
Stabilizer-based classical simulation of quantum computers with practical implementations

Author: Ricardo Rivera Cardoso (Student ID: 12735787)

1st supervisor: Dr. Michael Walter
daily supervisor: Dr. Jonas Helsen
2nd reader: Dr. Maris Ozols



*A thesis submitted in fulfillment of the requirements for
the joint UvA-VU Master of Science degree in Theoretical Physics.*

Submitted: August 26, 2021

“Strange about learning; the farther I go the more I see that I never knew even existed. A short while ago I foolishly thought I could learn everything - all the knowledge in the world. Now I hope only to be able to know of its existence, and to understand one grain of it. Is there time?”

– Daniel Keyes, *Flowers for Algernon*

Abstract

Current quantum computers are afflicted by hardware and decoherence errors with effects that are difficult to characterize, as the exact state of the quantum computer cannot be observed. Therefore, classical simulators, which mimic the unperturbed state of the quantum computer and provide a reference for comparison, are essential for the development of better quantum computers. Building upon two efficient classical simulation techniques of *Gottesman* (1) and *Aaronson* (2) of Clifford circuits, *Bravyi et al.* (3) recently presented a simulator capable of universal quantum computation with significantly smaller costs than a state vector simulator. Here, we discuss and implement in Python these three simulation algorithms, while focusing our attention on the Metropolis-based stabilizer rank simulator (also known as extended stabilizer simulator) of *Bravyi et al.* Here, we propose an experimental method to validate, as well as address some of the assumptions and open questions of the algorithm. Our results provide better estimates on simulation hyperparameters that improve simulation time by a factor of two hundred when compared with parameters derived from theoretic principles. Moreover, we observe that even in the regime of large qubit systems, the Metropolis algorithm produces, with low but still significant probability, non-ergodic chains that hinder simulation. In the case where the Metropolis algorithm succeeds, we observe that the chain's mixing time is significantly influenced by circuit structure. Finally, even with Python's large overhead, our simulator's runtime approached IBM's implementation by a factor of two. Moreover, our simulator presents the advantage of being written in a user-friendly language with a structure that facilitates improvements and expansions. It also allows users to define the gates necessary for their own simulations, and most importantly, it is not restricted to simulations of *Clifford+T* gates, and can evaluate small qubit systems, as well as systems above 63 qubits. Our software can be found in <https://github.com/amsqi/stabilizer-rank-simulator>.

Acknowledgements

First and foremost, I would like to thank my advisors Dr. Michael Walter and Dr. Jonas Helsen for their never-ending patience, support, and kindness. Truly, this is the way to mentor someone, and I wish more people knew this. I would also like to thank Dr. Maris Ozols for agreeing to evaluate my thesis, as well as all other QuSoft researchers for sharing your wisdom on the Friday seminars and the occasional hilarious group activities.

Second, I want to thank my parents for their hard work and love which have made literally (in the strongest sense of the word) everything that surrounds me possible. My brother for his cheerfulness and for being my personal consultant on topics I know nothing about. And my girlfriend for standing besides me (although not physically) all past 5 years, always believing in me and cheering me on even if it meant being in two different continents. No matter what I write here will it ever do justice for how much she means to me.

Last but not least, I would like to thank my friends for the much needed laughs and distractions that kept me from spiraling, and for liking me even if I don't answer your messages in months.

Contents

Notational Conventions	vii
1 Introduction	1
1.1 Classical simulators of quantum computers	2
1.2 Structure of this thesis	3
2 Preliminaries & Basics of implementation	5
2.1 Review of linear algebra	5
2.2 Quantum states	7
2.2.1 Qubits	8
2.3 Comparing states	9
2.4 Unitary gates	11
2.5 Pauli operators	12
2.5.1 Single-qubit and n-qubit Pauli operators	12
2.5.2 Pauli-to-binary mapping	13
2.5.3 Pauli group	14
2.6 Measurements and expectations	14
2.7 Basics of implementation	16
2.8 Summary	19
3 Stabilizer formalism	20
3.1 Stabilizers	20
3.2 Clifford group	23
3.3 Density matrix	25
3.4 Summary	26
4 The Gottesman algorithm	27
4.1 Clifford circuits	27
4.2 Pauli measurements and expectation values	29
4.2.1 P anticommutes with at least one generator	30
4.2.2 P commutes with all generators	30
4.3 Implementation	31
4.3.1 Main class	31
4.3.2 Run	32
4.3.3 Adding a Clifford gate	32
4.3.4 Benchmark	32

4.4	Summary	33
5	The Tableau algorithm	34
5.1	Pauli measurements and expectations	35
5.1.1	P anticommutes with at least one generator	35
5.1.2	P commutes with all generators	35
5.2	Implementation	36
5.3	Summary	37
6	Simulation of low-rank stabilizer decompositions	39
6.1	Stabilizer decompositions	40
6.1.1	Sum-over-Cliffords	41
6.2	CH-form	43
6.2.1	Representation	43
6.2.2	Phase-sensitive simulation	44
6.2.3	Density matrix	50
6.2.4	Pauli expectation	54
6.2.5	Overlap	55
6.3	Measurements in the computational basis	55
6.3.1	Metropolis-Hastings algorithm	55
6.3.2	Limitations	57
6.4	Sparsification	58
6.4.1	Proof of sparsification	60
6.5	Implementation	63
6.5.1	Recap	63
6.5.2	Sparsification grouped with Sum-over-Cliffords	63
6.5.3	Circuit decompositions	64
6.5.4	CH-form simulation	64
6.5.5	Sampling	65
6.5.6	Adding to the gate set	66
6.6	Summary	67
7	Validation & Benchmark	68
7.1	Validation	68
7.1.1	Experimental statement of success	69
7.1.2	Testing our simulator's success	69
7.1.3	Decomposition size	70
7.1.4	Heuristic Metropolis steps	72
7.1.5	Practical validation	74
7.2	Benchmark	74
8	Conclusions & Further work	77
8.1	Conclusion	77
8.2	Further work	79
8.2.1	General improvements	79
8.2.2	Our simulator	80

CONTENTS

References	81
Appendix A Hardware and third-party software	84
Appendix B Validation Circuit	86

Notational Conventions

- $j \in [n]$ is an abbreviation of $j = 0, \dots, n - 1$.
- We sometimes write $|\psi\rangle$ as ψ to avoid clutter.
- We will omit the dimensions of the identity operator as this is, most of the time, obvious from the context.
- For an n -qubit system, we adopt the qubit ordering where a basis state is of the form $|x_0 \dots x_{n-1}\rangle = |x_0\rangle \otimes \dots \otimes |x_{n-1}\rangle$. Qiskit orders the qubits in reverse.
- The tensor product symbol “ \otimes ” is omitted in most cases. In the case of n -qubits Pauli operators, we simply add a subindex to express in which location of the tensor product a non-identity single-qubit Pauli operator acts on. For example the four-qubit Pauli operator $P = X \otimes Z \otimes I \otimes I$ is written as $X_0 Z_1$. In addition, n -qubit basis states are simply written as $|i_0 i_1 \dots i_{n-1}\rangle$.
- $[A, B]$ refers to the commutation operator between A and B , while $\{A, B\}$ refers to the anti-commutation.

1

Introduction

In 1982, Richard Feynman proposed the idea of building a “programmable” computer relying on quantum mechanical principles to efficiently¹ simulate arbitrary quantum systems (4), much like we use classical computers to investigate classical problems. Feynman’s ideas were studied over the next years when in 1992, David Deutsch and Richard Jozsa proposed the first problem which could be solved efficiently by a quantum computer, while no efficient classical solution is known. The exciting prospect of a possible separation between quantum and classical computers gave rise to a whole new model of computation and motivated others to take part in the field. Soon after, Peter Shor developed an algorithm with exponential improvement over all known classical algorithms for the factorization of large numbers into its prime components (5), followed by computer scientist Lov Grover’s algorithm for searching for an item through an unordered database (6) polynomially faster than in a classical computer. Finally, in 1996 Seth Lloyd proved what Feynman had in mind. He showed that quantum computers can be programed to simulate all types of quantum systems given a sufficient amount of *qubits*, the main units for quantum computation. These events jump-started the field of quantum computation, and now, almost 40 years after Feynman’s idea many more algorithms for quantum computers have been developed. For example, using quantum computers to solve linear systems of equations (7) and to reduce the dimensionality of data in unsupervised machine learning (8).

Shor’s, Grover’s and other algorithms for quantum computers seem to provide a clear computational advantage over their classical counterparts, however, these require around hundreds of thousands of qubits. Present-day quantum computers possess at most hundreds of physical qubits and often display many errors during the computation. Therefore it is currently not possible to implement such algorithms without first constructing better computers.

There are three main challenges engineers and scientists have to face in order to develop large-scale quantum computers. The first is that it is difficult to design hardware capable of manipulating qubits with precision. For example, an ion trap quantum computer might require adjusting the frequency of lasers that manipulate the ions (qubits) by a matter of millihertz, or in the case of superconducting quantum computers, adjusting minute voltages. Second, is that qubits inevitably couple to the environment causing them to *decohere*

¹Problems that can be solved using a polynomial amount of time and memory are called *efficient*, while problems that require an exponential amount of resources are called *inefficient*.

1.1 Classical simulators of quantum computers

or “leak” information and slowly change the state of the system. The longer a computation takes, the more these qubits will decohere, and ultimately invalidate the computation. Lastly, it is the fact that the quantum superposition collapses when the state of the qubits is measured, taking a quantum state to a classical one. Hence, it is physically impossible to observe the intermediate state of a quantum computer and determine whether a given instruction or modification to the state had undesired effects that might lead to an erroneous result. A possible solution is to use classical simulators, which mimic the behavior and quantum state of a quantum computer unaltered by errors, and therefore allow us to compare this expected result with that of the quantum computer and hope to detect what might have gone wrong. Generally, classical simulators of quantum computers are limited to the simulation of small-qubit systems or short-depth circuits because the cost of simulation grows exponentially with these parameters. However, a smaller exponent could mean the difference between simulating a 100-qubit circuit in a matter of minutes or years. For these reasons, studying and devising new simulation techniques is an important active area of research.

1.1 Classical simulators of quantum computers

To be precise, classically simulating a quantum computer can refer to one of two things. *Strong simulation* refers to being able to compute the probabilities associated with measurement outcomes, while *weak simulation* refers to sampling from this probability distribution. Evidently, strong simulation offers more information about the supposed state of a quantum system, but it usually a harder task to implement than weak simulation (9).

The simplest method of strongly simulating the state of a quantum computer is by tracking the 2^n complex amplitudes of its *state vector*, where n is the number of qubits in the system. Then, evaluating a quantum circuit, a collection of unitary matrices or “gates” corresponds to matrix-vector operations which classical computers can perform with ease as long as the size of the matrices is not too large. However, the problem with this simulation technique is that the size of the matrices and vectors grow exponentially with the number of qubits n which quickly renders the process intractable. For example, simulating a 52 qubit system, much less than what is required for Shor’s and Grover’s algorithms, requires storing and manipulating 2^{52} complex amplitudes, the equivalent to 40 petabytes of RAM. This is 8 times more memory that the most powerful supercomputers currently in existence!¹

It is a widely believed conjecture that any classical simulator capable of *universal quantum computation*, *i.e.*, capable of creating any quantum state, should have exponential scaling in time and/or memory. If a classical computer could efficiently simulate any arbitrary quantum state using only a polynomial amount of resources, this would imply that quantum computers can offer at best polynomial speedups over their classical counterparts. Hence, quantum computers would not be as powerful and impressive as currently thought of, and understanding them would probably pose a less interesting challenge. This conjecture is also observed in the categorization of simulation techniques. Most techniques for simulating arbitrary quantum circuits can be divided into two main categories: the *Schrödinger* and the *Feynman* approach. The *Schrödinger* approach to simulation consists

¹As of June 2021, [Fugaku](#) is the most powerful supercomputer and has about 5 petabytes of RAM.

of finding creative ways of storing and manipulating the 2^n complex amplitudes of the state vector, accompanied by an exponential memory scaling. Some of the examples of simulators belonging to this category are parallel (10) and decision diagram (11) simulators. On the other hand, the *Feynman* approach evaluates subsets of the gates in a quantum circuit separately and finalizes by adding all individual contributions. Intuitively, simulators that belong in this class scale exponentially with the number of gates in the circuit (12).¹ It is evident that these two classes of simulators agree with the conjecture.

If one relaxes the condition of universal quantum computation, there are classical simulators that can efficiently simulate particular families of circuits that are relevant within the field and have a wide variety of applications. Some of these are *fermionic linear optics* circuits, where the evolution of the state is driven by time-evolving a quadratic Hamiltonian, and are relevant for the simulation of fermionic systems (13). Another type are Clifford simulators, which consist only of *CNOT*, *H* and *S* gates that are relevant for benchmarking quantum computers (14), detecting and correcting errors (15, 16), and to the study of entanglement (17). In addition, recent developments on the simulation of Clifford circuits has shown that it is possible to allow for simulation of universal quantum computation at costs that, while still exponential, are much better than a state vector simulator. This new technique results in an exotic Feynman-type simulation technique that considers approximate simulation of the state, and the cost is not only given by the number of gates in the circuit, but by the type of gates used. This simulation technique is given the name of *stabilizer rank simulator* or *extended stabilizer simulator*. In this thesis we seek to understand this new and exciting result, as well as create our own implementation of the algorithm. Our goals in this thesis are three-fold. First, is to discuss the theory of stabilizers which allow for efficient classical simulation of Clifford circuits, as well as present two algorithms on how to achieve this. Second, is to present the state-of-the-art simulation technique that allows for simulation of *approximate Clifford circuits*, a class of circuits capable of universal quantum computation, as well as provide a practical method of validating the simulator and address some of the open-question surrounding this technique. Lastly, we aim to create versatile and easy-to-use Python simulators based on these three simulation algorithms. For the extended stabilizer simulator, we aim for it to have similar runtime as IBM's implementation of the same algorithm found in their [quantum software development kit \(Qiskit\)](#). Moreover, we want our simulator to be flexible enough to meet user's needs, and possibly incorporate recently found expansions discussed in Refs. (18, 19) which are not available in Qiskit.

1.2 Structure of this thesis

This thesis is organized as follows. In Chapter 2 we recall some concepts from linear algebra and quantum computation necessary to understand the material presented in this thesis. Here, we also discuss the Python files and classes that are common to all of the simulators we built. In Chapter 3 we discuss the theory of *stabilizers*, which are the foundation to all simulation algorithms studied in the remaining chapters of the thesis. In Chapter 4 we

¹This technique is named after Richard Feynman due its resemblance to the Feynman path integral formalism.

discuss the first algorithm for efficient simulation of Clifford circuits and measurements in the standard basis (1). This algorithm has runtime $\mathcal{O}(n^3)$ and takes $\mathcal{O}(n^2)$ memory, where n is the number of qubits in the system. In Chapter 5 we show an improved version of the previous algorithm with simulation time of $\mathcal{O}(n^2)$ and $\mathcal{O}(n^2)$ memory (2). In Chapter 6 we extend the theory of stabilizers and present an algorithm that, based on the previous two chapters, allows for simulation of approximate Clifford circuits in time that scales exponentially with the number of non-Clifford gates in the circuit. At the end of each of the previous three chapters, we discuss details on how we implement each algorithm in Python, as well as some of the challenges we faced and our software’s simulation times. The purpose of these sections is to guide users who desire to use our simulator or to inform curious readers of our design choices. Finally, in Chapter 7 we verify that the claims and approximations made in Chapter 6 indeed result in a simulator that works as intended, as well as experimentally study in more detail some of the simulation hyperparameters, and finally, compare the simulation times of our software with Qiskit.

2

Preliminaries & Basics of implementation

The purpose of this chapter is for the reader to become familiar with the notation and tools used in the field of quantum computation and in this thesis. We will not present thorough discussions or proofs of properties and statements in this section as most of these can be found in common literature. The material in this thesis, we believe, is self-contained and this section should be enough for the reader to understand the following chapters. In the case that the reader requires or desires more information about the basics of the field, we recommend the book *Quantum Computation and Quantum Information* (20). Readers familiar with the field may skip until Section 2.7 where we discuss the common modules of our Python simulators and the general workflow.

2.1 Review of linear algebra

Here, we list some useful linear algebra concepts and properties that will be relevant for the rest of this thesis.

Definition 2.1.1 (Hilbert space). *A Hilbert space is an infinite or finite-dimensional vector space equipped with an inner product.*¹

Remark 1. *In this thesis we will assume that all Hilbert spaces are complex and with finite dimensions.*

Definition 2.1.2 (Positive semi-definite and definite operators). *Positive semi-definite operators are those whose eigenvalues λ_i follow $\lambda_i \geq 0$. And positive definite if $\lambda_i > 0$.*

Definition 2.1.3 (Hermitian operators). *An operator B is said to be Hermitian if $B = B^\dagger$, where “ \dagger ” denotes the adjoint or conjugate transpose.*²

¹The more familiar Euclidean space is an example of a real-valued Hilbert space.

²The conjugate transpose consists on transposing the matrix and taking the complex conjugate of all entries in the matrix. The order in which the operations take place is irrelevant.

2.1 Review of linear algebra

Moreover, Hermitian operators follow the *spectral theorem*, claiming that these operators are diagonalizable, have real eigenvalues λ_i , and have an orthonormal eigenbasis. Hence, a Hermitian operator B can be written as

$$B = \sum_i \lambda_i \Pi_i, \quad (2.1)$$

where Π_i are orthogonal projections related to the eigenvectors of B .

Definition 2.1.4 (Unitary operators). *An operator U is said to be unitary if $U^\dagger U = U U^\dagger = I$.*

And from the definition, it is clear that $U^{-1} = U^\dagger$.

Definition 2.1.5 (Commutation relations). *The commutator between two operators A and B in the same Hilbert space is given by*

$$[A, B] \equiv AB - BA. \quad (2.2)$$

These operators are said to commute if $[A, B] = 0$ or $AB = BA$. The anticommutator between the same operators is given by

$$\{A, B\} \equiv AB + BA, \quad (2.3)$$

and they are said to anticommute if $\{A, B\} = 0$ or $AB = -BA$.

Definition 2.1.6 (Inner product). *The inner product between two vectors u and v , both in $\mathcal{H} = \mathbb{C}^n$, is defined as*

$$\langle u, v \rangle \equiv \begin{pmatrix} u_1^* & \dots & u_n^* \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = u_1^* v_1 + u_2^* v_2 + \dots + u_n^* v_n, \quad (2.4)$$

and the asterisk represents the complex conjugate.

The inner product in a Hilbert space is *conjugate symmetric*, that is $\langle u, v \rangle = \langle v, u \rangle^*$.

Definition 2.1.7 (Outer product). *Let A be an $m \times n$ matrix on Hilbert space $\mathcal{H}_A = \mathbb{C}^{m \times n}$ and B be an $k \times l$ matrix on $\mathcal{H}_B = \mathbb{C}^{k \times l}$, the outer product or tensor product is defined as*

$$A \otimes B \equiv \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix}, \quad (2.5)$$

which results in an operator acting on $\mathcal{H}_A \otimes \mathcal{H}_B = \mathbb{C}^{mk \times nl}$.

In general, the outer product is not symmetric, i.e. $A \otimes B \neq B \otimes A$. The tensor product also has the property that

$$(A \otimes B)(C \otimes D) = AC \otimes BD \quad (2.6)$$

and

$$\alpha A \otimes \beta B = \alpha\beta(A \otimes B) \quad (2.7)$$

for $\alpha, \beta \in \mathbb{C}$.

Definition 2.1.8 (Trace). *The trace of a square $n \times n$ matrix C is defined as*

$$\text{Tr}(C) \equiv \sum_{i=1}^n c_{ii}. \quad (2.8)$$

The trace has the property that it is *cyclic*. That is for any n operators C_1, \dots, C_n , the trace of the product of these operators is the same for cyclic permutations of the operators. For example, if we let $n = 3$, then

$$\text{Tr}(C_1 C_2 C_3) = \text{Tr}(C_3 C_1 C_2) = \text{Tr}(C_2 C_3 C_1). \quad (2.9)$$

The trace is also a linear transformation, meaning that

$$\text{Tr}(\alpha C) = \alpha \text{Tr}(C) \quad \text{and} \quad \text{Tr}(C_1 + \dots + C_n) = \sum_{i=1}^n \text{Tr}(C_i), \quad (2.10)$$

for $\alpha \in \mathbb{C}$.

2.2 Quantum states

Here, we explain quantum states and the *qubit*, the most common unit for quantum computation.

Definition 2.2.1 (Pure quantum state). *A normalized pure quantum state $|\psi\rangle$ is a complex vector in a Hilbert space \mathcal{H} of magnitude one, i.e. $\|\psi\|^2 = \langle\psi|\psi\rangle = 1$.^{1,2}*

A related quantity to pure quantum states is the density matrix *density matrix*.

Definition 2.2.2. *The density matrix acting on a Hilbert space is a positive semi-definite, Hermitian operator of trace one given by*

$$\rho \equiv \sum_j p_j |\psi_j\rangle \langle\psi_j|, \quad (2.11)$$

where the $|\psi_j\rangle$ are pure states orthogonal to each other, $p_j \in \mathbb{R}$, and $\sum p_j = 1$.

¹The symbol $|\cdot\rangle$, called *ket*, stands for a column vector in Hilbert space \mathcal{H} , while the dual vector in Hilbert space \mathcal{H}^* is depicted by the symbol $\langle\cdot|$, called *bra*. We can obtain one from the other by transposing and conjugating the vector, so $|\cdot\rangle^\dagger = \langle\cdot|$. The inner product is then $\langle\cdot|\cdot\rangle$. This notation for vectors and dual vectors is known as *Dirac notation*.

²Just like in this statement, we sometimes write $|\psi\rangle$ as ψ to avoid clutter or possible confusions.

The density matrix thus represents uncertainty over the pure state at hand (also known as a statistical ensemble of pure states), and is useful particularly in the field of quantum information theory when noise introduced by manipulating states is taken into consideration. The density matrix of a pure state $|\psi\rangle$ (no uncertainty) is then

$$\rho = |\psi\rangle\langle\psi|. \quad (2.12)$$

In this thesis we use the density matrix as a method of comparison between our algorithms as in these cases it is easier to compute than the state vector.

In quantum computing, we usually work with a specific orthonormal basis of the Hilbert space called the *computational basis*, where $\mathcal{H} = \mathbb{C}^d$ for some integer d and the basis vectors are $\{|0\rangle, |1\rangle, \dots, |d-1\rangle\}$ where

$$|0\rangle = (1, 0, \dots, 0)^T, |1\rangle = (0, 1, \dots, 0)^T, \dots, |d-1\rangle = (0, \dots, 0, 1)^T \quad (2.13)$$

and T denotes the transpose. It is orthonormal since we impose that $\langle j|i\rangle = \delta_{ij}$ for all i and j . An arbitrary state in this space can then be written as

$$|\psi\rangle = \sum_{i=0}^{d-1} c_i |i\rangle, \quad (2.14)$$

where $c_i \in \mathbb{C}$ and $\sum_{i=0}^{d-1} |c_i|^2 = 1$. The condition demanding that the sum of the coefficients squared adds up to one is made to ensure that the state satisfies Definition 2.2.1. Also, this condition motivates the treatment of such coefficients as *probability amplitudes*, since when squared these represent probability values.

The computational basis states are sometimes called *classical states* as these are often attributed with physical meaning such as the occupation number of an optical cavity or the direction of the spin of a particle. Moreover, for reasons that will become apparent in Section 2.6, when the quantum state is measured, the state collapses to one of the classical states $|i\rangle$ with probability $|c_i|^2$. Eq. (2.14) then suggests that quantum states are in a *superposition* of classical states.

2.2.1 Qubits

In this thesis, we will only work with *qubits*, composite quantum systems where each individual *qubit* “lives” in the Hilbert space $\mathcal{H} = \mathbb{C}^2$. From Eq. (2.14) a qubit is simply

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \text{ with } |\alpha|^2 + |\beta|^2 = 1. \quad (2.15)$$

Fig. 2.1 offers a useful pictorial representation of a qubit, and it will prove useful for the reader to think of the unitary operations introduced in the next subsections as rotations and reflections in this sphere.

A system of n qubits, can be expressed by computing the tensor product between their individual spaces, the composite system then ‘lives’ in the space $\mathbb{C}^{2^n} = \mathbb{C}^2 \otimes \dots \otimes \mathbb{C}^2$ and the basis of the space becomes $|x_0\rangle \otimes \dots \otimes |x_{n-1}\rangle$ with $x_i \in \{0, 1\}$ which is abbreviated as

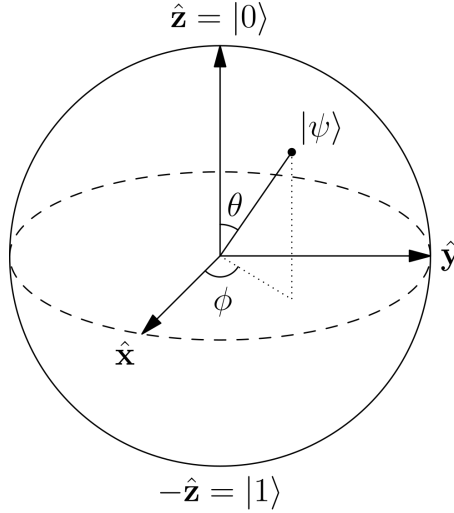


Figure 2.1: The state of an arbitrary qubit can be parametrized as $|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\phi}\sin(\theta/2)|1\rangle$ with $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$ resulting in the *Bloch sphere*. An specific qubit state is then a vector that points on the surface of the sphere. Here, $\hat{x} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $\hat{y} = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$. (From Wikipedia: “Bloch Sphere”).

$|x_0 \dots x_{n-1}\rangle$. In this case, the basis vectors can be uniquely represented by binary strings of length n . Finally, the state vector of the n -qubit system is given by

$$|\psi\rangle = \begin{pmatrix} c_0 \\ \vdots \\ c_{2^n} \end{pmatrix}. \quad (2.16)$$

Remark 2. We adopt the convention where the n -qubit system is $|x_0 \dots x_{n-1}\rangle = |x_0\rangle \otimes \dots \otimes |x_{n-1}\rangle$ instead of $|x_{n-1} \dots x_0\rangle = |x_{n-1}\rangle \otimes \dots \otimes |x_0\rangle$. In other words, we append new qubits by “tensoring” the existing ones from the right, which by Definition 2.1.7 is not the same as appending from the left. Qiskit uses the other convention.

2.3 Comparing states

An essential part of many computations in quantum computing involves comparing two distinct quantum states. In this section we present three methods of doing so.

Given that states are vectors in a space, the most intuitive method of comparison is to ask how far apart these two vectors are in the space. This is given by the Euclidean norm for complex vector spaces.

Definition 2.3.1 (Euclidean norm for states). Let $|\psi\rangle = \sum_{i=0}^{d-1} \alpha_i |i\rangle$ and $|\phi\rangle = \sum_{i=0}^{d-1} \beta_i |i\rangle$ be two states in the Hilbert space \mathbb{C}^d . The Euclidean norm between the states is defined as

$$\|\psi - \phi\| \equiv \sqrt{\sum_{i=0}^{d-1} |\alpha_i - \beta_i|^2}. \quad (2.17)$$

Another definition on the similarity of the states that arises naturally is the *fidelity*.

Definition 2.3.2 (Fidelity). *Let $|\psi\rangle$ and $|\phi\rangle$ be two states in the same Hilbert space H . The fidelity between the states is simply given by their inner product, so*

$$\mathcal{F}(\psi, \phi) \equiv |\langle\psi|\phi\rangle| = |\langle\phi|\psi\rangle|. \quad (2.18)$$

Finally, in Section 2.6 we will observe that measurement outcomes of quantum states are non-deterministic and are in fact random variables distributed according to some discrete probability distribution. More precisely, we will show that when measuring a state $|\psi\rangle$ of the form given in Eq. (2.14), we will obtain outcome i with probability $|\alpha_i|^2$. Therefore, another way of comparing states is through these probability mass functions. There are multiple ways of comparing two probability distributions, such as computing the coefficient of variation or the relative entropy that is widely used in the fields of quantum computation and information, however, here we make use of a particular one called the *total variation distance*.

Definition 2.3.3 (Total variation distance for states). *Let $p_i \equiv |\alpha_i|^2$ and $q_i \equiv |\beta_i|^2$ be the probabilities associated with obtaining outcomes i from measurements of the states $|\psi\rangle$ and $|\phi\rangle$, respectively. These specify probability mass functions P_ψ and Q_ϕ . The total variation distance between these probability mass functions is defined as*

$$\delta_{tvd}(P_\psi, Q_\phi) \equiv \frac{1}{2} \sum_{i=0}^{d-1} |p_i - q_i|. \quad (2.19)$$

In Chapter 7 we will make use of the following relation between the Euclidean norm and the total variation distance.

