built up in sequence, or involve applying additional basis changes gates like H and S, respectively.

These simulators also do not track global phase information. For the case of simulating individual stabilizer circuits, this is sufficient as global phase does not affect measurement outcomes. However, if we wish to extend our methods to simulating superpositions of stabilizer states, then phase differences between terms in the decomposition must also be recorded [107].

Here, we present two data structures, which we call the 'DCH' and 'CH' forms.

Definition 2.1. DCH Representation:
Any stabilizer state $|\phi\rangle$ can be written as

$$|\phi\rangle = \omega^e U_D U_{\text{CNOT}} U_H |\vec{\mathbf{s}}\rangle \tag{2.25}$$

where U_D is a diagonal Clifford unitary such that

$$U_D |\vec{\mathbf{x}}\rangle = i^{f(\vec{\mathbf{x}})} |\vec{\mathbf{x}}\rangle ,$$

U_{CNOT} is a layer of CNOT gates, U_H is a layer of Hadamard gates, acting on a computational state $|\vec{\mathbf{s}}\rangle$, and with a global phase factor w^e where $\omega = \sqrt{\mathrm{i}}$ and $e \in \mathbb{Z}_8$.

Any diagonal Clifford matrix of the form U_D is described by its 'weighted polynomial' $f(\vec{\mathbf{x}})$, evaluated mod 4, which can be expanded into linear and quadratic terms as [46, 111]

$$f(\vec{\mathbf{x}}) = \sum_i a_i x_i + 2 \sum_{c,t} x_j x_k \mod 4 = L(\vec{\mathbf{x}}) + 2Q(\vec{\mathbf{x}})$$

where the coefficients $a_i \in \mathbb{X}_4$. This was also the expansion used in the definition of the affine space representation in Equation. 2.22.

We observe that the linear terms can be entirely generated by the S, Z and $\mathrm{S}^\dagger$ gates acting on single qubits, and the quadratic terms by CZ gates acting on pairs of qubits [111]. Thus, any unitary U_D can be built up of these gates. As a corollary, we note that these 'DCH' circuits can be obtained from the 7-stage circuits given in Equation. 2.20, by commuting the X layer through to the beginning of the circuit and acting it on the $\left|\vec{\mathbf{0}}\right\rangle$ initial state. [46].

The computational string $\vec{\mathbf{s}}$ can be encoded as an n-bit binary row-vector. This is

also true of the Hadamard layer, which can be expanded in terms of a binary vector $\vec{\mathbf{v}}$ as

$$U_H = \bigotimes_{i=1}^{n} H^{v_i}. \tag{2.26}$$

A CNOT gate controlled on qubit c and targeting qubit t transforms the computational basis states as

$$\mathrm{CNOT}_{c,t} |\vec{\mathbf{x}}\rangle = \mathrm{CNOT}_{c,t} \bigotimes_{i=1}^{n} |x_i\rangle = \bigotimes_{i=1}^{n} |x_i \oplus \delta_{i,t} x_c\rangle$$

i.e. it adds the value of bit c to bit t, modulo 2. We can therefore encode the action of U_{CNOT} as an additional $n \times n$ binary matrix E which is equal to the identity matrix, with an additional one at $E_{c,t}$, such that

$$\mathrm{CNOT}_{c,t} |\vec{\mathbf{x}}\rangle = |xE\rangle \ : \ E_{i,j} = \begin{cases} 1 & i = j \\ 1 & i = c, j = t \\ 0 & \text{otherwise} \end{cases} \tag{2.27}$$

We can then build up U_{CNOT} from successive CNOT gates as

$$U_{\mathrm{CNOT}} |\vec{\mathbf{x}}\rangle = |xE_1E2E_3 \ldots E_m\rangle \equiv |xW\rangle \tag{2.28}$$

where $W = E_1 E_2 \cdots E_n$ is the matrix representing the full circuit, obtained by successive right multiplication of the matrices encoding a single CNOT.

Finally, we need to encode the action of U_D. The phase resulting from a single qubit diagonal Clifford is conditional on the qubits being in the $|1\rangle$ state. We write the linear part of the weighted polynomial as $L\vec{\mathbf{x}}^T$ for some row-vector L of integers mod 4, which we call the linear phase vector. Each value in L can be stored using just 2 bits.

Each gate $CZ_{i,j}$ between qubits i and j also contributes a factor of 2 to the overall phase, conditioned on the ith and jth qubits being in the $|1\rangle$ state. For a given computational string $\vec{\mathbf{x}}$, the overall phase from the CZ gates is thus $2\sum_{i,j:CZ_{i,j}} x_i x_j$.

We can encode the action of the CZ gates using an $n \times n$ symmetric binary matrix

Q where $Q_{i,j} = Q_{j,i} = 1$ if we apply $CZ_{i,j}$, and zero otherwise, which we call the quadratic phase matrix. We can then compute the phase from the CZ gates as

$$\begin{aligned}
\vec{\mathbf{x}} M \vec{\mathbf{x}}^t &= \sum_p x_p \left(Q x^T \right) \\
&= \sum_p x_p \left(\sum_q Q_{p,q} x_q \right) \\
&= \sum_{p,q} x_p x_q Q_{p,q} \\
&= 2 \sum_p \sum_{q>p} x_p x_q Q_{p,q} \\
&= 2 \sum_{i,j : CZ_{i,j} \in U_D} x_i x_j
\end{aligned}$$

where the last line follows from the definition of the matrix Q. Altogether, this allows us to write [49]

$$U_D \left| \vec{\mathbf{x}} \right\rangle = i^{f(\vec{\mathbf{x}})} \left| \vec{\mathbf{x}} \right\rangle = i^{L \vec{\mathbf{x}}^T + \vec{\mathbf{x}} Q \vec{\mathbf{x}}^T} \left| \vec{\mathbf{x}} \right\rangle = i^{\vec{\mathbf{x}} B \vec{\mathbf{x}}^T} \left| \vec{\mathbf{x}} \right\rangle \tag{2.29}$$

where B is a matrix such that $B_{ii} = L_i$, $B_{i,j} = Q_{i,j}$, as by definition Q has zero diagonal. We refer to B as simply the phase matrix, with diagonal elements stored mod 4 and off-diagonal elements stored mod 2.

Finally, we include the global phase factor, an integer modulo 8 and stored using just three bits. Overall the DCH representation is then specified by the tuple $(e, \vec{\mathbf{s}}, \vec{\mathbf{v}}, B, W)$. The spatial complexity is thus $O(n^2)$. In order to optimize certain subroutines, which we discuss later in this section, we also store a copy of W^{-1}, the inverse of the CNOT matrix, and W^T, the transpose of the CNOT matrix.

We further introduce two variables $p \in \{0, 1, \ldots, n\}$, $\epsilon = 0, 1$, which are used to ensure normalisation of the DCH state under certain operations. Together with the phase e, they define a coefficient we denote $c = 2^{-p/2} \epsilon \omega^e$. We store p as an unsigned integer, and ϵ as a single binary bit. We choose the 8th root of unity, w, for the phase as this ensures a correct global phase during Hadamard updates. Overall, then, the DCH form requires roughly $4n^2 + 4n + 36$ bits of memory.

Definition 2.2. CH Representation:
Any stabilizer state $|\phi\rangle$ can be written as

$$|\phi\rangle = \omega^e U_C U_H |\vec{\mathbf{s}}\rangle \tag{2.30}$$

where U_C is a Clifford operator such that

$$U_C \left|\vec{\mathbf{0}}\right\rangle = \left|\vec{\mathbf{0}}\right\rangle, \tag{2.31}$$

U_H is a layer of H gates, $|\vec{\mathbf{s}}\rangle$ is a computational basis state, and with global phase factor ω^e where $\omega = \sqrt{i}$ and $e \in \mathbb{Z}_8$.

The CH representation is based on a notion of a 'control-type' Clifford operator, introduced in [109]. Their name derives from the fact they leave the all-zero computational basis state unchanged, similar to classically controlled unitaries. Examples of control-type Clifford gates include the S, CZ and CNOT gates. A control-type operator U_C can be obtained from the DCH form, for example, by concatenating U_D and U_{CNOT} layers. From this, it again follows that any stabilizer state can be generated by a CH-type circuit.

Similarly to above, we encode the initial computational basis state $\vec{\mathbf{s}}$ and the Hadamard layer U_H as n-bit binary row-vectors. The control-type layer we then encode using a stabilizer tableau, made up of $2n$ Pauli operators $U_C^\dagger X_i U_C$ and $U_C^\dagger Z_i U_C$. This tableau resembles a CHP tableau for the state $U_C \left|\vec{\mathbf{0}}\right\rangle$, where the Pauli X entries are the destabilizers and the Pauli Z entries are the stabilizers. Alternatively, we can see this as characterising the operator U_C by its action on the generators of the Pauli group.

Using a CHP tableau, as discussed in Section 2.1, each Pauli requires $2n+1$ bits to encode. However, from the definition of the control-type operators, $U_C^\dagger Z_i U_C$ will never result in a Pauli X or Y operator, as otherwise $U_C \left|\vec{\mathbf{0}}\right\rangle \neq \left|\vec{\mathbf{0}}\right\rangle$. This means we can ignore the n 'x-bits' and phase-bits of each of the Pauli Z rows. Specifically, we

write

$$U_C^\dagger Z_j U_C = \bigotimes_{k=1}^{n} Z^{G_{j,k}} \tag{2.32}$$

$$U_C^\dagger X_j U_C = i^{\gamma_j} \bigotimes_{k=1}^{n} X^{F_{j,k}} Z^{M_{j,k}} \tag{2.33}$$

for binary matrices G, F, M, and a phase vector $\gamma : \gamma_i \in \mathbb{Z}_4$, as $Y = -\mathrm{i}XZ$. Note that this differs from the CHP method, where the string 11 encodes Pauli Y directly, without tracking a separate complex phase.

Finally, we again require three further bits to encode the global phase, meaning the CH representation is given by the tuple $(e, \vec{\mathbf{s}}, \vec{\mathbf{v}}, G, M, F)$. Overall, the CH form also has spatial complexity $\theta(n^2)$. In order to optimize some subroutines, we additionally store copies of M^T and F^T, and again include the variables p and ϵ, requiring a total of $5n^2 + 4n + 36$ bits of memory.

2.2.2 Simulating circuits with the DCH and CH Representations

In this section, we will outline how to update the DCH and CH representations under different stabilizer circuit operations, and how to compute the inner product. Some of the techniques employed will be common to both representations, differing only in their implementation on the underlying data-structure.

Gate updates: The DCH Representation

In the DCH picture, the complexity of a gate depends on whether it is a CNOT, or a diagonal Clifford operator S, Z, $\mathrm{S}^\dagger$ or CZ. Diagonal gates can be simulated in constant time $O(1)$ by simply updating the linear or quadratic part of the diagonal layer. Single qubit gates applied to qubit i update the ith element of the linear phase vector D, as they contribute only to the linear part of the weighted polynomial. Thus, we have

$$S_i |\phi\rangle \implies B_{i,i} \leftarrow B_{i,i} + 1 \mod 4 \tag{2.34}$$

$$Z_i |\phi\rangle = S^2 |\phi\rangle \implies B_{i,i} \leftarrow B_{i,i} + 2 \mod 4 \tag{2.35}$$

$$S_i^\dagger = s^3 |\phi\rangle \implies B_{i,i} \leftarrow B_{i,i} + 3 \mod 4. \tag{2.36}$$

Similarly, a CZ gate applied to qubits i and j will change entries of the quadratic phase matrix as

$$B'_{i,j} \leftarrow B_{i,j} \oplus 1, \tag{2.37}$$

and equivalently for $B_{j,i}$.

For CNOT gates, we first need to commute them past the diagonal layer before updating U_{CNOT}. The overall effect on the DCH form is then

$$\begin{aligned} \mathrm{CNOT}_{c,t} |\phi\rangle &= i^e \, \mathrm{CNOT}_{c,t} U_D U_{\mathrm{CNOT}} U_H |\vec{\mathbf{s}}\rangle \\ &= i^e \, \mathrm{CNOT}_{c,t} U_D \mathrm{CNOT}^\dagger_{c,t} U'_{\mathrm{CNOT}} U_H |\vec{\mathbf{s}}\rangle \\ &= i^e \, U'_D U'_{\mathrm{CNOT}} U_H |\vec{\mathbf{s}}\rangle \end{aligned} \tag{2.38}$$

updating U_{CNOT} using matrix multiplication as in Equation. 2.28, and where the last line relies on the following Lemma:

Lemma 1 *For any CNOT circuit U_{CNOT} and any diagonal Clifford circuit U_D, $U^\dagger_{CNOT} U_D U_{CNOT}$ is also a diagonal Clifford circuit U'_D with corresponding phase matrix $B' = WBW^T$.*

Proof of Lemma 1. Consider the case of a single CNOT gate on qubits c and t. We have

$$\begin{aligned} \mathrm{CNOT}^\dagger_{c,t} U_D \mathrm{CNOT}_{c,t} |\vec{\mathbf{x}}\rangle &= \mathrm{CNOT}_{c,t} U_D \mathrm{CNOT}_{c,t} \\ &= \mathrm{CNOT}_{c,t} U_D |\vec{\mathbf{x}} + x_c \vec{\mathbf{e}}_t \mod 2\rangle \\ &= \mathrm{i}^{f(\vec{\mathbf{x}} + x_c e_t)} \mathrm{CNOT}_{c,t} |\vec{\mathbf{x}} + x_c \vec{\mathbf{e}}_t \mod 2\rangle \\ &= \mathrm{i}^{f(\vec{\mathbf{x}} + x_c e_t)} |\vec{\mathbf{x}} + 2x_c \vec{\mathbf{e}}_t \mod 2\rangle \\ &= \mathrm{i}^{f(\vec{\mathbf{x}} + x_c \vec{\mathbf{e}}_t)} |\vec{\mathbf{x}}\rangle \end{aligned} \tag{2.39}$$

where $\vec{\mathbf{e}}_t$ is a binary vector that is all-zero except at entry t, and we have used the fact that a single CNOT gate is self-inverse. Thus, $\mathrm{CNOT}^\dagger_{c,t} U_D \mathrm{CNOT}_{c,t}$ acts as a diagonal Clifford gate. As any CNOT circuit is a sequence of individual CNOT gates, $U^\dagger_C U_D U_C$ is also a diagonal Clifford circuit.

Using the matrix representation of the action of U_C, it is easy to show that

$$\begin{aligned} U_C^\dagger U_D U_C &= U_{C^\dagger} U_D \left|\vec{\mathbf{x}}W\right\rangle \\ &= \mathrm{i}^{(\vec{\mathbf{x}}W)B(\vec{\mathbf{x}}W)^T} U_C^\dagger \left|\vec{\mathbf{x}}W\right\rangle \\ &= \mathrm{i}^{(\vec{\mathbf{x}}W)B(\vec{\mathbf{x}}W)^T} \left|\vec{\mathbf{x}}WW^{-1}\right\rangle \\ &= \mathrm{i}^{\vec{\mathbf{x}}WBW^T\vec{\mathbf{x}}^t} \left|\vec{\mathbf{x}}\right\rangle, \end{aligned} \tag{2.40}$$

completing the proof. □

In general, computing the updated form of $U_{\mathrm{CNOT}}^\dagger U_D U_{\mathrm{CNOT}}$ would require time $O(n^2)$. However, for the case of a single gate $\mathrm{CNOT}_{c,t}$, recall that the matrix E differs from the identity matrix at a single element, $E_{c,t} = 1$. This allows us to simplify the updates as

$$\left[E_{c,t} B E_{c,t}^T\right]_{i,j} = \sum_{k,l} E_{i,k} E_{j,l} B_{k,l} = \begin{cases} B_{i,j} & i,j \neq c \\ B_{c,j} + B_{t,j} & i = c,\, j \neq c \\ B_{i,c} + B_{i,t} & i \neq c,\, j = c \\ B_{c,c} + B_{t,t} + B_{c,t} + B_{t,c} & i = j = c \end{cases} \tag{2.41}$$

Additionally, we need to update W and W^{-1}. The inverse of U_C is the same sequence of CNOT gates, applied in reverse order. Thus, we have $W^{-1} = E_m E_{m-1} \cdots E_1$, and we update W^{-1} by left multiplication with the CNOT matrix. Using the definition of the CNOT matrix,

$$[WF]_{ij} = \sum_k W_{i,k} F_{k,j} = \begin{cases} W_{i,j} & j \neq t \\ W_{i,c} + W_{i,t} & j = t \end{cases}$$

$$\left[FW^{-1}\right]_{i,j} = \sum_k F_{i,k} W_{k,j}^{-1} = \begin{cases} W_{i,k}^{-1} & i \neq c \\ W_{c,j}^{-1} + W_{t,j}^{-1} & i = c \end{cases}$$

updating just the target column and the control row of W and W^{-1}, respectively.

Putting together these two pieces, we thus have

$$\begin{aligned}\mathrm{CNOT}_{c,t}\,|\phi\rangle \implies & \mathrm{row}_c(B) \leftarrow \mathrm{row}_c(B) + \mathrm{row}_t(B) \\ & \mathrm{col}_c(B) \leftarrow \mathrm{col}_c(B) + \mathrm{col}_t(B) \\ & \mathrm{col}_t(W) \leftarrow \mathrm{col}_t(W) + \mathrm{col}_c(W) \\ & \mathrm{row}_c(W^{-1}) \leftarrow \mathrm{row}_c(W^{-1}) + \mathrm{row}_t(W^{-1}) \end{aligned} \tag{2.42}$$

These updates take $O(n)$ time, as we update a constant number of rows and columns.

Gate Updates: The CH Representation

For the CH representation, whenever we apply a new control-type operator C we need to update the stabilizer tableau by conjugating each element $U_C^\dagger X_i$, $Z_i U_C$ with the matrix C. This can be implemented using the usual rules for updating Pauli operators under Clifford operations, with the additional note that we have to adjust the updates to correctly track the phases of the Pauli X terms, and that we are conjugating as $U_C^{-1} P U_C$, rather than $U_C P U_C^{-1}$.

The control-type circuit is built out of individual operations $U_C = C_m C_{m-1} \ldots C_1$. We we update U_C with some new operator C_{m+1}, change the tableau as

$$(C_{m+1}U_C)^\dagger\, P C_{m+1} U_C = U_C^\dagger \left(C_{m+1}^\dagger P C_{m+1}\right) U_C. \tag{2.43}$$

Because C_{m+1} is a Clifford operator, the term $C_{m+1}^\dagger P C_{m+1}$ is also a Pauli operator $P' = i^\alpha \prod_{i=1}^n X_i^{x_i} Z_i^{z_i}$ for some phase α and bit strings $\vec{\mathbf{x}}$ and $\vec{\mathbf{z}}$. This allows us to write

$$\begin{aligned} U_C^\dagger C_{m+1}^\dagger P C_{m+1} U_C &= i^\alpha U_C^\dagger \left(\prod_{i=1}^n X_i^{x_i} Z_i^{z_i}\right) U_C \\ &= i^\alpha \prod_{i=1}^n U_C^\dagger X_i^{x_i} Z_i^{z_i} U_C \\ &= i^\alpha \prod_{i=1}^n U_C^\dagger X_i^{x_i} U_C U_C^\dagger Z_i^{z_i} U_C \\ &= i^\alpha \prod_{i=1}^n \left(i^{\gamma_i} \prod_{j=1}^n X_i^{F_{i,j}} Z_i^{M_{i,j}}\right)^{x_i} \left(\prod_{i=1}^n Z_i^{G_{i,j}}\right)^{z_i} \end{aligned} \tag{2.44}$$

where the last line is a product of terms from the tableau of U_C.

As an example, consider the action of the S gate. For each term, we have

$$S^\dagger P S = \begin{cases} I & \to & I \\ X & \to & -\mathrm{i}XZ \\ Z & \to & Z \end{cases}$$

The Z stabilizers are unchanged, and the X/Y stabilizers flip from $\mathrm{i}^\alpha X^a Z^b$ to $\mathrm{i}^{\alpha+3} X^a Z^{b\oplus 1}$. On the tableau, acting an S gate on qubit q will only act non-trivially on the term $U_C^\dagger X_q U_C$, and thus

$$U_C^\dagger S^\dagger X_q S_q U_C = i^3 U_C^\dagger X_q U_C U_C^\dagger Z_q U_C \implies \begin{cases} \mathrm{row}_q(M) & \leftarrow & \mathrm{row}_q(M) + \mathrm{row}_q(G) \\ \gamma_q & \leftarrow & \gamma_q + 3 \mod 4 \end{cases}$$

We can compute the updates for CZ and CX in the same way, giving overall gate update rules

$$\begin{aligned}
S &\begin{cases} \mathrm{row}_q(M) & \leftarrow & \mathrm{row}_q(M) + \mathrm{row}_q(G) \\ \gamma_q & \leftarrow & \gamma_q + 3 \mod 4 \end{cases} \\
CZ_{q,p} &\begin{cases} \mathrm{row}_q(M) & \leftarrow & \mathrm{row}_q(M) + \mathrm{row}_p(G) \\ \mathrm{row}_p(M) & \leftarrow & \mathrm{row}_p(M) + \mathrm{row}_q(G) \end{cases} \\
\mathrm{CNOT}_{q,p} &\begin{cases} \mathrm{row}_p(G) & \leftarrow & \mathrm{row}_p(G) + \mathrm{row}_q(G) \\ \mathrm{row}_q(F) & \leftarrow & \mathrm{row}_q(F) + \mathrm{row}_p(G) \\ \mathrm{row}_q(M) & \leftarrow & \mathrm{row}_q(M) + \mathrm{row}_p(M) \\ \gamma_q & \leftarrow & \gamma_q + \gamma_p + 2\sum_i M_{q,i} F_{p,i} \mod 4 \end{cases}
\end{aligned} \tag{2.45}$$

Where on the final line, we apply an extra phase correction that results from re-ordering the Pauli operators in the CNOT updates. This arises as, expanding out the action on the X stabilizers,

$$\begin{aligned}
U_C^\dagger \mathrm{CNOT}_{q,p} X_q \mathrm{CNOT}_{q,p} U_C &= U_C^\dagger X_q X_p U_C \\
&= U_C^\dagger X_q U_C U_C^\dagger X_p U_C \\
&= i^{\gamma_q+\gamma_p} \prod_{i=1}^{n} X_i^{F_{q,i}} Z_i^{M_{q,i}} X_i^{F_{p,i}} Z_i^{M_{p,i}}
\end{aligned}$$

and we pick up an extra phase of -1 each time $M_{q,i} = F_{p,i} = 1$ as $ZX = -XZ$. All

of these updates take time $O(n)$, as we are updating the n-element rows of $n \times n$ matrices.

Hadamard gates and Pauli Measurements

Simulating Hadamard gates and arbitrary Pauli measurements is done using an algorithm with the same general structure in the DCH and CH representation. These routines employ an algorithm developed by Sergey Bravyi for application to the CH method, which can also be applied to the DCH case.

Hadamard gates and Pauli projectors can both be written as $\frac{1}{\sqrt{2}}(P_1 + P_2)$ for some Pauli operators P_1, P_2. In the Hadamard case, we have $P_1 = X_i, P_2 = Z_i$, and in the projector case $P_1 = I, P_2 = P$. Given this structure, we then commute these operators through to the computational basis state

$$\begin{aligned}\epsilon 2^{-p/2} i^e \frac{1}{\sqrt{2}} (P_1 + P_2) U_C U_H |\vec{\mathbf{s}}\rangle &= \epsilon 2^{-(p+1)/2} i^e U_C U_H (P_1' + P_2') |\vec{\mathbf{s}}\rangle \\ &= \epsilon 2^{-(p+1)/2} i^{e'} U_C U_H \left(\left|\vec{\mathbf{t}}\right\rangle + i^\beta |\vec{\mathbf{u}}\rangle \right)\end{aligned}$$

where $P_{1,2}'$ can be efficiently computed as the circuit $U_C U_H$ is Clifford, $\beta \in Z_4$, and $\vec{\mathbf{t}}$ and $\vec{\mathbf{u}}$ are two new computational basis states obtained from the action of $P_{1,2}$ on $\vec{\mathbf{s}}$. Note that we are writing U_C here as a shorthand, as the circuit $U_D U_{\mathrm{CNOT}}$ in the DCH representation is also a control-type unitary.

Once in this form, we employ the following proposition, called Proposition 4 in [109]:

Proposition 1 *Given a stabilizer state $U_H \left(\left|\vec{\mathbf{t}}\right\rangle + i^\beta |\vec{\mathbf{u}}\rangle \right)$, we can construct a circuit W_C built out of CNOT, CZ and S gates, and a new Hadamard circuit U_H', such that we can write*

$$U_H \left(\left|\vec{\mathbf{t}}\right\rangle + i^\beta |\vec{\mathbf{u}}\rangle \right) = i^{\beta'} W_C U_H' \left|\vec{\mathbf{s}'}\right\rangle.$$

As a means of proving this proposition, we will go through and construct W_C and U_H'.

Proof of Proposition 1. Firstly, consider the case $\vec{\mathbf{t}} = \vec{\mathbf{u}}$. Then we have $\vec{\mathbf{s}'} = \vec{\mathbf{t}}$, and the result depends on the phase β. If $\beta = 0$, then the state is unchanged. If $\beta = 1, 3$,

then we have

$$\frac{1}{\sqrt{2}}U_H\left(1+i^\beta\right)\left|\vec{\mathbf{s}'}\right\rangle = \frac{(1\pm \mathrm{i})}{\sqrt{2}}U_H\left|\vec{\mathbf{s}'}\right\rangle$$

and it suffices to update the global phase term

$$\begin{aligned}\beta = 1 \quad &\Longrightarrow \quad e \leftarrow e+1 \mod 8\\ \beta = 3 \quad &\Longrightarrow \quad e \leftarrow e+7 \mod 8\end{aligned}$$

Finally, if $\beta = 2$, we have $\left|\vec{\mathbf{s}'}\right\rangle - \left|\vec{\mathbf{s}'}\right\rangle$ and the state is canceled out. We denote this by setting $\epsilon \leftarrow 0$. This only arises in the case of applying a Pauli projector that is orthogonal to the state.

If $t \neq u$, then we instead note that we can always define some sequence of CNOT gates V_C such that

$$\left|\vec{\mathbf{t}}\right\rangle = V_C\left|\vec{\mathbf{y}}\right\rangle \quad \left|\vec{\mathbf{u}}\right\rangle = V_C\left|\vec{\mathbf{z}}\right\rangle$$

where $\vec{\mathbf{y}}, \vec{\mathbf{z}}$ are two n-bit binary strings such that $y_i = z_i$ everywhere except bit q where $z_q = y_q + 1$. We can assume without loss of generality that $\exists q : t_q = 0, u_q = 1$, else we swap the two strings and update the phase accordingly. Then

$$V_C = \prod_{i:i\neq q,\, t_i \neq u_i} \mathrm{CNOT}_{q,i}$$

and we can commute this circuit past U_H to obtain a new circuit V_C'. We can always freely pick $q : v_q = 0$, unless $v_i = 1 \forall i$, and thus V_C' is given by:

$$V_C' = \begin{cases} \prod_{i\neq q,\, v_i=0} \mathrm{CNOT}_{q,i} \prod_{i\neq q,\, v_i=1} CZ_{q,i} & v_q = 0 \\ \prod_{i\neq q} \mathrm{CNOT}_{i,q} & v_i = 1 \forall i \end{cases}$$

We complete the proof by considering the action of U_H on the new strings $\left|\vec{\mathbf{y}}\right\rangle + \mathrm{i}^\beta \left|\vec{\mathbf{z}}\right\rangle$. Again, fixing $y_q = 0, z_q = 1$, we can write

$$U_H\left(\left|\vec{\mathbf{y}}\right\rangle + i^\beta\left|\vec{\mathbf{z}}\right\rangle\right) = H^{v_q}S^\beta\left|+\right\rangle = \omega^a S_q^b H_q^c\left|d\right\rangle$$

for some bits $a, b, c, d \in \{0, 1\}$ that can be computed exactly from the values of β and v_q.

This completes the proof of Proposition 1, where $W_C = V_C' S_q^b$, $U_H' = U_H H_q^{v_q + c}$, and $\vec{\mathbf{s}'} = \vec{\mathbf{y}} \oplus d\vec{\mathbf{e}}_q$. □

Computing the circuits W_C and U_H' given the two strings t, u takes time $O(n)$, as it involves inspecting the $n-bit$ strings t, u and $\vec{\mathbf{v}}$. Given this proposition, we now need to show how to commute a Pauli operator through the stabilizer circuit in both representations, and then how to update the layers $U_D U_{\mathrm{CNOT}}$ and U_C by right multiplication with the circuit W_C. This can be rewritten in terms of binary vector-matrix multiplication, and we introduce the following notation:

$$\prod_{i=1}^n X_i^{x_i} \equiv X(\vec{\mathbf{x}}) \quad \prod_i Z_i^{z_i} \equiv Z(\vec{\mathbf{z}})$$

for binary strings $\vec{\mathbf{x}}$ and $\vec{\mathbf{z}}$.

Applying Proposition 1 to DCH States

When commuting a Pauli operator P through a Clifford circuit, it is important to fix the ordering of the X and Z terms, as Pauli operators can be expanded out as $P = i^a X(\vec{\mathbf{x}}) Z(\vec{\mathbf{z}}) = i^a (-1)^{x \cdot z} Z(\vec{\mathbf{z}}) X(\vec{\mathbf{x}})$, as $XZ = -ZX$, and where we use $\vec{\mathbf{x}} \cdot \vec{\mathbf{z}}$ to denote the binary inner product

$$\vec{\mathbf{x}} \cdot \vec{\mathbf{z}} = \sum_i x_i z_i \mod 2.$$

In the DCH case, we fix $P = i^a Z(\vec{\mathbf{z}}) X(\vec{\mathbf{x}})$, as this simplifies the phase terms when commuting past the U_D layer.

Pauli Z terms are unchanged by the DCH layer as they commute with diagonal Clifford operators. To commute the X terms past the U_D layer, we use $X(\vec{\mathbf{x}}) U_D = U_D \left(U_D^\dagger X(\vec{\mathbf{x}}) U_D \right)$, and compute the new Pauli $U_D^\dagger P' U_D = i^{a'} Z(\vec{\mathbf{z}}') X(\vec{\mathbf{x}})$.

The diagonal entries of the phase matrix B contribute as

$$(S^{B_{ii}})^\dagger X_i^{x_i} S^{B_{ii}} = \begin{cases} S^\dagger X^{x-i} S & \to & i(ZX)^{x_i} \\ ZXZ & \to & -X^{x_i} \\ SXS^\dagger & \to & -i(ZX)^{x_i} \end{cases} = i^{B_{ii}} X^{x_i} Z^{x_i B_{ii}} \ (\mathrm{mod}\ 2)$$

We also have that $CZ(X \otimes I)CZ = XZ$, $CZ(I \otimes X)CZ = ZX$, i.e. a CZ conjugated

with a Pauli X on the control (target) qubit adds a Pauli Z on the target (control) qubit. Qubit i picks up a Z operator each time there is a CZ between qubits i and j, and an X acting on qubit j. Using the off-diagonal entries of the phase matrix, we can write

$$Z_i^{z_i'} : \vec{\mathbf{z}}' = \sum_{j \neq i} x_j B_{j,i} \mod 2$$

Combining this with the fact we also pick up a Pauli Z from the diagonal if $B_{ii} = 1, 3$, we can write $z_i = aB \mod 2$. Finally, we need to consider the extra -1 phase contributions for each $i : x_i z_i' = 1$, as a result of preserving the ordering of P'. Together with the diagonal phases, this can be simplified to

$$\sum_i x_i B_{ii} + 2 \sum_i x_i \sum_{j \neq i} x_j B_{j,i} = \vec{\mathbf{x}} B \vec{\mathbf{x}}^T \mod 4$$

Overall then, we have

$$U_D^\dagger X(\vec{\mathbf{x}}) U_D = i^{\vec{\mathbf{x}} M \vec{\mathbf{x}}^T} Z(\vec{\mathbf{x}} M) X(\vec{\mathbf{x}}) \tag{2.46}$$

A similar result applies to commuting a Pauli operator through the U_{CNOT} layer. CNOT has the property that it maps $I_c Z_t \to Z_c Z_t$ and $X_c I_t \to X_c X_t$ under conjugation. Thus, we can compute the new strings $\vec{\mathbf{x}}', \vec{\mathbf{z}}'$ by applying an appropriate CNOT matrix.

For the X bits, we can simply apply $\vec{\mathbf{x}}' = \vec{\mathbf{x}} W^{-1}$, where we use the inverse matrix as we are computing $U_{\text{CNOT}}^\dagger X U_{\text{CNOT}}$ and thus the binary string is subject to the inverse sequence of CNOT gates.

For the string $\vec{\mathbf{z}}$, we need to apply a CNOT matrix with the controls and targets swapped. From the definition given in Equation. 2.27, we can see that if the binary matrix E encodes $\text{CNOT}_{c,t}$, then $\text{CNOT}_{t,c}$ is encoded by E^T. We then update the string $\vec{\mathbf{z}}$ under the sequence $E_m^t E_{m-1}^t \dots E_1^t = W^T$. This gives

$$U_{\text{CNOT}}^\dagger i^a Z(\vec{\mathbf{z}}) X(\vec{\mathbf{x}}) U_{\text{CNOT}} = i^a Z(\vec{\mathbf{z}} W^T) X(\vec{\mathbf{x}} W^{-1}). \tag{2.47}$$

As mentioned, we store copies of W^{-1} and W^T with the DCH representation. This helps to avoid the $O(n^3)$ computational cost associated with inverting W, and the

$O(n^2)$ cost of transposing W. We can thus compute this update in time $O(n^2)$.

Finally, to commute a Pauli operator past the U_H layer, we note that the Hadamard acts as

$$\begin{aligned} HXH &\to Z \\ HZH &\to X \\ HZXH &\to -ZX \end{aligned}$$

The $\vec{\mathbf{x}}$ and $\vec{\mathbf{z}}$ bits are only changed for those bits where $v_i = 1$, and so we can write

$$z_i' = z_i(1 - v_i) + x_i v_i$$

and vice-versa for the $\vec{\mathbf{x}}$ bits. In terms of boolean operations, this can also be written as $z_i' = z_i \wedge \neg v_i \oplus x_i \wedge v_i$. Finally, we have the phase correction whenever $x_i = z_i = z_i = 1$. Thus, overall, we can write

$$U_H^\dagger i^a Z(\vec{\mathbf{z}})X(\vec{\mathbf{x}})U_H = i^{a+\vec{\mathbf{v}}\cdot(\vec{\mathbf{x}}\wedge\vec{\mathbf{z}})} Z((\vec{\mathbf{z}} \wedge \neg\vec{\mathbf{v}}) \oplus (\vec{\mathbf{x}} \wedge \vec{\mathbf{v}})\, X((\vec{\mathbf{x}} \wedge \neg\vec{\mathbf{v}}) \oplus (\vec{\mathbf{z}} \wedge \vec{\mathbf{v}})) \quad (2.48)$$

and this update takes time $O(n)$ to compute.

To complete the application of Proposition 1, we also need to be able to update $U_D U_{\mathrm{CNOT}}$ by right multiplication with W_C. We can split $W_C = W_{CNOT} W_D$, where W_D is made up of CZ gates and the single S gate.

The U_{CNOT} layer updates as $U'_{\mathrm{CNOT}} = U_{\mathrm{CNOT}} W_{CNOT}$. Because of the ordering of the circuits, we here update the matrix W by left multiplication, and update $W^\dagger$ by right multiplication. Thus, for each CNOT gate in W_{CNOT}, we update the columns of W^{-1} and the rows of W using the rules given in Equation. 2.47.

We then need to commute the diagonal layer W_D past U'_{CNOT}. We can do this by adapting Equation. 2.40 to instead compute $U_{\mathrm{CNOT}} W_D U_{\mathrm{CNOT}}^\dagger$, giving a new phase matrix $C' = W^{-1} C W^{-1}$ where C encodes the action of W_D. This computation again benefits from storing W^{-1} in the DCH information, and can be further optimized by noting that many entries of C are zero. Finally, we can combine the two phase matrices by simply adding all the elements, keeping the diagonal entries $\bmod 4$ and the off-diagonal entries $\bmod 2$. All together, including the Pauli updates, applying

Proposition 1 takes time $O(n^2)$.

Applying Proposition 1 to CH States

Commuting a Pauli operator through the layers of the CH circuit can be done using methods already introduced in previous sections. Distinctly from the DCH case, here we fix $P = i^a X(\vec{\mathbf{x}})Z(\vec{\mathbf{z}})$.

To commute a Pauli past the U_C layer, we need to compute $U_C^\dagger P U_C$, and this can be expanded out in a similar manner to Equation. 2.44. This gives

$$\begin{aligned} U_C^\dagger X(\vec{\mathbf{x}})U_C = &\quad \textstyle\prod_{i:x_i=1} U_C^\dagger X_i U_C \\ U_C^\dagger Z(\vec{\mathbf{z}})U_C &\quad \textstyle\prod_{i:z_i=1} U_C^\dagger Z_i U_C \end{aligned}$$

We can thus build up P' term by term as

$$\begin{aligned} U_C^\dagger P U_C &= \prod_{j=1}^{n} x_j \left(\mathrm{i}^{\gamma_j} X(\mathrm{row}_j(F))Z(\mathrm{row}_j(M))\right) \prod_{j=1}^{n} z_i \left(Z(\mathrm{row}_j(G))\right) \\ &= \mathrm{i}^{\sum_{j=1}^{n} x_j\gamma_j + 2\sum_{j=1}^{n}\sum_{k>j} x_j x_k (\mathrm{row}_j(F)\cdot\mathrm{row}_j(M))} X(\vec{\mathbf{x}}F)Z(\vec{\mathbf{x}}M + \vec{\mathbf{z}}G) \\ &= \mathrm{i}^{\vec{\mathbf{x}}J\vec{\mathbf{x}}^T} X(\vec{\mathbf{x}}F)Z(\vec{\mathbf{x}}M + \vec{\mathbf{z}}G). \end{aligned} \tag{2.49}$$

The extra factor of 2 in the phase arises from having to commute the Pauli Z terms in $U_C^\dagger X_j U_C$ past the following Pauli X terms. We can encode these commutation relations as a binary matrix

$$MF^T : \left[MF^T\right]_{i,j} = \mathrm{row}_i(M)\cdot\mathrm{row}_j(F),$$

which is additionally symmetric as

$$\left[U_C^\dagger X_j U_C, U_C^\dagger X_k U_C\right] = [X_j, X_k] = 0.$$

Similar to the way we encode the phase polynomial in the DCH form, we can then simplify the overall phase calculation as

$$\vec{\mathbf{a}}J\vec{\mathbf{a}}^T : [J]_{i,j} = \begin{cases} \gamma_i & i = j \\ MF^T_{i,j} & i \neq j \end{cases}$$

where we pick up the correct factor of 2 from the symmetric nature of MF^T. Computing each of the matrix-vector multiplications to commute past U_C takes $O(n^2)$ time. We can then use the same update rule as for the DCH form to commute the Pauli operator past the U_H layer.

Finally, to finish applying Proposition 1, we need to update the tableau of U_C to $U_C W_C$. We have

$$(U_C W_C)^\dagger X_i, Z_i (U_C W_C) = W_C^\dagger \left(U_C^\dagger X_i, Z_i U_C \right) W_C$$

an thus we need to update the Paulis in the tableau by conjugation with CNOT, CZ and S gates. These rules for updating U_C by right-multiplication with a control type unitary are the same as for the CHP tableau, with some additional corrections for phase.

$$\begin{aligned}
S \begin{cases} \mathrm{col}_q(M) & \leftarrow & \mathrm{col}_q(M) + \mathrm{col}_q(G) \\ \gamma & \leftarrow & \gamma - \mathrm{col}_q(F) \bmod 4 \end{cases} \\
CZ_{q,p} \begin{cases} \mathrm{col}_q(M) & \leftarrow & \mathrm{col}_q(M) + \mathrm{col}_p(F) \\ \mathrm{col}_p(M) & \leftarrow & \mathrm{col}_p(M) + \mathrm{col}_q(F) \\ \gamma & \leftarrow & \gamma + \mathrm{col}_p(F) \cdot \mathrm{col}_q(F) \end{cases} \\
\mathrm{CNOT}_{q,p} \begin{cases} \mathrm{col}_q(G) & \leftarrow & \mathrm{col}_q(G) + \mathrm{col}_p(G) \\ \mathrm{col}_p(F) & \leftarrow & \mathrm{col}_p(F) + \mathrm{col}_q(F) \\ \mathrm{col}_q(M) & \leftarrow & \mathrm{col}_q(M) + \mathrm{col}_p(M) \end{cases}
\end{aligned} \tag{2.50}$$

There are $O(n)$ row and column updates to perform, and thus this final step runs in time $O(n^2)$. Overall, then, the complexity of applying Proposition 1 to the CH form is $O(n^2)$, arising from computing $U_C^\dagger P U_C$ and then updating the tableau under W_C.

