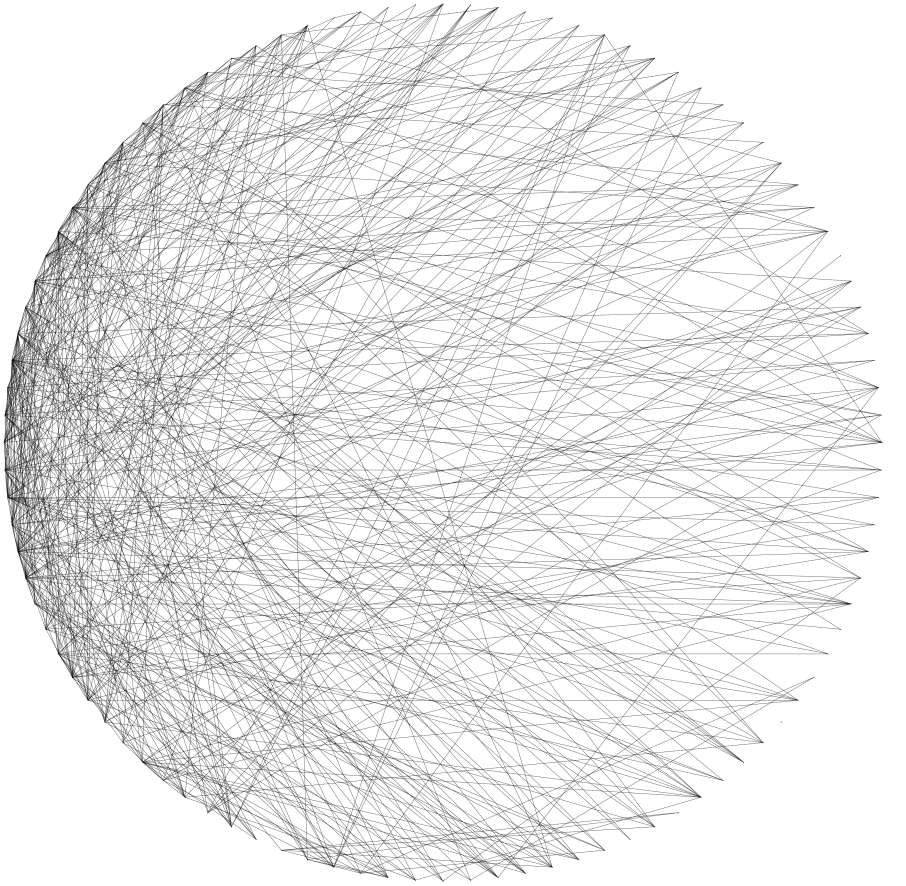
IMPROVEMENTS OF THE CLASSICAL SIMULATION OF QUANTUM CIRCUITS

USING GRAPH STATES WITH LOCAL CLIFFORDS



Matthijs S.C. RIJLAARSDAM

IMPROVEMENTS OF THE CLASSICAL SIMULATION OF QUANTUM CIRCUITS

USING GRAPH STATES WITH LOCAL CLIFFORDS

Master Thesis Computer Science

August 2020

Matthijs S.C. RIJLAARSDAM

Student number	4308417	
Committee members:	Associate Prof. Dr. ir. Z. Al-Ars	TU Delft
	Assistant Prof. Dr. D. Elkouss Coronas	TU Delft, supervisor
	Assistant Prof. Dr. J. Borregaard	TU Delft
Daily supervisor:	PhD-candidate T.J. Coopmans	TU Delft

Keep an open mind. But when you do choose a path, for heaven's sake be aware of what you have done and the choice you have made. Don't try to do both sides.

Richard Hamming

CONTENTS

ACKNOWLEDGEMENTS

This thesis was created in an extraordinary time, during which it was perhaps more important than ever to receive help from others yet it was more difficult for them to give it. I'd like to thank the people who have supported my throughout this journey, despite the circumstances, and without whom this work would have never possible.

First, I would like to thank the people from QINC, for their help and warm welcome. I've felt a part of the group from day one. I'd especially like to thank my fellow 'roommates' Kenneth, Sebastian and Kaushik for their help and advice, and introducing me to very exotic music tastes. I'd also like to thank professors Borregaard and Al-Ars for taking the time to be a part of my committee.

My supervisor professor David Elkouss has been instrumental in guiding the direction of my thesis, prioritizing results and managing my own expectations. I really appreciate his clarity and honesty, and have learned a lot from our bi-weekly feedback sessions.

Perhaps most importantly, I'd like to thank my daily supervisor, PhD candidate Tim Coopmans. Tim has gone far above and beyond what anyone can expect from a daily supervisor. During our hours long whiteboard (and later Zoom) sessions, he has both given me the space to explore new ideas and tempered unrealistic ambitions at the same time. I thoroughly enjoyed working together, and really appreciate the amount of time he has put into seeing me successfully finish my master thesis.

Last but not least, I'd like to thank my wonderful friends and family for supporting me and providing (at times much needed) distraction.

Matthijs Rijlaarsdam
Rotterdam, August 2020

1

INTRODUCTION

Feynman first suggested [1] that a quantum computer (a computer based on the principles of quantum mechanics) might be more powerful than classical computers. Intuitively, one can understand this idea by the fact that the size of the set of numbers necessary to represent a quantum state grows exponentially with its size: there are too many numbers to store and update classically. This suggests that quantum computers can solve problems classical computers cannot. For instance, Shor's algorithm [12] to factor prime numbers or Grover's algorithm [5] for database searching provide a speedup over the best known classical algorithms for their particular problems. Because quantum computers can solve problems that are probably intractable for classical computers, it is widely believed that universal quantum computers cannot be efficiently simulated on classical computers.[47]

Despite this, the simulation of quantum computers with classical computers is of interest for the development of quantum computers. It allows for the study of algorithms, computing architectures before the advent of functional quantum computers[51]. For example, researchers can explore an algorithm's scaling and its robustness against noise[31], or its behaviour on specific hardware (see e.g. [46], [28], [35]). One can evaluate the performance of quantum internet protocols without having access to a functional quantum internet [43]. Classical simulation also enables researchers to verify results obtained by quantum computers [48]. Moreover, it gives an indication of what the computational relation is between classical and quantum computation[24]; if a certain quantum circuit can be efficiently simulated, it cannot provide a "quantum speedup".

An example of a class of quantum circuits that *can* be classically simulated is stabilizer circuits. These circuits, consisting of Clifford gates and Pauli measurements, can be simulated for thousands of qubits using the Gottesman-Knill theorem [47] [9] [17]. Despite not being universal for quantum computation, stabilizer circuits contain several interesting and distinctly 'quantum' behaviors. For instance, they can generate high degrees of entanglement [22].

Additionally, stabilizer states have interesting use cases. Quantum network protocols can for example typically be simulated using stabilizer states (e.g. [40] [52]). The verifi-

1

cation of quantum computation can make use of the stabilizer formalism [48] [37]. Stabilizer states are used to research the effect of noise; they contain common noise models (the depolarizing channel and the dephasing channel). They can efficiently simulate realistic quantum noise, such as decoherence, by approximating it using a method called quantum twirling [27]. Another important use case is quantum error correction, a necessary feature of fault-tolerant quantum computation, which is generally based on stabilizer circuits [8][10].

Moreover, stabilizer circuits can be upgraded to universal quantum computation by adding any non-Clifford gate to the available gate set[20]. This result allows for the simulation of general quantum circuits containing few non-Clifford gates on classical computers using stabilizer states [34] [47]. Most modern quantum simulation platforms (e.g., [60]) include stabilizer-based simulators.

One can use different formalisms to simulate stabilizer states. One commonly used formalism is the tableau formalism, introduced by Aaronson and Gottesman [17]. Later work has extended this formalism by defining an algorithm for calculating the fidelity [30] and extending it to general quantum circuits containing few non-Clifford gates [34][47]. Another formalism is the graph state with local Cliffords ('GSLC'), introduced by Anders and Briegel [21]. This formalism is faster than tableaus in several cases, but no fidelity algorithm or general quantum simulation extension has previously been defined.

With this thesis project, we improve the classical simulation of quantum computers using stabilizers in the GSLC formalism. We do this in two ways: we present new algorithms that speed up their simulation and extend their applicability by defining new operations and subroutines for existing general circuit simulation using GSLC. To be precise: we present multiple new algorithms that speed up the simulation of the CZ gates, the most computationally expensive quantum operation in GSLC formalism. We define two new operations on GSLC that are useful when simulating stabilizer circuits: calculating fidelity (a measure of 'closeness' between two quantum states), and tracing out qubits (throwing away the information about the state contained in these qubits) from a GSLC. Finally, we present a new GSLC-based subroutine for a state of the art general quantum circuit simulation algorithm by Bravyi et al.[47] that allows for the usage of the faster CZ algorithms. We show that the GSLC formalism can give a speedup in practical simulation tasks by evaluating the complexity of simulating an algorithm with possible applications on near-term quantum hardware: the quantum approximate optimization algorithm.

The chapters of this thesis are structured around the following subjects:

- A definition of a 'canonical form' tableau that directly corresponds to a GSLC, and conversion algorithms between the two. GSLC written as matrices were used previously to prove properties of the GSLC algorithm[19]. We expand on this idea by defining a 'canonical form' of tableaus that directly corresponds to GSLC and conversion algorithms between GSLC and canonical form tableaus.

- A new algorithm to simulate CZ gates in GSLC formalism, based on our 'canonical form'. This algorithm is faster for single CZ gates. Additionally, it improves the run time of sequences of CZ gates. We empirically validate these claims on randomly generated states. We show that while the cost of simulating a CZ gate with

the original GSLC algorithm increases per CZ applied, it does not with our novel algorithm.

- A scheduling algorithm to schedule CZ gates in a CZ sequence, that further improves simulation run time for certain sequences of CZ. We empirically validate our performance claims on sequences of CZ.
- Definitions of two new algorithms for operations that are useful when simulating quantum states and were previously defined on tableaus. First, a definition of tracing out and representing the resulting part of the state (called 'a reduced density matrix') in GSLC formalism. Second, an algorithm to calculate fidelity that uses the CZ sequence algorithm.
- A new subroutine for a recent state-of-the-art general quantum circuit simulation algorithm by Bravyi et al.[47]. The subroutine is an extension of the GSLC formalism to make it phase-sensitive. Depending on the circuit simulated, our algorithm can provide a significant speedup compared to the original subroutine. For instance, simulating a Hadamard quantum gate in the original algorithm takes time order $O(n^2)$, where n is the number of qubits, whereas simulating this gate using our subroutine takes $\Theta(1)$. For a typical use case $n \approx 50$. Moreover, our GSLC-based subroutine allows for the usage of our faster CZ and CZ scheduling algorithms. To show the practical use of our results, we show that the set of circuits that can be simulated faster using the GSLC-based subroutine contains interesting use cases. In order to demonstrate this, we evaluate the complexity of simulating the quantum approximate optimization algorithm[32]. This use case is also used by Bravyi et al. to verify their original algorithm's performance[47].

2

PRELIMINARIES

2.1. QUANTUM STATES & GATES

This section gives an introduction to the basic building blocks of quantum information and operations on them: qubits and quantum gates. Its aim is to provide sufficient information to allow a non-expert reader with some technical background be able to understand this thesis. If the reader requires more information, we recommend [15].

Before delving deeper into qubits and quantum gates, we give some definitions and notes on notation.

If z is a complex number, we can write $z = a + bi$ where a and b are real numbers and i is the complex unit, z^* gives its complex conjugate:

$$z^* = (a + bi)^* = (a - bi). \tag{2.1}$$

For a matrix A, $A^\dagger$ is its adjoint matrix which can be obtained by transposing A and subsequently taking the complex conjugate of all entries.

Definition 2.1.1. An $k \times k$ matrix A is **Hermitian** if it satisfies $A = A^\dagger$.

Definition 2.1.2. A $k \times k$ complex matrix A is **unitary** if its conjugate transpose is equal to its inverse, i.e. $A^\dagger A = AA^\dagger = I$.

We will refer to the set of unitaries acting on n qubits (or equivalently of size $2^n \times 2^n$) as $U(2^n)$.

In this thesis, we will often use Dirac notation to describe complex vectors: we write a standing vector as $|\cdot\rangle$ (a "ket") and its adjoint as $\langle\cdot|$ (a "bra"). The inner product between two vectors $|\phi\rangle = \binom{\alpha_0}{\alpha_1}$ and $|\psi\rangle = \binom{\beta_0}{\beta_1}$ can then be written as: $\langle\phi|\psi\rangle = (\alpha_0^* \alpha_1^*)\binom{\beta_0}{\beta_1}$.

Definition 2.1.3. A pure state of a **qubit** can be described by a complex vector $|v\rangle \in \mathbb{C}^2$. In other words, for any pure state $|\phi\rangle$:

$$|\phi\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \alpha_0 |0\rangle + \alpha_1 |1\rangle, \tag{2.2}$$

where $\alpha_0, \alpha_1 \in \mathbb{C}, |\alpha_0|^2 + |\alpha_1|^2 = 1$ and we define $|0\rangle$ and $|1\rangle$ as the computational basis states $\binom{1}{0}$ and $\binom{0}{1}$ respectively.

Given the unit length constraint, we can also parametrize a qubit using two angles $\theta = [0, \pi)$ and $\phi = [0, 2\pi)$: $|\psi\rangle = \binom{cos(\theta/2)}{e^{i\phi} sin(\theta/2)}$. Graphically, we may then represent a qubit as a vector on a unit 2-sphere, where the antipodal points correspond to orthogonal state vectors.

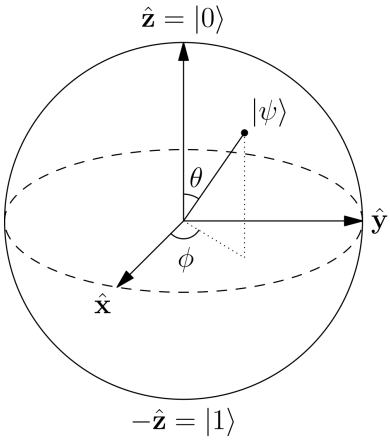


Figure 2.1: Graphical description of a qubit $|\psi\rangle = \binom{cos(\theta/2)}{e^{i\phi} sin(\theta/2)}$ on a Bloch sphere. The angle between the z-axis and the state vector is equal to θ, and the angle between the x-axis and the state vector is equal to ϕ. Here, $\hat{\boldsymbol{x}} = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle, -\hat{\boldsymbol{x}} = |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle), \hat{\boldsymbol{y}} = |i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$ and $-\hat{\boldsymbol{y}} = |-i\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$. As reproduced from [54].

To represent states of multiple qubits, we need an operation called the tensor product:

Definition 2.1.4. The **tensor product** of two matrices A (of dimension $m \times n$) and B (of any size) is equal to:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ \vdots & & & \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix} \tag{2.3}$$

Throughout this thesis, we will omit the tensor symbol and write the tensor product of two vectors as a single ket for readability, e.g.: $|0\rangle \otimes |1\rangle = |01\rangle$.

A state of two qubits we can describe as:

$$\alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle = \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{bmatrix}, \tag{2.4}$$

where $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$. In general, any pure n-qubit state can be described as the sum over 2^n basis states:

Definition 2.1.5. A **pure n-qubit quantum state** can be described as a normalized complex vector $\mathbb{C}^{2^n}$. In other words, for any n-qubit state $|\phi\rangle$:

$$|\phi\rangle = \sum_{k=0}^{2^n} \alpha_k |k\rangle, \tag{2.5}$$

where $\alpha_k \in \mathbb{C}$ and $\sum_{k=0}^{2^n} |\alpha_k|^2 = 1$.

A quantum state may be multiplied with a (complex) phase with length 1. These can be parametrized as $e^{i\alpha}$, where $\alpha \in [0, 2\pi)$. We can express a phase as a unitary in $U(1)$. Multiplying a quantum state with a phase yields the exact same physical state, i.e. $|\phi\rangle = e^{i\alpha}|\phi\rangle$. In other words, a global phase factor does not have any physical meaning.

If we want to know how 'close' two quantum states are to each other, we can use a measure called fidelity:

Definition 2.1.6. The **fidelity** between pure states $|\phi\rangle$ and $|\psi\rangle$ is defined as:

$$|\langle\phi|\psi\rangle|^2 \tag{2.6}$$

Some quantum states we can write as a single tensor product, e.g. $|01\rangle$. However, there also exist quantum states we cannot write as a single tensor product. As an example, take the Bell state (or EPR pair) $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. A quantum system might also be a probabilistic mixture of pure states. These we can represent conveniently using the density operator:

Definition 2.1.7. The **density matrix** for a quantum system with state space $\mathbb{C}^d$ that is in one of a number of states $|\psi_i\rangle$ with respective probability p_i is defined as:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|. \tag{2.7}$$

ρ is a $d \times d$ matrix with complex entries that satisfies:

1. $\rho \geq 0$;
2. $\mathrm{Tr}(\rho) = 1$.

If the rank of $\rho = 1$, then ρ is a pure state, otherwise it is mixed.

Here, $\mathrm{Tr}(\rho)$ stands for the trace of ρ, which we define as follows:

Definition 2.1.8. The **trace** of a matrix $M \in \mathscr{L}(\mathbb{C}^d, \mathbb{C}^d)$ is defined as

$$Tr(\rho) = \sum_i \langle i|\rho|i\rangle, \tag{2.8}$$

where $\{|i\rangle\}$ is any orthonormal basis of d-dimensional complex vector space $\mathbb{C}^d$.

A reduced density matrix is the result of an operation on the density matrix of a larger state called the 'partial trace'. Intuitively, this operation extracts the information contained in a smaller part of a state from the whole state.

Definition 2.1.9. The partial trace of a density matrix. Let ρ_{AB} be a density matrix over registers A and B, and ρ_A be the density matrix that describes the state of register A. Let $\{|k\rangle\}_{k\in K}$ be a basis for B. Then the partial trace of ρ_{AB} over B is defined as:

$$Tr_B(\rho_{AB}) = \sum_{k\in K} \langle k|_B \rho_{AB} |k\rangle_B \tag{2.9}$$

$$= \rho_A \tag{2.10}$$

If a classical observer wants to gain some information about a quantum state, she can measure it. Measurement is irreversible; it projects the state to a vector (dependent on in which direction the state is measured).

Definition 2.1.10. Quantum measurements are described by a collection of measurement operators $\{M_m\}$. These operators satisfy the *completeness equation*:

$$\sum_m M_m^\dagger M_m = I. \tag{2.11}$$

The probability $p(m)$ of measurement outcome m when measuring state $|\phi\rangle$ is:

$$p(m) = \langle\phi| M_m^\dagger M_m |\phi\rangle. \tag{2.12}$$

These probabilities sum to one:

$$\sum_m p(m) = \sum_m \langle\phi| M_m^\dagger M_m |\phi\rangle = \langle\phi| I |\phi\rangle = 1. \tag{2.13}$$

The state of the system after measurement is:

$$\frac{M_m |\phi\rangle}{\sqrt{\langle\phi| M_m^\dagger M_m |\phi\rangle}}. \tag{2.14}$$

As an example, we measure qubit $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ in the z-basis (with measurement operators $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$). The probability of measuring 0 is then:

$$p(0) = \frac{1}{2}(\langle 0| + \langle 1|)|0\rangle\langle 0|0\rangle\langle 0|(|0\rangle + |1\rangle) = \frac{1}{2}. \tag{2.15}$$

Conversely, $p(1) = 0.5$. The post measurement state after measuring 0 is:

$$\frac{|0\rangle\langle 0| \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)}{\sqrt{\frac{1}{2}}} = |0\rangle \tag{2.16}$$

Having introduced qubits, let us now introduce some of the quantum gates used in this thesis.

Definition 2.1.11. Pauli matrices. The Pauli matrices are defined as:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.17}$$

The matrices are both unitary and Hermitian.

From these matrices, we can generate a group:

Definition 2.1.12. Pauli group. The Pauli group $\mathscr{P}_1$ is defined as:

$$\mathscr{P}_1 = \langle X, Y, Z \rangle = \{\pm 1, \pm i\} \cdot \{I, X, Y, Z\} \tag{2.18}$$

Here, $I = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ is the identity operator. For n-qubits, the Pauli group $\mathscr{P}_n$ is defined as:

$$\mathscr{P}_n = \{P_1 \otimes P_2 \otimes \cdots \otimes P_n | P_i \in \mathscr{P}_1\} \tag{2.19}$$

Let $\mathscr{P}_n^* = \mathscr{P}_n \setminus \{I^{\otimes n}\}/U(1)$, i.e. the group of Pauli strings with phase +1. A set of operators of interest to us is called the Clifford group. The operators of dimension n in the set of Cliffords $\mathscr{C}_n$ map the Pauli strings in $\pm\mathscr{P}_n^*$ to Pauli strings in $\pm\mathscr{P}_n^*$.

Definition 2.1.13. The **Clifford group** $\mathscr{C}_n$ on n qubits is defined as: [26][25]

$$\mathscr{C}_n = \{V \in U(2^n) | P \in \pm\mathscr{P}_n^* \Rightarrow V^\dagger P V \in \pm\mathscr{P}_n^*\}/U(1). \tag{2.20}$$

For single qubits, the Cliffords can be seen as rotations of the Bloch sphere that permute the direction of the axes. For the x axis, there are six possibilities: $\pm x, \pm y$ and $\pm z$. After choosing one of these directions, we can still pick 4 possible directions for the z axis, because we cannot point the z axis in the same direction as the x axis. Hence, $|\mathscr{C}_1| = 6 \cdot 4 = 24$[25].