Claim 1. *For two states $|\psi\rangle = \sum_{i=1}^{d-1} \alpha_i |i\rangle$ and $|\phi\rangle = \sum_{j=0}^{d-1} \beta_j |j\rangle$ in the Hilbert space \mathbb{C}^d , if $\|\psi - \phi\| \leq \delta$ then the total variation distance of the probabilities related to measurement outcomes of the states is $\delta_{tvd}(P_\psi, Q_\phi) \leq \delta$.*

Proof.

$$\begin{aligned} \delta_{tvd}(P_\psi, Q_\phi) &= \frac{1}{2} \sum_{i=0}^{d-1} |\alpha_i^2 - \beta_i^2| \\ &= \frac{1}{2} \sum_{i=0}^{d-1} |\alpha_i - \beta_i| |\alpha_i + \beta_i| \\ &\leq \frac{1}{2} \sqrt{\sum_{i=0}^{d-1} |\alpha_i - \beta_i|^2} \sqrt{\sum_{i=0}^{d-1} |\alpha_i + \beta_i|^2} \\ &\leq \frac{\delta}{2} \sqrt{\sum_{i=0}^{d-1} |\alpha_i + \beta_i|^2} \\ &\leq \delta, \end{aligned} \quad (2.20)$$

where in the third line we used the Cauchy-Schwarz inequality followed by the fact that $\|\psi - \phi\| = \sqrt{\sum_i |\alpha_i - \beta_i|^2} \leq \delta$. \square

2.4 Unitary gates

Now, we describe the allowed transformations that take a quantum state in some Hilbert space to another quantum state in the same space.

To see which type of operator we need, suppose $|\phi\rangle = U|\psi\rangle$ for some normalized state $|\psi\rangle$ and U is a linear operator $U : \mathbb{C}^d \rightarrow \mathbb{C}^d$. For $|\phi\rangle$ to be a valid pure state, by Definition 2.2.1, we require that $\langle\phi|\phi\rangle = \langle\psi|U^\dagger U|\psi\rangle = 1$. Therefore, the operator U must satisfy $U^\dagger U = I_d$ *i.e.*, U is a unitary operator (See Definition 2.1.4).

In the circuit-based model of quantum computation (the one considered in this thesis), the operators and qubits in which they act are arranged in discrete time steps, where each qubit can be acted upon at most once in a single time step. Due to its resemblance with classical computing, this form of organizing qubits and operators is called a *quantum circuit*, and the unitary operators are called *gates*. Unlike in classical computation where the only non-trivial gate one can apply on a single bit is the *NOT* operator, the fact that that any vector pointing at the surface of the Bloch sphere (Fig. 2.1) is a valid quantum state suggests that there are an infinite number of unitaries. However, some unitary operators are more relevant than others. Fig. 2.2 shows some of the ones most used in this thesis, along with their circuit element.




Matrix representation	Action on basis states	Circuit element
$H \equiv \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$H : 0\rangle \mapsto \frac{1}{2}(0\rangle + 1\rangle)$ $H : 1\rangle \mapsto \frac{1}{2}(0\rangle - 1\rangle)$	
$R_\phi \equiv \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$	$R_\phi : 0\rangle \mapsto 0\rangle$ $R_\phi : 1\rangle \mapsto e^{i\phi} 1\rangle$	
$CNOT \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$CNOT_{0,1} : 01\rangle \mapsto 01\rangle$ $CNOT_{0,1} : 00\rangle \mapsto 00\rangle$ $CNOT_{0,1} : 10\rangle \mapsto 11\rangle$ $CNOT_{0,1} : 11\rangle \mapsto 11\rangle$	

Figure 2.2: The *Hadamard* H is useful for creating even superpositions of the basis states. It has the property that it is its own inverse, so applying H twice (consecutively) does not change the state. In the Bloch sphere, this operation is equivalent to a rotation of 90° around the \hat{y} -axis followed by a 180° rotation around the \hat{x} -axis. R_ϕ adds a phase ϕ to the basis state $|1\rangle$. The $CNOT$ is a two-qubit operator useful for controlling a qubit based on the the state of another qubit. Most importantly, it can also be used to create entanglement or quantum correlations between qubits. Like in the case of the $CNOT$, multi-qubit operators are usually accompanied with subindices that clarify on which qubits they act.

2.5 Pauli operators

Another important set of unitary operators, besides those of Fig. 2.2, are the *Pauli operators*, which play a fundamental role in all of quantum mechanics and in the stabilizer formalism.

2.5.1 Single-qubit and n-qubit Pauli operators

Definition 2.5.1 (Single-qubit Pauli operators). *We will denote the set of single-qubit Pauli operators as*

$$P^* \equiv \{I, X, Y, Z\}, \quad (2.21)$$

with matrix representation

$$I \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y \equiv \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z \equiv \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \quad (2.22)$$

The multiplication table of these matrices is given in Table 2.5.1, and some properties we will use repeatedly through the text are listed in Proposition 2.5.1.

\times	I	X	Y	Z
I	I	X	Y	Z
X	X	I	iZ	$-iY$
Y	Y	$-iZ$	I	iX
Z	Z	iY	$-iX$	I

Table 2.1: Table of multiplication for single-qubit Pauli operators. The order of the operations is to multiply the operator in the column followed by the one in the row.

Proposition 2.5.1.

1. *All Pauli operators are unitary operators.*
2. *All Pauli operators are Hermitian operators.*
3. *They form a basis of the the space of 2×2 Hermitian matrices.*
4. *X, Y and Z have eigenvalues ± 1 with multiplicity 1.*
5. *Any two operators $P, P' \in P^*$ commute or anticommute. All elements commute with the identity and with themselves, and $\{X, Y\} = 0, \{X, Z\} = 0$, and $\{Y, Z\} = 0$.¹*
6. *$Y = iXZ$.*
7. *$X^2 = Y^2 = Z^2 = I$.*
8. *$\text{Tr}(X) = \text{Tr}(Y) = \text{Tr}(Z) = 0$, but $\text{Tr}(I) = 2$.*

¹The commutation relations for these operators can also be seen in Table 2.5.1.

An n -qubit Pauli operator $P \in P_n^*$ is defined as follows.

Definition 2.5.2 (n -qubit Pauli operator). *An n -qubit Pauli operator is an operator of the form*

$$P = P_1 \otimes P_2 \otimes \dots \otimes P_n, \quad (2.23)$$

where $P_i \in P^*$.

Often, the tensor product is omitted and, for example, the 3-qubit Pauli operator $Z \otimes X \otimes X$ is abbreviated as $Z_0 X_1 X_2$. It can be shown that the n -qubit Pauli operators are still unitary, Hermitian, form a basis of the space of $2^n \times 2^n$ Hermitian matrices, and have eigenvalues ± 1 .

2.5.2 Pauli-to-binary mapping

The n -qubit Pauli operators will be an essential part of the simulation algorithms discussed in this thesis, and here we present a more appropriate representation of these operators in terms of bits that classical computers can process better.

Claim 2. *Any n -qubit Pauli operator can be parametrized by two vectors $z, x \in \mathbb{F}_2^n$ such that*

$$P(z, x) = i^{-\langle z, x \rangle} Z^z X^x, \quad (2.24)$$

where we define $Z^z = Z_0^{z_0} Z_1^{z_1} \dots Z_{n-1}^{z_{n-1}}$ and $X^x = X_0^{x_0} X_1^{x_1} \dots X_{n-1}^{x_{n-1}}$. Here, $\langle z, x \rangle = \sum_i^n z_i x_i \pmod 4$ is the inner product for vectors in \mathbb{F}_2^n .

The vectors z and x can be thought of binary strings of length n that indicate in which positions of the tensor product there is a single-qubit Pauli Z or X . If both z and x have a 0 on the same position, this then refers to the identity operator, while if they both have a 1, by Proposition 2.5.1, this refers to a Y . To make this even clearer consider the following example.

Example 2.5.1. *Suppose $z = (0, 1, 1)$ and $x = (0, 1, 0)$ or 011 and 010 as binary strings. Then, $P(011, 010) = i^{-(011, 010)} (I \otimes Z \otimes Z)(I \otimes X \otimes I) = -i Z_1 X_1 Z_2 = Y_1 Z_2$.*

We can shorten the notation even further by considering the vector $(z, x) \in \mathbb{F}_2^{2n}$, which can think of as a single binary string of length $2n$. Hence, $P(z, x) = P((z, x))$.

In this representation, the commutation relations of two elements of P_n^* are given by

$$\begin{aligned} P((z, x))P((z', x')) &= i^{-\langle z, x \rangle} i^{-\langle z', x' \rangle} Z^z X^x Z^{z'} X^{x'} \\ &= i^{-\langle z, x \rangle} i^{-\langle z', x' \rangle} (-1)^{\langle x, z' \rangle} Z^z Z^{z'} X^x X^{x'} \\ &= i^{-\langle z, x \rangle} i^{-\langle z', x' \rangle} (-1)^{\langle x, z' \rangle} Z^{z'} Z^z X^{x'} X^x \\ &= i^{-\langle z', x' \rangle} i^{-\langle z, x \rangle} (-1)^{\langle x, z' \rangle - \langle x', z \rangle} Z^{z'} X^{x'} Z^z X^x \\ &= (-1)^{\langle x, z' \rangle - \langle x', z \rangle} P((z', x'))P((z, x)), \end{aligned} \quad (2.25)$$

showing that the Pauli operators will commute if the sum in the exponent of the sign is even, and anticommute if it is odd. This exponent appears often in literature and is

2.6 Measurements and expectations

called the *symplectic inner product* of vectors (z, x) and (z', x') , usually represented by the symbol “ \odot ”. Hence,

$$P((z, x))P((z', x')) = (-1)^{(z, x) \odot (z', x')} P((z', x'))P((z, x)). \quad (2.26)$$

By a similar calculation to the one above, one can show that the multiplication of two Pauli operators is

$$\begin{aligned} P((z, x))P((z', x')) &= i^{-\langle z, x \rangle - \langle z', x' \rangle - 2\langle z', x \rangle} Z^{z+z'} X^{x+x'} \\ &= i^{(z, x) \odot (z', x')} P((z' + z, x' + x)), \end{aligned} \quad (2.27)$$

where the sum in the exponent of i is taken mod 4 and the sum in the exponents of X and Z is mod 2. One can check that this yields the correct representation of any product of Pauli operators. Eq. (2.27) then shows that multiplication of n -qubit Pauli operators is equivalent, up to a phase, to vector addition mod 2.

2.5.3 Pauli group

We can now define a more complicated object, the n -qubit Pauli group \mathcal{P}_n .

Definition 2.5.3 (n -qubit Pauli group). *The 1-qubit Pauli group is defined as*

$$\mathcal{P}_1 \equiv \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}, \quad (2.28)$$

with matrix multiplication as the group operation. Then, the n -qubit Pauli group is defined as the n -fold tensor product of elements of \mathcal{P}_1 or

$$\mathcal{P}_n \equiv \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}^{\otimes n}. \quad (2.29)$$

2.6 Measurements and expectations

In this section we describe how to measure quantum systems to obtain classical outcomes that we can understand and process, as well as compute expected values of observables (physical attributes of the system). There is much discussion in the scientific community about what measurements really, the collapse of the wavefunction and the division between the classical and quantum realms. In this thesis we abstain from discussing such matters and approach the problem from a purely axiomatic perspective.

In simple words, measurements should be represented by orthogonal projections that take a quantum state to a classical state.

Definition 2.6.1 (Projective measurement). *A projective measurement is carried out by a set of projectors Π_1, \dots, Π_m with $\sum_{j=1}^m \Pi_j = I_d$, where d is the dimension of the Hilbert space. The probability of observing outcome j given the state $|\psi\rangle$ is given by Born’s rule, stating that*

$$\Pr(\text{outcome } j) \equiv \text{Tr}(\Pi_j |\psi\rangle\langle\psi|). \quad (2.30)$$

2.6 Measurements and expectations

The normalized state after the measurement is then $|\phi\rangle = \Pi_j|\psi\rangle/||\Pi_j|\psi\rangle||$.

The most common projective measurement is called the *measurement in the computational basis*, where the projectors are given by $\{|0\rangle\langle 0|, \dots, |d\rangle\langle d|\}$ and d is the dimension of the Hilbert space. From Eq. (2.13) it is evident that the sum of these projectors is in fact the identity, and hence they are a valid projective measurement. Consider an arbitrary state of the form given by Eq. (2.14). The probability of obtaining outcome j (with projector $|j\rangle\langle j|$) when measuring state $|\psi\rangle$, is

$$\begin{aligned} \Pr(\text{outcome } j) &= \text{Tr}(|j\rangle\langle j|\psi\rangle\langle\psi|) \\ &= \langle j|\psi\rangle\langle\psi|j\rangle = \langle j|\psi\rangle\langle j|\psi\rangle^* \\ &= |\langle j|\psi\rangle|^2 = |\langle j|\sum_{i=1}^{d-1} c_i|i\rangle|^2 \\ &= |\sum_{i=1}^{d-1} c_i\delta_{ij}|^2 = |c_j|^2, \end{aligned} \tag{2.31}$$

where in the first line we used the cyclicity of the trace (Eq. (2.9)) followed by the conjugate symmetric property of the inner product (Definition 2.1.6), and in the last line we made use of the orthonormality condition of the computational basis states. Therefore, the coefficient c_i associated with a classical state $|i\rangle$ in a quantum superposition dictates the probability, that when measured, the quantum state collapses to the $|i\rangle$ classical state.

Another sensible thing to ask is what would be the expected value of an *observable*, a physical parameter of the system, such as position, momentum, spin, and so forth. Observables are represented by Hermitian operators, since the spectral theorem of Eq. (2.1) naturally associates outcomes (eigenvalues) with projective measurements. The expected value of observable O is

$$\begin{aligned} \langle O \rangle_\psi &= \sum_{i=1}^{d-1} \lambda_j \Pr(\text{outcome } j) = \sum_j \lambda_j \text{Tr}(\Pi_j|\psi\rangle\langle\psi|) \\ &= \text{Tr}\left(\sum_j \lambda_j \Pi_j |\psi\rangle\langle\psi|\right) = \text{Tr}(O|\psi\rangle\langle\psi|). \end{aligned} \tag{2.32}$$

A recurring topic in this thesis is that, most of the time, global phases of the state have no physical relevance. For example, consider an arbitrary state $|\psi\rangle$ and apply a global phase $e^{i\phi}$ to the state such that $|\psi'\rangle = e^{i\phi}|\psi\rangle$. Suppose we then want to calculate the expectation value of an observable O . Then,

$$\langle O \rangle_{\psi'} = \langle\psi|e^{-i\phi}Oe^{i\phi}|\psi\rangle = \langle\psi|O|\psi\rangle = \langle O \rangle_\psi, \tag{2.33}$$

meaning that there is no physical means of distinguishing $|\psi'\rangle$ from $|\psi\rangle$. We therefore say that these are “equal”.

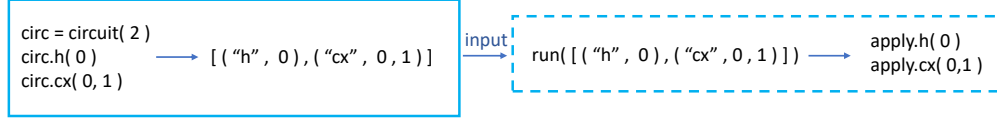


Figure 2.3: Shows an example on how to construct the 2 qubit circuit $U = CNOT_{0,1}H_0$. First, the list of tuples are created by the sequence of `GATE(args)` methods. This list then serves as input to a `run(circuit)` function associated with the simulator one wants to use to evaluate the circuit. The simulator then reads the list of circuit instructions and applies its own `apply_GATE(args)` methods that contain the rules for evolving the state of the system for that particular gate.

2.7 Basics of implementation

Here, we discuss the `Circuit` class¹ whose instances will be the inputs to the Python simulators we will construct over the next chapters. We also show the general workflow related to the use of our simulators, as well as summarize what each simulator is capable of. This section will mainly refer to the `circuit.py` file of the GitHub repository.

In our software, a quantum circuit object is simply an ordered list of tuples of the form (gate name, arg_1, \dots, arg_k) where arg_j refers to extra arguments that are used depending on the gate. For example, the tuple associated with the gate H is simply (H, q_i) where q_i is the qubit that the gate acts on. Another example is the CR_ϕ , which has three extra arguments: the control c_i and target t_j qubits, as well as the angle of rotation ϕ . Hence, this gate is represented by $(CR_\phi, \phi, c_i, t_j)$. Moreover, the list of instructions has an implicit ordering that should be respected when evaluating the circuit since, in general, the order in which the gates are applied to a state matter. We chose this simple representation of a circuit since each simulator interacts with it differently, and in particular, this structure will be useful in the discussion of the implementation of the extended stabilizer simulator in Chapter 6.

The first step towards defining a quantum circuit for a simulation task is to instantiate an object of the class `Circuit` via the `Circuit(n)` command, where n is the number of qubits in the system. Roughly, this command initializes an empty instruction list which we can then populate with gate instructions. We can create the tuples that populate this list via the `GATE(args)` method. Fig. 2.3 shows an example on how a circuit instance is initialized and evaluated by the simulators.

Adding to the gate set

Technically, there is no need to add any more gates to the set of Clifford gates already included since, as we will soon show, any other Clifford gate can be created by applying combinations of the $CNOT$, H , and S gates. However, finding a decomposition of a Clifford gate in terms of these three is not straightforward, and in fact, is an active area of research (9, 21, 22). For this reason, our software allows users to define their own Clifford gates by adding a new `apply_GATE(args)` method in the `Circuit` class. Evidently, the user will also

¹In computer programming, a *class* is a template for creating objects or *instances* (of the the class) that follow initial values, methods, and properties set by said class.

have to add the respective simulation rules for each of the simulators. We will show how to do this in their respective chapters. On the other hand, we expect that users will want to define their own non-Clifford gates frequently. We also allow them to do this, but the method is a bit different than for Clifford gates.

QASM

Given that throughout this thesis we will be comparing our work with Qiskit, we provide a method of transferring one representation of a quantum circuit to another. This is done via the *Open Quantum Assembly Language (OpenQASM)* (23), which aims to be a universal method of reading and writing quantum instructions. Most common software like Qiskit, Google's Cirq and Amazon's Braket are capable of reading and writing this file format. In our software, the method `from_qasm_file(file)` allows us to read quantum circuits in this format and translate them to our circuit object. The property `qasm` produces a QASM string that can either be saved into a file or read directly by other software.

Random circuit builder

The file `/simulator/randomcircuit.py` facilitates the creation of random Clifford or non-Clifford circuits. Instead of having to type `GATE(args)` for every gate one wants to add to the circuit, we provide a function `random_circuit(n, cliffords, noncliffords)` that given the number of qubits in the system and the desired number of Clifford and non-Clifford gates, it produces a random circuit with these characteristics. The circuit is constructed as follows:

1. Randomly select a gate from the set of available Clifford gates.
2. Randomly and without replacement, select however many qubits the gate chosen in (1) requires.
3. Add the gate to the list of instructions.
4. Repeat (1)-(3) for the desired number of Clifford gates in the circuit.

If the option to include non-Clifford gates is selected:

5. Randomly select a gate from the set of available non-Clifford gates.
6. Randomly and without replacement, select however many qubits the gate chosen in (5) requires.
7. Randomly select an index from the list of Clifford instructions, and add the non-Clifford instruction to the right of this index.
8. Repeat (5)-(7) for the desired number of Clifford gates in the circuit.

Summary of simulators

Fig. 2.4 shows the general structure of the simulators built in this thesis, as well as summarizes some of the most relevant characteristics of each simulator and its use cases.

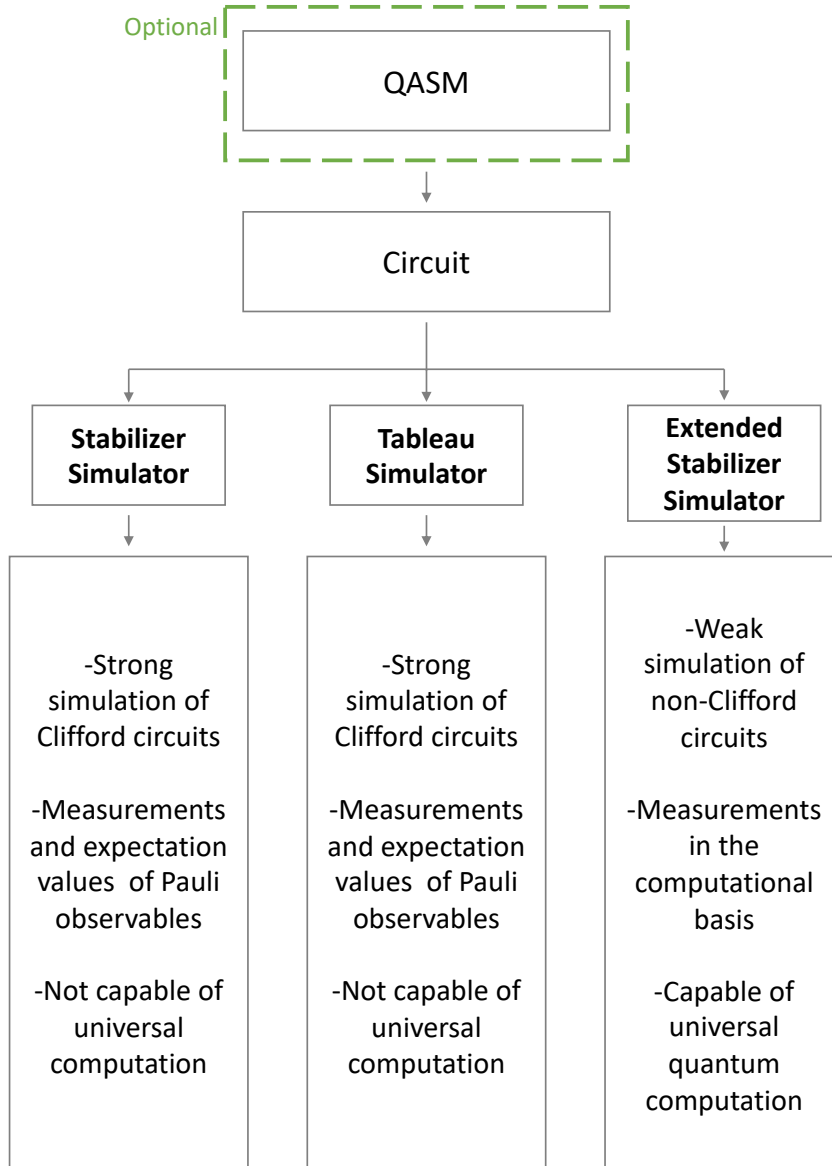


Figure 2.4: Summary and general structure of the three Python simulators built in this thesis.

2.8 Summary

In this chapter we introduced the definition of quantum states and qubits, which are the main units for quantum computation. We discussed measures of comparing two quantum states and how these relate to one another, which will be of relevance in Chapter 7. We showed that the allowed operations to manipulate qubits are unitary operators, and that these can be concatenated to form a quantum circuit that drive the state evolution. We discussed a subclass of the unitary operators called the Pauli operators which play a major role in the stabilizer formalism discussed in the next chapter as well as their binary representation apt for classical computers. We then showed how one can measure physical attributes of a quantum system resulting in classical outcomes that scientists can understand and process, and explained in mathematical terms the meaning of quantum superposition. Finally, we showed how circuits are represented in our software, how to create them and discussed how users can define their own gates. We also provided a summary of the algorithms developed in this thesis, their fundamental structure and reach.

3

Stabilizer formalism

In this chapter we discuss the theory behind the efficient classical simulation of Clifford circuits. The information presented in this chapter is the foundation to the algorithms we will study in Chapter 4 and Chapter 5. And in Chapter 6 we will build upon this theory to allow for an innovative simulator of quantum computers capable of universal quantum computation.

The stabilizer formalism was originally developed in the context of quantum error-correction (15, 24, 25), however Daniel Gottesman in his seminal paper *Heisenberg representation of quantum computers* applied this idea to the classical simulation of quantum computers and obtained an efficient classical simulation technique for simulating Clifford circuits. The key insight of his paper was to track a particular set of n operators that uniquely represent a state at a given time, called *stabilizers*. As we shall see in the next chapter when we discuss the algorithm, simulating circuits and measurements in the computational basis only require $\text{poly}(n)$ memory and time, instead of the $\text{exp}(n)$ scaling of the state vector simulator.

The theory of stabilizers described in this chapter of the thesis has important consequences to quantum error-correction, to the understanding of entanglement and non-locality (26, 27), and to the classical simulation of quantum computers (1, 2, 3, 18, 28).

3.1 Stabilizers

Here, we give the mathematical definition of stabilizer operators, show that these form a group with matrix multiplication as the group operation, and moreover that this group uniquely represents a single quantum state when the cardinality of the set is equal to the number of qubits of the system.

Definition 3.1.1 (Stabilizer). *In general, we say that an operator O stabilizes a state $|\psi\rangle$ if $O|\psi\rangle = |\psi\rangle$. That is, $|\psi\rangle$ is an eigenvector of O with eigenvalue $+1$.*

Now, consider only those that are unitary.

Claim 3. *The set of unitary operators that stabilize a state $|\psi\rangle$ in Hilbert space \mathcal{H} form a group with matrix multiplication as the group operation.*

Proof. To prove this statement we have to verify that the set of unitary operators that stabilize $|\psi\rangle$ satisfy: (i) the group contains the identity operator, (ii) the inverse of any

3.1 Stabilizers

unitary in the group is also in the group, (iii) the group operation is associative, (iv) the group is closed under matrix multiplication. Let us verify that this is indeed the case.

- (i) I is the unique unitary that stabilizes all states $|\psi\rangle$, i.e., $I|\psi\rangle = |\psi\rangle$ for all $|\psi\rangle$. And I is the identity of this group since $IU = UI = U$ for all $U \in G$.
- (ii) If U stabilizes a state, its inverse does as well and it is therefore part of the group. This is true since $U^{-1}|\psi\rangle = U^\dagger|\psi\rangle = U^\dagger U|\psi\rangle = |\psi\rangle$, where we recalled that the inverse of a unitary is its adjoint (Definition 2.1.4).
- (iii) By definition, matrix multiplication is associative.
- (iv) Note that for any two unitaries U, V which stabilize $|\psi\rangle$, the product VU also stabilizes $|\psi\rangle$ since $VU|\psi\rangle = V|\psi\rangle = |\psi\rangle$.

□

Let us impose more restrictions on the set of unitaries we are interested in and consider only those of the n -qubit Pauli group. Particularly, we will focus our attention on *abelian* subgroups S of \mathcal{P}_n , that is groups of n -qubit Pauli operators where all elements of the group commute with each other. With this new structure, we can make some additional statements.

Proposition 3.1.1. *Consider an abelian subgroup S of \mathcal{P}_n that stabilizes $|\psi\rangle$, then the following are true:*

- (i) $I \in S$ and $-I \notin S$.
- (ii) The Pauli operators of the form $\pm iP$ where $P \in P_n^*$ are not in S .
- (iii) For all $s \in S$, s cannot have phases $\pm i$.
- (iv) If $s \in S$, then $-s \notin S$.

Proof.

- (i) Both $\pm I$ are elements of \mathcal{P}_n , and it is clear to see that the identity operator stabilizes all states, while the negative identity stabilizes no states.
- (ii) Assume $-is|\psi\rangle = |\psi\rangle$ is true. Then, by the group properties, $(-is)^2$ should also stabilize $|\psi\rangle$. But $(-is)^2|\psi\rangle = -I|\psi\rangle$, and as shown in (i) the negative identity does not stabilize any state. We have arrived at a contradiction.
- (iii) Assume $-s \in S$ is true. Since S is a group, we must have that $-s = ss'$ for some other element $s' \in S$. The negative identity is the only operator that maps s to $-s$, thus $-I \in S$, which by (i) leads to a contradiction.

□

Proposition 3.1.1 shows that elements of S must be of the form $s = Q_1 \otimes Q_2 \otimes \dots \otimes Q_n$ where $Q_i \in \{I, \pm X, \pm Y, \pm Z\}$. Even better, by the properties of the tensor product given in Eq. (2.7), we can write the previous as $s = \pm P_1 \otimes P_2 \dots \otimes P_n$ with $P_i \in P^*$.

It is possible that a single subgroup S stabilizes multiple states. Take for example the trivial case where $S = \{I\}$. This subgroup stabilizes all states in the whole Hilbert space. More importantly, the set of states that are stabilized by S form a subspace, which we denote V_S .¹

Claim 4. *The projection operator to the subspace V_S is given by*

$$\Pi_S = \frac{1}{|S|} \sum_{s \in S} s, \quad (3.1)$$

where $|S|$ denotes the cardinality of the abelian subgroup S .

