An Introduction to Stabilizer Circuit Simulation

Student Name (Student ID – CO481)

March 12, 2009

Abstract

The Gottesman-Knill theorem states that a circuit made up of controlled-not, Hadamard, phase, and measurement gates can be simulated efficiently on a classical computer and provides a method for doing so in $O(n^3)$ time. Aaronson and Gottesman's paper *Improved Simulation of Stabilizer Circuits* describes how to reduce the simulation time to $O(n^2)$ and explores the computational complexity implications of the theorem. An immediate corollary of the Gottesman-Knill theorem is that the type of circuits to which the theorem applies – known as stabilizer circuits – are not universal for quantum computation. Aaronson and Gottesman answer the question of where stabilizer circuits lie in the realm of computational complexity theory by showing that they are complete for the classical complexity class $\oplus L$ and thus are likely not universal even for classical computation.

1 Introduction

One difficulty with designing quantum algorithms is the lack of a method for easily testing and debugging circuits. To debug circuits in the classical setting, programmers can add test conditions or read intermediate data to find where problems occur. In the quantum setting, tricks like these may disturb the coherence of the quantum states being manipulated by the quantum circuit. Furthermore, it seems likely that large-scale quantum computers may not be built for many years to come, while physicists and chemists are looking for ways to simulate quantum systems now. With these problems in mind, a method for simulating quantum circuits on a classical computer is sought.

Several packages already exist for simulating general quantum systems on classical computers [5, 6, 7, 8], but the running time of these simulators grows exponentially in the number of qubits being simulated. It is of course expected that no method exists for simulating *all* quantum circuits efficiently on a classical computer, as that would imply that quantum computers could at best offer a polynomial speed-up over classical computers – something that was shown not to be true for at least some problems by the n-bit version of the Deutsch-Jozsa algorithm [11]. Attention is thus restricted to trying to find specific classes of quantum circuits that *can* be simulated efficiently on classical computers.

The Gottesman-Knill theorem [2, Theorem 10.5.4] provides a simple example of one such class of quantum circuits, known as stabilizer circuits. The theorem gives a constructive method for simulating an n-qubit stabilizer circuit in time that is polynomial in n. Simulating measurements in the computational basis requires $O(n^3)$ time in the original proposed simulation method, while simulating the other gates that make up stabilizer circuits requires only $O(n)$ time. Aaronson and Gottesman provide a method for improving the efficiency of simulating measurements within stabilizer circuits, which then reduces the time needed to simulate a stabilizer circuit to $O(n^2)$ [1]. The cost of this speed-up is a doubling of the number of bits that must be kept track of throughout the algorithm. They also answer the question of where stabilizer circuits live in relation to other well-known computational complexity classes.

The remainder of this paper proceeds as follows. Section 2 outlines the notation and terminology necessary to discuss Aaronson and Gottesman's results, Section 3 describes the original approach for simulating stabilizer circuits, Section 4 outlines the algorithm that Aaronson and Gottesman proposed for improving measurement simulation efficiency, and Section 5 discusses some of the computational complexity implications of the fact that performing this simulation is possible.

2 Preliminaries

First, we recall the definitions of several 1-qubit quantum gates:

$$\begin{array}{ccc} I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \\ Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} & Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} & S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \end{array} \tag{1}$$

The Gottesman-Knill Theorem states that a quantum circuit consisting only of $CNOT$, H, S, and 1-qubit measurement gates can be simulated efficiently on a classical computer – a quantum circuit of this form is known as a *stabilizer circuit*. The group generated by H and S under multiplication is known as the *Clifford group*, which contains each of the four *Pauli matrices* I, X, Y, and Z. A stabilizer circuit that contains no measurement gates is thus referred to as a *Clifford group circuit*.

Define the group $\mathcal{P}_n$ of *n-qubit Pauli operators* to be the group of all tensor products of n Pauli matrices, together with the multiplicative factors ± 1 and $\pm i$. The multiplicative factors ± 1 and $\pm i$ are required to ensure that $\mathcal{P}_n$ is indeed a true group. As a notational convenience, tensor product signs are omitted when describing elements of $\mathcal{P}_n$ (eg. XYZ means $X \otimes Y \otimes Z$). Similarly, subscripts may also be used on a Pauli matrix to indicate that those Pauli matrices appear in the indicated tensor slots, with the identity matrix elsewhere (eg. $Z_{2,4}$ means $I \otimes Z \otimes I \otimes Z \otimes I \otimes \cdots \otimes I$, where the number of identity matrices being tensored will be clear from context).