The Clifford group is generated by a set of 3 Clifford gates, that we will use throughout this thesis:

Definition 2.1.14. $\mathscr{C}_n = \langle H, S, CZ \rangle$, where

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

Throughout this thesis, we will use subscripts to indicate on what qubit a gate is applied, e.g. $CZ_{a,b}$ means a CZ gate between qubits a and b. Here, when a gate has a control and a target qubit, the first position will denote the control and the second the target. For instance: $CNOT_{a,b} = H_b CZ_{a,b} H_b$ is a control-X gate with target b. Note that the CZ gate is symmetric, that is: $CZ_{a,b} = CZ_{b,a}$.

2.2. STABILIZER STATES

Circuits containing only Clifford gates (so called "stabilizer circuits") acting on the all zero state $|0\rangle^{\otimes n}$ generate stabilizer states. These states have a number of interesting properties that make them useful for classically simulating quantum circuits. In this section, we we give an introduction to these stabilizer states, and how to simulate them.

For single qubits, the H and S gate can only change $|0\rangle^{\otimes n}$ to any of the orthogonal basis states (ignoring phases), as is shown in the following diagram: [58]

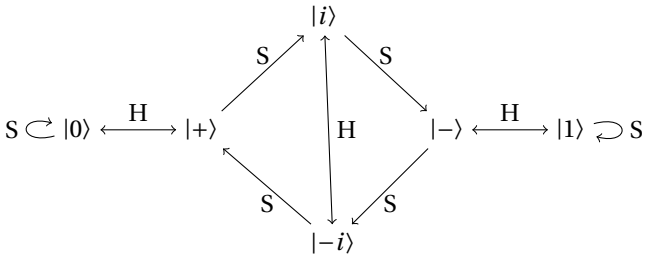


Figure 2.2: Graph depicting the working of single qubit Cliffords on a one qubit state.

These states are the single-qubit stabilizer states. With more qubits, we can reach product states of these single-qubit stabilizer states, as well as entangled states such as a Bell pair. These states share a number of interesting properties: they are always equal to a superposition of a power of 2 bit strings, and a measurement in the z-direction produces either deterministically 0 or 1, or non-deterministically 0 or 1 with equal probability.

For each of these single qubit stabilizer states, there is a Pauli that does not change the state, e.g. $X|+\rangle = |+\rangle$. This Pauli 'stabilizes' the state. These stabilizers also exist for multi-qubit stabilizer states.

Definition 2.2.1. A unitary U **stabilizes** a pure state $|\phi\rangle$ if $U|\phi\rangle = |\phi\rangle$. Equivalently, $|\psi\rangle$ is an eigenstate of U with eigenvalue +1.

If we have a pure n-qubit stabilizer state $|\phi\rangle$, we can make a sub-group $\mathcal{S}_\phi$ of the group of n-qubit Pauli operators in $\mathcal{P}_n$ that stabilize $|\phi\rangle$[36]. The Pauli operators in this group commute: for two stabilizers S_i, S_j, $[S_i, S_j] = S_iS_j - S_jS_i = 0$. This group does not include $-I$, as this stabilizes nothing.

As an example, we give the stabilizer group $\mathcal{S}_{\Psi^-}$ of the 2-qubit Bell state $|\Psi^-\rangle$:

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle), \tag{2.21}$$

$$\mathcal{S}_{\Psi^-} = \{II, -XX, -ZZ, YY\}. \tag{2.22}$$

Note that a group is closed under multiplication: if we have two unitaries U and V that stabilize a state $|\phi\rangle$, then their product also stabilizes $|\phi\rangle$: $UV|\phi\rangle = U|\phi\rangle = |\phi\rangle$. We can use this fact to efficiently represent $\mathcal{S}_\phi$ by its generators.

Definition 2.2.2. The **stabilizer generators** $\mathcal{S}_\phi^g$ of generate the stabilizer group $\mathcal{S}_\phi$:

$$\mathcal{S}_\phi = \langle \mathcal{S}_\phi^g \rangle. \tag{2.23}$$

In other words, every stabilizer in $\mathcal{S}_\phi$ can be expressed as the product of stabilizer(s) in $\mathcal{S}_\phi^g$. For a n-qubit stabilizer group, the number of stabilizer generators needed to represent the group is at most n[36]. Note that $\mathcal{S}_\phi^g$ is not unique.

To clarify, let us express the stabilizer group $\mathcal{S}_{\Psi^-}$ of the 2-qubit Bell state $|\Psi^-\rangle$ of our previous example by two different generator sets:

$$\mathcal{S}_{\Psi^-} = \langle\{-XX, -ZZ\}\rangle \tag{2.24}$$
$$= \langle\{-XX, YY\}\rangle \tag{2.25}$$

$|\phi\rangle$ is unique for the group $\mathcal{S}_\phi$: $\mathcal{S}_\phi$ only stabilizes the state $|\phi\rangle$. Thus, $|\phi\rangle$ is called the stabilizer state of $\mathcal{S}_\phi$. In fact, it is enough that $|\phi\rangle$ is stabilized by the generators of $\mathcal{S}_\phi$:

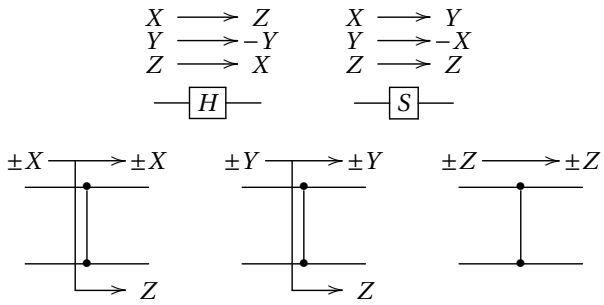
Definition 2.2.3. The **stabilizer state of stabilizer group** $\mathcal{S}_\phi$ is defined as the state $|\phi\rangle$ such that:

$$\forall S_i \in \mathcal{S}_\phi^g, S_i |\phi\rangle = |\phi\rangle. \tag{2.26}$$

This allows us to efficiently represent stabilizer states: instead of storing the 2^n size state vector of a stabilizer state, we can store n generators of length n that uniquely define that state. This takes $2n+1$ bit per generator to store: 2 bits for each of the Paulis, plus 1 for the sign of the generator. Because we have n generators, we can thus store stabilizer states using $O(n^2)$ bits. [58].

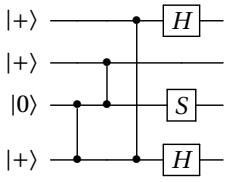
GRAPHICALLY DETERMINING STABILIZER GENERATORS

The commutation relations between the Pauli operators and Clifford operations allow us to determine the stabilizer generators of a stabilizer circuit's output state graphically [36] by 'pushing through' the stabilizers of the initial state. This is useful for understanding and working with stabilizer circuits (and their simulation algorithms) and is therefore repeated here. For instance, because $ZH - HX = 0 \rightarrow ZH = HX$, a qubit that is stabilized by a Z is stabilized by X after applying a Hadamard. Graphically, for the Clifford generating set {H, S, and CZ}, we have the following relations:

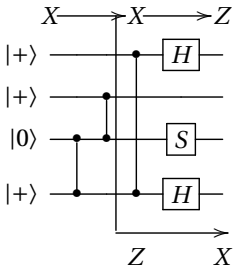


Using these relations, we can determine the stabilizer generators of a circuit graphically.

As an example, we push trough the stabilizer of the first qubit of the following circuit:



, which gives:



resulting in the generator $ZIIX$. If we continue for the other three qubits, we obtain $IXZI, IIZI$ and $XIZX$. With this set of four generators, we can uniquely represent the output state of the circuit.

2.3. SIMULATION OF STABILIZER STATES

Not only can stabilizer states be represented in an efficient way, they can also be simulated efficiently. In this section, we explain more about why this is the case, and give two commonly used stabilizer formalism: stabilizer tableau and graph state with local Cliffords. We finish this chapter by explaining in what applications GSLC are faster than Tableaus.

Stabilizer circuits can be simulated efficiently on a classical computer. To be precise:

Theorem 2.3.1 (Gottesman-Knill theorem)**.** *[9] Any quantum computer performing only:*

1. *Clifford group gates;*
2. *Measurements of Pauli group operators;*
3. *Clifford group operations conditioned on classical bits, which may be the results of earlier measurements;*

can be perfectly simulated in polynomial time on a probabilistic classical computer.

Stabilizer circuits are not universal for quantum computing. For instance, one can not obtain the state $\frac{1}{\sqrt{2}}(|0\rangle + (1+i)|1\rangle)$. One would need to add an extra non-Clifford gate to the Clifford gates to obtain an universal gate set. In fact, stabilizer circuits simulation is complete for the complexity class $\oplus L$, which means they can be simulated using only CNOT gates. Therefore, they are probably not even universal for classical computation [17]. However, they are enough to generate interesting quantum circuits, such as the GHZ experiment[3] and quantum teleportation [4]. Stabilizer circuits are also commonly used in quantum error correction schemes, such as: [29][7]. They can generate entanglement, which indicates that while entanglement is necessary to disallow efficient classical simulation [16] [23], it is not sufficient[24].

So how can we efficiently simulate stabilizer states? In the following subsections, we will introduce two simulation formalisms used in this thesis: stabilizer tableaus and graph states with local Cliffords.

2.3.1. STABILIZER TABLEAUS

The first formalism we introduce is the tableau formalism. We will first introduce a concept called a *check matrix*, before elaborating on the tableau algorithm itself.

As stated in section 2.2, we can represent stabilizer states using $2(n+1)$ bits. We do this by binary representation of the generators of a state in a *check matrix*[15]. Here we use the following encoding:

$$I = (0,0), X = (1,0), Y = (1,1), Z = (0,1). \tag{2.27}$$

For a n-qubit state, we store the n generators in a matrix of $n \times 2n$ bits, that looks as follows:

$$\left(\begin{array}{c|c} x_{11}\cdots x_{1n} & z_{11}\cdots z_{1n} \\ \vdots & \vdots \\ x_{n1}\cdots x_{nn} & z_{n1}\cdots z_{nn} \end{array}\right) \tag{2.28}$$

For a generator $G_i = \pm P_1 \cdots P_n$, bits (x_{ij}, z_{ij}) encode the j-th Pauli P_j of that generator. We store the $\pm$ phase of the generators in a vector of length n, with one bit for each generator. As an example, let us express the stabilizer generators $\{-XX, -ZZ\}$ of the 2-qubit Bell state $|\Psi^-\rangle$ as a check matrix and phase vector:

$$\left(\begin{array}{cc|cc} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right) \& \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{2.29}$$

This binary representation has a number of useful properties:

- It is an isomorphism (modulo the phase). This means that the product of two generators is equal to the binary addition of their corresponding rows (except for the phase). For example: $ZX = iY \Rightarrow (0,1) + (1,0) = (1,1)$. To account for the phase we need to update the phase vector separately, but this can be done efficiently.
- Because the stabilizer group is closed under multiplication, we can binary add two rows in the check matrix to obtain a new valid generator. This changes the generator set, but still represents the same state.
- We can swap rows, as this merely corresponds to a reordering of generators.

These properties allow us to transform the check matrix of a state to a different check matrix that represents the same state, for instance by using Gaussian elimination.

Gottesman and Knill showed that using check matrices, stabilizer states can be simulated efficiently[9]. This is done by rules that update the check matrix and phase vector after a Clifford gate in $O(n)$, and using Gaussian elimination (which takes $O(n^3)$ in practice) for measurements. Aaronson and Gottesman later improved on this by introducing stabilizer tableaus[17]. Tableaus extend check matrices by introducing an additional n rows (thus doubling the required amount of bits for a state) containing the 'destabilizers' of a state, which together with the stabilizer generators generate the entire Pauli group and are used to do measurements without Gaussian elimination in $O(n^2)$.

Throughout this thesis, stabilizer tableaus will be shown not in binary form, but as the generators they represent. We also do not depict the destabilizer rows of the tableau. This is solely for readability: e.g. when analyzing the complexity of stabilizer simulation using tableau formalism, we do also consider the cost of updating of the destablizer rows. As an example, the stabilizer tableau (with the phase vector appended to the check matrix) of the 2-qubit Bell $|\Psi^-\rangle$ state we represent as:

$$\left(\begin{array}{cc|cc|c} 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{array}\right) = \begin{array}{c} -XX \\ -ZZ \end{array} \tag{2.30}$$

To clarify the working of these update rules (which can be understood using the graphical technique described in 2.2), we will now give an example of simulating Clifford gates on a tableau. For more detail on these updates and the measurement routine, we refer to [17]. We start with the state $|0000\rangle$. Its tableau looks as follows:

$$\begin{array}{c} ZIII \\ IZII \\ IIZI \\ IIIZ \end{array} \tag{2.31}$$

When we apply a Hadamard on the second qubit, the tableau is update to the following:

$$\begin{array}{c} ZIII \\ IXII \\ IIZI \\ IIIZ \end{array} \tag{2.32}$$

Applying a S gate to the second and fourth qubit changes the tableau into:

$$\begin{array}{c} ZIII \\ IYII \\ IIZI \\ IIIZ \end{array} \tag{2.33}$$

If we finally apply a CZ gate between the second and third qubit, we get:

$$\begin{array}{c} ZIII \\ IYZI \\ IIZI \\ IIIZ \end{array} \tag{2.34}$$

The stabilizer tableau update rules can be understood by the commutation relations of Cliffords and the graphical method of obtaining circuit generators described in section 2.2. The k-th row of the tableau corresponds to the stabilizer generator one gets from 'pushing through' the stabilizer of the k-th qubit. When we apply a gate on a qubit, the commutation relations give us the update rule for all Paulis in the column corresponding to that qubit. For instance, after we apply a Hadamard on the second qubit, the commutation relation $ZH = HX$ means that all Z Paulis in the second column are changed to X.

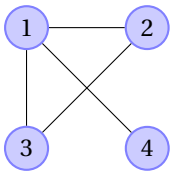
2.3.2. GRAPH STATES WITH LOCAL CLIFFORDS

The second formalism we introduce is the graph state with local (or single qubit) Cliffords, or GSLC for short[21]. To do this, we will first introduce graph states and then extend these with local Cliffords. After that, we will elaborate on how to simulate gates and measurement.

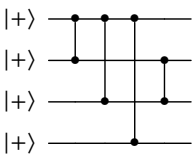
Definition 2.3.1. A n-qubit **graph state** $|G\rangle$ is a quantum state associated with the simple graph $G = (V, E)$ [21]. The $|V| = n$ vertices correspond to the n qubits (that are all in the $|+\rangle$ state), and the edges E correspond to CZ gates acting on the nodes they connect. That is:

$$|G\rangle = \prod_{(a,b)\in E} CZ_{a,b} |+\rangle^{\otimes |V|}. \tag{2.35}$$

As an example, consider the following graph:



The circuit of the corresponding graph state looks as follows:



Because the circuit in equation 2.35 only contains Clifford gates, every graph state is a stabilizer state. Therefore, we can alternatively represent a graph state using n stabilizer generators K_a, defined as:

$$K_a = X_a \prod_{b\in \text{ngbh(a)}} Z_b \quad \text{for all } a \in V. \tag{2.36}$$

If we push through the stabilizers of every qubit using the method described in section 2.2, we obtain $K_1 = XZZZ, K_2 = ZXZI, K_3 = ZZXI, K_4 = ZIIX$, which is equal to the definition in equation 2.36.

It has been shown that any stabilizer state can be transformed to a graph state that is equivalent up to a tensor product of local Clifford [14] [19] [13]. Therefore, we can represent any stabilizer state using a graph state and a tensor product of n single-qubit Cliffords.

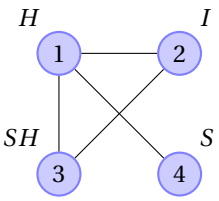
Definition 2.3.2. We define a **GSLC** $\left|G;\overline{C}\right\rangle$ as:

$$\left|G;\overline{C}\right\rangle = \bigotimes_{p=1}^{|V|} C_p \prod_{(a,b)\in E} CZ_{a,b} |+\rangle^{\otimes |V|} \tag{2.37}$$

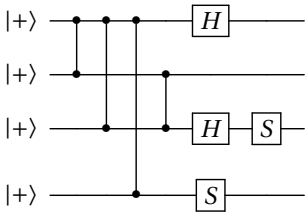
$$= \bigotimes_{p=1}^{|V|} C_p |G\rangle \tag{2.38}$$

where G is a graph with sets V and E of vertices and edges respectively, and C_p is a local Clifford on qubit p.

As an example, consider the following GSLC:



The circuit of this GSLC looks as follows:



We can store a GSLC in space of order $O(n\bar{d})$, where $\bar{d}$ is the average degree of the graph. As there are 24 local Cliffords, we can store the local Cliffords using n numbers between 0 and 23. The graph state can be stored as an adjacency list, where we need to store $\bar{d}$ neighbours on average per node. For instance, we can store the Bell state $\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ as:

Vertex	local Clifford	adjacency
1	H	2
2	I	1

We will now give an overview of how gates are simulated in the GSLC formalism. Single qubit Cliffords can be simulated in time $\Theta(1)$ in GSLC formalism. When one applies a single qubit Clifford $C \in \mathscr{C}_1$ (see 2.1.14) to a qubit a, we replace its local Clifford C_a with CC_a. The product of two Cliffords is also a single qubit Clifford ($CC_a \in \mathscr{C}_1$), and so we can store all possible products in a lookup table of size $24 \times 24 = 576$.

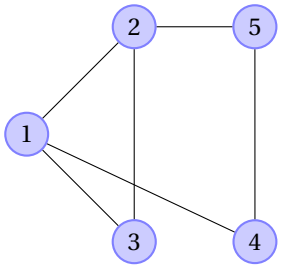
CZ gate simulation is more complicated. It involves changing the underlying graph with an operation called a local complementation, that we will define first.

Definition 2.3.3. A **local complementation** of graph $G = (V, E)$ on vertex a updates G to G' as follows:

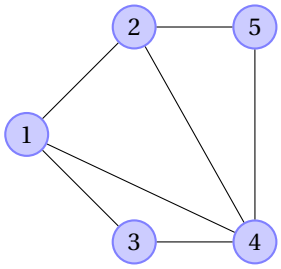
$$G' = (V, E\Delta\{(b,c) | b,c \in ngbh(a)\}). \tag{2.39}$$

That is, all edges between the neighbours of a are toggled.

As an example, consider the following graph:



A local complementation on node 1 yields:



[19] and [18] pose that local complementations transform a graph state into one that is equivalent up to a tensor product of local Cliffords. By updating the local Cliffords of a GSLC after a local complementation, we can therefore obtain a different GSLC representation of the same stabilizer state. More formally:

Theorem 2.3.2. *[21] A GSLC* $\left|G;\overline{C}\right\rangle$ *is invariant under a local complementation of qubit a followed by a right multiplication of the local Clifford* C_a *of qubit a with* $\sqrt{iX}$ *and a right multiplication of the local Clifford* C_b *of all neighbours b of a with* $\sqrt{-iZ}$.

Anders and Briegel note that all operators in the local Clifford group can be written as a product of at most 5 of the operators $\sqrt{-iX}$ and $\sqrt{iZ}$, which are the Hermetian adjoint of the operators in 2.3.2. This allows us to reduce the local Clifford of a qubit a in a GSLC to I by applying a sequence of local complementations on a and a randomly selected swapping partner $c \in nbgh(a)$. This is done in a procedure named 'remove_VOP' ('remove vertex operator') by Anders and Briegel. It takes as input qubits a and b and removes the local Clifford of a by a series of local complementations using a swapping partner not equal to b, if possible. Remove_VOP's run time is dominated by the sequence of local complementations and hence has time complexity $O(d^2)$, where d is the maximum degree of the graph.

Having introduced the remove_VOP procedure, we can elaborate on how CZ gates are simulated. A gate $CZ(a,b)$ falls into one of the following cases:

1. The local CLiffords of both qubits are in $\{I, Z, S, S^\dagger\}$. In this case, the CZ commutes with the local Cliffords, and can be simulated by simply toggling edge (a,b).

2. Both qubits have neighbours, and one or both local Cliffords do not commute with the CZ gate. Here, we use the 'remove_VOP' procedure to reduce both local Cliffords to I, after which the CZ gate commutes and (a,b) can be toggled.

3. One or both qubits have no neighbours or are only connected to the other qubit involved in the CZ. If one of the qubits does have neighbours not involved with the CZ, we first use 'remove_VOP' to reduce the local Clifford of that qubit to I. Afterwards, lookup the result of the CZ gate in a lookup table called 'cphase_table'.