Proof. First, we show that Π_S is indeed a projector. For this, observe that

$$\begin{aligned} \Pi_S^2 &= \frac{1}{|S|^2} \sum_{l \in S} \sum_{r \in S} lr \\ &= \frac{1}{|S|^2} \sum_{s \in S} \sum_{r \in S} s \\ &= \frac{1}{|S|^2} |S| \sum_{s \in S} s \\ &= \frac{1}{|S|} \sum_{s \in S} s = \Pi_S, \end{aligned} \quad (3.2)$$

where in the second line we used the fact that $\sum_{l \in S} lr$ is simply the sum over all group elements except that the elements of the group are now in different order (hence the change of labels). Now, we verify that it projects unto the desired subspace, that is $\text{Im}(\Pi_S) = V_S$. First, notice that for some $s \in S$, we have that

$$s\Pi_S = \Pi_S, \quad (3.3)$$

due to Eq. (3.1). Now, for any pure state $|\psi\rangle$, we have that $s\Pi_S|\psi\rangle = \Pi_S|\psi\rangle$, implying that $\Pi_S|\psi\rangle \in V_S$ and $\text{Im}(\Pi_S) \subseteq V_S$. Then if $|\psi\rangle \in V_S$, we find that $\Pi_S|\psi\rangle = |\psi\rangle$ since Π_S is a sum over elements that stabilize $|\psi\rangle$. So $V_S \subseteq \text{Im}(\Pi_S)$, and therefore $\text{Im}(\Pi_S) = V_S$. \square

The dimension of V_S is given by

$$\dim(V_S) = \text{Tr}(\Pi_S) = \frac{1}{|S|} \sum_{s \in S} \text{Tr}(s) = \frac{2^n}{|S|}, \quad (3.4)$$

where we used the fact that the dimension of a subspace is equal to the trace of its projector, and Proposition 2.5.1, which states that all elements of S are traceless except for the identity operator I_{2^n} which has trace 2^n . All of this analysis has one important result if we think of the independent generators of S . By independent generators we mean

¹In the field of quantum error-correction, this subspace is known as the *codespace*.

3.2 Clifford group

the set of generators that if we were to remove one, the size of the group generated would decrease. We then have that due to the nature of the Pauli group and the fact that we are asking S to be abelian, each independent generator exactly doubles the size of the subgroup generated. Then, $|S| = 2^l$ where l is the number of generators of S and the set of generators is denoted by $\Lambda_S = \{g_1, g_2 \dots g_l\}$. We can now express Eq. (3.4) as

$$\dim(V_S) = \frac{2^n}{2^l} = 2^{n-l}. \quad (3.5)$$

Finally, the following theorem states the result we were after.

Theorem 3.1.1. *An abelian subgroup S of \mathcal{P}_n stabilizes a single state $|\psi\rangle$ if it contains n independent generators.*

Proof. If we set $l = n$ in Eq. (3.5) the dimension of V_S becomes one, and S only stabilizes one state. \square

Theorem 3.1.1 motivates the following definition.

Definition 3.1.2. $S(\psi)$ is the maximally commuting subgroup of n -qubit Pauli operators that stabilizes the unique n -qubit state $|\psi\rangle$.

By Theorem 3.1.1, $S(\psi)$ is generated by n elements.

To summarize, we have shown that it is possible to uniquely represent a state by n generators of the group that stabilizes the state, which is exponentially better than representing the same state by its 2^n complex coefficients. However, if we want to keep this improved representation as we manipulate the state, we require that the generators remain Pauli operators at all times, and that these generators all commute with each other. Unfortunately, as it will soon become apparent, we cannot ensure that these conditions are maintained at all times as arbitrary unitaries are applied to the state. In fact, we will only be able to apply a subclass of the unitary gates, called the *Clifford gates*, that meet these conditions.

3.2 Clifford group

To begin, let us discuss how the generators, or any element of $S(\psi)$ for that matter, transform as we apply an arbitrary unitary gate U to the n -qubit state $|\psi\rangle$. The resulting state $|\psi'\rangle$ is then

$$|\psi'\rangle = U|\psi\rangle = Ug|\psi\rangle = UgU^\dagger U|\psi\rangle = UgU^\dagger |\psi'\rangle, \quad (3.6)$$

showing that UgU^\dagger stabilizes the state $|\psi'\rangle$ for all $g \in \Lambda_{S(\psi)}$. Therefore, to obtain the stabilizer representation of the new state, one must conjugate all generators under U . Furthermore, to ensure that the generators remain Pauli operators upon a gate action, UgU^\dagger has to result in a Pauli operator. We are then looking for a subgroup of the unitary group that maps elements of \mathcal{P}_n to \mathcal{P}_n under conjugation. The generalization of this idea where operators of some class remain within the class after conjugation is called the *normalizer* and is defined as follows.

3.2 Clifford group

Operation	Input	Output
$CNOT_{i,j}$	X_i	$X_i \otimes X_j$
	Z_i	Z_i
	X_j	X_j
	Z_j	$Z_i \otimes Z_j$
H_i	X_i	Z_i
	Z_i	X_i
S_i	X_i	Y_i
	Z_i	Z_i

Table 3.1: Shows the action of the generators of the Clifford group when conjugating $(C_i P C_i^\dagger)$ Pauli X_i and Z_i operators. There is no need to specify the action of the gates when the single-qubit Pauli operator is Y , since $Y = iXZ$ and $C_i Y_i C_i^\dagger = iC_i X_i Z_i C_i^\dagger = iC_i X_i C_i^\dagger C_i Z_i C_i^\dagger$. For the $CNOT$ gate i, j indicate the control and target qubits respectively, while in the other two cases, i represents the qubit acted on.

Definition 3.2.1 (Normalizer). *Let G be a group, and S a subset of G . The normalizer of S with respect to G is given by*

$$N(S) \equiv \{g \in G \mid \text{for all } s \in S : gsg^{-1} \in S\}. \quad (3.7)$$

In this statement, we have glanced over the fact that if $g \in N(S)$ then so does $e^{i\theta}g$ for all θ , and so the normalizer is actually an infinite group. In this thesis, we will ignore the possibility of a global phase and treat the normalizer as finite. We are interested in the case where G is $\mathcal{U}(n)$, the unitary group of n qubits, and S is \mathcal{P}_n . This results in a normalizer called the *Clifford group*, denoted \mathcal{C}_n . This group is generated by the *Clifford gates*, the n -qubit version of the $CNOT$, H , and S gates.¹ The single-qubit version of these gates was already defined in Fig. 2.2, with $S = R_\phi$ when $\phi = \pi/2$. The n -qubit version consists of simultaneously acting with the identity operator I on the remaining $n - 2$ or $n - 1$ qubits.

By definition of the Clifford group and as seen in Table 3.2, the Clifford gates meet the condition that they map Pauli operators to Pauli operators under conjugation. Then, the remaining condition to verify is whether the Clifford gates preserve the commutation relations when conjugating Pauli operators. Suppose $[P, P'] = 0$ (i.e. $PP' = P'P$) for some $P, P' \in \mathcal{P}_n$, then

$$\begin{aligned}
[CPC^\dagger, CP'C^\dagger] &= CPC^\dagger CP'C^\dagger - CP'C^\dagger CPC^\dagger \\
&= CPP'C^\dagger - CP'PC^\dagger \\
&= CPP'C^\dagger - CPP'C^\dagger \\
&= 0,
\end{aligned} \quad (3.8)$$

where in the first line we used the fact that the Clifford gates are unitaries and follow $C^\dagger C = CC^\dagger = I$. Hence, if the initial Pauli operators commute, the resulting conjugated operators will also commute. This shows that the Clifford gates are indeed a suitable

¹The term ‘‘Clifford gates’’ technically refers to any gate of the Clifford group, here we abuse this terminology to refer to the generators of the group and elements of the group interchangeably.

choice (and the only one) to transform the state and maintain efficient simulation. By a similar argument to the one above, we can also show that Clifford gates also preserve anti-commutation relations, which will be relevant when we discuss the tableau algorithm of Chapter 5.

In literature, the result stating that it is possible to simulate Clifford circuits efficiently is known as the *Gottesman-Knill theorem*.

Theorem 3.2.1 (Gottesman-Knill Theorem). *A quantum computation is efficiently simulatable in a classical computer if all of the following conditions are met:*

1. *State preparation happens in the computational basis.*
2. *The quantum circuit only employs Clifford gates.*
3. *The observables measured are elements of the Pauli group.*

A surprising consequence of the Gottesman-Knill theorem is that it encompasses both the H and the $CNOT$ gates, which are capable of producing two of the most important phenomena in quantum physics: superposition and entanglement. Moreover, it can simulate any amount of entanglement in the system. To observe this, consider the following example.

Example 3.2.1. *Let U be the two-qubit circuit $U = CNOT_{0,1}H_0$ applied to the state $|00\rangle$. The Hadamard gate creates an even superposition of the 0th qubit, and the $CNOT$ entangles the qubits, resulting in the state $|\Phi^+\rangle \equiv |\psi\rangle = U|00\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ which is a Bell state, a 2-qubit maximally entangled state.¹*

Clearly, the computation meets all of the conditions established by the Gottesman-Knill theorem, and is therefore efficiently simulatable in a classical computer. This indicates that entanglement is not a sufficient condition for quantum exponential speed-ups! However, Ref. (29) showed that it is indeed a necessary condition. The reason behind the computational power of quantum computers is a highly discussed topic in the quantum computing community, and if the reader is interested in reading more about it, we refer them to Ref. (30).

3.3 Density matrix

Before we discuss how we can apply this knowledge in practice to classically simulate a quantum computer, it will prove useful to show how one can transform the stabilizer representation of the state to a more common representation like the density matrix. This will prove to be useful in the implementations of the algorithms in the chapters so we can compare the output of our simulations with Qiskit's state vector simulator.

Remark 3. *This computation is purely optional and not included in the scope of “efficient simulation of Clifford circuits”.*

¹States might possess different degrees of entanglement. The Bell states possess the highest degree of entanglement possible for a system of two qubits, that is why they are called “maximally entangled”.

Recall that the density matrix of pure state $|\psi\rangle$ is the projector to the subspace spanned by the state. In the stabilizer representation, the density matrix is simply Eq. (3.1) with $|S| = 2^n$, and $S = S(\psi)$. Then,

$$|\psi\rangle\langle\psi| = \frac{1}{2^n} \sum_{s \in S(\psi)} s. \quad (3.9)$$

However, computing the density matrix this way is computationally prohibited as it requires finding all 2^n elements of $S(\psi)$ by multiplying exponentially large matrices, and finalize by adding these together. Instead, we resort to a more clever method that only requires multiplying n matrices together. Clearly, this is still inefficient but possible to compute in a laptop computer as long as the number of qubits is relatively small.

Claim 5. *The density matrix of state $|\psi\rangle$ is given by*

$$|\psi\rangle\langle\psi| = \prod_{g \in \Lambda_{S(\psi)}} \frac{I + g}{2}. \quad (3.10)$$

Proof. Expanding the product we observe that

$$\begin{aligned} \prod_{g \in \Lambda_{S(\psi)}} \frac{I + g}{2} &= \frac{1}{2^n} (I + g_1) \dots (I + g_n) \\ &= I + g_1 + \dots + g_n + g_1 g_2 + \dots + g_1 g_n + g_2 g_3 + \dots + \\ &\quad g_2 g_n + \dots + g_1 g_2 \dots g_n, \end{aligned} \quad (3.11)$$

and recalling that all elements in $S(\psi)$ commute, i.e. $g_i g_j = g_j g_i$, the sum effectively runs through all combinations of the generators. Thus, Eq. (3.10) is equivalent to Eq. (3.9). \square

3.4 Summary

In this chapter we showed that a particular set of states can be represented by n Pauli operators which are the simultaneous +1-eigenspace of the state. The Gottesman-Knill theorem stated that this representation leads to efficient classical simulation as long as the gates in the circuit are elements of the Clifford group, and measurements are Pauli observables. We discussed that this theorem has interesting implications to the understanding of quantum computers as it discards entanglement as the sole reason for quantum advantage.

Now, that we have discussed the theory behind the efficient simulation of Clifford circuits, for the next three chapters of this thesis, we will present three algorithms based on the stabilizer theory, as well as how to implement these on a classical computer. We will also discuss our Python implementations and compare the complexity dictated by the theory to what is observed in practice. Let us begin with the stabilizer simulator of Ref. (1), and that which follows directly from the theory discussed in this chapter.

4

The Gottesman algorithm

In the previous chapter we discussed the theory that leads to a better simulation algorithm by tracking the generators of $S(\psi)$ as Clifford gates are applied to the state. Here, we will discuss the algorithm presented in Ref. (1) on how to classically simulate Clifford circuits in $\mathcal{O}(n)$ time, as well as measurements of Pauli observables and expectations values. Depending on the Pauli observable, the measurement might take $\mathcal{O}(n^2)$ or $\mathcal{O}(n^3)$ time. Moreover, the three algorithms discussed in this thesis use the Pauli-to-binary mapping of Section 2.5.2 since this representation helps simplify some of the algorithm's steps, provides a better understanding of the time and space complexity, and creates a representation more adequate for a computer. At the end of the chapter we will also discuss the software implementation and compare the theory with experiments run in our simulator.

4.1 Clifford circuits

As discussed in the previous chapter, when a single-qubit Clifford gate C_i , acting on qubit i , is applied to the state $|\psi\rangle$, the generators of its stabilizer group $S(\psi)$ transform as $g \mapsto C_i g C_i^\dagger$. Crucially, since each individual generator is of the form $P = \pm P_1 \otimes \dots \otimes P_n$ with $P_i \in P^*$, the conjugation can be rewritten as the local update $g \mapsto P_1 \otimes \dots \otimes C_i P_i C_i^\dagger \otimes \dots \otimes P_n$ only taking $\mathcal{O}(1)$ time per generator, and hence $\mathcal{O}(n)$ time to update all generators. The generalization to two qubit gates is straightforward, as the update simply acts on two locations of the tensor product instead of one (also $\mathcal{O}(1)$ time). The result of the local update for all Clifford gates and single-qubit Pauli operators is given in Table 3.2.

From an implementation point of view, checking which single-qubit Pauli operator is at position i , consulting the associated rule in the table, and making the update for each of the n generators of $S(\psi)$ is unnecessarily complicated. We can do better by using the Pauli-to-binary mapping shown in Section 2.5.2 which allows us to update all n generators in a single step by re-expressing the update rules of Table 3.2 as sums and products of matrix columns. This mapping also presents several other advantages which we mentioned above. Let us see how to construct this matrix and re-express the rules related to the action of the Clifford gates in the binary representation.

As shown in Section 2.5.2, an n -qubit Pauli operator P can be represented by a binary string $(z, x) = z_1 \dots z_n x_1 \dots x_n$ of length $2n$. Next, let us stack the n generators of $S(\psi)$, in this binary string format, on top of each other to create an $n \times 2n$ matrix or *stabilizer*

matrix. In this matrix, z_i now refers to the i^{th} column and x_i to the $i + n^{\text{th}}$ column. Note that this matrix does not encapsulate the signs of the generators, and so we must also create a binary column vector r of length n to contain these signs (entry 0 for positive phase, and 1 for negative phase).

Proposition 4.1.1. *Using the binary format of Pauli operators, the update rules of Table 3.2 can be re-written as*

- $CNOT_{i,j} : (z_i, z_j, x_i, x_j) \mapsto (z_i \oplus z_j, z_j, x_i, x_j \oplus x_i)$. And $r = r \oplus x_i z_j (x_j \oplus z_i \oplus 1)$.
- $H_i : (z_i, x_i) \mapsto (x_i, z_i)$. And $r = r \oplus x_i z_i$.
- $S_i : (z_i, x_i) \mapsto (z_i \oplus x_i, x_i)$. And $r = r \oplus x_i z_i$.

Here, the multiplication of two columns $z_i x_i$ is computed element-wise.

To see how this works in practice, let us consider the following example.

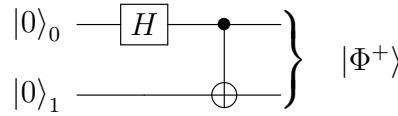


Figure 4.1: 2-qubit circuit containing a Hadamard gate acting on qubit 0 followed by a $CNOT$ gate with qubit 0 as the control and qubit 1 as the target. The output state is $|\Phi^+\rangle \equiv \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, one of the Bell states.

Example 4.1.1. *Consider the 2-qubit circuit given in Fig. 4.1. We will denote the top most qubit as qubit 0, and the one below as qubit 1. Initially, $S(|00\rangle) = \langle Z_0, Z_1 \rangle = \langle 1000, 0100 \rangle$. Stacking these two generators together and adding the vector of phases we obtain*

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \quad (4.1)$$

For the Hadamard acting on qubit 0, we swap the z_0 and the x_0 columns, and calculate the new phase, resulting in

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \quad (4.2)$$

Similarly, after applying the rules for the $CNOT$ gate with qubit 0 as the control, and qubit 1 as the target, the matrix becomes

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}. \quad (4.3)$$

4.2 Pauli measurements and expectation values

The resulting matrix above corresponds to generators $\langle X_1 X_0, Z_0 Z_1 \rangle$. Using the matrix representation of the Pauli operators defined in Eq. (2.22), and Eq. (3.10), we find that the density matrix corresponding the generators of the state is

$$|\psi\rangle\langle\psi| = |\Phi^+\rangle\langle\Phi^+| = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \quad (4.4)$$

which is indeed the density matrix of the maximally entangled state $|\Phi^+\rangle$.

A relevant characteristic of the stabilizer representation of states is that it cannot track the global phase of the state. To observe why this is the case, consider the the stabilizer representation of the single-qubit state $|0\rangle$. This state is stabilized by Z_0 , and hence its stabilizer matrix is $[0][10]$. One would think that the stabilizer matrix of $-|0\rangle$ should be $[1][10]$, however, this matrix represents $-Z_0$ which does not stabilize $-|0\rangle$ but $|1\rangle$. Similarly, it can be shown that neither of the other six stabilizer matrices on one qubit represent operators that stabilize $-|0\rangle$. Therefore, the stabilizer representation of states does not contain information about the global phase of the state. This caveat is not relevant in the simulation of Clifford circuits, but as will discuss in Chapter 6, the algorithm to simulate approximate Clifford circuits depends heavily on global phases.

4.2 Pauli measurements and expectation values

Here, we discuss how we can simulate measurements of Pauli observables with respect to the state $|\psi\rangle$ or compute their expectation value. As we shall see, this algorithm allows one to calculate the exact probability related to each measurement outcome, and so this simulator is capable of strong-simulation. Let us begin with measurements.

In Section 2.6, we discussed that projective measurements require a set of projectors (one per possible outcome) that add up to the identity. The outcomes corresponding to measurements of Pauli operators are their eigenvalues, which we know are ± 1 . The necessary projectors are then the projectors onto these eigenspaces, which we denote P_{\pm} . By the spectral decomposition of Pauli operators we have that $P = P_+ - P_-$, which by definition also form a complete set, so $I = P_+ + P_-$. Using these two relations to solve for P_+ and P_- we find

$$P_{\pm} = \frac{I \pm P}{2}. \quad (4.5)$$

The probability of obtaining outcomes ± 1 is given by Born's rule

$$\Pr(\text{outcome } \pm 1) = \text{Tr}(P_{\pm} |\psi\rangle\langle\psi|). \quad (4.6)$$

To find the explicit probability of the outcomes, we have to consider the commutation relations between the observable P we are trying to measure and the generators of the stabilizer state $|\psi\rangle$. There are two possible cases to consider. The first case is when there exists a generator $g \in \Lambda_{S(\psi)}$ such that $\{P, g\} = 0$, and the second case is when $[P, g] = 0$ for all $g \in \Lambda_{S(\psi)}$. As shown in Eq. (2.26), finding whether two Pauli operators in their binary

form commute, requires computing the symplectic inner product. Then, distinguishing between these two cases is done by computing the symplectic inner product between P and all n generators of $S(\psi)$, which can be accomplished in $O(n^2)$ time. Let us discuss these two cases separately.

4.2.1 P anticommutes with at least one generator

If P anticommutes with at least one generator, the first step in simulating the measurement of the observable, is to reduce the number of anticommuting generators to only one, which we will refer to as g_{ant} . Notice that if $S(\psi)$ contains two anticommuting generators g and g' then gg' commutes with P and remains a generator of $S(\psi)$. Hence, we can then replace g' with gg' and only keep g as the anticommuting generator. This procedure also generalizes for the case when there are more than two anticommuting generators. In the worst-case scenario the reduction to only one anti-commuting generator can be performed in $O(n^2)$ time.

Once this preparation step is complete, by plugging in the projector to the positive eigenspace in Eq. (4.6) we see that

$$\begin{aligned} \Pr(\text{outcome } +1) &= \text{Tr}(P_+ |\psi\rangle\langle\psi|) = \text{Tr}\left(\frac{I+P}{2} |\psi\rangle\langle\psi|\right) \\ &= \langle\psi| g_{ant}^\dagger \frac{I+P}{2} g_{ant} |\psi\rangle = \langle\psi| g_{ant}^\dagger g_{ant} \frac{I-P}{2} |\psi\rangle \\ &= \langle\psi| \frac{I-P}{2} |\psi\rangle = \Pr(\text{outcome } -1). \end{aligned} \quad (4.7)$$

Since probabilities must add up to one, we conclude that $\Pr(\text{outcome } +1) = \Pr(\text{outcome } -1) = 1/2$. The unnormalized post-measurement state after measuring observable P is $|\psi\rangle_\pm \equiv \frac{I \pm P}{2} |\psi\rangle$, where the sign depends on the observed outcome. However, we are interested in the stabilizer representation of the post-measurement state. To this end, note that g_{ant} can no longer be a generator of $S(\psi_\pm)$ since it anti-commuted with P . Furthermore, notice that P stabilizes the new state since

$$P|\psi\rangle_\pm = \frac{P \pm P^2}{2} |\psi\rangle = \frac{I \pm P}{2} |\psi\rangle = |\psi\rangle_\pm, \quad (4.8)$$

and so we can simply replace g_{ant} by $\pm P$ in the list of generators to obtain the accurate representation of $S(\psi_\pm)$.

To calculate the expectation value of P , we use Eq. (2.32) and the probabilities associated with each eigenvalue found in Eq. (4.7), which result in $\langle P \rangle_\psi = \frac{1}{2} - \frac{1}{2} = 0$.

4.2.2 P commutes with all generators

If P commutes with all generators of $S(\psi)$, it follows that P commutes with all elements of the group. This statement, together with the fact that $S(\psi)$ is a *maximally* commuting group of Pauli operators (Definition 3.1.2), implies that either $+P$ or $-P$ must be an element of the group. We cannot have that both are elements of $S(\psi)$, as this would contradict Proposition 3.1.1. Before we discuss how to find which of these is the correct element of the

group, we should discuss why this helps compute the measurement probabilities. Suppose $-P$ is an element of $S(\psi)$, then by Born's rule

$$\begin{aligned} \Pr(\text{outcome } -1) &= \text{Tr}\left(\frac{I - P}{2} |\psi\rangle\langle\psi|\right) \\ &= \text{Tr}\left(\frac{|\psi\rangle\langle\psi| + |\psi\rangle\langle\psi|}{2}\right) \\ &= 1, \end{aligned} \tag{4.9}$$

where we used the fact that $-P$ stabilizes $|\psi\rangle$. From this, it follows that $\Pr(\text{outcome } +1) = 0$. Notice that if $+P$ had been the element of the group instead of $-P$, by a similar analysis, we would have obtained the opposite result. Hence, the measurement outcome is deterministic, and only depends on the sign of P . Moreover the state remains unchanged after the measurement since $|\psi_{\pm}\rangle = \frac{I \pm P}{2} |\psi\rangle = \frac{|\psi\rangle}{2} + \frac{\pm P|\psi\rangle}{2} = |\psi\rangle$ so there is no need to update the list of generators. Finally, it is clear that the expectation value of the observable will result in a value of either $+1$ or -1 depending on the sign of P .

Determining whether $+P$ or $-P$ are part of the group consists of finding the generators that when multiplied yield one or the other. This problem is also easier in the binary representation as since it reduces to solving a system of equations mod 2 to find the coefficients α_g that satisfy

$$\sum_{g \in \Lambda_{S(\psi)}} \alpha_g (z, x)_g = \pm (z, x)_P \pmod{2}, \tag{4.10}$$

where $\alpha_g \in \{0, 1\}$ indicates whether the generator contributes to the sum or not, and $(z, x)_P$ is the string that corresponds to the Pauli observable P . However, since the sum over strings is not exactly operator multiplication, we also need to evaluate the symplectic inner product for all generators that contribute to the linear combination in order to arrive to the correct sign. Solving the system of linear equations requires inverting the $n \times 2n$ matrix. For simplicity we choose to do so via Gauss-Jordan elimination, which takes $\mathcal{O}(n^3)$ time.¹

4.3 Implementation

Here, we discuss the general structure and use of the *stabilizer simulator* based on the algorithm presented in this chapter. The implementation of the simulator is simple and quite true to the theory subsections mentioned above. The discussion will mostly refer to the `simulator/backend/stabilizer.py` file.

4.3.1 Main class

The `StabilizerState` class represents the state of an n -qubit quantum state at any given point during the simulation. The class consists of two `numpy` arrays that correspond to the stabilizer matrix of size $n \times 2n$ and the vector of signs of length n . There are two ways

¹There might be better algorithms for matrix inversion with runtimes of $\mathcal{O}(n^\omega)$ with $2 \leq \omega \leq 3$, but it is unclear whether these can be applied to modular systems of equations. Besides, in the next chapter we show a new algorithm on how to obtain a solution in time $\mathcal{O}(n^2)$ without requiring matrix inversion.

of initializing instances of this class. One is by feeding it a list of all n Pauli generators (each Pauli generator is itself a list of zeros and ones of length $2n$) and the list of signs, or alternatively via the `from_basis_state(basis_state)` method that given a basis state (list of zeros and ones of length n), calculates the generators of the group that stabilize the state, and constructs the stabilizer matrix and the vector of phases.

4.3.2 Run

The simulator counts with a `run(circuit)` function that given a Clifford circuit (instance of the `Circuit` class discussed in Chapter 2) evolves the initial state $|0^n\rangle$ (instance of the `StabilizerState` class) and produces the final state after the evaluation of the Clifford circuit. The function iterates through the list of circuit instructions and uses the information contained in the tuples to call the appropriate `apply_GATE(args)` method that performs one of the update rules given in Proposition 4.1.1 to the state's stabilizer matrix.

Once the circuit has been evaluated and we have obtained a final representation of the state, one can measure the state of qubits and ask for expectation values of Pauli observables using the `measure_pauli(pauli)` and `pauli_expectation(pauli)` functions. Here, the arguments to these functions are lists of binary numbers that represent an n -qubit Pauli operator.

For this and the simulator presented in the next chapter, we use sub-files like the `pauli.py` file that deal with the multiplication and summation of Pauli operators in their binary format, and `solvemod2.py` which performs the matrix inversion step in the case where the Pauli observable commutes with all generators.

4.3.3 Adding a Clifford gate

To add a desired Clifford gate to the allowed set of gates that the stabilizer simulator can understand and process, the user first has to compute $C_i X_i C_i^\dagger$ and $C_i Z_i C_i^\dagger$ for their Clifford gate C_i , and express the result as column operations. The resulting rules dictate the evolution of the stabilizer matrix, similar to Proposition 4.1.1. Next, one should add a new method called `apply_GATE(args)` in the `StabilizerState` class that contains these rules, as well as add a method `GATE(args)` to the `Circuit` class as discussed in Section 2.7.

4.3.4 Benchmark

We used Qiskit's state vector simulator to benchmark our implementation of the stabilizer simulator. The version of Qiskit used to produce the graphs is given in Appendix A. Moreover, the simulations presented in this and the next chapter were performed in a MacBook Pro with a 2.5GHz IntelCore i7 processor, and 16GB of RAM.

Fig. 4.2 compares the state vector simulator with the stabilizer simulator for a random Clifford circuit containing 100 gates. Here, we observe that, with the exception of the first 15 qubits, the runtime of the state vector simulator scales exponentially with the number of qubits, while the stabilizer simulator does not. We think that Qiskit implements several optimization routines that delay the exponential behavior for small qubit systems. Finally, we note that Qiskit only allows simulation up to 28 qubits since the computer does not have enough RAM to process bigger systems.

The figures demonstrating the algorithmic complexities are grouped with those of the

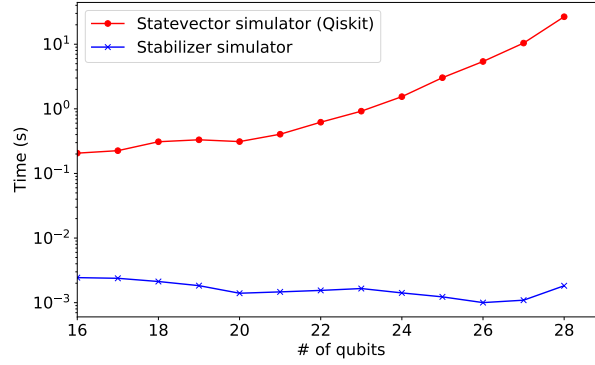


Figure 4.2: Logarithmic plot comparing the simulation time of Qiskit’s state vector simulator and our stabilizer simulation for a Clifford circuit containing 100 gates.

next chapter and shown in Fig. 5.1. Here, we observe that the stabilizer simulator, unlike the state vector simulator, is capable of evaluating circuits with 1000 qubits and 300 Clifford gates in a matter of seconds. Furthermore, the plots confirm the linear behavior for simulating Clifford gates and the cubic scaling for measurements of Pauli observables.