Given a pure state $|\psi\rangle$, it is said that a unitary matrix U *stabilizes* $|\psi\rangle$ if $U|\psi\rangle = |\psi\rangle$, where global phase is not ignored (eg. the Pauli X matrix stabilizes the state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$

and the Pauli Z matrix stabilizes $|0\rangle$). The identity matrix I stabilizes all states, while $-I$ stabilizes no states. It is clear that the set $\mathrm{Stab}(|\psi\rangle)$ of stabilizers of $|\psi\rangle$ is a group because if $U|\psi\rangle = |\psi\rangle$ and $V|\psi\rangle = |\psi\rangle$ then $U^{-1}|\psi\rangle = |\psi\rangle$ and $UV|\psi\rangle = |\psi\rangle$ as well.

3 Simulating stabilizer circuits

By the Gottesman-Knill Theorem, it is known that it is possible to efficiently simulate stabilizer circuits on a classical computer. In order to see how this simulation can be done, another theorem is needed:

Theorem 1 *Given an n-qubit state $|\psi\rangle$, the following are equivalent:*

1. *$|\psi\rangle$ can be obtained from $|0\rangle^{\otimes n}$ by $CNOT$, H, and S gates only.*
2. *$|\psi\rangle$ can be obtained from $|0\rangle^{\otimes n}$ by $CNOT$, H, S, and measurement gates only.*
3. *$|\psi\rangle$ is stabilized by exactly 2^n Pauli operators.*
4. *$|\psi\rangle$ is uniquely determined by $S(|\psi\rangle) \equiv Stab(|\psi\rangle) \cap \mathcal{P}_n$, the group of Pauli operators that stabilize $|\psi\rangle$.*

To see why this theorem is useful in this setting, first note that a general quantum state requires about 2^n parameters to be completely described. Theorem 1 says, however, that any state $|\phi\rangle$ that can be obtained from $|0\rangle^{\otimes n}$ via a stabilizer circuit can be described uniquely in terms of $S(|\psi\rangle)$, and $|S(|\psi\rangle)| = 2^n$. It then follows that any $|\psi\rangle$ of the type described by Theorem 1 can be described by exactly n elements of $\mathcal{P}_n$ (in particular, the n generators of $S(|\psi\rangle)$). Furthermore, each generator of $S(|\psi\rangle)$ can be described by $2n+1$ bits: 2 bits for each of the n Pauli matrices and 1 bit for the phase (note that the only possible phases are ± 1, since a phase of $\pm i$ would imply that $-I \cdots I \in S(|\psi\rangle)$, which is impossible).

Putting all of this together, Theorem 1 implies that any state $|\phi\rangle$ that can be obtained from $|0\rangle^{\otimes n}$ via a stabilizer circuit can be described uniquely by $n(2n+1) = 2n^2 + n$ bits. Equally important is the fact that these bits can be updated *efficiently* after a $CNOT$, H, S, or measurement gate is applied to $|\psi\rangle$. Throughout the rest of this paper, when a quantum state $|\psi\rangle$ is referred to, it is assumed that it satisfies the conditions of Theorem 1 unless stated otherwise.

3.1 Simulating Clifford group circuits

Figure 1 provides simple rules for updating the Pauli X and Z matrices under conjugation by various common gates from the Clifford group. These rules can easily be verified by noticing that the operator in the "Output" column is simply the operator in the "Input" column conjugated by the operator in the "Operation" column (eg. $HXH^\dagger = Z$).

In order to update a Pauli matrix not listed in Figure 1, break that matrix up into a product of X and Z matrices, update each term in the product, and multiply the updated

Operation	Input	Output
$CNOT$	X_1	X_1X_2
	X_2	X_2
	Z_1	Z_1
	Z_2	Z_1Z_2
H	X	Z
	Z	X
S	X	Y
	Z	Z

Operation	Input	Output
X	X	X
	Z	$-Z$
Y	X	$-X$
	Z	$-Z$
Z	X	$-X$
	Z	Z

Figure 1: The *update rules* for updating Pauli X and Z matrices upon conjugation by various Clifford group gates – for the $CNOT$ gate, qubit 1 is the control and qubit 2 is the target. Adapted from [2, Section 10.5.2].

matrices together in the same order. For example, the update rule for applying S to Y follows from the fact that $Y = iXZ$. The update rule applied to X yields Y and the update rule applied to Z yields Z, so the update rule applied to Y yields $i(Y)(Z) = -X$.