We will finish this section by explaining how to measure a qubit in a GSLC. If we measure a qubit a of a GSLC $\left|G;\overline{C}\right\rangle$ in the computational basis and get the measurement result ζ, the post measurement state we obtain is equal to:

$$\frac{I+(-1)^{\zeta}Z_a}{2}\left|G;\overline{C}\right\rangle, \tag{2.40}$$

which we can rewrite as:

$$(\prod_{b\in V\setminus\{a\}} C_b)C_a\frac{I+(-1)^{\zeta}C_a^{\dagger}Z_aC_a}{2}|G\rangle. \tag{2.41}$$

Because C_a is a Clifford operator, $P_a = C_a^{\dagger}Z_aC_a \in \{\pm X_a, \pm Y_a, \pm Z_a\}$. In other words, to measure qubit a of $\left|G;\overline{C}\right\rangle$ in the computational basis, we measure the observable P_a on the underlying graph state $|G\rangle$. How this underlying graph state changes has been studied before in [18], and is repeated in detail in [21], to which we refer if the reader wants more detail. The time complexity needed to do these updates is dominated by an update of edges that has a complexity of order $O(d^2)$.

2.3.3. COMPARISON OF SIMULATION COMPLEXITY

GSLC and tableaus differ in their time and space complexity, as can be seen in table 2.1.

	Tableau	GSLC
Single qubit Clifford	$O(n)$	$\Theta(1)$
Two qubit gate	$O(n)$	$O(d^2)$
Measurement	$O(n^2)$	$O(d^2)$
Space required	$O(n^2)$	$O(n\bar{d})$

Table 2.1: Complexity comparison of the GSLC and tableau simulation formalism. Here, n is the number of qubits, $d, \bar{d}$ is the maximum and average degree of the underlying graph of the GSLC, respectively.

The simulation complexity of tableaus scale with the number of qubit simulated, whereas the complexity of GSLC scale with the degree of the underlying graph. GSLC formalism is faster and requires less space than tableaus for graphs with low d[21]. Note here that the maximum degree of a state changes during simulation. A sequence of local complementations randomly distributed over a sparse graph will on average increase the degree of that graph, because for a sparse graph it is more likely that a local complementation toggles more edges on rather than off. Therefore, what formalism is preferred is highly application dependent.

3

CANONICAL FORM TABLEAUS

Van den Nest, Dehaene and De Moor, use the binary picture (or check matrix/tableau, as defined in section 2.3.1) of graph states to prove several useful theorems in [19]. These results were later used in the paper defining the GSLC formalism by [21]. In this chapter, we expand on this idea by defining a canonical form for tableaus that directly corresponds to a GSLC. By creating this isomorphism between both representations and investigating how we can restore a tableau to canonical form after an operation, we can improve on the simulation of CZ gates in GSLC form, which we will explain in a later chapter.

[19] showed that an n-qubit stabilizer tableau can be converted to a GSLC representation in time order $O(n^3)$ using a sort of Gaussian elimination. We present a novel algorithm to convert a tableau to a GSLC in time order $O(n^2)$, given that the tableau is in canonical form. We also present a straightforward way of converting a GSLC to a canonical tableau.

This chapter starts by defining what a canonical GSLC form tableau is. We then give two algorithms to convert to and from GSLC and canonical tableau form. Simulating a CZ on a canonical tableau can make it non-canonical. To restore the canonical form, we finalize this chapter by giving an algorithm to keep a tableau canonical after a CZ gate. We use this algorithm to keep a tableau canonical in chapter 4 to improve the simulation of CZ gates in GSLC formalism.

3.1. CANONICAL GSLC FORM

A tableau in canonical GSLC form directly corresponds to the tableau obtained after simulating the circuit of a GSLC. Conversely, a canonical tableau can be converted to a GSLC without using Gaussian elimination. We will first define a canonical tableau. After that, we will give an example.

Definition 3.1.1. A **canonical GSLC form tableau** is equivalent to the tableau one obtains after simulating the circuit of a GSLC in tableau formalism. A $n \times n$ tableau is in canonical GSLC form if and only if it has the following properties:

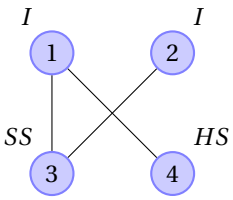
- At every position (i,i) for all $i \in [0,\ldots,n)$, there is an operator $\pm P_{ii} \in \{X,Y,Z\}$.
- Per column j, at all positions (i,j) for all $i \in [0,\ldots,n], i \neq j$, there can be either I or the operator $P_{ij} \in \{X,Y,Z\} \setminus \{P_{ii}\}$.
- For all positions (i,j), $(i,j) = I$ iff $(j,i) = I$.

We interpret the phase of a row in the tableau as the phase of the Pauli on the diagonal that row.

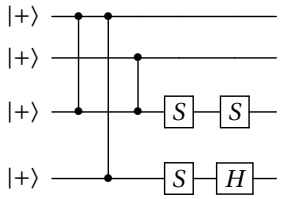
3

Another way to interpret this definition is by the graphical method of determining stabilizer states described in section 2.2. A tableau is in canonical GSLC form when its row i corresponds to the stabilizer generator obtained when 'pushing through' the stabilizer of the i-th qubit trough the corresponding GSLC circuit.

Consider, for example, the following GSLC:



The circuit corresponding to this GSLC looks as follows:



If simulate this circuit in tableau formalism (or 'push through' its stabilizers), we obtain the following tableau:

$$\begin{array}{cc} & XIZX \\ & IXZI \\ - & ZZXI \\ - & ZIIY \end{array} \tag{3.1}$$

This tableau is indeed in canonical form, as it has all the properties described in definition 3.1.1. Alternatively, one can quickly obtain a canonical tableau from a GSLC by first writing down the tableau of the underlying graph state: this tableau has only X on the diagonal, and Z at every position (i,j) and (j,i) if qubits i and j are connected through an edge. The local Cliffords that are appended to the circuit to complete the GSLC state then map the X and Zs in every *column* to some other Pauli, which can be looked up in

graph 2.2. If we consider our previous example, the tableau of its underlying graph state is equal to:

$$\begin{matrix} XIZZ \\ IXZI \\ ZZXI \\ ZIIX \end{matrix} \tag{3.2}$$

The local Clifford SS on qubit 2 then maps all Z, X operators in column 2 to $Z, -X$ respectively. The local Clifford HS on qubit 3 maps all Z, X operators in column 3 to $X, -Y$, thus yielding the final tableau in equation 3.1.

3.2. CONVERTING BETWEEN GSLC AND CANONICAL TABLEAU

Because canonical tableaus and GSLC directly correspond to each other, one can easily be mapped to the other representation. In this section, we provide two algorithms for doing so.

CANONICAL TABLEAU TO GSLC

As stated in the previous section, one can quickly obtain a canonical tableau by first writing down the tableau of the underlying graph state and then mapping the Paulis per column based on the local Cliffords of the qubits. The algorithm we present in this section is based on the inverse of these steps; namely, we first determine what local Cliffords transform the canonical tableau to a tableau corresponding to a graph state and then determine the edges of the GSLC based on the result. We will first give the intuition behind the algorithm, and then present pseudo-code.

The algorithm starts by inferring what the local Cliffords of the qubits are. We do this by determining, per column, what Clifford transforms the Pauli on the diagonal to $+X$, and the Paulis off the diagonal to $+Z$. The algorithm does this with a lookup table. The local Cliffords of the GSLC are equal to the conjugate transpose of the Cliffords that transform the column Paulis to $+X$ and $+Z$. Note that here, the phase of the row of the tableau is appended to the Pauli on the diagonal; that is, only the Paulis on the diagonal can have a negative phase.

The algorithm proceeds to determine the edges of the GSLC by looking at non-identity off-diagonal Paulis. The local Cliffords map the tableau of the underlying graph state to a tableau with different Paulis per column, but they do not change the placement of identity operators. Hence, we can determine what vertices are connected by an edge in the GSLC by looking at where there are non-I entries off-diagonal. If a off-diagonal entry $(i, j) \neq I$, vertex i and j are connected by an edge.

We use the above described algorithm in a procedure called 'Canonical_tableau_to_GSLC', that takes as input a $n \times n$ canonical tableau T, and yields as output a n-qubit GSLC $\left|G;\overline{C}\right\rangle$. It uses a lookup table 'Cliff_(A,B)' that returns the local Clifford C such that $CXC^\dagger = A$ and $CZC^\dagger = B$. In pseudo-code, this yields the following algorithm:

Algorithm 1 Canonical tableau to GSLC representation

procedure CANONICAL_TABLEAU_TO_GSLC(T)
  Initialize $\left|G;\overline{C}\right\rangle$ s.t. $|V|=n, E=\{\emptyset\}, \overline{C}=\bigotimes_{p=0}^{n} C_p = I^{\otimes n}$
  for all $i \in [0\dots n]$ **do**
    $A \leftarrow T(i,i)$
    for all $j \in [0\dots n]$ **do**
      if $T(j,i) \neq I$ **then**
        $B \leftarrow T(j,i)$
        $E \leftarrow E \cup (j,i)$
    $C_i \leftarrow$ Cliff_$(A,B)^\dagger$
  return $\left|G;\overline{C}\right\rangle$

The complexity of this algorithm is $O(n^2)$, as it needs to loop through the n columns of length n.

It must be noted that the above algorithm does not work for non-canonical tableaus, as the rows of the tableau then do not correspond to the stabilizers obtained by pushing them through the circuit of that state. If there are more than two different Paulis of the group $\pm\{X,Y,Z\}$ in a column, we cannot uniquely determine the local Clifford of the corresponding qubit. Likewise, we cannot determine the edges if the diagonal Pauli is not unique or I. As an example of this, consider the tableau corresponding to a fully connected 4 qubit graph that has local Clifford HSSS on the first qubit:

$$\begin{array}{cc} & ZZZZ \\ - & YXZZ \\ - & YZXZ \\ - & YZZX \end{array} \tag{3.3}$$

Applying a CZ between the first and second qubit yields:

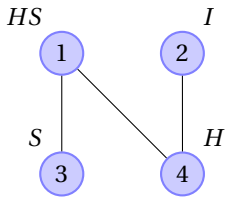
$$\begin{array}{cc} & ZZZZ \\ & XYZZ \\ - & YIXZ \\ - & YIZX \end{array} \tag{3.4}$$

Since this tableau is no longer in canonical form, the local Clifford on the first node and the neighbours of the second node cannot be determined directly.

GSLC TO CANONICAL TABLEAU

We define a way to convert a GSLC to a tableau with time complexity $O(\max(|E|,n))$, where E is the number of edges in the graph and n the number of nodes. This procedure is equivalent to the reverse of the algorithm to convert a canonical tableau to a GSLC. It works as follows: for every node i, we place the Pauli $C_i^\dagger X C_i$ on position (i,i) in the tableau (where C_i is the local Clifford of node i). Then for every edge (a,b), we place Pauli $C_b Z C_b^\dagger$ on (a,b) and $C_a^\dagger Z C_a$ on (b,a). The rest of the entries in the tableau are I.

Consider, for example, the following GSLC:



We start by placing the Pauli's $C_i X C_i^\dagger$ on the diagonal:

$$- \begin{matrix} YIII \\ IXII \\ IIYI \\ IIIZ \end{matrix} \tag{3.5}$$

Then for every edge (a,b), we place Pauli $C_b Z C_b^\dagger$ on (a,b) and $C_a Z C_a^\dagger$ on (b,a):

$$- \begin{matrix} YIZX \\ IXIX \\ XIYI \\ XZIZ \end{matrix} \tag{3.6}$$

thus yielding the canonicial tableau of the GSLC.

3.3. KEEPING A TABLEAU CANONICAL

The canonical form defined in definition 3.1.1 is not always preserved under application of a CZ gate. In this section, we describe a procedure to restore the canonical form of a tableau after simulating a CZ gate. It works by reasoning about how a tableau is transformed by applying $CZ_{a,b}$, given the local Cliffords of qubits a and b. This section starts by examining how local Cliffords change the working of CZs on tableaus. We then give the algorithm's pseudo-code, followed by an example on a tableau. The section is finished with a formal proof of the correctness of part of the algorithm.

In order to facilitate our reasoning about the workings of CZ gates on tableaus, we divide the set of 24 local Cliffords into 3 subsets of equal size:

Definition 3.3.1. *Clifford Types* Cliffords can be divided into three types, containing 8 Cliffords each:

- 'Z type', mapping Z to $\pm Z$
- 'H type', mapping X to $\pm Z$
- 'XY type', mapping Y to $\pm Z$

Note that swapping or multiplying the rows of a tableau does not change the state a tableau represents, as was stated in the properties of a tableau in section 2.3.1. Because the Paulis in the columns corresponding to qubit a and b before applying $CZ_{a,b}$ are a direct function of the local Cliffords of qubit a and b, we can infer the local Clifford of

$CXC^\dagger =$ \ $CZC^\dagger =$	X	Y	Z	-X	-Y	-Z
X		SHS	I		SSSHSSS	HSSH
Y	HSSS		S	SSHS		SHSSH
Z	H	SH		SSH	SSSH	
-X		SHSSS	SS		SSSHS	SSHSSH
-Y	HS		SSS	SSHSSS		SSSHSSH
-Z	HSS	SHSS		SSHSS	SSSHSS	

Table 3.1: Table that classifies the 24 local Cliffords by clifford type. Green cells are Z type, yellow cells are H type, and blue cells are XY type local Cliffords.

both qubits from the Paulis in their column, as in algorithm 1. After determining what local Clifford is on both qubits, we can translate this to a Clifford type. Our algorithm to restore a tableau back to canonical form after a CZ on two qubits chooses how to do so based on the types of the local Cliffords on both qubits. We distinguish the cases (Z type, Z type), (Z type, non Z) and (non Z, non Z):

- **(Z type, Z type)** When both qubits have a Z type local Clifford, the tableau stays in canonical form after simulating a CZ gate. [21] states that on a GSLC a CZ where both qubits have a local Clifford that maps Z to Z can be applied by simply toggling an edge. Our results implies that this is also the case for local Cliffords that map Z to -Z.

- **(Z type, non Z)** When one of the qubits has a Z type local Clifford, the tableau is restored to canonical form by multiplying the row of the Z type qubit with the row of the non Z qubit.

- **(non Z, non Z)** If neither qubit has a Z type qubit, the procedure is more involved.

The pseudo-code of the algorithm looks as follows:

Algorithm 2 Restore canonical form of tableau T after $CZ(a,b)$

```
1: type a ← local Clifford type of qubit a
2: type b ← local Clifford type of qubit b
3: neighbours a ← {i|for all i ≠ b s.t. T(i,a) ≠ I}
4: neighbours b ← {i|for all i ≠ a s.t. T(i,b) ≠ I}
5: connected ← true if T(a,b) ∧ T(b,a) ≠ I
6: Apply CZ(a,b) on T
7: if type a is Z type then
8:     row a ← row a · row b
9: else if type b is Z then
10:    row b ← row a · row b
11: else
12:    if type a or type b is XY type then
13:        for all neighbour in neighbours a do
14:            row neighbour ← row neighbour · row a
15:        for all neighbour in neighbours b do
16:            row neighbour ← row neighbour · row b
17:        if type a is H then
18:            row b ← row a · row b
19:        else if type b is H then
20:            row a ← row a · row b
21:    else
22:        for all neighbour in neighbours a do
23:            row neighbour ← row neighbour · row b
24:        for all neighbour in neighbours b do
25:            row neighbour ← row neighbour · row a
26:    if connected then
27:        swap row a and b
```

As an example of the above algorithm, take the following tableau of a GSLC with a XY type local Clifford on the first qubit, a H type on the third and Z type elsewhere:

$$\begin{matrix} XZXZ \\ YXII \\ YIZZ \\ YIXX \end{matrix} \tag{3.7}$$

We will apply $CZ_{1,3}$ on the tableau and restore it to canonical form. Before we start the actual algorithm, we store a number of variables we need later in lines 1-5. The first qubit has all the other qubits as neighbours, and the third qubit has the first and fourth qubit as neighbours. The first and third qubit are connected. We then apply the CZ on the first

and third qubit in line 6:

$$\begin{array}{cc} - & YZYZ \\ & YXZI \\ & YIIZ \\ & XIYX \end{array} \tag{3.8}$$

Now the tableau is not in canonical form anymore, and we need to restore it. Since the first qubit has a local Clifford of type XY, we jump to line 14 and multiply every row of its neighbours (except the third qubit) with the first row:

$$\begin{array}{cc} - & YZYZ \\ & IYXZ \\ & YIIZ \\ - & ZZIY \end{array} \tag{3.9}$$

We then do the same for the third qubit in line 16, multiplying the row of its neighbour with the third row:

$$\begin{array}{cc} - & YZYZ \\ & IYXZ \\ & YIIZ \\ - & XZIX \end{array} \tag{3.10}$$

Since the Clifford type of third qubit is H, the first row gets multiplied with the third in line 20:

$$\begin{array}{cc} - & IZYI \\ & IYXZ \\ & YIIZ \\ - & XZIX \end{array} \tag{3.11}$$

And finally, since qubits one and three were connected, we swap their rows to restore the tableau to canonical form in line 27.

$$\begin{array}{cc} & YIIZ \\ & IYXZ \\ - & IZYI \\ - & XZIX \end{array} \tag{3.12}$$

A special case of our algorithm can be found in section 4.4.3 of a paper by van den Berg and Temme [53]. Here, the authors start with a diagonal X-matrix and a symmetric Z-matrix; which is equivalent to a canonical tableau with only Z-type local Cliffords. They then perform a $CNOT_{a,b}$ gate between two qubits a and b, which is equivalent to simulating $H_a CZ_{a,b} H_a$. This makes it equivalent to a (Z type, non Z) CZ gate. Finally, they restore symmetry in the same way that our procedure does: by multiplying the Z row (the source row of the CNOT) with the non Z row (the target row).

CORRECTNESS PROOF OF THE (Z TYPE, Z TYPE) AND (Z TYPE, NON Z TYPE) CASES OF ALGORITHM 2

We give a proof of correctness of the **(Z type, Z type)** and **(Z type, non Z)** cases of algorithm 2, as we base a new algorithm, algorithm 3, for simulating CZ gates on GSLC on those two cases in the next chapter. The **(non Z, non Z)** cases follow similar logic, but we omit these for brevity and because these cases are not further used in this thesis. We divide the proof up in a number of cases, based on the Clifford type of both qubits a and b the gate $CZ(a,b)$ works on. Since a CZ gate is symmetric, each case holds for both $CZ(a,b)$ and $CZ(b,a)$. For simplicity, we ignore phases, as they do not change the arguments of our proof: from the definition of 3.1.1 these phases correspond to the phases of the Paulis on the diagonal and hence preserve a qubits Clifford type, as can be checked in table **??**.

Proof. Because the group of stabilizer generators is closed under multiplication, swapping or multiplying the rows of a tableau does not change the state a tableau represents. Therefore, using those two operations to transform a tableau to canonical form preserves the state of the tableau. In order to determine how to do so using those operations, we reason from what Paulis can be in the column of a qubit with a certain Clifford type, the properties of a canonical tableau, and the workings of a CZ gate on a tableau. For each Clifford type, the following Paulis can be in column i corresponding to qubit i:

- **Z type**:
$$(k,i) = \begin{cases} X \vee Y & \text{if } k = i \\ Z & \text{otherwise.} \end{cases}$$
- **H type**:
$$(k,i) = \begin{cases} Z & \text{if } k = i \\ X \vee Y & \text{otherwise.} \end{cases}$$
- **XY type**:
$$(k,i) = \begin{cases} X \vee Y & \text{if } k = i \\ X \vee Y & \text{otherwise.} \end{cases}$$

The symmetry property of a canonical tableau can be defined as follows: iff $(i,j) = I$, then $(j,i) = I$. Since there can be only one off-diagonal Pauli, the following holds as well:

$$(k,i) = \begin{cases} P_1 & \text{if } k = i \\ P_2 \vee I & \text{otherwise,} \end{cases}$$

where P_1 and P_2 are Paulis and $P_1 \neq P_2$.
We define $(i,j)'$ and $(i,j)''$ as the Pauli at position i,j after simulating a CZ, and after the subsequent application of algorithm 2 respectively. When applying a $CZ(a,b)$ on two qubits in a tableau, the Paulis in column $a(b)$ change as follows: $(k,a(b))' = (k,a(b)) \cdot Z$ iff $(k,b(a)) = X \vee Y$. A $CZ(a,b)$ falls in one of the following Clifford type categories:

- **(Z type, Z type)** Since only (a,a) and (b,b) are X or Y, simulating the CZ changes the tableau as follows:

$$(a,b)' = (a,b)\cdot Z \tag{3.13}$$
$$(b,a)' = (b,a)\cdot Z, \tag{3.14}$$

hence preserving all the properties of a canonical tableau and requiring no correction.