4.4 Summary

In this chapter we showed that by using the stabilizer representation of states in a binary matrix, one can efficiently simulate Clifford circuits in $\mathcal{O}(ln^2)$ time where l is the number of gates in the circuit, as well as perform measurements of Pauli observables in $\mathcal{O}(n^2)$ or $\mathcal{O}(n^3)$ time depending on the state and observable measured. We also observed that measurements reveal the probability with which an outcome happens, meaning that the simulation technique is capable of strong simulation.

Benchmarks confirmed that the simulation of Clifford circuits is indeed much faster than a state vector simulator. In spite of this gain, in practice, the $\mathcal{O}(n^3)$ time complexity of measurements might become restrictive since measurements are carried out quite often. Ref. (2) provides a better method of simulation that reduces the time complexity to $\mathcal{O}(n^2)$ by only increasing the memory requirements by a constant factor. This will be the topic of the next chapter.

5

The Tableau algorithm

The general idea of the algorithm with improved simulation cost presented in (2) is similar to the one before, except that now we will also track n conjugate generators that will help remove the matrix inversion step that shows up in Section 4.2.2, resulting in a better scaling. The conjugate generators are other n -qubit Pauli operators that together with the generators of $S(\psi)$ generate \mathcal{P}_n . To track these n conjugate generators, we again require $2n^2 + n$ bits, yielding a total of $4n^2 + 2n$ bits for both generators and conjugate generators. We then require twice the number of classical bits, but technically speaking, this is still $\mathcal{O}(n^2)$ bits of memory. Let us attach these conjugate generators in their binary form (along with its respective vector of phases) at the bottom of the stabilizer matrix we defined in the previous chapter, forming a $2n \times 2n$ matrix, or *tableau* (and a vector of phases of length $2n$).

Proposition 5.0.1. *For a stabilizer tableau, the following are always true:*

1. P_0, \dots, P_{n-1} generate $S(\psi)$, and P_0, \dots, P_{2n-1} generate \mathcal{P}_n .
2. All operators P_0, \dots, P_{n-1} commute with each other.
3. All operators P_n, \dots, P_{2n-1} commute with each other.
4. For all $i \in [n]$, P_i anticommutes with P_{i+n} .
5. For all $i, j \in [n]$ such that $i \neq j$, P_i commutes with P_{j+n} .

To see how this new tableau looks like in practice, consider the following example.

Example 5.0.1. *Consider the state $|00\rangle$. Its stabilizer group is $S(|00\rangle) = \langle Z_0, Z_1 \rangle$ and the conjugate generators are X_0 and X_1 . The tableau for this state is then*

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (5.1)$$

One can easily verify that the conditions established in Proposition 5.0.1 are indeed satisfied.

Remark 4. *In the original paper by Gottesman and Aaronson, they place the conjugate generators as the first n rows of the tableau, and the generators in the remaining rows.*

The action of the Clifford gates on this representation of the state changes the tableau in the same way as in Chapter 4, except that we now work with $2n$ rows instead of n . Since there is almost no change on gate actions, we will directly discuss measurements and expectation values of Pauli observables, which is where the improvement from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ is found.

5.1 Pauli measurements and expectations

Similarly to Chapter 4, the probability of the outcomes associated with a measurement of an observable forks into two cases.

5.1.1 P anticommutes with at least one generator

We can use the method described in the previous section to reduce the number of generators that anticommute with the observable to one. However, once the generator we want to keep has been chosen, we also have to multiply this with the conjugate generators that also anticommute with the observable. This is to ensure that Proposition 5.0.1 remains true at all times. One might be concerned that during this process replacing a generator g_k with $g_k g_j$ might change the anti-commutation relation with the conjugate generator q_k or viceversa, however, this is not the case. In fact,

$$\{g_k g_j, q_k\} = g_k g_j q_k + q_k g_k g_j = g_k g_j q_k - g_k q_k g_j = g_k g_j q_k - g_k g_j q_k = 0 \quad (5.2)$$

and

$$\{g_k g_j, g_k\} = q_k g_j g_k + g_k q_k g_j = q_k g_j g_k - q_k g_k g_j = q_k g_j g_k - q_k g_j g_k = 0, \quad (5.3)$$

where we have used the properties of Proposition 5.0.1. Hence, replacing the generators does not intervene with the commutation relations of their associated conjugate generators, or viceversa.

The measurement outcome is again uniformly distributed, but updating the post-measurement state is again slightly differently. If we let g_{ant} be the only generator that anticommutes with the observable, and q_{ant} its associated conjugate generator, then g_{ant} no longer stabilizes the new state and should be replaced. Then, we can simply update $g_{ant} \leftarrow \pm P$ and $q_{ant} \leftarrow g_{ant}$, which will give the correct form of the tableau.

5.1.2 P commutes with all generators

We know from the previous chapter that when P commutes with all generators, the outcome is deterministic and does not change the state, meaning that either $+P$ or $-P$ is an element of the group. Moreover, the explicit outcome is related to the phase of P , which can be determined by solving a system of equations mod 2 in $\mathcal{O}(n^3)$ time. The key idea of Ref. (2) is that the conjugate generators of $S(\psi)$ help simplify this computation and can determine a solution in $\mathcal{O}(n^2)$ time. Let us explain why this is so.

Consider a generator g and its associated conjugate generator q . Then, the value of coefficient c_g associated with this generator in the linear combination of Eq. (4.10) is

$$\begin{aligned}
c_g &= \sum_{h \in \Lambda_{S(\psi)}} c_h ((z, x)_q \odot (z, x)_h) \\
&= (z, x)_q \odot \sum_{h \in \Lambda_{S(\psi)}} c_h (z, x)_h \\
&= (z, x)_q \odot (z, x)_P,
\end{aligned} \tag{5.4}$$

where we initially used the commutation properties stated in Proposition 5.0.1 along with the fact that if the Pauli operators commute, then the value of the symplectic product is 0, and 1 if they anti-commute (discussed below Eq. (2.26)). And in the last line we used Eq. (4.10). Therefore, calculating the symplectic product between the measured observable and all of the conjugate generators of $S(\psi)$ (in their binary form) gives us the values of the coefficients in the linear combination. Evaluating the sum would then result in $+P$ or $-P$, in which case the measurement outcome would be $+1$ and -1 respectively.

Computing the coefficients this way takes $\mathcal{O}(n^2)$ time since there are n symplectic products to consider, and each evaluation of the product takes $\mathcal{O}(n)$ time. Summing the elements of the linear combination is $\mathcal{O}(n)$, which shows that this method has quadratic runtime instead of cubic. Theoretically, this may not seem like a significant advantage, but as shown in Fig. 5.1c it is in practice.

5.2 Implementation

Here, we discuss the general structure and use of the *tableau simulator* based on the algorithm presented in this chapter. The implementation of the simulator is quite similar to the one in the stabilizer simulator discussed in Chapter 4 since the tableau is an extension of the stabilizer matrix. The discussion refers to the `simulator/backend/tableau.py` file.

Main class and run method

The `Tableau` class represents the state of an n -qubit quantum state at any given time. Instances of the class are attributed with a stabilizer tableau (`numpy` matrix) of size $2n \times 2n$ and a vector of phases of length $2n$. The initialization and `run(circuit)` work in the same way as shown in the previous chapter, except that when the list of circuit instructions is read and the function `apply_GATE(args)` is called, it implements the rules of Proposition 4.1.1 but now on all $2n$ rows of the stabilizer tableau and the vector of phases.

Benchmark

Fig. 5.1a and Fig. 5.1b compare the behavior of the stabilizer and the tableau simulator when no measurements are performed. As expected, we can observe that the runtime of both algorithms is similar when considering only the evolution of the state dictated by a Clifford circuit. The tableau algorithm, however, takes a few seconds longer as it has to perform twice the amount of operations since it is also tracking n conjugate generators. For Fig. 5.1c, we considered random observables since it is hard to determine a non-trivial

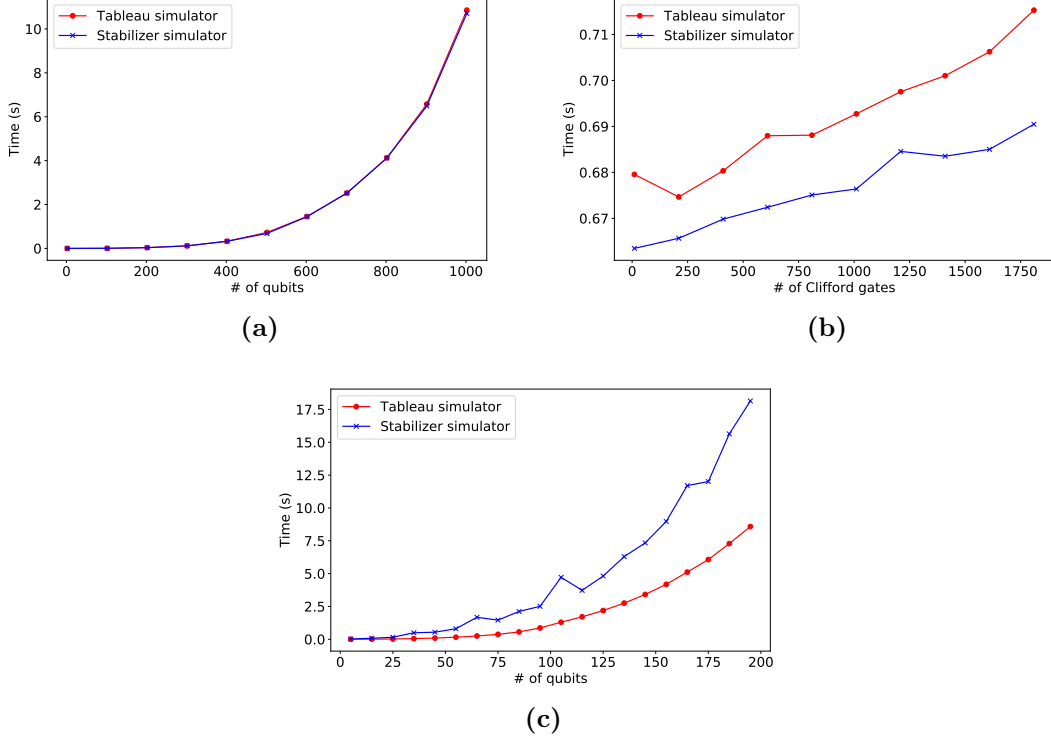


Figure 5.1: Demonstrates the behavior between the stabilizer and the tableau simulator. (a) and (b) show the simulation time of a random Clifford circuit when no measurements are performed. The data is averaged between 50 different random Clifford circuits. (a) uses 100 Clifford gates and (b) uses 500 qubits. (c) shows the simulation time for a Clifford circuit with 100 gates, and for each data point we considered 200 measurements of random Pauli observables.

Pauli operator that anticommutes with the generators of the final state. Then for 200 measurements we can expect around half to anticommute. Here, we observe that when measurements are performed, in the regime of thousands of qubits, the difference in time complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ has significant differences in practice.

5.3 Summary

In this section we have shown that it is possible to reduce the runtime of simulating Clifford and measuring the qubits in the computational basis from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ by only increasing the memory complexity by a constant factor of two. Although this is only a polynomial speedup, in practice, the improved algorithm does save noticeable amount of time when trying to simulate systems with hundreds of qubits and measurements.

It has also been shown in Ref. (31) that the simulation of Clifford circuits can be further improved to $\mathcal{O}(n \log n)$ time and memory by thinking of qubits as vertices on a graph (also known as *graph states*). However, we will not be discussing this simulation technique here

5.3 Summary

as it is quite different from the ones we have discussed thus far, and more importantly, is unrelated to the algorithm presented in the next chapter which is the focus of this thesis.

6

Simulation of low-rank stabilizer decompositions

Over the last two chapters we have shown that there are algorithms capable of efficiently simulating a particular class of quantum circuits called Clifford circuits, which consist only of $CNOT$, H , and S gates. One natural question to ask is whether one can extend this theory and efficiently simulate circuits capable of universal quantum computation. Ref. (32) demonstrated that adding any gate not already part of the Clifford group (with very few exceptions) to the set of Clifford gates, allows for universal quantum computation. Meaning that by adequately choosing a circuit, one is capable of producing any unitary gate, and therefore any quantum state.

Remark 5. *The statement by Ref. (32) assumes that one can apply any of the gates in the set an infinite number of times, if necessary.*

If we tried implementing some non-Clifford gate to the stabilizer simulation technique, the first problem one runs into is that Theorem 3.2.1 cannot apply, as it only admits Clifford gates and measurements of Pauli observables. In fact, it is conjectured that there is no classical algorithm capable of efficiently simulating an arbitrary quantum circuit, since this would imply that quantum computers can at best provide a polynomial speedup over their classical counterparts or that $\mathbf{P} = \mathbf{NP}$ (33). However, one can expect that for circuits that closely resemble a Clifford circuit, or states that closely resemble stabilizer states, we might be able to simulate them, not efficiently, but with much better runtime than a state vector simulator.

In the same paper, Aaronson and Gottesman (2) showed that it is possible to simulate arbitrary non-Clifford gates using the density matrix of the state at a cost of $\mathcal{O}(4^{bd})$ where d is the number of non-Clifford gates in the circuit and each gate acts on at most b qubits. Then, Ref. (28) demonstrated how to sample outcomes from the probability distribution of computational basis measurements of Clifford circuits with the addition of a non-Clifford T gate by decomposing the final non-stabilizer state into a linear combination of stabilizer states. They claimed the cost of this is $\tilde{\mathcal{O}}(2^{23t}t^3\omega^3)$, where t is the number of T gates in the circuit and ω is the size of the sampled bitstring. “ $\tilde{\mathcal{O}}$ ” means that polynomial factors are omitted. Here, they also mention that approximating any single-qubit gate with an error of 10^{-10} , would require approximately hundreds of T gates, which given the high simulation cost they could not do. Building on this, Bravyi, *et al.* (3), developed a method

to simulate any arbitrary non-Clifford gate and sample from the distribution of computational basis measurements with runtime that is $\mathcal{O}(knT + kn^2)$, where n is the number of qubits, T is a simulation hyperparameter, and k is the size of the stabilizer decomposition. The latter is exponentially large and its precise value depends on the types of gates in the circuit, however, for a non-Clifford circuit with s T gates it can be shown that $k \propto 1.1515^s$. This is a great improvement over the previous result. This is also important because if one wanted to approximate a unitary that would usually need hundreds of T gates, one could perhaps find a way of creating this unitary with less gates of some other type. This paper is the topic of discussion in this chapter of the thesis, and our goal is to create a fast Python simulator based on their analysis, discuss some of the open question of the paper, and experimentally test how approximations and unknowns affect simulation time.

In this chapter, we will first introduce extensions to the stabilizer simulation theory, starting with the notion of stabilizer decompositions and other related quantities. Next, we will discuss the simplest method on how to obtain stabilizer decompositions by decomposing the non-Clifford gates in the circuit into Clifford gates and evaluating these circuits using a new representation of the quantum state, called the *CH-form*. Crucially, this new representation is able to keep track of the global phase of the state, something that the algorithms of Chapter 4 and Chapter 5 could not do. Next, we will discuss how to compute relevant quantities using the CH-form of stabilizer states, such as their overlap with computational basis states, Pauli expectations, and their density matrices. We will then show how the previous pieces fit together and allow one to sample from the probability of computational basis measurements through Markov Chain Monte Carlo inference. It is important to mention that while Ref. (3) discusses both the strong and weak simulation of such circuits, here, we will only discuss the weak simulation technique. Next, we show how one can lower the size of the stabilizer decompositions and thus improve the simulation runtime through a technique called *randomized sparsification*. Finally, we discuss our implementation of the simulator and present a trick we devised that allows us to create stabilizer decompositions in less time than as suggested in Ref. (3).

6.1 Stabilizer decompositions

As mentioned before, one of the key ideas towards a better simulation of non-Clifford circuits is to decompose an arbitrary state $|\psi\rangle$ as a sum of stabilizer states. In this section, we provide a more formal definition of a stabilizer decomposition along with some related quantities, as well as the simplest method of constructing such a decomposition where the input is a non-Clifford circuit. In a later subsection, we will talk about how to use such a decomposition to sample outcomes from the probability distribution of computational basis measurements.

Mathematically, a stabilizer decomposition of a state $|\psi\rangle$ is

$$|\psi\rangle = \sum_{i=1}^m c_i |\phi_i\rangle, \quad (6.1)$$

where $|\phi_i\rangle$ are stabilizer states, $c_i \in \mathbb{C}$ are coefficients associated with each stabilizer state, and $m \in \mathbb{Z}^+$. Eq. (6.1) is said to be a stabilizer decomposition of state $|\psi\rangle$ with size m . Such a decomposition is always possible since the stabilizer states form an overcomplete

6.1 Stabilizer decompositions

basis of the Hilbert space. This is because computational basis states are stabilizer states¹, and because there are more stabilizer states that are not basis states, such as the Bell states. It might also be the case that the state $|\psi\rangle$ allows several distinct decompositions, and the smallest of these decompositions receives the name of *exact stabilizer rank*.

Definition 6.1.1 (Exact stabilizer rank). *The exact stabilizer rank $\chi(\psi)$ is the minimum m such that for a state $|\psi\rangle$, we may write it as in Eq. (6.1).*

Finding the decomposition with exact stabilizer rank is not straightforward², and in spite of it referring to the smallest decomposition, it usually results in extremely large ones. It can also be understood as the minimum cost of simulating state $|\psi\rangle$.

Similarly, the coefficients c_i of Eq. (6.1) are related to a measure indicating how far away the state is from being a stabilizer state, and the smallest value over all decompositions is given the name of *stabilizer extent*.

Definition 6.1.2 (Stabilizer extent). *The stabilizer extent $\xi(\psi)$ is the minimum value of $\|c\|_1^2$ over all possible decompositions of the state $|\psi\rangle$. Here, $\|c\|_1^2 = \sum_{i=1}^m |c_i|^2$.*

Here, we will refer to decompositions that satisfy the stabilizer extent as *optimal*.

Now that we have introduced stabilizer decompositions and other useful quantities, let us discuss a method on how to construct them starting from a non-Clifford circuit.

6.1.1 Sum-over-Cliffords

The main idea behind the creation of stabilizer decompositions is to decompose a non-Clifford circuit into several Clifford ones, evaluate each one of these to construct the stabilizer state $|\phi_i\rangle$, and finally add them all together. Let us begin by describing how to compute circuit decompositions.

Analogous to the quantum state case, any arbitrary circuit U can be decomposed as

$$U = \sum_j c_j C_j, \quad (6.2)$$

where C_j are Clifford gates, and $c_j \in \mathbb{C}$. Such a decomposition is always possible since the n -qubit Pauli operators form a basis of the $n \times n$ matrix space and $\mathcal{P}_n \subset \mathbb{C}_n$. Furthermore, the coefficients c_i provide a measure of how non-Clifford that decomposition is, and the minimal value over all decompositions is given the name of *stabilizer extent for unitaries*.

Definition 6.1.3 (Stabilizer extent for unitaries). *The stabilizer extent for unitaries $\xi(U)$ is the minimum value of $\|c\|_1^2$ between all possible decompositions of the unitary U .*

Computing the decomposition (optimal or not) of an n -qubit non-Clifford circuit in terms of Clifford gates for $n > 3$ is computationally prohibited. This is because one has to potentially consider all n -qubit unitaries of the Clifford group, which has been proved

¹One can obtain any basis state from the vacuum state $|0^n\rangle$ by applying Pauli X operators. For example $|01011\rangle = X_1 X_3 X_4 |00000\rangle$.

²The brute force approach to finding the stabilizer extent would require computing and comparing combinations of $2^{\mathcal{O}(n^2)}$ stabilizer states (2).

6.1 Stabilizer decompositions

to grow as $2^{O(n^2)}$ (34). A better approach, and the one used in this thesis, is to compute decompositions of each of the s non-Clifford gates V_p in the circuit, which we will assume to be one or two qubit unitaries. Computing decompositions and even optimal decompositions for these gates is possible since the Clifford group on two qubits contains 11520 Clifford gates (34). Another advantage is that we only have to compute the decomposition once and store it memory so it can be reused in the same or other circuits. Let us be more precise on how this takes place.

Let V_p be a non-Clifford gate with decomposition

$$V_p = \sum_i c_i^{(p)} K_i^{(p)}, \quad (6.3)$$

and U a circuit containing s non-Clifford gates. Without loss of generality, this circuit can be written as

$$U = C_s V_s C_{s-1} V_{s-1} \dots C_1 V_1 C_0, \quad (6.4)$$

where C_i are Clifford gates and V_j are non-Clifford gates. Substituting the decompositions of the non-Clifford gates as in Eq. (6.3), results in

$$U = \sum_{j_1, j_2, \dots, j_s} \left(\prod_{p=1}^s c_{j_p}^{(p)} \right) C_s K_{j_s}^{(s)} C_{s-1} \dots C_1 K_{j_1}^{(1)} C_0, \quad (6.5)$$

and in this case

$$\|c\|_1^2 = \prod_{p=1}^s \|c^{(p)}\|_1^2. \quad (6.6)$$

Furthermore, since every non-Clifford gate must be at least a combination of two Clifford gates, Eq. (6.5) results in a sum over $m \geq 2^s$ Clifford circuits. As mentioned before, to produce the stabilizer decomposition one then has to evaluate each Clifford circuit separately and add the resulting states $|\phi_i\rangle$ together. An important point in the evaluation of circuits is that these need to keep track of the global phases of each state (the coefficients c_i in the stabilizer decomposition), which we cannot do with the simulation techniques of the previous two chapters. Therefore, we are in need of better machinery that can distinguish between stabilizer states of the form $e^{i\theta}|\phi\rangle$ and $|\phi\rangle$. However, before we discuss this new simulation technique, let us point out the relation between optimal decompositions of circuits and individual non-Clifford gates.

It is easy to see from Eq. (6.2) and Eq. (6.5) that optimal decompositions of individual Clifford gates will result in worse decompositions of the circuit unitary than decomposing the circuit itself. This relation is summarized as

$$\xi(\psi) \leq \xi(U) \leq \prod_{p=1}^m \xi(V_p). \quad (6.7)$$

This will become relevant in Section 6.4 when we discuss the randomized sparsification procedure, which depends on the stabilizer extent to produce smaller decompositions. For the moment let us continue describing the algorithm for full-sized stabilizer decompositions

and in particular let us discuss how to simulate Clifford circuits that provide information about the global phase of the resulting state.

6.2 CH-form

To be able to build the stabilizer decomposition, we need a representation of stabilizer states where we can efficiently track the global phase of the state. As discussed in Chapter 4, the stabilizer matrix (and the tableau) are incapable of this. Here, we aim to discuss a representation of quantum states that contains information about the global phase, as well as an efficient algorithm capable of evolving the state while maintaining the representation.

6.2.1 Representation

To this end, we have to turn back to the properties of elements of the Clifford group. Recall that the Clifford group of $n > 1$ qubits is generated by the $CNOT$, S , and H gates. We can divide this group further into two categories, where the first category is the *control-type* (or *C-type*) unitaries created by taking products of the S and $CNOT$ gates, while the second category is the *Hadamard-type* (*H-type*) populated solely by the Hadamard gate. A layer of *C-type* gates can be jointly denoted by U_C , and U_H for a layer of Hadamard gates.¹ The control-type unitaries have the property that they satisfy $U_C|0^n\rangle = |0^n\rangle$, as well as map basis states to other basis states.

Ref. (21) showed that any Clifford circuit can be rewritten into a circuit of the form $-C-S-C-S-H-C-S-C-S$ where $-C-$ is a layer of $CNOT$ gates. This then shows that a stabilizer state can always be written as

$$|\phi\rangle = \omega U_C U_H |s\rangle, \quad (6.8)$$

where ω is the global phase and $|s\rangle$ is a computational basis state. This expression is referred to as the *CH-form* of $|\phi\rangle$.

The action of U_H is described by a binary vector ν of length n that indicates in which of the n qubits there is a Hadamard acting on it (1 for Hadamard gate, 0 for identity gate). Just like on the previous simulators, the action of the unitary U_C is encoded in a similar version of the Aaronson and Gottesman tableau defined by

$$U_C^\dagger Z_p U_C = \prod_{j=1}^n Z_j^{G_{p,j}} \quad \text{and} \quad U_C^\dagger X_p U_C = i^{\gamma_p} \prod_{j=1}^n X_j^{F_{p,j}} Z_j^{M_{p,j}}, \quad (6.9)$$

for all $p \in [n]$. It might be somewhat more difficult to see, but these are again the $2n$ stabilizers and conjugate stabilizers of $|s\rangle$ split into the three binary $n \times n$ matrices G, F, M and a vector of phases γ , whose entries are elements of \mathbb{Z}_4 .

Remark 6. *The stabilizer tableau defined by Eq. (6.9) is different as the one we have been using in previous sections. Here, the conjugation by the unitaries is of the form $U_C^\dagger P U_C$, while before, it was defined as $U P U^\dagger$, for $P = \{X, Z\}$. This seemingly small difference will be the cause of trouble once we discuss how to compute the density matrix from the CH-form of a state.*

¹“Layer” refers to a sequence of gates of some type applied to all or a subset of the qubits.

Eq. (6.8) and Eq. (6.9) together suggest that the CH-form of a state can be represented by the tuple $(G, F, M, \gamma, \nu, s, \omega)$. For the initial state $|0^n\rangle$, these parameters have values $G, F, M = I$, while $\gamma, s, \nu = 0^n$, and $\omega = 1$. The simulation procedure consists of tracking the elements of the tuple as gates are sequentially applied to the state.

Remark 7. *The CH-form of stabilizer state of Eq. (6.9) and hence the tuple that represents a state is not unique.¹ One should then be careful when making assumptions on how the CH-form of a state looks like at any given step during the simulation. To compare states in their CH-form, we show how to compute the density matrix from this representation.*

6.2.2 Phase-sensitive simulation

Now, let us discuss how Clifford gates to change the tuple. Suppose $|\phi\rangle$ is the state at some intermediate step in the simulation described by the tuple $(G, F, M, \gamma, \nu, s, \omega)$ and let Γ be an arbitrary Clifford gate, then

$$\Gamma|\phi\rangle = \omega\Gamma U_C U_H |s\rangle = \omega' U'_C U'_H |s'\rangle. \quad (6.10)$$

Depending on the gate that is applied, one obtains a different state described by the tuple $(G', F', M', \gamma', \nu', s', \omega')$. There are two cases to consider, that is when Γ is a C-type, in which the update can be performed in $\mathcal{O}(n)$ time, or H-type gate with $\mathcal{O}(n^2)$ time. Let's discuss the former first.

Applying a C-type gate

Before we discuss the explicit update rules on for control-type gates, let us introduce some terminology. If Γ is a C-type gate and if an operation is of the form ΓU_C , we say that Γ acts on U_C from the left, or “left-multiplies” it. Similarly, if the equation is $U_C \Gamma$ we say that Γ acts on U_C from the right, or “right-multiplies” it.

Let Γ be a C-type unitary. By Eq. (6.10) the gate can be “absorbed” directly into the C-layer resulting in a new state

$$|\psi'\rangle = \omega U'_C U_H |s\rangle, \quad (6.11)$$

where $U'_C = \Gamma U_C$. We then expect that only the tuple parameters G, F, M, γ associated with U_C change, and ν, s, ω should not. Furthermore, since Γ acts on U_C from the left, the tableaus of Eq. (6.9) associated with U_C change as

$$U_C^\dagger Z_p U_C \rightarrow U_C^\dagger \Gamma^\dagger Z_p \Gamma U_C \quad \text{and} \quad U_C^\dagger X_p U_C \rightarrow U_C^\dagger \Gamma^\dagger X_p \Gamma U_C. \quad (6.12)$$

It is evident from Eq. (6.11) that only left-multiplications are necessary when a C-type gate acts on the state, however, the right-multiplications will be relevant when we consider how the Hadamard acts on the state. For clarity, we will also derive these updates rules in this section. If Γ acts on U_C from the right, the tableaus become

$$U_C^\dagger Z_p U_C \rightarrow \Gamma^\dagger U_C^\dagger Z_p U_C \Gamma \quad \text{and} \quad U_C^\dagger X_p U_C \rightarrow \Gamma^\dagger U_C^\dagger X_p U_C \Gamma. \quad (6.13)$$

¹For example, consider the state $-i|1\rangle$. This can be represented by the two very different looking tuples $(G = 1, F = 1, M = 0, \gamma = 1, \nu = 0, s = 1, \omega = -1)$ or $(G = 1, F = 1, M = 1, \gamma = 3, \nu = 0, s = 1, \omega = -1)$.