Thus to update $S(|\psi\rangle)$ after a quantum gate is applied to the a^{th} qubit, simply apply the update rule to the a^{th} tensor slot of each of the its n generators. Updating $S(|\psi\rangle)$ in this way takes $O(n)$ time as the update rules needs to be applied n times to simulate an H, S, X, Y, or Z gate, and $2n$ updates need to be performed to simulate a $CNOT$ gate.

3.2 Simulating measurements in the computational basis

In what follows, denote the n generators of $S(|\psi\rangle)$ by $g_1, g_2, \ldots, g_n$. To understand how measurements can be simulated via the stabilizer formalism, consider the measurement of an n-qubit Pauli operator $P \in \mathcal{P}_n$ as in [2, Section 4.4] and recall that measuring the observable P corresponds to measuring an eigenvalue of P and leaving the system in a corresponding eigenstate of P. It is thus not difficult to see that measuring the k^{th} qubit in the computational basis corresponds to measuring the observable Z_k. Thus the ability to simulate measurements of observables from $\mathcal{P}_n$ implies the ability to simulate measurements in the computational basis.

Thus assume without loss of generality that $P = Z_k$ for some $k \in \{1, 2, \ldots, n\}$. To see how $S(|\psi\rangle)$ changes under measurement of P, consider two cases.

Case 1: P anti-commutes with one or more of the generators of $S(|\psi\rangle)$. Suppose that P anti-commutes with g_q. Then one can further assume that P commutes with $g_1, g_2, \ldots, g_{q-1}, g_{q+1}, \ldots, g_n$ because if there were a second element of $S(|\psi\rangle)$ that P anti-commuted with (say g_j), then P would commute with $g_q g_j$ so g_j could simply be replaced by $g_q g_j$ in the list of generators of $S(|\psi\rangle)$ – updating the generators in this way is a process that takes $O(n^2)$ time in general. It is not difficult to show that the measurement probabilities

for the two eigenvalues $+1$ and -1 of P are both $\frac{1}{2}$ and thus the measurement outcome can be determined uniformly randomly with equal probability. After randomly choosing a measurement outcome the rule for updating $S(|\psi\rangle)$ is to replace g_q in the list of generators by $\pm P$, where the sign in front of P corresponds to the measurement outcome chosen.

Case 2: P commutes with all the generators of $S(|\psi\rangle)$. In this case we know that $g_j P|\psi\rangle = P g_j|\psi\rangle = P|\psi\rangle$ for all generators g_j and thus $S(P|\psi\rangle) = S(|\psi\rangle)$. This implies that $P|\psi\rangle$ is a multiple of $|\psi\rangle$ which, together with the fact that $P^2 = I$, implies that $P|\psi\rangle = \pm|\psi\rangle$. Thus, the measurement of P gives ± 1 with probability 1 and does not disturb the state of the system (and thus does not alter $S(|\psi\rangle)$). However, determining whether the measurement outputs $+1$ or -1 (equivalently $|0\rangle$ or $|1\rangle$, respectively) seems to require inverting an $n \times n$ matrix, which takes $O(n^\omega)$ time, where ω is the *exponent of matrix multiplication* [9]. The current lowest known exponent of matrix multiplication is 2.376 [10], although in practice the naive (standard) algorithm of matrix multiplication is used for numerical stability and implementation reasons, giving a running time of $O(n^3)$.

Note that Z_k commutes with $Q \in \mathcal{P}_n$ if and only if the k^{th} tensor slot of Q looks like Z or I. Thus the two cases above can be distinguished in $O(n)$ time by simply looking at the k^{th} tensor slot of each of $g_1, g_2, \ldots, g_n$.

This procedure shows how to efficiently simulate stabilizer circuits on a classical computer. Recall, however, that if a measurement is deterministic (ie. it falls into case 2 above) then the simulation requires $O(n^3)$ time; a significant increase over the $O(n)$ time needed to simulate Clifford group gates and the $O(n^2)$ time needed to simulate random measurements (as in case 1 above). This large running time for simulating certain measurements is the motivation for looking for an improved algorithm for simulating stabilizer circuits.