Sampling Pauli Measurements with Proposition 1

Proposition 1 can also be extended to apply to sampling measurements of arbitrary Pauli operators. Measuring a Pauli operator P is closely related to applying a projector $\Pi_{\pm P} = \frac{1}{\sqrt{2}}(I \pm P)$. As mentioned previously, there are three possible

outcomes for a Pauli measurement

$$\begin{aligned} \Pi_{+P}\left|\phi\right\rangle = \left|\phi\right\rangle \quad & P\left|\phi\right\rangle = \left|\phi\right\rangle \quad \text{Deterministic Outcome } +1 \\ \Pi_{+P}\left|\phi\right\rangle = 0 \quad & P\left|\phi\right\rangle = -\left|\phi\right\rangle \quad \text{Deterministic Outcome } -1 \\ \Pi{+}P\left|\phi\right\rangle = \left|\phi\right\rangle + \left|\varphi\right\rangle \quad & P\left|\phi\right\rangle = \left|\varphi\right\rangle \quad \text{Random Outcome} \end{aligned}$$

In terms of measuring an operator P, then we can begin by commuting the projector $I+P$ through the Clifford circuit as described in the previous sections. Dropping the normalisation, we have

$$\begin{aligned} (I+P)\,V\left|\vec{\mathbf{s}}\right\rangle &= V\left(I+V^{\dagger}PV\right)\left|\vec{\mathbf{s}}\right\rangle \\ &= V\left(\left|\vec{\mathbf{s}}\right\rangle + P'\left|\vec{\mathbf{s}}\right\rangle\right) = V\left(\left|\vec{\mathbf{s}}\right\rangle + i^{\beta}\left|\vec{\mathbf{s}'}\right\rangle\right) \end{aligned}$$

which is the equivalent to the statement of Proposition 1, with $\vec{\mathbf{t}} = \vec{\mathbf{s}}$ and $\vec{\mathbf{u}} = \vec{\mathbf{s}'}$.

If $\vec{\mathbf{s}} = \vec{\mathbf{s}'}$, then the measurement outcome is deterministic. As we have used the projector Π_{+P}, the measurement outcome is $+1$ unless $\beta = 2$, in which case the outcome is -1. Otherwise, if $\vec{\mathbf{s}} \neq \vec{\mathbf{s}'}$, the measurement outcome is random and equiprobable. We can sample the ± 1 outcome using random number generation techniques, and then apply the corresponding projector $(I \pm P)$. As computing P' takes in general $O(n^2)$ time, deciding on the measurement outcome also takes $O(n^2)$ time. However, compare to other stabilizer simulators, we note that this algorithm works for arbitrary Pauli operators P as opposed to just single-qubit Pauli Z measurements.

Computational Amplitudes and Sampling Output Strings

Commuting Pauli operators through the layers of control-type operators can also be used to compute the probability of a given computational basis state. Recall that a control-type Clifford circuit U_C is defined such that $U_C\left|\vec{\mathbf{0}}\right\rangle = \left|\vec{\mathbf{0}}\right\rangle$. Recall also that for the DCH representation, U_D and U_{CNOT} are also a control-type operators. Thus,

$$\begin{aligned} \left\langle\vec{\mathbf{0}}\middle|\phi\right\rangle &= w^e\left\langle\vec{\mathbf{0}}\middle|U_C U_H\middle|\vec{\mathbf{s}}\right\rangle \\ &= w^e\left(\left\langle\vec{\mathbf{0}}\right|U_C\right)U_H\left|\vec{\mathbf{s}}\right\rangle \\ &= w^e\left\langle\vec{\mathbf{0}}\middle|U_H\middle|s\right\rangle. \end{aligned}$$

This trick, using the definition of a control-type operator to simplify the inner product, can be extended to any computational basis state. Writing $\left|\vec{\mathbf{t}}\right\rangle = X(t)\left|\vec{\mathbf{0}}\right\rangle$, we can then commute the X operators past the control-type layer(s) to obtain

$$\begin{aligned}\langle t|U_C U_H|s\rangle &= \left\langle\vec{\mathbf{0}}\right|P' U_H\left|\vec{\mathbf{s}}\right\rangle \\ &= \left\langle\vec{\mathbf{0}}\right|i^{\mu} Z(\vec{\mathbf{z}'})X(\vec{\mathbf{x}'})U_H\left|\vec{\mathbf{s}}\right\rangle = \left\langle\vec{\mathbf{x}'}\right|U_H\left|\vec{\mathbf{s}}\right\rangle \end{aligned} \tag{2.51}$$

where we have used the 'ZX' convention in the definition of the Pauli operator. If instead we use the 'XZ' convention, then we pick up an additional phase factor of $-1^{\vec{\mathbf{x}}'\cdot\vec{\mathbf{z}}'}$.

The action of the Hadamard layer on a computational basis state can be expanded out as

$$U_H\left|\vec{\mathbf{s}}\right\rangle = 2^{-|\vec{\mathbf{v}}|/2}(-1)^{\vec{\mathbf{s}}\cdot\vec{\mathbf{v}}}\sum_{\vec{\mathbf{x}}\leq\vec{\mathbf{v}}}(-1)^{\vec{\mathbf{s}}\cdot\vec{\mathbf{x}}}\left|\vec{\mathbf{s}}\oplus\vec{\mathbf{x}}\right\rangle \tag{2.52}$$

where $\vec{\mathbf{x}}\leq\vec{\mathbf{v}}$ denotes the binary strings $\vec{\mathbf{x}}: x_i = v_i \iff v_i = 0$ and $|\vec{\mathbf{v}}|$ is the Hamming weight of the string $\vec{\mathbf{v}}$. Thus, we have overall that

$$\left\langle\vec{\mathbf{t}}\right|\left.\phi\right\rangle = 2^{-|v|/2} i^{\mu}\prod_{j:v_j=1}(-1)^{x'_j s_j}\prod_{j:v_j=0}\left\langle x'_j\right|\left.s_j\right\rangle, \tag{2.53}$$

which equals 0 if any $u_j \neq s_j$ for $v_j = 0$, and is proportional to $2^{-|v|/2}$ otherwise. As this requires commuting a Pauli operator through the C/DC layer (s), computing these amplitudes takes time $O(n^2)$.

This result can also be extended to sample strings from the probability distribution $P(x) = \left|\left\langle\vec{\mathbf{t}}\right|V\left|\vec{\mathbf{s}}\right\rangle\right|^2$, where V_C is a Clifford circuit such that $V_C = U_C U_H \equiv U_D U_{\mathrm{CNOT}} U_H$. From the above, we know that any string with a non-zero amplitude occurs with equal probability. This, it is sufficient to start with a binary string

$$\vec{\mathbf{w}}: w_j = \begin{cases} s_j & v_j = 0 \\ 0 & \text{otherwise} \end{cases}$$

and then pick each of the remaining $|\vec{\mathbf{v}}|$ bits at random with equal probability.

Computing Inner Products

The computational basis are a special case of stabilizer state inner products. Here, we present a general method for computing inner products $\langle\varphi|\phi\rangle$ using the DCH and CH forms. Both methods proceed by combining the two control-type layers, and then breaking down the computation into a sum of different computational basis state amplitudes

$$\begin{aligned}\langle\varphi|\phi\rangle &= \langle t|V_H V_C^\dagger U_C U_H|\vec{\mathbf{s}}\rangle \\ &= \left\langle\vec{\mathbf{t}}\middle|V_H\middle|\Phi\right\rangle : |\Phi\rangle = V_C^\dagger|\phi\rangle\,.\end{aligned}$$

Proposition 2 *Given a stabilizer inner product of the form*

$$\left\langle\vec{\mathbf{t}}\middle|V_H\middle|\Phi\right\rangle$$

where $|\Phi\rangle$ is encoded in DCH or CH form, we can compute the inner product by computing the computational state ampliude $\left\langle\vec{\mathbf{t}}\middle|\Phi'\right\rangle$ where $|\Phi'\rangle = V_H|\Phi\rangle$, in time $O(n^3)$.

Proof of Proposition 2. In both the DCH and CH form, we can simulate the action of a single Hadamard gate in time $O(n^2)$. The Hadamard circuit V_H contains at most n Hadamard gates, and so we can compute $V_H|\Phi\rangle$ in time $O(n^3)$. The amplitude then reduces to computing the amplitude $\left\langle\vec{\mathbf{t}}\middle|\Phi'\right\rangle$, which takes time $O(n^2)$. The overall worst-case complexity is thus $O(n^3)$. □

This method bares a strong resemblance to the 'basis circuit' method described in [105], with the advantage that the 'basis circuit' is explicitly stored in the DCH and CH data-structures, rather than needing to be computed from a tableau. In the following sections, we will show how to compute $|\Phi\rangle$ from the DCH/CH data of $|\varphi\rangle$ and $|\phi\rangle$.

The DCH Case

In this representation, we need to compute $U'_D U'_{\mathrm{CNOT}} = V_{CNOT}^\dagger V_D^\dagger U_D U_{\mathrm{CNOT}}$.

We begin by combining the two phase layers, noting that

$$U_D^\dagger |\vec{\mathbf{x}}\rangle = \mathrm{i}^{-\vec{\mathbf{x}}B\vec{\mathbf{x}}^T} |\vec{\mathbf{x}}\rangle$$

and thus given the two phase matrices A, B, the phase matrix encoding the combined circuit is

$$V_D^\dagger U_D |\vec{\mathbf{x}}\rangle = \mathrm{i}^{\vec{\mathbf{x}}(A-B)\vec{\mathbf{x}}^T} |\vec{\mathbf{x}}\rangle$$

where, as per the definition, the subtraction is $\bmod 2$ on the off-diagonal entries and $\bmod 4$ on the diagonal entries.

We then need to commute $V_{CNOT}^\dagger$ past the new U_D' layer, and combine it with U_{CNOT}. As this circuit is an inverse, it is characterised by the binary matrix Q^{-1}, and its inverse is Q. Thus

$$\begin{aligned} B' &\leftarrow Q^{-1}B'Q \\ W &\leftarrow WQ^{-1} \\ W^{-1} &\leftarrow QW^{-1} \end{aligned} \tag{2.54}$$

Altogether then, the updated DCH information of $|\Phi\rangle$ can be computed in time $O(n^2)$.

The CH Case

Given two tableau describing control-type unitaries V_C and U_C, we can combine them using Equation. 2.49, as

$$\begin{aligned} (V_C U_C)^\dagger X_j V_C U_C &= U_C^\dagger \left(V_C^\dagger X_j V_C\right) U_C \\ &= \mathrm{i}^{\gamma_j'} U_C^\dagger P U_C \\ &= \mathrm{i}^{\gamma_j' + \mathrm{row}_j(F')J\mathrm{row}_j(F')^T} X(\mathrm{row}_j(F')F) Z(\mathrm{row}_j(M')M), \end{aligned}$$

and similarly for the Z_j entries. Combining two tableau in this way will require time $O(n^3)$, as there are $2n$ entries and each update takes time $O(n^2)$. However, to compute the tableau of $|\Phi\rangle$, we will require the following Lemma:

Lemma 2 *Given the tableau of a control type operator U_C, specified by the*

binary matrices F, M and G, then the inverse tableau has matrices G', F' and M' such that

$$\begin{aligned} G' &\equiv G^{-1} \\ F' &\equiv G^T \\ M' &\equiv M^T. \end{aligned} \tag{2.55}$$

Proof of Lemma 2. The entries of the tableau for $U_C^\dagger$ have the property

$$U_C \left(U_C^\dagger X_j, Z_j U_C\right) U_C^\dagger = U_C^\dagger \left(U_C X_j, Z_j U_C^\dagger\right) U_C = X_j, Z_j$$

Consider first the Pauli Z terms. Using Equation. 2.49, can see that

$$U_C \left(U_C^\dagger Z_j U_C\right) U_C^\dagger = Z(\mathrm{row}_j(G) G') = Z_j$$

for all $j \in \{1, 2, \ldots, n\}$. Expanding out this requirement, we can see that $\mathrm{row}_j(G) \cdot \mathrm{col}_k(G') = \delta_{jk} \forall\ j, k$. If we change the order of the multiplications, we obtain the additional constraint $\mathrm{row}_j(G') \cdot \mathrm{col}_k(G) = \delta_{jk}$. We thus require that

$$GG' = G'G = I \tag{2.56}$$

and thus, $G' = G^{-1}$.

A feature of CHP tableaux is that the jth stabilizer and destabilizer anti-commute. Here, similarly

$$U_C^\dagger X_j U_C\, U_C^\dagger Z_k U_C = (-1)^{\delta_{jk}}\, U_C^\dagger Z_k U_C\, U_C^\dagger X_j U_C$$

where the extra phase arises from the commutation relations of Pauli operators. In terms of the entries of the tableau, this tells us that

$$\mathrm{row}_j(F) \cdot \mathrm{row}_k(G) = \delta_{jk} \forall\, j, k \implies F G^T = I.$$

This also holds for the tableau of $U_C^\dagger$. From this, we can conclude that $F = \left(G^{-1}\right)^T$, and similarly $F' = G^T$.

Finally, consider the X_j entries. Again applying Equation. 2.49, we have

$$U_C \left(U_C^\dagger X_j U_C \right) U_C^\dagger = X(\text{row}_j(F)F')Z(\text{row}_j(F)M' + \text{row}_j(M)G') = X_j.$$

As the Pauli Z terms cancel, we have

$$\begin{aligned} &\text{row}_j(F) \cdot \text{col}_k(M') + \text{row}_j(M) \cdot \text{col}_k(G') = 0 \; \forall j,k \\ &\qquad \implies \text{row}_j(F) \cdot \text{col}_k(M') = \text{row}_j(M) \cdot \text{col}_k(G') \; \forall j,k. \end{aligned}$$

Using $F^T = (G^{-1})$, and Equation. 2.56, we thus have

$$\text{row}_j(F) \cdot \text{col}_k(M') = \text{row}_j(M) \cdot \text{row}_k(F) \; \forall j,k \implies M_{j,k} = M'_{k,j} \tag{2.57}$$

completing the proof. □

Special case for 'Equatorial' Stabilizer States

As part of the simulation routines we will introduce in the following chapters, are are especially interested in computing the inner product when the state $|\varphi\rangle$ is of the form

$$|\varphi\rangle = \sum_{\vec{\mathbf{x}} \in \mathbb{Z}_2^n} i^{\vec{\mathbf{x}} A \vec{\mathbf{x}}^T} |\vec{\mathbf{x}}\rangle$$

a superposition of all 2^n computational basis states with relative phases. We call these 'equatorial' stabilizer states, as they are like n-qubit generalisations of single qubit states $|0\rangle + \mathrm{e}^{i\theta}|1\rangle$ which lie on the equator of the Bloch sphere. As such, we introduce an optimized routine for these kind of inner products.

Claim 1 *If $|\varphi\rangle$ is an equatorial state, we can instead write the inner product as*

$$\langle \phi | \varphi \rangle = 2^{-(n+|\vec{\mathbf{v}}|)/2} \mathrm{i}^{\vec{\mathbf{s}} K \vec{\mathbf{s}}^T + 2\vec{\mathbf{s}} \cdot \vec{\mathbf{v}}} \sum_{\vec{\mathbf{x}} \in \mathbb{Z}_2^{|\vec{\mathbf{v}}|}} \mathrm{i}^{\vec{\mathbf{x}} K(1,1) \vec{\mathbf{x}}^T + 2\vec{\mathbf{x}}[\vec{\mathbf{s}} + \vec{\mathbf{s}} K](1)^T} \tag{2.58}$$

where $\vec{\mathbf{s}}(1)$ denotes the elements of a vector $s_j : v_j = 1$, and $K(1,1)$ is the sub-matrix with rows i and columns j such that $v_i = v_j = 1$.

Proof of Claim 1. Let us assume that, given a control-type unitary $U_C \equiv U_D U_{\text{CNOT}}$, we can write $U_C^\dagger |\varphi\rangle = \sum_{\vec{\mathbf{x}} \in \mathbb{Z}_2^n} i^{\vec{\mathbf{x}} K \vec{\mathbf{x}}^T} |\vec{\mathbf{x}}\rangle$ for an appropriate phase matrix K. We will show in the following section how to construct this matrix K given the CH and DCH representation of a state $|\phi\rangle$. Given this form then, we have

$$\begin{aligned}\langle\varphi|\phi\rangle &= (\langle\phi|\varphi\rangle)^* \\ &= 2^{-n/2} \left(\sum_{\vec{\mathbf{x}} \in \mathbb{Z}_2^n} \mathrm{i}^{\vec{\mathbf{x}} K \vec{\mathbf{x}}^T} \langle \vec{\mathbf{s}} | U_H | \vec{\mathbf{x}} \rangle \right)^*\end{aligned}$$

Using Equation. 2.52 to expand out the left hand side of this expression, we obtain a sum over terms

$$\sum_{\vec{\mathbf{x}} \in \mathbb{Z}_2^n} \mathrm{i}^{\vec{\mathbf{x}} K \vec{\mathbf{x}}^T} \langle \vec{\mathbf{s}} | U_H | \vec{\mathbf{x}} \rangle = 2^{-|\vec{\mathbf{v}}|/2} (-1)^{\vec{\mathbf{s}} \cdot \vec{\mathbf{v}}} \sum_{\vec{\mathbf{y}} < \vec{\mathbf{v}}} (-1)^{\vec{\mathbf{s}} \cdot \vec{\mathbf{y}}} \sum_{\vec{\mathbf{x}} \in \mathbb{Z}_2^n} \mathrm{i}^{\vec{\mathbf{x}} K \vec{\mathbf{x}}^T} \langle \vec{\mathbf{s}} \oplus \vec{\mathbf{y}} | \vec{\mathbf{x}} \rangle$$

From the orthogonality of computational basis states, we can set $\vec{\mathbf{x}} = \vec{\mathbf{s}} \oplus \vec{\mathbf{y}}$ and drop all other terms in the sum. Doing so changes the phase calculation to

$$(\vec{\mathbf{s}} \oplus \vec{\mathbf{v}}) K (\vec{\mathbf{s}} \oplus \vec{\mathbf{y}})^T = \vec{\mathbf{s}} K \vec{\mathbf{s}}^T + \vec{\mathbf{y}} K \vec{\mathbf{y}}^T + \vec{\mathbf{y}} K \vec{\mathbf{s}}^T + \vec{\mathbf{s}} K \vec{\mathbf{y}}^T = \vec{\mathbf{s}} K \vec{\mathbf{s}}^T + \vec{\mathbf{y}} K \vec{\mathbf{y}}^T + 2 \vec{\mathbf{y}} K \vec{\mathbf{s}}^T$$

where the final equality follows from the symmetric nature of K. From the the definition of $\vec{\mathbf{y}} \leq \vec{\mathbf{v}}$, $y_j = 0 \iff v_j = 0$. Thus, we can take the global phase of $\vec{\mathbf{s}} K \vec{\mathbf{s}}^T$ out and reduce the sum to the sum over strings $\vec{\mathbf{y}} \in \mathbb{Z}_2^{|\vec{\mathbf{v}}|}$, as in Claim 1.

To complete the proof, we need to show how to obtain K in both cases. In the DCH form, we have

$$\langle \phi | \varphi \rangle = \langle \vec{\mathbf{s}} | U_H U_{\text{CNOT}}^{-1} U_D^{-1} | \varphi \rangle .$$

Using the definition of an equatorial stabilizer state, we can write $|\varphi\rangle = V_D \left| +^{\otimes n} \right\rangle$, and simply compute $|\varphi'\rangle = U_D^{-1} V_D \left| +^{\otimes n} \right\rangle$ by combining the two phase layers to obtain a new phase matrix $(A - B)$.

Another feature of the state $\left|+^{\otimes n}\right\rangle$ is that it is invariant under CNOT circuits, as it is a superposition of all computational basis states and subsequently invariant under their permutation. Applying Lemma 1, we can commute the circuit U_{CNOT}^{-1} past $U'_D = U_D^{-1} V_D$ and eliminate it. This gives a new phase matrix $K = G(A-B)G^T$.

In the CH case, using Equation. 2.49, we can write

$$U_C^{-1}\left|\vec{\mathbf{x}}\right\rangle = U_C^{-1} X(\vec{\mathbf{x}}) U_C \left|\vec{\mathbf{0}}\right\rangle = \mathrm{i}^{\vec{\mathbf{x}} J \vec{\mathbf{x}}^T}\left|\vec{\mathbf{x}} F\right\rangle$$

Applying this to $|\varphi\rangle$ thus gives

$$U_C^{-1} \sum_{\vec{\mathbf{x}}\in\mathbb{Z}_2^n} \mathrm{i}^{\vec{\mathbf{x}} A \vec{\mathbf{x}}^T}\left|\vec{\mathbf{x}}\right\rangle = \sum_{\vec{\mathbf{x}}\in\mathbb{Z}_2^n} \mathrm{i}^{\vec{\mathbf{x}}(A+J)\vec{\mathbf{x}}^T}\left|\vec{\mathbf{x}} F\right\rangle.$$

Using $FG^T = I$, as introduced in the previous section, and setting $\vec{\mathbf{x}} = \vec{\mathbf{y}} G^T$, we have

$$\sum_{\vec{\mathbf{y}}\in\mathbb{Z}_2^n} \mathrm{i}^{\vec{\mathbf{y}} G^T (A+J) G \vec{\mathbf{y}}^T}\left|\vec{\mathbf{y}}\right\rangle = \sum_{\vec{\mathbf{y}}\in\mathbb{Z}_2^n} \mathrm{i}^{\vec{\mathbf{y}} K \vec{\mathbf{y}}^T}\left|\vec{\mathbf{y}}\right\rangle$$

as required where $K = G^T(A+J)G$. □

Once the calculation is in this form, we can compute the inner product in time $O(|\vec{\mathbf{v}}|^3)$ using the algorithm for exponential sums developed by Sergey Bravyi [109]. Computing the phase matrix K takes time $O(n^2)$ in both cases, and thus as $|\vec{\mathbf{v}}| \leq n$ we have a general performance $O(n^3)$.

2.2.3 Implementations in Software

The DCH and CH data structures and most routines were implemented in `C++`, to produce a stabilizer circuit simulator. The one exception was the arbitrary stabilizer state inner product, which was derived but left unimplemented due to time constraints. In this section, we will review some of the optimizations employed, and present data comparing their performance with existing software implementations.

The resulting simulators were also validated through the use of testing ran-

Property	CH	DCH	CHP	Canonical	Graph States [104]
Memory	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(nd)$
Z	$O(n)$	$O(1)$	$O(n)$	$O(n^2)$	$O(1)$
X	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(1)$
S	$O(n)$	$O(1)$	$O(n)$	$O(n^2)$	$O(1)$
H	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$
CZ	$O(n)$	$O(1)$	$O(n)$	$O(n^2)$	$O(d^2)$
CX	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(d^2)$
Measurement	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(d^2)$
Inner Product	$O(n^3)$	$O(n^3)$	$O(n^3)$	$O(n^3)$	N/A

Table 2.1: Comparison of the asymptotic complexity of different stabilizer circuit simulators, including common operations and their memory footprint. We include the graph based representation of Anders & Briegel, discussed later in this section, and omit the 'Affine Space' simulator as it has no current implementation for gate updates.
Here, d is the degree of the graph used as an internal representation, which varies from $\log n$ to n [104]. We further note that, while all algorithms for measurement are in principle extensible beyond single qubit measurements, only the DCH and CH simulators currently implement arbitrary Pauli measurements.

dom circuits. The CH representation was validated by comparison to a `MATLAB` version of the simulator developed independently by David Gosset. The DCH representation was then validated against this successfully tested CH representation, using random circuits and conversion to state-vectors through 2^n calls of the computational amplitude routine.

Efficient Binary Operations

The data-structures and subroutines underpinning the CH and DCH representations are built out of arithmetic performed modulo 2 and 4, depending on the context. This allows us to efficiently store the representations using binary bits as opposed to integers, and then use boolean operations as part of the simulation routines.

Addition and subtraction modulo 2, such as is required in the U_C updates of the CH representation and the U_D updates in the DCH representation, is

equivalent to the boolean 'XOR' operation, defined as

a	b	$a \oplus b$	$a+b \,(\mathrm{mod}\, 2)$	$a-b \,(\mathrm{mod}\, 2)$
0	0	0	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	0	0

For addition modulo 4, we encode each number using two binary bits a and b as $2*a+b$. In this context, a is typically referred to as the '2s' bit and b as the '1s' bit. Addition can be done for the 1s and 2s terms separately, with an additional carry correction

$$x+y \pmod 4 = 2*(a_x \oplus a_y \oplus (b_x \wedge b_y)) + (b_x \oplus b_y)\,.$$

In the case of subtraction modulo 4, we note that adding and subtracting 2 can be achieved using just the *xor* operation, as only the two bit is changed. Otherwise, we note that

a	$a-3 \pmod 4$	$a-1 \pmod 4$
0	1	3
1	2	0
2	3	1
3	0	2

i.e. $a-3 = a+1$, and $a-1 = a+3$, where the addition is again modulo 4. This trick allows us to simplify $a-b \pmod 4$ by setting $b_2 \leftarrow b_2 \oplus b_1$, and then using addition.

Vector and matrix multiplications modulo 2 can also be reduced to a set of binary operations. Each element $[aM]_i]$, $[LM]_{i,j}$ can be written as a binary inner product, respectively $a \cdot \mathrm{col}_i(M)$ and $\mathrm{row}_i(KL) \cdot \mathrm{col}_j(M)$. Computing the binary inner product can then be expanded out in terms of boolean operations

as

$$x \cdot y = (x_1 \wedge y_1) \oplus (x_2 \wedge y_2) \cdots \oplus (x_n \wedge y_n) .$$

Typically, we are applying the same operation to entire vectors, rows or columns of a binary matrix. Thus, we can employ a technique called 'bit-packing' to efficiently store and update these binary values. In `C++`, integers can by stored using 8, 16, 32 or 64 binary bits (1, 2, 3 and 4 bytes, respectively). The built-in in `bool` data-type is also typically stored using 1 byte, as this is the smallest unit of memory addressable by a processor [112].

Bitpacking instead stores up to 64 binary bits in a single variable, manipulating them through the use of 'bitwise' operators [113]. Bitpacking typically achieves an 8-fold reduction in the memory footprint. Additionally, a bitwise operation between two variables acts on all bits simultaneously in a single time-step. For example, considering the XOR between two binary vectors, we can write

$$\vec{\mathbf{x}} \oplus \vec{\mathbf{y}} = [x_1 \oplus y_1, \cdots x_n \oplus y_n] \iff \texttt{uint64\_t z = x ^ y //bitwise XOR}$$

We can also make use of so called 'intrinsic' functions to optimise computing the binary inner product, and sums of terms modulo 4. Intrinsic functions allow certain special processor instructions to be called directly. Specifically, we use two intrinsics for calculating the hamming weight and the parity of a binary string, each of which are computed in a single time step. Using these operations, we can write the binary inner product as

$$\sum_i x_i y_i = |x \wedge y| \bmod 2 \iff \texttt{parity(x \& y)}$$

and a sum of integers modulo 4 as

$$2 * \sum_i a_i + \sum_i b_i \iff \texttt{(2*parity(2bits)+hamming\_weight(1bits))\%4}$$

where `%` is the `C++` modulo operator.

Using these operations allows us to reduce the effective complexity of many

common subroutines by a factor of n, as long as the number of variables n is less than 64, the largest available integer on most modern computers. For example, instead of $O(n)$ time, computing the binary inner product now requires just two operations: a bitwise logical AND, and the parity intrinsic. However, above 64 bits, we need to pack the bits across multiple variables, and so the number of calls to intrinsic functions will again asymptotically as $O(n)$. Specifically, the number of operations required will go as $n/64$.

Case study: Stabilizer simulations with Affine Spaces

As an example of the use of bitpacking to optimize stabilizer simulators, we developed a `C++` implementation of the stabilizer state simulator introduced in Appendices B, C and E of [49]. While not a full simulator, they provide explicit algorithms for performing Pauli measurements and computing stabilizer inner products. These methods were implemented by the authors in `MATLAB`, using matrices of integers and repeated calls to the `mod` function in MATLAB.

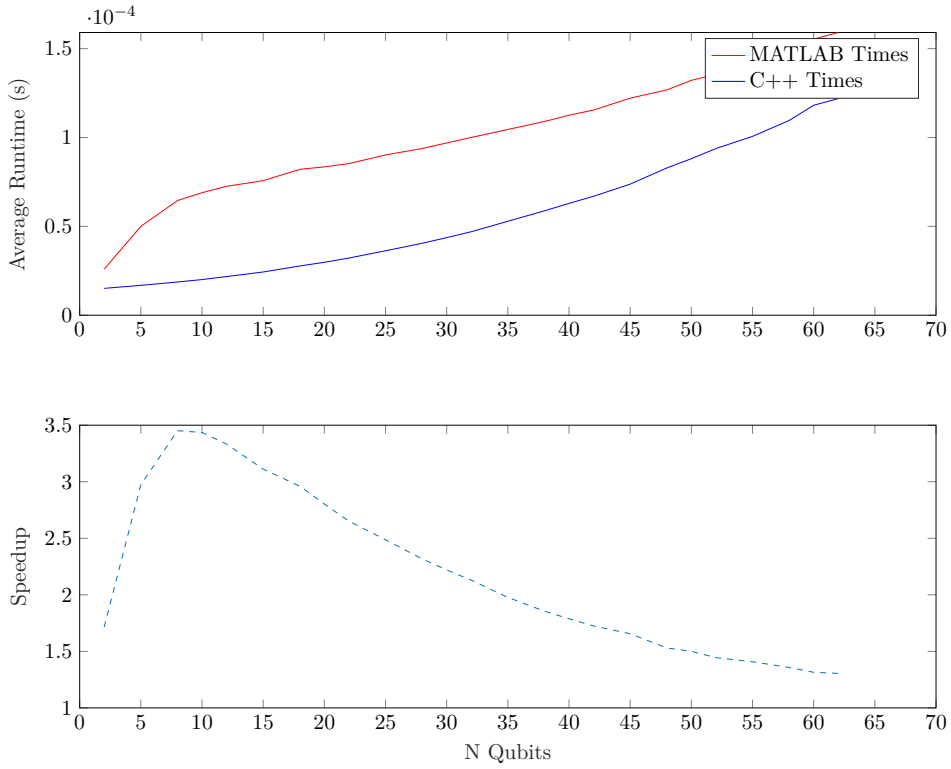
In particular, in their encoding a stabilizer state is based on Equation. 2.21, described by a tuple

$$|\phi\rangle = \left(n, k, \vec{\mathbf{h}}, G, G^{-1}, Q, D, J\right)$$

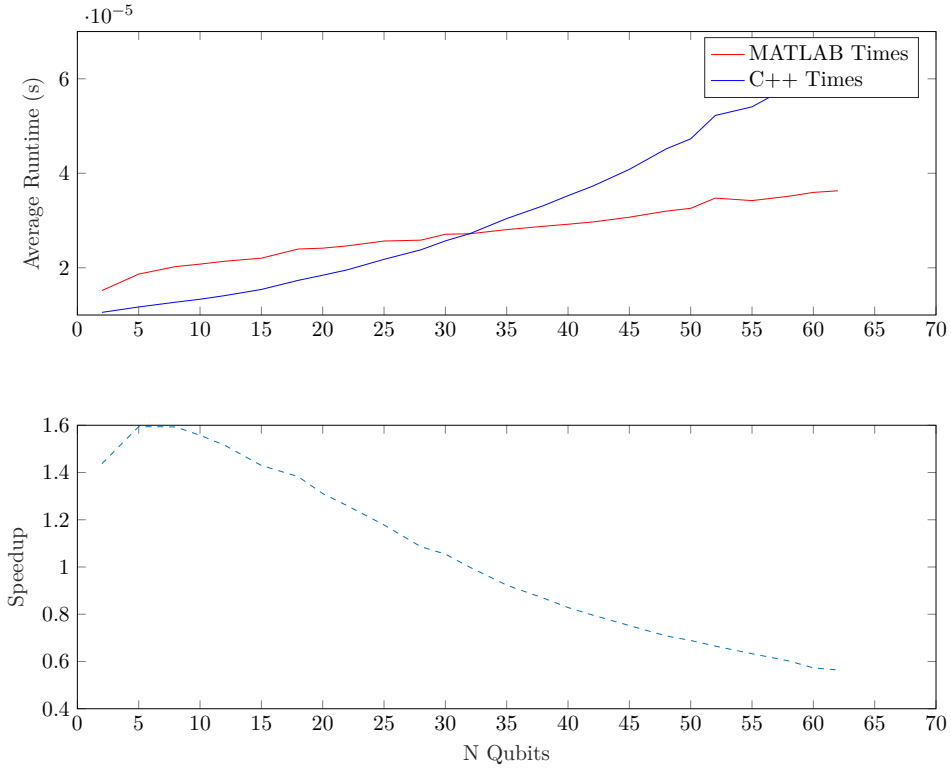
where n is the number of qubits, k is dimension of the the affine space $\mathcal{K}$, generated by the first k columns of the $n \times n$ binary matrix G and an n-bit binary vector $\vec{\mathbf{h}}$. The inverse matrix G^{-1} is also stored. The phase terms are encoded in a quadratic form using a constant offset $Q \in \mathbb{Z}_4$, a vector D of elements $\bmod 4$, and a symmetric $n \times n$ binary matrix J.

The `C++` simulator makes use of bitpacking to efficiently store $\vec{\mathbf{h}}$, G, G^{-1} and J. Additionally, we store the elements of D using two binary variables, separating the 1s and 2s bits. The routines were verified and benchmarked against the existing `MATLAB` implementation using the `MATLAB` EXternal languages (MEX) interface, which allows compiled code to be called from within `MATLAB` applications [114].

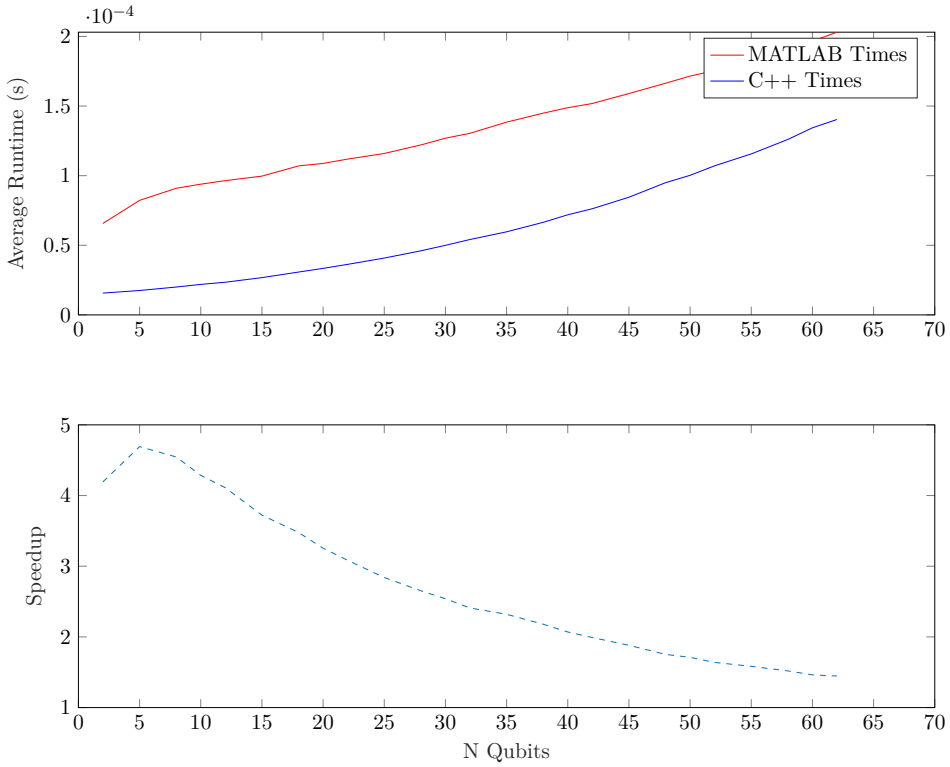
Figure 2.3: Figures showing the performance of the `MATLAB` and `C++` implementations of a stabilize simulator based on Affine Spaces.



(a) Average runtime and resulting speedup of the `Shrink` routine.

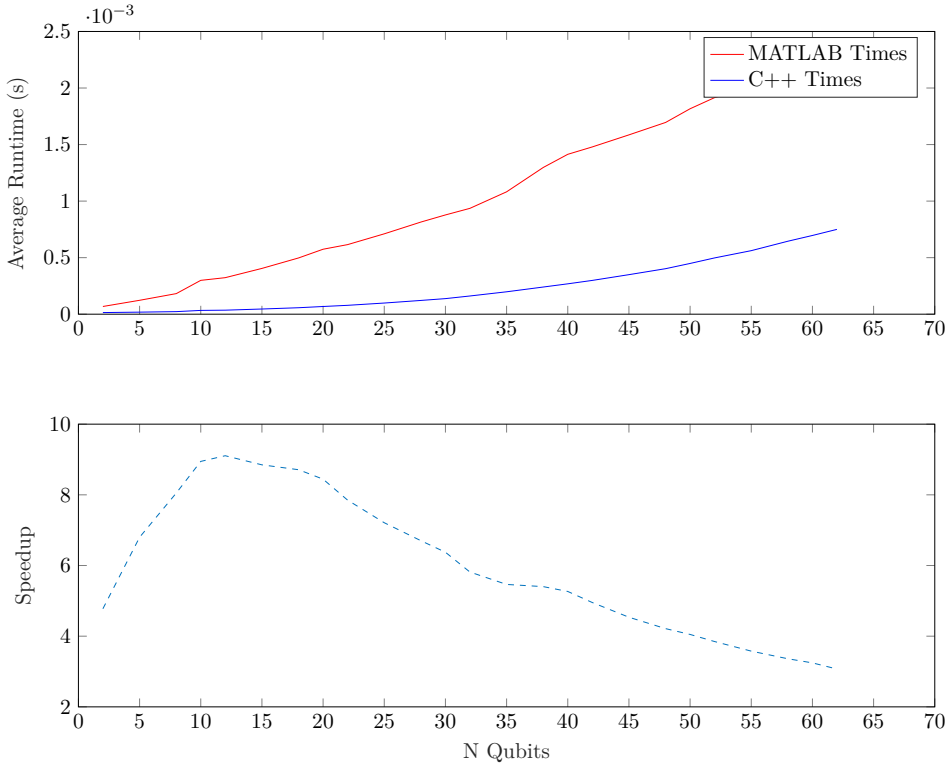


(b) Average runtime and resulting speedup of the `Extend` routine.



(c) Average runtime and resulting speedup of Pauli measurements.

(d) Average runtime and resulting speedup of stabilizer inner products.



The results of the benchmark are shown in Figure 2.3. We include two core subroutines specific to the affine space simulator, called `Shrink` and `Extend`, defined in Appendices B and E of [49]. `Shrink` computes the intersection between an affine space and a binary vector, and is called as part of computing stabilizer inner products. `Extend` instead computes the new affine space obtained by adding a new basis vector, and is called as part of simulating Pauli measurements. We also present results for arbitrary n qubit Pauli measurements and computing the inner product between stabilizer states.

The observed differences in runtime are relatively consistent across each routine. In general, the `C++` implementation has a significant advantage in the 5–15 qubit range, with a speedup of anywhere from 1.6 to 10 times. This advantage then drops off as the number of qubits increases, tending to a constant speedup of between 1.5 to 3 times. The notable exception to this is in the `Extend` routine, which actually performs worse than the `MATLAB` version above 35 qubits. All benchmarks have a cutoff below 64 qubits, which is enforced by the use of 64 bit integers for bitpacking in the `C++` simulator.

Specific Optimizations for the CH and DCH Forms

We make use of bitpacking to efficiently store the CH and DCH forms. As many subroutines require computing vector-matrix multiplications of the form aM, we store the matrices in 'column format' where each bitpacked variable stores one column of the binary matrix. This allows us to make use of intrinsic functions to speedup these multiplications.

Transposed matrices are computed using 'lazy evaluation'. When the transposed matrix is required, we compute it and store it. We then additionally store a flag to indicate if the transposed matrix is up to date. If later function calls change the values of the transposed matrix, the flag is set to false and the transpose will be recomputed only when required.

Whenever the result of a calculation is expected to be symmetric, we an halve the number of operations by copying values across. This gives a constant

factor speedup in, for example, computing the phase matrices K as part of inner product calculations. We can also make use of this symmetric structure to avoid transposing a matrix when accessing a row.

Typically, phase matrices are stored as binary matrices with 0 diagonal, and then a separate pair of bitpacked variables storing the diagonal entries which are modulo 4. When required, we update the diagonals separately using an explicit expansion of the matrix multiplications.