One important thing to notice from Eq. (6.12) and Eq. (6.13) is that the conjugation of the Pauli operator is now of the form $C_i^\dagger P C_i$, which differs from the conjugations in previous chapters and that of Table 3.2. Table 6.2.2 now provides the correct conjugation, with the addition of the CZ Clifford gate rule. With the exception of the order of conjugation, the update procedure is the same as in the previous chapters, which we already showed take $\mathcal{O}(n)$ time per gate.

Operation	Input	Output
$CNOT_{i,j}$	X_i	$X_i \otimes X_j$
	Z_i	Z_i
	X_j	X_j
	Z_j	$Z_i \otimes Z_j$
H_i	X_i	Z_i
	Z_i	X_i
S_i	X_i	$-Y_i$
	Z_i	Z_i
$CZ_{i,j}$	X_i	$X_i \otimes Z_j$
	Z_i	Z_i
	X_j	$Z_i \otimes X_j$
	Z_j	Z_j

Table 6.1: Transformation of the Pauli X_i and Z_i operators under conjugation of the form $C_i^\dagger P C_i$ for four Clifford gates C_i .

Now that we know how the tableaus change when U_C is either left or right-multiplied, and how the Clifford gates act on Pauli operators upon conjugation of the form $C_i^\dagger P C_i$, we can now derive the explicit form of the tuple parameters.

Update rule derivation

The explicit action how left-multiplication by Γ changes the tuple depends on the specific C-type gate applied. For brevity, we will only derive the update rules produced by the CX gate, as the the rest of the rules follow from the same logic.

If $CX_{q,r}$ acts on U_C from the left, then the tableau changes according to $U_C^\dagger C X_{q,r}^\dagger Z_p C X_{q,r} U_C$ and $U_C^\dagger C X_{q,r}^\dagger X_p C X_{q,r} U_C$, where one should perform the operation for all $p \in [n]$. If $p = q$

then

$$\begin{aligned}
 U_C^\dagger CX_{q,r}^\dagger Z_q CX_{q,r} U_C &= U_C^\dagger Z_p U_C \\
 U_C^\dagger CX_{q,r}^\dagger X_q CX_{q,r} U_C &= U_C^\dagger X_q X_r U_C \\
 &= U_C^\dagger X_q U_C U_C^\dagger X_r U_C \\
 &= i^{\gamma_q} \prod_j X_j^{F_{q,j}} Z_j^{M_{q,j}} i^{\gamma_r} \prod_j X_j^{F_{r,j}} Z_j^{M_{r,j}} \\
 &= i^{\gamma_q + \gamma_r + 2 \sum_j M_{q,j} F_{r,j}} \prod_j X_j^{F_{q,j} \oplus F_{r,j}} Z_j^{M_{q,j} \oplus M_{r,j}} \\
 &= i^{\gamma_q + \gamma_r + 2(MF^T)_{q,r}} \prod_j X_j^{F_{q,j} \oplus F_{r,j}} Z_j^{M_{q,j} \oplus M_{r,j}},
 \end{aligned} \tag{6.14}$$

where on the second to last line, we obtained a phase by commuting the $Z_j^{M_{q,j}}$ operator with $X_j^{F_{r,j}}$. If $p = r$, then

$$\begin{aligned}
 U_C^\dagger CX_{q,r}^\dagger Z_r CX_{q,r} U_C &= U_C^\dagger Z_q Z_r U_C \\
 &= \prod_j Z_j^{G_{q,j}} \prod_j Z_j^{G_{r,j}} \\
 &= \prod_j Z_j^{G_{q,j} \oplus G_{r,j}} \\
 U_C^\dagger CX_{q,r}^\dagger X_r CX_{q,r} U_C &= U_C^\dagger X_r U_C.
 \end{aligned} \tag{6.15}$$

Taking both cases just discussed into consideration, the update rule for the left action of the $CX_{q,r}$ gate is

$$\mathcal{L}[CX_{q,r}] : \begin{cases} \gamma'_q \leftarrow \gamma_q + \gamma_r + 2(MF^T)_{q,r} \\ G'_{r,p} \leftarrow G_{r,p} \oplus G_{r,j} \\ F'_{q,p} \leftarrow F_{q,p} \oplus F_{r,p} \\ M'_{q,p} \leftarrow M_{q,p} \oplus M_{r,p}. \end{cases} \tag{6.16}$$

An important note to mention is that the updates are performed with the parameters before the gate was applied.

For the right action of the $CX_{q,r}$ gate, we have

$$\begin{aligned}
 CX_{q,r}^\dagger U_C^\dagger Z_p U_C CX_{q,r} &= CX_{q,r}^\dagger \left(\prod_j Z_j^{G_{p,j}} \right) CX_{q,r} \\
 &= \prod_j CX_{q,r}^\dagger Z_j^{G_{p,j}} CX_{q,r} \\
 &= Z_q^{G_{p,r}} \prod_j Z_j^{G_{p,j}} \\
 CX_{q,r}^\dagger U_C^\dagger X_p U_C CX_{q,r} &= CX_{q,r}^\dagger (i^{\gamma_p} \prod_j X_j^{F_{p,j}} Z_j^{M_{p,j}}) CX_{q,r} \\
 &= i^{\gamma_p} \prod_j CX_{q,r}^\dagger X_j^{F_{p,j}} CX_{q,r} CX_{q,r}^\dagger Z_j^{M_{p,j}} CX_{q,r} \\
 &= i^{\gamma_p} \prod_j X_j^{F_{p,j}} X_r^{F_{p,q}} Z_j^{M_{p,j}} Z_q^{M_{p,r}}.
 \end{aligned} \tag{6.17}$$

Notice that there is no need to consider the two cases ($p = q$ and $p = r$) separately, as these are both considered within the product. The update rules are therefore

$$\mathcal{R}[CX_{q,r}] : \begin{cases} G'_{p,q} \leftarrow G_{p,q} \oplus G_{p,r} \\ F'_{p,r} \leftarrow F_{p,r} \oplus F_{p,q} \\ M'_{p,q} \leftarrow M_{p,q} \oplus M_{p,r}. \end{cases} \tag{6.18}$$

The rules for the most common control-type gates is summarized as follows

$$\begin{aligned}
 \mathcal{R}[S_q] : \begin{cases} M'_{p,q} \leftarrow M_{p,q} \oplus F_{p,q} \\ \gamma'_q \leftarrow \gamma_q - F_{p,q} \end{cases} & \quad \mathcal{L}[S_q] : \begin{cases} M'_{q,p} \leftarrow M_{q,p} \oplus G_{q,p} \\ \gamma'_q \leftarrow \gamma_q - 1 \end{cases} \\
 \mathcal{R}[CZ_{q,r}] : \begin{cases} \gamma'_p \leftarrow \gamma_p + 2F_{p,q}F_{p,r} \\ M'_{p,q} \leftarrow M_{p,q} \oplus F_{p,r} \\ M'_{p,r} \leftarrow M_{p,r} \oplus F_{p,q} \end{cases} & \quad \mathcal{L}[CZ_{q,r}] : \begin{cases} M'_{q,p} \leftarrow M_{q,p} \oplus G_{r,p} \\ M'_{r,p} \leftarrow M_{r,p} \oplus G_{q,p} \end{cases} \\
 \mathcal{R}[CX_{q,r}] : \begin{cases} G'_{p,q} \leftarrow G_{p,q} \oplus G_{p,r} \\ F'_{p,r} \leftarrow F_{p,r} \oplus F_{p,q} \\ M'_{p,q} \leftarrow M_{p,q} \oplus M_{p,r} \end{cases} & \quad \mathcal{L}[CX_{q,r}] : \begin{cases} \gamma'_q \leftarrow \gamma_q + \gamma_r + 2(MF^T)_{q,r} \\ G'_{r,p} \leftarrow G_{r,p} \oplus G_{r,j} \\ F'_{q,p} \leftarrow F_{q,p} \oplus F_{r,p} \\ M'_{q,p} \leftarrow M_{q,p} \oplus M_{r,p} \end{cases}
 \end{aligned} \tag{6.19}$$

Applying a Hadamard

Now, let Γ be a Hadamard gate acting on qubit p , so $H_p|\phi\rangle = \omega H_p U_C U_H |s\rangle$. To express the new state in its CH-form, we have to commute it through the C -layer and hopefully “absorb” it into the H -layer, just like we did above. However, as we will show, the resulting operator after commuting through the C -layer is not always necessarily a Hadamard, and so cannot be absorbed. To begin, note that H_p can be expressed as $H_p = 2^{-1/2}(Z_p + X_p)$, so

$$H_p|\phi\rangle = \omega 2^{-1/2}(Z_p U_C U_H |s\rangle + X_p U_C U_H |s\rangle). \tag{6.20}$$

Commuting the X_p and Z_p operators through the C -layer and H -layer together with Eq. (6.9) results in

$$\begin{aligned}
 H_p|\phi\rangle &= \omega 2^{-1/2} (U_C \prod_{j=1}^n Z_j^{G_{p,j}} U_H |s\rangle + U_C i^{\gamma_p} \prod_{j=1}^n X_j^{F_{p,j}} Z_j^{M_{p,j}} U_H |s\rangle) \\
 &= \omega 2^{-1/2} U_C U_H (\prod_{j=1}^n X_j^{G_{p,j}\nu_j} Z_j^{G_{p,j}\bar{\nu}_j} |s\rangle + \prod_{j=1}^n X_j^{F_{p,j}\bar{\nu}_j} Z_j^{F_{p,j}\nu_j} Z_j^{M_{p,j}\bar{\nu}_j} X_j^{M_{p,j}\nu_j} |s\rangle) \\
 &= \omega 2^{-1/2} U_C U_H (\prod_{j=1}^n X_j^{G_{p,j}\nu_j} Z_j^{G_{p,j}\bar{\nu}_j} |s\rangle \\
 &\quad + (-1)^{\sum_{j=1}^n F_{p,j} M_{p,j} \nu_j} \prod_{j=1}^n X_j^{F_{p,j}\bar{\nu}_j \oplus M_{p,j}\nu_j} Z_j^{F_{p,j}\nu_j \oplus M_{p,j}\bar{\nu}_j} |s\rangle),
 \end{aligned} \tag{6.21}$$

where $\bar{\nu} \equiv 1 - \nu$, and in the last line the commutation of the X and Z operators gave rise to a phase. Noting that the Pauli X flips the basis state $|0\rangle$ to $|1\rangle$, and viceversa, while the Z adds a negative phase to $|1\rangle$, Eq. (6.21) can be rewritten as

$$H_p|\phi\rangle = \omega 2^{-1/2} U_C U_H [(-1)^\alpha |t\rangle + i^{\gamma_p} (-1)^\beta |u\rangle]. \tag{6.22}$$

Here, $t, u \in \{0, 1\}^n$ and are defined by

$$t_j = s_j \oplus G_{p,j}\nu_j \quad \text{and} \quad u_j = s_j \oplus F_{p,j}\bar{\nu}_j \oplus M_{p,j}\nu_j, \tag{6.23}$$

and

$$\alpha = \sum_{j=1}^n G_{p,j}\bar{\nu}_j s_j \quad \text{and} \quad \beta = \sum_{j=1}^n M_{p,j}\bar{\nu}_j s_j + F_{p,j}\nu_j (M_{p,j} + s_j), \tag{6.24}$$

where the sums are taken mod 2. Recovering the CH-form of $H_p|\phi\rangle$, in the case that $t = u$, is simple, as Eq. (6.22) would then read

$$H_p|\phi\rangle = \omega 2^{-1/2} [(-1)^\alpha + i^{\gamma_p} (-1)^\beta] U_C U_H |t\rangle, \tag{6.25}$$

which is in its CH-form. This indicates that only two modifications of the tuple are needed, $s' = t$ and $\omega' = \omega 2^{-1/2} [(-1)^\alpha + i^{\gamma_p} (-1)^\beta]$.

The case where $t \neq u$ requires more work. First, it is useful to rewrite Eq. (6.21) as

$$H_p|\phi\rangle = \omega 2^{-1/2} (-1)^\alpha U_C U_H [|t\rangle + i^\delta |u\rangle] \tag{6.26}$$

where $\delta = \gamma_p + 2(\alpha + \beta)$, and make use of the following claim.

Claim 6. $U_H [|t\rangle + (-1)^\delta |u\rangle]$ with $t \neq u$ is a stabilizer state, with a CH-form of its own, and hence can be written as

$$U_H [|t\rangle + (-1)^\delta |u\rangle] = \hat{\omega} W_C W_H |l\rangle. \tag{6.27}$$

With this at hand, then Eq. (6.26) becomes

$$H_p|\phi\rangle = \omega 2^{-1/2}(-1)^\alpha \hat{\omega} U_C W_C W_H |l\rangle, \quad (6.28)$$

which is a CH-form with

$$\omega' = \omega 2^{-1/2}(-1)^\alpha \hat{\omega}, \quad U'_C = U_C W_C, \quad U'_H = W_H, \quad \text{and } s' = l. \quad (6.29)$$

Note that the update of U'_C requires right-multiplying U_C with W_C , which is the reason why in the previous section we derived the right action rules for control-type unitaries.

Proof of Claim 6. The strategy of the proof consists of shaping $U_H[|t\rangle + (-1)^\delta |u\rangle]$ to its CH-form. To begin, we would like to construct a hermitian unitary V_C such that

$$U_H V_C U_H = \prod_{i \in [n] \setminus q : r_i \neq r'_i} CX_{q,i}, \quad (6.30)$$

where $r, r' \in \{0, 1\}^n$ are two strings that differ in at least one qubit, and q is the index of a qubit where the strings differ. This operation applied to basis states $|r\rangle$ and $|r'\rangle$, makes them differ only on qubit q . If we let $r = t$ and $r' = u$, then

$$U_H V_C U_H[|t\rangle + i^\delta |u\rangle] = |y\rangle + (-1)^\delta |z\rangle, \quad (6.31)$$

where $y, z \in \{0, 1\}^n$ and $y_i = z_i$ for all $i \in [n] \setminus q$. Moreover, due to the hermiticity of the Hadamard and V_C gates, one can rewrite Eq. (6.31) as

$$U_H[|t\rangle + (-1)^\delta |u\rangle] = V_C U_H[|y\rangle + i^\delta |z\rangle], \quad (6.32)$$

which begins to resemble a CH-form.

To construct V_C we start by defining the sets

$$\mathcal{V}_b = \{i \in [n] : \nu_i = b \text{ and } t_i \neq u_i\}, \quad (6.33)$$

for each $b \in \{0, 1\}$. Suppose that $\mathcal{V}_0 \neq \emptyset$, and let q be the first element of the set, then the Clifford circuit

$$V_C = \prod_{i \in \mathcal{V}_0 \setminus q} CX_{q,i} \cdot \prod_{i \in \mathcal{V}_1} CZ_{q,i}, \quad (6.34)$$

performs the desired operation while simultaneously respecting equation Eq. (6.31). In the case that $\mathcal{V}_0 = \emptyset$, by assumption \mathcal{V}_1 is non-empty, and q is then the first element of \mathcal{V}_1 . The circuit

$$V_C = \prod_{i \in \mathcal{V}_1 \setminus q} CX_{i,q}, \quad (6.35)$$

also produces the desired strings while respecting Eq. (6.31). A good hand rule, is that in either of the two cases, if $t_q = 1$ then $y = u \oplus e_q$ and $z = u$, where e_q is a string of zeros except at position q where $e_q = 1$. On the other hand, if $t_q = 0$, then $y = t$ and $z = t \oplus e_q$.

Now we proceed by shaping the right hand side of Eq. (6.32). Let us consider only the qubit where $y_q \neq z_q$. Then note that for that qubit we can write

$$H^{\nu_q} [|y_q\rangle + i^\delta |z_q\rangle] = \hat{\omega} S^a H^b |c\rangle \quad (6.36)$$

for some $a, b, c \in \{0, 1\}$. Solving for a, b, c , and $\hat{\omega}$ analytically is complicated, but by checking all 16 possible cases, one can come up with the following relations. If $y_q = 0$, then

$$\begin{aligned} a &= \delta \pmod{2} \\ b &= \nu_q(\delta + 1) + 1 \pmod{2} \\ c &= \lfloor (\delta + \nu_q)/2 \rfloor \pmod{2} \\ \hat{\omega} &= \sqrt{2}(\nu_q\delta + 1) + (1 + (-1)^{\lfloor \delta/2 \rfloor} i) \nu_q \delta \pmod{2} \end{aligned} \quad (6.37)$$

while if $y_q = 1$, then

$$\begin{aligned} a &= \delta \pmod{2} \\ b &= \nu_q(\delta + 1) + 1 \pmod{2} \\ c &= \lceil (\delta + \nu_q)/2 \rceil \pmod{2} \\ \hat{\omega} &= i^\delta \sqrt{2}(\nu_q\delta + 1) + (1 + (-1)^{\lfloor \delta/2 \rfloor} i) \nu_q \delta \pmod{2}. \end{aligned} \quad (6.38)$$

Finally, we obtain

$$U_H[|t\rangle + i^\delta |u\rangle] = \hat{\omega} (V_C S_q^a) (U_H H_q^{b \oplus \nu_q}) |l\rangle, \quad (6.39)$$

where the basis state $l_i = y_i = z_i$ for all $i \in [n] \setminus q$ and $l_q = c$. This has the exact form as the equation stated in Claim 6. We conclude the proof by noting that $W_C = V_C S_q^a$ and $W_H = H_q^{b \oplus \nu_q}$ in Eq. (6.29). \square

Now that we have discussed how to evolve stabilizer states in their CH-form depending on the Clifford unitary applied, we show how to compute quantities that are useful for debugging the Python simulator or computations that will be relevant on the subsection describing how to sample from the state $|\psi\rangle$ given its stabilizer decomposition.

6.2.3 Density matrix

To debug our Python simulator, we are interested in comparing the state after evaluation of the circuit with either the state vector simulator, or the simulators based on Ref. (1) and Ref. (2). However, as we mentioned before, the tuples that specify the state are not unique and so one must use some other technique for comparisons. We choose to compare these by computing the density matrix of the state.

Before we present how we can compute the density matrix we show how to derive the relation $GF^T = I$ which will be useful for computation of the density matrix. This relation stems from the preservation of the commutation relations of the conjugated Pauli X and Z operators as discussed in Chapter 3. That is, we require that

$$\{U_C^\dagger Z_p U_C, U_C^\dagger X_p U_C\} = 0 \quad (6.40)$$

for all $p \in [n]$, and

$$[U_C^\dagger Z_p U_C, U_C^\dagger X_q U_C] = 0 \quad (6.41)$$

for all $p \neq q$. Let us see what condition is required to satisfy the first equation. Using the definition of the anti-commutation relations (Eq. (2.2)) the equation becomes,

$$U_C^\dagger Z_p U_C U_C^\dagger X_p U_C + U_C^\dagger X_p U_C U_C^\dagger Z_p U_C = 0 \quad (6.42)$$

or

$$i^{\gamma_p} \prod_{j=1}^n X_j^{F_{p,j}} Z_j^{M_{p,j}} \prod_{l=1}^n Z_l^{G_{p,l}} = -i^{\gamma_p} \prod_{l=1}^n Z_l^{G_{p,l}} \prod_{j=1}^n X_j^{F_{p,j}} Z_j^{M_{p,j}}. \quad (6.43)$$

Expanding the left-hand side of the equation, we obtain that

$$\begin{aligned} i^{\gamma_p} \prod_{j=1}^n X_j^{F_{p,j}} Z_j^{M_{p,j}} \prod_{l=1}^n Z_l^{G_{p,l}} &= i^{\gamma_p} X_0^{F_{p,0}} Z_0^{M_{p,0}} \dots X_n^{F_{p,n}} Z_n^{M_{p,n}} Z_0^{G_{p,0}} \dots Z_n^{G_{p,n}} \\ &= X_0^{F_{p,0}} Z_0^{G_{p,0}} Z_0^{M_{p,0}} \dots X_n^{F_{p,n}} Z_n^{G_{p,n}} Z_n^{M_{p,n}} \\ &= i^{\gamma_p} (-1)^{\sum_{j=1}^n F_{p,j} G_{p,j}} \prod_{l=1}^n Z_l^{G_{p,l}} \prod_{j=1}^n X_j^{F_{p,j}} Z_j^{M_{p,j}} \\ &= i^{\gamma_p} (-1)^{(GF^T)_{p,p}} \prod_{l=1}^n Z_l^{G_{p,l}} \prod_{j=1}^n X_j^{F_{p,j}} Z_j^{M_{p,j}}, \end{aligned} \quad (6.44)$$

where we used the commutation and anti-commutation relations of the X and Z operators leading to a new phase. Hence, to satisfy Eq. (6.43) we find that $(GF^T)_{p,p} = 1$ for all p . A similar calculation for Eq. (6.41) shows that $(GF^T)_{p,q} = 0$ for all $p \neq q$, and so we conclude that

$$GF^T = I. \quad (6.45)$$

With this relation at hand, we can now discuss how to obtain the density matrix of the state from its CH-form. Recall from Eq. (2.11) that the density matrix of a pure state is $\rho = |\phi\rangle\langle\phi|$, then in the CH-form representation

$$\begin{aligned} |\phi\rangle\langle\phi| &= \omega U_C U_H |s\rangle\langle s| \omega^* U_H^\dagger U_C^\dagger \\ &= |\omega|^2 U_C U_H \left(\prod_{i=1}^n \frac{1}{2} (I + (-1)^{s_i} Z_i) \right) U_H^\dagger U_C^\dagger \\ &= \frac{|\omega|^2}{2^n} \prod_{i=1}^n (I + (-1)^{s_i} U_C U_H Z_i U_H^\dagger U_C^\dagger), \end{aligned} \quad (6.46)$$

where we used the fact that

$$|s\rangle\langle s| = \prod_{i=1}^n \frac{1}{2} (I + (-1)^{s_i} Z_i). \quad (6.47)$$

Recall from Table 3.2 that conjugation of Pauli X by H results in Z , and viceversa. Hence, depending on the value of U_H at position i , Eq. (6.46) will consist of terms of the form $U_C Z_i U_C^\dagger$ or $U_C X_i U_C^\dagger$, which are not quite the stabilizer tableaux of Eq. (6.9). This is the problem that we mentioned before, as there is no straightforward relation of obtaining one from the other. Thus we need to define new tableaux. Let $\tilde{U}_C \equiv U_C^\dagger$, then we are interested in the tableaux of the form

$$\tilde{U}_C Z_p \tilde{U}_C^\dagger = \prod_{j=1}^n Z_j^{\tilde{G}_{p,j}} \quad \text{and} \quad \tilde{U}_C X_p \tilde{U}_C^\dagger = i^{\tilde{\gamma}_p} \prod_{j=1}^n X_j^{\tilde{F}_{p,j}} Z_j^{\tilde{M}_{p,j}}, \quad (6.48)$$

with new matrices and vectors $\{\tilde{G}, \tilde{F}, \tilde{M}, \tilde{\gamma}\}$. From these definitions it follows that

$$\begin{aligned} U_C^\dagger Z_p U_C &= \tilde{U}_C Z_p \tilde{U}_C^\dagger \\ U_C^\dagger X_p U_C &= \tilde{U}_C X_p \tilde{U}_C^\dagger. \end{aligned} \quad (6.49)$$

To solve for the new tuple $\{\tilde{G}, \tilde{F}, \tilde{M}, \tilde{\gamma}\}$ in terms of $\{G, F, M, \gamma\}$, we rewrite the equation above as

$$\tilde{U}_C^\dagger U_C^\dagger Z_p U_C \tilde{U}_C = Z_p \quad (6.50a)$$

$$\tilde{U}_C^\dagger U_C^\dagger X_p U_C \tilde{U}_C = X_p. \quad (6.50b)$$

Starting with Eq. (6.50a) we can expand it as

$$\begin{aligned} \tilde{U}_C^\dagger U_C^\dagger Z_p U_C \tilde{U}_C &= \tilde{U}_C^\dagger \prod_j Z_j^{G_{p,j}} \tilde{U}_C = \prod_j (\tilde{U}_C^\dagger Z_p \tilde{U}_C)^{G_{p,j}} \\ &= \prod_j \prod_l Z_l^{\tilde{G}_{j,l} G_{p,j}} = \prod_l Z_l^{\sum_j \tilde{G}_{j,l} G_{p,j}} \\ &= \prod_l Z_l^{(G\tilde{G})_{p,l}}, \end{aligned} \quad (6.51)$$

where the sums over the Pauli operators are performed mod 2 and the sum over the complex phase is mod 4. Given that we desire this to be equal to Z_p for all $p \in [n]$, then $G\tilde{G} = I$ and therefore

$$\tilde{G} = G^{-1} = F^T. \quad (6.52)$$

where we used Eq. (6.45) to avoid the inverse.

For Eq. (6.50b) we have that

$$\begin{aligned}
 \tilde{U}_C^\dagger U_C^\dagger X_p U_C \tilde{U}_C &= i^{\gamma_p} \tilde{U}_C^\dagger \left(\prod_j X_j^{F_{p,j}} Z_j^{M_{p,j}} \right) \tilde{U}_C \\
 &= i^{\gamma_p} \prod_j (\tilde{U}_C^\dagger X_j \tilde{U}_C)^{F_{p,j}} (\tilde{U}_C^\dagger Z_j \tilde{U}_C)^{M_{p,j}} \\
 &= i^{\gamma_p} \prod_j (i^{\tilde{\gamma}_j F_{p,j}} \prod_l X_l^{\tilde{F}_{j,l} F_{p,j}} Z_l^{\tilde{M}_{j,l} F_{p,j}}) \left(\prod_l Z_l^{\tilde{G}_{j,l} M_{p,j}} \right) \\
 &= i^{\gamma_p + \sum_j \tilde{\gamma}_j F_{p,j}} \prod_l \prod_j X_l^{\tilde{F}_{j,l} F_{p,j}} Z_l^{\tilde{M}_{j,l} F_{p,j} + \tilde{G}_{j,l} M_{p,j}} \\
 &= i^{\gamma_p + (F\tilde{\gamma})_p + 2V_p} \prod_l X_l^{\sum_j \tilde{F}_{j,l} F_{p,j}} Z_l^{\sum_j \tilde{M}_{j,l} F_{p,j} + \tilde{G}_{j,l} M_{p,j}} \\
 &= i^{\gamma_p + (F\tilde{\gamma})_p + 2V_p} \prod_l X_l^{(F\tilde{F})_{p,l}} Z_l^{(F\tilde{M} + M\tilde{G})_{p,l}},
 \end{aligned} \tag{6.53}$$

where in the line before the last, we expanded the product over j and commuted the X and Z operators to position all the X 's together on the right, resulting in a new phase

$$V_p = \sum_l \sum_{r=1}^{n-1} \tilde{F}_{r,l} F_{p,r} \left(\sum_{g=0}^{r-1} \tilde{M}_{g,l} F_{p,g} + \tilde{G}_{g,l} M_{p,l} \right). \tag{6.54}$$

We require that Eq. (6.53) is equal to X_p for any given p , so the constraints to satisfy this statement can be written as the following vector equations

$$F\tilde{F} = I \tag{6.55a}$$

$$(F\tilde{M} + M\tilde{G}) = 0 \tag{6.55b}$$

$$\gamma + F\tilde{\gamma} + 2V = 0 \tag{6.55c}$$

where the 0's refer to the vector and matrix where all elements are zeros, respectively. Solving for \tilde{F} and \tilde{M} in Eq. (6.55a) and Eq. (6.55b) we obtain the relations

$$\tilde{F} = G^T \quad \text{and} \quad \tilde{M} = -G^T M F^T. \tag{6.56}$$

where we used Eq. (6.52) and Eq. (6.45). Unfortunately, solving Eq. (6.55c) is not straightforward as it requires solving an equation mod 4 which is problematic as inverses are not well-defined. We can work around this obstacle by re-expressing the problem as a system of two equations mod 2.

Claim 7. *The equation $\gamma + (F\tilde{\gamma}) + 2V = 0 \pmod{4}$ can be rewritten as*

$$\begin{aligned}
 \gamma_1 + F\tilde{\gamma}_1 &= 0 \pmod{2} \\
 \gamma_0 + F\tilde{\gamma}_0 + V + \omega &= 0 \pmod{2},
 \end{aligned} \tag{6.57}$$

where we have defined γ as $\gamma \equiv 2\gamma_0 + \gamma_1$ with $\gamma_0, \gamma_1 \in \mathbb{Z}_2$ and similarly for $\tilde{\gamma}$. Also, $\omega \in \mathbb{Z}_2$.

With this at hand and Eq. (6.45), one can easily show that

$$\tilde{\gamma}_0 = G^T \gamma_0 + V + \omega \quad \text{and} \quad \tilde{\gamma}_1 = -G^T \gamma_1, \quad (6.58)$$

and all the operations are mod 2.