4 Improving simulation of stabilizer circuits

This section describes the *tableau algorithm* [1] for simulating stabilizer circuits. The tableau algorithm matches the $O(n)$ running time for simulating Clifford group gates, but is able to simulate *any* (ie. deterministic or random) measurement in the computational basis in time $O(n^2)$. The cost of this speed-up is that now $4n^2 + 2n$ bits are needed to describe the state that is being worked with; twice as many bits as were needed for the algorithm outlined in Section 3. These extra bits are used to keep track of n *destabilizer generators*; n-qubit Pauli operators that together with the stabilizer generators (and iI [15]) generate all of $\mathcal{P}_n$.

4.1 Representing a state by a tableau

Write the i^{th} destabilizer generator of $S(|\psi\rangle)$ as $R_i = \pm P_{i1} P_{i2} \cdots P_{in}$ and the i^{th} stabilizer generator of $S(|\psi\rangle)$ as $R_{n+i} = \pm P_{(n+i)1} P_{(n+i)2} \cdots P_{(n+i)n}$, where each P_{ij} is a Pauli matrix. We then can define a *tableau*:

Definition 2 *The tableau of a state $|\psi\rangle$ is a matrix consisting of binary variables x_{ij}, z_{ij} for all $i \in \{1, \ldots, 2n\}$, $j \in \{1, \ldots, n\}$, and r_i for all $i \in \{1, \ldots, 2n\}$:*

$$\left(\begin{array}{ccc|ccc|c} x_{11} & \cdots & x_{1n} & z_{11} & \cdots & z_{1n} & r_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n1} & \cdots & x_{nn} & z_{n1} & \cdots & z_{nn} & r_n \\ \hline x_{(n+1)1} & \cdots & x_{(n+1)n} & z_{(n+1)1} & \cdots & z_{(n+1)n} & r_{n+1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{(2n)1} & \cdots & x_{(2n)n} & z_{(2n)1} & \cdots & z_{(2n)n} & r_{2n} \end{array}\right)$$

where (see Appendix I):

$$x_{ij} = \begin{cases} 1 & \text{if } P_{ij} = X \text{ or } P_{ij} = Y \text{ and} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{ij} = \begin{cases} 1 & \text{if } P_{ij} = Y \text{ or } P_{ij} = Z \text{ and} \\ 0 & \text{otherwise} \end{cases}$$

$$r_i = \begin{cases} 1 & \text{if } R_i \text{ has negative phase and} \\ 0 & \text{otherwise.} \end{cases}$$

Recalling that $Y = iXZ$, the above rule for x_{ij} and z_{ij} can be intuitively thought of as $x_{ij} = 1$ if and only if the j^{th} tensor slot of R_i contains an X and $z_{ij} = 1$ iff the j^{th} tensor slot of R_i contains a Z. It can also be seen that the first n rows of a tableau represent the n destabilizer generators for that state and the last n rows represent the n stabilizer generators for that state.

As a simple example of a tableau, if $|\psi\rangle = |00\rangle$ then $S(|\psi\rangle) = \{+ZI, +IZ\}$. That is, the stabilizers are $R_3 = +ZI$ and $R_4 = +IZ$. It is not difficult to see that we can then choose the destabilizers to be $R_1 = +XI$ and $R_2 = +IX$, as the set $\{+XI, +IX, +ZI, +IZ, +iI\}$ generates all of $\mathcal{P}_n$. It is then easy to verify that $r_i = 0\ \forall i$, $x_{11} = 1$, $x_{22} = 1$, $x_{ij} = 0$ for all other i, j, $z_{31} = 1$, $z_{42} = 1$, and $z_{ij} = 0$ for all other i, j. It thus follows that a tableau for this state is:

$$\left(\begin{array}{cc|cc|c} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right) \tag{2}$$

Note that any given state will have several tableaux, so the obvious generalization of the above tableau to more qubits will be the starting point of the tableau algorithm, as it is a "nice" way of representing the state $|0\rangle^{\otimes n}$. The procedures for simulating $CNOT$, H, and S gates are provided in Appendix II and follow directly from the update rules given in Figure 1. It is not difficult to see that simulating these gates requires only $O(n)$ time, as in the original algorithm.