Some updates for the DCH form are further optimised by using explicit expansions of the matrix multiplications. For example, when commuting a Pauli Z through the CNOT layer as in Equation. 2.47, we avoid a call to the transpose W^T by noting that

$$[\vec{\mathbf{z}}W^T]_i = \sum_j z_j W^T_{j,i} = \sum_j z_j W_{i,j}$$

i.e. each entry $[\vec{\mathbf{z}}W^T]_i$ is a sum of some entries in row i. We can thus build up the new vector $\vec{\mathbf{z}}' = \vec{\mathbf{z}}W^T$ by repeatedly doing $\vec{\mathbf{z}}' \leftarrow \vec{\mathbf{z}}' \oplus \mathrm{col}_j(W)$ for each $j : z_j = 1$.

2.2.4 Performance Benchmarks

To establish the performance of the DCH and CH implementations, we benchmark them against two existing stabilizer circuit simulators, which are available publicly online. The first is the `C` implementation of the CHP method, developed by Scott Aaronson [106]. This uses a variant of bitpacking based on 32-bit integers. The second method is a radically different representation of stabilizer states, based on the fact that any stabilizer state can be generated by a local Clifford circuit (single qubit Clifford gates), acting on a special class of stabilize state called a graph state [115, 116].

Graph states are named as their structure is described by a mathematical graph of vertices V and edges E, where each qubit is a vertex. From this

graph, a graph-state is then built-up as

$$|(V,E)\rangle = \left(\prod_{i,j\in E} CZ_{i,j}\right)\left|+^{\otimes n}\right\rangle,$$

by performing a CZ gate between every pair of qubits connected by an edge of the graph [116].

The so called 'Anders & Briegel' simulator describes a stabilizer state by its corresponding graph, and by sequences of local Clifford operators acting at each vertex. A `C++` implementation of this simulator also exists, called `GraphSim` [117]. This stores a graph as a list of vertices, each with local information about the vertices connected to it.

The expected runtime of different routines using the Anders & Briegel method are also given in Table 2.1. Importantly, in their analysis, routines are quoted with a runtime that scales as d, the maximum 'degree' or number of edges involving a given vertex. By definition, $d \leq n$, the number of vertices in the graph, and thus the simulator has a worst case performance comparable to the DCH, CH and tableau methods. However, this analysis makes explicit a feature of stabilizer circuit simulators; their runtime in practice depends on the state/circuit being considered.

This phenomenon was first described in [66], who observed that the runtime for Pauli measurements seemingly varied between linear and quadratic scaling in the number of qubits, despite the expected asymptotic quadratic scaling. In particular, the algorithm for computing a given measurement in the CHP representation requires between 1 and n calls to a subroutine which takes $O(n)$ to evaluate, and the exact number is determined by the sparsity of the X-bits of the stabilizers, which is in turn related to the number of entangling gates in the circuit.

Similar results hold in detailed analysis of the CH and DCH representations, where the exact number of calculations required will depend on the sparisty of the matrices/vectors encoding different features of the stabilizer circuits.

Consider for example the inner product algorithm of Proposition 2, where we need to apply $|\vec{\mathbf{v}}|$ H gates at a cost of $O(n^2)$ each.

As a result, Aaronson & Gottesman introduced a heuristic for evaluating stabilizer circuit simulators. We begin by applying a random stabilizer circuit to the state, choosing H, S and CNOT gates at random, before applying the operation we are benchmarking and recording the runtime. Using an argument based on message passing, the authors claim that in general we need $O(n \log n)$ gates in the circuit to observe this transition between easier and harder instances of stabilizer circuit simulation, and so we apply $\beta n \log n$ gates where β is a parameter that varies between 0.5 and 1.2. This heuristic is also employed by Garcia et al. in their paper presenting an algorithm for computing stabilizer inner products, where they observe a transition between quadratic and cubic scaling with varying β [105].

Here we present results comparing the performance of different operations between the DCH, CH, CHP and GraphSim methods, for different values of the parameter β. All run-times are averages taken over 100000 repetitions, where we first apply a random stabilizer circuit of $\beta n \log n$ gates, and then record the time taken by the particular operation.

We also present data for routines specific to the DCH and CH routines. In particular, we present data demonstrating the runtime of arbitrary n-qubit Pauli measurements, and for the specialized 'equatorial' inner product defined in Claim 1. We also consider the effect of weight on the complexity of Pauli measurements.

2.3 Discussion

In this chapter, we have introduced two new representations for simulating stabilizer circuits, including their implementation in software, and presented data evaluating their performance against previous methods.

In particular, we make use of bitpacking techniques to try and further improve their runtime. Figure 2.3 introduced results comparing a bitpacked simulator

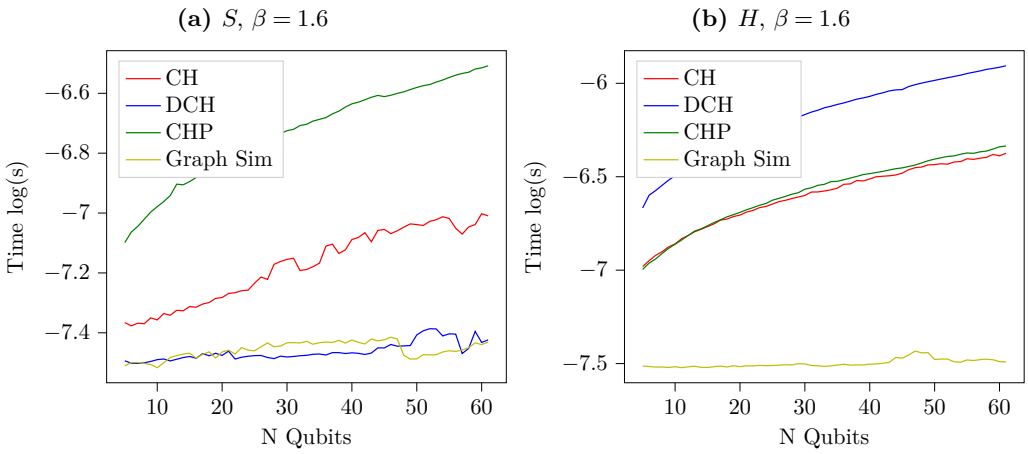


Figure 2.4: Average runtime of the single qubit H and S gates as a function of the number of qubits across different stabilizer simulators. Single qubit gates show no dependence on length of the preceding circuit, encoded as the β parameter.

Figure 2.5: Average runtime of entangling CNOT and CZ gates as a function of the number of qubits for different stabilizer simulators, for extremal values of β. The Anders & Briegel method shows a significant dependence on circuit length.

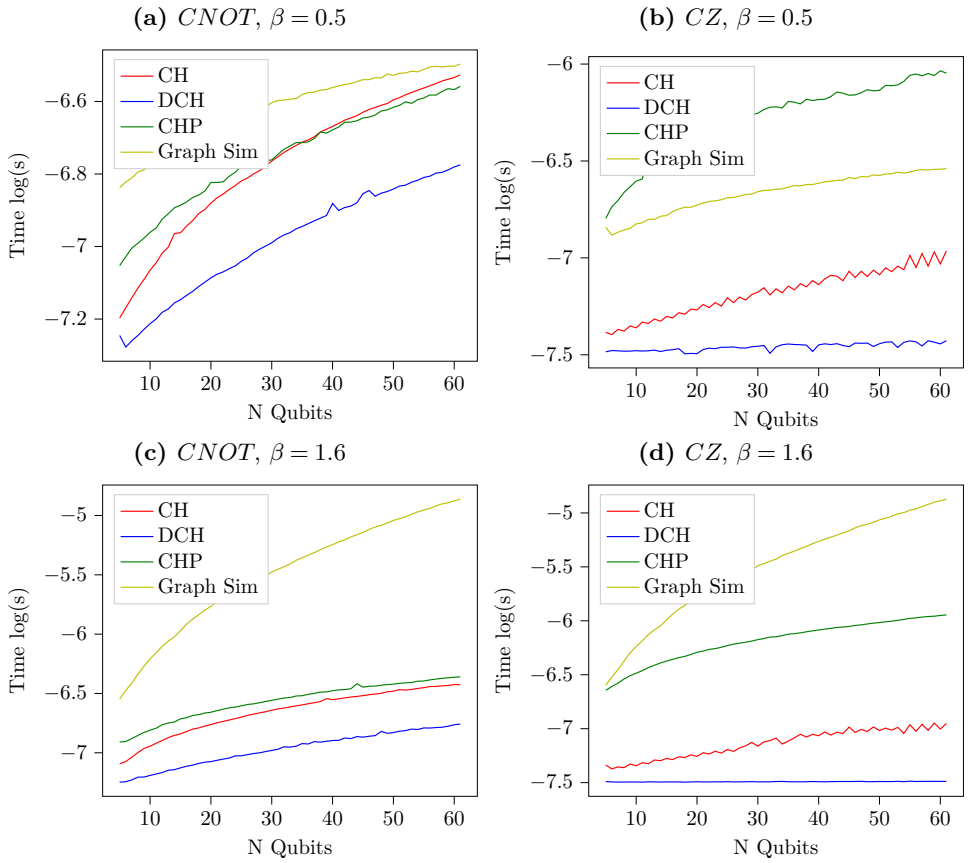
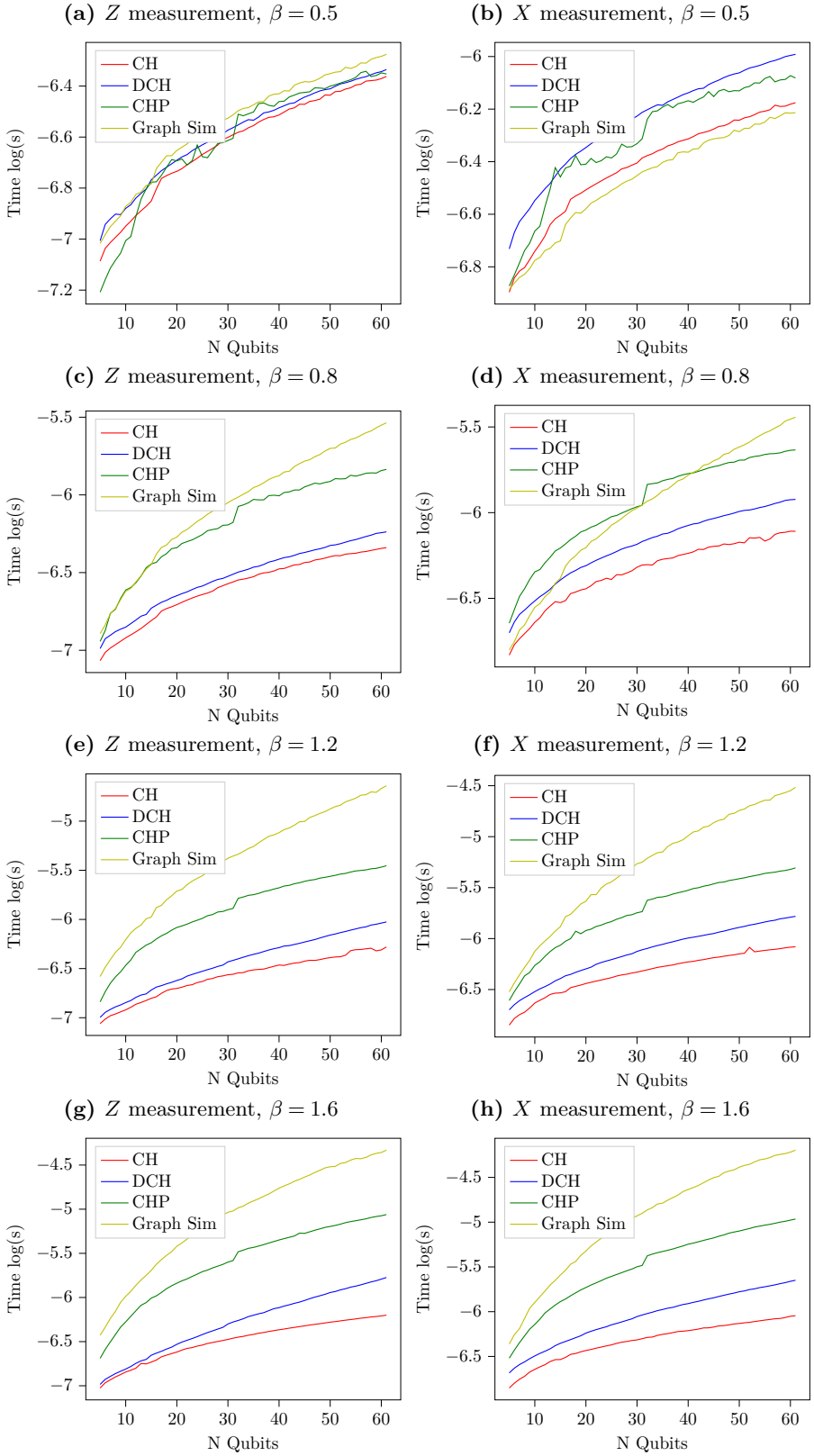


Figure 2.6: Average runtime of single qubit measurements in the X and Z basis, as a function of the number of qubits and the length of the preceding stabilizer circuit, for 4 stabilizer simulators.



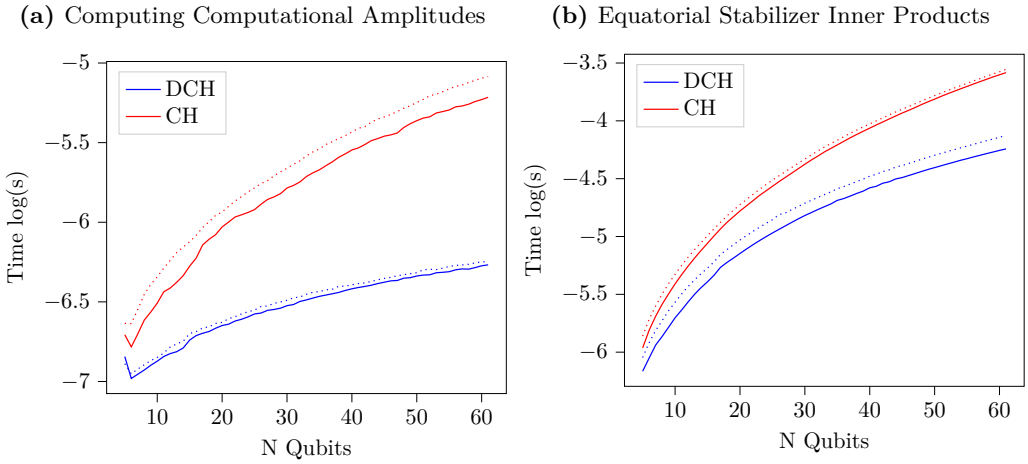
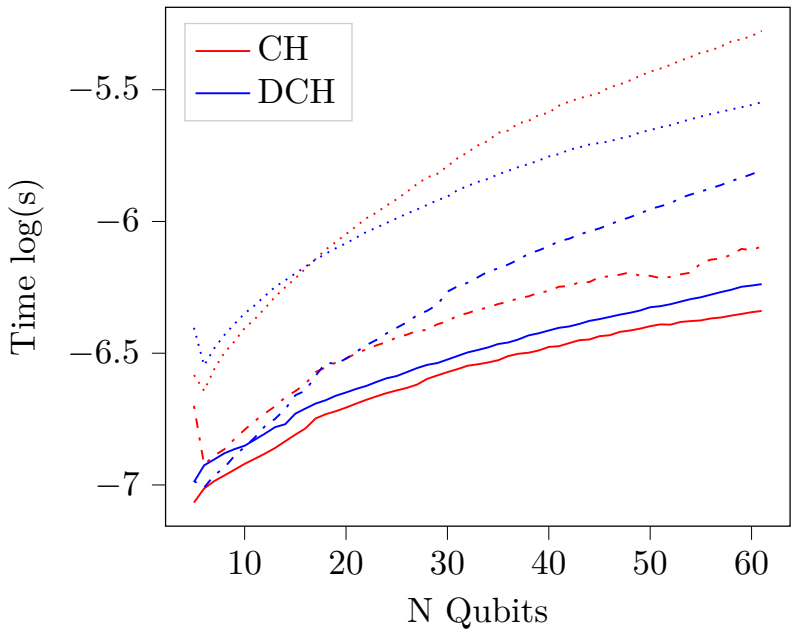


Figure 2.7: Average runtime of two routines specific to the DHC and CH routines, as a function of the number of qubits. Solid lines are for $\beta = 0.5$, and dash lines for $\beta = 1.6$. A slight dependence on circuit length is observed.

Figure 2.8: Average runtime of Pauli measurements for the CH and DCH simulators. A solid line represents a single Pauli Z measurement. The dashed lines represent n-qubit Pauli Z measurements, and the dotted line random n-qubit Paulis.



with a prior `MATLAB` implementation. In general, we see a broad speedup over the `MATLAB` version across the full parameter range, though the exact degree of this speedup decreases with increasing n. The main exception is the `Extend` routine, where we observe the `C++` implementation scaling roughly quadratically with the input size, whereas the `MATLAB` version exhibits a closer to linear scaling.

One possible explanation for this effect is an unfortunate side-effect of the use of MEX files, namely that the `C++` version additionally needs to convert the MATLAB data into a C++ data-structure. This adds an additional $O(n^2)$ overhead to the runtime of the `C++` simulator. Otherwise as coded, the `Extend` algorithm has only $O(n)$ steps. In more complex functions like measurement, `Shrink` and inner products, which have run-times $10-100$ times longer than `Extend`, this effect is less significant, but nonetheless likely contributes to the steeper gradient of the `C++` scalings.

The difference in performance is most significant for the inner product routine, which has an overall complexity that scales as $O(n^3)$ resulting from up to n calls to the `Shrink` routine, and a call to Sergey Bravyi's Exponential Sum routine which also has runtime $O(n^3)$. In this case, the effect of the additional data-copying is suppressed by the overall runtime of the algorithm.

It is important to note that the `MATLAB` implementations also benefit from a degree of parallelization, through a combination of multi-threading, and so called 'Single Instruction stream Multiple Data stream' (SIMD) operations [118]. Matrix and vector multiplications are intrinsically parallelisable, as each element in the result is computed from a unique set of multiplication and addition operations. One option for optimising parallel code is to make multiple 'threads' available to the program, which each tackle a different part of the computation. However, as we are frequently performing lots of identical operations over different inputs, they can also benefit from SIMD CPU instructions. These are optimizations which speedup computations by loading multiple values into a special shared binary registers, applying a common operation to the entire

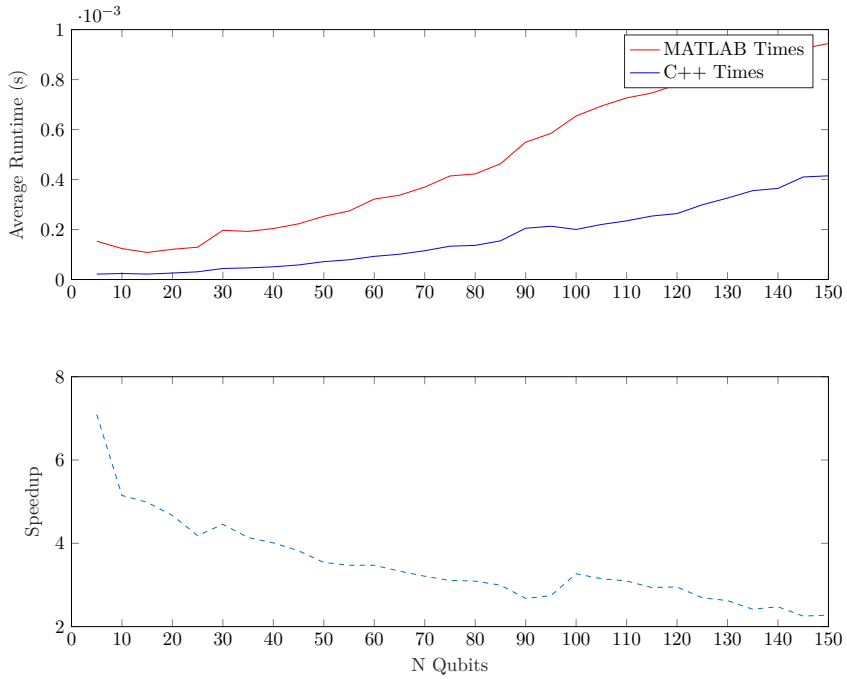


Figure 2.9: Figure comparing the runtime of the `C++` and `MATLAB` implementations of Sergey Bravyi's Exponential Sum routine, up to 150 qubits.

register, and then reading the result back out [119]. `MATLAB` is built atop the long established `LAPACK` and `BLAS` libraries for linear algebra, which implement these types of optimization [120, 121, 122].

The effect of these optimizations becomes apparent when we try to extend a bitpacked simulation beyond 64 qubits. In this case, we need to use an array of integer values to encode each binary vector, and each operation now incurs the overhead of looping over these arrays. As an example, Figure 2.9 shows the runtime of the Exponential Sum algorithm of [109], extended up to 150 qubits. We choose Exponential Sum for this benchmark as it has a complexity that scales as $O(n^3)$, reducing the impact of the MEX interface on performance. As before, the speedup shown by the `C++` implementation continually decreases with increasing n. Given that some measure of performance improvement is expected by virtue of using a compiled language, compared with the dynamic language `MATLAB`, we can see that the bitpacking method is no longer providing significant speedup.

It would also be possible to further optimize the implementation developed

here with the addition of SIMD operations. Instead of looping over each integer variable used to encode large bitpacked vectors, the variables could instead be loaded into SIMD registers. THis would significantly optimize the computations up to 512 qubits, as 512 bits is the largest register currently supported [119]. An SIMD implementation is outside the scope of this thesis, but would be a significant performance upgrade to the CH and DCH simulators.

CH and DCH Performance

Comparisons between the DCH and CH forms and previous stabilizer simulators are shown in Figures 2.4, 2.5 and 2.6. Broadly, we see that the DCH and CH representations are competitive with previous techniques, in spite of tracking additional phase information and offering additional 'functionality'.

Specifically, for single qubit Clifford gates, we see that the Graph Sim method has the best overall performance. Because applying a single qubit operator in this picture only requires updating 'local' information, it can be implemented using a lookup table and thus has constant complexity. This is a significant advantage over the other methods.

However, as mentioned in Table 2.1, the graph based data-structure employed by Anders & Breigel has a runtime that scales as the maximum degree d of the graph for entangling gates, as they alter this underlying graph. This effect becomes clear with increasing β, where the complexity of graph and subsequently the runtime of entangling gates significantly increases. At the largest tested value, 61 qubits, the runtime of an entangling gate grew from $\approx 3\text{x}10^{-7}$ at $\beta = 0.5$, to $\approx 1\text{x}1-^{-5}$ at $\beta = 1.6$. In contrast, we note that the CHP, CH and DCH methods also show no apparent dependence on β. This would be expected from the update algorithms, which rely on binary operations that are independent of the sparsity of the data structures.

The DCH also benefits from a constant time complexity for all phase gates, leading to its improved performance for the 'CZ' gate. The CH representation has no constant time operations, but is broadly competitive in terms of single qubit gate performance. This is especially true in the case of the Hadamard

gate, in spite of the theoretical $O(n^2)$ complexity of this operation. However, the DCH representation shows a significantly increased overhead in simulating Hadamard gates. This suggests the simulator is a poor choice for circuits involving many basis changes.

The origin of this increased overhead can potentially be explained by comparing the performance of Pauli measurements, where the CH simulator also out-performs the DCH method. This suggests that the additional overhead is incurred when commuting Pauli operators through the circuit layers. We might also expect that applying the circuit correction of Proposition 1 is slower for the DCH form, as it involves explicit matrix operations. In contrast, the CH form here requires only column updates, which take a single time-step as we store binary matrices as bitpacked column matrices.

The effect of commuting Paulis can be clarified by also considering Figure 2.8. We see that the CH method has a significant advantage for both single and n qubit Z-rotations, but that the DCH method shows slightly better performance for arbitrary Pauli operators. This likely follows from the need to compute a transpose of the F and M matrices, whereas the DCH method is optimized to avoid then need for transposition.

Transposition is also likely the cause of the increased overhead incurred by the CH representation in computing the equatorial inner products, and in computing computational state amplitudes, shown in Figure 2.7. Importantly, as discussed before, transposed matrices are stored 'lazily', computed only when required and then cached until outdated. Thus, in computing multiple amplitudes or inner products as is likely in a practical simulation, this performance gap between the two representations would likely decrease.

An interesting feature of computing computational basis state amplitudes and equatorial inner products is that they do not show only a small dependence on the length of the preceding stabilizer circuit. This is in contrast to the results of [105], which observed a transition from quadratic to cubic scaling

in the number of qubits when computing stabilizer inner products, even for computational state amplitudes. This would be expected from the implementation of both routines, making use of intrinsic functions. These allow us to avoid inspecting matrices and vectors element-wise, instead operating on rows and columns at a time, and thus makes us less sensitive to the sparsity of the DCH/CH encoding.

Finally, if we consider simulating of Pauli measurements, we again observe that as implemented the DCH and CH forms have little apparent dependence on the sparsity of the underlying data-structures. At low values of β, each method shows a similar performance for Pauli X and Z measurements, with a slight advantage for the CH and GraphSim methods when simulating X measurements. However, as previously mentioned, Pauli measurements in the CHP method have a scaling that increases with the number of non-zero entries in the tableau. The measurement routine of the GraphSim method, like the entangling gates, also depends on the maximal degree of the underlying graph. Thus, both routines see a significant increase in runtime as β increases. The GraphSim method in particular sees an almost 100 times increase in runtime between the smallest and largest values of β at $n = 60$.

Again, likely as a result of the bitpacked implementation, the DCH and CH methods are mostly unaffected by increasing β, with their runtime growing by a factor of $1.33 - 2$ between the extremal values. This small shift can be attributed to an increase in the number of non-zero entries, and thus the number of operations required in commuting a Pauli through the circuit and applying Proposition 1.

If we were to extend the CH and DCH methods above 64 qubits, we might expect this effect to become slightly more pronounced, as we would also incur the overhead of checking multiple binary variables. This effect can in fact be observed in the CHP data, which employs a version of bitpacking based on 32-bit integers. Above 32 qubits, we see a sharp jump in the runtime, which arises from the need to employ two integers for each bitpacked variable.

In conclusion then, we have developed two novel stabilizer simulators which are performant, and offer improved 'functionality' over previous methods. To further develop these tools, it would be important to extend them beyond the current 64 qubit limit, and to finish the implementation of arbitrary stabilizer inner products. With the addition of these routines, this software would form a very versatile tool-set for simulating different aspects of stabilizer circuits.

Chapter 3

Stabilizer Decompositions of Quantum States

3.1 Introduction

In the previous chapter, we discussed in detail efficient simulations of stabilizer circuits. Recalling the discussion in Section 1.2.3, this classical simulability in turn implies that non-stabilizer states are a resource for quantum computation. In this section, we will introduce a particular model of quantum computation that makes explicit the computational role of 'magic' states, Pauli Based Computation [51, 123].

This model forms the basis for the definition of 'Stabilizer Rank', a quantity which tries to relate the computational power of non-stabilizer states to the task of classical simulation. This chapter will be focused on extending stabilizer rank decompositions, whereas the following chapter will focus on implementing classical simulations based on these decompositions.

3.1.1 Pauli Based Computations

A Pauli Based Computation (PBC) is a measurement-based model of quantum computing, whereby a computation is realised by applying a sequence of Pauli measurements to a set of non-stabilizer magic states, and post-processing of the measurement outcomes. In general, this sequence will be 'adaptive': the choice of measurement operator will depend on the outcome of previous measurements.

It is well known that quantum circuits built out of Clifford gates and the T gate are universal for quantum computation [68]. Thus, any arbitrary computation

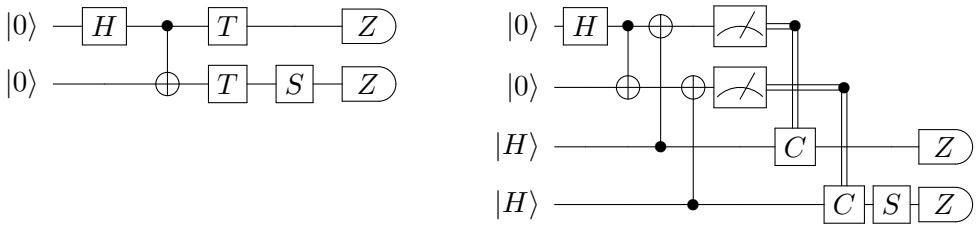


Figure 3.1: Figure illustrating two equivalent forms of a small circuit built form the Clifford + T gate set. The lower circuit is obtained from the former by replacing each T gate with a teleportation or 'state-injection' gadget that consumes one magic state$|H\rangle = \cos\frac{\pi}{8}|0\rangle + \sin\frac{\pi}{8}|1\rangle$. This performs a T gate (up to a measurement controlled correction operation C which is a Clifford gate) [67].

U acting on a computational input state can be expressed as a circuit with m Clifford operations and t T gates.

By replacing each T gate in a Clifford+T circuit with a state-injection gadget [68], we instead end up with a circuit built exclusively from Clifford gates and Pauli measurements, acting on n qubits in a computational basis state, and t qubits in a non-stabilizer state. An example of a gadgetized circuit is given in Figure 3.1. Once in this form, we can convert the circuit to a PBC [123, 51].

In the following discussion, we assume that the only intermediate measurements in the circuit arise from the state-injection gadgets. Circuits with clasically controlled gates conditioned on intermediate measurements are also called 'adaptive'. We note that the PBC construction works for both adaptive and non-adaptive circuits, so this assumption can be made without loss of generality, but helps to simplify the discusion [51, 123].

Once in this form, we can commute every Clifford operator through the circuit and past the final Pauli measurement layer. As we do, we update each measurement operator $P \to P'$ under conjugation, and the Clifford gate can then be discarded as it occurs after the measurement layer and thus has no effect on the outcome. These updates can be efficiently computed using the methods discussed in Chapter 2. The result is some new sequence of Pauli measurements $P'_1, \ldots, P'_r$, acting on $n+t$ qubits.

It is then possible to show that these measurements can be rearranged such that

all measurements commute, and act non-trivially on only the t magic states. The key technique is a lemma showing that if any pair P_j, P_k anticommute, they can be updated by sampling a measurement outcome $\lambda_k = \pm 1$ uniformly at random, and replacing the P_j with a Clifford operator $V_{j,k} = \frac{\lambda_j P_j + \lambda_k P_k}{\sqrt{2}}$, where λ_j was the outcome of measuring P_j. This Clifford can then be commuted through the rest of the measurement layer [51].

Now consider prepending the circuit with Pauli Z measurements on the n computational qubits. By definition, these measurements are deterministic and do not change alter the computation. Application of the above Lemma ensures that these computational measurements all commute with the final measurement operators P_i, and thus that the P_i act trivially on the n computational qubits [123].

Overall then, the PBC model allows us to realise quantum computation using only a supply of non-stabilizer resource states, Pauli measurements, and probabilistic classical computation, used to compute and update the Pauli measurement sequence [51]. The classical component of the computation is efficient, as the updated measurement sequence can be computed with a runtime that scales polynomially in the number of qubits.

A PBC obtained from some Clifford + T circuit U, can be said to efficiently simulate the original circuit, in both the weak [51] and strong sense [123]. Weak simulation follows immediately as, given a method to sample from the measurement operators of the PBC, this also corresponds to a sample of the output distribution of the original circuit [51]. Strong simulation then follows from the result that an adaptive circuit with postselection has a corresponding PBC with postselected Pauli measurements [51]. In particular, we can fix both the measurement outcomes of the circuit, and the measurement-controlled correction operations introduced by state-injection. The result is a non-adaptive circuit, which is translated to a non-adaptive PBC with a fixed Pauli projector $\Pi_{\vec{\mathbf{x}},\vec{\mathbf{s}}}$ [51], where $\vec{\mathbf{x}}$ and $\vec{\mathbf{s}}$ are the postselected binary bits corresponding to the measurement outcome and the state-injection gadgets, respectively [123, 49].

The corresponding probability amplitude is thus given by

$$\left\langle \vec{\mathbf{x}} \middle| U \middle| 0^{\otimes n} \right\rangle \equiv 2^t \left\langle T^{\otimes t} \middle| \Pi_{\vec{\mathbf{x}},\vec{\mathbf{s}}} \middle| T^{\otimes t} \right\rangle \tag{3.1}$$

where we reweight the probability to account for the fact that each of the 2^t different outcomes on the state-injection gadgets is equiprobable.

3.1.2 Stabilizer State Decompositions

In the PBC model of quantum computation, the role of non-stabilizer states as a resource for quantum computation is made explicit. It is also clear that the PBC would require exponential time to simulate clasically, as Pauli expectation values on non-stabilizer states cannot in general be efficiently computed [65].

In the context of resource theories for quantum computation, we can consider studying quantum computations by decomposing computations in terms of the 'free' set of operations. This is what Bravyi, Smith & Smolin did when considering stabilizer state decompositions of magic states. We define a stabilizer state decomposition of a general state $|\psi\rangle$ as

$$|\psi\rangle = \sum_{i=1}^{\chi} c_i |\phi_i\rangle , \tag{3.2}$$

where each $|\phi_i\rangle$ is a stabilizer state and the total number of terms in the decomposition, χ, is called the *Stabilizer Rank* of the state $|\psi\rangle$.

Given a PBC, and a stabilizer state decomposition of the magic states $|T\rangle^{\otimes t}$, then strong simulation of a PBC reduces to computing a Pauli expectation value for each term in the decomposition. As these are stabilizer states, this expectation value can be computed efficiently. Using that fact that Pauli projectors map stabilizer states to stabilizer states, we can write

$$\Pi \left| H^{\otimes t} \right\rangle = \sum_{i=1}^{\chi} c_i \Pi |\phi_i\rangle = \sum_{i=1}^{\chi} c_i \left| \phi_i' \right\rangle = \left| \psi' \right\rangle$$

and thus, the overall expectation value is given by

$$\left\langle H^{\otimes t}\middle|\Pi\middle|H^{\otimes t}\right\rangle = \left\langle H^{\otimes t}\middle|\psi'\right\rangle = \sum_{i,j} c_i^* c_j \left\langle \phi_i\middle|\phi_j'\right\rangle, \tag{3.3}$$

a sum of χ^2 stabilizer inner products. Thus, the overall runtime of the simulation scales as $O\left(\chi^2 \text{poly}(n)\right)$ [123].

An explicit method for weak sampling using stabilizer state decompositions was also outlined in [123], based on computing individual measurement probabilities and using them to sample marginals. In particular, consider sampling the jth bit of an output string x, given outcomes for bits $x_1, x_2, \ldots x_{j-1}$. We can sample x_j but computing two probability terms, as [49]

$$P(x_j|x_1, x_2, \cdots x_{j-1}) = \frac{P(x_1, \ldots x_j)}{P(x_1, \ldots x_{j-1})} \equiv \frac{\left\langle H^{\otimes t}\middle|\Pi_{x_1,\ldots x_j}\middle|H^{\otimes t}\right\rangle}{\langle H^{\otimes t}|\Pi_{x_1,\cdots x_{j-1}}|H^{\otimes t}\rangle}. \tag{3.4}$$

Fixing $x_j = 0$, and computing the conditional probability, we can thus sample the jth bit by generating uniform random numbers. If $r \leq P(0|x_1, x_2, \cdots x_{j-1})$, we return 0, else we return 1.

Importantly, the authors were able to show that stabilizer rank decompositions of magic states can be smaller than expected. As a simple example, consider two copies of the $|T\rangle$ magic state:

$$\begin{aligned} |H\rangle &= \cos\tfrac{\pi}{8}|0\rangle + \sin\tfrac{\pi}{8}|1\rangle & \chi(|H\rangle) &= 2 \\ \left|H^{\otimes 2}\right\rangle &= \tfrac{1}{2}(|00\rangle + \mathrm{i}|11\rangle) + \tfrac{1}{2\sqrt{2}}(|01\rangle + |10\rangle) & \chi\left(\left|H^{\otimes 2}\right\rangle\right) &= 2 \end{aligned} \tag{3.5}$$

This is a quadratic reduction in the number of terms in the decomposition, compared to an expansion in the computational basis. The authors in fact improved this asymptotic bound by using random walk methods to search for other stabilizer state decompositions. They were able to set an upper bound $\left|H^{\otimes 6}\right\rangle \leq 7$, and thus

$$\chi\left(\left|H^{\otimes t}\right\rangle\right) \leq 7^{t/6} = 2^{\frac{\log_2(7)}{6}t} \approx 2^{0.47t} \tag{3.6}$$

giving strong simulation with stabilizer state decompositions a smaller exponential overhead than state-vector methods, even with the dependence on χ^2 in the runtime.

Previous works have also explored stabilizer decompositions of universal quantum computations, containing non-Clifford gates. In their original paper, Aaronson & Gottesman explored expanding gates in the Pauli operator basis. Each branch in the expansion will produce a different stabilizer state [66].

$$U|\phi\rangle = \sum_i a_i P_i |\phi\rangle = \sum_i a_i \left|\phi_i'\right\rangle$$

In general, this will require up to 4^m stabilizer states for each m-qubit non-Clifford gate. For the T gate in particular, we can write

$$T \equiv \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\sqrt{2}}(1+i) \end{pmatrix} = \frac{\sqrt{2}+\mathrm{i}}{\sqrt{2}} I - \frac{\mathrm{i}}{\sqrt{2}} Z$$

and thus this extension of the CHP method requires 2^t stabilizer states for t T gates.

A different method was also proposed by Garcia et al., which they call stabilizer frames [107]. These are stabilizer state decompositions built out of so-called 'co-factors', made from post-selecting the results of single qubit computational basis measurements. For example, the action of a controlled-S gate can be expanded into two stabilizer state terms, by post-selecting on the control bit being $|0\rangle$ or $|1\rangle$. For the T gate, stabilizer frames similarly require a number of terms that that scales as 2^t.

Norm Estimation and Approximate Decompositions

The stabilizer rank method as introduced in [123] already compares favourably to similar methods of simulating quantum circuits through stabilizer state decompositions. However, the method was further refined in a successive paper, which extended its results to the case of approximate simulation [49].

The first development of [49] was an algorithm for estimating the norm of

states, that can be used to optimize the computation of Pauli expectation values on stabilizer state decompositions. A detailed discussion of this norm estimation routine will be given in Chapter 4. Importantly however, this method allows Pauli expectation values to be approximated to within ϵ relative error, with a runtime that scales as $O(\chi t^3 \epsilon^{-2})$, a quadratic reduction in terms of the stabilizer rank [49].

The second component was a method for construction approximate stabilizer state decompositions

$$\left|\tilde{\psi}\right\rangle = \sum_{i=1}^{\chi_\epsilon} c_i \left|\phi_i\right\rangle \,:\, F\left(\left|\tilde{\psi}\right\rangle, \left|\psi\right\rangle\right) \geq 1 - \epsilon \tag{3.7}$$

where F is the fidelity and χ_ϵ is called the approximate stabilizer rank. Using a method which we will discuss in detail in Section 3.2.2, the authors showed that

$$\chi_\epsilon\left(\left|H\right\rangle^{\otimes t}\right) \approx 2^{0.23t}\epsilon^{-2}. \tag{3.8}$$

Thus giving a further quadratic reduction in the number of terms in the stabilizer state decomposition.

3.2 Results

As established, the stabilizer rank method offers reasonably efficient decompositions of universal quantum circuits. Importantly however, these results only apply to the T magic state. While Clifford+T is known to be a universal gate set for quantum computation, in practice the number of T gates required to synthesize a circuit grows rapidly. For example, synthesising arbitrary-angle Pauli Z rotations from Clifford+T gates can quickly result in a T count on the order of 100 per gate [124, 125]. In the rest of this chapter, we seek to extend our understanding of stabilizer state decompositions beyond the $|H\rangle$ magic state, and discuss the interpretation of the stabilizer rank as it relates to quantum computation.

3.2.1 Exact Stabilizer Rank

As well as having an interpretation in terms of classical simulations, the stabilizer rank of a state has three properties that make it interesting as a potential measure of 'magic' as a resource in quantum computation [81, 75].