- **(Z type, H type)** Here, applying $CZ(a,b)$ changes the tableau as follows:

$$(k,a)' = (k,a)\cdot Z \qquad \text{If } k\neq b \text{ and } (k,b)\neq I \tag{3.15}$$
$$(a,b)' = (a,b)\cdot Z \tag{3.16}$$

We bring the resulting tableau back to canonical form by multiplying row a with row b, resulting in:

$$(a,k)'' = (a,k)'\cdot(b,k)' \tag{3.17}$$

If $k=a$, we get:

$$(a,a)'' = \begin{cases} (a,a)\cdot Z\cdot Z = (a,a) & \text{if } (a,b)\neq I \\ (a,a) & \text{if } (a,b) = I, \end{cases} \tag{3.18}$$

since $(a,b)=I$ iff $(b,a)=I$, by the symmetry of a canonical tableau. For $k=b$, we get:

$$(a,b)'' = (a,b)\cdot Z\cdot(b,b) = (a,b) \tag{3.19}$$

For all other k, we get:

$$(a,k)'' = \begin{cases} (a,k)\cdot(b,k) = (a,k) & \text{if } (b,k) = I \\ (a,k)\cdot Z & \text{if } (b,k)\neq I \end{cases} \tag{3.20}$$

Since a canonical tableau is symmetric, $(b,k)\neq I$ iff $(k,b)\neq I$, and hence $(k,a)'\neq I$ iff $(a,k)''\neq I$, restoring symmetry in the tableau. Because no other entries are changed, the tableau is back in canonical form.

For example:

$$\begin{matrix} X\,X\,Z\,Z \\ Z\,Z\,Z\,I \\ Z\,X\,X\,I \\ Z\,I\,I\,X \end{matrix} \xrightarrow{\text{CZ}(0,1)} \begin{matrix} Y\,Y\,Z\,Z \\ Z\,Z\,Z\,I \\ I\,X\,X\,I \\ Z\,I\,I\,X \end{matrix} \xrightarrow{\text{row } 0\cdot 1} \begin{matrix} X\,X\,I\,Z \\ Z\,Z\,Z\,I \\ I\,X\,X\,I \\ Z\,I\,I\,X \end{matrix} \tag{3.21}$$

- **(Z type, XY type)** simulating the CZ changes the tableau as follows:

$$(a,b)' = (a,b)\cdot Z \tag{3.22}$$
$$(k,a)' = (k,a)\cdot Z \quad \text{If } (k,b)\neq I. \tag{3.23}$$

We bring the resulting tableau back to canonical form by multiplying row a with row b, resulting in:

$$(a,k)'' = (a,k)'\cdot(b,k)' \tag{3.24}$$

If $k = a$, we get:

$$(a,a)'' = \begin{cases} (a,a)\cdot Z\cdot(b,a)\cdot Z = (a,a)\cdot Z & \text{if } (a,b)\neq I \\ (a,a)\cdot(b,a)\cdot Z = (a,a)\cdot Z & \text{if } (a,b) = I, \end{cases} \tag{3.25}$$

since $(a,b) = I$ iff $(b,a) = I$, by the symmetry of a canonical tableau. For $k = b$, we get:

$$(a,b)'' = (a,b)\cdot Z\cdot(b,b) \tag{3.26}$$

Since $(b,a)' = (b,a)\cdot Z$, and $(a,b)\neq I$ iff $(b,a)\neq I$, symmetry of the tableau is preserved. Since $Z\cdot(b,b)$ is equal to the off-diagonal Pauli in column b, the maximum of one other off-diagonal Pauli is preserved as well.

For all other k, our argument is the same as for the (Z type, H type) case:

$$(a,k)'' = \begin{cases} (a,k)\cdot(b,k) = (a,k) & \text{if } (b,k) = I \\ (a,k)\cdot Z & \text{if } (b,k)\neq I \end{cases} \tag{3.27}$$

Since a canonical tableau is symmetric, $(b,k)\neq I$ iff $(k,b)\neq I$, and hence $(k,a)'\neq I$ iff $(a,k)''\neq I$, restoring symmetry in the tableau. Because no other entries are changed, the tableau is back in canonical form.

For example:

$$\begin{matrix} X & I \\ I & Y \end{matrix} \xrightarrow{\text{CZ(0,1)}} \begin{matrix} X & Z \\ Z & Y \end{matrix} \xrightarrow{\text{row 1 to 0}} \begin{matrix} Y & X \\ Z & Y \end{matrix} \tag{3.28}$$

□

4

FASTER CZS FOR GSLC

Aaronson and Gottesman introduced an efficient way to simulate stabilizer states with the tableau formalism in [17]. Subsequently, Anders and Briegel presented a more efficient way to simulate the set of stabilizer states that can be represented as sparse graph states with local Cliffords in [21]. In this chapter, we present two contributions that improve the simulation of CZ gates in GSLC formalism:

1. An algorithm for simulating single CZ gates. This algorithm is faster than the original algorithm in all cases, except for CZ gates that commute with the local Cliffords of the qubits they are applied on (which has run time $\Theta(1)$), where it is equally fast. It also requires less local complementations than the original algorithm. This can provide an additional speedup when simulating sequences of CZ, as CZ gates on states with a high degree are more expensive to simulate (see section 2.3), and local complementations on sparse graphs are likely to increase the degree of a graph.

2. An algorithm for simulating sequences of CZ by smartly scheduling and subsequently simulating them using our algorithm for single CZ gates. This algorithm provides a speedup on 'concentrated' sequences of CZ; sequences of CZ gates that share a small number of qubits they work on, such as a star graph.

The algorithm for simulating single CZ gates is based on the 'CZ_ONE_Z' procedure for CZs between a Z type qubit and a non-Z type qubit (as defined in 3.3.1), that we describe first. Afterwards, we give the full algorithm for all types of CZ, analyze its complexity, and empirically validate its performance. The last section of this chapter describes the sequence scheduling algorithm. We describe the algorithm and the intuition behind it, and empirically validate its performance.

4.1. THE 'CZ_ONE_Z' PROCEDURE

As stated in section 3.3, we can improve the simulation of single CZ gates in GSLC formalism by considering their workings on a canonical tableau. 'CZ_ONE_Z, an algorithm

to perform a **(Z type, non Z)** type CZ in GSLC form, was derived in such a way. In this section, we describe the 'CZ_ONE_Z' procedure by first showing the reasoning used to transform lines 1-10 of algorithm 2 to the GSLC formalism, and subsequently giving its pseudo-code.

Algorithm 2 restores the canonical form of a tableau after a CZ on a pair of qubits (a,b) of **(Z type, non Z)** respectively by multiplying row a with row b. We convert this application of a CZ and subsequent restoring of canonical form to GSLC form. We go over the lines of the algorithm to explain the reasoning behind them. Here, we use the same notation for entries in the tableau representation of the state, that is: (a,b) refers to the Pauli at row a, column b before the CZ, $(a,b)'$ refers to that same entry after applying the CZ and $(a,b)''$ is entry after the multiplication of row a with row b. The correctness of all local Clifford multiplications can be understood by table **??**.

- **[1-20]** Algorithm 2 is based on the assumption that in a canonical tableau phases of the rows are interpreted as being placed on the diagonal (as stated in section 3.1). That is, if row a has a negative phase, (a,a) has a negative phase and all other Paulis in row a have a positive phase. Hence, to be able to convert algorithm 2 to GSLC formalism, we need the local Cliffords qubits a and b to map Z to a positive Pauli. If a or b's local Cliffords does not, we multiply the local Clifford such that it maps Z to the same Pauli but with positive phase, and X to the same Pauli with the same phase in lines 8-13. We multiply the local Cliffords C_c of all neighbours c of this qubit such that the phase of C_c^c is multiplied with -1 in lines 14-19. Therefore, all rows in the tableau representation of the state maintain the same phase and Paulis, and thus lines 1-20 do not change the state. We note that the local Cliffords of the other qubits do not need to map Z to a positive Pauli in order for our reasoning to be correct.

- **[21-22]** For every $k \neq a \vee b$ s.t. $(k,b) \neq I$, all entries $(a,k)'' = (k,a)' = I$ if $(a,k) \neq I$, or conversely $(a,k)'' = (k,a)' \neq I$ if $(a,k) = I$ (as can be seen in equations 3.16, 3.20, 3.23 and 3.27 in the proof of algorithm 2). As stated in section 3.1, if entries $(a,k) \neq I$ and $(k,a) \neq I$ in a canonical tableau, there exists an edge between nodes a and k in the corresponding GSLC. Therefore, the corresponding operation in GSLC formalism is a toggling of all edges between the neighbours of b and qubit a: $E\Delta\{(a,c)|c \in \text{ngbh}(b)\}$.

- **[23-33]** If b was a XY type qubit, a number of additional changes occur to the state:

  - (b,b) was X or Y, and thus $(a,b)'' = (a,b)\cdot Z\cdot(b,b) = I$ if $(a,b) \neq I$, or conversely $(a,b)'' = Z\cdot(b,b) \neq I$ if $(a,b) = I$ (see equation 3.26). Also, $(b,a)'' = (b,a)\cdot Z$. Therefore, the edge between a and b is toggled in line 24.
  - If the qubits were connected and $\pm C_a X C_a^\dagger = C_b X C_b^\dagger$ (or equivalently in tableau form, if $\pm(a,a) = (b,b)$) the CZ and subsequent row multiplication of algorithm 2 introduce a phase to row a. Therefore, we left-multiply the local Clifford of a with SS in line 26. The same holds if the qubits were not connected and $\pm C_a X C_a^\dagger \neq C_b X C_b^\dagger$, in which case we left-multiply the local Clifford of a with SS in line 28.

  - $(a,a)'' = (a,a) \cdot Z$ (see equation 3.25). This changes the Pauli at (a,a) from $X(Y)$ to $Y(X)$. To capture the change of Pauli at (a,a) in the GSLC formalism, we left-multiply local Clifford C_a in lines 29-32 such that we change the Pauli C_a^a from $X(Y)$ to $Y(X)$, but maintain its phase.

- **[34-36]** If b was a H type qubit and both qubits were connected, the CZ and subsequent row multiplication of algorithm 2 introduce a phase to row a. In this case we therefore multiply the local Clifford of a with SS in line 36.

- **[37-38]** If row b had a negative phase (i.e. if $C_b X C_b^\dagger = (-X \vee -Y \vee -Z)$), the phase of row a is multiplied with -1 in the row multiplication step of algorithm 2. To account for this, we left-multiply C_a with SS.

The pseudo-code for CZ_ONE_Z is shown in algorithm 3.

Algorithm 3 CZ on Z type qubit a and non-Z type qubit b of GSLC $\left|G;\overline{C}\right\rangle$

procedure CZ_ONE_Z(a, b)
  type a← Clifford type qubit a
  type b← Clifford type qubit b
  neighbours a ← $\{i|$for all i s.t. $(a,i) \in E\}$
  neighbours b ← $\{i|$for all i s.t. $(b,i) \in E\}$
  connected ← true if $(a,b) \in E$
  for all $i \in \{a,b\}$ **do**
    if $C_i Z C_i^\dagger = (-X \vee -Y \vee -Z)$ **then**
      if type i is XY **then**
        $C_i \leftarrow SSC_iSS$
      else if type i is H **then**
        $C_i \leftarrow SSC_i$
      else
        $C_i \leftarrow XC_i$
      for all c in neighbours i **do**
        type c← Clifford type qubit c
        if type c is Z **then**
          $C_c \leftarrow SSC_c$
        else
          $C_c \leftarrow C_cSS$
  for all $c \in$ neighbours b $\setminus a$ **do**
    $E \leftarrow E\Delta(a,c)$
  if type b is XY **then**
    $E \leftarrow E\Delta(a,b)$
    if connected and $\pm C_a X C_a^\dagger = C_b X C_b^\dagger$ **then**
      $C'_a \leftarrow SSC'_a$
    else if not connected and $\pm C_a X C_a^\dagger \neq C_b X C_b^\dagger$ **then**
      $C'_a \leftarrow SSC'_a$
    if $\pm C_a X C_a^\dagger = \pm X$ **then**
      $C'_a \leftarrow SC'_a$
    else
      $C'_a \leftarrow SSSC'_a$
    $C_a \leftarrow C'_a$
  if type b is H **then**
    if connected **then**
      $C_a \leftarrow SSC_a$
  if $C_b X C_b^\dagger = (-X \vee -Y \vee -Z)$ **then**
    $C_a \leftarrow SSC_a$

The complexity of the algorithm is $O(d)$, as opposed to the the $O(d^2)$ complexity of the original CZ algorithm by [21]. Its correctness follows from the correctness of algorithm 2 and the fact that a canonical tableau directly corresponds to a GSLC state: it is

simply a direct mapping of lines 1-10 of algorithm 2 to GSLC form.

We have implemented algorithm 3 in Python. Our code is partially based on a publicly available implementation of the original GSLC algorithm [57]. In order to verify the correctness of our implementation, we compare the simulation results of our implementation of algorithm 3 with that of a Python implementation of the tableau-based 'CHP' simulator by Aaronson and Gottesman [17]. We first generate a random GSLC state, which we represent as a GSLC and convert to a tableau by simulating its circuit on a tableau in the all zero state. We then simulate a random **(Z type, non Z)** CZ gate on these two representations of the state using both algorithms. Afterwards, we restore the tableau to canonical form by applying algorithm 2. Finally, we convert the resulting GSLC to a tableau representation, and compare both tableaus. After simulating thousands of random CZ gates on random states we were not able to find any non-identical tableaus. As the probability of two independent algorithms making the same error is low, this provides strong evidence that our implementation of 3 is correct.

4.2. A NOVEL ALGORITHM FOR SIMULATING CZ GATES

In this section, we present a novel algorithm to simulate all types of CZ gates that uses the 'CZ_ONE_Z' procedure. It is faster for most non-commuting single CZ gates and equally fast in all other cases, compared to the original algorithm (as can be seen in figure 4.1). It also provides an additional speedup when simulating sequences of CZ by requiring less local complementations, and improves the run time of 'concentrated' sequences of CZ. We first give intuition behind the algorithm, followed by its pseudo-code. We then analyze its complexity.

We go over the lines of the algorithm to explain the reasoning behind them:

- **[1-11]** When both qubits have a Z-type local Clifford, the CZ commutes. We simulated using a toggling of the edge between qubits a and b. If the a qubit's local Clifford maps Z to $-Z$, commuting this CZ through the local Cliffords introduces a phase on the row of the other qubit in the tableau. To account for this, we left-multiply the local Clifford of the other qubit with SS. The correctness of this multiplication can be inferred from table **??**.

- **[12-15]** If one of the qubits has a Z-type local Clifford, we simulate the CZ by a call to algorithm 3.

- **[16-26]**When one of the qubits is only connected to the other and both qubits are non Z type, we use the same lookup-table 'cphase_table' as in the original algorithm (as explained in subsection 2.3.2).

- **[27-33]** If both qubits are non-Z type, we do a series of local complementations on the qubit with the fewest neighbours by a call to 'remove_VOP*', so that it has local Clifford I (and becomes Z-type). We can then simulate the CZ by a call to 'CZ_ONE_Z',

Because 'CZ_ONE_Z' can simulate **(Z-type, non Z)** CZs faster, we would like to have as many Z-type qubits in our state as possible. However, if a Z-type qubit is selected

as 'swapping partner' in the 'remove_VOP' procedure, performing a local complementation on the qubit can change its Clifford type. This might slow down the simulation of subsequent CZ gates. Therefore, we adjust the 'remove_VOP' procedure from its original definition. We name this new procedure 'remove_VOP*':

Definition 4.2.1. The **remove_VOP* procedure** on qubits a and b is the 'remove_VOP' procedure as defined by [21] (see subsection 2.3.2), but where a non-Z type qubit is selected from the set of neighbours of a (excluding b) as a swapping partner, if possible.

The pseudo-code for a CZ gate is shown in algorithm 4.

The complexity of algorithm 4 is dependent on the Clifford type of the qubits the CZ is applied on:

- When both qubits have a Z-type local Clifford, the CZ is simulated using a toggling of the edge between qubits a and b and possibly an update of the local Cliffords of the nodes in time of order $\Theta(1)$.
- When one of the qubits has a Z-type local Clifford, the CZ is simulated using a $O(d)$ call to 'CZ_ONE_Z'.
- When both qubits are non Z-type, the algorithm makes a $O(d^2)$ call to remove_VOP*, followed by a $O(d)$ call to 'CZ_ONE_Z'.
- When one of the qubits is only connected to the other and both qubits are non Z type, our algorithm is identical to the original algorithm: it makes at most 3 $O(d^2)$ calls to 'remove_VOP*', followed by a lookup in table 'cphase_table'.

We compare the complexity of algorithm 4 with the complexity of the original algorithm by [21] described in subsection 2.3.2 for the cases where both qubits are connected to other qubits in figure 4.1.

4.2.1. EMPIRICAL VALIDATION

We empirically validate the performance of algorithm 4 on single CZ gates and CZ sequences, and compare it to the performance of the original algorithm. To validate our performance and compare it to the run time of the original algorithm, we time the simulation of CZ gates on random states in GSLC formalism using both algorithms in our own Python implementation (partially based on the code found in [57]), on cloud-based hardware that has up to 128 CPU cores available. The random n qubit states used are generated by simulating a random circuit that is obtained by a method similar to that described in [17]:

Definition 4.2.2. Procedure for generating a random circuit. For some factor β, we choose $\lfloor \beta n log_2(n) \rfloor$ random unitary gates from $\{CNOT_{a,b}, H_a, S_a\}$, each with probability 1/3. a and b are drawn uniformly at random from the qubits, such that $a \neq b$.

Algorithm 4 Faster CZ algorithm on GSLC $\left|G;\overline{C}\right\rangle$

procedure CZ(a, b)
  type a← Clifford type qubit a
  type b← Clifford type qubit b
  neighbours a ← $\{i|$for all $i \neq b$ s.t. $(a,i) \in E\}$
  neighbours b ← $\{i|$for all $i \neq a$ s.t. $(b,i) \in E\}$
  if type a is Z & type b is Z **then**
    $E \leftarrow E\Delta(a,b)$
    if $C_a Z C_a^\dagger = -Z$ **then**
      $C_b \leftarrow SSC_b$
    if $C_b Z C_b^\dagger = -Z$ **then**
      $C_a \leftarrow SSC_a$
  else if type a is Z **then**
    CZ_ONE_Z(a,b)
  else if type b is Z **then**
    CZ_ONE_Z(b,a)
  else if neighbours a = ∅ or neighbours b = ∅ **then**
    if neighbours a ≠ ∅ **then**
      remove_VOP*(a,b)
    neighbours b ← $\{i|$for all $i \neq a$ s.t. $(b,i) \in E\}$
    if neighbours b ≠ ∅ **then**
      remove_VOP*(b,a)
    neighbours a ← $\{i|$for all $i \neq b$ s.t. $(a,i) \in E\}$
    if neighbours a ≠ ∅ **then**
      remove_VOP*(a,b)
    edge ← true if $(a,b) \in E$ else false
    (edge, C_a, C_b) ← cphase_table[edge, C_a, C_b]
  else
    if |neighbours a| < |neighbours b| **then**
      remove_VOP*(a,b)
      CZ_ONE_Z(a,b)
    else
      remove_VOP*(b,a)
      CZ_ONE_Z(b,a)

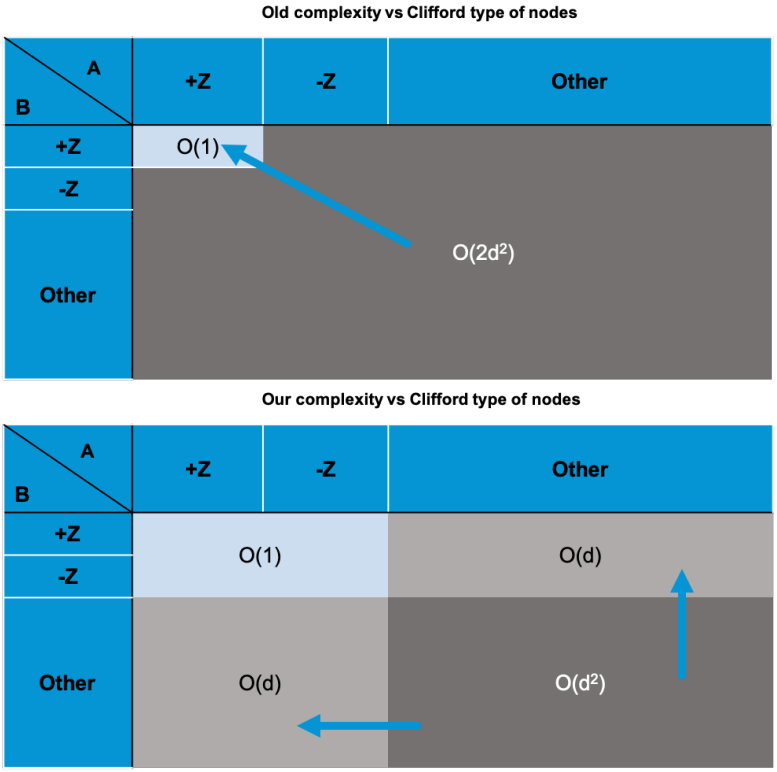


Figure 4.1: Time complexity of simulating a CZ gate on two qubits with neighbours with the original CZ algorithm by [21] and with our algorithm, per combination of Clifford type of qubits a and b. The arrows indicate the use of the 'Remove_VOP' procedure; in the original algorithm, it is applied twice: once for both nodes to make the CZ gate commute with the local Cliffords. In our algorithm, it is applied only on the qubit with the fewest neighbours, after which 'CZ_ONE_Z' is called.

SIMULATING SINGLE CZ GATES

We empirically validate the performance of algorithm 4 on single CZ gates and compare it to the performance of the original algorithm. Because algorithm 4 has an equal or lower run time complexity than the original algorithm in all cases, we expect it to be faster when simulating random single CZ gates on random states. We test this hypothesis by generating random states using procedure 4.2.2. For the random $CZ_{a,b}$ gates, we draw a and b uniformly at random from the n qubits. We then simulate a random gate on a random state using both our algorithm and the original algorithm, and measure the time needed.