4.2 Improving measurement simulations

In order to simulate a measurement of qubit a in the computational basis, a procedure called rowsum(h, j) is repeatedly used. The rowsum(h, j) procedure replaces R_h by $R_h R_j$ – the details of its implementation are provided in Appendix II. Two cases are considered depending on whether the measurement will output a random (case 1) or deterministic (case 2) result. Note that these two cases correspond exactly to the two cases described in Section 3.2 for the original algorithm. As was the case earlier, these two cases can be distinguished in $O(n)$ time.

Case 1: There exists $p \in \{n+1, \ldots, 2n\}$ such that $x_{pa} = 1$. Let q be the smallest such p. This case corresponds to the situation where the a^{th} tensor slot of at least one stabilizer of $|\psi\rangle$ looks like either X or Y (ie. the a^{th} tensor slot of some stabilizer does not look like I or Z). This means that the a^{th} qubit of $|\psi\rangle$ does not look like $|0\rangle$ or $|1\rangle$ and thus the measurement outcome is random. The state therefore needs to be updated, which is done as follows.

For all $j \in \{1, \ldots, q-1, q+1, \ldots, 2n\}$ such that $x_{ja} = 1$ call rowsum(j, q). This step corresponds to replacing g_j by $g_q g_j$ in the list of generators of $S(|\psi\rangle)$ whenever g_q anti-commutes with g_j. Next, set the $(q-n)^{th}$ row of the tableau equal to the q^{th} row of the tableau. Then set $x_{qj} = z_{qj} = 0$ for all $j \in \{1, \ldots, n\}$, except set $z_{qa} = 1$. Finally, set r_q to be 0 or 1 with equal probability. These last three steps correspond to replacing g_q by $\pm P$, where $P = Z_a$ can be thought of as the observable that is being measured. The measurement outcome is r_q.

Note that it takes $O(n^2)$ time to simulate a measurement using this method, as the rowsum procedure is run $O(n)$ times and each execution of the rowsum procedure takes $O(n)$ time. This agrees with the $O(n^2)$ running time of simulating random measurements in the original algorithm.

Case 2: There does not exist $p \in \{n+1, \ldots, 2n\}$ such that $x_{pa} = 1$. In this case $|\psi\rangle$ looks like $|0\rangle$ or $|1\rangle$ on the a^{th} qubit and thus the measurement is deterministic and the state does not need to be updated. All that needs to be determined is whether the measurement will result in 0 or 1. To this end, create a $(2n+1)^{st}$ row on the tableau and set it identially equal to zero. Then for all $j \in \{1, \ldots, n\}$ such that $x_{ja} = 1$ call rowsum$(2n+1, j+n)$ and return r_{2n+1} as the measurement outcome.

This procedure corresponds to multiplying together all of the stabilizer generators R_{j+n} such that the destabilizer generator R_j anti-commutes with Z_a, and returning the resulting phase (± 1) as the measurement outcome. Notice that this procedure takes $O(n^2)$ time to carry out in general, as the rowsum procedure is called $O(n)$ times and takes $O(n)$ time to run each time. Also notice that this procedure is the reason that the n destabilizer generators were added to the algorithm; in no other part of the tableau algorithm are they needed nor do they provide a speed-up elsewhere.

To see why this procedure correctly computes the measurement outcome, notice that Z_a must commute with all generators of $S(|\psi\rangle)$ (just as in case 2 of the original algorithm). It's not hard to then show that either $Z_a \in S(|\psi\rangle)$ or $-Z_a \in S(|\psi\rangle)$, but not both (if both were in the stabilizer, then $-Z_a Z_a = -I$ would also be in the stabilizer, which can't happen).

Furthermore, the sign in front of Z_a in the stabilizer corresponds exactly to whether the measurement outcome is 0 or 1, since $|0\rangle$ is stabilized by Z and $|1\rangle$ is stabilized by $-Z$.

In order to determine the sign of Z_a in the stabilizer, a method for computing $\pm Z_a$ by multiplying together stabilizer generators is developed. Notice for the initial Tableau (2) that the j^{th} stabilizer generator anti-commutes with the j^{th} destabilizer generator but commutes with the rest of the destabilizer generators. It is not difficult to prove that these properties remain after any number of gates from a stabilizer circuit are applied. It is similarly easy to see that Z_a will anti-commute with the destabilizer generator R_j if and only if the stabilizer generator R_{j+n} appears in the list of stabilizer generators that are multiplied together to compute $\pm Z_a$. Thus, multiplying together all of the stabilizer generators R_{j+n} such that R_j anti-commutes with Z_a computes $\pm Z_a$, where the sign corresponds to the measurement outcome.