Claim 2 *Properties of the exact stabilizer rank:*

1. ***Faithfulness:*** $\chi(|\psi\rangle) = 1$ *iff* $|\psi\rangle$ *is a stabilizer state.*
2. ***Submultiplicativity:*** $\chi(|\psi\rangle \otimes |\Psi\rangle) \leq \chi(|\psi\rangle)\chi(|\Psi\rangle)$.
3. ***Monotonicity:*** χ *is invariant under Clifford gates and monotonically decreasing under Pauli measurements.*

Proof of Claim 2. The faithfulness property of χ follows from its definition (see Eq. 3.2).
Given a tensor product of two states, we can expand out their stabilizer state decompositions as

$$|\psi\rangle \otimes |\Psi\rangle = \sum_{i=1}^{\chi(|\psi\rangle)} \sum_{j=1}^{\chi(|\Psi\rangle)} c_i c_j |\phi_i\rangle \otimes |\phi_j\rangle .$$

A tensor product of two stabilizer states is also a stabilizer state, and thus we obtain a potential stabilizer state decomposition with $\chi(|\psi\rangle) \cdot \chi(|\Psi\rangle)$ terms. However smaller decompositions, including entangled stabilizer states rather than these separable states, may exist. Thus, the stabilizer rank is submultiplicative under tensor product.
Invariance under Clifford unitaries follows from the linearity of quantum mechanics, and the definition of the Clifford group. Expanding out the decomposition, we have

$$V |\psi\rangle = \sum_i c_i V |\phi_i\rangle = \sum_i c_i \left|\phi_i'\right\rangle$$

where the new states in the decomposition can be efficiently computed [65].
After performing a Pauli measurement, the decomposition will be updated by

applying a projector $\frac{1}{2}(\mathbb{I}+\lambda P)$, where $\lambda = \pm 1$ is the outcome of the measurement.

$$\frac{1}{2}(I+\lambda P)|\psi\rangle = \left|\psi'\right\rangle = \sum_i c_i (|\phi\rangle + \lambda P|\phi\rangle)$$

As discussed in the previous chapter, applying a Pauli projector to a stabilizer state either produces a new stabilizer state, or the null-vector if $\lambda P|\phi\rangle = -|\phi\rangle$. If no states are orthogonal to the Pauli projector applied, then the stabilizer rank is unchanged and the decomposition is updated. Otherwise, $\chi(|\psi'\rangle) < \chi(|\psi\rangle)$. □

No general method is known for finding low rank stabilizer state decompositions of general quantum states. The number of stabilizer states grows exponentially with the number of qubits [66], even before considering the combinatoric growth in the number of candidate decompositions. Additionally, checking the validity of a candidate stabilizer state decomposition has a computational complexity that also scales exponentially in the number of qubits.

In [123], the authors made use of computational searches to find the upper bounds on the stabilizer rank of the $|H\rangle$ magic state shown in Table 3.1. They also make the following conjecture, called Conjecture 1 in the paper.

Conjecture 1 *Let* $\chi_n = \chi\left(\left|H^{\otimes n}\right\rangle\right)$. *Then for a single qubit state* $|\phi\rangle$

$$\begin{aligned}
\chi\left(\left|\phi^{\otimes n}\right\rangle\right) &= 1 \quad && \textit{If } |\phi\rangle \textit{ is a stabilizer state} \\
\chi\left(\left|\phi^{\otimes n}\right\rangle\right) &= \chi_n && \textit{If } |\phi\rangle \textit{ is a magic state} \\
\chi\left(\left|\phi^{\otimes n}\right\rangle\right) &> \chi_n && \textit{Otherwise.}
\end{aligned}$$

The $|H\rangle$ state is one of a family of 12 single qubit magic states, which can be transformed into each other by applying Clifford gates. Thus, they also have

n copies	1	2	3	4	5	6
χ_n	2	2	3	4	6	7

Table 3.1: Optimal rank of stabilizer state decompositions for the $|H\rangle$ magic state, from [123].

equivalent stabilizer rank. We refer to these magic states as 'edge states', from their location on the Bloch sphere [68]. However, there also exist a second set of 8 single qubit magic states that cannot be generated from the edge states by Clifford unitaries. In this text, we call these 'face states'.

$$|F\rangle = \cos\beta\,|0\rangle + \mathrm{e}^{\mathrm{i}\pi/4}\sin\beta\,|1\rangle \;:\; 2\beta = \cos^{-1}\frac{1}{\sqrt{3}} \tag{3.9}$$

Denoted $|R\rangle$ in [123], the authors comment that numeric results appear to show it has the same stabilizer rank as the edge type-states, and thus put forward Conjecture 1.

We further examined the stabilizer rank for different quantum states by extending the computational searches of [123].

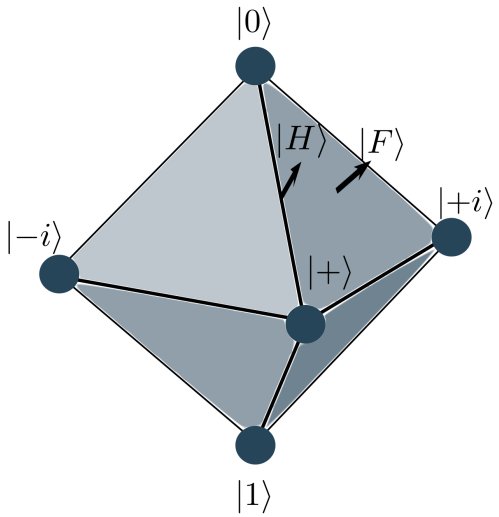


Figure 3.2: Diagram showing the location of single-qubit stabilizer states and magic states on the Bloch sphere. Single qubit Clifford gates act as the symmetry group of an octahedron in the Bloch sphere, who's vertices are the individual stabilizer states. 'Edge' and 'face' magic states are named for their positions relative to this octahedron. Based on diagrams given in from [68].

Computation Searches for Decompositions

We employ a combination of brute force and random walk searches for stabilizer state decompositions to establish the stabilizer rank of different families of quantum states, using a custom program developed in Python.

To test a candidate decomposition of χ stabilizer states $\Phi = \{|\phi_i\rangle\}$, we compute

the projector on to the subspace generated by the states Π_Φ, and the compute the projection of the target state into this subspace $\|\Pi_\Phi |\psi\rangle\|$. If the norm is equal to 1, then the state lies within this subspace and we terminate the search.

Given a collection of χ stabilizer states, we can build their projector by first constructing a $2^n \times \chi$ matrix A where each column is one of the χ stabilizer states. We then apply the QR decomposition, a standard linear algebra technique, to compute χ orthogonal basis-vectors for the subspace spanned by these stabilizer states. Given the column matrix Q built from these orthogonal basis-vectors, we then have $\Pi_\Phi \equiv Q Q^\dagger$. This was implemented using the `Numpy` library, with additional optimization provided by using the `Numba` Just-In-Time compiler [126, 127].

The random walk method follows the description in Appendix B of [123]. The search algorithm takes as input a state $|\psi\rangle$ to decompose, and a candidate stabilizer rank χ. We begin with a candidate decomposition $\Phi = \{|\phi_i\rangle\}$, and compute the 'distance' from the generated subspace $F = 1 - \|\Pi_\Phi |\psi\rangle\|$. We then update one state, chosen uniformly at random, by applying a random Pauli operator P. We then compute the updated distance $F' = 1 - \left\|\Pi_{\tilde{\Phi}} |\psi\rangle\right\|$ using this updated set of stabilizer states. If $F' < F$, we accept the move and proceed. Otherwise, we accept the move with a probability given by the Boltzmann distribution $p = \mathrm{e}^{-\beta\left(F'-F\right)}$, where β is a parameter we set. As the search continues, we gradually increase β. This method is thus similar to the simulated annealing approach in optimization, where we are seeking to minimize the distance between $|\psi\rangle$ and the generated subspace.

Our random walks were run using the same parameters of [123], testing 100 values of the inverse-temperature parameter $\beta \in [100, 4000]$, and running for 1000 steps for each value of β. For any given candidate and value of χ, we repeated the random walk 5 times. The smallest decomposition found across all runs was taken as an upper bound on the stabilizer rank. $\chi = 2$ was taken as a lower bound, and the largest value tested was either $2^n - 1$, or else a value

derived using submultiplicativity and results for fewer copies of the target state.

The brute force method, in contrast, takes as input a target state $|\psi\rangle$, and an upper and lower bound of stabilizer rank to test. The typical lower bound given is 2. The upper-bound is set by either the computational basis expansion, which is a valid stabilizer state decomposition, or else a bound based on submultiplicativity and known results for fewer copies of a state.

Pseudocode descriptions of the search methods are given in Algorithms 1 and 2. We also note an additional optimisation that, in the case where the target state has only real coefficients, we can restrict ourselves to considering decompositions built only from stabilizer states with real values. In the random walk case, we additionally restrict the moves we generate such that they do not introduce any imaginary coefficients. We do this by requiring that the random Pauli operators have an even number of Pauli Y operators.

As mentioned above, the number of stabilizer states grows exponentially with the number of qubits. In particular, we have [66]

$$N_\phi = 2^n \prod_{k=1}^{n-1} \left(2^{n-k} + 1\right). \tag{3.10}$$

In practice, brute force searches were tractable up to about 3 qubits. Some examples of the growth in possible combinations are given in Table 3.2.

n qubits	1	2	3	4
N_ϕ	6	60	1080	36720
$\binom{N_\phi}{2^n-1}$	6	33240	3.33×10^{17}	2.27×10^{56}

Table 3.2: Table showing how the number of combinations of stabilizer states grows as a function for the number of qubits. We consider $2^n - 1$ as this is the largest possible stabilizer rank that is below the trivial computational basis bound.

Generating Stabilizer States

As an input for Algorithms 1 and 2, we need a way to quickly generate random stabilizer states, as well as a library of all stabilizer states for small n. To

Algorithm 1 Random Walk Search for Stabilizer State Decomposition

Require: $\beta_{init}, \beta_{max}, M$, target integer χ
Require: PROJECTOR(Φ) ▷ Returns projector onto subspace spanned by Φ
$\Phi \leftarrow (\phi_1, \cdots \phi_\chi)$ where each ϕ_a is chosen at random.
$\beta \leftarrow \beta_{init}$
while $\beta < \beta_{max}$ **do**
  for $i = 0$ to 1000 **do**
    $\Pi_\Phi \leftarrow$PROJECTOR(Φ)
    $F \leftarrow 1 - \|\Pi_\Phi |\psi\rangle\|$
    if $F = 1$ **then**
      return Φ
    end if
    Pick random integer $a \in [1, n]$ and random Pauli $P \in \mathcal{P}_n$
    $|\phi_a\rangle' \leftarrow c(\mathbb{I} + P)|\phi_a\rangle$ ▷ If $|\phi_a\rangle' = 0$, pick new a and P.
    $\tilde{\Phi} \leftarrow (|\phi_1\rangle, \cdots, |\phi_a\rangle', \cdots |\phi_\chi\rangle)$
    $\Pi_{\tilde{\Phi}} \leftarrow$PROJECTOR($\tilde{\Phi}$)
    $F' \leftarrow 1 - \|\Pi_{\tilde{\Phi}} |\psi\rangle\|$
    if $F' < F$ **then**
      $|\phi_a\rangle \leftarrow |\phi_a\rangle'$
    else
      $p_{accept} \leftarrow \exp[-\beta(F' - F)]$
      Pick random $r \in [0, 1]$
      if $r < p_{accept}$ **then**
        $|\phi_a\rangle \leftarrow |\phi_a\rangle'$
      end if
    end if
  end for
  $\beta \leftarrow \beta + \left(\frac{\beta_{\max} - \beta_{\text{init}}}{M}\right)$
end while
return No decomposition found.

Algorithm 2 Brute Force Search for stabilizer rank

Require: $\{\phi\}_n$ ▷ The set of n qubit stabilizer states
Require: $\chi_{\max}$ ▷ Upper bound on χ
Require: PROJECTOR(Φ) ▷ Returns projector onto subspace spanned by Φ.
$\chi = 2$
while $\chi < \chi_{\max}$ **do**
  for $\Phi = \{|\phi_1\rangle\}, \ldots, |\phi_\chi\rangle$ ▷ For all combinations of i states. **do**
    $\Pi_\Phi \leftarrow$ PROJECTOR(Φ)
    if $\|\Pi_{\tilde{\Phi}} |\psi\rangle\| = 1$ **then**
      return χ, Φ
    end if
  end for
  $\chi \leftarrow \chi + 1$
end while
return $\chi_{\max}$ ▷ The previous expansion is still is the best found.

accomplish this, we make use of the canonical form for stabilizer tableaux introduced by Garcia et al., and discussed in Section 2.1 [105].

Like the CHP method, a canonical stabilizer tableau is a $n \times (2n+1)$ matrix, where each row encodes a Pauli operator

$$P(\vec{\mathbf{s}}) = -1^{s_0} \otimes_{i=1}^{n} X^{s_i} Z^{s_{i+n}}, : \vec{\mathbf{s}} \in \mathbb{Z}_2^{2n+1}.$$

There are in general multiple tableau corresponding to a given stabilizer state, but using Algorithm 1 of [105] any tableau can be converted to a standard form.

To quickly generate random stabilizer states then, we generate a random $n \times (2n+1)$ binary matrix. We then apply the canonical form algorithm. If any rows of the tableau are the all-0 string, then the tableau does not correspond to a stabilizer state and so we discard it. Else, we build up a Pauli projector from the rows of the tableau, and compute the stabilizer state as the unique $+1$ eigenstate.

To generate a complete library of stabilizer states, first recall that Pauli operators in a stabilizer group have only phase of ± 1. For a stabilizer group with n generators, there are thus 2^n possible combinations of phase for each generator, each of which correspond to a given stabilizer state. We can thus focus on generating just the $N_\phi/2^n$ stabilizer groups with all positive phase.

We begin by generating all $2^{2n}-1$ possible binary strings, which correspond to all possible choices of Pauli operator. We ignore the all 0 string, as this corresponds to the identity operator which cannot generate a stabilizer group. Then, for all $\binom{2^{2n}-1}{n}$ possible combinations of n strings, we build the stabilizer tableau and convert it to canonical form. If it is not full rank, or corresponds to a tableau already found, we discard it. Otherwise we store the tableau. We terminate after generating $N_\phi/2^n$ groups. For each group then, we test all 2^n possible phase combinations, and then compute the stabilizer state as described above. This process was computationally intensive, but overall we

were able to generate a library of stabilizer states on up to 4 qubits.

Both the random stabilizer state generation and the deterministic stabilizer state generation were implemented in Python. Stabilizer tableau were stored as bitpacked `Numpy` arrays. Computing the corresponding Pauli projector and stabilizer state made use of `Numpy` linear algebra routines, including the optimised eigensovler for hermitian matrices, with additional optimization using `Numba` [126, 127].

Results of Computational Searches

We extend the computational searches for copies of single-qubit magic states up to $n = 10$, and give explicit results for the face-type magic states. We used brute force searches for $n \leq 3$ qubits. Otherwise, we made use of random walk searches.

For all values of n tested, the edge and face type magic states had the same observed stabilizer rank. Despite extending the range of the numeric search, however, above $n = 6$ copies, we found no decomposition smaller than the submultiplicative bound. Thus, the asymptotic scaling shown in [123] remains the best result known for single-qubit magic states..

As a means of probing Conjecture 1, we also explored the stabilizer rank of 'typical' single qubit states, generated uniformly at random. The target states were prepared by applying a Haar random single-qubit unitary to the $|0\rangle$ states [128]. We began by applying brute force searches to 1000 typical states up to 3 copies, and observed that all states tested had the same stabilizer rank, and also that their stabilizer rank grew more slowly than the computational basis expansion. All results for single qubit states are shown in Table 3.3.

Applying the argument of Eq. 3.6, then for typical single qubit states their stabilizer rank is upper bounded by

$$\chi\left(\left|\phi^{\otimes t}\right\rangle\right) \leq 8^{t/6} = 2^{\log_2 8t/6} = 2^{0.5t}. \tag{3.11}$$

To further explore the claim in Conjecture 1, we also performed computational

searches for specific states with a structure related to the magic states. In particular, we performed computational searches for the $|CS\rangle$ and $|CCZ\rangle$ magic states, which can be used to inject the two-qubit CS gate and the three-qubit CCZ gate, respectively. Both of these gates, like the T gate, belong to the third level of the Clifford hierarchy. We also considered the single qubit resource states $T^{\frac{1}{2}}|+\rangle$ and $T^{\frac{1}{4}}|+\rangle$. These resource states can be used to inject gates from the 4th and 5th levels of the Clifford hierarchy, though potentially requiring a non-Clifford correction operation. We limited our searches up to 6 qubits, which meant considering up to 3 copies of the $|CS\rangle$ state and just two copies of the CCZ state. Results are shown in table 3.4.

Interestingly, we observe that the single qubit resource states corresponding to gates in higher levels of the Clifford hierarchy show no difference from the stabilizer rank of typical single qubit states. However, magic states on 2 and 3 qubits also show significantly reduced stabilizer rank. In fact, the asymptotic scaling of the $|CS\rangle$ and $|CCZ\rangle$ is significantly smaller when compared to the naive computational basis expansion, scaling as $\approx 2^{0.79t}$ and 2^t versus 2^{2t} and 2^{3t}, respectively.

t Copies	2	3	4	5	6	7	8	9	10
$\chi\left(\left\|T^{\otimes t}\right\rangle\right)$	2	3	4	6	7	12	14	21	28
$\chi\left(\left\|F^{\otimes t}\right\rangle\right)$	2	3	4	6	7	12	14	21	28
χ(Typical)	3	4	5	6	8	14	24	30	36

Table 3.3: Results of computational searches for stabilizer rank decompositions of different single-qubit quantum states. The results would appear to agree with Conjecture 1, that stabilizer rank is smaller for magic states.

t Copies	1	2	3	4
$T^{\frac{1}{2}}\|+\rangle$	2	3	4	5
$T^{\frac{1}{4}}\|+\rangle$	2	3	4	5
$\|CS\rangle$	2	3	6	-
$\|CCZ\rangle$	2	4	-	-

Table 3.4: Results of computational searches for stabilizer rank decompositions of different types of non-stabilizer resource state. We extended the searches for the $T^{\frac{1}{2}}$ and $T^{\frac{1}{4}}$ gate resource states up to 6 copies, but found no decompositions smaller than the results for typical single qubit states.

Decompositions of the Symmetric Subspace

When taking multiple copies of any given n-qubit state $|\psi\rangle$, the result will always lie within the symmetric subspace $\mathrm{Sym}_{n,t} \subseteq \mathbb{C}^{2^n}$. This is a subspace of the full n-qubit Hilbert space with dimension

$$\dim\left(\mathrm{Sym}_{n,t}\right) = \binom{2^n + t - 1}{t} \tag{3.12}$$

We can thus consider searching for stabilizer state decompositions of a subspace. We define the exact stabilizer rank of a subspace P as

$$\chi(P) = |\Phi| : P \in \mathrm{span}[\Phi]. \tag{3.13}$$

Computationally, we employ the Random Walk method, to build decompositions of the subspace $\mathrm{Sym}_{1,t}$. As our objective function, we replace the projection onto the subspace Π_Φ with the largest principle angle between the subspaces Π_Φ and $\Pi_{\mathrm{Sym}_{1,t}}$. If $\mathrm{Sym}_{1,t} \subseteq \mathrm{Span}(\Phi)$, this angle is zero. The formula for the largest principle angle is shown in Eq. 3.14 [129]. The projector onto the symmetric subspace, $\Pi_{\mathrm{Sym}_{1,t}}$, was computed using the method based on superpositions of computational basis states with equal Hamming weight, outlined in [130].

$$\theta\left(\Pi_\Phi, \Pi_{\mathrm{Sym}_{1,t}}\right) = \sin^{-1}\left(\left\|(I - \Pi_\Phi)\,\Pi_{\mathrm{Syn}_{1,t}}\right\|\right) \tag{3.14}$$

For all values tested, the best decomposition found for the projector onto the single qubit symmetric subspace were equal to the results for typical single qubit states. Additionally, we note that for $t \leq 5$

$$\chi\left(\mathrm{Sym}_{1,t}\right) = \dim\left(\mathrm{Sym}_{1,t}\right) \leq t + 1 = \binom{2 + t - 1}{t}, \tag{3.15}$$

and the smallest stabilizer rank found for the single qubit symmetric subspace is equal to its dimension.

In fact, in [109], we make the following claim

Claim 3 $\chi\left(Sym_{n,t}\right) = dim\left(Sym_{n,t}\right) : \forall n,\, t \leq 5$

For $t \leq 3$, this claim follows from the property that stabilizer states form a projective 3 design [131]. Thus, for a given n qubits and $t \leq 3$

$$\frac{1}{N_\phi}\sum_i |\phi_i\rangle\langle\phi_i| = \frac{\Pi_{\mathrm{Sym}_{n,t}}}{\dim\left(\mathrm{Sym}_{n,t}\right)}, \tag{3.16}$$

a superposition of t copies of all n-qubit stabilizer states is proportional to the projector onto the symmetric subspace.

From this, we can conclude that $\mathrm{Span}\left(\left\{\left|\phi_i^{\otimes t}\right\rangle\right\}\right) \subseteq \Pi_{\mathrm{Sym}_{n,t}}$. We can thus find a minimal spanning set of vectors $\{\left|\phi_j^{\otimes t}\right\rangle\}$ such that $\mathrm{Span}\left(\{\left|\phi_j^{\otimes t}\right\rangle\}\right) = \mathrm{Sym}_{n,t}$, and $\left|\{\left|\phi_j^{\otimes t}\right\rangle\}\right| = \dim\left(\mathrm{Sym}_{n,t}\right)$, completing the claim for $t \leq 3$. In [109], we present a proof by Earl Campbell that also extends this result up to $t = 5$ using the fact that stabilizer states are 'almost' a projective 4-design [131].

Clifford Symmetries

The results of computational searches, and the proof for the decomposition of the symmetric subspace, are consistent with Conjecture 1. In Table 3.5, we compare the bounds for the symmetric subspace with the stabilizer rank decompositions found for different magic states, and show that in general the magic states exhibit a smaller stabilizer rank.

Table 3.5: Tables comparing the dimension, and thus stabilizer rank, of the symmetric subspace up to 5 copies with that of magic states, for 1, 2 and 3 qubits.

(a)

n Copies	2	3	4	5
$\dim\left(\mathrm{Sym}_{n,t}\right)$	3	4	5	6
$\chi(\|T,F\rangle)$	2	3	4	6

n Copies	1	2	3
$\dim\left(\mathrm{Sym}_{n,t}\right)$	4	10	20
$\chi(\|CS\rangle)$	2	3	6

(b)

n Copies	1	2
$\dim\left(\mathrm{Sym}_{n,t}\right)$	8	36
$\|CCZ\rangle$	2	4

(c)

A property common to all the magic states tested is that they each have an

associated Clifford symmetry. This is in fact always true for a magic state that can be used to inject a gate from $\mathcal{C}_3$. These magic states have the form $|U\rangle = U|\phi\rangle$, where $|\phi\rangle$ is a stabilizer state [67]. Updating the stabilizer group under conjugation, we obtain a new set of operators that stabilizer the resource state $|U\rangle$

$$S|\phi\rangle = |\phi\rangle \rightarrow USU^\dagger|U\rangle = USU^\dagger U|\phi\rangle = U|\phi\rangle \ \forall S \in \mathcal{S}_\phi. \tag{3.17}$$

From the definition of $\mathcal{C}_3$, these operators are then Clifford as $USU^\dagger \in \mathcal{C}_2$, and also form a group which we call $\mathcal{M}$. We introduce the following nomenclature.

Definition 3.1 (Clifford Magic State)**.** Consider a magic state $|R\rangle$, with an associated group of Clifford symmetries $\mathcal{M}$ such that

1. $\mathcal{M} \subseteq \mathcal{C}_2$
2. $m|R\rangle = |R\rangle \ \forall m \in \mathcal{M}$
3. $|R\rangle\langle R| = \frac{1}{|\mathcal{M}|}\sum_{m\in\mathcal{M}} m$

Then $|R\rangle$ is a Clifford magic state.[1]

Any state that can be consumed in a state-injection gadget is also a Clifford magic state.. For example, the $|H\rangle$ magic state is so labeled as it has the property that $H|H\rangle = |H\rangle$, and thus has the group $\{I, H\}$ as its Clifford symmetry. We note however that the face-type magic states are also Clifford magic states. The state $|F\rangle$, for example, is fixed by a group generated by the Clifford gate F. The F gate corresponds to a rotational symmetry of the faces of the stabilizer octahedron, as can be seen by its action on the single qubit stabilizer states.

$$F|0\rangle = |+\rangle \quad F|+i\rangle = |+\rangle \quad F|+\rangle = 0 \tag{3.18}$$

[1]Note that this differs from the definition in [109]. We introduce this definition in this thesis as we consider slightly broader classes of magic state which nonetheless share the property of Clifford symmetries.

It was shown by Earl Campbell that quotient groups of Clifford symmetries can be used to find the stabilizer rank of Clifford magic states $|R\rangle$.

Lemma 3 *Consider a stabilizer state $|\phi_0\rangle : \langle\phi_0|\psi\rangle \neq 0$. We will denote by $\mathcal{M}$ the group of Clifford symmetries of $|\phi\rangle$. Let $\mathcal{N} \subseteq \mathcal{M}$ be the subgroup of $\mathcal{M}$ such that $n|\phi_0\rangle = |\phi_0\rangle \forall n \in \mathcal{N}$, and define $\mathcal{Q}$ as the quotient group $\mathcal{M}/\mathcal{N}$. Then*

$$\chi(|\phi\rangle) \leq \frac{|\mathcal{M}|}{|\mathcal{N}|} \tag{3.19}$$

with stabilizer state decomposition

$$|\psi\rangle \propto \sum_{q\in\mathcal{Q}} q|\phi_0\rangle \tag{3.20}$$

Proof of Lemma 3. We can expand out $|\phi_0\rangle$ as

$$|\phi_0\rangle = \frac{1}{|\mathcal{N}|}\sum_{n\in\mathcal{N}} n|\phi_0\rangle\,.$$

Making this substitution for $|\phi_0\rangle$, we thus have

$$\begin{aligned}\sum_{q\in\mathcal{Q}} q|\phi_0\rangle &= \sum_{q\in\mathcal{Q}}\sum_{n\in\mathcal{N}} qn|\phi_0\rangle \\ &= \sum_{m\in\mathcal{M}} m|\phi_0\rangle\end{aligned}$$

where on the last line we use the definition of the quotient group. From the definition of $\mathcal{M}$, we can write

$$\sum_{m\in\mathcal{M}} = |\psi\rangle\langle\psi|$$

and thus

$$\begin{aligned}\sum_q q|\phi_0\rangle &= \frac{|\mathcal{M}|}{|\mathcal{N}|}|\psi\rangle\langle\psi|\phi_0\rangle \\ \implies |\psi\rangle &= \frac{|\mathcal{N}|}{|\mathcal{M}|\langle\psi|\phi_0\rangle}\sum_{q\in\mathcal{Q}} q|\phi_0\rangle\end{aligned}$$

completing the proof. □

As an example, consider the state $\left|H^{\otimes 2}\right\rangle$. This state has the Clifford symmetry group $\{I, H\otimes I, I\otimes H, H\otimes H\}$. We can build a 2-element normal subgroup $\{I, H\otimes H\}$, which stabilizes the state $|0\rangle|+\rangle + |1\rangle|-\rangle$. This give a stabilizer rank of $|\mathcal{M}|/|\mathcal{N}| = \frac{4}{2} = 2$, as expected.

One interesting extension of this result is that any resource state used to inject controlled diagonal Clifford gate, such as CCZ or CS, also has a stabilizer rank of 2, which agrees with the results of the computational searches in Table 3.5. The stabilizer state decompositions for these states can in fact be found by considering the resource state itself. Expanding out the action of the control, we have

$$U \equiv |0\rangle\langle 0|\otimes I + |1\rangle\langle 1|\otimes C \implies U\left|+^{\otimes n}\right\rangle \propto |0\rangle|+\rangle^{\otimes n-1} + |1\rangle C\left|+^{\otimes n-1}\right\rangle \quad (3.21)$$

which is a stabilizer state decomposition with $\chi = 2$ as C is a Clifford operator, and thus $C\left|+^{\otimes n-1}\right\rangle$ is a stabilizer state.

However, here we show that this method does not always produce optimal stabilizer state decompositions. For example, consider the $|F\rangle$ state. A single copy has Clifford symmetry group $\{I, F, F^2\}$, which has no non-trivial subgroups. This would suggest $\chi(|F\rangle) \leq 3$, which is larger than just the computational basis bound.

For two copies, $\left|F^{\otimes 2}\right\rangle$ has the 9-element symmetry group

$$\{I, FI, IF, F^2I, IF^2, FF, F^2F, FF^2, F^2F^2\}, \quad (3.22)$$

where we omit the tensor product symbol for brevity. From the Lagrange theorem, we know that the order of any subgroup $\mathcal{N}\subseteq\mathcal{M}$ must divide the order of the group [132]. Thus, the smallest possible quotient group has $|\mathcal{Q}| = 3$. Again, this is larger than the known optimal decomposition $\chi\left(\left|F^{\otimes 2}\right\rangle\right) = 2$.

We can further consider extending this method to include permutation symmetries. For t copies of single qubit states, the permutation symmetries cor-

respond to the symmetric group $S(t)$, and can be generated using swap permutations [132]. In terms of quantum gates, these permutations correspond to the $SWAP$ gate, which is a Clifford operator as it can be realised by a sequence of 3 $CNOT$ gates.

Extending the groups to incorporate these permutation symmetries allows us to generate subgroups with the correct index. For example, for the $\left|H^{\otimes 2}\right\rangle$ state, incorporating permutations gives an order 8 symmetry group, with a subgroup of order 4 and thus index 2. This subgroup $\mathcal{N} = \{I, SWAP, HH, SWAPHH\}$, fixes the same stabilizer state $|0\rangle|+\rangle + |1\rangle|-\rangle$, and thus we again have $\chi = 2$.

Similarly, for the $\left|F^{\otimes 2}\right\rangle$ state, we obtain an order 18 Clifford symmetry group by incorporating permutations, and can construct a subgroup of order 9 and thus index 2. However, this subgroup $\mathcal{N}$ corresponds to the group given in Eq. 3.22, which fixes the state $\left|F^{\otimes 2}\right\rangle$. Thus, there is no stabilizer state $|\phi_0\rangle$: $n|\phi_0\rangle = |\phi_0\rangle \ \forall n \in \mathcal{N}$, and we cannot use this result to build a smaller stabilizer state decomposition.

3.2.2 Approximate Stabilizer Rank

In this section, we show how to construct approximate stabilizer state decompositions for Clifford magic states, and more generally. Both methods start with an exact stabilizer state decomposition, which is not required to be optimal, and show how to construct an approximate decomposition by discarding terms.

Clifford Magic States

A method for constructing approximate stabilizer state decompositions of the $|H\rangle$ magic state was described in [49]. Here, we outline their argument, showing how it can naturally be extended to any Clifford magic state, such as $|F\rangle$ or $|CCZ\rangle$.

The authors begin by considering an exact stabilizer state decomposition of

$\left|H^{\otimes t}\right\rangle$ in terms of the states $|0\rangle$ and $|+\rangle$.

$$\left|H^{\otimes t}\right\rangle = \frac{1}{2\cos\left(\pi/8\right)^t} \sum_{\tilde{x}\in\mathbb{Z}_2^t} |x\rangle \tag{3.23}$$

where $|\tilde{x}\rangle$ is a t-qubit state such that

$$|\tilde{x}\rangle = \otimes_{i=1}^{t} H^{\tilde{x}_i} |0\rangle . \tag{3.24}$$

Each term in the decomposition is a tensor product of stabilizer states, generated by a subgroup of the Clifford group. Recalling Eq. 3.20, we can construct a stabilizer state decomposition for a Clifford magic state $|R\rangle$ from a group $\mathcal{Q} \subseteq \mathcal{C}_2$, and a state $|\phi_0\rangle : \langle\phi_0|R\rangle > 0$. We can write

$$|R\rangle \propto \sum_{q\in\mathcal{Q}} |\phi_q\rangle \ : |\phi_q\rangle = q\,|\phi_0\rangle .$$

To normalize the decomposition, we note that $\langle\phi_q|R\rangle = \langle\phi_0|q^{-1}|R\rangle = \langle\phi_0|R\rangle$. Thus,

$$\begin{aligned} |R\rangle &= \frac{1}{|Q|\,\langle\phi_0|R\rangle} \sum_{q\in\mathcal{Q}} |\phi_q\rangle \\ \implies |R\rangle^{\otimes t} &= \frac{1}{\left(|\mathcal{Q}|\,\langle\phi_0|R\rangle\right)^t} \sum_{\vec{\mathbf{q}}\in\mathbb{Z}_{|\mathcal{Q}|}^t} \left|\phi_{\vec{\mathbf{q}}}\right\rangle \end{aligned} \tag{3.25}$$

where $|\phi\rangle_{\vec{\mathbf{q}}} \equiv \otimes_{i=1}^{t} \left|\phi_{\vec{\mathbf{q}}_i}\right\rangle$ and $\vec{\mathbf{q}}$ is t-element vector where each entry denotes a member of the group. Setting $|\mathcal{Q}| = 2$ and $|\phi_0\rangle = |0\rangle$, gives the same decomposition for $\left|H^{\otimes t}\right\rangle$ given in Eq. 3.23.

We can also define states $|\mathcal{L}\rangle$, built from subspaces $\mathcal{L} \subseteq \mathbb{Z}_{|\mathcal{Q}|}^t$

$$|\mathcal{L}\rangle = \frac{1}{\sqrt{|\mathcal{L}|\,Z\left(\mathcal{L}\right)}} \sum_{\vec{\mathbf{q}}\in\mathcal{L}} \left|\phi_{\vec{\mathbf{q}}}\right\rangle, \tag{3.26}$$

where $|\mathcal{L}|$ is the number of elements in the subspace, and $Z\left(\mathcal{L}\right)$ is a normali-

sation factor, given by

$$\begin{aligned}\langle\mathcal{L}|\mathcal{L}\rangle = 1 &= \frac{1}{|\mathcal{L}|\, Z(\mathcal{L})} \sum_{\vec{\mathbf{p}},\vec{\mathbf{q}}\in\mathcal{L}} \langle\phi_p|\phi_q\rangle \\ &= \frac{1}{|\mathcal{L}|\, Z(\mathcal{L})} \sum_{\vec{\mathbf{p}}\vec{\mathbf{q}}} \left\langle \phi_{\vec{\mathbf{0}}} \middle| \phi_{\vec{\mathbf{p}}^{-1}\vec{\mathbf{q}}} \right\rangle \\ &= \frac{1}{Z(\mathcal{L})} \sum_{\vec{\mathbf{q}}} \left\langle \phi_{\vec{\mathbf{0}}} \middle| \phi_{\vec{\mathbf{q}}} \right\rangle\end{aligned}$$

where in the last line we have used the group properties of $\mathcal{Q}$ to simplify the sum, and where $\left|\phi_{\vec{\mathbf{0}}}\right\rangle = \left|\phi_0^{\otimes t}\right\rangle$.

How well does a given subspace with $|\mathcal{L}| < 2^t$ approximate the full stabilizer state decomposition? Each term in the subspace state has overlap $(\langle\phi_0|R\rangle)^t$, and thus

$$\left|\left\langle R^{\otimes t} \middle| \mathcal{L} \right\rangle\right|^2 = \frac{|\mathcal{L}|^2 f_{\phi_0}^t(R)}{|\mathcal{L}|\, Z(\mathcal{L})} = \frac{|\mathcal{L}|\, f_{\phi_0}(R)}{Z(\mathcal{L})} \tag{3.27}$$

where we define $f_{\phi_0}(R) \equiv |\langle\phi_0|R\rangle|^2$, the fidelity of $|\phi_0\rangle$ with $|R\rangle$.

The fidelity of the $\mathcal{L}$ approximation thus depends on the size of the subspace, the initial overlap, and the quantity $Z(\mathcal{L})$ which depends on the subspace we choose. Bravyi & Gosset then showed for the case of the $|H\rangle$ state that we can achieve $Z(\mathcal{L}) \sim 1 + |\mathcal{L}|\, f_{\phi_0}(H)^t$ by choosing subspaces at random [49].

This argument also extends to the more general case of Clifford symmetries. Choosing subspaces uniformly at random, we can compute the expectation value of the weight $Z(\mathcal{L})$. Every subspace must contain $\left|\phi_{\vec{\mathbf{0}}}\right\rangle$, which contributes $\left\langle \phi_{\vec{\mathbf{0}}} \middle| \phi_{\vec{\mathbf{0}}} \right\rangle = 1$ to the weight. Otherwise, each state $\left|\phi_{\vec{\mathbf{q}}}\right\rangle$ is equiprobable, and occurs with probability $\frac{|\mathcal{L}-1|}{|Q|^t-1}$. Thus,

$$\mathbb{E}[Z(\mathcal{L})] = 1 + \frac{|\mathcal{L}|-1}{|Q|^t-1} \left(\sum_{\vec{\mathbf{q}}\in\mathbb{Z}_{|\mathcal{Q}|}^t-1} \left\langle \phi_{\vec{\mathbf{q}}} \middle| \phi_{\vec{\mathbf{0}}} \right\rangle \right). \tag{3.28}$$

By replacing $\sum_{\vec{\mathbf{q}}\in\mathbb{Z}_{|\mathcal{Q}|}^t-1} \left\langle \phi_{\vec{\mathbf{q}}} \middle| \phi_{\vec{\mathbf{0}}} \right\rangle$ with $\sum_{\vec{\mathbf{q}}\in\mathbb{Z}_{|\mathcal{Q}|}^t} \left\langle \phi_{\vec{\mathbf{q}}} \middle| \phi_{\vec{\mathbf{0}}} \right\rangle - 1$, and substituting

Eq. 3.25, we can write

$$\mathbb{E}\left[Z\left(\mathcal{L}\right)\right] = 1 + \frac{\left|\mathcal{L}\right| - 1}{\left|Q\right|^t - 1}\left(\left|\mathcal{Q}\right|^t f_{\phi_0}\left(R\right) - 1\right) \approx 1 + \left(\left|\mathcal{L}\right| - 1\right) f_{\phi_0}^t \approx 1 + \left|\mathcal{L}\right| f_{\phi_0}^t, \tag{3.29}$$

where we have assumed t and $\mathcal{L}$ are large.

Following the argument in [49], there is thus at least one subspace $\mathcal{L}$ such that $Z\left(\mathcal{L}\right) \leq 1 + \left|\mathcal{L}\right| f_{\phi_0}(R)^t$. Substituting this value into Eq. 3.27 gives

$$\left|\left\langle R^{\otimes t} \middle| \mathcal{L} \right\rangle\right|^2 \approx \frac{\left|\mathcal{L}\right| f^t}{1 + \left|\mathcal{L}\right| f^t}$$

, which can be rearranged to solve for how large we require $|\mathcal{L}|$ to obtain a given fidelity. More formally, and again extending the argument of [49], we can use Eq. 3.29 and Markov's lemma to show that

$$P\left[\frac{Z\left(\mathcal{L}\right)}{\left(1 + \left|\mathcal{L}\right| f_{\phi_0}(R)^t\right)\left(1 + \frac{\epsilon}{2}\right)} \geq 1\right] \leq \frac{\mathbb{E}\left[Z(\mathcal{L})\right]}{\left(1 + \left|\mathcal{L}\right| f_{\phi_0}(R)^t\right)\left(1 + \frac{\epsilon}{2}\right)} \leq 1 - \frac{\epsilon}{2 + \epsilon}.$$

Thus, with $O(\frac{1}{\epsilon})$ samples, we can generate a subspace $\mathcal{L}' : Z\left(\mathcal{L}'\right) \leq \left(1 + \left|\mathcal{L}\right| f_{\phi_0}(R)^T\right)\left(1 + \frac{\epsilon}{2}\right)$ [49].

If we now fix the size of the subspace such that

$$\begin{aligned} 2 &\leq \left|\mathcal{L}'\right| f_{\phi_0}^t \epsilon \leq 4 \\ \implies \left|\mathcal{L}'\right|^{-1} f_{\phi_0}^{-t} &\leq \frac{\epsilon}{2} \end{aligned}$$

then the corresponding subspace state $|\mathcal{L}'\rangle$ achieves a fidelity

$$\begin{aligned} \left|\left\langle R^{\otimes t} \middle| \mathcal{L}' \right\rangle\right|^2 &= \frac{\left|\mathcal{L}'\right| f_{\phi_0}(R)}{\left(1 + \left|\mathcal{L}'\right| f_{\phi_0}(R)\right)\left(1 + \frac{\epsilon}{2}\right)} \\ &= \frac{1}{(1 + \left|\mathcal{L}'\right|^{-1} f_{\phi_0}^{-t})(1 + \frac{\epsilon}{2})} \\ &\geq \frac{1}{\left(1 + \frac{\epsilon}{2}\right)^2} \approx 1 - \epsilon. \end{aligned} \tag{3.30}$$

We can thus generate an approximate stabilizer state decomposition that is ϵ

close in fidelity by choosing a random subspace, provided that we have sufficiently many terms in the decomposition. Again applying the inequality from above, we have

$$\chi_\epsilon\left(\left|R^{\otimes t}\right\rangle\right) = |\mathcal{L}| \leq 4 f_{\phi_0}^{-t} \epsilon^{-1} = O\left(f_{\phi_0}^{-t} \epsilon^{-1}\right), \tag{3.31}$$

where the asymptotic scaling depends on the term f_{ϕ_0}. Thus, we introduce the concept of stabilizer fidelity

Definition 3.2 (Stabilizer Fidelity)**.** $F(\psi) = \max_{\phi \in \mathcal{S}_n} |\langle\phi|\psi\rangle|^2$

with the corollary that

Corollary 1 *The approximate stabilizer rank of a Clifford magic state*

$$\chi_\epsilon\left(\left|R^{\otimes t}\right\rangle\right) = O\left(F(R)^{-t}\right).$$

We refer to this method of generating an approximation stabilizer state decomposition as the 'random codes' method, because it can be conceptualised as approximately encoding the state using a code-space of dimensionality $|\mathcal{L}|$.