Collecting the results of Eq. (6.56) and Eq. (6.52) we see that in order to obtain the tableau representation associated with terms of the form $U_C P U_C^\dagger$ for $P \in \{X, Z\}$, the parameters $\{\tilde{G}, \tilde{M}, \tilde{F}, \tilde{\gamma}\}$ have to be related to the original in the following way:

$$\begin{aligned} \tilde{G} &= F^T \\ \tilde{F} &= G^T \\ \tilde{M} &= -G^T M F^T \\ \tilde{\gamma} &= 2G^T(\gamma_0 + V + \omega) - G^T \gamma_1. \end{aligned} \quad (6.59)$$

This finally allows us to compute Eq. (6.46) and obtain the density matrix of the state given its CH-form.

Proof of Claim 7. Let us begin the proof by defining $\gamma \equiv 2\gamma_0 + \gamma_1$ where 0, 1 are superscripts and $\gamma_0, \gamma_1 \in \mathbb{Z}_2$. We also decompose $\tilde{\gamma}$ in the same way. Then, the exponent over the complex phase in Eq. (6.53) becomes

$$\gamma + F\tilde{\gamma} + 2V = 2(\gamma_0 + F\tilde{\gamma}_0 + V) + \gamma_1 + F\tilde{\gamma}_1 \pmod{4}. \quad (6.60)$$

Notice that that since $2(\gamma_0 + F\tilde{\gamma}_0 + V)$ is either 0 or 2 and the equation as a whole must be equal to 0 (mod 4), then

$$\gamma_1 + F\tilde{\gamma}_1 = 0 \pmod{2}, \quad (6.61)$$

which is the first equation of Claim 7. Next, since $\gamma_1 + F\tilde{\gamma}_1$ is a number $\omega \in 0, 1$ multiplied by two, we obtain that

$$2(\gamma_0 + F\tilde{\gamma}_0 + V + 2\omega) = 0 \pmod{4}, \quad (6.62)$$

which is equivalent to solving the equation

$$\gamma_0 + F\tilde{\gamma}_0 + V + \omega = 0 \pmod{2}. \quad (6.63)$$

This is the second equation of Claim 7. \square

6.2.4 Pauli expectation

Again, we ask how it is possible to compute the expected value of some n -qubit Pauli operator $\langle P \rangle_\psi$ for the normalized stabilizer state $|\psi\rangle$ in its CH-form. By Eq. (2.32) and the cyclicity of the trace, this is

$$\begin{aligned} \langle P \rangle_\phi &= \langle \phi | P | \phi \rangle \\ &= \langle s | U_H^\dagger U_C^\dagger \omega^* P \omega U_C U_H | s \rangle \\ &= |\omega|^2 \langle s | Q | s \rangle, \end{aligned} \quad (6.64)$$

6.3 Measurements in the computational basis

where Q is the Pauli operator arising from commuting P through the C and H layers. Therefore, the expectation value only depends on the Pauli Q . Notice that if Q contains at least one single-qubit X operator, the inner product between the basis states automatically becomes 0. If Q only consists of Z operators, the expectation is then ± 1 . This is in agreement with the stabilizer representation we discussed in the previous chapters.

6.2.5 Overlap

Let $|\phi\rangle$ be an n -qubit stabilizer state in its CH-form and b a basis state $b \in \{0, 1\}^n$. The overlap between the two is given by

$$\begin{aligned} \langle b|\phi\rangle &= \langle b|\omega U_C U_H|s\rangle \\ &= \langle 0^n|(\prod_{p=1}^n X_p^{b_p})U_C U_H|s\rangle \\ &= \langle 0^n|(\prod_{p=1}^n U_C^\dagger X_p^{b_p} U_C)U_H|s\rangle \\ &= \langle 0^n|QU_H|s\rangle, \end{aligned} \tag{6.65}$$

where in the second to last line we made use of Eq. (6.9), and Q is a new Pauli operator specified by the string b and the stabilizer tableau. Once the explicit form of Q is known, one can simplify the expression for the overlap by rewriting Q in its binary form (Eq. (2.24)) such that $Q = i^{-z \cdot x} Z^z X^x$, where z and x are binary strings of length n . By carefully considering both cases where $\nu_j = 0$ and $\nu_j = 1$ one finds that

$$\langle b|\phi\rangle = \langle 0^n|QU_H|s\rangle = 2^{-|\nu|/2} i^{-z \cdot x} \prod_{j:\nu_j=1} (-1)^{x_j s_j} \prod_{j:\nu_j=0} \langle x_j|s_j\rangle. \tag{6.66}$$

Computing Q as in Eq. (6.65) takes $\mathcal{O}(n)$ time, and evaluating the inner product of Eq. (6.66) also takes $\mathcal{O}(n)$ time. Therefore, computing the inner product between a basis state and a state in its CH-form takes $\mathcal{O}(n^2)$ time total.

6.3 Measurements in the computational basis

So far, we have shown that a state resulting from the evolution of a non-Clifford circuit can be represented as a sum of $m > 2^s$ stabilizer states, where s is the number of non-Clifford gates in the circuit. Here, we discuss how one can simulate computational basis measurements of the state using its stabilizer decomposition. Ref. (3) considers algorithms for strong simulation and weak simulation of the system. Here, we will only deal with weak simulation.

6.3.1 Metropolis-Hastings algorithm

The goal of weak simulation is to sample an outcome $x \in \{0, 1\}^n$ from the probability distribution

$$\Pr(x) = \frac{|\langle x|\psi\rangle|^2}{\|\psi\|^2}, \tag{6.67}$$

6.3 Measurements in the computational basis

where $|\psi\rangle = \sum_{i=1}^m c_i |\phi_i\rangle$ and $|\phi_i\rangle$ are stabilizer states. To accomplish the goal of sampling from this target distribution, we can use standard Markov Chain Monte Carlo sampling methods. We start by defining a Markov chain \mathcal{M} with state-space $\Omega = \{x \in \{0,1\}^n : \Pr(x) > 0\}$, and define the transition probabilities according to the following Metropolis-Hastings algorithm.

1. Select an initial basis vector x_0 .
2. For $j = 1, \dots, T$ perform the following Metropolis step:
 - (a) Draw a candidate $y = x_{j-1} \oplus e_a$, where a is an index of the n bit string chosen uniformly at random.¹
 - (b) Compute the ratio $\alpha = \Pr(x_{j-1})/\Pr(y)$.
 - (c) If $\alpha \geq 1$ the candidate is accepted and $x_j \leftarrow y$.
 - (d) If $0 < \alpha < 1$, set $x_j \leftarrow y$ with probability α , otherwise $x_j \leftarrow x_{j-1}$.
3. Output x_T .

It is a standard result from probability theory and stochastic processes that if the resulting chain is *ergodic*², then it is guaranteed that the steady-state distribution of the chain is Eq. (6.67). We will not discuss the proof of this statement here, but the reader can find a good explanation in Ref. (35)(Chapter 24). The constraint of ergodicity of the chain is necessary because non-ergodic chains do not have a unique steady-state distribution. The simulation algorithm of this section does not guarantee that the resulting Markov chains are ergodic, but let us put a pause on this. Now, let us discuss how to compute the ratio of probabilities in step (2.2) in terms of the stabilizer decompositions, and the computational cost of the sampling algorithm.

In step (2.2) of the algorithm mentioned above, one is required to compute the ratio of probabilities $\Pr(x_{j-1})/\Pr(y)$ at every step of the Metropolis algorithm. Explicitly, this is

$$\frac{\Pr(x_{j-1})}{\Pr(y)} = \frac{|\langle x_{j-1} | \psi \rangle|^2}{|\langle y | \psi \rangle|^2} = \frac{|\sum_{i=1}^m c_i \langle x_{j-1} | \phi_i \rangle|^2}{|\sum_{i=1}^m c_i \langle y | \phi_i \rangle|^2} \quad (6.68)$$

At the very beginning algorithm, computing the initial probability $\Pr(x_0)$ (up to a normalization constant) can be performed using Eq. (6.65) and Eq. (6.66) in $\mathcal{O}(mn^2)$ time. Similarly, computing the probability $\Pr(y)$ associated with the candidate y using the same equations would also require $\mathcal{O}(mn^2)$ time. However, if we store the m Pauli operators Q_l with $l = 1, \dots, m$ (increasing the memory cost by $\mathcal{O}(m)$) that arise from taking the overlap of all m states of the stabilizer decomposition in the computation of $\Pr(x_0)$, the cost of computing probabilities $\Pr(y)$ can be improved to $\mathcal{O}(mn)$ time. This is because y is simply x_i with the a th bit flipped, and hence updating the Pauli operator Q associated with x_i means updating only the single-qubit operator in the a th position of the tensor product of Q . Mathematically,

$$\langle y | \phi_i \rangle = \langle x \oplus e_a | U_C U_H | s \rangle = \langle 0^n | U_C^\dagger X_a U_C Q_{x_{j-1}} U_H | s \rangle, \quad (6.69)$$

¹In other words, the proposal distribution is the discrete uniform distribution over binary strings of length n .

²Roughly speaking, an ergodic Markov chain is a chain where every state is reachable from any other state (not necessarily in a single step).

6.3 Measurements in the computational basis

which can be computed in $\mathcal{O}(n)$ time. Updating all m Pauli operators Q_l this way results in a total time cost of $\mathcal{O}(mn)$ per Metropolis step.

Finally, the total time and memory costs, when considering all T steps of the Metropolis algorithm are

$$\text{Time: } \mathcal{O}(mn^2 + mnT) \quad \text{and} \quad \text{Memory: } \mathcal{O}(m). \quad (6.70)$$

6.3.2 Limitations

As discussed before, the sampling procedure only works when the Markov chains generated from the decompositions of $|\psi\rangle$ are ergodic. To elaborate more on this point consider the following instructional example.

Example 6.3.1. *Suppose the unnormalized state $|\psi\rangle$ is of the form*

$$|\psi\rangle = \alpha|00\rangle + \beta|11\rangle \quad (6.71)$$

for some $\alpha, \beta \in \mathbb{C}$. The goal of the algorithm in this case is to sample a string x from the probability distribution

$$\text{Pr}(x) = \frac{|\langle x | \Phi^+ \rangle|^2}{\|\Phi^+\|^2}. \quad (6.72)$$

The Markov chain has state-space $\Omega = \{00, 11\}$, and because of the way the Metropolis steps are defined, the only two possible candidates that can be drawn in step (1) are 01 and 10, neither which are in the state space of the chain. Therefore, no matter in which of the states (either 00 or 11) the chain begins, it will never be able to transition to the other state. Therefore, one will always be sampling from the wrong probability distribution.

In reality, the state space of the chain will be much larger than just two elements, but it is very much a possibility that sections of the chain will be disjoint and hence unreachable by the chain. A trick that might help in sampling from the correct probability is that for every sample, one picks the initial state uniformly at random. In the example above if one then obtains a large number of samples, the empirical distribution will approximate the target distribution. However, this is only a trick that works occasionally, because as stated before, non-ergodic chains might not have a unique steady-state distribution. A real advantage of choosing the initial state at random is that this will help produce independent samples. There is no downside of adding this step to the simulation, so we implement it.

Another limitation of the sampling algorithm is that the *mixing-time* or number of Metropolis steps T it takes for the chain to reach its steady-state distribution is unknown, however, it can be proved that it grows polynomially with the number of qubits. Roughly, its mixing-time cannot be properly characterized because the interference effects of quantum mechanics create unpredictable stabilizer decompositions, and hence the state space of the chain also varies unpredictably. In practice, one must then consider a sufficient number of steps T to ensure that the chain mixes. Evidently, this estimate of T should be obtained via experimental simulations for the type of circuit of interest. In Chapter 7 we will investigate the convergence to the steady state distribution for simulations of random non-Clifford circuits.

The last observation is that in Eq. (6.70) the number of stabilizer states m in a decomposition is an exponentially large variable. In the best-case scenario, that is when every non-Clifford gate in the circuit can be decomposed into two Clifford gates, m is $m = 2^s$ where s is the number of non-Clifford gates in the circuit. However, this is rarely the case and m is usually much larger. Now we explain how a procedure called *randomized sparsification* helps produce smaller decompositions in most scenarios.

6.4 Sparsification

Here, we explain randomized sparsification, which is a randomized technique that helps reduce the size of stabilizer decompositions by approximating the final state of the simulation. Even though the procedure still results in exponentially large decompositions, these are usually smaller than the best-case scenario of $m = 2^s$ as discussed before.

A key part of the procedure is to consider the *approximate stabilizer rank* of a state, instead of the exact stabilizer rank of Definition 6.1.1.

Definition 6.4.1 (Approximate stabilizer rank). *Consider a normalized state $|\psi\rangle$, and let $\delta > 0$. The approximate stabilizer rank $\chi_\delta(\psi)$ is the smallest decomposition between all states $|\Omega\rangle$ that satisfy $\|\Omega - \psi\| < \delta$. And $\|\cdot\|$ is the Euclidean norm.*

Clearly, $\chi_\delta(\psi) \leq \chi(\psi)$, however, we can find a different upper-bound on the approximate stabilizer rank using the stabilizer extent defined in Definition 6.1.2. Crucially, this bound motivates a randomized procedure which allows us to produce smaller decompositions by considering approximations of the final state $|\psi\rangle$.

Theorem 6.4.1 (Upper-bound on $\chi_\delta(\psi)$). *For a normalized state $|\psi\rangle$ with decomposition $|\psi\rangle = \sum_{i=1}^m c_i |\phi_i\rangle$, its approximate stabilizer rank is*

$$\chi_\delta(\psi) \leq 1 + \frac{\xi(\psi)}{\delta^2}. \quad (6.73)$$

From the Theorem 6.4.1 and Eq. (6.7) it follows that

$$\chi_\delta(\psi) \leq 1 + \delta^{-2} \prod_{p=1}^s \xi(V_p) \leq 1 + \|c\|_1^2 \delta^{-2}, \quad (6.74)$$

where now $\|c\|_1^2 = \prod_{p=1}^s \|c^{(p)}\|_1^2$ as in Eq. (6.6).

Eq. (6.74) and Definition 6.4.1 then show that there exists a state $|\Omega\rangle$ with a decomposition of size approximately $\|c\|_1^2 \delta^{-2}$ whose difference in Euclidean norm with state $|\psi\rangle$ is less than δ . Randomized sparsification then proposes to construct the state $|\Omega\rangle$ (with smaller decomposition) by sampling $k \propto \|c\|_1^2 \delta^{-2}$ states from the decomposition of state $|\psi\rangle$. Let us formalize this procedure.

Definition 6.4.2 (Randomized sparsification). *Let $|\psi\rangle$ be an n -qubit state with decomposition $|\psi\rangle = \sum_{i=1}^m c_i |\psi_i\rangle$ (does not have to be the one with minimum m). Applying randomized sparsification on this state consist of the following steps:*

1. Choose the approximation error tolerance δ .

2. Set α to be $\alpha = 30 \times 2^{-n} \delta^{-2}$.
3. Sample $k \approx \alpha \|c\|_1^2 \delta^{-2}$ stabilizer states $|\phi_i\rangle$ that form part of the decomposition of $|\psi\rangle$ (up to a phase), picking each with probability $|c_i|/\|c\|_1$.
4. Set $|\Omega\rangle$ to be the even superposition of the sampled stabilizer states.

And the claim of randomized sparsification, which we will prove in the next subsection, is the following.

Claim 8. *Let $|\psi\rangle$ be an n -qubit state with decomposition $|\psi\rangle = \sum_{i=1}^m c_i |\phi_i\rangle$, and assume m is large. Applying randomized sparsification to state $|\psi\rangle$ produces (with probability above 95%) a state $|\Omega\rangle$ with stabilizer decomposition of size k that satisfies $\|\Omega - \psi\| \leq \delta$ and $k < m$.*

Before proving Theorem 6.4.1 and Claim 8 let us consider an example to see how sparsification helps reduce the size of the stabilizer decomposition.

Example 6.4.1. *Consider a circuit U as in Eq. (6.4) where all s non-Clifford gates are T gates, acting on the 5-qubit basis state $|0^5\rangle$. Moreover, it is known that the optimal decomposition of the T gate is $T = (\cos(\pi/8) - \sin(\pi/8))I + \sqrt{2}e^{\frac{-i\pi}{4}} \sin(\pi/8)S$ with $\|c\|_1^2 = \xi(T) \approx 1.1715$ (3). Hence, the stabilizer decomposition (without sparsification) of the state $|\psi\rangle = U|0^5\rangle$ has size $m = 2^s$. Now, let us see for which values of s the sparsification procedure will produce a smaller decomposition. Say we choose $\delta = 0.2$, and hence $\alpha \approx 23$. These parameters then amount to sampling $k \approx (1.1715^s \times 23)/0.2^2$ states. Therefore, in the regime where $s \geq 11$, the sparsification procedure will produce a smaller decomposition.*

The example shows that even for systems with small number of qubits (relatively large α), it is generally a good idea to consider decompositions via randomized sparsification. For larger systems and circuits with more non-Clifford gates, the difference between m and k continues to increase and we benefit more from using sparsification. In the next chapter we will continue to use this example. Now, let us briefly revisit the algorithm for sampling measurement outcomes.

Measurements with sparsification

Now, the task of performing computational basis measurements can be rephrased as sampling outcomes from the probability distribution

$$P(x) = \frac{\langle x | \Omega \rangle}{\|\Omega\|^2} \quad (6.75)$$

for a state $|\Omega\rangle = \frac{\|c\|_1^2}{k} \sum_{i=1}^k |\phi_i\rangle$. The algorithm's complexity becomes

$$\text{Time: } \mathcal{O}(kn^2 + knT) \quad \text{and} \quad \text{Memory: } \mathcal{O}(k), \quad (6.76)$$

where most of the time $k < m$.

6.4.1 Proof of sparsification

Before jumping straight into the proof of Claim 8, we present the following definition and theorem.

Definition 6.4.3 (Stabilizer fidelity). *The stabilizer fidelity $\mathcal{F}(\psi)$ of a state $|\psi\rangle$ is given by*

$$\mathcal{F}(\psi) \equiv \max_{\phi} |\langle \phi | \psi \rangle|, \quad (6.77)$$

where the maximization is performed over normalized stabilizer states.

Moreover, when $|\psi\rangle$ is normalized, $0 \leq \mathcal{F}(\psi) \leq 1$ and $\mathcal{F}(\psi) = 1$ if and only if $|\phi\rangle = |\psi\rangle$.

Theorem 6.4.2 (Hoeffding's inequality (36)). *Let X_1, \dots, X_k be independent random variables where $0 \leq X_i \leq 1$, and let \bar{X} denote their mean. Hoeffding's inequality states that*

$$\Pr(|\bar{X} - \mathbb{E}(\bar{X})| \geq t) \leq 2e^{-2nt^2} \text{ or } \Pr(|\bar{X} - \mathbb{E}(\bar{X})| \leq t) \geq 1 - 2e^{-2nt^2} \quad (6.78)$$

With these at hand, let us prove Claim 8.

Proof of Claim 8. The structure of the proof will consist of constructing the states as mentioned, followed by demonstrating that there exists a state ψ' that satisfies $\|\psi' - \psi\| \leq \delta$, and finally that this is true with high probability.

Consider a normalized state $|\psi\rangle$ with decomposition $|\psi\rangle = \sum_{j=1}^m c_j |\phi_j\rangle$, where $c_i \in \mathbb{C}$ and $|\phi_i\rangle$ are stabilizer states. This state can be rewritten as

$$|\psi\rangle = \|c\|_1 \sum_{j=1}^m \frac{|c_j|}{\|c\|_1} e^{i\theta_j} |\phi_j\rangle = \|c\|_1 \sum_{j=1}^m p_j |W_j\rangle, \quad (6.79)$$

where $\theta_j \in [0, 2\pi)$, and we have defined $|W_j\rangle \equiv e^{i\theta_j} |\phi_j\rangle$ and $p_j \equiv |c_j|/\|c\|_1$. Notice that p_j are probabilities, and so we can define a new random variable $|w\rangle$ to be $|W_j\rangle$ with probability p_j . Then Eq. (6.79) can be rewritten as

$$|\psi\rangle = \|c\|_1 \mathbb{E}(|w\rangle), \quad (6.80)$$

and its normalization condition

$$\langle \psi | \psi \rangle = \|c\|_1^2 \mathbb{E}(\langle \omega_\alpha |) \mathbb{E}(|\omega_\beta\rangle) = \|c\|_1^2 \mathbb{E}(\langle \omega_\alpha | \omega_\beta \rangle) = 1. \quad (6.81)$$

Now, let us construct a new unnormalized state $|\Omega\rangle$ by drawing $k < m$ random states from this probability distribution and creating an even superposition. This is

$$|\Omega\rangle = \frac{\|c\|_1}{k} \sum_{l=1}^k |\omega_l\rangle, \quad (6.82)$$

where this particular choice of the coefficients will become apparent in just a moment. The first objective is to show that it is possible to have $\|\Omega - \psi\| \leq \delta$ for a state $|\Omega\rangle$ of the form given in Eq. (6.82). Notice that this problem is equivalent to verifying that

$$\mathbb{E}(\|\Omega - \psi\|^2) \leq \delta^2 \quad (6.83)$$

is true, since the expectation implies that there is at least one $|\Omega\rangle$ for which $\|\Omega - \psi\| \leq \delta$. The left hand side of Eq. (6.83) can be rewritten as

$$\mathbb{E}(\|\Omega - \psi\|^2) = \mathbb{E}(\langle \Omega | \Omega \rangle) - \mathbb{E}(\langle \Omega | \psi \rangle) - \mathbb{E}(\langle \psi | \Omega \rangle) + \mathbb{E}(\langle \psi | \psi \rangle), \quad (6.84)$$

so we have to compute the expectation values with the states we defined above, and show that the sum is indeed bounded by δ^2 . Unfortunately, since $|\Omega\rangle$ is not normalized $\langle \Omega | \Omega \rangle \neq 1$ and so computing the expectation value then requires a bit more work. Then,

$$\begin{aligned} \mathbb{E}(\langle \Omega | \Omega \rangle) &= \frac{\|c\|_1}{k^2} \mathbb{E}\left(\sum_{\alpha=1}^k \langle \omega_\alpha | \omega_\alpha \rangle\right) + \frac{\|c\|_1^2}{k^2} \mathbb{E}\left(\sum_{\alpha \neq \beta} \langle \omega_\alpha | \omega_\beta \rangle\right) \\ &= \frac{\|c\|_1^2}{k} + \frac{1}{k^2} k(k-1) \\ &\leq 1 + \frac{\|c\|_1^2}{k^2} \end{aligned} \quad (6.85)$$

where in the first line we used the linearity of expectation and in the second line Eq. (6.81). Next, we see that

$$\begin{aligned} \mathbb{E}(\langle \psi | \Omega \rangle) &= \frac{\|c\|_1^2}{k} \sum_{\alpha=1}^k \mathbb{E}(\mathbb{E}(\langle \omega_\alpha |) | \omega_\beta \rangle) \\ &= \frac{\|c\|_1^2}{k} \sum_{\alpha=1}^k \mathbb{E}(\langle \omega_\alpha | \omega_\beta \rangle) \\ &= \frac{1}{k} \sum_{\alpha=1}^k \langle \psi | \psi \rangle \\ &= 1 \end{aligned} \quad (6.86)$$

where we made use of properties of the expectation and Eq. (6.81). Plugging in Eq. (6.85) and Eq. (6.86) into Eq. (6.84) and noting that $\mathbb{E}(\langle \psi | \psi \rangle) = 1$ since $\langle \psi | \psi \rangle = 1$ results in

$$\mathbb{E}(\|\Omega - \psi\|^2) \leq \frac{\|c\|_1^2}{k}. \quad (6.87)$$

Therefore, if one chooses $k \geq \|c\|_1^2 / \delta^2$, Eq. (6.83) is satisfied, and so there exists at least one state $|\Omega\rangle$ that satisfies $\|\Omega - \psi\| \leq \delta$. This proves Theorem 6.4.1.

For the second part of the proof, we need to show that by choosing an appropriate k , sparsification will produce one of the states $|\Omega\rangle$ such that

$$\Pr(\|\Omega - \psi\|^2 < \delta^2) \approx 1. \quad (6.88)$$

We begin by defining random variables X_1, \dots, X_k as

$$X_\alpha = \|c\|_1 \operatorname{Re}(\langle \psi | \omega_\alpha \rangle), \quad (6.89)$$

each which is bounded as

$$|X_\alpha| \leq \|c\|_1 |\langle \psi | \omega_\alpha \rangle| \leq \|c\|_1 \mathcal{F}(\psi). \quad (6.90)$$

The empirical mean of these random variables is

$$\bar{X} = \frac{1}{k} \sum_{\alpha=1}^k X_{\alpha} = \text{Re}(\langle \psi | \Omega \rangle), \quad (6.91)$$

so

$$\begin{aligned} |\bar{X} - \mathbb{E}(\bar{X})| &= |\text{Re}(\langle \psi | \omega_{\alpha} \rangle) - \mathbb{E}(\text{Re}(\langle \psi | \Omega \rangle))| \\ &= |\text{Re}(\langle \psi | \omega_{\alpha} \rangle) - \text{Re}(\mathbb{E}(\langle \psi | \Omega \rangle))| \\ &= |\text{Re}(\langle \psi | \omega_{\alpha} \rangle) - 1|, \end{aligned} \quad (6.92)$$

where in the last line we used Eq. (6.86). From the triangle inequality, we can find that

$$|\Omega - \psi|^2 = \langle \Omega | \Omega \rangle - 1 + 2|1 - \text{Re}(\langle \Omega | \psi \rangle)|. \quad (6.93)$$

Solving for $|1 - \text{Re}(\langle \Omega | \psi \rangle)|$ and substituting in Eq. (6.92) yields

$$|\bar{X} - \mathbb{E}(\bar{X})| = \frac{|\Omega - \psi|^2 - \langle \Omega | \Omega \rangle + 1}{2}. \quad (6.94)$$

Finally, applying Hoeffding's theorem with $t = \delta/2$ and Eq. (6.94), we obtain

$$\Pr(|\Omega - \psi|^2 \leq \langle \Omega | \Omega \rangle - 1 + \delta^2) \geq 1 - 2 \exp\left(\frac{-k\delta^4}{8\|c\|_1^2 \mathcal{F}(\psi)}\right). \quad (6.95)$$

This probability does not yet have the necessary form of Eq. (6.88) because the term $\langle \Omega | \Omega \rangle$ is unknown. However, if we consider the average case where $\mathbb{E}(\langle \Omega | \Omega \rangle) = \delta^2 + 1$ (Eq. (6.85)), then

$$\Pr(|\Omega - \psi|^2 \leq \delta^2) \geq 1 - 2 \exp\left(\frac{-k\delta^4}{8\|c\|_1^2 \mathcal{F}(\psi)}\right) \quad (6.96)$$

now has the correct form.

Now, we only have to specify the stabilizer fidelity. Unfortunately, the stabilizer fidelity is hard to compute, as it requires evaluating the inner product of the state with all stabilizer states of the same dimension. However, an educated guess is that for an n -qubit state $|\psi\rangle$, the stabilizer fidelity is $\mathcal{F}(\psi) \approx 2^{-n}$, which arises from the fact that the overlap between two randomly selected states is inversely proportional to the dimension of the Hilbert space. By choosing the number of sampled states k to be $k \approx \alpha\|c\|_1^2 \delta^{-2}$ for some new hyperparameter $\alpha \in \mathbb{R}_+$, we can counteract the effect of a large stabilizer fidelity and ensure that the procedure succeeds with a chosen probability. Eq. (6.96) then reads

$$\Pr(|\Omega - \psi|^2 \leq \delta^2) \geq 1 - 2 \exp\left(\frac{-\alpha\delta^2 2^n}{8}\right). \quad (6.97)$$

Finally, lower-bounding the equation by a probability $p = 0.95$ and solving for α we obtain

$$\alpha \geq \frac{8 \log(\frac{2}{1-0.95})}{2^n \delta^2} = 30 \times 2^{-n} \delta^{-2}. \quad (6.98)$$

Therefore, choosing α this way result in $|\Omega - \psi| \leq \delta$ with probability above 95%. This concludes the proof. \square

An important distinction between our approach and that of Ref. (3), is that we introduce the new hyperparameter α so the procedure still works when $\mathcal{F}(\psi)$ is large. Ref. (3) however, simply assumes the simulation algorithm will be used with for large qubit systems. Although, as we shall soon discuss, the simulator available in Qiskit cannot deal with systems above 63 qubits.

Now, let us explain some of the most relevant features and design choices of our simulator.

6.5 Implementation

So far in this chapter we have described the many moving pieces towards the goal of simulating non-Clifford circuits and sampling outcomes from the probability distribution of computational basis measurements. Here, we discuss how these pieces all fit together, along with a neat trick that allows us to perform sparsification before computing the full-size stabilizer decomposition. We also show how these are implemented in our software.