As can be seen in figure 4.2, our algorithm achieves an average speedup of up to 9x compared to the original algorithm when simulating random single CZ gates in our runs, confirming our hypothesis. The relative speedup of our algorithm is larger for states with higher β, i.e. states with higher degree. Both the absolute and relative difference in run time increases with the number of qubits.

SIMULATING SEQUENCES OF CZ GATES

We validate our claim that algorithm 4 provides an additional speedup for sequences of CZ empirically. We hypothesise that algorithm 4 provides an additional speedup for sequences for two reasons.

First, because it requires less local complementations than the original algorithm. CZ gates on states with a high degree are more expensive to simulate (see section 2.3), and local complementations on sparse graphs are likely to increase the degree of a graph. Therefore, the average run time of simulating a CZ on a random sparse state increases per CZ simulated, up to some maximum. We expect this increase in run time to be less large when using algorithm 4 compared to the original algorithm, because it uses less local complementations.

Second, because we expect the number of Z-type qubits to increase, up to some maximum, with the number of CZ simulated. Because the 'REMOVE_VOP*' procedure attempts to pick a non-Z swapping partner, it will increase the number Z-type qubits by 1 if it succeeds in picking a non-Z swapping partner. Since **(Z-type, non Z)** and **(Z-type, Z-type)** CZ gates are less expensive to simulate, this decreases the average simulation time of the CZ gates later in the sequence when using our algorithm.

We validate these hypotheses by timing the simulation of a random CZ sequence of n CZ gates on a random state with n qubits in GSLC formalism using both algorithms. The state and CZ gates in the sequence are generated in the same way as in the single CZ experiment. We plot the average simulation time per CZ gate versus the index of the CZ gate in the sequence, i.e. the leftmost values are the run times of the CZ gates at the start of the sequence.

As can be seen in figure 4.3, the run time of simulating a single CZ gate with the original algorithm indeed increases for the CZs later in the sequence. The increase is steepest for the first number of CZ gates, before becoming more gradual. The run time of single CZ gates in a sequence using algorithm 4 decreases slightly with every CZ simulated, thus providing evidence our hypotheses are correct. It must be noted that this result holds for random sequences and gates, but will not hold for every circuit simulated. E.g. for a trivial circuit containing only commuting CZ gates, the run time of a gate will not change with the index.

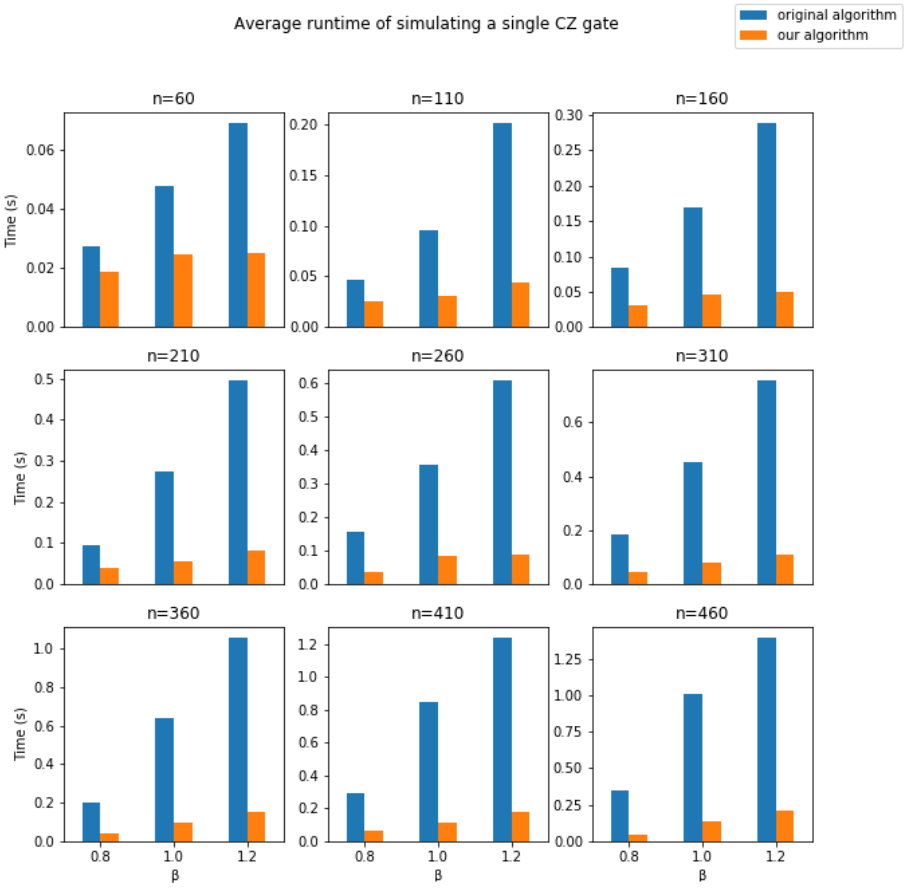


Figure 4.2: Average simulation time of a random single CZ gate on a randomly generated state using the original algorithm by [21] versus algorithm 4, for various number of qubits n and factor β (as defined in definition 4.2.2). We generate 100 pairs of a state and CZ gate to run per n and β, and show the average measured run times over those 100 runs.

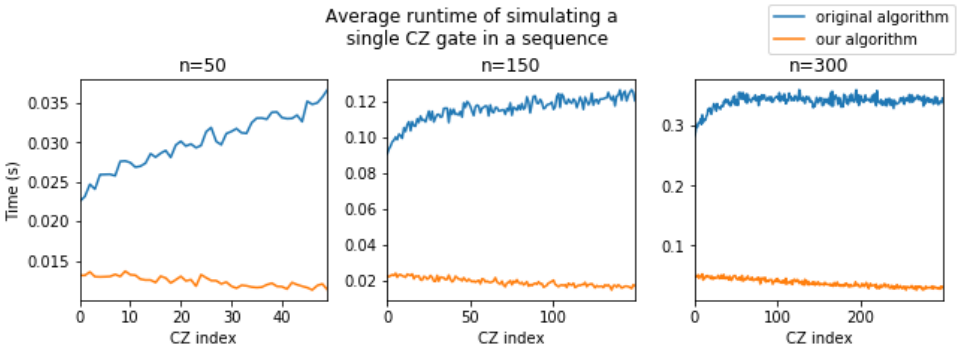


Figure 4.3: Average run time of simulating a single CZ gate in a sequence versus the index of the CZ gate in that sequence, for algorithm 4 and the original CZ algorithm by [21]. Every data-point consists of the average of 1000 different sequences simulated on a different random state each.

4.3. CZ SEQUENCE SCHEDULER

In this section, we present an algorithm for scheduling sequences of CZ and subsequently simulating them using our algorithm for single CZ gates. This algorithm provides a speedup additional to that of our novel single CZ algorithm when simulating concentrated CZ sequences; i.e. sequences where the CZ gates share a small number of nodes. It does this by minimizing the number of local complementations needed to simulate the sequence, which achieves a speedup in two ways:

1. Local complementations in the 'remove_VOP*' procedure (see 4.2.1) are the most expensive step in the CZ algorithm with complexity $O(d^2)$, where d is the maximum degree of the GSLC (as stated in subsection 2.3.2).

2. Local complementations can increase d in sparse graphs with each subsequent local complementation, increasing the run time of the CZ gates at the end of the sequence (as shown in the previous section).

Because CZ gates commute with each other, we can simulate the CZ gates in a sequence of CZs in an arbitrary order. Algorithm 3 allows us to minimize the number of calls to 'remove_VOP*' (and thus the number of local complementations) when simulating a sequence by scheduling the CZs in a smart way. If multiple CZ gates in a sequence operate on qubit a, we can call 'remove_VOP*' once on qubit a and then use algorithm 3 to simulate all CZ gates on qubit a. As an example: when one performs a sequence of CZs that all originate from one qubit on a state (e.g. a star graph, such as the graph of a GHZ state), only one call to 'remove_VOP*' is needed, instead of $O(n)$ as with the original algorithm by [21].

The scheduling algorithm presented here aims to minimize the number of calls to 'remove_VOP*' by greedily selecting the node a that the most CZ gates in the sequence work on. If this node does not yet have a Z type local Clifford, 'remove_VOP*' is called to change its local Clifford to I. Since now all CZs on node a have at least one Z type Clifford, algorithm 3 can now be used to simulate all CZs on this node. We repeat this

cycle until all CZs in the sequence have been performed. If node a is a non Z-type qubit that does not have any neighbours, 'remove_VOP*' cannot be called as it needs to have a 'swapping partner' to do the necessary local complementations. We deal with this by first simulating a randomly selected CZ on a from the sequence, until a has a neighbour or no more CZs on a are left. After this, all remaining CZs on a are simulated, and the algorithm's cycle starts over.
The pseudo-code of the above described algorithm can be seen in algorithm 5.

Algorithm 5 Scheduling and simulating CZ sequence $\{CZ\}$ on GSLC with $\left|G;\overline{C}\right\rangle$.

```
1:  procedure CZ_SEQUENCE({CZ})
2:      G' ← (V, {CZ})
3:      while |{CZ}| > 0 do
4:          a ← max_{a∈V} d_{G'}(a)
5:          neighbours a ← {i|(a, i) ∈ E}
6:          type a← Clifford type qubit a
7:          if type a is not Z then
8:              while d_{G'}(a) > 0 and |neighbours a| = 0 do
9:                  b ← random b s.t. (a, b) ∈ CZ
10:                 CZ(a,b)                    ▷ Simulate the CZ gate using algorithm 4
11:                 {CZ} ← {CZ} \ {(a, b)}
12:             if d_{G'}(a) > 0 then
13:                 c ← random node c ∈ neighbours a
14:                 remove_VOP*(a,c)
15:         for all b s.t. (a, b) ∈ {CZ} do
16:             CZ(a,b)
17:             {CZ} ← {CZ} \ {(a, b)}
```

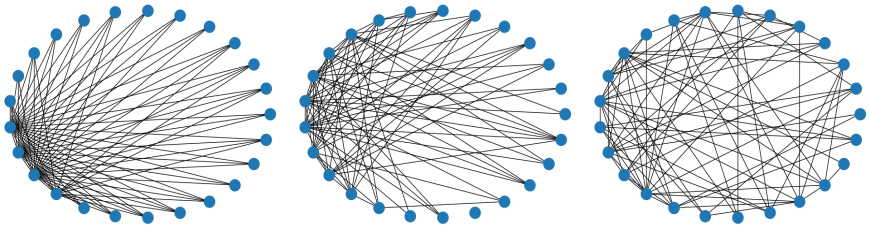
Here we define $\mathrm{d}_{G'}(a)$ as the degree of node a in a graph that has V as nodes and $\{CZ\}$ as edges. $\mathrm{d}_{G'}(a)$ is thus equal to the number of CZs in the sequence that are performed on node a.

The run time of algorithm 5 is dominated by the calls to 'remove_VOP*' and algorithm 4, both with time complexity $O(d^2)$. Compared to naively simulating all CZ gates in sequence $\{CZ\}$ with algorithm 4, it minimizes the number of calls to 'remove_VOP*'. Therefore, algorithm 5 is faster for concentrated sequences. The overhead necessary to schedule the CZ gates is relatively small compared to the simulation of the CZ gates.

4.3.1. EMPIRICAL VALIDATION

We will now present empirical results to validate our claim that CZ_SEQUENCE provides an additional speedup to algorithm 4 when simulating concentrated sequences of CZ. All results are obtained using a Python implementation of our algorithms, on cloud-based hardware that has 126 CPU cores available.

To validate our claim we time the scheduling and simulation of randomly generated CZ sequences with varying concentration on a random state.



(a) $\sigma = 0$ (b) $\sigma = 0.1$ (c) $\sigma = 0.2$

Figure 4.4: Randomly generated CZ sequences plotted as graphs with $n = 25$ nodes, $nlog(n)$ CZ gates, and varying sigma. Note that not all edges originate from a single node for $\sigma = 0$. Since a node cannot have more than $n-1$ edges, already fully connected nodes are removed from the normal distribution, thus allowing the edges to originate from multiple nodes.

For the generation of the random state, we use procedure 4.2.2. We generate CZ sequences with varying concentration with the following procedure:

Definition 4.3.1. Procedure for generating a random CZ sequence $\{CZ\}$ with varying concentration, on a quantum state with n qubits. For every $CZ_{a,b}$ in $\{CZ\}$, we draw the starting node a from a normal distribution with $\mu = n/2$ and varying standard deviation σ, truncated between $[0, n)$. The end node b is drawn uniformly from all nodes in $[0, n)$. This allow us to change the concentration of the CZ gates in the sequence by changing the standard deviation σ of the truncated normal distribution.

To clarify our procedure for generating random CZ sequences, we visualize a number of generated random sequences in figure 4.4.

We simulate these sequences on randomly generated states and plot their simulation time versus sigma, for different numbers of qubits n and sequence lengths $|\{CZ\}|$.
Figure 4.5 shows that there is indeed a decrease in run time for sequences with low sigma, making simulating those sequences 2 to 3 times faster compared to less concentrated sequences. Algorithm 5 does not provide a significant speedup for states with a sigma higher than $0.2n$. The speedup is larger, both relatively and absolutely, for larger sequence lengths. The speed advantage is larger for states with more qubits.

We note that for random graphs, algorithm 4 will also show similar increase performance on concentrated sequences without scheduling the CZ gates using algorithm 5. Algorithm 5 aims to call remove_VOP* on the node on which the most CZ in a sequence work. Without the scheduling algorithm, remove_VOP* is called on the qubit with the smallest number of neighbours (see lines 23-29 of algorithm 4). Because for a random graph most nodes will have about the same amount of neighbours, algorithm 4 already has a high probability of calling remove_VOP* on the node on which the most CZ in a sequence work. However, for non-random sequences it will not, because then such a node might have more neighbours than the other nodes the CZ in the sequence work on. In those cases, scheduling algorithm 5 is necessary to optimize the number of local complementations.

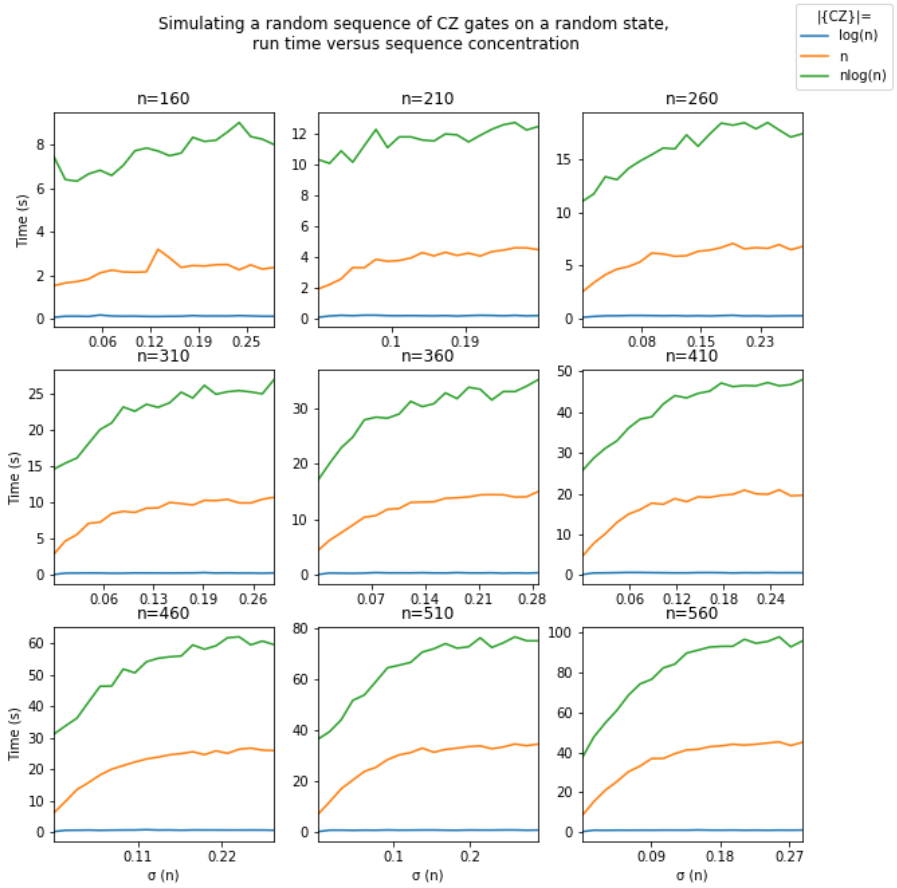


Figure 4.5: Run time of simulating a random sequence of CZ gates on a random state using algorithm 5 versus sequence concentration. Here, σ is the standard deviation of the normal distribution as defined in definition 4.4, expressed relative to the number of qubits n. As σ increases, the concentration of the CZ sequence decreases. We run our experiment for various n and sequence size $|\{CZ\}|$. Every data-point consists of the average of 100 runs. The random states are generated using the procedure outlined in 4.2.2, with $\beta = 1$.

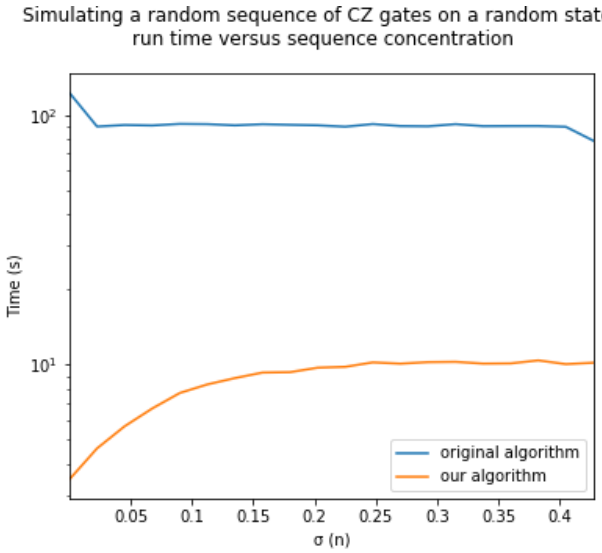


Figure 4.6: Log of the run time of simulating a random sequence of CZ gates on a random state versus sigma (see definition 4.3.1), as expressed relative to the number of qubits n. Both the measured run times for our algorithm and that of the original algorithm from [21] are depicted. Here, $n = 300$ and the sequence simulated contains 300 CZ gates. Every data-point consists of the average of 100 runs.

In figure 4.6 we plot the run time of our novel CZ sequence algorithm versus that of the original algorithm of [21] when simulating sequences of CZ on a log-scale. For lower sigma, our CZ scheduling algorithm performs more than 50 times faster in our runs.

5

OPERATIONS ON GSLC

In this chapter, we introduce two novel algorithms to perform operations that are useful when simulating stabilizer circuits in GLSC form: calculating fidelity and tracing out qubits. Fidelity is a measure of "closeness" of two quantum states. It is used, for instance, to determine how close the obtained state from a (noisy) simulated circuit is to some ideal state, as well as in algorithms for general quantum circuit simulation such as [47]. Tracing out part of a multi qubit quantum state allow us to determine the "partial state" associated to a subset of qubits. Intuitively, this operation extracts the information contained in a smaller part of a state from the whole state. For example, tracing out can be used if one simulates a large quantum network but is only interested in the state of the qubits of two particular end-users.

Algorithms for calculating fidelity and tracing out qubits were previously introduced for the tableau formalism by [30] and [17], respectively. In the tableau formalism, these operations include a form of Gaussian elimination to get the tableau in some normal form, before the actual operation can be performed. For GSLC this is not required, as the circuit of the state is already in a normal form. The time complexity of the operations we define here can be seen in table 5.1.

	Tableau	GSLC
Fidelity	$O(n^3)$	$O(\|E\| \cdot n^2)$
Tracing out k qubits	$O(n^2)$	$O(k)$

Table 5.1: Comparison of the time complexity of various operations in tableau and GSLC formalism. For the tableau formalism, the fidelity algorithm referenced is that described by [30]. The tracing out algorithm is that described by [17].

This chapter begins with a section on how to trace out qubits of a GSLC and how to represent the resulting mixed state. We then present an algorithm to calculate the fidelity of two GSLC, followed by a proof that this algorithm can be used on two states of different size.

5.1. TRACING OUT GSLC

In this section, we present a novel way to trace out GSLC and represent the resulting reduced density matrix. We start by deriving a lemma that describes the reduced density matrices of graph states, followed by an extension of this lemma to one that describes reduced density matrices of GSLC. We finish this section by giving an example of a traced out GSLC, how to represent it, and sketch a way to obtain such a representation.

Let the basis K be the computational basis over B. Then we can rewrite the partial trace (definition 2.10) as:

$$Tr_B(\rho_{AB}) = \sum_{k=0}^{2^{|B|}} \langle k| \rho_{AB} |k\rangle \tag{5.1}$$

$$= \sum_{k=0}^{2^{|B|-1}} \langle 0k| \rho_{AB} |0k\rangle + \langle 1k| \rho_{AB} |1k\rangle, \tag{5.2}$$

where $|0k\rangle$ ($|1k\rangle$) represents a tensor product of the computational basis state $|0\rangle$ ($|1\rangle$) with some number k in binary representation. Note that this holds for any qubit; e.g. including tensor products where the computational basis state is placed between digits of k.