Sparsification

The method outlined above works by taking exact, even potentially 'overcomplete' stabilizer state decompositions, and dropping terms to obtain an approximate decomposition with a given fidelity. This principle is similar to approximate classical simulation methods based on quasiprobability representations of quantum states [64]. In these models, the number of terms that must be sampled scales as the 'negativity' of the state, given by the the one-norm of the coefficients with respect to the phase-space under consideration.

It was shown by David Gosset that a similar strategy can be applied to stabilizer state decomposition [109]. Given any valid stabilizer state decomposition, we can sample terms at random using a probability distribution based on their coefficients to build an approximate decomposition. We can rewrite a given

stabilizer state decomposition as

$$|\psi\rangle = \sum_i c_i |\phi_i\rangle = \sum_i \frac{|c_i|}{\|\vec{\mathbf{c}}\|} |w_i\rangle = \sum_i p_i |w_i\rangle \tag{3.32}$$

where $|w_i\rangle \equiv \frac{c_i}{|c_i|} |\phi_i\rangle$. The new coefficients $\frac{|c_i|}{\|\vec{\mathbf{c}}\|}$ are positive, real-valued, and also have the property that $\sum_i p_i = 1$. Thus, they define a probability distribution. Let $|\omega\rangle$ be one of the $|w_i\rangle$ states, sampled with probability p_i. It can be shown that$\mathbb{E}[|\omega\rangle] \propto |\psi\rangle$ and, by taking multiple samples $|\omega_i\rangle$, we can obtain an approximate state

$$\left|\tilde{\psi}\right\rangle = \frac{\|c\|_1}{\chi_\epsilon} \sum_{i=1}^{\chi_\epsilon} |\omega_i\rangle \ : \ \left\| \left|\tilde{\psi}\right\rangle - |\psi\rangle \right\| \leq \epsilon \tag{3.33}$$

with high-probability, provided that we set [109]

$$\chi_\epsilon(|\psi\rangle) = O\left(\|c\|_1^2 \epsilon^{-2}\right), \tag{3.34}$$

Importantly, we note that this method guarantees an approximation that is ϵ close in the one-norm, as opposed to fidelity as in Eq. 3.31.

Based on this approximation, we subsequently introduce the notion of 'Stabilizer Extent':

Definition 3.3 (Stabilizer Extent)**.** For a normalised quantum state $|\psi\rangle$, the stabilizer extent $\xi(\psi)$ is defined as the minimum value of $\|c\|_1^2$ over all stabilizer state decompositions $|\psi\rangle = \sum_i c_i \phi_i$.

Thus, we approximate stabilizer rank of a sparsified decomposition scales as

$$\chi_\epsilon(\psi) = \lceil \xi(\psi) \epsilon^{-2} \rceil. \tag{3.35}$$

The one-norm is a convex quantity, and thus we can be computed efficiently using convex optimisation techniques [133]. However, as the search space scales with the number of stabilizer states, which in turn scales exponentially with the number of qubits, in practice extent is difficult to compute for more than a

few qubits. An equivalent quantity to extent was also introduced in Section 5.4 of [78], a general study of convex resource measures for quantum computing. It follows from [78] stabilizer extent is a faithful measure, such that $\xi(\psi) = 1$ if and only if $|\psi\rangle$ is a stabilizer state.

Stabilizer Fidelity

Interestingly, as a convex optimization problem, it is also possible to use the 'dual' convex problem to find a lower bound on the stabilizer extent. The proof was given by Earl Campbell in Section VI.A of [109], but we quote the key result here, namely

$$\xi(\psi) \geq \frac{1}{F(\psi)}, \tag{3.36}$$

where $F(\psi)$ is the stabilizer fidelity introduced in Definition 3.2. It can also be shown that this lower-bound is tight for injectable Clifford magic states [109]. Thus, despite the slightly different definition of approximation, the approximate stabilizer rank of Clifford magic states coincides under both the random codes and the sparsification methods.

An important question is whether stabilizer extent is multiplicative. As extent is lower bounded by the stabilizer fidelity, we can thus ask if the stabilizer fidelity is multiplicative, namely

$$F\left(\left|\psi^{\otimes t}\right\rangle\right) \stackrel{?}{=} F(\psi)^t$$

This can also be expressed as asking if there exists some entangled stabilizer state $|\varphi\rangle$, such that

$$\left|\left\langle\varphi\middle|R^{\otimes t}\right\rangle\right| > \left|\left\langle\phi^{\otimes t}\middle|R^{\otimes t}\right\rangle\right| = |\langle\phi|R\rangle|^t.$$

Lemma 4 *For single qubit states $|S\rangle$, the stabilizer fidelity of t copies $F\left(S^{\otimes t}\right) = F(S)^t$*

Proof of Lemma 4. Consider a single qubit state $|S\rangle$. Using the Bloch-vector representation, we can write this state as $|S\rangle = \frac{1}{2}\left(1 + \vec{\mathbf{r}} \cdot \vec{\mathbf{P}}\right)$, where $\vec{\mathbf{P}} =$

(X,Y,Z), and $\vec{\mathbf{r}} = (r_x, r_y, r_z)$ is the Bloch-vector with coefficients $r_i = \langle S|P_i|S\rangle$. We also consider a single qubit stabilizer state $|P\rangle$, with corresponding stabilizer group $\mathcal{S}_\phi = \{I, \pm P\}$, where $P \in \{X,Y,Z\}$.

The fidelity between $|S\rangle$ and a single qubit stabilizer state can be written in terms of the stabilizer group as

$$\begin{aligned} |\langle P|S\rangle|^2 = \langle S|P\rangle \langle P|S\rangle &= \frac{1}{2} \langle S| (I \pm P) |S\rangle \\ &= \frac{1}{2} (1 + |r_P|) \end{aligned}$$

where r_P is the Bloch-vector coefficient associated with P, and we use the result that $|\phi\rangle\langle\phi| = \frac{1}{2^n} \sum_{s \in \mathcal{S}_\phi} s$ for an n-qubit stabilizer state. Thus, the stabilizer fidelity of $|S\rangle$ is given by

$$F(S) = \frac{1}{2} \left(1 + \max_i |r_i| \right).$$

Let us assume throughout the following that

$$|r_z| \geq |r_y| \geq |r_x| .$$

This assumption can be made without loss of generality, substituting X or Y for Z through the following argument. We will also assume that $r_z > 0$, otherwise the argument follows but replacing Z with $-Z$. We define $|\phi\rangle$ to the be single qubit stabilizer state with maximum fidelity with $|S\rangle$. For t copies, the stabilizer group of $\left|\phi^{\otimes t}\right\rangle = \{Z(\vec{\mathbf{z}})\}$, where $\vec{\mathbf{z}}$ is a t-bit binary string and we employ the same notation for tensor products of Pauli operators as in Chapter 2. The fidelity of $\left|\phi^{\otimes t}\right\rangle$ with $\left|S^{\otimes t}\right\rangle$ is given by

$$\begin{aligned} \left|\left\langle \phi^{\otimes t} \middle| S^{\otimes t} \right\rangle\right|^2 &= \frac{1}{2^t} \sum_{\vec{\mathbf{z}} \in \mathbb{Z}_2^t} \left\langle S^{\otimes t} \middle| Z(\vec{\mathbf{z}}) \middle| S^{\otimes t} \right\rangle \\ &= \frac{1}{2^t} \left(\sum_{i=0}^{t} \binom{t}{i} |r_z|^i \right) \\ &= \frac{1}{2^t} \left(1 + t|r_z| + \sum_{i=2}^{t} \binom{t}{i} |r_z|^i \right) \end{aligned} \tag{3.37}$$

where we have joined together the expectation values for elements of the group with equal numbers of Pauli Z operators, and used the fact that $|r_z| \leq 1$.

Assume now that stabilizer fidelity is multiplicative up to $t-1$ copies. We will prove $\nexists |\varphi\rangle$, a t qubit stabilizer state, such that $\left|\left\langle \varphi \middle| S^{\otimes t} \right\rangle\right|^2 > F(S)^t$.

For a general t-qubit stabilizer state, we have the stabilizer group $\mathcal{S}_\varphi$, and the corresponding fidelity with $\left|S^{\otimes t}\right\rangle$ is given by

$$\left|\left\langle \varphi \middle| S^{\otimes t} \right\rangle\right|^2 = \frac{1}{2^t} \left(\sum_{s \in \mathcal{S}_\varphi} \left\langle S^{\otimes t} \middle| s \middle| S^{\otimes t} \right\rangle \right).$$

For a t-qubit Pauli operator Q, we define the weight $|Q| = \bigotimes_{i=1}^t P_i$ as the number of qubits where $P_i \neq I$ [134]. Using the assumption above, we can write the expectation value of Q on $|S\rangle$ in terms of the weight as

$$\left\langle S^{\otimes t} \middle| Q \middle| S^{\otimes t} \right\rangle = \prod_{i=1}^{n} \langle S|P_i|S\rangle \leq |r_z|^{|Q|}$$

and thus

$$\left|\left\langle \varphi \middle| S^{\otimes t} \right\rangle\right|^2 \leq \frac{1}{2^t} \left(\sum_{s \in \mathcal{S}\varphi} |r_z|^{|s|} \right).$$

All t-qubit stabilizer states are equivalent to a graph state, up to a sequence of local Clifford operations. As this is a local circuit, the weight of the stabilizers is left unchanged, and thus we can characterize the weights of $\mathcal{S}_\varphi$ using the stabilizer group of a graph states.

The stabilizer group of a graph state is generated from the underlying graph $G = (V, E)$. The jth generator is given by

$$g_j = \bigotimes_{i=1}^{t} X^{\delta_{ij}} Z^{\mathbf{1}_E(i,j)}$$

where $\mathbf{1}_E(i,j)$ is an indicator function that returns 1 if qubits/vertices i and j are connected by an edge. A product of m graph-state generators has weight $\geq m$, arising from the Pauli X term acting on each qubit. There are $\binom{t}{m}$ elements which are the product of m generators. We can also limit ourselves to

considering only fully-connected graphs, as otherwise the state $|\varphi\rangle$ is separable with respect to some bipartition, and thus

$$\left|\left\langle\varphi\middle|S^{\otimes t}\right\rangle\right| = \left|\left\langle\varphi_A\middle|S^{\otimes|A|}\right\rangle\right|\left|\left\langle\varphi_B\middle|S^{\otimes|B|}\right\rangle\right| \leq F(S)^{|A|}\,F(S)^{|B|}$$

as stabilizer fidelity is multiplicative up to $t-1$ copies. Thus, every generator must have $|g_j| \geq 2$.

Splitting up the sum, we thus have

$$\begin{aligned}\left|\left\langle\varphi\middle|S^{\otimes t}\right\rangle\right|^2 &\leq \frac{1}{2^t}\left(1+\sum_j |r_z|^{|g_j|} + \sum_{s\in\mathcal{S}_\varphi\setminus\{I,g_j\}} |r_z|^{|s|}\right)\\ &\leq \frac{1}{2^t}\left(1+t\,|r_z|^2+\sum_{i=2}^{t}\binom{t}{i}|r_z|^i\right)\\ &\leq \frac{1}{2^t}\left(1+t\,|r_z|+\sum_{i=2}^{t}\binom{t}{i}|r_z|^i\right),\end{aligned} \tag{3.38}$$

where in the final line, we have brought down the expression from Eq. 3.37.

For $t=2$, we can verify explicitly that

$$\frac{1}{4}\left(1+2\,|r_z|^2+|r_z|^2\right) \leq \frac{1}{4}\left(1+2\,|r_z|+|r_z|^2\right)$$

for all $|r_z| \leq 1$. Thus, from Eq. 3.38 and using proof by induction, stabilizer fidelity is multiplicative for all single qubit states. □

It was subsequently shown by David Gosset that stabilizer fidelity is multiplicative for 1, 2 and 3 qubit states, but that in fact for typical states with $n \geq 4$ qubits the stabilizer fidelity is not multiplicative [109]. This in turn suggests that, in general, stabilizer extent is submultiplicative.

3.3 Discussion

In this chapter, we have presented a number of results that extend our understanding of both exact and approximate stabilizer rank beyond just the 'edge-type' family of single qubit magic states.

In the case of exact stabilizer rank states, we focused on examining Conjecture 1, which asserts that the exact stabilizer rank of single-qubit states is smallest for the Clifford magic states. Explicit computational searches up to 3 copies, and an upper bound based on groups of Clifford symmetries, both provide evidence in favor of this being true. However, for other families of states, stabilizer rank has proven difficult to quantify precisely. The performance of numeric searches breaks down as the number of qubits increases. However, we were able to provide explicit upper bounds for the stabilizer rank of up to 5 copies of any quantum state, by considering the properties of the symmetric subspace.

Nonetheless, our results would suggest that in general the stabilizer rank is smallest for Clifford magic states on any number of qubits, as both the $|CS\rangle$ and $|CCZ\rangle$ magic states have stabilizer ranks significantly smaller than either their computational basis expansion, or the upper bounds obtained from the symmetric subspace.

It is also interesting to note that the $|T\rangle$, $|CS\rangle$ and $|CCZ\rangle$ magic states, all of which can be used to inject an operator from $\mathcal{C}_3$, also all have the same stabilizer rank. Given the properties of exact stabilizer rank discussed at the beginning of Section 3.2.1, in particular invariance under Clifford unitaries, this equal stabilizer rank might appear to suggest that there exists some Clifford circuit V and stabilizer states $|\phi\rangle$ such that $V\left(|T\rangle \otimes |\phi\rangle\right) = |CCZ\rangle$. This would in turn suggest that a $|CCZ\rangle$ gate can be realised using just Clifford gates and a single T gate. However, the best known unitary circuit for synthesising a CCZ gate from Clifford+T operations requires 7 T-gates [30].

Smaller circuits, have also been shown to exist that allow a CCZ gate to be synthesised using Clifford gates, T-gates, and Pauli measurements. Again, appealing to the properties of the exact stabilizer rank, we might expect that as $\chi\left(\left|T^{\otimes 3}\right\rangle\right) > \chi(|CCZ\rangle)$, we could find such a circuit with a T-count of 3. However, the current optimal circuit known has a T-count of 4 [135]. Similar results exist for the CS gate, for which the optimal circuit known requires 3

T-gates.

Additionally, there is evidence from alternative resource formulations of 'magic' that these circuits are in fact optimal. For example, in the 'Robustness of Magic' picture, it can be shown that $R\left(\left|T^{\otimes 3}\right\rangle\right) < R(|CCZ\rangle) < R\left(\left|T^{\otimes 4}\right\rangle\right)$ [75]. It is also important to note that no circuits with smaller T-counts have been found since these circuits were first proposed, despite continued research into gate-synthesis by the community. This includes efforts employing computational searches [136].

This has important consequences for the interpretation of the exact stabilizer rank as a resource measure. In particular, it is clear that the exact stabilizer rank is not a useful quantifies of magic as it relates to problems of gate synthesis. While Clifford equivalent states naturally have the same stabilizer rank, equal stabilizer rank does not imply Clifford equivalence.

However, this observation does have an important consequence for how we interpret other resource measures of magic. As previously mentioned, for example, the $|CCZ\rangle$ state has a greater robustness of magic than the $|T\rangle$. However, both states have equivalent stabilizer rank, and subsequently given an appropriate algorithm a circuit with either a single CCZ or a single T gate would be broadly equivalently difficult to simulate. Phrased another way, large 'magic' does not guarantee that a circuit shows significant non-classical behaviour.

We also present results showing how to construct approximate stabilizer state decompositions for broad classes of states. This is especially true of the sparsification method and the associated quantity of stabilizer extent. As a convex resource measure [78], it lends itself to easier explicit computation than the exact stabilizer rank, which as a form of sparse optimization is **NP**-hard even non-withstanding the exponentially growing search space [137]. As mentioned previously, the number of stabilizer states grows exponentially with the number of qubits and so in practice extent is difficult to compute for more than a few qubits. However, in some cases stabilizer rank calculations can be sped

up by taking into account features of the state like all-real amplitudes. Recent work by Gross et al. looked and optimizing a similar computation for Robustness of Magic, using symmetries in the Clifford group to significantly reduce the number of states to be considered [138].

As a convex measure, extent is also well behaved as a magic monotone [78]. For example, unlike in the case of the exact stabilizer rank, subsequent work has used the stabilizer extent to find lower bounds on the 'non-Clifford' resources required for different quantum computations [4].

From a simulation perspective, we can demonstrate the impact of building direct stabilizer state decompositions by comparing the stabilizer fidelity of states. As quoted above, the current optimal T-counts known to be required for synthesing the CS and CCZ gates are 3 and 4, respectively. Comparing the stabilizer fidelities, we can show that

$$\begin{aligned} F(T)^{-2} \approx 1.373 < \quad & F(CS)^{-1} = 1.6 \quad < F(T)^{-3} \approx 1.608 \\ F(T)^{-3} \approx 1.608 < \quad & F(CCZ)^{-1} \approx 1.778 \quad < F(T)^{-4} = 1.884. \end{aligned} \tag{3.39}$$

Asymptotically, these savings in stabilizer rank become significant. For example, a circuit built out of 40 CCZ gates would require $\sim 11\%$ of the resources compared to an equivalent circuit built in the Clifford+T basis. At 80 CCZ gates, a direct decomposition needs just 1.1% of the terms that a synthesised decomposition would require. We also note that, like in the case of robustness, these inequalities support the claim that 3 and 4 T-gates is the optimum number required to synthesise CS and CCZ.

The comparison becomes even more significant if we consider resource states that could be used to realise gates from outside of the 3rd level of the Clifford hierarchy. For example, consider a state

$$|\theta\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{i\theta/2}|1\rangle\right). \tag{3.40}$$

These resource states can be used to realize single-qubit rotations around the

Pauli Z axis through an angle θ. As previously discussed, such rotations can require up to 100 T-gates to synthesize [125] as $\theta \to 0$. However, the stabilizer fidelity of $|\theta\rangle$ actually increases as θ gets smaller. In fact, any rotation smaller than a T gate will require multiple T gates to synthesize, but has a smaller approximate stabilizer rank.

A caveat of this method is that to be used in a state-injection circuit, resource states like $|\theta\rangle$ require non-Clifford 'correction operations'. In the approximate simulation case, however, we cannot post-select on the measurement gadgets to avoid these additional non-Clifford gates [49]. Thus, to be used in a simulation scheme, we need a different formulation than the PBC method discussed in Section 3.1.1. We will discuss the problem of applying the sparsification method to simulation in the following chapter.

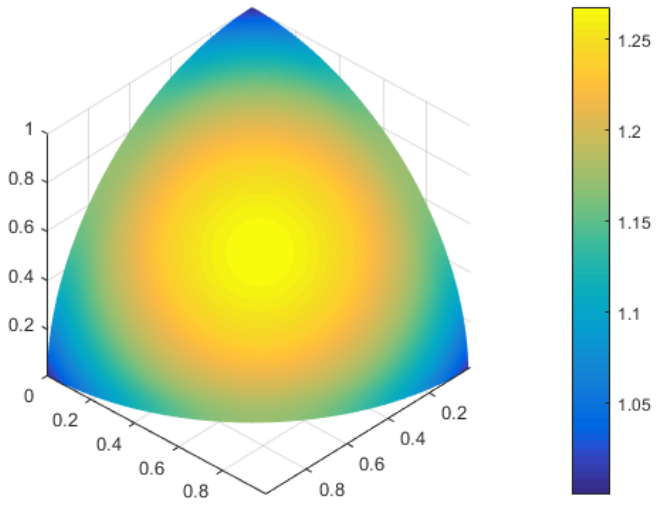


Figure 3.3: Heatmap showing the inverse stabilizer fidelity, $F(\psi)^{-1}$, for all single-qubit states lying in a single quadrant of the Bloch sphere. We note that the state with the largest inverse fidelity, and thus largest stabilizer extent, is the face-type magic state. Figure taken from [109].

It is interesting to note that there is a tension between trying to minimize the exact or the approximate stabilizer rank. For example, consider the two classes of single-qubit magic state. While the face state has a smaller stabilizer fidelity that the edge-type states — in fact it has the smallest possible

stabilizer fidelity, c.f. Fig 3.3 — both states have equivalent exact stabilizer rank. Alternatively, single-qubit states exponentially close to a stabilizer state will have a small approximate stabilizer rank, but have a larger exact stabilizer rank. In fact, for any single qubit state with stabilizer fidelity $\geq \frac{1}{\sqrt{2}}$, its approximate stabilizer rank will have an asymptotic scaling smaller than $2^{0.5t}$, the upper bound obtained from computational searches in Section 3.2.1.

Importantly, we note that all the results shown here serve only as upper bounds on the stabilizer rank. While stabilizer fidelity is capable of lower bounding the stabilizer extent, this is itself also an upper bound on the approximate stabilizer rank. Currently, the only lower bounds that have been explicitly proven apply to the case of the $|T\rangle$ state. It was argued in [123] that $\chi\left(\left|H^{\otimes n}\right\rangle\right) = \Omega\left(\sqrt{n}\right)$, based on constructing states with finite stabilizer rank but which require a large number of T gates to create. In [109], it was also shown that the approximate stabilizer rank $\chi_\epsilon\left(\left|H^{\otimes n}\right\rangle\right) = \Omega\left(F(H)^{-n}\epsilon^{-2}\right)$, but only under the assumption that the decomposition is built from the states $|0\rangle$ and $|+\rangle$, as used in the random codes construction.

Part of this difficulty likely arises from the fact that, as a sparse optimization problem, even approximately computing the optimal stabilizer rank is **NP**-hard [137]. Currently, the best explicit lower bound is a complexity theoretic argument. It was shown by Dalzell that for any classical simulation of a Clifford+T circuit must have a runtime that scales asymptotically as $2^{\gamma t}$, where $\gamma > \frac{1}{128}$, otherwise the polynomial hierarchy would collapse to the third level [38]. This result acts as a lower-bound on the approximate stabilizer rank of the $|T\rangle$ magic states, and is significantly looser than the current best known decompositions with $\gamma \approx 0.23$. It is a open question if this complexity theoretic argument can be tightened.

In the case of approximate stabilizer rank, we might also ask if alternative strategies for building decompositions could be yield a smaller approximate stabilizer rank.

For example, inspired by the random codes method, we might consider using Schumacher compression to efficiently encode many copies of a resource state [33]. In [49], the authors compared the Shannon entropy of the $|H\rangle$ state with the asymptotic $2^{0.23t}$ scaling obtained using random codes, showing that it outperforms Schumacher compression. Figure 3.4, taken from [110], shows a similar analysis comparing the Shannon entropy with the stabilizer extent for $|\theta\rangle$ states as a function of the angle θ. We can see that in fact, at small angles, Schumacher compression can achieve a smaller decomposition than sparsification. However, over most of the parameter range, the sparsification method performs significantly better.

Alternatively, we could consider techniques that construct an approximate stabilizer state decomposition by discarding only the terms which contribute least to the overall decomposition. Indeed, an interesting feature of both the random codes and sparsification methods is that the resulting decompositions are uniform mixtures of the sampled stabilizer states.

In the most general case, the simulation overhead achieved by discarding terms is related to the notion of ϵ-sparsity, namely how many terms in the decomposition can be discarded while retaining an additive error ϵ in the output distribution [47]. The notion of ϵ-sparsity is closely related to the smooth max entropy H^{ϵ}_{max}, the logarithm of the number of terms with coefficients $|c_i| > \epsilon$. An ideal truncation method of this type would have approximate stabilizer rank $2^{H^{\epsilon}_{max}}$, but as previously stated computing optimal sparse decompositions of this type constitutes an **NP**-hard problem.

In some tensor network methods for classical simulation, such as `Rollright` and `qFlex`, the output state of the computation before measurement is broken up into a decomposition of largely independent states, which contribute roughly equally to the norm [139, 140]. Thus, they can achieve an approximate fidelity f by dropping all but a fraction f of the states [139]. In practice, this method does slightly worse than the target fidelity due to small overlaps $\sim 10^{-6}$ between states [140], but the reduction in the number of terms achieved

is significant.

The overlap between general n-qubit stabilizer states, in contrast, varies significantly from 0 to $2^{-s} : s \in [1, n]$. The sparsification method works around this, using the fact that sampling states according to $\frac{c_j}{\|\vec{c}\|_1}$, the expectation value

$$\mathbb{E}\left[|\omega\rangle\right] = \frac{|\psi\rangle}{\|c\|},$$

i.e. the sampled states have equal norm on average [109]. Additionally, grouping like terms, we can see that in the sampled state

$$\mathbb{E}\left[\left|\tilde{\psi}\right\rangle\right] = \mathbb{E}\left[\frac{\|c\|}{\chi_\epsilon}\sum_{i=1}^{\chi_\epsilon}|\omega_i\rangle\right] = \mathbb{E}\left[\|c\|\sum_j \frac{\#|w_j\rangle}{\chi_\epsilon}|w_j\rangle\right] = \|c\|\sum_j p_j|w_j\rangle\,.$$

On average, then, building uniform decompositions in this way does effectively weight each stabilizer state according to its contribution to the decomposition. It might however be interesting to test alternative sampling strategies, such as sampling without replacement or excluding terms with a coefficient below some small threshold value.

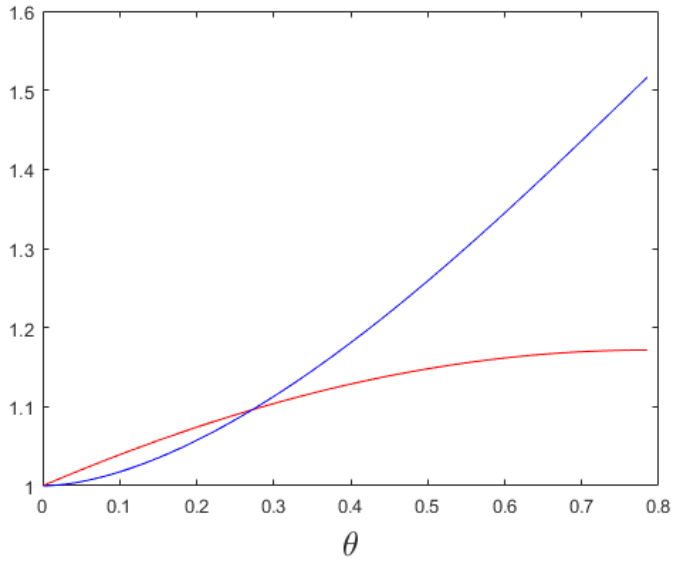


Figure 3.4: Graph comparing the Shannon entropy (blue) and stabilizer extent (red) of $|\theta\rangle$ states, defined in Eq. 3.40, as a function of the angle $|\theta\rangle$.

Chapter 4

Simulating Quantum Circuits with the Stabilizer Rank Method

4.1 Introduction

Previously, we have discussed decompositions of quantum computations, where each individual term can be efficiently simulated classically. This connection to classical simulation gives an easy operational interpretation to these decompositions, and suggests a way of building a classical simulator along these lines. In this chapter, we will make this connection explicit, introducing methods that can be used to simulate universal quantum circuits, and discussing their implementations.

Despite the fact that they are believed to be intractable in the general case, classical simulations play an important role in the research and development of quantum technologies. In recent years, as quantum hardware has continued to improve, an increasingly important role of simulations has been to support the transition and adoption of quantum technology. Providing classical simulators as a test bed enables the development of software engineering, protocols and applications that take into account non-classical features, even while access to actual quantum devices is still limited.

For example, `SimulaQron` and `NetSquid` are classical simulations of quantum communications networks, developed as part of an effort to promote the development of practical quantum communications [141, 142]. These tools have been used to develop proposals for link layer protocols in quantum networks, which can then be tested in the lab [143].

In the context of quantum computing, many classical simulators in use today form part of 'Quantum Development Kits' (QDKs), software environments for the development of quantum software. These tools broadly follow a similar architecture to that of `ProjectQ`, described in [144]. Typically, the user-facing component is a 'high-level' description of a quantum programme, either as an API or with a domain-specific language (DSL), which is agnostic to how it will be evaluated. These programmes can be built out of algorithms and meta-algorithms, such as the Variational Quantum Eigensolver; subroutines and operations, such as the quantum Fourier transform; or even individual gates. The resulting description of the programme can then be compiled to a quantum circuit, and either simulated clasically, or else dependent on requirements further compiled and dispatched to a quantum processor. Multiple such QDKs have been developed over the past 5 years, and a brief summary of the some of the available options is shown in Table 4.1.

Framework	High-level Description	Classical Methods	Supported Hardware
Microsoft QDK [19]	`Q#` [145]	State vector	None
`ProjectQ` [146]	DSL	State vector	IBMQ [7]
`Qiskit` [1]	QASM [147], `Python` API	Various	IBMQ [7]
`Circ` [148]	`Python` API	State vector, density matrix	Bristlecone[1] [20]
`Forest` [149]	Quil, Python API [150]	State vector density matrix	Rigetti QPU [151]

Table 4.1: A non-exhaustive list of different quantum software frameworks or QDKs. We note that many of these frameworks have additional components aimed at supporting application development that are not mentioned here.

In practice, most of the QDKs mentioned above make use of what could be described as 'textbook' classical simulations of quantum computing, where a circuit is simulated by matrix multiplication of the unitary associated with each gate, acting on either a state-vector or a density matrix description [30].

These simulators have the advantage that they are relatively straightforward to implement, and can leverage mature computational libraries for linear algebra such as `Numpy` [126]. Probabilities in the computational basis can also be

trivially obtained, either by reading off the right diagonal entry in the density matrix or computing the absolute value squared of the corresponding amplitude in a state-vector. Noise in these models is also relatively straightforward to model, either by using a stochastic noise model inserting extra operators into the circuit in the state-vector case, or else applying Kraus operators directly in the density matrix case [30].

However, the main drawback to these simulators is their spatial complexity. A state-vector requires 2^n complex numbers to define, and a density matrix requires up to 2^{2n}, where n is the number of qubits. As each complex number requires two 64-bit floating point numbers to specify, the memory requirements can quickly approach the limits of personal computing. A simulation on 30 qubits requires 16GB of memory, and up to 45 qubits this requires 0.5PB [152]. The current top-ranked supercomputer in the world has access to 2.7PB of memory, meaning it could simulate up to 47 qubits using these methods [153].

These classical representations also have a significant temporal overhead. In the most straightforward implementation, applying gates requires multiplying $2^n \times 2^n$ matrices. These updates require time 2^{2n} in the state-vector case, and $2^{4.746n}$ for density matrices. In practice though, significant optimizations are possible that can make state-vector simulators reasonably performant at accessible sizes. For example, the $2^n \times 2^n$ matrices representing single qubit gates are sparse, with the vast majority of entries being 0 or 1. Other optimizations that have been applied include parallelising single-qubit gate updates [154, 155, 154, 156], optimising permutation operations such as CNOT and Pauli X [155], replacing certain arithmetic operations with classical equivalents [152], and accelerating algorithms using parallel execution via `OpenMP`, `MPI` or GPUs [157, 154, 155, 156].

In practice, this limit of approximately 30 qubits when simulating circuits with personal computers roughly corresponds to the kind of quantum programmes that can be run on current publicly accessible devices, which have anywhere from 5-20 qubits [7, 151]. However, with continued development of quantum

hardware into the 50–72 qubit range [158, 159], classical simulations need to be pushed further, to continue in their other role in the verification and benchmarking of quantum devices.

Given that the complexity of classical simulations generally scales exponentially in the system size, the question of verifying quantum computations without simulation constitutes a separate branch of research, based on the idea of 'interactive proofs' [160, 161]. Nonetheless, classical simulations offer a unique opportunity in verification as at any point the simulation can be paused and the system state inspected. Large-scale classical simulations also provide a performance baseline, as part of attempts to establish Quantum Supremacy [95, 99].

Recent work has focused on tensor-network methods, as introduced in Section 1.2.2, to push classical simulations up past 45 qubits, and have achieved some of the largest scale classical simulations to date [162, 163, 164, 139, 102]. These papers on large classical simulations focus on on simulating quantum circuits on grids of qubits with local connectivity. This restriction is motivated by the designs of current quantum processors, and also allows for specific optimizations that reduce the temporal complexity of the simulation. State of the art methods typically split this grid into sub-blocks which are locally contracted, leaving only connections between blocks [162, 164, 139, 140]. The remaining s contractions are then 'sliced', fixed to one of 2^s values and contracted fully [162]. This has a natural operational interpretation in terms of a sum-over-paths expansion [139], and has the advantage that the contractions within blocks can be parallelised.

These methods all achieve runtimes that scale as $\max\left[2^{dl}, 2^n\right]$, where l is the length of the longest edge of the grid [59] and d is the circuit depth. They also have exponential spatial requirements, though these are reduced compared to a state-vector method by virtue of tensor slicing. Through application of supercomputing resources, these methods have simulated random universal circuits of up to depth 40 on 72 qubits [140], depth 35 on 100 qubits [163], and

depth 24 on 121 qubits [102].

Because this generation of quantum hardware aims to maximize qubit count, it will not employ full error-correction routines. As a result, noise is a significant factor in the system, and limits the depth of circuits that can be run. We refer to this regime of quantum computing as 'Noisy Intermediate Scale Quantum' or NISQ [100]. Thus, much like system size for the state-vector simulator, the exponential simulation overhead in the depth does not render the simulations intractable. In fact, simulators can benefit from the increased noise level, by dropping terms from the simulation and reducing the overall computational time required by a constant factor, as discussed in Section 3.3.

4.2 Results

In the rest of this chapter, we will discuss a distinct method for simulating universal quantum circuits, based on stabilizer state decompositions. These methods are then implemented in software, including a version integrated into the Qiskit QDK. Using these implementations, we will present simulation results for several types of quantum circuit, and argue that this method has a great potential for simulating circuits on current and near-term quantum hardware.

4.2.1 Methods for Manipulating Stabilizer Decompositions

At a high level, simulating a quantum circuit U using a decomposition into efficiently simulable terms requires two main stems. Firstly, we need to build a representation of the circuit state $U|x\rangle$ for an input state $|x\rangle$, which is itself part of our efficiently representable set of states. Then, we need a routine for computing output variables from the distribution, either computing explicit probabilities if we interested in strong simulation, or else sampling from the output distribution if we are interested in weak simulation.

In the following, we will use `U` to denote a classical description of the quantum circuit U. We store `U` as a sequence of gates, where each gate includes its label, e.g. 'H', and the labels of the qubits it acts on.

Stabilizer states will be encoded classically using either the CH or the DCH representations, introduced in Chapter 2.

Building Decompositions

The main method for constructing a stabilizer state decomposition, given a description of a quantum circuit U, is the PBC method introduced in [123] and [49], and outlined previously in Sections 3.1.1 and 3.1.2. We will review the method briefly here, with a focus on implementation in software.

Implementing a PBC requires rewriting U as an equivalent Clifford circuit V. We achieve this by walking through the circuit `U`, and replacing each of the m non-Clifford gate with an appropriate magic state or states, and state-injection gadget, such as the example shown in Figure 3.1. We note that this requires a library of known gadgets for implementing different gates. The result is a new circuit U', acting on n qubits and m magic states.

State-injection gadgets include additional, measurement controlled 'correction' operations. By post-selecting on these measurement-outcomes, we can expand out U' as a sum of different Clifford circuits $V_{\vec{\mathbf{y}}}$

$$U|\vec{\mathbf{x}}\rangle = \sum_{\vec{\mathbf{y}}} \langle \vec{\mathbf{y}}|V_y|\vec{\mathbf{x}} \otimes \psi\rangle$$

where $\vec{\mathbf{y}}$ is the post-selection string with length $O(m)$, and $|\psi\rangle$ is the joint state of all the magic states.

It was shown in [49] that given some approximate stabilizer state decomposition of the magic states $\left|\tilde{\psi}\right\rangle$, we can construct a PBC to sample from the output distribution of the circuit by sampling the post-selection string at random. Thus, for each gadget, we sample the measurement outcomes appropriately to build-up the Clifford circuit `V`.

As previously discussed, when injecting a gate U the correction operation has the form $UPU^\dagger$ for some Pauli operator P. If $U \in \mathcal{C}_3$, then by definition $UPU^\dagger$ is a Clifford operator and we are done. Otherwise, we will need to introduce additional layers of state-injection until we build an all-Clifford circuit `V`.

Finally, we need to construct an approximate stabilizer state decomposition for the magic states $|\psi\rangle$. In general, $|\psi\rangle$ will be a tensor-product of different 'species' of magic state, and so we build the full approximation using the multiplicative upper bound

$$\chi_\epsilon\left(|\psi\rangle\right) = \chi_\epsilon\left(|T\rangle^{\#T}\right)\chi_\epsilon\left(|CCX\rangle^{\#CCX}\right)\chi_\epsilon\left(|\theta\rangle^{\#\theta}\right)\cdots$$

For Clifford magic states, we can make use of the random codes construction. Otherwise, we can use sparsification. We note that this again implies a library of best-known decomposition strategies for each magic-state we introduce.

Overall then, the gadgetization method takes as input a classical description of an n-qubit circuit `U` and target error ϵ, and returns a new description of a Clifford circuit `V` acting on n qubits and m magic states, the corresponding post-selection string $\vec{\mathbf{y}}$, and an approximate stabilizer state decomposition $\left|\tilde{\psi}\right\rangle$. A pseudo-code description of this method is given in Algorithm 3.

The Sum-over-Cliffords picture

The PBC model has an interesting feature where the number of qubits in the stabilizer state expansion depends only on the magic states, and not on the number of qubits in the circuit. Stabilizer circuits are efficient to simulate in terms of the number of qubits, but the $O(n^3)$ overhead is still considered significant in practice. Thus, if there are fewer magic states, the PBC can reduce the number of variables in the simulation. But, in general, universal quantum computations have a number of gates that scales as $\mathrm{poly}\,(n)$, and gadgetization will result in more qubits.

An alternative strategy for building stabilizer state decompositions makes use of the equivalence between stabilizer circuits and stabilizer states. If we consider a Clifford gate decomposition $Q = \sum_i \alpha_i V_i$, then the action of Q on a stabilizer state results in a stabilizer state decomposition

$$Q\,|\phi\rangle = \sum_i \alpha_i V_i\,|\phi\rangle = \sum_i \alpha_i\,|\phi_i\rangle\,, \tag{4.1}$$

which we can then turn into an approximation stabilizer state decomposition with sparsification, giving a decomposition with a rank $O(\|\vec{\alpha}\|^2)$.

From this, we can define a notion of 'extent' for a unitary

$$\xi(Q) = \min_V \|\vec{\alpha}\|^2 \; : \; Q = \sum_i \alpha_i V_i. \tag{4.2}$$

For example, considering single-qubit rotations in around the Z axis of a Bloch sphere with $\theta \in [0, \pi/2]$, we can expand them into two Clifford branches

$$R_Z(\theta) = \left(\cos\frac{\theta}{2} - \sin\frac{\theta}{2}\right) I + \mathrm{e}^{-\mathrm{i}\pi/4}\sqrt{2}\sin\frac{\theta}{2} S, \tag{4.3}$$

with corresponding extent $\xi(R(\theta)) = \left(\cos\frac{\theta}{2} + \tan\frac{\pi}{8}\sin\frac{\theta}{2}\right)^2$ [109]. Similar results can be found for all Z rotations, where we slightly adjust the phase and the Clifford operations on each branch.

This expansion corresponds with the stabilizer extent of the $|T\rangle$ state by setting $\theta = \frac{\pi}{4}$. In fact, it was shown by Earl Campbell that for injectable Clifford magic states, such as $|T\rangle$ and $|CCZ\rangle$, the extent-optimal stabilizer state decomposition can be used to 'lift' a Clifford gate expansion of the corresponding unitary (i.e. T and CCZ), that is also optimal [109].

Using submultiplicativty, we can thus upper-bound the stabilizer extent of the circuit U as

$$\xi(U) = \prod_{i=1}^{m} \xi(U_i) \tag{4.4}$$

for each non-Clifford gate U_i. We can then build up a term in the stabilizer state decomposition by iterating through `U`. If the gate is Pauli or Clifford, we just apply it and update the state. Otherwise, for each non-Clifford gate U_i we sample a branch j from the Clifford expansion with $p_{i,j} = \frac{|\alpha_{i,j}|}{\|\vec{\alpha_i}\|}$ as in the sparsification method, and apply the corresponding Clifford gate $V_{i,j}$. We can repeat this $O(\xi(U))$ times, to produce a stabilizer state decomposition of the $U|\vec{\mathbf{x}}\rangle$. This algorithm is outlined in Algorithm 4.

Algorithm 3 Pseudocode description of the computational routine for construction a stabilizer state decomposition of a quantum circuit using state-injection gadgets.