6.5.1 Recap

The method of performing weak-simulation of a non-Clifford circuit as described by the theory is as follows. The first step is to decompose all non-Clifford gates into an exponentially large sum of Clifford circuits (with coefficients β_i), as discussed in Section 6.1.1. Then, all these circuits are efficiently simulated using the CH-form algorithm outlined in Section 6.2, which results in a stabilizer decomposition of size $m \geq 2^s$. One then applies sparsification which consists of sampling $k \approx \alpha \|c\|_1^2 \delta^{-2}$ states of the decomposition according to the probabilities associated with the coefficients c_i . This in return reduces the size of the decomposition and produces a new state that approximates the original one with high probability. Finally, this new decomposition of size k is used to sample measurement outcomes via a Metropolis-Hastings algorithm.

6.5.2 Sparsification grouped with Sum-over-Cliffords

The procedure, as just described, requires storing in memory all m circuits, as well as evaluating all of these to produce the stabilizer decomposition. If one lets l be the number of gates in the initial non-Clifford circuit, then creating the stabilizer decomposition would require $\mathcal{O}(mln^2)$ since we know that the phase-sensitive simulator takes at most $\mathcal{O}(n^2)$ time per Clifford gate. Only when the full decomposition of the state has been created, one performs sparsification to end up with a decomposition with k states. There is a method to improve these costs by at all times considering k circuits and states.

Consider the following procedure using the same notation as in Section 6.1.1.

1. Make an empty circuit with the same number of qubits as the initial non-Clifford circuit. Call this empty circuit “circuit B”.
2. For a gate in the circuit evaluate if it is a Clifford C_i or non-Clifford gate V_p .
3. If it is a Clifford gate add this gate to the circuit B, but if it is a non-Clifford gate then sample one gate K_i from its decomposition with probability $|c_i^{(p)}|/\|c^{(p)}\|_1$ and add K_i to circuit B.

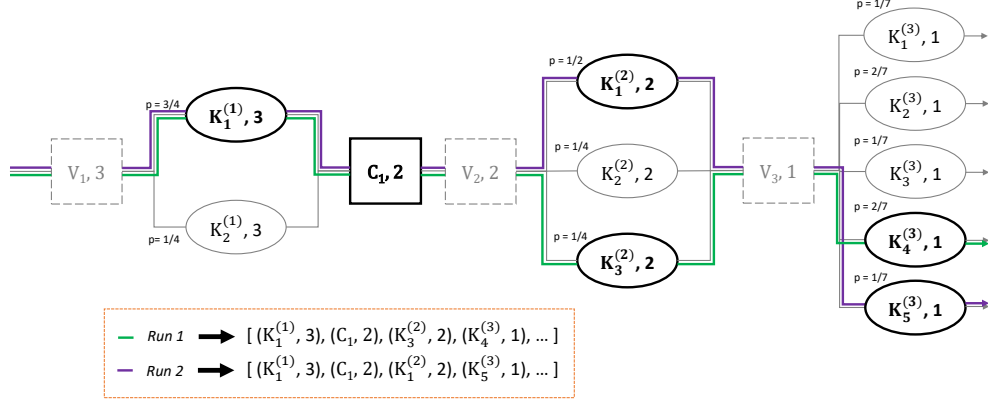


Figure 6.1: Illustration on how randomized sparsification is performed in practice. The figure shows two possible runs or paths of the sparsification procedure which result in two sampled Clifford circuits.

4. Repeat (2)-(3) for every gate in the circuit and when finished store circuit B in memory.
5. Repeat (1)-(4) k times.

At the end of the procedure one then ends up with k sampled Clifford circuits. This method takes $\mathcal{O}(kln^2)$ time instead of $\mathcal{O}(mln^2)$ time. Fig. 6.1 illustrates the procedure for sampling two Clifford circuits.

Notice that each of the paths can be associated with one of the terms of the full stabilizer decomposition with weight given by the product of the weights in the path. Therefore, sampling a state from the stabilizer decomposition is equivalent to sampling from the decomposition of individual non-Clifford gates in the circuit k times.

6.5.3 Circuit decompositions

The file that creates circuit decompositions using our joint sum-over-Cliffords and sparsification technique is found in `simulator/samplingch/statedecomposer.py`. Given a non-Clifford circuit object of the `Circuit` class, the parameter `alpha`, and a file containing the non-Clifford decompositions into Clifford gates (found in `cliffords/ncdecompositions.json`) the function `stabilizer_decomposition(circuit, alpha, gate_decompositions)` produces k Clifford circuits, each being an object of the `Circuit` class.

This file also evaluates all k circuits using the phase-sensitive Clifford simulator we will now describe to produce the stabilizer decomposition with k states of the class `CHState`.

6.5.4 CH-form simulation

The phase-sensitive simulator can be found in `simulator/backend/chtableau.py`. This file contains the `CHState` class whose instances contain the tuple $(G, F, M, \gamma, \nu, s, \omega)$. In practice, G, F and M are $n \times n$ `numpy` arrays, while γ, ν, s are arrays of length n , and ω is simply a complex number. One begins the simulation by running `run(circuit)` where the circuit is a Clifford circuit object from the `Circuit` class. The simulator reads the

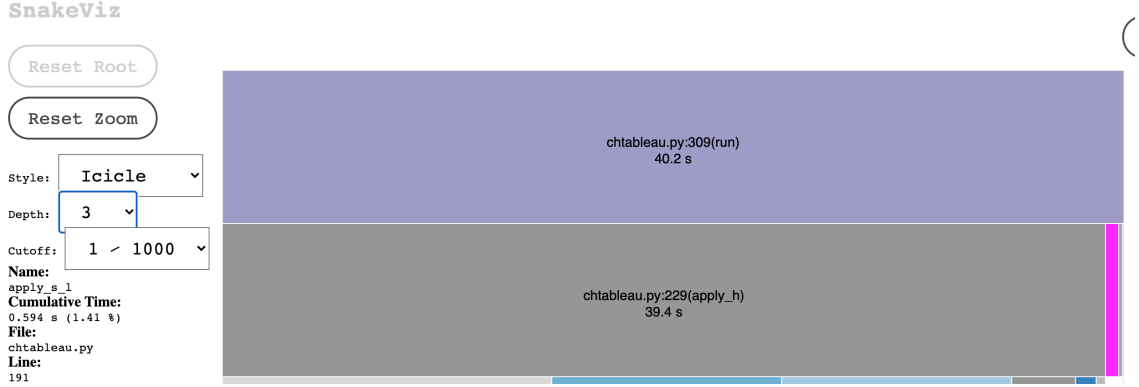


Figure 6.2: SnakeViz profile of the CH-form `run(circuit)` function that evaluates a 1000 qubit circuit with an equal number of H and S gates. Highlighted in pink, is the time taken by the `apply_s_1(qubit)` function, and its runtime is shown in the bottom left corner.

lists of circuits, and for every circuit computes the final state by applying the methods `apply_x_y(qubit)` and `apply_h(qubit)` depending on whether the gate is C -type or H -type respectively. Here, x refers to the specific C -type gate to be applied and y whether it is a left or right-multiplication. Internally, these two methods simply apply the rules found in Section 6.2.2 and Section 6.2.2 to the `numpy` matrices and vectors.

To compare the runtime of our Python functions, we use a third-party profiler called *SnakeViz*. More information about the software can be found in Appendix A. Fig. 6.3 shows the time taken to simulate the action of the Hadamard gates in a circuit with the same number of H gates as S gates. Observe that the time taken by the H gates is about 80 times greater than for C -type gates, which is due to the higher amount of instructions and operations that have to be carried out as was shown in Section 6.2.2.

6.5.5 Sampling

To begin collecting samples from the probability distribution of computational basis measurements for a non-Clifford circuit, one has to first decompose the circuit and evaluate using the phase-sensitive simulator. We have already discussed how each of these are performed within our software. The function that actually implements the Metropolis algorithm is `obtain_samples(samples, chstates, T)` found in `simulator/samplingch/sampling.py`. This function takes as arguments the number of samples to collect, the k states of the `CHState` class, and the number of Metropolis steps for the simulation.

The steps of the Metropolis algorithm for creating a random initial state, drawing candidates, and updating the string x_j is straightforward as it simply consists of manipulating binary strings. The important part of the algorithm is the calculation of the ratio of probabilities in step (2.2) that use the k tuples of matrices and arrays developed before. For this computation we use a supporting file called `overlap.py` that contains the `overlap_given_basis_sampling(args)` method that calculates the initial probability $\Pr(x_0)$ given a basis state and the tuple of a stabilizer decomposition. In addition, the method `overlap_given_pauli_sampling(args)` computes the probability $\Pr(y)$ for a candidate $y = x_j \oplus e_m$ given the Pauli operators corresponding to string x_0 , m , and the tuple of

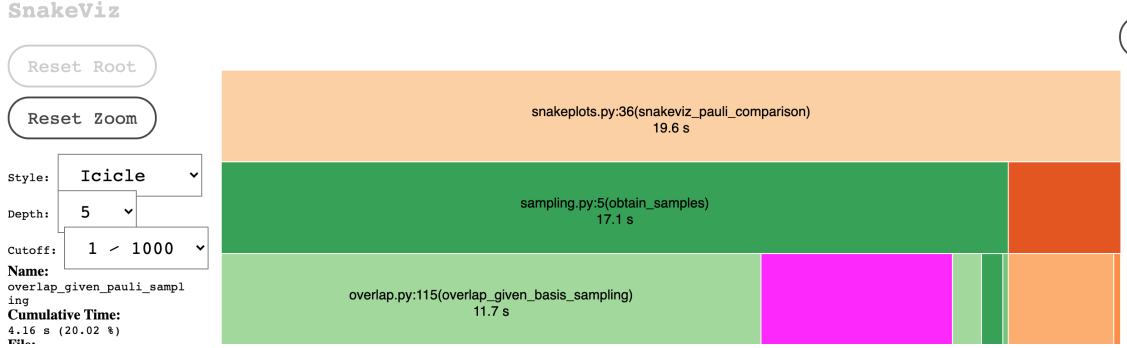


Figure 6.3: Shows the profile of a simulation of a non-Clifford circuit acting on 10 qubits where we collected 1000 samples, and for every sample performed only a single Metropolis step.

matrices of a stabilizer decomposition. Recall that the probabilities $\Pr(x_0)$ and $\Pr(y)$ are computed up to a normalization constant.

Fig. 6.3 shows that the time taken by the `overlap_given_basis_sampling` method takes 3 times more than `overlap_given_pauli_sampling` for a non-Clifford circuit acting on 10 qubits. It also shows that the most expensive part of the process is the sampling time. For this reason, we used JobLib, a parallel computation module to allow us to collect samples in parallel. More information about Joblib and its role in our implementation are given in Appendix A. The function that performs parallel sampling is called `obtain_samples(samples, chstates, T, cores)` where `cores` is the number of CPU cores to be used and can be found in `simulator/samplingch/parallelsampling.py`. We will show graphs of the advantage of the parallel implementation in Chapter 7.

6.5.6 Adding to the gate set

In this subsection we will refer to files inside the `cliffords/` folder.

We provide two methods of adding non-Clifford gates to the existing gate set. If the user already counts with a decomposition (optimal or not) of the gate they want to simulate, they can just write it on the `decomposition.json` file. Inside this file there are already a few examples on how to type it. However, the decomposition must be in terms of the operators $CNOT$, S , H , or CZ , which are the gates that the phase-sensitive simulator understands.

On the other hand, we also provide a file called `cliffdecomposer.py` where the user provides the matrix of the 1 or 2-qubit gate they want to simulate and our software produces the optimal decomposition. It does so via a brute-force algorithm which loads all 1 or 2-qubit Clifford unitaries (stored in the binary files `1q_cliffords.p` and `2q_cliffords.p`) and solves a convex optimization problem. After computing the optimal decomposition our software automatically adds it to the `decomposition.json` file.

Remark 8. Our software cannot calculate decompositions of gates acting on 3 or more qubits as simply storing all Clifford matrices would require gigabytes of storage. This is because the size of the Clifford group on 3 qubits consists on 92897280 matrices! (34).

In either case, the last thing the user must do is to add a `GATE(args)` in the `circuit.py` file class.

6.6 Summary

In the first four sections of this chapter we discussed how the algorithm of Bravyi, *et al.* (3) allows for weak-simulation of non-Clifford circuits. The simulation algorithm consists on creating stabilizer decompositions, which are linear combinations of stabilizer states. We described the simplest way of constructing these by decomposing each non-Clifford gate in the circuit in terms of Clifford gates, resulting in exponentially large sums of Clifford circuits. We called m the size of this decomposition. Each Clifford circuit was then evaluated using a phase-sensitive Clifford simulator that can crucially keep track of the global phases of the state and resulted in a stabilizer decomposition of size m . We saw that computing the stabilizer decomposition takes $\mathcal{O}(mln^2)$ time and $\mathcal{O}(m)$ memory, where l is the total number of gates in the initial non-Clifford circuit. Before using the decomposition to sample from the probability of computational basis measurements, they proposed to reduce the size of the decomposition to one of size k (with $k < m$) by sampling $k \approx \|c\|_1^2 \delta^{-2}$ states from the decomposition according to the probability associated with each state's coefficient. Finally, we discussed how to sample measurement outcomes in $\mathcal{O}(knT + kn^2)$ time and $\mathcal{O}(k)$ memory. Two short-comings of the Metropolis algorithm is that the parameter T , related to the mixing-time of the Markov chain, is said to be polynomial in n but its exact value is not known. Also, the simulation will only succeed with high probability when considering large qubit systems.

We showed how to improve two features of the algorithm just described. First, is that we introduced a new simulation parameter α that counteracts the case where the stabilizer fidelity can be small. This change implied sampling $k \approx \alpha \|c\|_1^2 \delta^{-2}$ states instead of $k \approx \|c\|_1^2 \delta^{-2}$ states as in the original algorithm, with the advantage that simulations with small number of qubits will now succeed with high probability as well. Second, is that we proposed a method of grouping the sum-over-Clifford procedure together with sparsification to reduce the time it takes to compute stabilizer decompositions. We showed that creating decompositions of size k could be performed in $\mathcal{O}(kln^2)$ time and $\mathcal{O}(n)$ memory, instead of the regular procedure of $\mathcal{O}(mln^2)$ time and $\mathcal{O}(m)$ memory.

In the last section of this chapter we discussed our implementation of the algorithm, along with profiles of some of our functions tested in real-life scenarios. Furthermore, we also highlighted our simulators' versatility, which allows user to define and simulate any non-Clifford gate they want, as long as a decomposition into Clifford unitaries is provided.

Validation & Benchmark

In the previous chapter we discussed that the algorithm of Ref. (3) succeeds with high probability when both the number of qubits in the system to be simulated is large¹ and when one considers a sufficient number of Metropolis steps to ensure the Markov chain thermalizes. The algorithm describes “success” as being able to sample measurement outcomes from a state $|\Omega\rangle$ such that $\|\Omega - \psi\| < \delta$ for the true state $|\psi\rangle = U|0^n\rangle$ and a chosen error tolerance δ . However, as we shall see, this exact statement cannot be verified experimentally.

In this chapter we develop a method of validating the simulation algorithm, as well as demonstrate that our Python implementation discussed in Section 6.5 satisfies the criteria. Moreover, we will also seek to provide practically oriented estimates on simulation hyperparameters to speedup computation. Finally, we compare the runtime of our extended stabilizer simulator with IBM’s implementation found in their quantum software development kit (Qiskit).

7.1 Validation

Let us restate the main goal of weak-simulation and what is meant when it is said that simulation “succeeds”.

The task of weak-simulation is the following. Given a non-Clifford circuit U , the goal is to sample outcomes from the probability distribution of computational basis measurements $P(x)$, where $P(x) \propto \langle x|\Omega\rangle$, and $|\Omega\rangle$ is a state that satisfies $\|\Omega - \psi\| \leq \delta$ with high probability. Here, $|\psi\rangle = U|0\rangle$ is the true state and δ is the chosen error tolerance. Then, the simulation succeeds if one can sample from $P(x)$ and $|\Omega\rangle$ is a state that δ -approximates the true state $|\psi\rangle$. Otherwise, the simulator fails.

To verify whether the simulator actually succeeds one has to check that $\|\Omega - \psi\| < \delta$ which from Definition 2.3.1 requires constructing the vector of amplitudes for both $|\Omega\rangle$ and $|\psi\rangle$. The problem is then that we cannot construct the state vector $|\Omega\rangle$ since the simulation technique of Chapter 6 only allows us to sample outcomes from the probability distribution of computational basis measurements, and it is unknown if one can compute the state vector from a state’s stabilizer decomposition. At best, by obtaining a large number of

¹Ref. (3) does not consider the introduction of the simulation hyperparameter α to allow the simulator to work in the regime of small number of qubits. This was our original idea.

samples, we are able to estimate the probability distribution associated with each basis state. The solution is to find an equivalent condition in terms of values we are able to compute. We found one using the total variation distance.

7.1.1 Experimental statement of success

Recall from Claim 1 that if $\|\Omega - \psi\| < \delta$ is true, so will $\delta_{td}(P_\Omega, Q_\psi) < \delta$, and viceversa. Then, we can simply verify the latter, which we can do given the tools at our disposal. In the one hand, we can compute the probability distribution Q_ψ exactly from the state vector (simulated using Qiskit) by squaring the probability amplitudes α_i as discussed in Section 2.6. On the other hand, we can estimate P_Ω by collecting a large number of samples from our extended stabilizer simulator. Notice that we cannot compute P_Ω exactly, as this would require collecting an infinite number of samples. In principle, this is then sufficient to validate the simulation algorithm.

7.1.2 Testing our simulator's success

To verify the new criteria, we will use a non-Clifford circuit U that has a small number of qubits and gates such that we can simulate $|\psi\rangle = U|0\rangle$ exactly using Qiskit's state vector simulator and obtain Q_ψ , as well as be able to produce several stabilizer decompositions with our extended stabilizer simulator. For each stabilizer decomposition $|\Omega\rangle$ we will estimate P_Ω and compute the total variation distance between the two distributions. If our simulator was built correctly, we hope to see that the collection of total variation distances (one for each stabilizer decomposition) all have value less than δ .

Remark 9. *Because our goal requires us to estimate empirical discrete distributions in high dimensional spaces a repeated number of times, we require more computational power than that of a laptop computer. For the simulations presented in this chapter we use SURF-sara's cluster computer called "Lisa" that better suits our needs. The specifications of Lisa's hardware can be found in Appendix A. Practical applications of the simulator do not usually require computing the full probability mass function, and so in practice, a cluster computer is not necessary.*

We choose a random non-Clifford circuit of the form given in Example 6.4.1 with 5 qubits, 50 Clifford gates and 5 T gates. The explicit circuit we used for the simulations in this section can be found in Appendix B. For this particular circuit, Example 6.4.1 demands that $\alpha = 23$, and to ensure that the Markov chain converges, we choose 3200 Metropolis steps, which is $100\times$ the maximum number of possible states in the chain's state space. Given that beforehand we do not know anything about the state space of the chain, we have to assume that all 2^n states form part of the chain, and $100\times$ this value seems like a reasonable number of steps for the chain to mix. Fig. 7.1 shows the result of the simulation. Here, we observe that all of the points lie well within the value of δ , confirming that we are indeed sampling measurement outcomes from a state $|\Omega\rangle$ such that $\|\Omega - \psi\| \leq \delta$. Furthermore, we also note that sparsification produces stabilizer decompositions that define ergodic Markov chains. With these simulation parameters, the

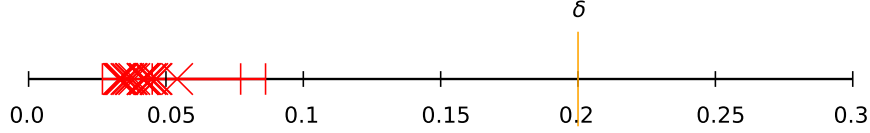


Figure 7.1: Shows the total variation distance between the estimated state and the one computed via the state vector simulator for 20 different stabilizer decompositions. The orange line corresponds to the value of $\delta = 0.2$. The circuit used for the simulation can be found in Appendix B. The simulation parameters were $\alpha = 23$, and 3200 Metropolis steps and samples. For clarity, we only show the error bars associated with the extremal points.

simulation took 50 core hours¹. Given that is computationally expensive to produce a single data point, we used a resampling technique to provide 95% confidence intervals on the total variation distance per data point.

For most practical applications of the simulator, one will not usually estimate the full probability distribution of the state nor use a cluster computer. Therefore, a much more useful statement about the runtime of the algorithm is the time taken to produce a sample using a single core. For the circuit used in Fig. 7.1 this is about 200 seconds in a standard laptop computer.

Even though we have successfully demonstrated that the simulator we have built succeeds with high probability with the chosen hyperparameters, the fact that all stabilizer decompositions represent states that are very close in total variation distance to the true state, ignoring the chosen error-tolerance, suggests that the simulation parameters are larger than necessary. This can be traced back to the simulation choices and worst-case bounds used in Chapter 6. We think that by experimentally testing the simulation hyperparameters that shape the probability distribution the algorithm samples from, we can reduce the amount of resources needed for simulations and improve the simulation runtime, while still producing a state $|\Omega\rangle$ that δ -approximates $|\psi\rangle$.

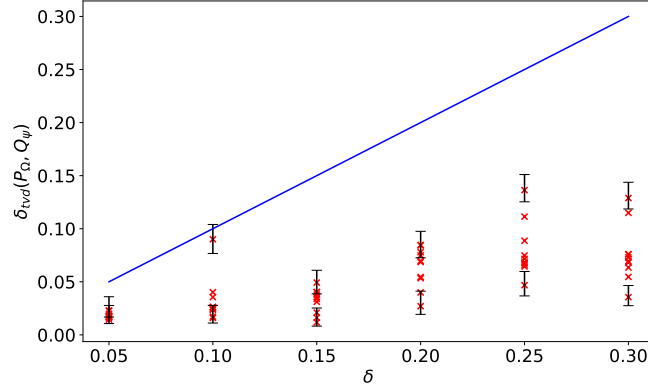
In the algorithm, there are two hyperparameters that influence how close the stabilizer decomposition created compares with the true state. These are the number of states in the stabilizer decomposition that result from randomized sparsification, and the number of Metropolis steps necessary for the chain to converge. Let us discuss the former first.

7.1.3 Decomposition size

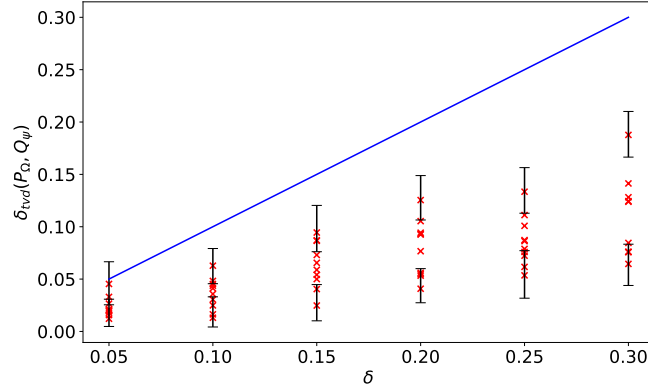
In Section 6.4 and Section 6.5 we discussed that choosing the value of α to be proportional to the stabilizer fidelity, as in Eq. (6.98), and sampling $k \approx \alpha \|c\|_1^2 \delta^{-2}$ circuits, one can produce with 95% probability a state $|\Omega\rangle$ that approximates the true state $|\psi\rangle = U|0\rangle$ for a non-Clifford circuit U . However, in doing so, we considered decomposing individual gates instead of the circuit unitary, approximating the stabilizer fidelity and Hoeffding's worst-case bound, all which culminated in Eq. (6.97). Here, we test how well Eq. (6.97)

¹This means that using only a single core, the simulation would take 50 hours. If m cores are available, then the simulation will take approximately $50/m$ hours. Most laptop computers only have 4 physical cores at their disposal.

does in practice, and provide a practically oriented value of the parameter α in order to save resources and time.



(a)



(b)

Figure 7.2: Plots comparing the total variation distance between the real probability mass function and the estimated one for different stabilizer decompositions of an $-H-T-H-$ with 3 qubits. (a) creates stabilizer decompositions with $\alpha = 2$ and (b) with $\alpha = 1$.

For the test, we use a 3-qubit $-H-T-H-$ circuit, where each layer acts on all qubits. We choose this circuit since it is easy to show that $\mathcal{F}(\psi) \geq 0.3$ ¹, plus we know that the true state is not a stabilizer state and hence we expect the weight to be spread across many states in the stabilizer decomposition. Fig. 7.2 shows plots on the total variation distance for this circuit as a function of δ , where we manually set $\alpha = 2$ and $\alpha = 1$ and again consider 100×2^n Metropolis steps and samples. The data points in the graph correspond to different stabilizer decompositions of the same circuit. Notice that all of the points in both plots lie within the value of δ , however, the data points of Fig. 7.2b lie slightly above those of Fig. 7.2a and the error bars show that there is more variation for stabilizer decom-

¹By simulating the state vector, one can see that the overlap with the $|000\rangle$ stabilizer state is 0.3.

positions that were created with $\alpha = 1$. Fig. 7.2a shows an outlier at the value of $\delta = 0.2$, which we think is due to a non-ergodic Markov chain. With the exception of this outlier, this plot shows that it is probably enough to choose the hyperparameter α to be $\alpha \approx 2$ for any system size or type of circuit and the procedure will still succeed with high probability as long as sufficient number of Metropolis steps are allowed. We will continue to use this value in the rest of the simulations in this thesis.

Before discussing the influence of the number of Metropolis steps, let us revisit Example 6.4.1 with this new value of α .

Example 7.1.1 (Example 6.4.1 revisited.). *Sampling measurement outcomes from a circuit of the form discussed in Example 6.4.1, in practice, only requires constructing stabilizer decompositions of size $k \approx (1.1715^s \times 2)/0.2^2$. In addition, using sparsification will be more advantageous than considering the full decomposition when the number of non-Clifford gates in the circuit s is $s \geq 8$.*

7.1.4 Heuristic Metropolis steps

Since the mixing time of the Markov chain, choosing the number of steps blindly or making an estimate only from the maximum size of the chain's state space, result in computationally expensive simulations. Here, we provide a heuristic model on how to estimate an appropriate number of Metropolis steps for the weak simulation of random non-Clifford circuits. Clearly, sampling from other quantum circuits will require a different amount of steps, and so we suggest that everyone using the software developed in this thesis (or Qiskit's) that they carry out experiments on short-depth circuits to establish the general convergence behavior before simulating their actual circuits.

There are two parameters one must consider in order to provide a good estimate on the number of steps necessary for the chain to thermalize. First, is the scaling with the number of qubits n , and second, the scaling with the number of non-Clifford gates in the circuit s . The dependence on the number of qubits is due to the fact that each qubit added to the system has the potential to double the size of the state space of the chain (some bitstrings x will not meet the requirement that $p(x) > 0$ and will therefore not contribute to the state space), and so the chain will require more Metropolis steps before it reaches its steady-state distribution. The dependence with the number of non-Clifford gates is more subtle, as increasing s also increases k thus adding more bitstrings to the state space. Beforehand, it is difficult to tell which parameter will have the strongest effect, and so we run some simulations using random Clifford circuits to better characterize this dependence.

To study such dependencies we again compare using the total variation distance. The idea is to plot the total variation distance as a function of Metropolis steps for circuits with different number of non-Clifford gates and qubits. We expect that when the chain has converged to its steady-state distribution, the total variation distance between the estimated probability P_Ω and the true one obtained via the state vector simulator Q_ψ must be less than δ . Moreover, by the analysis on the dependence of the size of the decomposition (Fig. 7.2) we know that choosing $\alpha = 2$ should be sufficient to produce good results. We also make sure to collect a large number of samples to ensure that the only free parameters that affect the experiment are s and n .