The new tableau algorithm provides a method for simulating both random and deterministic measurements on a classical computer in $O(n^2)$ time. Note that in both case 1 and case 2 above that the number of times that rowsum(h, j) is called depends on the number of stabilizer generators R_{n+j} and destabilizer generators R_j such that their a^{th} tensor slot looks like X or Y. One can observe from Figure 1 and the Tableau (2), however, that it is expected that states that started as $|0\rangle^{\otimes n}$ and have had very few $CNOT$, H and S gates applied to them will have fewer than $O(n)$ such stabilizer and destabilizer generators, resulting in measurements that take less than $O(n^2)$ time to simulate. It is thus expected that measurements performed sufficiently early in the circuit will be simulated by the tableau algorithm much quicker than measurements that are performed later in the circuit, a fact that is confirmed by Aaronson and Gottesman [1, Figure 2].

5 Computational complexity implications

One of the immediate questions that the Gottesman-Knill theorem raises is that of where stabilizer circuits sit in terms of computational complexity theory. Clearly stabilizer circuits are an extremely important class of quantum circuits, as they are sufficient to implement many important quantum algorithms and protocols such as quantum teleportation [3] and the superdense coding [4]. One can observe, however, that stabilizer circuits are not universal for quantum computation, as the Gottesman-Knill theorem would then imply that quantum computers can be simulated efficiently by classical computers; something we know not to be true because $O(2^{n-1})$ queries are needed to solve the constant vs. balanced problem classically, whereas just 1 query suffices quantumly [11].

It turns out that familiarity with the classical complexity class $\oplus L$ (pronounced "parity-L") [12] is required to understand the computational complexity of stabilizer circuits. Some complexity class definitions are provided here for completeness.

Definition 3 *P is the class of problems solvable by a Turing machine in time that is polynomial in n, where n is the size of the input.*

Definition 4 *L is the class of problems solvable by a Turing machine restricted to use an amount of memory in $O(\log(n))$, where n is the size of the input.*

Definition 5 *$\oplus L$ is the class of all problems that are reducible to simulating a circuit whose size is polynomial in n and is composed entirely of X and $CNOT$ gates, acting on the initial state $|0\rangle^{\otimes n}$.*

It is clear from these definitions that $\oplus L \subseteq P$ because – roughly speaking – any problem that can be reduced to simulating a circuit consisting of only a polynomial number of X and $CNOT$ gates must be solvable in polynomial time. It is currently not known whether this inclusion is strict or not, but it is believed that $\oplus L \neq P$. Similarly, it is known that $L \subseteq \oplus L$ – a fact that is made clearer by a more complicated alternate definition for $\oplus L$ [1, 12]. It is conjectured that $L \neq \oplus L$, but this is also currently unknown.

It is simple to see that $X = HSSH$ and thus it follows that any circuit consisting entirely of X and $CNOT$ gates is a stabilizer circuit – that is, any problem in $\oplus L$ is reducible to simulating a polynomial-size stabilizer circuit. In other words, simulating stabilizer circuits is $\oplus L$-hard. Aaronson and Gottesman further show that simulating stabilizer circuits is a problem that is *in* $\oplus L$, which in turn shows that simulating stabilizer circuits is $\oplus L$-complete. It thus follows that it is likely the case that stabilizer circuits are not universal even for *classical* computation, because if they were then it would follow that $\oplus L = P$.

In order to prove that simulating stabilizer circuits is a problem that is in $\oplus L$, Aaronson and Gottesman argue that the principle of deferred measurement shows that they can assume without loss of generality that the stabilizer circuit contains only one measurement gate, and it is located at the end of the circuit. They then show that the tableau algorithm allows for the stabilizer circuit to be simulated using only $O(\log(n))$ memory provided they can simulate subcircuits consisting of X and $CNOT$ gates with an oracle. This then shows that the problem of simulating a stabilizer circuit with an oracle for simulating subcircuits of X and $CNOT$ gates is in L. It is then a result of Hertrampf, Reith, and Vollmer [13] that shows that this implies that simulating a stabilizer circuit is a problem that must be in $\oplus L$.