We can rewrite the definition of a graph state in ket form (equation 2.35) to a normalized sum of all kets from 0 to $2^{|V|}-1$, where all kets that have an *uneven* number of CZs on two qubits that are both 1 are multiplied by a factor of -1:

$$\prod_{i,j\in E} CZ_{i,j} |+\rangle^{\otimes|V|} = (\frac{1}{\sqrt{2}})^{|V|} \prod_{i,j\in E} CZ_{i,j} (|0\rangle + |1\rangle)^{\otimes|V|} \tag{5.3}$$

$$= (\frac{1}{\sqrt{2}})^{|V|} \sum_{k=0}^{2^{|V|}-1} \prod_{(i,j)\in E} CZ_{i,j} |k\rangle, \tag{5.4}$$

Using this, we will now give a description of a density matrix of a graph state. In order to facilitate our reasoning about the partial trace of such a density matrix, we will first rewrite equation 5.4. Rewriting equation 5.4 as a summation over the $2^{|V|}$ subsets K in the powerset of V, we obtain:

$$(\frac{1}{\sqrt{2}})^{|V|} \sum_{k=0}^{2^{|V|}-1} \prod_{(i,j)\in E} CZ_{i,j} |k\rangle \tag{5.5}$$

$$=(\frac{1}{\sqrt{2}})^{|V|} \sum_{K\subseteq V} \prod_{(i,j)\in E} CZ_{i,j} |K\rangle, \tag{5.6}$$

where $|K\rangle$ is a ket with 1 at the position of the qubits in K, and 0 elsewhere. The CZ gate performs a Z gate when its control qubit is $|1\rangle$. In a graph state, this means that a Z gate is performed on a qubit j if it has a neighbour i that is $|1\rangle$. From this, we obtain the following corollary:

Corollary 5.1.0.1 (Density matrix of a graph state)**.** *The density matrix of a graph state* $|G\rangle$ *is defined as:*

$$|G\rangle\langle G| = \prod_{(i,j)\in E}\prod_{(k,l)\in E} CZ_{i,j}\,|+\rangle^{\otimes|V|}\langle+|^{\otimes|V|}\,CZ_{k,l} \tag{5.7}$$

Given this corollary, and the definition of the partial trace (definition 2.10), we get the following lemma:

Lemma 5.1.1. *[22] After tracing out a set of* $B \subset V$ *qubits from a graph state* $|G\rangle$*, the resulting density matrix is equal to:*

$$Tr_B(|G\rangle\langle G|) = \frac{1}{2^{|B|}}\sum_{B'\subseteq B}\prod_{\substack{(i,j)\in E\\ \forall(i\in B'\wedge j\in A)}}\prod_{\substack{(p,q)\in E\\ \forall(p\in B'\wedge q\in A)}} Z_j\,|A\rangle\langle A|\,Z_q, \tag{5.8}$$

where $|A\rangle$ *is a graph state equal to* $\big|V\setminus B, \{(i,j)|\textit{for all } i\wedge j\in(V\setminus B)\}\big\rangle$.

Proof. We will prove lemma 5.1.1 by induction on $|B|$.
Base case: let $|B|=0$. Then the partial trace over b is equal to:

$$Tr_\emptyset(|G\rangle\langle G|) = \frac{1}{2^0}\sum_{B'\subseteq\emptyset}\prod_{\substack{(i,j)\in E\\ \forall(i\in B'\wedge j\in A)}}\prod_{\substack{(p,q)\in E\\ \forall(p\in B'\wedge q\in A)}} Z_j\,|A\rangle\langle A|\,Z_q \tag{5.9}$$

$$= |G\rangle\langle G|. \tag{5.10}$$

Inductive step: let lemma 5.1.1 be true for a given $0\le|B|<n$, with n being the number of qubits in G. Then for $|B|+1$, the partial trace of G over $B+\psi$ with ψ being some qubit in $V\setminus B$ is equal to:

$$Tr_{B+\psi}(|G\rangle\langle G|) = \sum_{k=0}^{2^{|B|+1}-1}\langle 0k|G\rangle\langle G|0k\rangle + \langle 1k|G\rangle\langle G|1k\rangle \tag{5.11}$$

$$= \sum_{k=0}^{2^{|B|}}\langle 0k|G\rangle\langle G|0k\rangle + \langle 1k|G\rangle\langle G|1k\rangle \tag{5.12}$$

By equation 2.35, $\langle 0k|G\rangle$ and $\langle 1k|G\rangle$ are equal to:

$$\prod_{(i,j)\in E}\langle 0k|\,CZ_{i,j}\,|+\rangle^{\otimes|V|} = \frac{1}{\sqrt{2}}\prod_{\substack{(i,j)\in E\\ \forall i\wedge j\neq\psi}}\langle k|\,CZ_{i,j}\,|+\rangle^{\otimes|V|-1} \tag{5.13}$$

$$= \frac{1}{\sqrt{2}}\langle k|G'\rangle \tag{5.14}$$

$$\prod_{(i,j)\in E}\langle 1k|\,CZ_{i,j}\,|+\rangle^{\otimes|V|} = \frac{1}{\sqrt{2}}\prod_{(\psi,p)\in E}\prod_{\substack{(i,j)\in E\\ \forall i\wedge j\neq\psi}}\langle k|\,Z_pCZ_{i,j}\,|+\rangle^{\otimes|V|-1} \tag{5.15}$$

$$= \frac{1}{\sqrt{2}}\prod_{(\psi,p)\in E}\langle k|\,Z_p\,|G'\rangle, \tag{5.16}$$

where $G' = (V \setminus \psi, \{(i,j) \in E | \forall i \wedge j \neq \psi\})$. If we plug this into equation 5.12, we obtain:

$$\sum_{k=0}^{2^{|B|}} \langle 0k|G\rangle \langle G|0k\rangle + \langle 1k|G\rangle \langle G|1k\rangle \tag{5.17}$$

$$= \sum_{k=0}^{2^{|B|}} \frac{1}{2} \langle k|G'\rangle \langle G'|k\rangle + \frac{1}{2} \prod_{(\psi,p)\in E} \prod_{(\psi,q)\in E} \langle k| Z_p |G'\rangle \langle G'| Z_q |k\rangle \tag{5.18}$$

$$= Tr_B(\frac{1}{2}\{|G'\rangle \langle G'| + \prod_{(\psi,p)\in E} \prod_{(\psi,q)\in E} Z_p |G'\rangle \langle G'| Z_q\}) \tag{5.19}$$

$$= Tr_B(\frac{1}{2^{|1|}} \sum_{B'\subseteq\{\psi\}} \prod_{\substack{(i,j)\in E \\ \forall(i\in B' \wedge j\in G')}} \prod_{\substack{(p,q)\in E \\ \forall(p\in B' \wedge q\in G')}} Z_j |G'\rangle \langle G'| Z_q) \tag{5.20}$$

$$= Tr_B(Tr_\psi(|G\rangle \langle G|)) \tag{5.21}$$

$$= Tr_\psi(Tr_B(|G\rangle \langle G|)), \tag{5.22}$$

that is, lemma 5.1.1 is also true for $|B|+1$. Since both the base case and the inductive step have been shown, lemma 5.1.1 holds. □

The reduced density matrix of a GSLC is an extension of 5.1.1. Similar to graph states, we start with the definition of GSLC in ket notation, as given in definition 2.37. From this, we extend to describing density matrices of GSLC as:

Corollary 5.1.1.1 (Density matrix of a GSLC)**.** *The density matrix of a GSLC* $\left|G;\overline{C}\right\rangle$ *is defined as:*

$$\left|G;\overline{C}\right\rangle \left\langle G;\overline{C}\right| = \bigotimes_{p\in V} \bigotimes_{q\in V} \prod_{(i,j)\in E} \prod_{(k,l)\in E} C_p C Z_{i,j} |+\rangle^{\otimes|V|} \langle +|^{\otimes|V|} C Z_{k,l} C_q^\dagger \tag{5.23}$$

$$= \bigotimes_{p\in V} \bigotimes_{q\in V} C_p |G\rangle \langle G| C_q^\dagger. \tag{5.24}$$

This definition allows us to extend 2.10 to GSLC in the following lemma.

Lemma 5.1.2. *After tracing out a set of* $B \subset V$ *qubits from a GSLC* $\left|G;\overline{C}\right\rangle$*, the resulting density matrix is equal to:*

$$Tr_B(\left|G;\overline{C}\right\rangle \left\langle G;\overline{C}\right|) = \bigotimes_{p\in(V\setminus B)} \bigotimes_{q\in(V\setminus B)} C_p Tr_B(|G\rangle \langle G|) C_q^\dagger. \tag{5.25}$$

Proof. We can write the density matrix of a GSLC as:

$$\bigotimes_{p,q\in(V\setminus B)} \bigotimes_{l,m\in B} (C_p \otimes C_l) |G\rangle \langle G| (C_q \otimes C_m)^\dagger. \tag{5.26}$$

Using the fact that the trace is cyclic, the partial trace over B then becomes:

$$Tr_B\{\bigotimes_{p,q\in(V\setminus B)}\bigotimes_{l,m\in B}(C_p\otimes C_l)\,|G\rangle\langle G|\,(C_q\otimes C_m)^\dagger\} \tag{5.27}$$

$$=Tr_B\{\bigotimes_{p,q\in(V\setminus B)}\bigotimes_{l,m\in B}(C_p\otimes I)\,|G\rangle\langle G|\,(C_q\otimes I)^\dagger C_m^\dagger C_l\} \tag{5.28}$$

$$=Tr_B\{\bigotimes_{p,q\in(V\setminus B)}C_p\,|G\rangle\langle G|\,C_p^\dagger\} \tag{5.29}$$

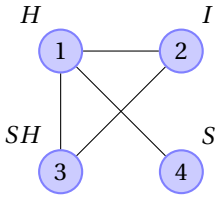
$$=\bigotimes_{p,q\in(V\setminus B)}C_p\,Tr_B(|G\rangle\langle G|)C_q^\dagger. \tag{5.30}$$

Thus proving lemma 5.1.2.

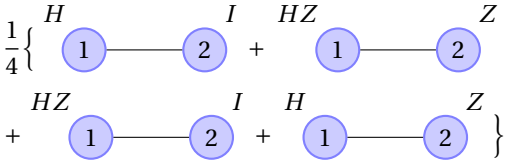
□

In other words, tracing out a qubit of a GSLC produces a mixed state that is an equal mixture of the remaining subgraph, where the local Cliffords of the neighbours of all possible subsets of the set of traced out qubits are right-multiplied with Z.

As an example, take the following GSLC:



Tracing out qubits 3 and 4 gives us a state that is equal to the following mixture:



This mixed state can be fully represented by storing the original graph and flagging what qubits were traced out. For our example, that would be:

Vertex	local Clifford	adjacency	traced out
1	H	2,3,4	false
2	I	1,3	false
3	SH	1,2	true
4	S	1	true

Such a mixed state can be obtained in time $O(k)$, where k is the number of traced out qubits, as we only need to flag all k traced out qubits.

5.2. CALCULATING FIDELITY

In this section, we give an algorithm for calculating fidelity between two stabilizer states that can be seen as a generalization of the fidelity algorithm for tableaus defined by [30]. We start by explaining the reasoning behind the algorithm, that applies the circuit of one of the states to the other. We then give pseudo-code and analyze its complexity when using the GSLC formalism to represent the stabilizer states. This is followed by heuristic to select which state to apply the circuit of the other state on. We finalize this section by giving a proof that our algorithm can be used to calculate fidelity between states of different size in the GSLC formalism.

The fidelity between pure states (and therefore for stabilizer states), it is defined as $|\langle\phi|\psi\rangle|^2$, for two states $|\phi\rangle$ and $|\psi\rangle$.

For any circuit U consisting of unitaries, we have that:

$$\langle\phi|\psi\rangle = \langle\phi| U^\dagger U |\psi\rangle. \tag{5.31}$$

5

Let us pick U such that $U|\phi\rangle = |0\rangle^{\otimes n}$, then:

$$\langle\phi|\psi\rangle = \langle 0|^{\otimes n} U |\psi\rangle. \tag{5.32}$$

Therefore, we can calculate the fidelity between two stabilizer states $|\langle\phi|\psi\rangle|^2$ by calculating $|\langle 0|^{\otimes n} U|\psi\rangle|^2$. This we can do by measuring every qubit of $U|\psi\rangle$ in the Z-direction. We can terminate this algorithm early: if we measure one qubit of $U|\psi\rangle$ in the Z basis and its outcome is deterministically equal to -1, we already know the fidelity between $U|\psi\rangle$ and $|0\rangle^{\otimes n}$ is 0 and we can return 0.

For states in GSLC form, determining the circuit that brings back a given state to the all zero state is easy:

$$|\phi\rangle = \bigotimes_{i=0}^{n} \prod_{(a,b)\in E} C_i CZ_{a,b} H_i |0\rangle^{\otimes n} \tag{5.33}$$

$$\Rightarrow \bigotimes_{i=0}^{n} \prod_{(a,b)\in E} H_i CZ_{a,b} C_i^\dagger |\phi\rangle = |0\rangle^{\otimes n} \tag{5.34}$$

$$\Rightarrow U = \bigotimes_{i=0}^{n} \prod_{(a,b)\in E} H_i CZ_{a,b} C_i^\dagger. \tag{5.35}$$

That is, U is equal to a circuit that consists of a tensor product of the Hermitian conjugate of the local Cliffords of $|\phi\rangle$, followed by a CZ gate for each of its edges, followed by a Hadamard gate on each qubit. In pseudo-code, the general fidelity algorithm for stabilizer states looks as follow:

Algorithm 6 Fidelity between stabilizer states $|\phi\rangle$, $|\psi\rangle$

```
1:  procedure FIDELITY(|φ⟩, |ψ⟩)
2:      overlap ← 1
3:      C ← Determine U such that U|φ⟩ = |0⟩^⊗n
4:      |ψ'⟩ ← U|ψ⟩
5:      for all k ∈ {1, ..., n} do
6:          m_k, deterministic ← MEASURE_+1(|ψ'⟩, k)
7:          if deterministic and m_k = −1 then
8:              return 0
9:          else if not deterministic then
10:             overlap ← overlap ∗ 1/√2
        return overlap^2
```

Here, the 'MEASURE_+1($|\phi\rangle, k$)' procedure measures the $k-th$ qubit of $|\phi\rangle$ and updates the state *as if the measurement outcome +1 was measured.* It returns the measurement result, and whether the result was deterministic.

The complexity of algorithm 6 is dependent on the run time of three parts: the determination of U, the calculation of $U|\phi\rangle$, and the sequence of measurements. For the tableau algorithm, [30] [41] have shown that this can be done in time complexity $O(n^3)$. For two GSLC of n qubits of maximum degree d, the algorithm's complexity per step is as follows:

- When determining U, we iterate through the qubits and add a local Clifford and a Hadamard per qubit, as well as a CZ gate per edge in the graph. Its complexity is therefore $\Theta(1) \cdot 2n + \Theta(|E|_\phi)$.
- Calculating $U|\psi\rangle$ is dominated by the $O(|E|_\phi)$ number of CZ gates, that are in turn dominated by local complementations of order $O(d_\psi^2)$. However, simulating CZ gates can increase the degree of that graph. Therefore d_ψ can increase, up to the maximum possible degree n. The resulting upper bound on this step of the algorithm is therefore $O(|E|_\phi n^2)$. We note that this bound is loose; in use practical use cases where for instance $|\psi\rangle$ and $|\phi\rangle$ are states with a high fidelity, or $|\psi\rangle$ has few edges, d_ψ will not increase a lot. Because U will by definition contain a sequence of CZ gates, we use algorithm 5 to optimize the run time of calculating $U|\psi\rangle$.
- In the rest of the algorithm, we measure at most n qubits, which in GSLC form takes at most $O(d^2)$ time. Here, a measurement might also increase d. This makes the overall complexity of this step $O(n^3)$. As with the previous step, this upper bound can be loose in practise.

This leads to an overall complexity of the fidelity algorithm of $O(|E| \cdot n^2)$.

As noted above, this bound can be loose dependent on the states used to calculate fidelity on. The practical cost of algorithm 6 is driven by the amount of edges of both states, as well as the growth of the degree of the state the circuit is applied on. By using our novel CZ algorithm 4 and the CZ sequence algorithm 5, we can minimize the number of local complementations used, and therefore the degree growth.

Additionally, we can pick what state to apply circuit U on in a such a way that we minimize the run time of algorithm 6. For instance, if we want to calculate the fidelity between a product state $|\phi\rangle$ and a GSLC with a high degree $|\psi\rangle$, it would take time order $\Theta(n)$ to calculate $U|\psi\rangle$, but $O(|E|_\psi n^2)$ to calculate $U|\phi\rangle$ (where U is the circuit of the other state). In order to determine what state to apply U on, we use a heuristic algorithm to calculate the costs of applying U, that we will present in the following subsection.

A HEURISTIC ALGORITHM TO ESTIMATE THE COST OF APPLYING CIRCUIT U

We estimate the cost of applying the circuit U of a GSLC $|\phi\rangle$ on a GSLC $|\psi\rangle$ using our CZ sequence algorithm as follows:

1. Make a graph $G' = (V', E')$, where V' are all non-z type vertices of $|\psi\rangle$, and $E' = \{(a,b) \in E_\phi | \text{s.t. } a,b \in V'\}$. That is, E' contains all edges of $|\phi\rangle$ between two qubits that are non-z type qubits in $|\psi\rangle$.

2. Calculate size k of the approximate vertex cover using the algorithm by Gavril and Yannakakis [11]. The minimal vertex cover of a graph is the minimal set of vertices that includes at least one endpoint of every edge of the graph. The approximate vertex cover approximates this up to a factor of 2.

The total estimated cost of applying circuit U of $|\phi\rangle$ on $|\psi\rangle$ is then:

$$\frac{k^3}{2} \cdot d_\psi^2 + |E|_\phi \cdot (k+1) d_\psi, \tag{5.36}$$

as we approximately need to do k local complementations that cost approximately $\sum_{i=0}^{k} (i d_\psi + d_\psi)^2 = \frac{k^3}{2} \cdot d_\psi^2$, as every local complementation increases the degree of $|\psi\rangle$ by at most the initial degree $d_\psi - 1$. Afterwards, we still need to do $|E|_\phi$ CZ gates using algorithm 3, that has complexity order $O(d)$. Because the k local complementations increase the degree to at most $(k+1)(d_\psi - 1)$, this step takes approximately $|E|_\phi \cdot (k+1) d_\psi$.

We use this heuristic algorithm to estimate the cost of applying circuit U on both states, and simulate the option with the one with the lowest costs.

CALCULATING FIDELITY ON A REDUCED DENSITY MATRIX

We can also use algorithm 6 to calculate the fidelity between two states of different size, or between part of large state and a smaller state. This can be useful in applications were one is only interested in part of a larger state, for instance when one simulates a repeater network and wants to know the fidelity between an ideal Bell pair and two endpoints in the network. In this subsection, we will prove that algorithm 6 can be used for this purpose. We will first define the fidelity between two states of different size, before proving our claim.

Calculating the fidelity between a register A of a larger GSLC ρ_{AB} and a smaller state $|\phi\rangle$ means calculating the fidelity between the reduced density matrix of a GSLC $Tr_B(\rho_{AB})$ and a smaller state $|\phi\rangle$. This is equal to:

$$\langle\phi| Tr_B(\rho_{AB}) |\phi\rangle \tag{5.37}$$

$$= \langle\phi| \bigotimes_{p\in(V\setminus B)} C_p Tr_B(|G\rangle\langle G|) C_p^\dagger |\phi\rangle \tag{5.38}$$

$$= \langle\phi| \bigotimes_{p\in(V\setminus B)} C_p \frac{1}{2^{|B|}} \sum_{B'\subseteq B} \prod_{\substack{(i,j)\in E \\ \forall(i\in B'\wedge j\in A)}} \prod_{\substack{(p,q)\in E \\ \forall(p\in B'\wedge q\in A)}} Z_j |A\rangle\langle A| Z_q C_p^\dagger |\phi\rangle \tag{5.39}$$

$$= \frac{1}{2^{|B|}} \sum_{B'\subseteq B} \langle\phi| \bigotimes_{p\in(V\setminus B)} C_p \prod_{\substack{(i,j)\in E \\ \forall(i\in B'\wedge j\in A)}} \prod_{\substack{(p,q)\in E \\ \forall(p\in B'\wedge q\in A)}} Z_j |A\rangle\langle A| Z_q C_p^\dagger |\phi\rangle \tag{5.40}$$

$$= \frac{1}{2^{|B|}} \sum_{B'\subseteq B} (\langle\phi| \bigotimes_{p\in(V\setminus B)} C_p \prod_{\substack{(i,j)\in E \\ \forall(i\in B'\wedge j\in A)}} Z_j |A\rangle)^2. \tag{5.41}$$

In other words, it is equal to the squared average inner product between $|\phi\rangle$ and all the terms that make up the mixed GSLC.