Fig. 7.3 shows the results of the simulation for some random non-Clifford circuits with different values of s and n . The graph shows two things. First, is that the total variation

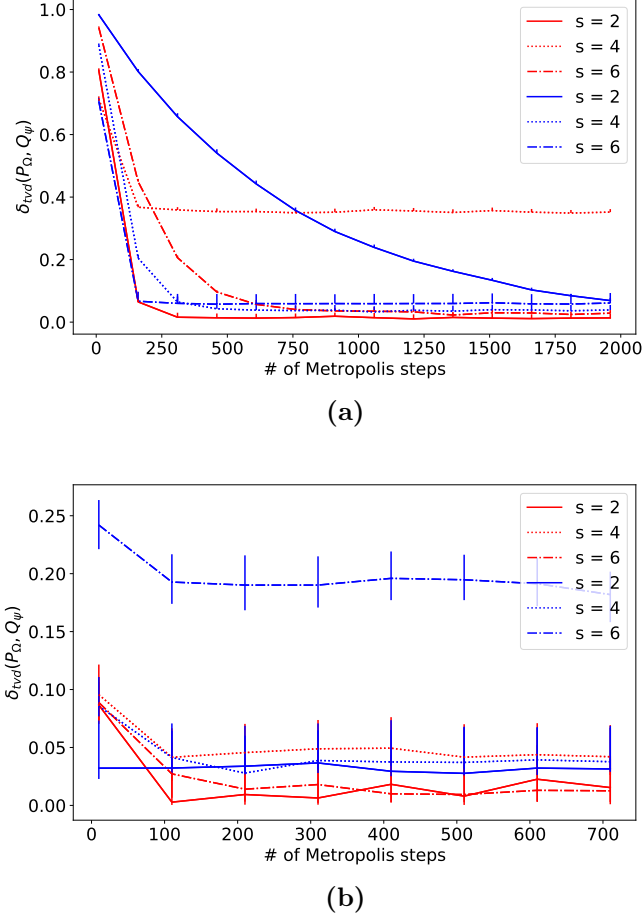


Figure 7.3: Total variation distance as a function of the number of Metropolis steps for $s \in \{2, 4, 6\}$ T gates in the circuit and an error tolerance of $\delta = 0.1$. (a) shows a plot for 10 (red lines) and 13 (blue lines) qubits using 25000 samples and $\alpha = 2$. This plot took approximately 960 core hours to produce. (b) plot for 4 (red lines) and 6 (blue lines) qubits using 5000 samples and $\alpha = 2$.

distance between the estimated state and the actual final state decreases exponentially with increasing number of Metropolis steps, which is in agreement with the theory of stochastic processes. Second, is that in each of the subfigures, there is a line that never converges to the desired probability distribution ($s = 2, n = 15$ in Fig. 7.3a, and $s = 6, n = 6$ in Fig. 7.3b). This could mean that either the algorithm is not producing a large enough stabilizer decomposition that approximates the true state, or the Metropolis algorithm is producing a non-ergodic Markov chain. However, as just shown in Fig. 7.1 and Fig. 7.2 picking $k \approx 2\|c\|_1^2\delta^{-2}$ results in sampling from the correct probability distribution most of the time (1 out of 50 points failed), and so we conclude that this is due to the Metropolis algorithm. Contrary to the statement made in Ref. (3), this shows that even in the regime of large number of qubits and considering $\alpha = 2$, the algorithm can fail occasionally. We therefore warn users of our simulator or Qiskit's of this possibility and urge them to verify

in some way that the probability from which they are sampling is in fact the desired one.

We also note that from these plots it is difficult to characterize how s and n influence the mixing time of the chain since the convergence depends quite heavily on each individual circuit structure. To fully characterize this behavior we would have to average over many circuits. Given that a single curve of Fig. 7.3a takes around 150 core hours to create, computing averages would be computationally prohibited. These plots also verify that the statement that the mixing time of the chain increases polynomially with n .

For our example, however, we can see from Fig. 7.3a that 200 Metropolis steps should suffice for the chain to converge.

7.1.5 Practical validation

Now that we shown that simulating arbitrary non-Clifford circuits with $\alpha = 2$ succeed with high probability, and that 200 Metropolis steps should be enough for the chain to converge for circuits with small number of qubits, we can show how these perform for Example 6.4.1. Simulating the circuit with this new set of parameters results in Fig. 7.4. Here, we see that the the value of the parameters we have chosen still produce states whose total variation distance with ψ are still within the value of δ . Moreover we note that producing this graph only took 0.3 core hours, and collecting a single sample using one core takes 0.4 seconds. This is around $200\times$ faster than the previous simulation! This proves that in practice, one can usually pick smaller values of the hyperparameters and run simulations in a fraction of the time than those suggested by the theory. Obviously, the results are no longer guaranteed by the theory, but if one is careful in characterizing the behavior of α and T for the circuits of interest, one can do much better than to blindly follow the theory and approximations made.

Now let us see how our extended stabilizer simulator compares with the one found in Qiskit.

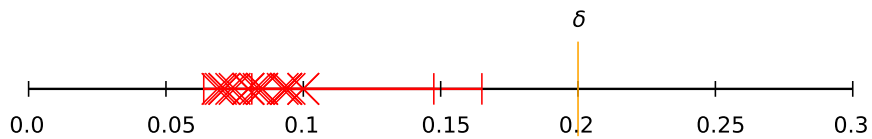


Figure 7.4: Simulation of the same circuit as before. The parameters we considered are 200 Metropolis steps, $\alpha = 2$, and obtained 800 samples.

7.2 Benchmark

In this section we compare and assess the performance of our Python simulator against Qiskit’s version of the extended stabilizer simulator. An important remark is that while Qiskit’s simulator is interfaced and configured in Python, the actual computation is performed in C++. This is relevant for the comparison of simulators since compiled languages,

like C++, are inherently faster than interpreted languages like Python. In addition, Qiskit usually uses a procedure called *transpilation* where before execution of the circuit, it analyzes the circuit and looks for consecutive gates that when multiplied together result in the identity operator. If it finds any such gates, it drops them from the circuit to reduce simulation costs. We manually disable this option as it would give Qiskit’s extended stabilizer simulator an unfair advantage over ours.

Remark 10. *Qiskit offers several ways of using the `extended_stabilizer` backend. We noticed that each method, as well as Qiskit version produces different simulation times. Here, we implemented the backend in the same way as shown in the Jupyter Notebook guide that accompanies this simulator, which can be found on the [IBM’s Quantum Experience website](#). The version of Qiskit used (and its submodules) can be found in Appendix A.*

Finally, for the simulations we will set $\alpha = 1$ since this is the value that Ref. (3) and Qiskit consider at all times. Moreover, we will only use T gates as the only non-Clifford gates in the circuit since Qiskit’s extended stabilizer simulator cannot simulate any other non-Clifford gate.

The results for the time it takes to sample from a non-Clifford circuit using a single CPU core are shown in Fig. 7.5a and Fig. 7.5b. Here, we observe that despite Python’s large overhead, our simulator is able to approach Qiskit’s simulation times to within a factor of two, when we consider collecting more than one sample. Other simulations not shown here, demonstrated that collecting more than 100 samples would not decrease the gap.

Our simulator also presents some additional advantages. For example, our software’s clear design and Python’s user-friendliness facilitate improvements and expansions. Most importantly, our simulator can simulate any non-Clifford gate as long as its decomposition into Clifford is provided, while Qiskit’s implementation can only simulate T gates. Finally, our simulator can evaluate small qubit systems, as well as systems above 63 qubits.¹

Fig. 7.5c shows the case where we allow both simulators to use parallel techniques to improve the runtime of the process. We observe that the difference in computation time is again a factor of two, showing that our parallel implementation is also similar to the one of Qiskit’s.

¹We found that Qiskit’s simulator can only simulate systems of up to 63 qubits. We conjecture that this is for their simulator to maintain good memory management and optimized communication.

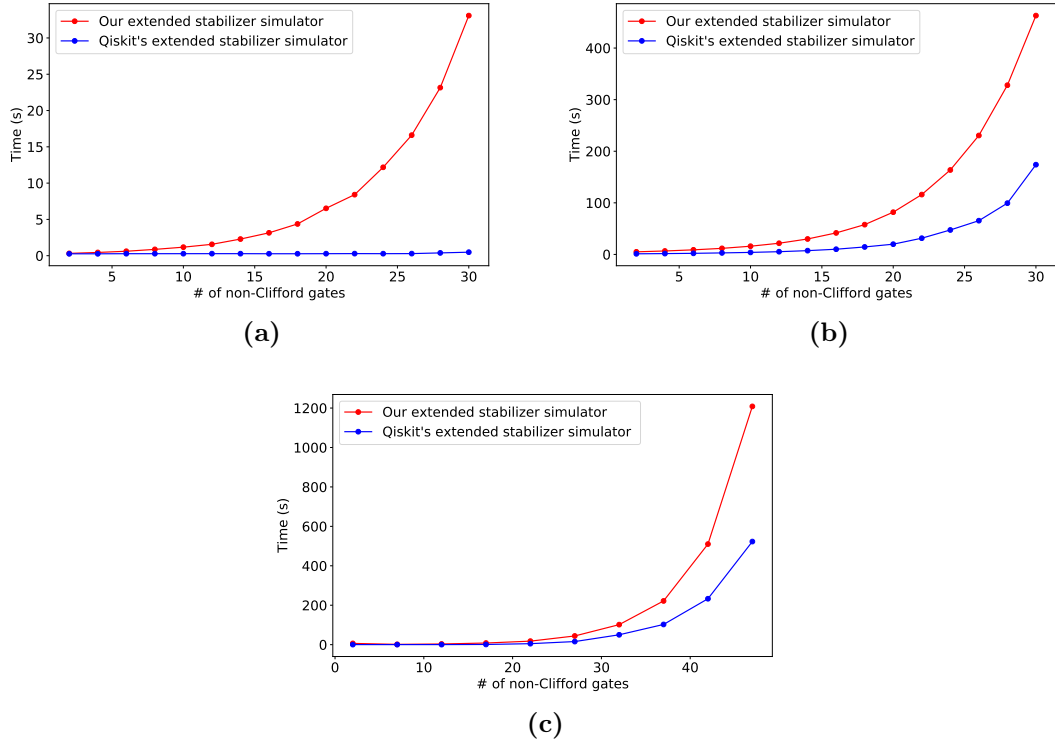


Figure 7.5: Time it takes to produce 1 (a) and 100 (b) samples of a 50-qubit random circuit using a single CPU core. We also chose $\delta = 0.1$, 100 Metropolis steps, and 50 Clifford gates. We averaged the data over 8 different random circuits. For (c) we allowed the simulators to use parallel techniques. Each simulator obtains a total of 1000 samples using 20 cores in Lisa, while keeping a value of $\delta = 0.1$, 100 Metropolis steps, and 50 Clifford gates. We averaged over 2 random circuits.

Conclusions & Further work

8.1 Conclusion

Classical simulators of quantum computers are an essential tool for building and validating new quantum computers. However, the cost of simulation generally increases exponentially with the size of the system. Still, some simulators may have smaller exponents than others and be able to simulate systems of hundreds of qubits or gates, which would have positive practical consequences. Therefore, creating better simulators of quantum computers is necessary for building better quantum computers.

Building upon the two efficient classical simulation algorithms (1, 2) for simulating Clifford circuits, Bravyi, *et al.* (3) recently proposed a unique algorithm capable of weak and strong simulation of arbitrary circuits at a cost of $\mathcal{O}(knT + kn^2)$ where n is the number of qubits in the system, T is the number of Metropolis steps, and k is the exponential variable whose exact value depends on the number of non-Clifford gates in the circuit and the gate type. The paper presents several unknowns such as the necessary amount of Metropolis steps T needed for a simulation, the best way of creating stabilizer decompositions, and it also relies on some worst-case approximations. In this thesis we first sought to review the theory behind these three ground-breaking algorithms, discuss the two efficient classical simulators for Clifford circuits, and construct our own Python implementations of these algorithms. For the extended stabilizer simulation of Bravyi, *et al.*, we sought to improve some of the features mentioned in the weak simulation algorithm, as well as build a fast and versatile Python implementation and use it to experimentally test the impact that these approximations and open question have on simulation runtime.

As mentioned before, the extended stabilizer simulator builds on top of two efficient simulation algorithms. Implementing these in Python helped us understand the practical aspect of the simulation algorithms, as well as define the basic structure in a simple setting that could easily be improved to accommodate the needs of the extended stabilizer simulator. Benchmarks between our implementation of the extended stabilizer simulator and Qiskit's implementation show that, even with Python's large overhead, our simulator's runtime approaches Qiskit's within a factor of two. Furthermore, our simulator presents a number of features which Qiskit lacks. Firstly, our software allows users to compute optimal non-Clifford gate decompositions if the gate they want to simulate acts on one or two qubits. If the gate is larger, we still allow simulation as long as the decomposition

into Clifford gates is provided. In contrast, Qiskit can only simulate *Clifford+T* circuits. Secondly, our simulator can successfully simulate systems above 63 qubits, as well as circuits with small number of qubits (due to our introduction of the hyperparameter α), both which Qiskit is not capable of doing. Finally, our simulator is written in a user-friendly language with a simple structure that facilitates improvements and expansions. In turn, C++ is not as user-friendly, and expanding on Qiskit's software can be difficult given their fragile software's structure for optimization of memory and communication.

On the theory side of the research we directly improved the algorithm described in the paper in two ways. First, is that we introduced a new simulation parameter α that counteracts the case where the stabilizer fidelity can be small. This change implied sampling $k \approx \alpha \|c\|_1^2 \delta^{-2}$ states instead of $k \approx \|c\|_1^2 \delta^{-2}$ states as in the original algorithm, with the advantage that simulations with small number of qubits will now succeed with high probability as well. This is why our simulator can produce reliable results even for small qubit systems. Ref. (3) simply assumes that the simulator will only be used in the regime of a large number of qubits. Our second contribution to the algorithm is that we proposed a method to group the sum-over-Cliffords and sparsification procedures to produce a stabilizer decomposition using only $\mathcal{O}(k \ln^2)$ time and $\mathcal{O}(k)$ memory instead of the regular procedure that takes $\mathcal{O}(m \ln^2)$ time and $\mathcal{O}(m)$ memory and $k < m$. Briefly, this was due to iterating through the circuit and sampling individual Clifford gates from a non-Clifford gate decomposition k times, which at the end of the procedure resulted in k Clifford circuits. One would then evaluate these circuits using the phase-sensitive simulation algorithm which would result in a stabilizer decomposition of size k equivalent to the one obtained by sampling k states from the full stabilizer decomposition of size m .

Furthermore, we observed that the measure of success, as explained in the paper, is not experimentally verifiable, since the output of the weak simulation does not output the probability amplitudes necessary to evaluate if $\|\Omega - \psi\| \leq \delta$ is true. Therefore, we proposed an equivalent condition using the total variation distance that can be verified using the output of the simulation and Qiskit's state vector simulator. Using this validation procedure we verified that our Python simulator worked correctly. Furthermore, by performing simulations with parameters dictated by the theory, we observed that the simulations succeeded but completely disregarded the chosen error tolerance δ . This meant that the parameters were being overestimated which in turn resulted in unnecessarily slow simulations. Therefore, we sought to provide better estimates on simulation hyperparameters α and T in order to improve simulation time.

We showed that setting $\alpha = 2$ is enough to produce simulations with arbitrary circuits that succeed with high probability even in the regime of small number of qubits, showing that the bound on Eq. (6.97) does not depend strongly on the stabilizer fidelity. For the parameter T , we showed that our validation approach was unable to characterize its dependence with the number of qubits and non-Clifford gates in the system, and therefore failed to provide practical estimates of its value that could be used generally. We found that T depends strongly on the structure of the simulated circuit, and we recommend a deeper study of this phenomena in future work. Even though we could not characterize T , our simulations did show that its dependence with the number qubits and non-Clifford gates is indeed polynomial. Furthermore, we also noticed that even in the regime of large number of qubits, the Metropolis algorithm fails to produce ergodic Markov chains with non-negligible probability and we therefore warn users of our simulator and Qiskit's of

this result. Finally, we showed that with better estimates of simulation hyperparameters obtained through experimental methods, one can simulate systems approximately $200\times$ faster than with parameter values obtained from the theory.

8.2 Further work

Fig. 7.2 and Fig. 7.1 show that the techniques, bounds, and assumptions of unknown parameters in the algorithms for the simulation of non-Clifford circuits result in unnecessarily high simulation parameters that slow down simulation times. Moreover, experimental simulations demonstrate that it is possible to obtain correct results, while using smaller values of the simulation parameters. Here, we discuss some possible improvements in the theory that could match experimental observations with theoretic constraints. Finally, we also discuss possible improvements targeted to our extended stabilizer simulator that could result in better simulation times.

8.2.1 General improvements

1. We considered the simplest method of creating stabilizer decompositions. That is, we create stabilizer decompositions by decomposing each of the non-Clifford gates in the circuit in terms of Clifford gates and then sample $k \approx \prod_{p=1}^m \xi(V_p)/\delta^2$ Clifford circuits that will make up the decomposition. However, recall from Eq. (6.7) that $\xi(U) \leq \prod_{p=1}^m \xi(V_p)$, which suggests that one could, in theory, produce smaller decompositions by decomposing the circuit unitary U all at once. As mentioned before, this is incredibly inefficient with brute-force algorithms since these have to consider all of unitaries in the Clifford group. However, we think that there might be algorithms that could compute approximately optimal decompositions in reasonable time.
2. Proving the mixing-time of the Markov chain or characterizing it more accurately. As we showed, the mixing-time of the chain depends strongly on the circuit to be simulated, making it harder to study. Our method of constructing an empirical distribution and considering the total variation distance between this distribution and the true one for increasing number of Metropolis steps proved to be too slow in order to characterize the mixing-time. Maybe one could come up with a better method.
3. Obtain semi-independent samples from the Metropolis algorithm. The Metropolis algorithm, as stated in Ref. (3), consists on collecting independent samples, forcing one to spend computation time waiting for the Markov chain to mix before collecting a single sample, and then restarting the whole process to collect another sample, and so on. One practical improvement could be wait for the chain to converge and then collect a small percentage of the total number of samples required before restarting the process again from scratch. We think it might be interesting to see what the trade-off between the success probability and runtime is. Unfortunately this idea depends strongly on accurately knowing the mixing-time of the chain.

8.2.2 Our simulator

1. Improving the `apply_h` and `pauli_given_basis_sampling` functions. As shown in Fig. 6.2 and Fig. 6.3, the functions `apply_h` and `pauli_given_basis_sampling` are the ones that take the longest in a simulation of a non-Clifford circuit. These are also used quite often during the simulation, so even a small improvement in the execution of the functions could have great impact in reducing the runtime of our simulator.
2. Other improvements to the simulator would be to take advantage of its versatility and implement the most recent extensions to the simulation technique discussed in Refs. (18, 19).

References

- [1] DANIEL GOTTESMAN. **The Heisenberg representation of quantum computers.** *arXiv preprint quant-ph/9807006*, 1998. [ii](#), [4](#), [20](#), [26](#), [27](#), [50](#), [77](#)
- [2] SCOTT AARONSON AND DANIEL GOTTESMAN. **Improved simulation of stabilizer circuits.** *Physical Review A*, **70**(5):052328, 2004. [ii](#), [4](#), [20](#), [33](#), [34](#), [35](#), [39](#), [41](#), [50](#), [77](#)
- [3] SERGEY BRAVYI, DAN BROWNE, PADRAIC CALPIN, EARL CAMPBELL, DAVID GOSSET, AND MARK HOWARD. **Simulation of quantum circuits by low-rank stabilizer decompositions.** *Quantum*, **3**:181, 2019. [ii](#), [20](#), [39](#), [40](#), [55](#), [59](#), [63](#), [67](#), [68](#), [73](#), [75](#), [77](#), [78](#), [79](#)
- [4] RICHARD P FEYNMAN. **Simulating physics with computers.** In *Feynman and computation*, pages 133–153. CRC Press, 2018. [1](#)
- [5] PETER W SHOR. **Algorithms for quantum computation: discrete logarithms and factoring.** In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994. [1](#)
- [6] LOV K GROVER. **A fast quantum mechanical algorithm for database search.** In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996. [1](#)
- [7] ARAM W HARROW, AVINATAN HASSIDIM, AND SETH LLOYD. **Quantum algorithm for linear systems of equations.** *Physical review letters*, **103**(15):150502, 2009. [1](#)
- [8] SETH LLOYD, MASOUD MOHSENI, AND PATRICK REBENTROST. **Quantum principal component analysis.** *Nature Physics*, **10**(9):631–633, 2014. [1](#)
- [9] M NEST. **Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond.** *arXiv preprint arXiv:0811.0898*, 2008. [2](#), [16](#)
- [10] KEVIN M OBENLAND AND ALVIN M DESPAIN. **A parallel quantum computer simulator.** *arXiv preprint quant-ph/9804039*, 1998. [3](#)
- [11] ALWIN ZULEHNER, STEFAN HILLMICH, AND ROBERT WILLE. **How to efficiently handle complex values? Implementing decision diagrams for quantum computing.** In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2019. [3](#)

REFERENCES

- [12] LUKAS BURGHOLZER, HARTWIG BAUER, AND ROBERT WILLE. **Hybrid Schrödinger-Feynman Simulation of Quantum Circuits With Decision Diagrams.** *arXiv preprint arXiv:2105.07045*, 2021. [3](#)
- [13] BARBARA M TERHAL AND DAVID P DiVINCENZO. **Classical simulation of noninteracting-fermion quantum circuits.** *Physical Review A*, **65**(3):032325, 2002. [3](#)
- [14] EMANUEL KNILL, DIETRICH LEIBFRIED, ROLF REICHLE, JOE BRITTON, R BRAD BLAKESTAD, JOHN D JOST, CHRIS LANGER, ROEE OZERI, SIGNE SEIDELIN, AND DAVID J WINELAND. **Randomized benchmarking of quantum gates.** *Physical Review A*, **77**(1):012307, 2008. [3](#)
- [15] DANIEL GOTTESMAN. *Stabilizer codes and quantum error correction.* California Institute of Technology, 1997. [3](#), [20](#)
- [16] PIOTR CZARNIK, ANDREW ARRASMITH, PATRICK J COLES, AND LUKASZ CINCIO. **Error mitigation with Clifford quantum-circuit data.** *arXiv preprint arXiv:2005.10189*, 2020. [3](#)
- [17] CHARLES H BENNETT, DAVID P DiVINCENZO, JOHN A SMOLIN, AND WILLIAM K WOOTTERS. **Mixed-state entanglement and quantum error correction.** *Physical Review A*, **54**(5):3824, 1996. [3](#)
- [18] HAMMAM QASSIM, JOEL J WALLMAN, AND JOSEPH EMERSON. **Clifford recompilation for faster classical simulation of quantum circuits.** *Quantum*, **3**:170, 2019. [3](#), [20](#), [80](#)
- [19] HAKOP PASHAYAN, OLIVER REARDON-SMITH, KAMIL KORZEKWA, AND STEPHEN D BARTLETT. **Fast estimation of outcome probabilities for quantum circuits.** *arXiv preprint arXiv:2101.12223*, 2021. [3](#), [80](#)
- [20] MICHAEL A. NIELSEN AND ISAAC L. CHUANG. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000. [5](#)
- [21] DMITRI MASLOV AND MARTIN ROETTELER. **Shorter stabilizer circuits via Bruhat decomposition and quantum circuit transformations.** *IEEE Transactions on Information Theory*, **64**(7):4729–4738, 2018. [16](#), [43](#)
- [22] HECTOR J GARCIA, IGOR L MARKOV, AND ANDREW W CROSS. **Efficient inner-product algorithm for stabilizer states.** *arXiv preprint arXiv:1210.6646*, 2012. [16](#)
- [23] ANDREW W CROSS, LEV S BISHOP, JOHN A SMOLIN, AND JAY M GAMBETTA. **Open quantum assembly language.** *arXiv preprint arXiv:1707.03429*, 2017. [17](#)
- [24] A ROBERT CALDERBANK AND PETER W SHOR. **Good quantum error-correcting codes exist.** *Physical Review A*, **54**(2):1098, 1996. [20](#)

-
- [25] ANDREW STEANE. **Multiple-particle interference and quantum error correction.** *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, **452**(1954):2551–2577, 1996. [20](#)
 - [26] CHANGCHUN ZHONG, YAT WONG, AND LIANG JIANG. **Entanglement trimming in stabilizer formalism.** *arXiv preprint arXiv:2103.09932*, 2021. [20](#)
 - [27] LI-YI HSU AND CHING-HSU CHEN. **Exploring Bell nonlocality of quantum networks with stabilizing and logical operators.** *Physical Review Research*, **3**(2):023139, 2021. [20](#)
 - [28] SERGEY BRAVYI AND DAVID GOSSET. **Improved classical simulation of quantum circuits dominated by Clifford gates.** *Physical review letters*, **116**(25):250501, 2016. [20](#), [39](#)
 - [29] GUIFRÉ VIDAL. **Efficient classical simulation of slightly entangled quantum computations.** *Physical review letters*, **91**(14):147902, 2003. [25](#)
 - [30] MARK HOWARD, JOEL WALLMAN, VICTOR VEITCH, AND JOSEPH EMERSON. **Contextuality supplies the ‘magic’ for quantum computation.** *Nature*, **510**(7505):351–355, 2014. [25](#)
 - [31] SIMON ANDERS AND HANS J BRIEGEL. **Fast simulation of stabilizer circuits using a graph-state representation.** *Physical Review A*, **73**(2):022334, 2006. [37](#)
 - [32] YAORYUN SHI. **Both Toffoli and controlled-NOT need little help to do universal quantum computation.** *arXiv preprint quant-ph/0205115*, 2002. [39](#)
 - [33] CUPJIN HUANG, MICHAEL NEWMAN, AND MARIO SZEGEDY. **Explicit lower bounds on strong quantum simulation.** *IEEE Transactions on Information Theory*, **66**(9):5585–5600, 2020. [39](#)
 - [34] MARIS OZOLS. **Clifford group.** *Essays at University of Waterloo, Spring*, 2008. [42](#), [66](#)
 - [35] KEVIN P MURPHY. *Machine learning: a probabilistic perspective*. MIT press, 2012. [56](#)
 - [36] WASSILY Hoeffding. **Probability inequalities for sums of bounded random variables.** In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994. [60](#)

Appendix A

Hardware and third-party software

Here, we explain in more detail both the hardware and software components used in this thesis. We also summarize software properties and configurations that were mentioned throughout the text.

Hardware

SURFsara’s cluster computer.

Lisa is a cluster computer from SURF, an association of Dutch educational and research institutions. For this thesis, we were given access to a partition dedicated for quantum computing simulations, which contained a total of 6 nodes, each equipped with a $1.5TB$ of RAM, and a 3.2GHz Intel Xeon Gold 5118 Processor with 20 cores.

Personal computer

For the less demanding simulations of Chapter 4 and Chapter 5, we used a MacBook Pro with a 2.5GHz IntelCore i7 processor, and 16GB of RAM.

Software

Qiskit

Qiskit refers to the quantum software development kit built by IBM. The module is developed for Python, however, most computations (including the extended stabilizer simulator) take place in C++. The version of Qiskit (and its submodules) used in the simulations seen in this thesis are given in Table A.

Module	Version
qiskit	0.26.0
qiskit-terra	0.17.3
qiskit-aer	0.8.2

Table A.1: Qiskit versions used.

Regarding the configuration of the extended stabilizer simulator, we implement it as shown precisely in the Jupyter Notebook set of instructions that accompanies the simulator. This can be found [here](#). Other types of documentation showed various ways of configuring the backend, each which produced different computation times.

Moreover, we also disable the transpilation procedure in the simulation, which is a method to shorten circuits and hence improve simulation times. This procedure is not related to stabilizer simulation in any way and gives Qiskit an unfair advantage over ours, therefore, we turn it off.

JobLib

[Joblib](#) It is a third-party Python module that simplifies parallel computation. The module presents many different methods of performing parallel computations, however, here we simply use it to process the same routines and computations in the available cores of the computer simultaneously. In particular, a single CPU core produces one of the samples.

SnakeViz

[SnakeViz](#) is a third-party Python graphical profiler. We used this profiler to evaluate the simulations of our software, determine which functions were taking the longest and try to improve them.

Appendix B

Validation Circuit

```
1  OPENQASM 2.0;
2  include "qelib1.inc";
3  qreg q[5];
4  h q[2];
5  cz q[0], q[2];
6  cz q[1], q[0];
7  t q[2];
8  s q[1];
9  h q[1];
10 s q[1];
11 cx q[3], q[1];
12 s q[3];
13 s q[3];
14 cz q[0], q[4];
15 cz q[4], q[0];
16 h q[1];
17 t q[2];
18 cx q[3], q[0];
19 s q[3];
20 h q[4];
21 h q[1];
22 t q[4];
23 s q[4];
24 cx q[2], q[1];
25 cz q[0], q[4];
26 cz q[3], q[2];
27 s q[1];
28 h q[0];
29 h q[4];
30 s q[1];
31 cz q[0], q[1];
32 h q[4];
33 s q[1];
34 s q[3];
35 cz q[3], q[0];
36 cz q[1], q[4];
37 cx q[2], q[3];
38 h q[0];
39 s q[4];
40 cx q[1], q[4];
```

```

41     h q[1];
42     cx q[0], q[2];
43     h q[3];
44     cx q[0], q[3];
45     s q[1];
46     t q[1];
47     cx q[3], q[4];
48     s q[3];
49     cz q[4], q[3];
50     t q[1];
51     cx q[2], q[1];
52     cz q[2], q[3];
53     cz q[4], q[1];
54     s q[0];
55     h q[1];
56     h q[1];
57     s q[4];
58     cx q[0], q[3];

```

Listing B.1: Circuit used to validate the extended stabilizer simulator. Contains 50 randomly selected Clifford gates and 5 T gates.