Require: Known set of gadgets for non-Clifford gates.

function GadgetDecomposition($\mathtt{U}, \epsilon$)
  $\mathtt{V} \leftarrow \emptyset$ ▷ Output Clifford circuit
  $|\psi\rangle \leftarrow \emptyset$ ▷ Magic states
  for $\mathtt{U_i} \in \mathtt{U}$ **do**
    if $U_i \notin \mathcal{C}_2$ **then**
      Sample a measurement outcome z
      $\mathtt{V} \leftarrow \mathtt{V} \cap \mathtt{G} \cap \mathtt{V}_z$ ▷ $\mathtt{G}$ is the gadget for U_i.
      $|\psi\rangle \leftarrow |\psi\rangle \otimes |\psi_G\rangle$ ▷ Add magic state associated with $\mathtt{G}$
    else
      $\mathtt{V} \leftarrow \mathtt{V} \cap \mathtt{U_i}$
    end if
  end for
  Reorder qubits in $\mathtt{V}, |\psi\rangle$ to join common species of magic state
  $\left|\tilde{\psi}\right\rangle = \emptyset$
  for $\left|\psi_{U_j}^{\otimes \# U_j}\right\rangle \in |\psi\rangle$ **do**
    $\left|\tilde{\psi}\right\rangle \leftarrow \left|\tilde{\psi}\right\rangle \otimes \left|\tilde{\psi_{U_j}}\right\rangle$ ▷ Rank is set by ϵ.
  end for
  return $\mathtt{V}$, $\left|\tilde{\psi}\right\rangle$
end function

Algorithm 4 Pseudocode description of building stabilizer state decompositions in the sum-over-Cliffords picture.

Require: Clifford decompositions of non-Clifford gates.

function SumOverCliffordDecomposition($\mathtt{U}, \epsilon, |x\rangle$)
  $\left|\tilde{\psi}\right\rangle = \emptyset$
  $\xi \leftarrow$ ComputeExtent($\mathtt{U}$)
  $i \leftarrow 0$
  while $i < \chi_\epsilon = O\left(\xi \epsilon^{-2}\right)$ **do**
    $|\phi\rangle \leftarrow |x\rangle$
    $c \leftarrow 1$
    for $U_i \in \mathtt{U}$ **do**
      if $U_i \notin \mathcal{C}_2$ **then**
        Sample Clifford branch j of gate U_i
        $|\phi\rangle \leftarrow V_{i,j} |\phi\rangle$
        $c \leftarrow \frac{\alpha_{i,j}}{|\alpha_{i,j}|} c$
      else
        $|\phi\rangle \leftarrow U_i |\phi\rangle$
      end if
    end for
    $\left|\tilde{\psi}\right\rangle \leftarrow \left|\tilde{\psi}\right\rangle + c|\phi\rangle$
    $i \leftarrow i + 1$
  end while
  return $\left|\tilde{\psi}\right\rangle$
end function

Algorithm 5 Pseudocode outline of the Norm Estimation routine for computing expectation values of Pauli projectors.

Require: L, number of samples to take, n, number of qubits, Π, Pauli projector
  function NormEstimation(Π, $\left|\tilde{\psi}\right\rangle$)
    $\vec{\eta} \leftarrow \{\eta_i = 0\}$
    $\{|\eta_i\rangle\} \leftarrow \{$RandomEquatorialState(n)$\}$
    for α_i, $|\phi_i\rangle \in \left|\tilde{\phi}\right\rangle$ **do**
      $\Gamma \leftarrow 1$
      for $P \in \Pi$ **do**
        $\Gamma_P, |\phi_i\rangle \leftarrow$MeasurePauli($P$,$|\phi\rangle$)
        if $\Gamma_P = 0$ **then**
          $\Gamma \leftarrow 0$, Break loop
        end if
        $\Gamma \leftarrow \Gamma\Gamma_P$
      end for
      if $\Gamma \neq 0$ **then**
        for $|\eta_i\rangle \in \{|\eta_i\rangle\}$ **do**
          $\vec{\eta_i} \leftarrow \Gamma\,\alpha_i\,\langle\eta_i|\phi_i\rangle + \vec{\eta_i}$
        end for
      end if
    end for
    return $\frac{2^n}{L}\sum_i |\vec{\eta_i}|^2$
  end function

Algorithm 6 Pseudocode description of the Metropolis-style Monte Carlo method for sampling a computational basis string x from the output distribution of a stabilizer state decomposition.

Require: n, number of qubits
  function MetropolisSampling($\left|\tilde{\psi}\right\rangle$, m)
    $\vec{\mathbf{x}} \leftarrow$ Random initial n-bit binary string
    $p_x \leftarrow |\sum_i \alpha_i \langle x|\phi_i\rangle|^2$
    for $j \in [1,\ldots,m]$ **do** ▷ m repetitions of the random walk step
      $j \leftarrow$ Random integer $\in [1,\ldots,n]$
      $\vec{\mathbf{x}}' \leftarrow \vec{\mathbf{x}} \oplus \vec{\mathbf{e}}_j$
      $p_{x'} \leftarrow |\sum_i \alpha_i \langle x'|\phi_i\rangle|^2$
      if $p_x = 0$ **then** ▷ Always move away from 0 amplitudes
        $\vec{\mathbf{x}} \leftarrow \vec{\mathbf{x}}'$, $p_x \leftarrow p_{x'}$
      else
        Generate $r \in [0,1)$ uniformly at random
        if $r < \frac{p_{x'}}{p_x}$ **then** ▷ Always accept if $p_{x'} > p_x$
          $\vec{\mathbf{x}} \leftarrow \vec{\mathbf{x}}'$, $p_x \leftarrow p_{x'}$
        end if
      end if
    end for
    return $\vec{\mathbf{x}}$
  end function

Output Variables

There are two main methods for computing output variables from a given stabilizer state decomposition. The first is the 'norm estimation' routine, introduced in [49] and refined in [109]. Norm estimation can be used to compute measurement probabilities, and also to sample as described in Section 3.1.2. The second is a Metropolis-style Monte Carlo method, which can be used to return samples in the computational basis. Both methods were developed by Sergey Bravyi, and we introduce them here with a view to their implementation. The two methods are also outlined in Algorithms 5 and 6, respectively.

Norm Estimation

This routine allows us to quickly compute an approximation to $\|\psi\|$. Importantly, given a projector Π, we can compute the probability of that outcome as

$$p(\Pi) = \frac{\|\Pi\psi\|^2}{\|\psi\|^2}. \tag{4.5}$$

In particular, it is possible to show that if we generate equatorial stabilizer states $|\eta\rangle$ uniformly at random, then the random variable $\eta \equiv 2^{n/2}\,|\langle\eta|\psi\rangle|$ has the property that

$$\mathbb{E}\left(\eta^2\right) = \|\psi\|^2 \quad \mathbb{E}\left(\eta^4\right) \leq 2\|\psi\|^4$$

and thus, the average inner product of $|\psi\rangle$ with equatorial stabilizer states is equal to norm of $|\psi\rangle$ squared, with variance at most $\|\psi\|^4$ [109].

The number of samples we need depends on the accuracy desired. It can be shown that given an estimate

$$\bar{\eta} = \frac{1}{L}\sum_i |\eta_i|^2$$

then $\bar{\eta}$ approximates $\|\psi\|$ to within ϵ relative error $\bar{\eta} = (1 \pm \epsilon)\,\|\psi\|$ with probability $\frac{3}{4}$, provided $L = 4\epsilon^{-2}$ [109]. We can then decrease the failure probability to $\delta \leq \frac{1}{4}$ by taking $O\left(\log \delta^{-1}\right)$ estimates of $\bar{\eta}$.

In Section 2.2.2, we introduced an algorithm for computing inner products

between stabilizer states $|\phi\rangle$ and equatorial stabilizer states. This method has computational complexity $O(n^3)$. Thus, given a stabilizer state decomposition $\left|\tilde{\psi}\right\rangle$, we can use compute $\left\|\tilde{\psi}\right\|$ in time $O(L\chi n^3)$, where L is the number of samples of η.

As part of the sampling routine described in Section 3.1.2, we want to compute marginal probabilities $P(x_1, x_2, \ldots x_j)$ for some j-bits. These marginals correspond to fixing j qubits, and projecting the rest onto a 2^{n-j} dimensional codespace, generated by j Pauli operators [49]. This codespace can be generated by j Pauli operators, giving

$$\Pi = \prod_{i=1}^{j} \frac{1}{2}(I + P_i)$$

where P_i are n qubit Pauli operators. We can thus apply this projector by measuring each of the Pauli generators in turn. Recall that each Pauli measurement takes time $O(n^2)$, (c.f. Section 2.2.2) and thus computing $\left\|\Pi\tilde{\psi}\right\|^2$ also has runtime $O\left(L\chi n^3\right)$.

To avoid accumulation of errors, each marginal probability needs to be computed with multiplicative error $O(w^{-1})$ when sampling from w output bits. Using the bound on the approximation accuracy above, this implies $L = O(w^2)$. As there are w marginals to compute, sampling with the norm estimation method thus takes time $O(\chi n^3 w^3)$.

In the gadgetized picture, we can employ norm estimation by first setting the measurement projector Π, and then updating it to obtain the corresponding PBC Π_s by conjugating the projector with the Clifford circuit $\mathtt{V_y}$. These Pauli updates can be computed efficiently classically, using similar update rules as for a stabilizer tableau [66]. Otherwise, for decompositions obtained using the sum-over-Cliffords method, no further preprocessing is required.

We note that norm estimation is also required to compute individual computational basis amplitudes, and to convert a stabilizer state decomposition into the state vector picture. Recalling that in the CH and DCH representations,

we can compute $\langle\vec{\mathbf{x}}|\phi\rangle$ in time $O(n^2)$, this means that for a given stabilizer state decomposition we can compute $\left\langle\vec{\mathbf{x}}\middle|\tilde{\psi}\right\rangle$ as

$$p(\vec{\mathbf{x}}) = |\langle\vec{\mathbf{x}}|\psi\rangle|^2 \approx \left|\left\langle\vec{\mathbf{x}}\middle|\tilde{\psi}\right\rangle\right|^2 = \frac{\left|\sum_i \alpha_i \langle\vec{\mathbf{x}}|\phi_i\rangle\right|^2}{\|\psi\|^2} \tag{4.6}$$

in time $O(\chi n^2)$. To avoid potential floating point errors, stabilizer states in the decomposition are stored only with their relative phase coefficients. Thus, these amplitudes needs to be reweighted by $\left\|\tilde{\psi}\right\|$.

Metropolis Estimation

One advantage of norm estimation is that it can be used to compute the probability of arbitrary Pauli measurements. However, as discussed, while technically polynomial it has a runtime up to $O(n^6)$ in the number of qubits. Thus, an alternative strategy based on Metropolis Monte Carlo methods was proposed by Sergey Bravyi, that also makes use of the ability to compute individual computational basis amplitudes.

The idea is to define a random walk through the set of computational basis strings, flipping one bit at a time and computing the amplitude of the new string. If at some time-point we have computational basis string $\vec{\mathbf{x}}$ and amplitude $\left|\left\langle x\middle|\tilde{\psi}\right\rangle\right|$, then we obtain x' by flipping a single bit at random, and compute $\left|\left\langle\vec{\mathbf{x}}'\middle|\tilde{\psi}\right\rangle\right|$. If the new amplitude is larger, we accept the move. Otherwise, we accept with fixed probability

$$p = \frac{\left|\left\langle\vec{\mathbf{x}}'\middle|\tilde{\psi}\right\rangle\right|^2}{\left|\left\langle\vec{\mathbf{x}}\middle|\tilde{\psi}\right\rangle\right|^2}.$$

It can be argued that, assuming that the random walk is 'irreducible' such that for any pair of strings x, y there exists a path of single-bit moves between them with non-zero amplitude, the steady state distribution of this walk converges to the output distribution of the circuit in time $\text{poly}(n)$ [109]. In practice, we have used this method to obtain samples from the output distribution on

50 qubit circuits with mixing time of ~ 2000 steps [109]. Importantly, once the chain has been mixed, we can then obtain samples by continuing to run the core random-walk step (contained within the `For` loop in Algorithm 6) for a further s repetitions, recording the string $\vec{\mathbf{x}}$ at the end of each step as one sample.

In general, computing amplitudes requires time $O\left(\chi n^2\right)$, combining the contributions from each term in the decomposition. While we store an unnormalised description of the approximate state $\left|\tilde{\psi}\right\rangle$, here we can avoid the need to perform norm estimation as in Eq. 4.6, as we are interested in ratios of amplitudes and so the norms cancel. We might expect then that the Metropolis method to have a runtime that scales as $O\left(\chi n^2\right)$. However, we can actually remove a factor of n from the runtime of the random-walk step by exploiting the fact we are flipping single bits at a time.

Recall from Section 2.2.2 that we compute the computational amplitudes $\vec{\mathbf{x}}$ in the CH and DCH picture by commuting a Pauli $X(\vec{\mathbf{x}})$ past the CH/DCH layers which we denote here as a Clifford circuit W. We can store this resulting Pauli operator $P' = W^\dagger X(\vec{\mathbf{x}})W$, which takes $O(n^2)$ to compute, for a constant memory cost. We can then compute the operator Q', obtained by commuting $X(\vec{\mathbf{x}'})$, as

$$Q' = WX(\vec{\mathbf{x}'})W^\dagger = WX(\vec{\mathbf{e}}_j)X(\vec{\mathbf{x}})W^\dagger = WX(\vec{\mathbf{e}}_j)W^\dagger P'.$$

Because $X(\vec{\mathbf{e}}_j)$ acts as the identity everywhere except qubit j, commuting this operator through the Clifford layer can be optimised to ignore any terms except for those involving qubit j. By inspection of Eqs. 2.46, 2.47,2.48 and 2.49, this takes time $O(n)$ as each vector-matrix multiplication will involve only a single row or column.

Thus, overall then, if we run the Metropolis method for time $m + s = T$ to obtain s samples, the runtime scales as $O\left(\chi n^2\right) + O\left(T\chi n\right)$.

4.2.2 Implementation of the Simulator

To implement these simulation methods, the fundamental data-structures we consider are arrays of stabilizer states, and their complex coefficients. The stabilizer states themselves are encoded using either the CH or the DCH representation, as each encoding supports the necessary update routines including fast inner-product calculations with equatorial states.

Building on the existing implementations of the CH and DCH simulators discussed in Section 2.2.3, the simulator was written in `C++`. In the previous section, we introduced two distinct approaches for building stabilizer state decompositions, and two distinct methods for computing output variables. Thus, the simulator was designed using the 'strategy' design pattern [165], which allows different algorithms for the same task to be used interchangeably.

The core of the simulator is a class we call `Runner`, which is responsible for maintaining the stabilizer state decomposition. The `Runner` class is initialized with the target stabilizer rank, the number of qubits and, optionally, the initial (stabilizer) state of the computation $|\vec{\mathbf{x}}\rangle$. By default, we set $|\vec{\mathbf{x}}\rangle = \left|\vec{\mathbf{0}}\right\rangle$

Because the specifics of building a stabilizer state decomposition will depend extensively on the circuit, including factors like the choice of gadget or Clifford decomposition, the `Runner` accepts user-defined strategies. These can be implemented using 'function objects', classes that can be called like functions [166]. This allows the decomposition strategy to have internal state information, e.g. the choice of 'subspaces' used to decompose Clifford magic states, which is kept separate from the resulting stabilizer state decomposition. The user defines their decomposition strategy by sub-classing the `DecompositionBuilder` class, and at runtime the `Runner` class simply calls the function object χ times to build up the decomposition. The `Runner` method then also implements both the norm estimation and metropolis methods.

Details of the specific strategies used to build stabilizer state decompositions will be given in the descriptions of simulations in Section 4.2.3.

In their implementation, the DCH and CH classes have the same set of public methods, differing only in their internal representation of the stabilizer state. We formalise this using the notion of 'template' programming [165, 167]. Templates allow the implementation of the simulator to be agnostic to the choice of internal representation. The choice of encoding is made at compile-time, by specifying either the `CHState` or `DCHState` classes.

Parallelization

An important feature of all the routines outline in Algorithms 3–6 is that they each include a step where we operate on every single term in the stabilizer state decomposition independently. In the decomposition routines, each stabilizer state term is built up separately. Similarly, in the output routines we use a 'map-reduce' model, where the same calculation is applied to every state before combining the results at the end. For example, computing a probability amplitude requires summing the value of $\langle \vec{\mathbf{x}} | \phi_i \rangle$ for every state.

These kind of computations are called 'embarrassingly parallelisable', as there is little to no dependency between the tasks, and thus they can be easily sped-up by providing multiple parallel workers. Importantly, these loops are also the only parts of the computation where the complexity scales as $O(\chi)$; other steps, such as gadgetizing a circuit or computing a PBC, are efficiently computable. Thus, these parallelisable loops dominate the runtime, and by Amdahl's law we can significantly reduce the runtime of the programme by adding parallel workers [168].

In contrast to 'data parallelism', such as the SIMD methods discussed in Section 2.3, this kind of computation is called Multiple Instruction Multiple Data (MIMD) computation [118]. MIMD programmes can be further subdivided into 'shared memory' execution, where parallel threads run on a single computer, or 'distributed' execution, where separate processes run independently on multiple processing units.

Shared memory parallelism is the most straightforward to implement. The programme is mainly executed by a single thread, with additional parallel

threads 'forked' from the programme for specific subroutines [169]. In `C++`, this can be implemented using `OpenMP`, which allows parallelising loops and map-reduce operations with single-line annotations [170].

However, the benefits of shared memory parallelism are limited by the kind of hardware available, in particular the maximum memory and number of threads. While this kind of parallelism is sufficient for personal computers, scaling the simulator to large problem sizes requires distributed memory techniques.

We used a 'message passing' model of distributed memory parallelism, where multiple processes each execute a unique copy of the programme, and synchronise and share results through inter-process communication [169]. In particular, we use `Open-MPI`, an open source implementation of the Message Passing Interface standard [171, 172].

We implement a subclass of `Runner`, called `MPIRunner`, for distributed memory computations. On initialization, each process is assigned a 'rank', with the rank-0 process designated the 'master' [172]. All processes run the same setup steps to initialize the simulation, and the 'master' process then splits the decomposition, allocating a unique fraction of states f_i to each of the 'worker' processes. The worker processes then perform computations locally on their share of the decomposition. Initialization is done entirely locally, with the only communication being to pause the programme until all processes have computed their terms [172]. For output variables, processes again apply the map-reduce model locally, before sending their results to the master process which performs a final reduction step [172]. We can also allow for 'hybrid' parallelism, where each distributed process also uses local, shared memory execution to further speed up its part of the simulation task.

Through distributed memory execution, the stabilizer rank simulator can be scaled up to even larger problem sizes. In this thesis, the largest simulation we considered used 32GB of memory, running on UCL's Myriad supercomputing

cluster, but this method could be scaled to even larger instances.

Integration with `Qiskit-Aer`

Building on the `Runner` class outlined above, we were also able to integrate our simulation method with `Qiskit-Aer`, the component of IBMs `Qiskit` QDK that is responsible for classical simulations. Here, we briefly outline the `Qiskit` execution model, and show how our simulator is incorporated with it.

The fundamental data-structure in `Qiskit` is the `Qobj` or 'Quantum Object', which contains information about a quantum programme in the form of the available quantum and classical registers, and the circuits to be run. The `Qobj` is then converted to Javascript Serial Object Notation (JSON), such that it can be transmitted over the internet to the IBM Quantum Experience, or dispatched to the `Aer` suite of classical simulators.

This classical backend also employs a version of the strategy pattern. The `Qobj` is first parsed by a `Controller`, which is responsible for setting up the simulation, including configuring the shared-memory parallel execution, and creating an internal representation of the quantum circuit as an sequence of `Gate` objects. This includes reading configuration options related to the choice of strategy, or else picking a strategy automatically by inspecting the memory requirements for the circuit. The `Controller` class is also responsible for implementing noisy simulations using a stochastic noise model, where additional random gates and measurements are inserted according to a specified noise model. It does this by sampling additional gates, and inserting them into the circuits. The controller then initializes a `State` class for each circuit in the `Qobj`, passing in the details of the circuit and quantum and classical registers.

We integrate the stabilizer rank simulator by creating a custom `State` class. These objects are responsible for parsing the quantum circuit, and maintaining an internal representation of the quantum state called a `qreg` or 'quantum register' object. In our case, the `qreg` object is a version of the `Runner` class. We begin by first iterating through the circuit, checking it contains only gates we now how to decompose, computing the (multiplicative upper-bound) on

the circuit extent ξ, and initializing the `Runner` with $\chi_\epsilon = \lceil \xi \epsilon^{-2} \rceil$ copies of the initial stabilizer state $|\vec{\mathbf{x}}\rangle$.

The simulation strategy then depends on whether the circuit contains intermediate measurements, whether as a result of sampled noise operators or just as part of the circuit to be run. If there are no intermediate measurements, then the simulation is embarrassingly parallel up until the final measurement stage. Thus, we can iterate through the circuit in parallel, building up each term in the decomposition using the sum-over-Cliffords method.

Otherwise, we need to coordinate the simulator at each measurement operation. Thus, we instead build up the circuit one gate at a time. For each gate, we then begin a parallel loop, taking χ_ϵ samples of the corresponding Clifford branches if it is a non-Clifford gate. When we reach a measurement step, we then run the metropolis method to produce a single sample, and apply the corresponding Pauli projector to decomposition, again parallelising over the $\chi\epsilon$ terms. This model is performs less well, as it requires blocking the computation until all parallel workers have finished, and also as there it requires entering and exiting parallel execution multiple times, which has some associated overhead.

The current implementation in `Qiskit-Aer` only uses the metropolis method, as this is the most general method for sampling from the output distribution of the circuit. However, the output distribution of some circuits will not satisfy the irreducibility requirement. We can in practice achieve good performance, passing the benchmark suite of test circuits for `Qiskit`, by re-mixing the metropolis method for each sample. This avoids us becoming stuck in a non-zero amplitude, and returning the same bit-string for 100% of the samples.

Finally, as well as implementing the software for simulating circuits, we also introduce additional wrapper code for automatically switching to the stabilizer rank method based on the memory requirements of the circuit. Circuits too large to simulate with the previous default method will now automatically be

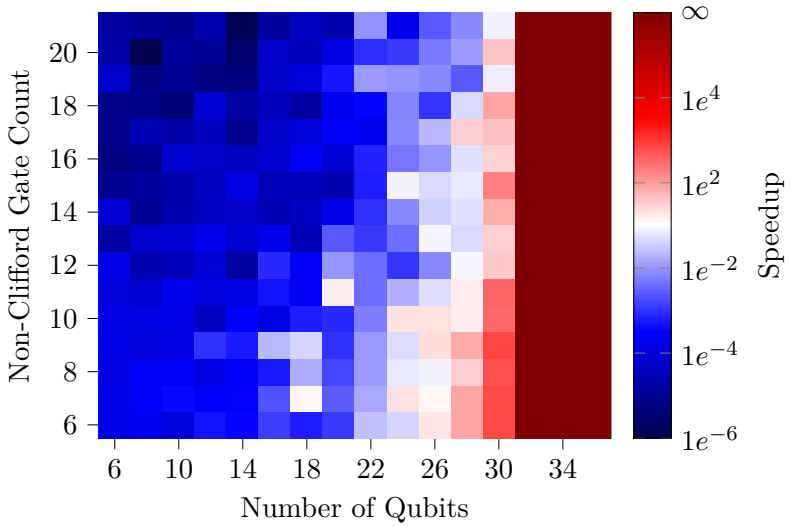


Figure 4.1: Figure comparing the runtime of stabilizer rank-based simulations to the `Qiskit` state-vector simulation for random circuits, using `Qiskit-Aer`. The solid red region corresponds to the regime where quantum circuits required too much memory to simulate with state-vector methods.

run using the stabilizer rank method, provided the memory usage does not exceed the available memory.

This version of the simulator was made public in April, 2019, in the `0.1.0` release of `Qiskit-Aer`. As an example of the capabilities of our simulator, we ran a small random circuit benchmark using both the default `Qasm` simulator of `Qiskit-Aer`, which is based on the state-vector method, and our simulator, which is called `extended_stabilizer` in the `Qiskit-Aer` package. We used the default parameters for the stabilizer rank-based method, which sets $\epsilon = 0.05$ and mixes the Metropolis method for 3000 steps.

We generated random circuits with a fixed number of non-Clifford gates, and simulated these circuits 10 times each with both methods, running on the UCL Myriad cluster with access to 4 2.3GHz processors and 16GB of RAM, and a maximum of 90 minutes of compute time. These conditions are intended to simulate typical personal computers. We then recorded the runtime, and plotted the 'speedup' as the ratio of $\frac{\text{Extended Stabilizer Runtime}}{\text{Qasm Runtime}}$. The results are shown in Figure 4.1. As we can see, the runtime of the stabilizer-rank based methods increases with the number of non-Clifford gates, which we expect as

the extent also increases. However, even for circuits with small extent, below 20 qubits the stabilizer rank method requires significantly increased runtime compared to the state-vector approach.

As the number of qubits continues to increase, the stabilizer rank method becomes increasingly efficient. Above a hard upper limit of 30 qubits, the spatial requirements of the state-vector simulator exceed the available memory, and so the circuits can no longer be run. The stabilizer rank simulator, however, is still capable of running the simulation. it is also important to note that, as expected, the runtime of the stabilizer state method does not increase significantly with the number of qubits. For example, a circuit on 50 qubits and with 20 non-Clifford gates required on average 246 seconds to simulate with the stabilizer-rank method, compared to the 213 seconds required by the state-vector method to simulate a similar circuit on 30 qubits.

4.2.3 Simulations of Quantum Circuits

In this section, we will present results using the stabilizer rank method to simulate three common classes of quantum circuit: 'oracle' or black-box circuits, variational methods, and random circuits.

Hidden Shift Circuits

Oracle-based circuits are a common technique for designing quantum algorithms, encompassing everything from toy methods such as the Deutsch-Jozsa algorithm up to famous algorithms like Grover search and Shor's algorithm [173]. These methods generally involve initializing the quantum state in a superposition of computational basis states, and then applying a black-box unitary O_f that computes some classical function f [30].

The hidden-shift problem is an example of a computational task where quantum algorithms require fewer invocations or queries to the oracle than any classical method [174]. Consider a 'bent' boolean function $f(\vec{\mathbf{x}}) : \mathbb{Z}_2^n \to \pm 1$, which has the property that its Fourier coefficients $\hat{f}(\vec{\mathbf{w}}) = \frac{1}{2^n} \sum_{\vec{\mathbf{x}}} (-1)^{\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + f(\vec{\mathbf{x}})}$ are equal for all n-bit strings w.

For any boolean function, we can also define a shifted function $f_{\vec{\mathbf{s}}}$ as $f_{\vec{\mathbf{s}}}(\vec{\mathbf{x}}) =$

$f(\vec{\mathbf{x}} \oplus \vec{\mathbf{s}})$, where $\vec{\mathbf{s}} \in \mathbb{Z}_2^n$ is a fixed binary string. Finally, we can also define the Fourier transformed dual of the bent function as [174]

$$\tilde{f}(\vec{\mathbf{x}}) = 2^{-n/2} \sum_{\vec{\mathbf{y}} \in \mathbb{Z}_2^n} (-1)^{\vec{\mathbf{x}} \cdot \vec{\mathbf{y}}} f(\vec{\mathbf{y}})$$

Given oracles $O_{f_{\vec{\mathbf{s}}}}$ and $O_{\tilde{f}}$ that will evaluate both the shifted function and its unshifted dual for some input string $\vec{\mathbf{x}}$, it will take a classical method $O(n)$ queries to determine the 'hidden' shift string $\vec{\mathbf{s}}$. However, a quantum algorithm can determine $\vec{\mathbf{s}}$ in just two queries.

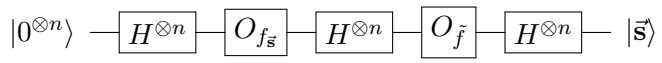


Figure 4.2: Circuit diagram of the quantum method for solving the hidden-shift problem, described in [174].

It is interesting to note that, if we further restrict this problem to only have access to $f_{\vec{\mathbf{s}}}$ and f, and not the dual bent function, there nonetheless exists an alternative quantum algorithm capable of solving for $\vec{\mathbf{s}}$ in $O(n)$ queries. The authors conjecture that in this case a classical method would require an exponential number of queries [174].

The specific class of bent functions considered in [174] are called Majorana-McFarland functions. In practice, we can construct a quantum oracle for random bent functions from this family using a fixed number of CCZ gates. This method was outlined in Appendix F of [49], which used the hidden-shift problem to benchmark the performance of the stabilizer rank simulation method. Because we specify the string s in the construction of the oracle, this method has 'built-in' verification that the simulator is running correctly.

In particular, [49] detail how to construct a bent function starting from a random boolean function $g : \mathbb{Z}_2^{n/2} \to \pm 1$, which they generate using a random sequence of Z and CZ gates and a fixed number of CCZ gates. If we denote

this circuit O_r, then the oracles for the hidden-shift problem are defined as

$$O_{f_{\vec{\mathbf{s}}}} = \left[\left(\prod_{i=1}^{n/2} CZ_{i,i+n/2}\right) I \otimes O_r\right] Z(\vec{\mathbf{s}}) \quad O_{\tilde{f}} = \left(\prod_{i=1}^{n/2} CZ_{i,i+n/2}\right) O_r \otimes I \tag{4.7}$$

For m CCZ gates, the overall circuit thus contains $2m$ non-Clifford gates. In [49], they simulate these circuits by using a gadget for CCZ gates built out of 4 T gates. Here, we use the hidden-shift circuits on 40 qubits as a way to test our results on decomposing alternate Clifford magic states, and the sum-over-Cliffords picture. In particular, we simulate the hidden-shift circuits using four distinct methods

- A gadgetized decomposition, using 4 $|T\rangle$ magic states to synthesis each CCZ gate
- A gadgetized decomposition, using $|CCZ\rangle$ magic states directly
- A sum-over-Cliffords decomposition, using 4 T gates per CCZ gate
- A sum-over-Cliffords decomposition, using CCZ gates

This allows us to directly compare the sum-over-Cliffords and gadgetized methods, and to compare direct decompositions with Clifford+T synthesis.

The simulation was developed in collaboration with Mark Howard, based on the previous simulations developed by David Gosset in [49]. The setup for the simulation, including constructing the oracle circuits, and constructing the PBC projectors in the gadgetized method, are implemented in `MATLAB`.

As previously stated, sampling from n output bits with the norm estimation routine has a runtime that scales as $O\left(\chi_\epsilon n^6\right)$. To circumvent this, we exploit the fact that we can clasically cache the state of the simulation before measurement, and that the output state of the simulation is an approximation of single output string $|s\rangle$, and learn the string $\vec{\mathbf{s}}$ by sampling single qubits a time. An example of this is shown in Figure 4.3. Overall, this method takes time $\left(\chi_\epsilon n^4\right)$ to sample from all n bits.

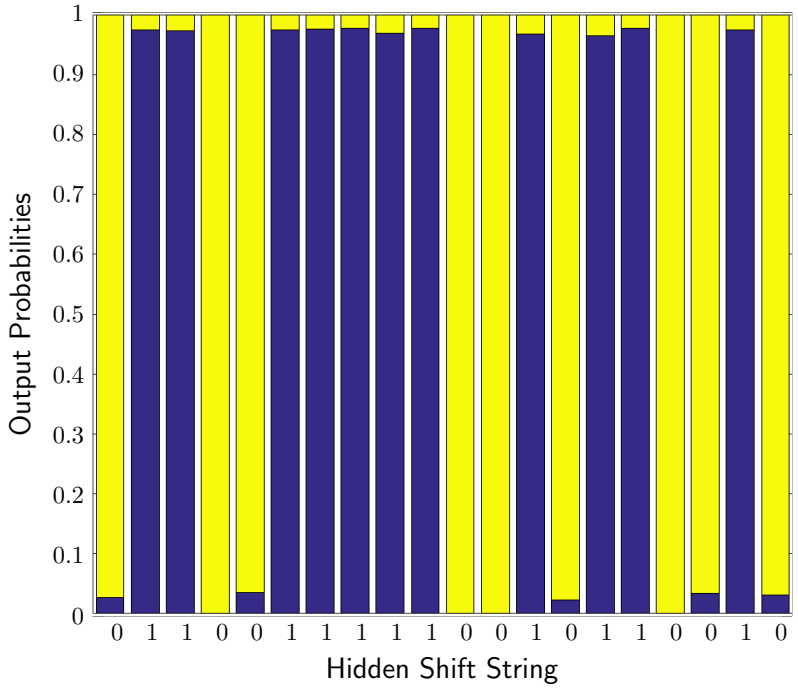


Figure 4.3: Figure showing the probability $P(x_i = 1)$ for 20 bits, obtained using the sum-over-Cliffords methods and synthesizing the CCZ gate with 4 T gates. Each output probability is computed individually.

To cache the decomposition between each norm estimation step, we store the choice of subspace or Clifford branches in MATLAB. We then make use of the `MEX` API to pass this data, and the Pauli projectors to be applied, to the `C++` simulator. This computes and returns the probability $p_{x_i=1}$, and we then sample bits by generating uniform random numbers $r \in [0,1)$ and returning 1 iff $r < p_{x_i=1}$.

In the gadgetized case, the number of terms in the decomposition depends on the stabilizer fidelity and the target infidelity, which we will denote here as Δ. Using the stabilizer fidelity of the $|T\rangle$ and $|CCZ\rangle$, then for a bent function using m CCZ gates the size of the decomposition χ scales as

$$\begin{aligned} F(T) &\approx 0.853 \qquad \chi = \lfloor \tfrac{4F(T)^{-8m}}{\Delta} \rfloor \approx \lfloor 4\left(\tfrac{3.57^m}{\Delta}\right) \rfloor \\ F(CCZ) &\approx \tfrac{9}{16} \quad \chi = \lfloor \tfrac{4F(CCZ)^{-2m}}{\Delta} \rfloor \approx \lfloor 4\left(\tfrac{3.16^m}{\Delta}\right) \rfloor \end{aligned} \tag{4.8}$$

Similarly, for the sum-over-Cliffords method, the number of terms is given by the stabilizer extent, and the target error ϵ. For m CCZ gates, the number of

terms is given by

$$\begin{aligned} \xi(T) &\approx 1.17 \qquad \chi = \lceil 1.17^{8m}\epsilon^{-2} \rceil = \lceil 3.57^{m}\epsilon^{-2} \rceil \\ \xi(CCZ) &= \tfrac{16}{9} \quad \chi = \lceil 1.78^{2m}\epsilon^{-2} \rceil = \lceil 3.16^{m}\epsilon^{-2} \rceil \end{aligned} \tag{4.9}$$

Recall that for Clifford magic states, the stabilizer fidelity and stabilizer extent coincide, which explains the correspondence in the scaling between the random codes and sum-over-Cliffords method. In all the simulations, we set $\Delta = \epsilon = 0.3$. Due to platform limitations, the `C++` component was executed serially, though decompositions were built in parallel using `MATLAB`'s built in parallel execution. All simulations were run on a dual-core 1.9GHz Intel Xeon, with 32GB of RAM. The results are shown in Figure 4.4.

QAOA

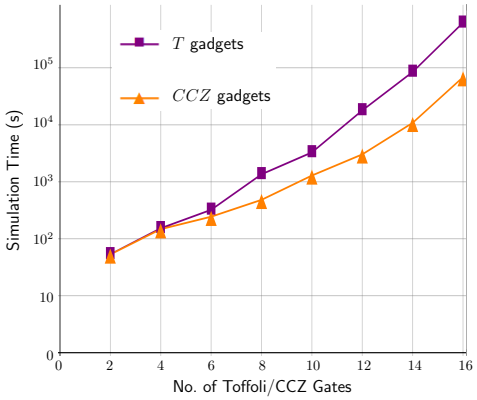
As mentioned in Section 4.1, current NISQ computers lack full error correction and thus accumulate noise as the circuit depth increases. Thus, there is a great deal of interest in relatively low-depth quantum algorithms that can run on NISQ devices. The main class of these algorithms are 'variational methods', hybrid quantum-classical algorithms with applications in optimization and quantum computational chemistry [12].

In general, variational methods use a simple processing scheme where the quantum computer is used to prepare an 'ansatz' state using a low-depth circuit. The gates in the circuit are typically parameterized. We then perform a series of measurements, and these outcomes are post-processed by a classical algorithm. This can be iterated, where the parameters of the ansatz state are updated by the classical algorithm, to converge to a heuristic solution [12, 100].

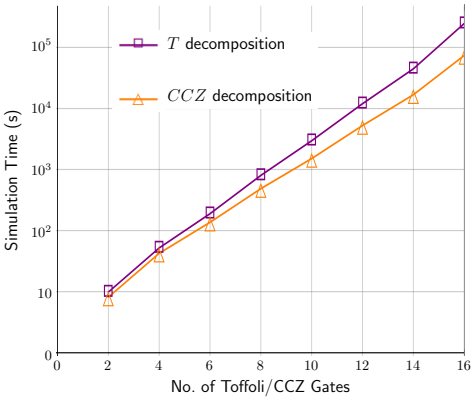
The Quantum Approximate Optimization Algorithm (QAOA) is an example of a variational method, applied to classical combinatorial optimization problems [175]. These kind of optimization problems are usually specified by a number of boolean clauses, and we are interested in optimizing a function 'satisfied clauses'

$$C(\vec{\mathbf{z}}) = \sum_{\alpha} C_{\alpha}(\vec{\mathbf{z}}),$$

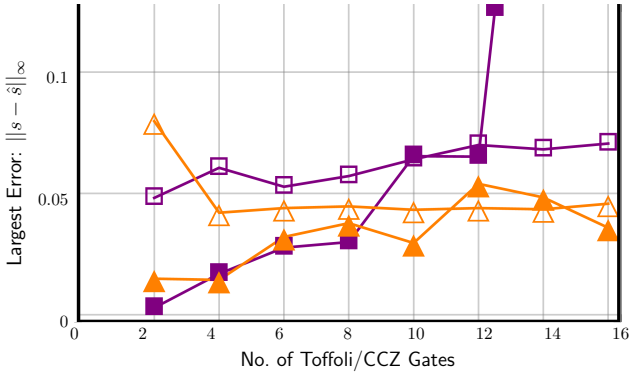
Figure 4.4: Figures demonstrating the performance of the stabilizer rank method on simulating the hidden shift problem, using 4 methods of building the stabilizer state decomposition. Figures originally created for [109]



(a) Runtime of the hidden shift simulation as a function of the number of CCZ gates, using the random codes method to build decompositions, using Clifford+T synthesis and direct decomposition.



(b) Runtime of the hidden shift simulation as a function of the number of CCZ gates, using the sum-over-Cliffords method to build decompositions, comparing Clifford+T synthesis with CCZ gates.



(c) The maximum observed approximation error in single-qubit probabilities over all bits in the hidden-shift string. The colour and markers correspond to the timing plots above. Not shown are two data-points in the T random-code methods, for 14 and 16 Toffoli gates, with large errors 0.304 and 0.512 respectively.

where each sub-clause C_α acts on a subset of bits from the full n-bit string $\vec{\mathbf{z}}$, and evaluates to either $\{0,1\}$ or ± 1 depending on the definition of the problem [175]. A clause is said to be 'satisfied' if it evaluates to 1. Examples of combinatorial optimization problems include MaxSat, where we are tasked with finding the string $\vec{\mathbf{z}}$ that satisfies the most clauses simultaneously.

Given a clause C_α, we can define an operator $\hat{C}_\alpha$ by replacing the bits z_{α_i} in the clause with Pauli Z operators, and in turn we can define $\hat{C} = \sum_\alpha \hat{C}_\alpha$ [175]. Different computational basis strings are eigenstates of this operator, such that

$$\hat{C}\,|\vec{\mathbf{z}}\rangle = C\,(\vec{\mathbf{z}})\,|\vec{\mathbf{z}}\rangle\,.$$

The QAOA algorithm proceeds by preparing an ansatz state parameterized by $2p$ angles β_i and γ_i, for some fixed value of p. The system is initialized in the ground state $|g\rangle$ of the operator $\hat{C}$, which depends on the definition of the problem but is typically a trivial assignment such as $\left|0^{\otimes n}\right\rangle$ or $\left|+^{\otimes n}\right\rangle$ [175, 176]. We then apply p rounds of a pair of parameterised rotation operators

$$U_C\,(\gamma_i) = \mathrm{e}^{-\mathrm{i}\gamma\hat{C}} \quad U_B\,(\beta_i) = \textstyle\prod_j \mathrm{e}^{\mathrm{i}\beta_i X(\vec{\mathbf{e}}_j)}\ .$$

For each paramaterised sate

$$\left|\psi_{\vec{\gamma},\vec{\beta}}\right\rangle = \prod_{i=1}^{\mathbf{p}} \left(U_B\,(\beta_i)\,U_C\,(\gamma_i)\right)|g\rangle$$

has been prepared, we then perform measurements to determine the expectation value of the $\hat{C}$ operator

$$E_{\vec{\gamma},\vec{\beta}} = \left\langle \psi_{\vec{\gamma},\vec{\beta}}\middle|\hat{C}\middle|\psi_{\vec{\gamma},\vec{\beta}}\right\rangle.$$

Importantly, it can be shown that as $p \to \infty$, the maximum of this expectation value corresponds to the maximum of $C\,(\vec{\mathbf{z}})$ [175]. The authors further show that even with $p = 1$, reasonable results that some classical strategies can be obtained [175, 176].