Measuring a qubit a in GSLC form in the computational basis means measuring the observable $P_a = C_a^\dagger Z C_a$ on the underlying graph state $|G\rangle$ [21] (where C_a is the local Clifford of qubit a). When after tracing out a qubit the local Clifford of a qubit is right-multiplied with Z, we measure $P'_a = Z_a C_a^\dagger Z C_a Z_a = Z_a P_a Z_a = \pm P_a$ on the underlying graph.

Since the mixed state is a summation over the power set of the traced out qubits, a qubit's local Clifford is right-multiplied with Z in half the states that make up the mixed state if it has a traced-out neighbour. This means that after tracing out, a measurement of qubit a in the measurement sequence of algorithm 6 falls into one of two cases:

- If the measurement result of P_a is non-deterministic, the measurement result P'_a is non-deterministic, because $P'_a = \pm P_a$. I.e. the measurement direction stays the same.
- If the measurement result is deterministic, the qubit has no neighbours and can therefore not have any traced-out neighbours. Hence $P'_a = P_a$

Therefore, the result of all the measurements in the measurement sequence of algorithm 6 are the same after tracing out, and we can use it to calculate fidelity between a large and a smaller state in GSLC formalism.

6

GENERAL QUANTUM CIRCUIT SIMULATION

General quantum circuits containing few non Clifford gates can be simulated on classical computers using stabilizer states [34] [47]. In a recent result, Bravyi et al.[47] present a simulation algorithm (the sum-over-Cliffords method) that can simulate a larger set of non-Clifford gates and has significantly improved performance over previous state-of-the-art. This allows for the simulation of quantum algorithms with 50 qubits and 60 non-Clifford gates on a laptop computer. It is currently implemented in Qiskit Aer, a widely used quantum simulation platform by IBM, where it is defaulted to when simulation using full state vectors becomes unfeasible [60].

The sum-over-Clifford method as proposed by [47] has a subroutine, called 'Phase-sensitive Clifford simulator', based on tableaus (see section 2.3.1) in a certain CH-canonical form to simulate a large set of stabilizer circuits.
In this chapter, we further improve the performance of the results by Bravyi et al. with a subroutine based on the results presented in the previous chapters. The subroutine is an extension of the GSLC formalism to make it phase-sensitive. Depending on the degree of the simulated stabilizer states, stabilizer circuits can be simulated faster using this subroutine compared to CH-form tableaus. Additionally, our new single CZ algorithm and CZ scheduling algorithm (see chapter 4) can then be used to further reduce the simulation time. We show that the class of circuits that can be simulated faster using our subroutine contains contains practical use cases. For this purpose, we evaluate the simulation of a quantum algorithm: the quantum approximate optimization algorithm (QAOA) applied to the Max E3LIN2 problem[32]. This problem was also used by Bravyi et al. to benchmark the performance of the sum-over-Clifford method in [47].

This chapter starts with a section that gives an overview of general quantum circuit simulation using stabilizer states. This is followed by a description of the sum-over-Clifford Metropolis simulator by Bravyi et al. as presented in [47]. Afterwards, we present an extension of the GSLC formalism to make it phase sensitive. This chapter concludes with a section where we give an overview of the workings of the QAOA applied to the Max

E3LIN2 problem and why our subroutine achieves a speedup over the original subroutine.

6.1. GENERAL QUANTUM CIRCUIT SIMULATION

We start this section by exploring some earlier results regarding the computational complexity of simulating general quantum circuits. We then elaborate on earlier results that show how the stabilizer formalism can be used to efficiently simulate quantum circuits containing a few non-Clifford gates. This section ends with a number of formal definitions used later in this chapter: stabilizer rank, approximate stabilizer rank and stabilizer extent.

Throughout this section, we will classify computational problems into complexity classes. A complexity class is a set of problems of related complexity. These are usually defined as the type of problems contained in that class, what computational method can be used, and how the resources used are bounded. As an example, we formally define two important complexity classes [59]:

Definition 6.1.1. The complexity class **P** is defined as the class of decision problems solvable in polynomial time by a Turing machine.

Definition 6.1.2. The complexity class **NP** is defined as the class of decision problems solvable by an nondeterministic polynomial-time Turing machine such that:

1. If the answer is "yes," at least one computation path accepts.

2. If the answer is "no," all computation paths reject.

Put in simplified terms, **P** contains the problems for which an efficient algorithm exists, whereas **NP** contains hard problems. **P** is contained in **NP**, but it has not been proven whether these sets are equal or not (though it is commonly assumed that **P**≠**NP**). In fact, whether **P**=**NP** remains an important open problem in theoretical computer science[6].

It is widely believed that universal quantum computers cannot be efficiently classically simulated. This belief can intuitively be understood by considering the space required to represent a general quantum state: without using some form of compression, storing the state vector of a n qubit state would take exponential space of order $\Theta(2^n)$. Because of this exponential scaling, state-of-the-art simulators are limited to around 50 qubits [42].

Before we continue, we will make a distinction between two valid classical simulation techniques[24]:

1. **Strong simulation:** computing the probabilities of the output measurement with high accuracy using a classical computer.

2. **Weak simulation:** sampling from the output measurement distribution efficiently using a classical computer.

In [24], Van den Nest showed that strong simulation of certain quantum circuits is a **#P**-complete problem, which is a class of problems at least as hard as **NP** [2]. However, efficient weak simulation of these circuits is possible on a classical computer using a sampling technique. Van den Nest notes that "these examples highlight that any serious attempt to understand the relationship between classical and quantum computation should not rely on the notion of strong simulation". The sum-over-Cliffords method discussed in this chapter is a weak simulation technique, that samples from a large set of stabilizer states.

As stated by the Gottesman-Knill theorem[9] (see theorem 2.3.1), stabilizer circuits can be simulated efficiently using a classical computer. Bravyi and Kitaev showed in [20] that stabilizer circuits can be upgraded to universal quantum computation by adding any non-Clifford gate to the available gate set. This gives the possibility to simulate quantum circuits with few non-Clifford gates, for which a method was first proposed by Aaronson and Gottesman in [17]. This method scales linearly in the number of Clifford gates and exponentially in the number of non-Clifford gates. The practical applications of this work were limited because of the large exponent[47].

Recent work has decreased this exponent. Garcia, Markov and Cross [30] [34] [41] proposed simulation by 'stabilizer frame': a decomposition of a quantum state in a superposition of stabilizer states. In [47] Bravyi et al. present methods that improve on these results in both performance and applicability, and develop a mathematical theory of stabilizer rank (a concept we will introduce later). In this chapter, we aim to further improve their 'sum-over-Cliffords' method, that has state-of-the-art performance and the new feature of being capable to simulate circuits consisting of Clifford gates and arbitrary diagonal gates.

The decomposition of quantum states in a superposition of stabilizer states was formalized in a number of definitions and theorems. We will reproduce a number of these here to facilitate the explanation of the sum-over-Clifford algorithm. For more background on and proof of these definitions and theorems, we refer to their original publications. The minimal number of stabilizer states needed to represent a quantum state is called the stabilizer rank:

Definition 6.1.3. The **exact stabilizer rank** $\chi(\psi)$ [39] of n qubit pure state $|\psi\rangle$ is the smallest integer k such that $|\psi\rangle$ can be written as:

$$|\psi\rangle = \sum_{\alpha=1}^{k} c_\alpha |\phi_\alpha\rangle, \tag{6.1}$$

for some n qubit stabilizer states $|\phi_\alpha\rangle$ and some complex coefficients c_α.

For all $|\psi\rangle$, $\chi(\psi) \geq 1$. $\chi(\psi) = 1$ iff $|\psi\rangle$ is a stabilizer state. The exact stabilizer rank can be seen as a measure of computational non-classicality [47].

Closely related to this is the approximate stabilizer rank, for which we give the definition as stated in [47]:

Definition 6.1.4. The **approximate stabilizer rank** $\chi_\delta(\psi)$[38] of n qubit pure state $|\psi\rangle$ with $||\,|\psi\rangle|| = 1$ is the smallest integer k such that $||\,|\psi\rangle - |\psi'\rangle|| \leq \delta$ for some error pa-

rameter $\delta > 0$ and some state $|\psi'\rangle$ with exact stabilizer rank k. I.e.:

$$|| |\psi\rangle - \sum_{\alpha=1}^{k} c_\alpha |\phi_\alpha\rangle || \leq \delta \tag{6.2}$$

In [47], Bravyi et al. derive an upper bound on the stabilizer rank:

Theorem 6.1.1. ***Upper bound on*** χ_δ ***[47].*** *Let $|\psi\rangle$ be a normalized n qubit state with a stabilizer decomposition $|\psi\rangle = \sum_{\alpha=1}^{k} c_\alpha |\phi_\alpha\rangle$ where $|\phi_\alpha\rangle$ are normalized stabilizer states and c_α are complex coefficients. Then*

$$\chi_\delta(\psi) \leq 1 + ||c||_1^2/\delta^2. \tag{6.3}$$

Here $||c||_1 \equiv \sum_{\alpha=1}^{k} |c_\alpha|$. Note that k does not have to be optimal; it can include all stabilizer states.

This theorem allows us to simulate a state $|\psi\rangle$ up to some error δ[47]. We decompose a state $|\psi'\rangle$ into a superposition of $\chi' \approx \delta^{-2}||c||_1^2$ stabilizer states (such that $|| |\psi\rangle - |\psi'\rangle || \leq \delta$). We obtain the χ' stabilizer states $|\phi_\alpha\rangle$ by sampling them from the (exact) stabilizer decomposition of $|\psi\rangle$ at random with probabilities proportional to $|c_\alpha|$. $|\psi'\rangle$ is then defined as the superposition of the sampled χ' states with *equal* weights (but possibly a different phase). The set of states in this superposition are stabilizer states, and can therefore be efficiently simulated on a classical computer, thus allowing us to simulate the circuit of $|\psi\rangle$ up to error δ on a classical computer (given that χ' is not prohibitively large). Note that χ grows exponentially with the number of non-Clifford gates. This should be expected, as there exists postselective quantum circuits with a polynomial number of non-Clifford gates that solve problems in **NP**[56]. This implies that non-exponential scaling of χ with the number of non-Clifford gates would mean **P=NP**[47].

We will now show how to obtain the stabilizer decomposition of a general quantum circuit by giving an example[49].

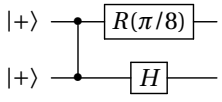
Consider the (non-Clifford) single-qubit rotation gate $R(\theta)$, that rotates a qubit by angle θ around the Z axis:

$$R(\theta) = e^{-i(\theta/2)Z} = \begin{bmatrix} e^{-\frac{i\theta}{2}} & 0 \\ 0 & e^{\frac{i\theta}{2}} \end{bmatrix}. \tag{6.4}$$

We can write this gate as a linear combination of two Cliffords:

$$R(\theta) = \left(\cos\left(\frac{\theta}{2}\right) - \sin\left(\frac{\theta}{2}\right)\right) \cdot I + e^{-i\pi/4}\sqrt{2}\sin\left(\frac{\theta}{2}\right) \cdot S. \tag{6.5}$$

Such a decomposition into a sum of Cliffords allows us to write a circuit containing non-Clifford gates as a sum over Clifford circuits. For example, we draw the following circuit containing a non-Clifford gate:



We can write this circuit as a product of Clifford and non-Clifford gates:

$$H_1 R_0(\pi/8) CZ_{0,1} |{+}{+}\rangle . \tag{6.6}$$

Using the decomposition of $R(\theta)$ into Clifford gates, we rewrite this expression into a sum over Clifford circuits:

$$= (\cos(\pi/16) - \sin(\pi/16))\, H_1 I CZ_{0,1} |{+}{+}\rangle \tag{6.7}$$

$$+ \left(e^{-i\pi/4}\sqrt{2}\sin(\pi/16)\right) H_1 S_0 CZ_{0,1} |{+}{+}\rangle . \tag{6.8}$$

In general, a circuit containing m rotation gates can be written as a sum over 2^m terms in this way. Having obtained such a decomposition, we are able to estimate how many stabilizer states k are needed to δ-approximate the circuit:

$$\chi' \delta^{-2} ||c||_1^2 = \delta^{-2} \prod_{j=1}^{m} \left(\cos\left(\frac{\theta_j}{2}\right) + \tan\left(\frac{\pi}{8}\right)\sin\left(\frac{\theta_j}{2}\right)\right)^2 . \tag{6.9}$$

Here, θ_j is the rotation angle of the j-th rotation gate. Note that for our earlier given example with $m = 1$, this estimation of χ' is loose: for small δ, it is much higher than the exact stabilizer rank $\chi = 2$. However, for larger m, $\chi' << \chi$. We sample these χ' stabilizer states $|\phi_\alpha\rangle$ (with their respective phase) from the exact decomposition with probabilities proportional to their weights $|c_\alpha|$. The resulting δ-approximate state is then equal to the sum of these states with equal weight.

The method described above can be used to decompose circuits containing different non-Clifford gates (such as the CCZ gate) by using their respective Clifford decompositions. For more background on how these decompositions are derived, we refer to [47] section 2.3.2. and [55].

6.2. SUM-OVER-CLIFFORD METROPOLIS SIMULATOR

This section gives an overview of the Sum-over-Clifford Metropolis simulator as introduced by Bravyi et al. in [47]. It starts by giving an outline of the three steps of the algorithm, and ends with an overview of the two subroutines used and their complexity. For more details and mathematical background on these subroutines, we refer to [47] section 4.1 and 4.2.

The Sum-over-Clifford Metropolis simulation of a quantum circuit on n qubits U containing few non-Clifford gates can be divided into three steps:

1. Obtain a δ-approximation of U by sampling χ' stabilizer states $|\phi_\alpha\rangle$ from its stabilizer decomposition using the method described in section 6.1.

2. Simulate the χ' stabilizer states using a phase-sensitive Clifford simulator based on tableaus in a certain CH-form.

3. Simulate measuring all the qubits in the Z direction by using a heuristic Metropolis Markov chain. That is, approximately sample $x \in \{0,1\}^n$ from the probability distribution $P(x) = \frac{|\langle x|\psi\rangle|^2}{||\psi||^2}$, where $\psi = U|0\rangle^{\otimes n}$.

The phase-sensitive Clifford simulator is based on previous work [30][44][24] that shows that any n-qubit stabilizer state $|\phi\rangle$ can be expressed as:

$$|\phi\rangle = \omega U_C U_H |s\rangle, \tag{6.10}$$

where ω is a complex number, U_C is a block of gates from the set $\{S, CZ, CNOT\}$, U_H is a block of H gates and $s \in \{0,1\}^n$ is a basis vector. The authors refer to this canonical form as the (possibly not unique) CH-form of a stabilizer state. The stabilizer state is represented using the tableau formalism, whose update rules are adjusted to account for the phase ω. Similarly to the GSLC formalism, the complexity of simulating stabilizer states in CH-form differs depending on the gate, as can be seen in table 6.1. The CH-form also allows for the calculation of the inner product between the CH-form state and a bit string $|x\rangle$ in $O(n^2)$ by a method similar to our fidelity algorithm. This is used in the Metropolis Markov chain step.

Gate	Complexity
S, CZ, CNOT	$O(n)$
H	$O(n^2)$

Table 6.1: Complexity of simulating gates in the CH-form tableau formalism.

The heuristic Metropolis Markov chain aims to sample $x \in \{0,1\}^n$ from the probability distribution $P(x) = \frac{|\langle x|\psi\rangle|^2}{||\psi||^2}$. Assuming the chain is in some current state x, its next state x' is generated in the next Metropolis step as follows:[47]

1. Pick an integer $j \in [n]$ uniformly at random and let $y = x \oplus e_j$.
2. If $P(y) \geq P(x)$, set $x' = y$.
3. Otherwise generate a random bit b such that $Pr(b = 1) = P(y)/P(x)$.
4. If $b = 1$, set $x' = y$. Otherwise $x' = x$.

In the above algorithm, the ratio

$$\frac{P(y)}{P(x)} \approx \left| \frac{\sum_{\alpha=i}^{\chi'} \langle y|\phi_\alpha\rangle}{\sum_{\alpha=i}^{\chi'} \langle x|\phi_\alpha\rangle} \right|^2 \tag{6.11}$$

is δ- approximated by summing the fidelities between the bit strings x, y and the χ' stabilizer states $|\phi_\alpha\rangle$ sampled from the decomposition of $|\psi\rangle$. The run time costs of calculating the initial probability $P(x_{in})$ for some random initial state x_{in} is $O(\chi' n^2)$, as the $O(n^2)$ CH-form fidelity algorithm is used χ' times to calculate the inner products $\langle x_{in}|\phi_\alpha\rangle$. By saving some data from the previous Metropolis step, subsequent fidelity calculations can be done in $O(n)$, resulting in a total run time complexity of T Metropolis steps of:

$$O(\chi' nT) + O(\chi' n^2). \tag{6.12}$$

This Metropolis Markov chain algorithm has not been proven to be irreducible and its mixing time is unknown. Hence, it should be considered a heuristic. However, the authors have shown it to be capable to sample from the output distribution of non-trivial quantum circuits accurately[47], and it is used in practise [60].

6.3. PHASE-SENSITIVE GSLC SUBROUTINE

In this section, we present a new subroutine for the sum-over-Clifford Metropolis simulator. The subroutine is an extension of the GSLC formalism to make it phase-sensitive. Depending on the circuit simulated, this can provide a significant speedup over the original CH-tableau form subroutine in the simulation step of the sum-over-Clifford simulator. For the Metropolis Markov chain step of the sum-over-Clifford algorithm, all states in the stabilizer deposition need to be represented in CH-tableau form. We present a simple procedure to convert the GSLC-based states to CH-tableau form. This section starts with an explanation of the new subroutine. It ends with a short description of the procedure to convert a GSLC state to a CH-tableau.

We first extend the GSLC formalism to account for global phases in this subsection, i.e., we make it phase-sensitive. This extension allows for the usage of the GSLC formalism with our novel CZ algorithm and CZ sequence scheduling algorithm presented in chapter 4 to simulate the stabilizer states of the stabilizer decomposition. Compared to the tableau based CH-form, this phase-sensitive GSLC subroutine can provide a significant speedup for certain circuits. We provide an example of such a circuit in the next section.

As stated in subsection 2.3.2, a GSLC is defined as:

$$\left|G;\overline{C}\right\rangle = \bigotimes_{p=1}^{|V|} C_p \prod_{(a,b)\in E} CZ_{a,b} \,|+\rangle^{\otimes|V|} \tag{6.13}$$

$$= \bigotimes_{p=1}^{|V|} C_p \,|G\rangle\,, \tag{6.14}$$

where V, E are the nodes and edges of graph G respectively, and C_p is the local Clifford of node p. For a block of CZ gates U_{CZ}, the following holds:

$$U_{CZ}\,|0\rangle^{\otimes n} = |0\rangle^{\otimes n}\,. \tag{6.15}$$

Because of this, the global phase of the underlying graph state $|G\rangle$ of a GSLC is fixed. In other words, only the local Cliffords of the nodes of a GSLC can change its global phase.

A local Clifford can be updated in two ways in the GSLC formalism:

1. By a right- or left- multiplication with another local Clifford, which is done by a Clifford multiplication lookup-table;

2. By a lookup to the 'cphase_table' used in the CZ algorithm (see section 2.37).

We update the lookup table for local Clifford multiplication and the 'cphase_table' to include the phases introduced to the local Cliffords by the Clifford multiplications and CZ gates. To make the GSLC formalism phase-sensitive, we store the phases of the local Cliffords in a vector of length n containing complex numbers, one for each local Clifford. We

initialize these phases to 1. When a local Clifford of qubit i is multiplied with a Clifford, we multiply the ith entry of the phase vector with the corresponding phase entry of the lookup table. When we simulate a CZ gate CZ_{ij} through a lookup to the 'cphase_table', we multiply the phase of i with the phase of the corresponding entry of the lookup table. The global phase of the GSLC is equal to the product of all phases in the phase vector.

To use our novel subroutine in the sum-over-Clifford simulator, we need to convert the states represented as GSLC to CH-tableau form. The following procedure can do this conversion of a n-qubit GSLC to a CH-tableau:

1. Initialize a CH-tableau T in the state $|+\rangle^{\otimes n}$.
2. For every edge $(a,b) \in E$, where E are the edges of the GSLC, simulate CZ_{ab} on T.
3. For every local Clifford C_i of the GSLC, simulate C_i on T.

Simulating a CZ, CNOT, or S gate in CH-tableau form takes time of order $O(n)$, and simulating an H gate takes time of order $O(n^2)$ (see section 6.1). Because there are at most n^2-1 edges and n local Cliffords in $|\phi\rangle$, this procedure has time complexity $O(n^3)$. This results in a overall run-time complexity of $O(\chi' nT) + O(\chi' n^3)$ for T Metropolis steps. As $T >> n$, this procedure gives only a small increase in the run time of the Metropolis Markov chain compared to the original algorithm (see equation 6.12) whilst allowing for the usage of our faster GSLC-based subroutine to simulate the states in the stabilizer decomposition.