Since simulating stabilizer circuits is a problem that is in $\oplus L$, it is the case that stabilizer circuits can be simulated efficiently by circuits consisting entirely of X and $CNOT$ gates. This provides another way of looking at the Gottesman-Knill theorem. Any stabilizer circuit can be simulated efficiently by a circuit made up entirely of X and $CNOT$ gates. X and $CNOT$ gates can be thought of as classical operations, though (NOT and controlled-NOT, respectively), so it follows that stabilizer circuits can of course be simulated efficiently on a classical computer.

6 Conclusion

The Gottesman-Knill theorem, tableau algorithm, and several related results all help pin down exactly where stabilizer circuits reside in the realm of computational complexity. Even though stabilizer circuits are an extremely important class of quantum circuits as $CNOT$, H, S, and Pauli gates form the foundation of several important quantum algorithms, it was

shown that they are not universal for quantum computation and likely not universal even for classical computation.

The work of Aaronson and Gottesman in [1] improves greatly on the already-important Gottesman-Knill theorem, but certainly there are related avenues of interest that are still open. It was shown in [14] that generalizing stabilizer circuits by adding any 1- or 2-qubit gate not generated by $CNOT$, H, and S results in a set of gates that is universal for quantum computation. Is there a set of quantum gates that is neither universal for quantum computation nor efficiently simulable on a classical computer? Is there a way to improve the tableau algorithm so that simulation of measurements on stabilizer circuits can be performed in $O(n)$ time? Answers to these questions will help shed more light on the delicate link between quantum computational complexity and classical computational complexity that the Gottesman-Knill theorem highlights.

Acknowledgements. This paper was primarily based on the paper *Improved Simulation of Stabilizer Circuits* by S. Aaronson and D. Gottesman [1]. Additional information about the Gottesman-Knill theorem and its proof were provided by the book *Quantum Computation and Quantum Information* by M. Nielsen and I. Chuang [2].

References

[1] Aaronson, S. and Gottesman, D., *Improved Simulation of Stabilizer Circuits*. http://xxx.lanl.gov/abs/quant-ph/0406196

[2] M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*, Cambridge University Press, Cambridge, 2000.

[3] C. H. Bennett, G. Brassard, C. Crpeau, R. Jozsa, A. Peres, W. K. Wootters, *Teleporting an Unknown Quantum State via Dual Classical and Einstein-Podolsky-Rosen Channels*, Phys. Rev. Lett. 70, 1895-1899, 1993.

[4] C. Bennett and S. J. Wiesner. *Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states*. Phys. Rev. Lett., 69:2881, 1992.

[5] B. Oemer, 2003. http://tph.tuwian.ac.at/ oemer/qcl.html.

[6] G. F. Viamontes, I. L. Markov, and J. P. Hayes, Quantum Information Processing 2(5), 347, 2004. quant-ph/0309060.

[7] G. F. Viamontes, M. Rajagopalan, I. L. Markov, and J. P. Hayes, in *Proc. Asia and South-Pacific Design Automation Conference*, 295, 2003. quant-ph/0208003.

[8] K. M. Obenland and A. M. Despain, in *High Performance Computing*, 1998. quant-ph/9804039.

[9] Cohn, H., Umans, C., Kleinberg, R., Szegedy, B., *Group-theoretic Algorithms for Matrix Multiplication,* Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science 2005, IEEE Computer Society, 2005, 438-449.

[10] D. Coppersmith and S. Winograd, J. Symbolic Comput. 9(3), 251, 1990.

[11] D. Deutsch and R. Jozsa, *Rapid solutions of problems by quantum computation.* Proceedings of the Royal Society of London A, 439-553, 1992.

[12] S. Aaronson, 2005. http://www.complexityzoo.com.

[13] U. Hertrampf, S. Reith, and H. Vollmer. *A note on closure properties of logspace MOD classes*, Information Processing Letters 75(3):91-93, 2000.

[14] Y. Shi. Quantum Information and Computation 3(1), 84, 2003. quant-ph/0205115.

[15] The matrix iI needs to be added to the list of generators in general to ensure that it is possible to find n destabilizer Pauli operators that, together with $S(|\psi\rangle)$, truly generate all of $\mathcal{P}_n$ including the four possible multiplicative factors.