We consider the application of QAOA to a combinatorial optimization problem called MaxE3Lin2, where QAOA has been shown to outperform random classical guesses at $p = 1$ [176]. In particular, we consider randomly generated instances of MaxE3Lin2 with 50 variables and 66 clauses, requiring 50 qubits and 66 Pauli Z rotations.

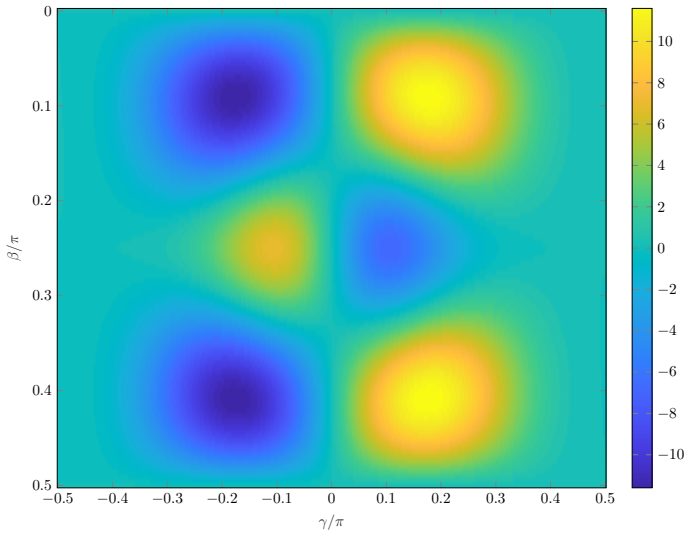


Figure 4.5: Heat-map showing the expectation value $E_{\beta,\gamma}$ for the simulated instance of MaxE3Lin2, generated using the method of [56], implemented in `MATLAB`.

The goal of MaxE3Lin2 is to maximize an objective-function

$$C(\vec{\mathbf{z}}) = \frac{1}{2} \sum_{1 \leq u < v < w \leq n} d_{uvw} z_u z_v z_w$$

where each clause acts on 3 variables [176, 109], and the coefficients $d_{uvw} = \{0, \pm 1\}$. The number of clauses is given by the number of non-zero coefficients. When generating the problems, restrict ourselves to instances with fixed degree 4, such that each qubit appears in at most 4 terms. We then prepare a state

$$\left|\psi_{\gamma,\beta}\right\rangle = U_B(\beta) U_C(\gamma) \left|+^{\otimes n}\right\rangle$$

with two parameters [176].

Classical preprocessing methods exist for estimating $E_{\gamma,\beta}$, which can be used to speed-up the classical step of the variational algorithm. In particular, we use a method that allows the expectation variables of a sparse Hamiltonian with 'computationally tractable' states, states which can be efficiently specified in the computational basis [56]. This allows us to approximately compute

$$\langle \psi_\gamma | U_B(\beta)^\dagger \hat{C} U_B(\beta) | \psi_\gamma \rangle$$

with additive error ϵ in time $O(m^4\epsilon^{-2})$ [109]. Figure 4.5 plots the estimates of $E_{\gamma,\beta}$ for a particular instance of MaxE3Lin2.

In their original paper, Farhi et al. fix $\beta = \pi/4$. This has the advantage that the rotation U_B becomes a Clifford operator

$$\mathrm{e}^{-\mathrm{i}\pi/4X(\vec{\mathbf{e}}_i)} = H(\vec{\mathbf{e}}_i)\, S(\vec{\mathbf{e}}_i)\, H(\vec{\mathbf{e}}_i),$$

meaning all non-Clifford terms arise from the Z rotations. We can also see from Figure 4.5 that the line from $\beta = \pi/4$ passes through a local minima and maxima of $C(\vec{\mathbf{z}})$. Thus, in our simulation, we also fix $\beta = \pi/4$. The cost function is antisymmetric about $\gamma = 0$, and so we sweep γ from 0 to π.

Each rotation in U_C has a sum-over-Cliffords expansion [109]

$$\mathrm{e}^{-\mathrm{i}\frac{\gamma}{2}d_{uvw}Z_uZ_vZ_w} = \begin{cases} \alpha I + \beta CNOT_{u,v}CZ_{v,w}S_vS_wCNOT_{u,v} & d_{uvw} = 1 \\ \alpha I + \beta \mathrm{i}CNOT_{u,v}CZ_{v,w}S_v^\dagger S_w^\dagger CNOT_{u,v} & d_{uvw} = -1 \end{cases}$$

where the coefficients α and β are the phase terms associated with each branch

$$\begin{array}{ll} b_0 = \mathrm{e}^{\mathrm{i}\gamma} - \mathrm{i} & \alpha = \frac{b_0}{|b_0|} \\ b_1 = 1 - \mathrm{e}^{\mathrm{i}\gamma} & \beta = \frac{b_1}{|b_1|} \end{array}.$$

For each value of γ, we build the corresponding stabilizer state decomposition with

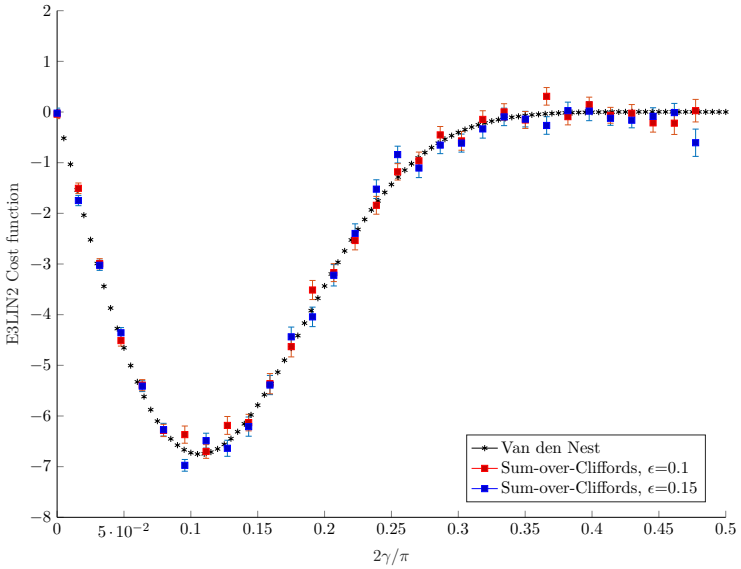
$$\chi = \lceil \xi(\gamma)\, \epsilon^{-2} \rceil = \lceil (|b_0| + |b_1|)^{2m} \epsilon^{-2} \rceil$$

terms for m clauses. We then run the Metropolis method to take 40000 samples from the output distribution of the state $|\psi_\gamma\rangle$, and compute an estimate of the expectation value

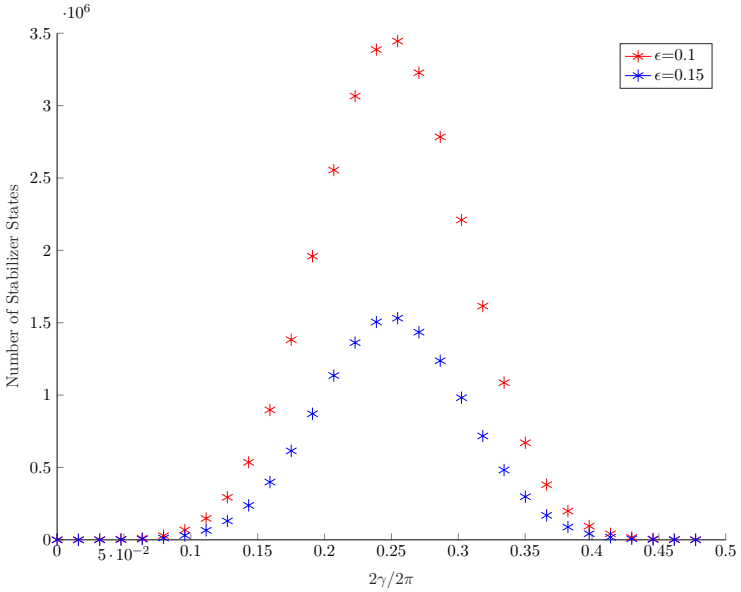
$$E_{sim}(\gamma) \frac{1}{40000} \sum_{s=1}^{40000} C(z_s). \tag{4.10}$$

The simulations were run on the UCL Legion supercomputing cluster, running on Dell C6100 compute nodes, using a shared-memory parallelism model with 12 parallel threads and 24GB of RAM. We ran these methods with $\epsilon = 0.1$

and 0.15, and compare our estimates to the results from the heuristic method of [56]. Results are shown in Figure 4.6.



(a) Plot comparing the estimates of E_γ obtained using the methods of [56], with the estimates obtained using the sum-over-Cliffords simulator and sampling from the output distribution with the Metropolis method.



(b) Plot showing how the stabilizer rank of the decomposition $\xi(\gamma)\epsilon^{-2}$, as a function of the QAOA parameter γ.

Figure 4.6: Graphs showing the results of the sum-over-Cliffords simulations of a 50-qubit instance of MaxE3Lin2 with 66 clauses. We use the same plotting code as in [109], where the error in $E_{sim}(\gamma)$ is computed using the methods of [177].

Random Circuit Models

The final simulation task we consider are random circuit models. It is well known that the output distribution of Haar random unitaries satisfy a property called anti-concentration [48], and that random quantum circuits also satisfy this property for sufficient depth [99, 178, 92]. Random circuit models like this are not computationally useful but, as discussed in Section 1.2.4, satisfy complexity theoretic conjectures that make them hard to sample from using classical simulation.

Here, we consider a class of random circuits introduced by the Google AI group, referred to as either 'Google Circuits' or as 'Qubit Speckle' [99, 179]. Circuits are built up using alternating layers of entangling two-qubit gates, and randomly placed single qubit gates, either Clifford rotations $\mathrm{e}^{-i\frac{\pi}{4}X} = HSH$, $\mathrm{e}^{-\mathrm{i}\frac{\pi}{4}Y} = S^\dagger HSHS$, or the T gate [180]. These choices are designed to try and frustrate commuting gates through the circuit to, for example, combine T rotations and cancel them to reduce the overall depth of the circuit [92]. This gate-set also has the property that it forms an approximate unitary t-design, and thus that the problem of sampling from their output distributions satisfies both the average-case hardness [181, 182] and anti-concentration [48, 183] criteria sought for a test of quantum supremacy.

These circuits have the property that with increasing depth, their output distribution converges to the Porter-Thomas distribution [180]. Based on this property, the authors introduce a metric called the cross-entropy difference that quantifies the accuracy of a given sample of m bitstrings from the output distribution of a random circuit [180].

$$\alpha = \log 2^n + \gamma - \frac{1}{m}\sum_{j=1}^{n} \log\left(\frac{1}{p_U\left(\vec{\mathbf{x}}_j\right)}\right) \tag{4.11}$$

where γ is the Euler-Mascheroni constant, and $p_U\left(\vec{\mathbf{x}_\mathbf{j}}\right)$ is the probability of the output string string $\vec{\mathbf{x}}_j$ [180]. This quantity has the property that $\alpha = 1$ for an ideal sample, and $\alpha = 0$ if the sample is from a uniform distribution.

Recalling the discussion Section 1.2.4, we can thus define two distinct classical tasks as part of a quantum supremacy test using random circuits: 'cross-entropy benchmarking' (XEB), where we compute the probabilities p_U, and 'heavy output generation' (HOG), sampling from the output distribution of the circuit U [99, 102].

The depth of the circuit required to achieve $\alpha = 1$ depends on the locality restrictions of two-qubit gates. For example, if we can perform two-qubit gates between arbitrary qubits, then the distribution will converge to Porter-Thomas with a depth $O(\log n)$ in the number of qubits [178, 180]. If instead we are limited to two-qubit interactions only between neighbouring qubits on a 2D lattice, then the depth required scales as $O(\sqrt{n})$ [184].

As discussed in Section 4.1, large scale simulations of Google circuits have focused on lattices with 2D connectivity, a restriction which is driven by comparisons with current and future quantum hardware which also uses qubits connected on a 2D grid. The simulation techniques employed also exploit this locality restriction in their design [162, 164, 139, 102]. These simulators are capable of both the XEB and HOG tasks [102].

Our simulation method, in contrast, makes no restrictions on qubit connectivity in its design. Here, we will explore the feasibility of using the stabilizer rank method for HOG. We introduce an extension of the Google circuits to different connectivities, and examine the stabilizer extent and simulation runtime as a function of the circuit depth.

Google's random circuits are constructed with the following method

1. Initialize the system in the $\left|+^{\otimes n}\right\rangle$ state.
2. Apply CZ gates to a subset of qubits, following a 'CZ Schema'.
3. Apply single-qubit gates from the set $\{\mathrm{e}^{-i\frac{\pi}{4}X}, \mathrm{e}^{-i\frac{\pi}{4}Y}, T\}$, according to one of two rules.
4. Repeat steps 2 and 3 $d-1$ more times for depth d.

5. Apply a Hadamard gate to each qubit.

6. Sample in the computational basis.

The 'CZ Schema' defines how we place CZ gates. A limitation of current quantum hardware is the inability to reliably apply CZ gates on neighbouring qubits [180, 140]. Thus, for each layer of the circuit, we apply a pattern of CZ gates obeying this hardware restriction, and such that for sufficient depth d every qubit is involved it at least one CZ gate. The authors describe CZ patterns for 2D lattices [140, 180], which are made up of 8 layers. For each time-step l in the random circuit, the authors use the CZ pattern of layer l mod 8.

We can extend these schema to arbitrary-dimensional connectivity using the method outlined in Algorithm 7. For each layer, we iterate along one dimension of the lattice, greedily adding edges. Each time we add an edge, we drop those

Algorithm 7 Pseudo-code description of a greedy algorithm for constructing a 'CZ Schema', covering every edge in a d dimensional square lattice or 'grid' graph. Each axis of the lattice d_i has $|d_i|$ points.

Require: d-dimensional square lattice graph
  $G = \{V = \{v_i = (c_{1,i}, \ldots, c_{d,i})\}, E = \{v_i, v_j\}\}$
Require: $N(v)$, the neighbourhood of $v \in G$.
  $\mathcal{E} \leftarrow \emptyset$ ▷ Set of visited edges.
  $S \leftarrow \{\}$ ▷ Initialize an empty array S.
  while $\mathcal{E} \neq E$ **do**
    $H = (V', E' = E \setminus \mathcal{E}) \leq G$, ▷ Graph minor from deleting $\mathcal{E}$
    for $i \in \{1, \ldots, d\}$ **do**
      $\mathcal{L} \leftarrow \emptyset$ ▷ New layer in the CZ schema
      for $j \in \{1, \ldots |d_i|\}$ **do**
        for $v_k \in V' : c_{i,k} = j$ **do**
          if $\{v_k, v_{k'} : c_{i,k'} = j+1\} \in E'$ **then**
            $\mathcal{L} \leftarrow \mathcal{L} \cap \{v_k, v_{k'} : c_{i,k} = j+1\}$
            $W \leftarrow \{v_k, v_{k'}, N(v_k), N(v_{k'})\}$ ▷ Set of vertices to exclude.
            $H \leftarrow H' = (V', E') \leq H$ Minor induced by deleting vertices W.
          end if
        end for
      end for
      $\mathcal{E} \leftarrow \mathcal{E} \cap \mathcal{L}$
      $S \leftarrow S + \{\mathcal{L}\}$ ▷ Append layer $\mathcal{L}$ to the schema.
    end for
  end while

vertices and their neighbourhood from being involved in any other CZ gate in that layer. Applying this to a 2D grid gives the same pattern described in [140]. Examples of 1, and 2D CZ schema are given in Figure 4.7. For all-to-all connectivity, we instead apply $\frac{fn}{2}$ CZ gates to random pairs of qubits, such that we involve some fraction f of qubits in each layer of the circuit.

Previous work has described two distinct rules for placing single-qubit gates. In the first scheme, we place one of the three gates with equal probability on any qubit that was acted on by a CZ gate in the previous layer [180]. However, with this strategy can produce configurations like $TCZT$, which can be rearranged to cancel the T gates as diagonal unitaries commute. Thus, an updated rule was proposed. The first single-qubit gate applied must always be a T gate. Then, we apply either $\mathrm{e}^{-\mathrm{i}\frac{\pi}{4}X}$ or $\mathrm{e}^{-\mathrm{i}\frac{\pi}{4}Y}$ to qubits acted on by CZ in the previous layer, and T if a qubit was acted on by a one of these two rotations on the previous layer [140]. We use the second rule to place single-qubit gates, except in the 1D case as otherwise this rule will never place more than a single T gate on each qubit. An example random circuit for a 1D lattice is shown in Figure 4.8.

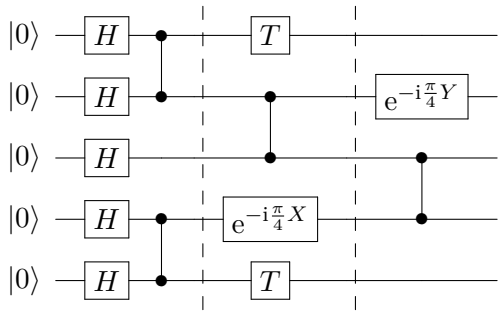
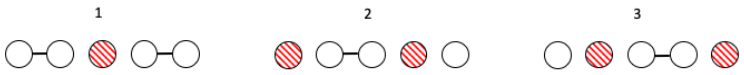


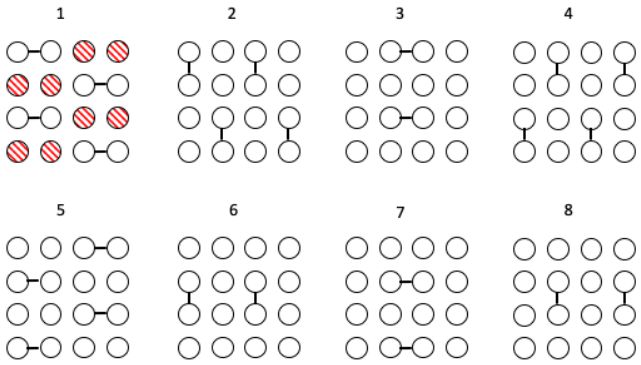
Figure 4.8: Circuit diagram showing 3 layers of the random circuit applied to a 5-qubit, 1D lattice. Each dashed line represents the end of a single layer.

We examined the performance and resource requirements of the stabilizer rank method for HOG, sampling 1000000 amplitudes in the computational basis from the output distribution of Google circuits. We first consider how the runtime and requirements scale as a function of the circuit depth and the precision ϵ, for a 4×5 qubit grid, for $d \in [10, 20]$. We then pick a precision

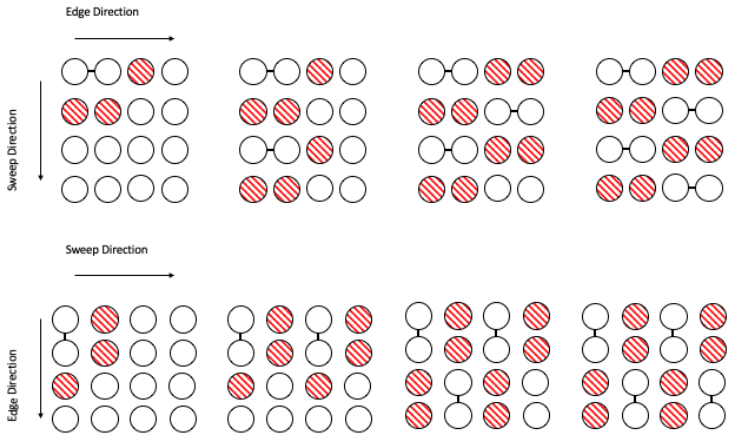
Figure 4.7: Examples of CZ schema for different dimensionalities of square qubit lattice. Connected qubits are subject to a CZ gate in that layer, and each layer appears sequentially according to the numbering. In some layers, we have marked qubits excluded by the neighbouring CZ restriction in red.



(a) CZ schema for a 5-qubit 1D lattice. Here, we highlight in each layer the qubits excluded by the neighbouring CZ restriction.



(b) CZ schema for a 4×4 qubit grid. As described in Algorithm 7, we apply CZ gates along alternating the dimensions of the grid in each layer.

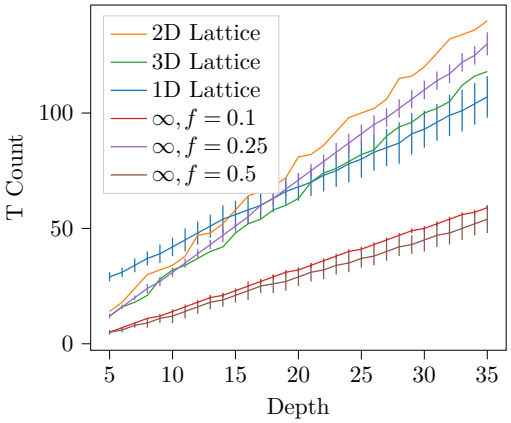


(c) Example showing how two layers of 2D schema are built up step-by-step using Algorithm 7, read left-to-right. Striped qubits indicated those excluded by the neighbouring CZ restriction.

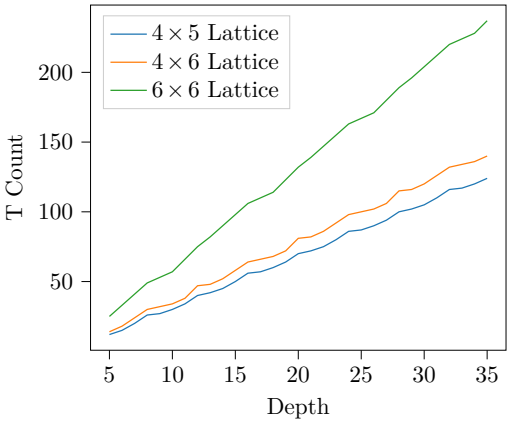
value $\epsilon = 0.995$, comparable to that used in [140], and explore how the runtime varies with depth for each connectivity pattern, for depth $d \in [15, 20]$.

Circuits were generated using custom `C++` code, and interfaced with Python using `Pybind`. The resulting circuits were then convert to a `Qobj`, and run with our simulator as part of `Qiskit-Aer`. All simulations were run on UCL's Myriad computing cluster, with 2.3GHz processors. The 4×5 qubit simulations were run with 36 parallel workers and 32GB of RAM. Due to scheduling restrictions on the cluster, the simulations for differing connectivities were run with 28 parallel workers, and 28GB of RAM.

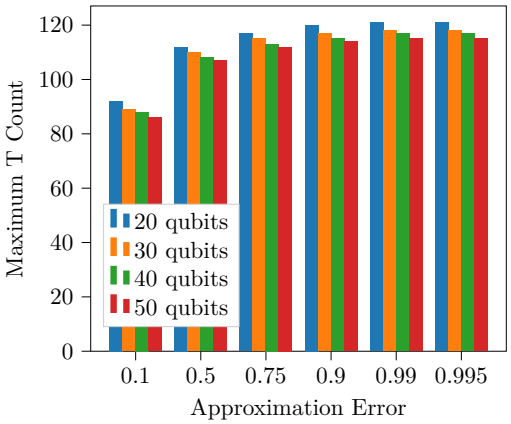
Figure 4.9: Resource Analysis of Google circuits on a 20 qubit lattice.



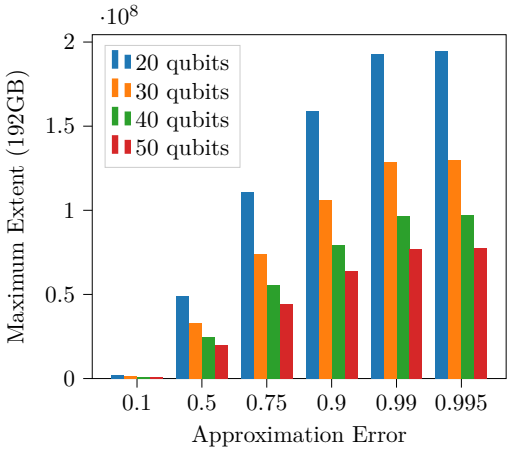
(a) Average T count as a function of depth for a 24 qubit lattice with different connectivity patterns.



(b) T count as a function of depth for different sizes of 2D qubit lattices.



(c) Plot showing the maximum number of T gates that can be simulated for different system sizes, assuming access to 192GB of RAM.



(d) Plot showing the maximum circuit extent that can be simulated for different system sizes, assuming access to 192GB of RAM.

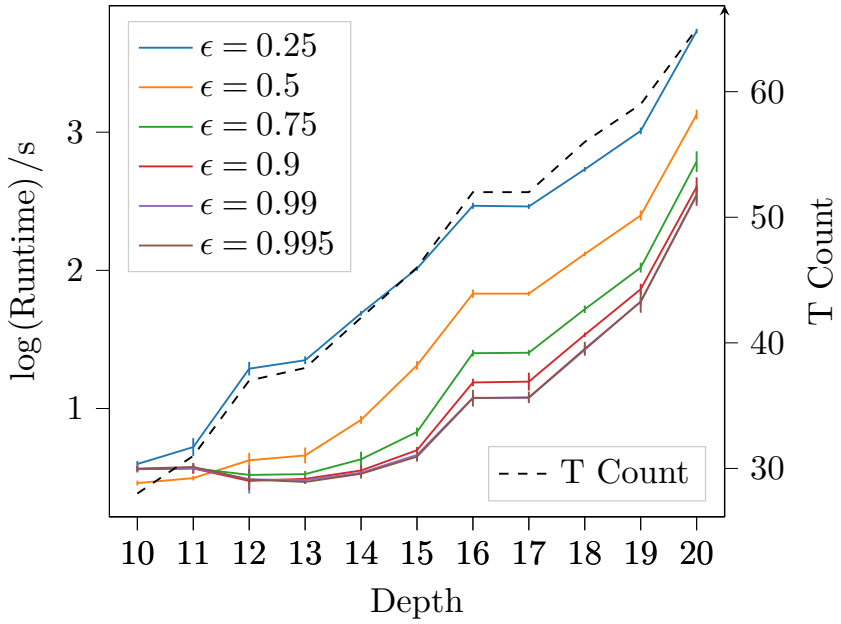


Figure 4.10: Average time required in seconds to take 1000000 samples from the output distribution of a Google circuit, for different values of the precision ϵ. Also shown is the corresponding T count of the circuit.

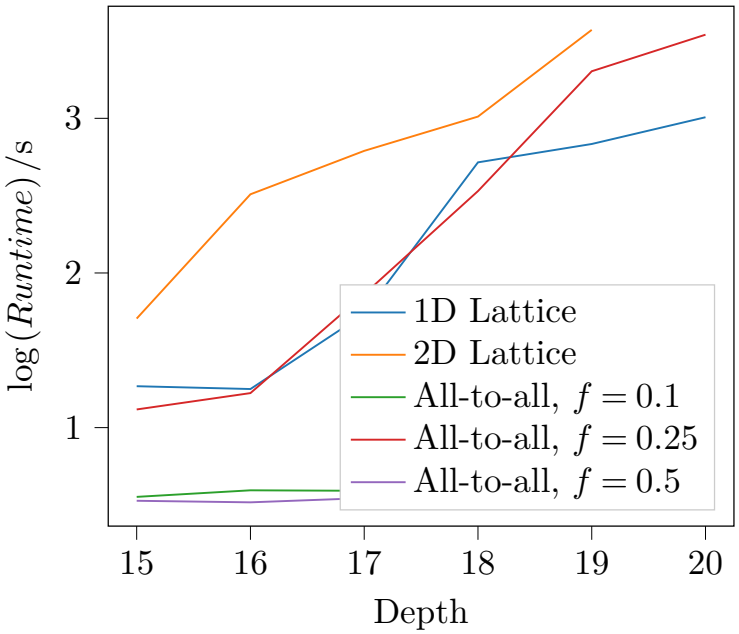


Figure 4.11: Average time required in seconds to take 1000000 samples from the output distribution of 1, 2 and 3D versions of a Google circuit. We also consider circuits with All-to-All connectivity, for different fractions of CZ gates f. No data is available for the 2D lattice with depth 20 due to memory requirements.

4.3 Discussion

We have presented in this chapter a broad range of simulation results, using the techniques introduced in Section 4.2.1 and 4.2.2. These represent the state of the art in simulating quantum circuits using stabilizer state decompositions.

As a previously studied benchmark, and as a method with in-built validation, the hidden-shift circuits offer the clearest method to compare our work again the previous results in [49]. Looking at Figure 4.4a, it is clear to see the impact of direct decompositions of non-Clifford gates. The small difference in stabilizer fidelity is blown-up by the exponential scaling, resulting in a 7-fold reduction in the number of terms in the decomposition of 16 CCZ gates, and a similar reduction in the overall runtime of the simulation.

As discussed, decompositions built using the sum-over-Cliffords method require the same number of terms for the T and CCZ gates. However, we might expect it performs better than the gadgetized method when the number of magic states required would be larger than the initial quantum register. In fact, we observe a decrease in runtime using the sum-over-Cliffords method across all values of $\#CCZ$. We also similarly observe smaller gradient in the runtime of the CCZ decompositions

The increased performance of the sum-over-Cliffords method is likely due to the greatly simplified preprocessing required. While building the Pauli projector for a PBC is efficient, $O(n^2)$ run-times can amount to a significant overhead in practice. This improved performance is why the sum-over-Cliffords strategy was also employed for the random circuit simulations, despite these also being based only on T gates.

Finally, it is interesting to compare the maximum observed error in sampling from the output distributions of circuits, compared to the theoretical bounds. In almost all cases, the decompositions achieved maximum error that was well below the bound of 0.3 used when fixing the size of the decomposition. The two exceptions were the decompositions built using the random codes method

and T-gate gadgets, with 52 and 64 T-gates respectively. This suggests that, despite the probability of picking a valid subspace being large, nonetheless the random subspace method can fail at these problem sizes. In contrast, the smaller decompositions used for the CCZ states, and the sum-over-Clifford methods, all showed relatively consistent error performance across the parameter range.

The failures of the random codes method suggest that for large scale simulations, explicit calculation of the fidelity of the approximation would be necessary. Interestingly, in the sum-over-Cliffords case, a consequence of the 'tail bound' proven in Lemma 7 of [109] is that whenever $F(|\psi\rangle)$ falls off exponentially with the number of copies,

$$\left\| |\psi\rangle - \left|\tilde{\psi}\right\rangle \right\|^2 \leq \left\langle \tilde{\psi} \middle| \tilde{\psi} \right\rangle - 1 + \epsilon^2. \tag{4.12}$$

Thus, using norm estimation to compute $\left\langle \tilde{\psi} \middle| \tilde{\psi} \right\rangle$, we can quickly obtain an estimate of the error achieved. Thus, as we move to larger simulations, we can adapt the sum-over-Cliffords simulation method to get better guarantees of the achieved error rate.

4.3.1 Simulating NISQ Circuits

Looking at the plots in Figure 4.6a, we can see that the estimates E_γ obtained using the sum-over-Cliffords method agree closely with the estimates obtained using the methods of [56], across the entire parameter range. This serves as a useful validation of the Metropolis method of sampling from the output distribution of the circuit. We can also compare the relative performance of the estimate for both values of the precision ϵ. Importantly, it must be noted that our reference value is itself a classical estimate. Thus, these results cannot be used to ask about the overall error performance of the simulation. However, it is interesting as a means of comparing between the two runs. In general, as expected, we observed the $\epsilon = 0.1$ results agree better with the classical estimate, with an average error $|E_{sim} - E| = 0.125$, smaller than the 0.1505 achieved with $\epsilon = 0.15$. However, this comes with an associated 2.25-fold in-

crease in the number of terms in the decomposition, as shown in Figure 4.6b. Due to the aforementioned overhead associated with shared-memory parallelism, this translates to a roughly 2.3-fold increase in computational runtime.

In [109], we also presented simulation results for MaxE3Lin2, with $\epsilon = 0.15$. These simulations were run using only a single thread, and with access to just 8GB of RAM. Overall, it took several days to generate the data required. In contrast however, the data shown here was obtained running with 12 parallel workers and up to 16GB of memory. Taking advantage of this parallelization, the runtime of the simulation is significantly reduced. For example, computing $E(\frac{\pi}{8})$, which has the largest stabilizer rank across the parameter range, with $\epsilon = 0.15$ required 270 minutes when running serially. With access to 12 parallel threads, the same simulation could be completed in just 33 minutes. In general, we achieved a roughly 8-fold speedup through parallelization. As previously discussed, the computational overhead associated with entering and leaving parallel regions, and portions that cannot be parallelised such as reading and writing files, account for this discrepancy between the number of parallel threads and the achieved performance boost [168].

These QAOA simulations represent a significant increase in the types of circuits that can be simulated compared to previous sate of the art. As discussed, using T gate synthesis, the number of gates required per rotation $\mathrm{e}^{-\mathrm{i}\frac{\gamma}{2}Z}$ could vary from 1 when $\gamma = \pi/4$, to 100 for $\gamma = 1e^{-8}$. Given that a circuit with as T-count of 120 requires ~ 370GB of memory to simulate, these QAOA circuits are only accessible to the sum-over-Cliffords method.

In fact, in general the memory requirements of the stabilizer rank method is significantly reduced compared to other methods. For example, a 100 qubit simulation of the MaxCut problem with `qTorch` requires 96GB of RAM [185]. Using the sum-over-Cliffords method with $\epsilon = 0.15$, we can simulate similar 50-qubit circuits on a laptop computer with just 8GB of RAM, increasing to just 32GB if we extended these simulations to 100 qubits.

In general, circuits targeting NISQ architectures make a good candidate for simulating with the stabilizer rank methods. These circuits are typically limited in depth and in the number of qubits, meaning the problem sizes stay within ranges accessible to our simulator.

Another important aspect of NISQ devices is that noise in the circuit increases with the depth, due to accumulation of individual gate errors. While our simulation method as described does not account for noise directly, we can accordingly relax the target error rate ϵ, which helps to reduce the simulation overhead as the number of qubits or the depth of the circuit grows.

Recent work has questioned the computational advantage offered by the QAOA algorithm, even when accounting for extending the algorithm beyond $p = 1$ repetitions, by demonstrating classical algorithms which show similar performance [186]. This, coupled with the relatively accessibility of QAOA circuits to our sum-over-Cliffords approach, highlight the importance of considering classical techniques when developing variational quantum algorithms.

One example application could be examining the stabilizer extent of different families of ansatz states could potentially be used to rule out classically accessible parameter ranges. However, if we return to the instance of MaxE3Lin2 considered here, then the parameter value that maximises the expectation, $\frac{\gamma}{\pi} \sim 0.1$, is in fact not the ansatz with maximal extent. This could be taken as an indication of the limitations of QAOA over classical methods, but also shows that large extent does not directly correlate to 'computationally interesting'.

4.3.2 Simulating Random Circuits

To simulate the Google circuits here, we used the most straightforward approach, decomposing each T gate with the sum-over-Cliffords method using the version of the simulator available in `Qiskit-Aer`. Unfortunately, as shown in Figure 4.9, the T-count of these circuits grows rapidly in both the depth and the size of the system. This presents a significant limit on the kind of parameter ranges we can explore. While they incorrectly claimed that stabilizer rank necessarily doubles with each non-Clifford gate, the authors of [102] nonethe-

less note that the large non-Clifford gate counts in Google circuits might make them intractable to the stabilizer rank method.

Focusing on Figures 4.9c and 4.9d, we can see that the number of qubits in the system only linearly impacts the maximum extent. This is a consequence of the stabilizer state representations developed in Chapter 2. As we pack our representations into 64-bit integers, up to $n = 64$ qubits we see only a linear increase in the spatial complexity, as we require $O(n)$ integers to encode each state. In turn, there is a quadratic dependence on the precision in the decomposition, which we expect from Equation 3.35.

As we are focusing on large NISQ circuits, the fidelity of an experimental realization can be very low as the circuit depth and system size increase [140, 102], meaning that precision values of $\epsilon = 0.995$ are potentially acceptable. These precisions can keep the memory requirements, and correspondingly the runtime, reasonably small. For example, a circuit with T-count 100 requires just 9GB of memory to simulate at this precision. They do not, however, prevent the eventual exponential blow up in the circuit extent; with access to the full 192GB of memory available to a compute node on the Myriad cluster, the maximum achievable T-count at $\epsilon = 0.995$ is still just $\sim .110$. Using 0.5PB of memory as in [102] would only add an additional 40 T-gates to the accessible range.

However, the results of Figures 4.10 and 4.11 underline that it is the stabilizer rank, controlled directly by the extent and the desired precision, that is the only significant factor on the runtime of our simulator. This represents a significant potential advantage over `qFlex` and comparable methods, the runtime of which depends on being able to decompose the circuits into large blocks with as few multi-qubit gates between blocks as possible.

Interestingly, examining the T-count of the all-to-all connectivity pattern in Figure 4.9b, we also see that there appears to be a value of the CZ fraction $0.1 < f < 0.5$ which maximises the number of T-gates in the circuit. At too

low a CZ-fraction, the circuits are too sparse, as we only add single qubit gates following a CZ gate. Similarly, at too high a fraction, qubits are frequently involved in an entangling gate, and we add fewer single-qubit gates. We in fact observe the same behaviour for the 1, 2 and 3D lattices, where the 2D connectivity has the largest T-count of all. Thus, not only is our simulator able to handle arbitrary connectivities, but the reduction in T-count means the extent of these random circuits actually decreases as the connectivity of the architecture increases.

In future, it would be interesting to examine how the cross-entropy difference behaves as a function of the precision and the connectivity. This would require an additional ideal realisation of the circuit, but given the circuit sizes considered this could be provided by a vector-based model. While `Qiskit-Aer` does implement a simulator of this type, access to the underlying state-vector is not yet supported, but is intended in a future update.

It would also be interesting to examine if the stabilizer rank method could be used to perform XEB. By definition, an approximate stabilizer rank decomposition cannot be used to compute an exact probability $P_U(\vec{\mathbf{x}})$; this could be achieved with an exact stabilizer rank expansion, using the results presented in Section 3.2.1, at the expense of a significant increase in the number of terms required.

Alternatively, we could consider computing estimates of $p_U(\vec{\mathbf{x}})$. We can compute a computational basis amplitude in time $O\left(\chi_\epsilon n^2\right)$, but also additionally need to reweight the result as

$$\tilde{p}_U(\vec{\mathbf{x}}) = \frac{1}{\left\|\tilde{\psi}\right\|} \sum_i \alpha_i \langle \vec{\mathbf{x}} | \phi_i \rangle ,$$

which requires $O\left(\chi_\epsilon L n^3\right)$ for L rounds of norm estimation. Recall that to achieve relative error ϵ in the norm estimate, we need $L = 4\epsilon^{-2}$ samples.

The key term in Equation 4.11 is an arithmetic mean of the logarithm of the

inverse probability of each sampled string, which we can rewrite as a geometric mean. Thus, given a set of m sampled strings $\vec{\mathbf{x}}_s$, and approximate probabilities $\tilde{p}_U(\vec{\mathbf{x}}_s)$, our estimate is given by

$$\tilde{\alpha} = \left[\prod_{s=1}^{m} \frac{1}{\tilde{p}_U(\vec{\mathbf{x}}_s)}\right]^{\frac{1}{m}} = \bar{\eta}\left[\prod_{s=1}^{m} \frac{1}{\left|\left\langle \vec{\mathbf{x}}_s \middle| \tilde{\psi} \right\rangle\right|}\right]^{\frac{1}{m}},$$

where $\bar{\eta}$ is the estimate of the norm of $\tilde{\psi}$ used to reweight each amplitude. $\bar{\eta}$ thus also gives a relative error contribution to $\tilde{\alpha}$. From the sparsification lemma, we also know that $\left\|\psi - \tilde{\psi}\right\|_1 \leq \delta$, and thus each computational amplitude estimate has an average additive error $O(\frac{\epsilon}{2^n})$.

Recalling that the Porter-Thomas distribution has significant support on terms with $p_U \leq \frac{1}{2^{-n}}$ [180], this would suggest we need to target $\epsilon = O\left(2^{-n/2}\right)$ to obtain good estimates of the cross-entropy, and this in turn would imply a stabilizer ranks and a number of samples L that are $O(2^n)$, suggesting that exact decompositions would likely be better suited to the XEB task.

Otherwise, if the stabilizer rank method is to be applied to problems of HOG, how can its performance be improved to access random circuits with greater depth and greater number of qubits? In Section 4.3.3, we will discuss more technical methods that could be used to better scale the simulator to HPC resources, and optimize its resource requirements. Here, we will consider possible methods looking at compiling circuits for the stabilizer rank method.