6.4. EXAMPLE OF A USE CASE

Our novel phase-sensitive GSLC subroutine for the sum-over-Cliffords simulator by Bravyi et al.[47] can simulate certain circuits significantly faster compared to the CH-tableau subroutine. Examples of these circuits are those with a lot of single-qubit Clifford gates, those that result in GSLC with a low maximum degree, that have CZ gates that commute with the local Cliffords of the resulting GSLC, or circuits with concentrated sequences of CZ gates. In this section, we show that the class of circuits that can be simulated faster using our subroutine contains practical use cases. For this purpose, we evaluate the simulation of a quantum algorithm: the quantum approximate optimization algorithm (QAOA)[32] applied to the Max E3LIN2 problem[33]. This problem was also used by Bravyi et al. to benchmark the performance of the sum-over-Clifford method in [47]. First, we briefly discuss why the QAOA is interesting to simulate and what the E3LIN2 problem is. For more details on the E3LIN2 problem and QAOA, we refer to [33] and [32]. Following this, we describe the circuit of the QAOA applied on the E3LIN2 problem, and why our phase-sensitive GSLC subroutine achieves a speedup when simulating this circuit.

The quantum approximate optimization algorithm introduced by Farhi et al. in [32] is a hybrid quantum-classical algorithm that can be considered the emerging paradigm for solving optimization problems using near-term quantum technology [45]. As the name implies, it produces approximate solutions for classical combinatorial optimization problems. Whether near-term noisy intermediate-scale quantum (NISQ) computers running QAOA will provide quantum supremacy over classical algorithms remains

an open question[45]. As such, the simulation of this algorithm to research its behaviour is of interest, as it can guide the development of one of the most promising near-term applications of quantum computers.

The E3LIN2 problem is an example of a classical combinatorial optimization problem QAOA can be applied on. It aims to maximize binary linear equations (LIN2) containing 3 variables (E3). That is, it aims to maximize objective function

$$C(z) = \frac{1}{2} \sum_{1 \le u < v < w \le n} d_{uvw} z_u z_v z_w, \tag{6.16}$$

that depends on n binary variables $z_1, \cdots, z_n \in \{-1, 1\}$[47]. An instance of the E3LIN2 has a degree D is every variable z_u appears in exactly D terms $\pm z_u z_v z_w$.

In order to explain how the QAOA generates an approximate solution to this problem, we repeat the succinct explanation by Bravyi et al. in [47]. For more detail, we refer to [33].

The QAOA generates an approximate solution to the E3LIN2 problem by considering states that are obtained with the following circuit:

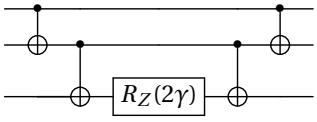
$$\left|\psi_{\beta,\gamma}\right\rangle = e^{-i\beta B} e^{-i\gamma \hat{C}} H^{\otimes n} |0\rangle^{\otimes n}, \tag{6.17}$$

where $\beta, \gamma \in \mathbb{R}$ are variational parameters, $B = X_1 + \cdots + X_n$ (the so-called 'transverse field operator'), and $\hat{C}$ is a diagonal operator obtained from C by replacing the variables z_u with Pauli operators Z_u. The QAOA algorithm chooses β and γ such that it maximizes the expected value of the objective function,

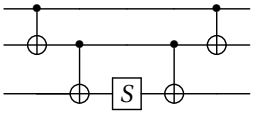
$$E(\beta, \gamma) = \left\langle \psi_{\beta,\gamma} \right| \hat{C} \left| \psi_{\beta,\gamma} \right\rangle. \tag{6.18}$$

The QAOA algorithm then samples a n-bit binary string z from probability distribution $P(z) = |\langle z | \psi_{\beta,\gamma} \rangle|^2$ by measuring every qubit of $\left|\psi_{\beta,\gamma}\right\rangle$ in the Z direction. By preparing and measuring enough samples, one can produce a string z such that $C(z) \ge E(\beta, \gamma)$.

To explain why our phase-sensitive GSLC subroutine can efficiently simulate the stabilizer decomposition of $\left|\psi_{\beta,\gamma}\right\rangle$, we consider its circuit and Clifford decomposition. We first consider $e^{-i\gamma \hat{C}}$. This operator consists of $\lfloor (n \cdot D)/3 \rfloor$ (where D is the degree of the instance of the E3LIN2 problem and n is the number of variables/qubits) applications of unitary $U = e^{-i\gamma Z_u Z_v Z_w}$. This unitary has the following circuit[50]:



Following equation 6.5, we can decompose this unitary in a linear combination of an identity operator and the following circuit:



Note that $CNOT_{a,b} = H_b CZ_{a,b} H_b$. Because every qubit is involved in D unitaries, and most CZ gates in the circuit of $e^{-i\gamma\hat{C}}$ either commute or are one **(Z type, non Z)** qubits, the degree d of the simulated GSLC remains low for low D. We empirically find that the maximum degree of the state is $2D$ using our novel CZ algorithm by simulating thousands of random instances of the E3LIN2 problem using our GSLC subroutine. For the $D = 4$ instances of the E3LIN2 problem used by Bravi et al., the maximum degree d of any GSLC in the decomposition is therefore $8 \approx \sqrt{n}$, where n is the total number of qubits.

$e^{-i\beta B}$ is equivalent to a tensor product of single-qubits X rotations. This tensor product can be decomposed using into single-qubit gates, which can be simulated in $\Theta(1)$ in GSLC formalism.

Hence, the worst-case gate in the GSLC formalism can be simulated in $O(n)$, and most gates are either $O(d) = O(\sqrt{n})$ one Z-type CZ gates or $\Theta(1)$ single-qubit Cliffords or commuting CZ gates. In the CH-tableau formalism, the fastest gate is $O(n)$ (whereas the worst-case Hadamard gate is $O(n^2)$). Thus, our GSLC-base subroutine provides a speedup of up to $O(n)$ over the CH-form tableaus when simulating these QAOA circuits. For a typical use case for the sum-over-Cliffords algorithm, $n \approx 50$.

7

CONCLUSIONS AND OUTLOOK

The simulation of quantum circuits is an essential tool in the development of quantum computers. It guides scientists in their research by allowing them to evaluate open questions such as the behavior of quantum approximate optimization algorithm under noise in superconducting qubits[46], the number of gates and qubits needed to perform quantum chemistry problems not attainable with classical computers [35] or the performance of a new quantum internet protocol [43].

In this thesis, we have improved the simulation of quantum circuits using classical computers by presenting new algorithms for the simulation of stabilizer states using GSLC that improve simulation run time and allow for operations to be performed. Moreover, we extended their applicability by defining a new subroutine for the simulation of general quantum circuits.

We first defined a canonical form for stabilizer tableaus in chapter 3 that allows us to study the simulation of GSLC by considering their tableau representation. In addition, we presented procedures to convert between GSLC and canonical tableau form, and methods to restore canonical form after performing a CZ gate. We later used these results on canonical form tableaus to develop procedures to faster simulate most CZ gates in the GSLC formalism. An open question remains if our canonical form can be used to improve other parts of the GSLC algorithm, such as the simulation of measurements.

In chapter 4, we presented two improvements to the simulation of CZ gates in the GSLC formalism. First, we gave a new method to simulate single CZ gates in the GSLC formalism based on the procedure to restore canonical form in a tableau. This method improves the run time of simulating single CZ gates on average up to 9 times compared to the original algorithm by [21] in our empirical validation. It additionally improves the run time of sequences of CZ gates by requiring less local complementations and increasing the number of Z-type qubits. We validated these improvements empirically by timing the simulation of random single CZ gates and CZ sequences on random states. Second, we introduced a scheduling procedure for sequences of CZ gates that builds on our single CZ algorithm. This scheduler provides an additional speedup on concentrated

sequences of CZ gates. Compared to the original algorithm, this can speedup simulation of concentrated sequences more than 50 times, which we showed by timing the simulation of random CZ sequences with varying concentration. This scheduling algorithm is, for instance, used when calculating the fidelity between two GSLC (as noted in section 5.2). All our improvements combined can significantly shorten the simulation time of stabilizer circuits in the GSLC formalism, and thus enlarge the possible advantage obtained by simulating stabilizer states using GSLC instead of tableaus.

Following our improvements of the simulation run time, we expanded the GSLC formalism by defining novel algorithms for operations useful when simulating quantum circuits in chapter 5. We presented an $O(k)$ algorithm for tracing out k qubits from a state. We also described an algorithm for calculating fidelity. The simulation of a CZ sequence dominates its run time, which we speed up by using our CZ sequence scheduling algorithm. In addition, we showed that our fidelity algorithm could be used to calculate fidelity between a part of a large state and a smaller state.

In chapter 6, we used our previous results to develop a new subroutine for a state-of-the-art general quantum circuit simulation algorithm by Bravyi et al.[47]. This simulator roughly consists of two steps: the simulation of a large set of stabilizer states, followed by simulating measurement by a Metropolis Markov chain. It is used in popular simulation platforms like Qiskit[60] to simulate circuits containing few non-Clifford gates. Our subroutine is an extension of the GSLC formalism to make it phase-sensitive. This subroutine can speed up the simulation of gates up to $O(n^2)$ (where n is the number of qubits) compared to the original tableau-based subroutine. In typical use cases, $n \approx 50$. It can thus provide a significant speedup when simulating the large set of stabilizer states, depending on the circuit simulated. Our subroutine also allows for the usage of our CZ scheduling algorithm, which can further decrease the simulation time of concentrated CZ sequences in the circuits of the large set of stabilizer states. We also presented a simple $O(n^3)$ (where n is the number of qubits) procedure to convert a state from a phase-sensitive GSLC representation to a CH-tableau representation. This allows us to perform the Metropolis Markov chain step in $O(\chi' nT) + O(\chi' n^3)$ for T Metropolis steps and χ' stabilizer states. As $T >> n$, this procedure gives only a small increase in the run time of the Metropolis Markov chain compared to the original algorithm's complexity of $O(\chi' nT) + O(\chi' n^2)$ whilst allowing for the usage of our faster GSLC-based subroutine to simulate the states in the stabilizer decomposition.

We have shown that our subroutine can achieve a significant speedup on practical use cases. That is, that the set of circuits our subroutine can simulate faster contains interesting simulation tasks. As an example, we have examined the simulation complexity of the quantum approximation optimization algorithm, an algorithm to solve combinatorial optimization problems in near-term gate-based noisy quantum devices [46][32]. We've determined our subroutine can simulate the gates in these circuits up to $O(n)$ faster. We note that the quantum approximation optimization algorithm is also used by Bravyi et al. to validate the performance of the original subroutine in [47].

Several open questions remain, the answers to which could further increase the practical use and speed advantage of our results. One open question is whether the circuits of the large set of stabilizer states in the first step of the general simulation algorithm can be compiled in such a way that it is optimized for GSLC. For instance, by compiling in such

a way that all CZ gates are placed in (concentrated) sequences. Because small variations of this compiled circuit are simulated thousands of times, this compilation step could be relatively expensive and still provide a speedup. A second open question we identify is whether one can determine beforehand (in reasonable time) in what formalism to simulate a circuit. Similar procedures are currently used in practice; e.g., Qiskit contains a rule that determines when to use stabilizer based methods instead of state vectors[60]. One could perhaps base the formalism choice on a heuristic that looks at the properties of the circuit, such as the number and concentration of CZ gates, similar to the heuristic procedure defined in subsection 5.2.

The new algorithms in this thesis contribute to the development of quantum computers (and quantum networks), which is guided by simulation on classical hardware, as they allow for the simulation of larger circuits and networks. By decreasing the amount of time and resources needed to simulate quantum computers, we hope that our contributions will allow researchers to make more informed design decisions and, in turn, accelerate the development of quantum computers ever so slightly.

BIBLIOGRAPHY

[1] R. P. Feynman, "Simulating physics with computers", *Int. J. Theor. Phys*, vol. 21, no. 6/7, 1982.

[2] J. S. Provan and M. O. Ball, "The complexity of counting cuts and of computing the probability that a graph is connected", *SIAM Journal on Computing*, vol. 12, no. 4, pp. 777–788, 1983.

[3] D. M. Greenberger, M. A. Horne, and A. Zeilinger, *Bell's theorem, quantum theory, and conceptions of the universe*, 1989.

[4] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels", *Physical review letters*, vol. 70, no. 13, p. 1895, 1993.

[5] L. K. Grover, "A fast quantum mechanical algorithm for database search", in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[6] M. Sipser, "Introduction to the theory of computation", *ACM Sigact News*, vol. 27, no. 1, 1996.

[7] A. Steane, "Multiple-particle interference and quantum error correction", *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 452, no. 1954, pp. 2551–2577, 1996.

[8] D. Gottesman, "Stabilizer codes and quantum error correction", *arXiv preprint quant-ph/9705052*, 1997.

[9] ——, "The heisenberg representation of quantum computers", *arXiv preprint quant-ph/9807006*, 1998.

[10] ——, "Theory of fault-tolerant quantum computation", *Physical Review A*, vol. 57, no. 1, p. 127, 1998.

[11] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[12] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[13] D. Schlingemann, "Stabilizer codes can be realized as graph codes", *arXiv preprint quant-ph/0111080*, 2001.

[14] M. Grassl, A. Klappenecker, and M. Rotteler, "Graphs, quadratic forms, and quantum codes", in *Proceedings IEEE International Symposium on Information Theory,*, IEEE, 2002, p. 45.

[15] M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*, 2002.

[16] R. Jozsa and N. Linden, "On the role of entanglement in quantum-computational speed-up", *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 459, no. 2036, pp. 2011–2032, 2003.

[17] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits", *Physical Review A*, vol. 70, no. 5, p. 052 328, 2004.

[18] M. Hein, J. Eisert, and H. J. Briegel, "Multiparty entanglement in graph states", *Physical Review A*, vol. 69, no. 6, p. 062 311, 2004.

[19] M. Van den Nest, J. Dehaene, and B. De Moor, "Graphical description of the action of local clifford transformations on graph states", *Physical Review A*, vol. 69, no. 2, p. 022 316, 2004.

[20] S. Bravyi and A. Kitaev, "Universal quantum computation with ideal clifford gates and noisy ancillas", *Physical Review A*, vol. 71, no. 2, p. 022 316, 2005.

[21] S. Anders and H. J. Briegel, "Fast simulation of stabilizer circuits using a graph-state representation", *Physical Review A*, vol. 73, no. 2, p. 022 334, 2006.

[22] M. Hein, W. Dür, J. Eisert, R. Raussendorf, M. Nest, and H.-J. Briegel, "Entanglement in graph states and its applications", *arXiv preprint quant-ph/0602096*, 2006.

[23] R. Jozsa, "On the simulation of quantum circuits", *arXiv preprint quant-ph/0603163*, 2006.

[24] M. Nest, "Classical simulation of quantum computation, the gottesman-knill theorem, and slightly beyond", *arXiv preprint arXiv:0811.0898*, 2008.

[25] M. Ozols, *Clifford group (paper)*, 2008. [Online]. Available: http://home.lu.lv/~sd20008/papers/essays/Clifford%5C%20group%5C%20[paper].pdf.

[26] ——, *Clifford group (presentation)*, 2008. [Online]. Available: http://home.lu.lv/~sd20008/papers/essays/Clifford%5C%20group%5C%20[presentation].pdf.

[27] M. Silva, E. Magesan, D. W. Kribs, and J. Emerson, "Scalable protocol for identification of correctable codes", *Physical Review A*, vol. 78, no. 1, p. 012 347, 2008.

[28] H. Neven, V. S. Denchev, G. Rose, and W. G. Macready, "Training a large scale classifier with the quantum adiabatic algorithm", *arXiv preprint arXiv:0912.0779*, 2009.

[29] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation", *Physical Review A*, vol. 86, no. 3, p. 032 324, 2012.

[30] H. J. Garcia, I. L. Markov, and A. W. Cross, "Efficient inner-product algorithm for stabilizer states", *arXiv preprint arXiv:1210.6646*, 2012.

[31] P. Gawron, J. Klamka, and R. Winiarczyk, "Noise effects in the quantum search algorithm from the viewpoint of computational complexity", *International Journal of Applied Mathematics and Computer Science*, vol. 22, no. 2, pp. 493–499, 2012.

[32] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm", *arXiv preprint arXiv:1411.4028*, 2014.

[33] ——, "A quantum approximate optimization algorithm applied to a bounded occurrence constraint problem", *arXiv preprint arXiv:1412.6062*, 2014.

[34] H. J. Garcia and I. L. Markov, "Simulation of quantum circuits via stabilizer frames", *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2323–2336, 2014.

[35] D. Wecker, B. Bauer, B. K. Clark, M. B. Hastings, and M. Troyer, "Gate-count estimates for performing quantum chemistry on small quantum computers", *Physical Review A*, vol. 90, no. 2, p. 022 305, 2014.

[36] K. Fujii, *Quantum computation with topological codes: from qubit to topological fault-tolerance*. Springer, 2015, vol. 8.

[37] M. Hayashi and T. Morimae, "Verifiable measurement-only blind quantum computing with stabilizer testing", *Physical review letters*, vol. 115, no. 22, p. 220 502, 2015.

[38] S. Bravyi and D. Gosset, "Improved classical simulation of quantum circuits dominated by clifford gates", *Physical review letters*, vol. 116, no. 25, p. 250 501, 2016.

[39] S. Bravyi, G. Smith, and J. A. Smolin, "Trading classical and quantum computational resources", *Physical Review X*, vol. 6, no. 2, p. 021 043, 2016.

[40] M. Epping, H. Kampermann, and D. Bruss, "Robust entanglement distribution via quantum network coding", *New Journal of Physics*, vol. 18, no. 10, p. 103 052, 2016.

[41] H. J. Garcıa, I. L. Markov, and A. W. Cross, "On the geometry of stabilizer states", *arXiv preprint arXiv:1711.07848*, 2017.

[42] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E. Solomonik, E. W. Draeger, E. T. Holland, and R. Wisnieff, "Breaking the 49-qubit barrier in the simulation of quantum circuits", *arXiv preprint arXiv:1710.05867*, 2017.

[43] A. Dahlberg and S. Wehner, "Simulaqron—a simulator for developing quantum internet software", *Quantum Science and Technology*, vol. 4, no. 1, p. 015 001, 2018.

[44] D. Maslov and M. Roetteler, "Shorter stabilizer circuits via bruhat decomposition and quantum circuit transformations", *IEEE Transactions on Information Theory*, vol. 64, no. 7, pp. 4729–4738, 2018.

[45] J. Preskill, "Quantum Computing in the NISQ era and beyond", *Quantum*, vol. 2, p. 79, Aug. 2018, ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. [Online]. Available: https://doi.org/10.22331/q-2018-08-06-79.

[46] M. Alam, A. Ash-Saki, and S. Ghosh, "Analysis of quantum approximate optimization algorithm under realistic noise in superconducting qubits", *arXiv preprint arXiv:1907.09631*, 2019.

[47] S. Bravyi, D. Browne, P. Calpin, E. Campbell, D. Gosset, and M. Howard, "Simulation of quantum circuits by low-rank stabilizer decompositions", *Quantum*, vol. 3, p. 181, 2019.

[48] A. Gheorghiu, T. Kapourniotis, and E. Kashefi, "Verification of quantum computation: An overview of existing approaches", *Theory of computing systems*, vol. 63, no. 4, pp. 715–808, 2019.

[49] D. Gosset, *David gosset: Simulation of quantum circuits by low-rank stabilizer decompositions*, A talk by David Gosset on [47] at the Workshop on Noisy Intermediate-Scale Quantum Technologies (NISQ), Day 2. NISQ was hosted June 6-7, 2019., 2019. [Online]. Available: https://www.youtube.com/watch?v=GPe7mqx6GRQ.

[50] S. Hadfield, *Quantum gate-model approaches to exact and approximate optimization (presentation)*, 2019. [Online]. Available: https://ntrs.nasa.gov/citations/20190028737.

[51] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, "Quest and high performance simulation of quantum computers", *Scientific reports*, vol. 9, no. 1, pp. 1–11, 2019.

[52] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing", *arXiv preprint arXiv:2003.06557*, 2020.

[53] E. v. d. Berg and K. Temme, "Circuit optimization of hamiltonian simulation by simultaneous diagonalization of pauli clusters", *arXiv preprint arXiv:2003.13599*, 2020.

[54] Glosser.ca, *Bloch sphere*, Distributed under a Creative Commons Attribution-Share Alike 3.0 Unported license., 2020. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/f/f4/Bloch_Sphere.svg/1000px-Bloch_Sphere.svg.png.

[55] A. Heimendahl, F. Montealegre-Mora, F. Vallentin, and D. Gross, "Stabilizer extent is not multiplicative", *arXiv preprint arXiv:2007.04363*, 2020.

[56] C. Huang, M. Newman, and M. Szegedy, "Explicit lower bounds on strong quantum simulation", *IEEE Transactions on Information Theory*, 2020.

[57] A. Kelly, *Libtangle/graph-state*, https://github.com/libtangle/graph-state, 2020.

[58] S. Aaronson, *Lecture 28 of introduction to quantum information science*. [Online]. Available: https://www.scottaaronson.com/qclec/28.pdf.

[59] S. Aaronson, G. Kuperberg, C. Granade, and V. Russo, *Complexity zoo*. [Online]. Available: https://complexityzoo.uwaterloo.ca/Complexity_Zoo.

[60] P. Calpin, *The extended stabilizer simulator*. [Online]. Available: https://qiskit.org/documentation/tutorials/simulators/6_extended_stabilizer_tutorial.html.