Appendix I: Corrections

In Aaronson and Gottesman's paper [1] it is said on page 4 that $x_{ij} = 1$ if $P_{ij} = Y$ or $P_{ij} = Z$ (and $x_{ij} = 0$ otherwise) and that $z_{ij} = 1$ if $P_{ij} = X$ or $P_{ij} = Y$ (and $z_{ij} = 0$ otherwise). There was a typo in these definitions, however, as they would lead to the tableau for the state $|00\rangle$ (ie. the "identity matrix") being

$$\left(\begin{array}{cc|cc|c} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{array}\right).$$

Similarly, it is easy to see that the measurement protocol described on page 5 of their paper does not work as intended under the given definitions of x_{ij} and z_{ij}. Their paper provides the correct definitions of $x_{ij}^{(T)}$ and $z_{ij}^{(T)}$ (which are very related to x_{ij} and z_{ij}) on page 8. The correct definitions are $x_{ij} = 1$ if $P_{ij} = X$ or $P_{ij} = Y$ (and $x_{ij} = 0$ otherwise) and $z_{ij} = 1$ if $P_{ij} = Y$ or $P_{ij} = Z$ (and $z_{ij} = 0$ otherwise), as stated in Definition 2.

Appendix II: Details of the tableau algorithm

Here we describe how to simulate $CNOT$, H and S gates using the tableau representation of a state.

Simulating $CNOT$ from control a to target b. For all $i \in \{1, \ldots, 2n\}$ set $r_i = r_i \oplus x_{ia} z_{ib} (x_{ib} \oplus z_{ia} \oplus 1)$, $x_{ib} = x_{ib} \oplus x_{ia}$, and $z_{ia} = z_{ia} \oplus z_{ib}$.

Simulating H on qubit a. For all $i \in \{1, \ldots, 2n\}$ set $r_i = r_i \oplus x_{ia} z_{ia}$ and swap x_{ia} with z_{ia}.

Simulating S on qubit a. For all $i \in \{1, \ldots, 2n\}$ set $r_i = r_i \oplus x_{ia} z_{ia}$ and $z_{ia} = z_{ia} \oplus x_{ia}$.

These update rules simply implement the update rules given in Figure 1. For example, if the i^{th} destabilizer has its a^{th} tensor slot occupied by X then $x_{ia} = 1$ and $z_{ia} = 0$. Thus, after simulating an S gate on qubit a we have $x_{ia} = 1$, $z_{ia} = 0 \oplus 1 = 1$ (recall from Definition 2 that this corresponds to the a^{th} tensor of the destabilizer now being occupied by Y) and $r_i = r_i \oplus 0 = r_i$ (ie. the phase of the destabilizer does not change). This agrees with the rule given in Figure 1 that says S updates X to Y.

The rowsum(h, j) procedure. First, define the function $g : \{0, 1\}^4 \mapsto \{-1, 0, 1\}$ by:

$$g(x_1, z_1, x_2, z_2) = \begin{cases} 0, \text{ if } x_1 = z_1 = 0 \\ z_2 - x_2, \text{ if } x_1 = z_1 = 1 \\ z_2(2x_2 - 1), \text{ if } x_1 = 1, z_1 = 0 \\ x_2(1 - 2z_2), \text{ if } x_1 = 0, z_1 = 1 \end{cases}$$

Intuitively, g is a function that returns the exponent to which the imaginary number i is raised (either 0, 1, or -1) when the Pauli matrices represented by $x_1 z_1$ and $x_2 z_2$ are multiplied together. For example, if $x_1 = z_2 = 0$ and $z_1 = x_2 = 1$ then Definition 2 shows that $x_1 z_1$ and $x_2 z_2$ represent Z and X, respectively. Multiplying Z and X together gives $ZX = iY$. Since the exponent on i is 1, we know that $g(0, 1, 1, 0) = 1$. This fact is confirmed by the formula for g given earlier.

The rowsum procedure takes two arguments h and j and performs the following set of updates to the tableau:

For all $k \in \{1, \ldots, n\}$ set $x_{hk} = x_{jk} \oplus x_{hk}$ and set $z_{hk} = z_{jk} \oplus z_{hk}$.

Next, define $m \equiv 2r_h + 2r_j + \sum_{k=1}^{n} g(x_{jk}, z_{jk}, x_{hk}, z_{hk}) \bmod 4$ and set $r_h = \frac{m}{2}$. Note that this is a valid operation because m will never equal 1 or 3.

Updating the x's and z's corresponds to multiplying the h^{th} and j^{th} generators together, while updating r_h keeps track of the phase of the new h^{th} generator.