In particular, recent work by Qassim et al. introduced a method for recompiling circuits based on sum-over-Clifford expansions of non-Clifford unitaries. As Clifford operators can be written in terms of Pauli rotations as

$$V = \prod_i \mathrm{e}^{\mathrm{i}\theta_i P_i}$$

for some Pauli operator P and θ_i is a multiple of $\frac{\pi}{4}$. Thus, given a sum-over-Cliffords expansion $U = \sum_j \alpha_j V_j$ of a non-Clifford gate U, we can commute a

Clifford operator C through it as [2]

$$\begin{aligned} CU = CUC^\dagger C &= \left(\sum_j \alpha_j CVC^\dagger\right) C \\ &= \left(\sum_j \left(\prod_j C\mathrm{e}^{\mathrm{i}\theta_{i,j}P_{i,j}}C^\dagger\right)\right) C \\ &= \left(\sum_j \left(\prod_j C\mathrm{e}^{\mathrm{i}\theta_{i,j}CP_{i,j}C^\dagger}\right)\right) C \\ &= \left(\sum_j \left(\prod_j \mathrm{e}^{\mathrm{i}\theta'_{i,j}P'_{i,j}}\right)\right) C. \end{aligned} \tag{4.13}$$

Starting with a circuit built up of interleaved Clifford and non-Clifford layers acting on an initial stabilizer state

$$U = C_m U_m C_{m-1} \cdots C_1 U_1 |\phi\rangle$$

Clifford recompilation allows us to commute all Clifford operations through to the beginning of the circuit

$$\begin{aligned} U &= U'_m U'_{m-1} \cdots U'_1 C_m C_{m-1} \cdots C_1 |\phi\rangle \\ &= U'_m \cdots U'_1 C' |\phi\rangle \\ &= U'_m \cdots U'_1 \left|\phi'\right\rangle, \end{aligned}$$

where we have used the fact that the input is a stabilizer state to remove the Clifford terms. This reduces the runtime of the simulation as we only have to apply the Clifford sequence once, to compute the initial state, rather than applying each operator χ times for every term in the decomposition.

For Google circuits, the recompilation task is made easier as the only non-Clifford gate is already specified as a Pauli rotation, meaning we don't need to first make use of its sum-over-Cliffords expansion. This allows us to rewrite the circuit as a sequence of multi-qubit Pauli rotations acting on an initial stabilizer state.

In addition, the authors also show concrete cases where it is possible to build sum-over-Cliffords expansions of products of unitaries $U_i U_k$ that are 'contractive' — they have smaller extent than the multiplicative expansion [2]. In particular, the authors show that a product of any two Pauli rotations with the same angle has a contractive expansion. The argument relies on the existence Clifford circuit W that maps two multi-qubit Pauli operators P and Q to operators P', Q' with support on the same pair of qubits [2]. We provide an explicit description of how to construct such a Clifford circuit W in Algorithm 8. As a Clifford-recompiled Google circuit is just a sequence of exactly these rotations, contractive expansions could significantly reduce the stabilizer extent.

Applying Clifford recompilation to Google circuits would significantly reduce both the runtime of the simulation, and the value of the extent of the circuit, and thus expand the parameter space accessible to the stabilizer rank method.

4.3.3 Optimizing Decompositions and Sampling

Finally, there are several strategies that could be used to reduce the memory requirements and otherwise improve the scalability of the sum-over-Cliffords simulations. We focus only on sum-over-Cliffords here as this method is substantially more versatile than the gadgetized methods, and showed better performance overall.

Firstly, and recalling the discussion at the end of Section 3.3, the sampling method used to build a sum-over-Cliffords decomposition can generate multiple copies of the same state with non-zero probability. In current implementations, samples are taken a gate at a time, independently, and typically across multiple parallel threads, and so there is no deterministic way to check if a given term previously exists in the decomposition without sacrificing parallelization. Additionally, these inclusion checks would incur an additional cost of $O(\chi' n^2)$, where here we denote $\chi' < \chi$ as the number of terms in the decomposition obtained by grouping states.

One possible strategy then would to be precompute the samples for each non-Clifford gate in the circuit, and store this path. For a circuit with m Clifford gates, checking equality of two paths with require time $O(m)$ rather than $O(n^2)$. Sampling sum-over-Cliffords paths in this way would also enable us to optimize building stabilizer state decompositions. For example, if we have two paths s, s' that are equal for the first c Clifford gates, we can first compute $V_c V_{c-1} \dots V_1 |\phi\rangle$, then copy this state and use it as the input for the remaining fractions of the Clifford circuits.

Similar strategies could be employed to produce multiple samples using the Norm Estimation method. For example, say we want to take m samples from the full output distribution of a circuit. We begin by computing $P(x_0 = 0)$, and we sample bits 0 or 1 m times using the result. Say now we obtained a strings with $x_0 = 0$. We now compute the next probability, $P(x_0 = 0, x_1 = 0)$, which together with the previous result allows us to sample from the distribution

Algorithm 8 Explicit algorithm for constructing a Clifford circuit W that takes two n-qubit Pauli operators P and Q and maps them to new operators P', Q'' that have support on at most 2 qubits.

Require: n-qubit Pauli operators $P = \otimes_{i=1}^{n} P_i$, $Q = \otimes_{i=1}^{n} Q_i$
  Pick qubit $i : P_i = X$ or Y.
  $W_P \leftarrow I$ ▷ Initialize empty Clifford circuit
  for $j \neq i : P_j = \{X, Y\}$ **do**
    $W_P \leftarrow CNOT_{i,j} W_P$ ▷ $CNOT\,(XX)\,CNOT^\dagger = XI$
  end for
  for $j \neq i\, P_j = \{Z, Y\}$ **do**
    $W_P \leftarrow CZ_{i,j} W_P$ ▷ $CZ\,(XZ)\,CZ^\dagger = XI$
  end for
  $P' \leftarrow W_P P W_P^\dagger$ ▷ $|P'| = 1$
  $Q' \leftarrow W_P Q W_P^\dagger$
  Pick qubit $k \neq i : Q'_k = X$ or Y.
  $W_Q \leftarrow I$ ▷ Initialize empty Clifford circuit
  for $j \neq i, k : Q'_j = \{X, Y\}$ **do**
    $W_Q \leftarrow CNOT_{i,j} W_Q$
  end for
  for $j \neq i, k\, Q'_j = \{Z, Y\}$ **do**
    $W_Q \leftarrow CZ_{k,j} W_Q$
  end for
  $Q'' = W_Q Q' W_Q^\dagger$ ▷ $|Q'| \leq 2$
  return $W = W_Q W_P$ ▷ $P' \equiv W P W^\dagger\, Q'' = W Q W^\dagger$ as required.

$P(x_1 = 0|x_0 = 0)$. We take a samples, and repeat this process for the next bit. This method has the advantage that we do not need to run the full $O\left(\chi n^6\right)$ norm estimation step to obtain every sample. Instead, we build up multiple samples one bit at a time.

Finally, it is also interesting to note that as our simulation method can also produce estimates of output probabilities $\tilde{p}_U(\vec{\mathbf{x}})$, including for subsets of qubits using the norm estimation routine, the rejection sampling method of [140] should in principle also be implementable with our simulator.

Chapter 5

General Conclusions

Classical simulation has been integral to the study and development of quantum computation from its beginnings [31]. Continued development of classical methods has cast light on the requirements for a quantum advantage [53], and even guided the development of quantum hardware by excluding potential systems such as NMR quantum computation [187]. Now, in the NISQ era, classical simulations are also key to quantum supremacy experiments [95, 99]. This project considered classical simulation of quantum circuits based on the stabilizer rank method. Stabilizer rank decompositons are of particular interest as they have a clear connection to the notion of non-stabilizer states as a resource for quantum computing, and an immediate interpretation in terms of hardness of classical simulation.

In Chapter 2, we introduce novel classical simulators for stabilizer circuits, with additional capabilities beyond commonly used existing methods. Our implementations also have performance that is comparable to or improves on existing publicly available tools. It would also be interesting to compare our method for stabilizer inner products with those of [105], which are currently closed source.

Simulating stabilizer circuits has applications in quantum communication [188, 189], and in studying encoding and decoding circuits for stabilizer error correcting codes [66, 134]. Decompositions into stabilizer circuits can also be used to simulate universal quantum computations, and the additional information and routines in our classical data structures makes them advantageous for this purpose [109].

We discuss these kind of decompositions in Chapter 3, where we are able to

both extend previous results [49, 123] to different species of magic states, and also present techniques for building stabilizer rank decompositions of arbitrary quantum states.

Exact stabilizer rank is a non-convex quantity, and has proven difficult to characterise. We present evidence linking exact stabilizer rank to symmetries of the state, in particular with respect to the Clifford group. We also introduce the notion of stabilizer extent, a convex quantity that acts as an upper bound on approximate stabilizer rank, and show that it can be lower-bounded by the stabilizer fidelity.

Finally, in Chapter 4, we combine these two ingredients and show how they can be used to construct classical simulations of quantum circuits. In particular, we show how to perform strong and weak classical simulations, both in the exact case or approximate to within additive error [109]. The corresponding spatial and temporal complexity of our simulations scales as

$$O\left(\chi \mathrm{poly}\left(n\right)\right), \tag{5.1}$$

n the exact case, or as

$$O\left(\chi_\epsilon \mathrm{poly}\left(n, \epsilon\right)\right) \tag{5.2}$$

in the approximate case.

This method is especially appealing as its spatial requirements scale only polynomially with the number of qubits, enabling simulations of quantum circuits on large system sizes and with a bounded number of non-Clifford gates tractable even on a personal computer. However, our techniques also have a great propensity to be scaled to HPC systems, and we identify some interesting potential optimizations to the simulation method that could improve performance in this context. Finally, we note that the ability to both strongly and weakly simulate quantum circuits means the stabilizer rank methods also have the potential to act as both verifiers and 'heavy output generators' in the context of quantum supremacy experiments [99].

An important caveat to the discussions in this thesis is that all the methods discussed relate to simulating noiseless quantum circuits on pure states. The only method for incorporating noise into these simulations methods is with stochastic sampling of Pauli or Clifford errors, including measurements and resets. Incorporating these kind of operations cannot increase the simulation complexity, as can be seen from the properties of stabilizer rank discussed in Section 3.2. In fact, it is likely that the stabilizer rank would decrease if we included resets and measurements in the noise model. However, sampling noise in this way does incur an overhead, in that it requires running many repetitions of the circuit.

This restriction to pure circuits appears in tension with the knowledge that noisy quantum circuits can be efficiently simulated clasically. It would thus be interesting to investigate possible extensions of the stabilizer rank method to mixed states. One possible candidate would be simply to employ stabilizer rank decompositions to each term in a pure state decomposition; using this method, we could avoid an additional negativity overhead by using positive decompositions of the mixed state. However, there is no guarantee on how the overall stabilizer rank would behave for such a decomposition.

A natural mixed-state analogue of stabilizer rank would appear to be the Robustness of Magic, which can be defined as [75]

$$\mathcal{R}_{\mathcal{M}}(\rho) = \min_{\langle\phi|\phi\rangle} \|\vec{\mathbf{c}}\|_1 \ : \ \rho = \sum_i \vec{\mathbf{c}}_i \langle\phi_i|\phi_i\rangle . \tag{5.3}$$

Continuing research of the robustness of magic has looked at characterising the 'non-stabilizerness' of noisy quantum channels, and presented several techniques for simulating mixed state quantum computations with this method [81]. However, it is interesting to note that for pure states, it can be shown that $2\xi(\psi) - 1 \leq \mathcal{R}_{\mathcal{M}}(\langle\psi|\psi\rangle)$ [78]. This may suggest that there are potential savings to be found in extending stabilizer extent to the mixed state case.

Finally, we briefly consider the consequences of this thesis in terms of the hardness of simulating quantum computation. As discussed in Section 3.3, while we have presented various upper bounds on stabilizer rank, few lower bounds are known. While the work of [38] presents evidence of an exponential lower-bound for T-gates, we have shown that operations with large T-synthesis costs can in fact have much smaller stabilizer extent.

This leaves open the question of what causes the stabilizer rank of a system to grow exponentially, and in what cases it grows at most polynomially in the number of non-Clifford gates. Indeed, from Equations 5.1 and 5.2, the stabilizer rank method is explicitly capable of efficiently classically simulating any circuit with this property. However, the results presented in this thesis are often for individual magic states or non-Clifford gates, and extended to circuits through the submultiplicativity of stabilizer rank. Thus, even while we can conceive of states with small stabilizer extent ~ 1, their approximate stabilizer rank would still exhibit exponential growth. The existence of families of state of gate that admit such an efficient stabilizer rank simulation is an important open question. The results of [2] represent an important step in improving over these multiplicative bounds, and their application has the potential to greatly extend the range of circuits accessible to the stabilizer rank method. Work in this direction could also potentially help to close the gap between the bounds of [38], and the decompositions given in this thesis.

Bibliography

[1] G. Aleksandrowicz, T. Alexander, P. Barkoutsos *et al.* Qiskit: An Open-source Framework for Quantum Computing (2019).

[2] H. Qassim, J. J. Wallman, and J. Emerson. Clifford recompilation for faster classical simulation of quantum circuits (2019). arXiv:1902.02359.

[3] Y. Huang and P. Love. Approximate stabilizer rank and improved weak simulation of Clifford-dominated circuits for qudits. *Phys. Rev. A*, **99**, 052307 (2019).

[4] M. Beverland, E. Campbell, M. Howard *et al.* Lower bounds on the non-Clifford resources for quantum computations (2019). arXiv:1904.01124.

[5] UK National Quantum Technologies Programme. http://uknqt.epsrc.ac.uk/. Last Accessed: 2019-08-15.

[6] EU Quantum Flagship. https://qt.eu/. Last Accesssed: 2019-08-15.

[7] IBMQ: Quantum Systems. https://www.research.ibm.com/ibm-q/technology/devices/. Last Accessed: 2019-07-25.

[8] Google Quantum Computing. Last Accessed: 2019-08-17.

[9] Microsoft Quantum Computing. https://www.microsoft.com/en-us/quantum. Last Accessed: 2019-08-17.

[10] S. Lloyd. Universal Quantum Simulators. *Science*, **273**, 1073 (1996).

[11] K. L. Brown, W. J. Munro, and V. M. Kendon. Using Quantum Computers for Quantum Simulation. *Entropy*, **12**, 2268 (2010). arXiv:1004.5528.

[12] N. Moll, P. Barkoutsos, L. S. Bishop *et al.* Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, **3**, 030503 (2018). arXiv:1710.01022.

[13] L. K. Grover. A fast quantum mechanical algorithm for database search (1996). arXiv:quant-ph/9605043.

[14] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134 (1994).

[15] V. M. Kendon. A random walk approach to quantum algorithms. *Philosophical Transactions of the Royal Society of London Series A*, **364**, 3407 (2006). arXiv:quant-ph/0609035.

[16] A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum Algorithm for Linear Systems of Equations. *Phys. Rev. Lett.*, **103**, 150502 (2009). arXiv:0811.3171.

[17] J. Biamonte, P. Wittek, N. Pancotti *et al.* Quantum machine learning. *Nature*, **549**, 195 (2017). arXiv:1611.09347.

[18] Z. Cai, M. A. Fogarty, S. Schaal *et al.* A Silicon Surface Code Architecture Resilient Against Leakage Errors (2019). arXiv:1904.10378.

[19] Welcome to the Microsoft Quantum Development Kit Prview. https://docs.microsoft.com/en-gb/quantum/. Last Accessed: 2019-07-25.

[20] Google AI Blog: Announcing Circ. https://ai.googleblog.com/2018/07/announcing-cirq-open-source-framework.html. Last Accessed: 2019-07-25.

[21] C. Horsman, S. Stepney, R. C. Wagner *et al.* When does a physical system compute? *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, **470**, 20140182 (2014).

[22] C. Bissell. Historical perspectives - The Moniac A Hydromechanical Analog Computer of the 1950s. *IEEE Control Systems Magazine*, **27**, 69 (2007).

[23] A. Schönhage. On the power of random access machines. In H. A. Maurer, editor, *Automata, Languages and Programming*, pages 520–529. Springer Berlin Heidelberg, Berlin, Heidelberg (1979).

[24] S. Aaronson. NP-complete Problems and Physical Reality (2005). `arXiv:quant-ph/0502072`.

[25] F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General*, **15**, 3241 (1982).

[26] V. Choi. Adiabatic Quantum Algorithms for the NP-Complete Maximum-Weight Independent Set, Exact Cover and 3SAT Problems (2010). `arXiv:1004.2226`.

[27] A. Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, **2**, 5 (2014).

[28] S. F Edwards and P. W Anderson. Theory of Spin Glasses. *Journal of Physics F: Metal Physics*, **5**, 965 (1975).

[29] P. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, **22**, 563 (1980).

[30] M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information*. Cambridge University Press (2000).

[31] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, **21**, 467 (1982).

[32] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, **400**, 97 (1985).

[33] B. Schumacher. Quantum coding. *Phys. Rev. A*, **51**, 2738 (1995).

[34] T. Toffoli. Reversible Computing. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 632–644. Springer-Verlag, Berlin, Heidelberg (1980).

[35] S. Aaronson. Quantum Computing, Postselection, and Probabilistic Polynomial-Time (2004). `arXiv:quant-ph/0412187`.

[36] J. T. Gill,III. Computational Complexity of Probabilistic Turing Machines. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 91–95. ACM, New York, NY, USA (1974).

[37] S. Aaronson. Reasons to Believe. `https://www.scottaaronson.com/blog/?p=122` (2006). Last Accessed: 2019-08-19.

[38] A. M. Dalzell. *Lower bounds on the classical simulation of quantum circuits for quantum supremacy*. Bacherlor's, Massachusetts Institute of Technology (2017).

[39] A. Chi-Chih Yao. Quantum circuit complexity. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 352–361 (1993).

[40] E. Bernstein and U. Vazirani. Quantum Complexity Theory. *SIAM Journal on Computing*, **26**, 1411 (1997).

[41] J. Watrous. Quantum Computational Complexity (2008). `arXiv:0804.3401`.

[42] J. Kempe, A. Kitaev, and O. Regev. The Complexity of the Local Hamiltonian Problem (2004). `arXiv:quant-ph/0406180`.

[43] M. N. Vyalyi. QMA = PP implies that PP contains PH. In *In ECCCTR: Electronic Colloquium on Computational Complexity, technical reports* (2003).

[44] D. Aharonov and T. Naveh. Quantum NP - A Survey (2002). arXiv:quant-ph/0210077.

[45] C. H. Bennett, E. Bernstein, G. Brassard *et al.* Strengths and Weaknesses of Quantum Computing. *SIAM J. Comput.*, **26**, 1510 (1997).

[46] M. Van den Nest. Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond (2008). arXiv:0811.0898.

[47] H. Pashayan, S. D. Bartlett, and D. Gross. From estimation of quantum probabilities to simulation of quantum circuits (2017). arXiv:1712.02806.

[48] D. Hangleiter, J. Bermejo-Vega, M. Schwarz *et al.* Anticoncentration theorems for schemes showing a quantum speedup (2017). arXiv:1706.03786.

[49] S. Bravyi and D. Gosset. Improved Classical Simulation of Quantum Circuits Dominated by Clifford Gates. *Phys. Rev. Lett.*, **116**, 250501 (2016). arXiv:1601.07601.

[50] M. J. Bremner, R. Jozsa, and D. J. Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. *Proceedings of the Royal Society of London Series A*, **467**, 459 (2011). arXiv:1005.1407.

[51] M. Yoganathan, R. Jozsa, and S. Strelchuk. Quantum advantage of unitary Clifford circuits with magic state inputs. *Proceedings of the Royal Society of London Series A*, **475** (2019). arXiv:1806.03200.

[52] A. Ekert and R. Jozsa. Quantum algorithms: entanglement-enhanced information processing. *Philosophical Transactions of the Royal Society of London Series A*, **356**, 1769 (1998). arXiv:quant-ph/9803072.

[53] R. Jozsa and N. Linden. On the role of entanglement in quantum-computational speed-up. *Proceedings of the Royal Society of London Series A*, **459**, 2011 (2003). arXiv:quant-ph/0201143.

[54] P. Niemann, R. Wille, D. M. Miller *et al.* QMDDs: Efficient Quantum Function Representation and Manipulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **35**, 86 (2016).

[55] A. Zulehner and R. Wille. Advanced Simulation of Quantum Computations (2017). arXiv:1707.00865.

[56] M. Van den Nest. Simulating quantum computers with probabilistic methods (2009). arXiv:0911.1624.

[57] M. Schwarz and M. Van den Nest. Simulating Quantum Circuits with Sparse Output Distributions (2013). arXiv:1310.6749.

[58] G. Vidal. Efficient Classical Simulation of Slightly Entangled Quantum Computations. *Phys. Rev. Lett.*, **91**, 147902 (2003). arXiv:147902.

[59] I. L. Markov and Y. Shi. Simulating quantum computation by contracting tensor networks (2005). arXiv:quant-ph/0511069.

[60] W. K. Wootters. A Wigner-function formulation of finite-state quantum mechanics. *Annals of Physics*, **176**, 1 (1987).

[61] D. Gross. Hudson's theorem for finite-dimensional quantum systems. *Journal of Mathematical Physics*, **47**, 122107 (2006). arXiv:quant-ph/0602001.

[62] S. D. Bartlett, B. C. Sanders, S. L. Braunstein *et al.* Efficient Classical Simulation of Continuous Variable Quantum Information Processes. *Phys. Rev. Lett.*, **88**, 097904 (2002).

[63] A. Mari and J. Eisert. Positive Wigner Functions Render Classical Simulation of Quantum Computation Efficient. *Phys. Rev. Lett.*, **109**, 230503 (2012). arXiv:1208.3660.

[64] H. Pashayan, J. J. Wallman, and S. D. Bartlett. Estimating Outcome Probabilities of Quantum Circuits Using Quasiprobabilities. *Phys. Rev. Lett.*, **115**, 070501 (2015). arXiv:1503.07525.

[65] D. Gottesman. The Heisenberg Representation of Quantum Computers (1998). arXiv:quant-ph/9807006.

[66] S. Aaronson and D. Gottesman. Improved simulation of stabilizer circuits. *Phys. Rev. A*, **70**, 052328 (2004). arXiv:quant-ph/0406196.

[67] D. Gottesman and I. L. Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, **402**, 390 (1999). arXiv:quant-ph/9908010.

[68] S. Bravyi and A. Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A*, **71** (2005). arXiv:quant-ph/0403025.

[69] M. Howard, J. Wallman, V. Veitch *et al.* Contextuality supplies the 'magic' for quantum computation. *Nature*, **510**, 351 (2014). arXiv:1401.4174.

[70] J. S. Bell. On the Problem of Hidden Variables in Quantum Mechanics. *Rev. Mod. Phys.*, **38**, 447 (1966).

[71] S. Kochen and E. P. Specker. The Problem of Hidden Variables in Quantum Mechanics. *Journal of Mathematics and Mechanics*, **17**, 59 (1967).

[72] J. Bermejo-Vega, N. Delfosse, D. E. Browne *et al.* Contextuality as a Resource for Models of Quantum Computation with Qubits. *Phys. Rev. Lett.*, **119**, 120505 (2017).

[73] F. G. S. L. Brandão and G. Gour. Reversible Framework for Quantum Resource Theories. *Phys. Rev. Lett.*, **115**, 070503 (2015). arXiv:1502.03149.

[74] V. Veitch, S. A. Hamed Mousavian, D. Gottesman *et al.* The resource theory of stabilizer quantum computation. *New Journal of Physics*, **16**, 013009 (2014). arXiv:1307.7171.

[75] M. Howard and E. Campbell. Application of a Resource Theory for Magic States to Fault-Tolerant Quantum Computing. *Phys. Rev. Lett.*, **118**, 090501 (2017). arXiv:1609.07488.

[76] M. Piani, M. Cianciaruso, T. R. Bromley *et al.* Robustness of asymmetry and coherence of quantum states. *Physical Review A*, **93**, 042107 (2016). arXiv:1601.03782.

[77] J. Bermejo-Vega, C. Yen-Yu Lin, and M. Van den Nest. Normalizer circuits and a Gottesman-Knill theorem for infinite-dimensional systems (2014). arXiv:1409.3208.

[78] B. Regula. Convex geometry of quantum resource quantification. *Journal of Physics A Mathematical General*, **51**, 045303 (2018). arXiv:1707.06298.

[79] E. Chitambar and G. Gour. Quantum resource theories. *Reviews of Modern Physics*, **91**, 025001 (2019). arXiv:025001.

[80] L. Kocia and P. Love. Discrete Wigner formalism for qubits and noncontextuality of Clifford gates on qubit stabilizer states. *Physical Review A*, **96**, 062134 (2017). arXiv:1705.08869.

[81] J. R. Seddon and E. Campbell. Quantifying magic for multi-qubit operations (2019). arXiv:1901.03322.

[82] R. Raussendorf, J. Bermejo-Vega, E. Tyhurst *et al.* Phase space simulation method for quantum computation with magic states on qubits

(2019). arXiv:1905.05374.

[83] S. Toda. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, **20**, 865 (1991).

[84] C. M. Dawson, H. L. Haselgrove, A. P. Hines *et al.* Quantum computing and polynomial equations over the finite field Z_2 (2004). arXiv:quant-ph/0408129.

[85] A. Montanaro. Quantum circuits and low-degree polynomials over F_2. *Journal of Physics A Mathematical General*, **50**, 084002 (2017). arXiv:1607.08473.

[86] D. Aharonov, I. Arad, E. Eban *et al.* Polynomial Quantum Algorithms for Additive approximations of the Potts model and other Points of the Tutte Plane (2007). arXiv:quant-ph/0702008.

[87] G. Kuperberg. How hard is it to approximate the Jones polynomial? (2009). arXiv:0908.0512.

[88] S. Aaronson and A. Arkhipov. The Computational Complexity of Linear Optics (2010). arXiv:1011.3245.

[89] L. A. Goldberg and H. Guo. The complexity of approximating complex-valued Ising and Tutte partition functions (2014). arXiv:1409.5627.

[90] K. Fujii and T. Morimae. Commuting quantum circuits and complexity of Ising partition functions. *New Journal of Physics*, **19**, 033003 (2017).

[91] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, **8**, 189 (1979).

[92] A. W. Harrow and A. Montanaro. Quantum computational supremacy. *Nature*, **549**, 203 (2017). arXiv:1809.07442.

[93] M. J. Bremner, A. Montanaro, and D. J. Shepherd. Average-Case Complexity Versus Approximate Simulation of Commuting Quantum Computations. *Physical Review Letters*, **117**, 080501 (2016). arXiv:080501.

[94] L. Stockmeyer. On Approximation Algorithms for # P. *SIAM Journal on Computing*, **14**, 849 (1985).

[95] J. Preskill. Quantum computing and the entanglement frontier (2012). arXiv:1203.5813.

[96] X. Gao, S.-T. Wang, and L. M. Duan. Quantum Supremacy for Simulating a Translation-Invariant Ising Spin Model. *Physical Review Letters*, **118**, 040502 (2017). arXiv:040502.

[97] J. Bermejo-Vega, D. Hangleiter, M. Schwarz *et al.* Architectures for Quantum Simulation Showing a Quantum Speedup. *Physical Review X*, **8**, 021010 (2018). arXiv:021010.

[98] J. Haferkamp, D. Hangleiter, A. Bouland *et al.* Closing gaps of a quantum advantage with short-time Hamiltonian dynamics (2019). arXiv:1908.08069.

[99] S. Aaronson and L. Chen. Complexity-Theoretic Foundations of Quantum Supremacy Experiments (2016). arXiv:1612.05903.

[100] J. Preskill. Quantum Computing in the NISQ era and beyond (2018). arXiv:1801.00862.

[101] A. Neville, C. Sparrow, R. Clifford *et al.* No imminent quantum supremacy by boson sampling (2017). arXiv:1705.00686.

[102] B. Villalonga, D. Lyakh, S. Boixo *et al.* Establishing the Quantum Supremacy Frontier with a 281 Pflop/s Simulation (2019). arXiv:1905.00444.

[103] J. Dehaene and B. de Moor. Clifford group, stabilizer states, and linear and quadratic operations over GF(2). *Phys. Rev. A*, **68**, 042318 (2003). arXiv:quant-ph/0304125.

[104] S. Anders and H. J. Briegel. Fast simulation of stabilizer circuits using a graph-state representation. *Phys. Rev. A*, **73**, 022334 (2006). arXiv:quant-ph/0504117.

[105] H. J. García, I. L. Markov, and A. W. Cross. Efficient Inner-product Algorithm for Stabilizer States (2012). arXiv:1210.6646.

[106] CHP. https://www.scottaaronson.com/chp/. Last Accessed: 2019-05-13.

[107] H. J. García and I. L. Markov. Simulation of Quantum Circuits via Stabilizer Frames. *IEEE Transactions on Computers*, **64**, 2323 (2017). arXiv:1712.03554.

[108] K. N. Patel, I. L. Markov, and J. P. Hayes. Efficient Synthesis of Linear Reversible Circuits (2003). arXiv:quant-ph/0302002.

[109] S. Bravyi, D. Browne, P. Calpin *et al.* Simulation of quantum circuits by low-rank stabilizer decompositions (2018). arXiv:1808.00128.

[110] S. Bravyi, D. Gosset, and R. König. Quantum advantage with shallow circuits. *Science*, **362**, 308 (2018). arXiv:1704.00690.

[111] E. T. Campbell and M. Howard. Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost. *Phys. Rev. A*, **95**, 022316 (2017). arXiv:1606.01904.

[112] C++ Reference: Fundamental Types. https://en.cppreference.com/w/cpp/language/types. Last Accessed: 2019-06-19.

[113] C++ Reference: Bitwise Operators. https://en.cppreference.com/w/cpp/language/operator_arithmetic#Bitwise_logic_operators. Last Accessed: 2019-06-19.

[114] Mathworks Documentation: External Language Interfaces. https://uk.mathworks.com/help/matlab/external-language-interfaces.html. Last Accessed: 2019-06-20.

[115] D. Schlingemann. Stabilizer codes can be realized as graph codes (2001). arXiv:quant-ph/0111080.

[116] M. Van den Nest, J. Dehaene, and B. De Moor. Efficient algorithm to recognize the local Clifford equivalence of graph states. *Phys. Rev. A*, **70** (2004). arXiv:quant-ph/0302002.

[117] Github.com: GraphSim. https://github.com/Roger-luo/GraphSim. Last Accessed: 2019-06-23.

[118] M. J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, **C-21**, 948 (1972).

[119] Intel Streaming SIMD Extensions Technology. https://www.intel.com/content/www/us/en/support/articles/000005779/processors.html. Last Accessed: 2019-06-24.

[120] LAPACK in MATLAB. https://uk.mathworks.com/help/matlab/math/lapack-in-matlab.html. Last Accessed: 2019-06-24.

[121] J. Dongarra, R. Pozo, and D. Walker. LAPACK++: A design overview of object-oriented extensions for high performance linear algebra. , pages 162– 171 (1993).

[122] C. L. Lawson, R. J. Hanson, D. R. Kincaid *et al.* Basic Linear Algebra Subprograms for Fortran Usage. *ACM Trans. Math. Softw.*, **5**, 308 (1979).

[123] S. Bravyi, G. Smith, and J. A. Smolin. Trading Classical and Quantum Computational Resources. *Phys. Rev. X*, **6** (2016). arXiv:1506.01396.

[124] N. J. Ross and P. Selinger. Exact and approximate synthesis of quantum circuits. `https://www.mathstat.dal.ca/~selinger/newsynth/`. Last Accessed: 2019-07-08.

[125] N. J. Ross and P. Selinger. Optimal ancilla-free Clifford+T approximation of z-rotations (2014). `arXiv:1403.2975`.

[126] S. van der Walt, S. C. Colbert, and G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, **13**, 22 (2011).

[127] S. K. Lam, A. Pitrou, and S. Seibert. Numba: A LLVM-based Python JIT Compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, pages 7:1–7:6. ACM, New York, NY, USA (2015).

[128] M. Lundberg and L. Svensson. The Haar measure and the generation of random unitary matrices. In *Processing Workshop Proceedings, 2004 Sensor Array and Multichannel Signal*, pages 114–118 (2004).

[129] A. Knyazev and M. Argentati. Principal Angles between Subspaces in an A-Based Scalar Product: Algorithms and Perturbation Estimates. *SIAM Journal on Scientific Computing*, **23**, 2008 (2002).

[130] A. W. Harrow. The Church of the Symmetric Subspace (2013). `arXiv:1308.6595`.

[131] H. Zhu, R. Kueng, M. Grassl *et al.* The Clifford group fails gracefully to be a unitary 4-design (2016). `arXiv:1609.08172`.

[132] M. Artin. *Algebra: 2nd Edition*. Pearson (2010).

[133] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press (2004).

[134] D. Gottesman. *Stabilizer codes and quantum error correction*. Ph.D. thesis, California Institute of Technology (1997). arXiv:quant-ph/9705052.

[135] C. Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Phys. Rev. A*, **87**, 022328 (2013). arXiv:1212.5069.

[136] L. E. Heyfron and E. T. Campbell. An efficient quantum compiler that reduces T count. *Quantum Science and Technology*, **4**, 015004 (2018). arXiv:1712.01557.

[137] B. Natarajan. Sparse Approximate Solutions to Linear Systems. *SIAM Journal on Computing*, **24**, 227 (1995). arXiv:https://doi.org/10.1137/S0097539792240406.

[138] D. Gross, S. Nezami, and M. Walter. Schur-Weyl Duality for the Clifford Group with Applications: Property Testing, a Robust Hudson Theorem, and de Finetti Representations (2017). arXiv:1712.08628.

[139] I. L. Markov, A. Fatima, S. V. Isakov *et al.* Quantum Supremacy Is Both Closer and Farther than It Appears (2018). arXiv:1807.10749.

[140] B. Villalonga, S. Boixo, B. Nelson *et al.* A flexible high-performance simulator for the verification and benchmarking of quantum circuits implemented on real hardware (2018). arXiv:1811.09599.

[141] A. Dahlberg and S. Wehner. SimulaQron - A simulator for developing quantum internet software (2017). arXiv:1712.08032.

[142] W. G. T. Delft. NetSquid. Last Accessed: 2019-07-24.

[143] A. Dahlberg, M. Skrzypczyk, T. Coopmans *et al.* A Link Layer Protocol for Quantum Networks (2019). arXiv:1903.09778.

[144] T. Häner, D. S. Steiger, K. Svore *et al.* A software methodology for compiling quantum programs. *Quantum Science and Technology*, **3**, 020501 (2018). arXiv:1604.01401.

[145] K. M. Svore, A. Geller, M. Troyer *et al.* Q#: Enabling scalable quantum computing and development with a high-level domain-specific language (2018). arXiv:1803.00652.

[146] D. S. Steiger, T. Häner, and M. Troyer. ProjectQ: An Open Source Software Framework for Quantum Computing (2016). arXiv:1612.08091.

[147] A. W. Cross, L. S. Bishop, J. A. Smolin *et al.* Open Quantum Assembly Language (2017). arXiv:1707.03429.

[148] Github.com: Google Circ. https://github.com/quantumlib/Cirq. Last Accessed: 2019-07-25.

[149] Rigetti Computing: Forest SDK. https://www.rigetti.com/forest. Last Accessed: 2019-07-25.

[150] R. S. Smith, M. J. Curtis, and W. J. Zeng. A Practical Quantum Instruction Set Architecture (2016). arXiv:1608.03355.

[151] Rigetti Computing: QPU Specifications. https://www.rigetti.com/qpu. Last Accessed: 2019-07-25.

[152] T. Häner, D. S. Steiger, M. Smelyanskiy *et al.* High Performance Emulation of Quantum Circuits. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 74:1–74:9. IEEE Press, Piscataway, NJ, USA (2016).

[153] Top500.org: Summit. https://www.top500.org/system/179397. Last Accessed: 2019-07-25.

[154] M. Smelyanskiy, N. P. D. Sawaya, and A. Aspuru-Guzik. qHiPSTER: The Quantum High Performance Software Testing Environment (2016). arXiv:1601.07195.

[155] N. Khammassi, I. Ashraf, X. Fu *et al.* QX: A high-performance quantum computer simulation platform. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 464–469 (2017).

[156] G. Aleksandrowicz, T. Alexander, P. Barkoutsos *et al.* https://qiskit.org/aer. Last Accessed: 2019-07-25.

[157] T. Jones, A. Brown, I. Bush *et al.* QuEST and High Performance Simulation of Quantum Computers (2018). arXiv:1802.08032.

[158] D. Gil. Discovering a New Era of Computing. https://rebootingcomputing.ieee.org/rebooting-computing-week/industry-summit-2017 (2017). IEEE Industry Summit 2017.

[159] Google AI Blog: A Preview of Bristlecone. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html. Last Accessed: 2019-07-26.

[160] D. Aharonov, M. Ben-Or, E. Eban *et al.* Interactive Proofs for Quantum Computations (2017). arXiv:1704.04487.

[161] U. Mahadev. Classical Verification of Quantum Computations (2018). arXiv:1804.01082.

[162] E. Pednault, J. A. Gunnels, G. Nannicini *et al.* Breaking the 49-Qubit Barrier in the Simulation of Quantum Circuits (2017). arXiv:1710.05867.

[163] J. Chen, F. Zhang, C. Huang *et al.* Classical Simulation of Intermediate-Size Quantum Circuits (2018). arXiv:1805.01450.

[164] Z.-Y. Chen, Q. Zhou, C. Xue *et al.* 64-Qubit Quantum Circuit Simulation (2018). arXiv:1802.06952.

[165] E. Gamma, R. Helm, R. Johnson *et al. Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley (1994).

[166] C++ Reference: Name requirements - Function Object. https://en.cppreference.com/w/cpp/named_req/FunctionObject. Last Accessed: 2019-07-31.

[167] C++ Reference: Templates. https://en.cppreference.com/w/cpp/language/templates. Last Accessed: 2019-07-31.

[168] G. M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485. ACM, New York, NY, USA (1967).

[169] C. Kessler and J. Keller. Models for Parallel Computing: Review and Perspectives. *PARS-Mitteilungen, ISSN 0177-0454*, **24**, 13 (2007).

[170] OpenMP Architecture Review Board. OpenMP 4.5 API C/C++ Syntax Reference Guide. https://www.openmp.org/wp-content/uploads/OpenMP-4.5-1115-CPP-web.pdf (2015).

[171] Open MPI: Open Source High Performance Computing. https://www.open-mpi.org/. Last Accessed: 2019-08-03.

[172] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 3.1.* High Performance Computing Center Stuttgart (2015).

[173] M. Mosca. Quantum Algorithms (2008). arXiv:0808.0369.

[174] M. Roetteler. Quantum algorithms for highly non-linear Boolean functions (2008). arXiv:0811.3208.

[175] E. Farhi, J. Goldstone, and S. Gutmann. A Quantum Approximate Optimization Algorithm (2014). arXiv:1411.4028.

[176] E. Farhi, J. Goldstone, and S. Gutmann. A Quantum Approximate Optimization Algorithm Applied to a Bounded Occurrence Constraint Problem (2014). arXiv:1412.6062.

[177] U. Wolff and Alpha Collaboration. Monte Carlo errors with less errors. *Computer Physics Communications*, **156**, 143 (2004). arXiv:hep-lat/0306017.

[178] J. Emerson, Y. S. Weinstein, M. Saraceno *et al.* Pseudo-Random Unitary Operators for Quantum Information Processing. *Science*, **302**, 2098 (2003).

[179] J. Martinis. The Quantum Space Race. Quantum Information Processing 2018.

[180] S. Boixo, S. V. Isakov, V. N. Smelyanskiy *et al.* Characterizing Quantum Supremacy in Near-Term Devices. *Nature Physics*, **14**, 595 (2018). `arXiv:1608.00263`.

[181] A. Bouland, B. Fefferman, C. Nirkhe *et al.* Quantum Supremacy and the Complexity of Random Circuit Sampling (2018). `arXiv:1803.04402`.

[182] R. Movassagh. Efficient unitary paths and quantum computational supremacy: A proof of average-case hardness of Random Circuit Sampling (2018). `arXiv:1810.04681`.

[183] F. G. S. L. Brandao and M. Horodecki. Exponential Quantum Speed-ups are Generic (2010). `arXiv:1010.3654`.

[184] A. Harrow and S. Mehraban. Approximate unitary t-designs by short random quantum circuits using nearest-neighbor and long-range gates (2018). `arXiv:1809.06957`.

[185] E. Schuyler Fried, N. P. D. Sawaya, Y. Cao *et al.* qTorch: The Quantum Tensor Contraction Handler (2017). `arXiv:1709.03636`.

[186] M. B. Hastings. Classical and Quantum Bounded Depth Approximation Algorithms (2019). `arXiv:1905.07047`.

[187] S. L. Braunstein, C. M. Caves, R. Jozsa *et al.* Separability of Very Noisy Mixed States and Implications for NMR Quantum Computing. *Physical Review Letters*, **83**, 1054 (1999). `arXiv:quant-ph/9811018`.

[188] C. H. Bennett and S. J. Wiesner. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Phys. Rev. Lett.*, **69**, 2881 (1992).

[189] C. H. Bennett, G. Brassard, C. Crépeau *et al.* Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Lett.*, **70**, 1895 (1993).