Fast simulation of stabilizer circuits using a graph-state representation

Simon Anders[1,*] and Hans J. Briegel[1,2]
[1]*Institut für Theoretische Physik, Universität Innsbruck, Innsbruck, Austria*
[2]*Institut für Quantenoptik und Quanteninformation der Österreichischen Akademie der Wissenschaften, Innsbruck, Austria*
(Received 25 April 2005; published 21 February 2006)

According to the Gottesman-Knill theorem, a class of quantum circuits—namely, the so-called stabilizer circuits—can be simulated efficiently on a classical computer. We introduce an algorithm for this task, which is based on the graph-state formalism. It shows significant improvement in comparison to an existing algorithm, given by Gottesman and Aaronson, in terms of speed and of the number of qubits the simulator can handle. We also present an implementation.

DOI: 10.1103/PhysRevA.73.022334 PACS number(s): 03.67.Lx, 02.70.−c

I. INTRODUCTION

Protocols in quantum-information science often use entangled states of a large number of qubits. A major challenge in the development of such protocols is to actually test them using a classical computer. This is because a straight-forward simulation is typically exponentially slow and hence intractable. Fortunately, the Gottesman-Knill theorem [1,2] states that an important subclass of quantum circuits can be simulated efficiently: namely, so-called *stabilizer circuits*. These are circuits that use only gates from a restricted subset, the so-called Clifford group. Many techniques in quantum information use only Clifford gates, most importantly the standard algorithms for entanglement purification [3–7] and for quantum error correction [8–11]. Hence, if one wishes to study such networks, one can simulate them numerically.

The usual proof of the Gottesman-Knill theorem (as stated, e.g., in [2]) contains an algorithm that can carry out this task in time $O(N^3)$, where N is the number of qubits. Especially for the applications just mentioned, one is interested in a large N: For entanglement purification one might want to study large ensembles of states and for quantum error correction concatenations of codes. The cubic scaling renders this extremely time consuming, and a more efficient algorithm should be of great use.

Recently, Aaronson and Gottesman presented such an algorithm (and an implementation of it called CHP) in Ref. [12], whose time and space requirements scale only quadratically with the number of qubits. In the present paper, we further improve on this by presenting an algorithm that for typical applications only requires time and space of $O(N \log N)$. While Aaronson and Gottesman's simulator, when used on an ordinary desktop computer, can simulate already systems of several thousands of qubits in a reasonable time, we have used our simulator for over 10^6 qubits. This provides a valuable tool for investigating complex protocols such as our study of multiparty entanglement purification protocols in Ref. [13].

The crucial new ingredient is the use of so-called graph states. Graph states have been introduced in [14] for the study of entanglement properties of certain multiqubit systems; they were used as starting point for the one-way quantum computer (i.e., measurement-based quantum computing) [15] and found to be suited to give a graphical description of Calderbank-Shor-Steane (CSS) codes (for quantum error correction) [16]. Graph states take their name from the concept of graphs in mathematics: Each qubit corresponds to a vertex of the graph, and the graph's edges indicate which qubits have interacted (see below for details).

There is an intimate correspondence between stabilizer states (the class of states that can appear in a stabilizer circuit) and graph states: Not only is every graph state a stabilizer state, but also every stabilizer state is equivalent to a graph state in the following sense: Any stabilizer state can be transformed to a graph state by applying a tensor product of local Clifford (LC) operations [17–19]. We shall call these local Clifford operators the *vertex operators* (VOp's).

To represent a stabilizer state in computer memory, one stores its tableau of stabilizer operators, which is an $N \times N$ matrix of Pauli operators and hence takes space of order $O(N^2)$ (see below for details). Gottesman and Aaronson's simulator extends this matrix by another matrix of the same size (which they call the destabilizer tableau), so that their simulator has space complexity $O(N^2)$. A graph state, on the other hand, is described by a mathematical graph, which, for reasons argued later, only needs space of $O(N \log N)$ in typical applications. Hence, much larger systems can be represented in memory if one describes them as graph states supplemented with the list of VOp's. However, we also need efficient ways to calculate how this representation changes, when the represented state is measured or undergoes a Clifford-gate application. The effect of measurements has been extensively studied in [20], and gate application is what we will study in this paper, so that we can then assemble both to a simulation algorithm.

This paper is organized as follows: We first review the stabilizer formalism, the Gottesman-Knill theorem, and the graph-state formalism in Sec. II. There, we will also explain our representation in detail. Section III explains how the state representation changes when Clifford gates are applied. This is the main result and the most technical part of the paper. For the simulation of measurements, we can rely on the studies of Ref. [20], which are reviewed and applied for

*Electronic address: sanders@fs.tum.de

our purpose in Sec. IV. Having exposed all parts of the simulator algorithm, we continue by presenting our implementation of it. A reader who only wishes to use our simulator and is not interested in its internal working may want to read only this section. Section VI assesses the time requirements of the algorithm's components described in Secs. III and IV in order to prove our claim of superior scaling of performance. We finish with a conclusion (Sec. VII).

II. STABILIZER AND GRAPH STATES

We start by explaining the concepts mentioned in the introduction in a formal manner.

Definition 1. The Clifford group $\mathcal{C}_N$ on N qubits is defined as the normalizer of the Pauli group $\mathcal{P}_N$:

$$\mathcal{C}_N = \{U \in \mathrm{SU}(2^N) | UPU^\dagger \in \mathcal{P}_N \quad \forall \ P \in \mathcal{P}_N\},$$

$$\mathcal{P}_N = \{\pm 1, \pm i\}\{I,X,Y,Z\}^{\otimes N}, \tag{1}$$

where I is the identity and X, Y, and Z are the usual Pauli matrices.

The Clifford group can be generated by three elementary gates (see, e.g., [2]): the Hadamard gate H, the $\pi/4$ phase rotation S, and a two-qubit gate, either the controlled NOT (CNOT) gate ΛX or the controlled phase gate ΛZ:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix},$$

$$\Lambda X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \Lambda Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{2}$$

The significance of the Clifford group is due to the Gottesman-Knill theorem ([1]; see also [2]).

Theorem 1. A quantum circuit using only the following elements (called a stabilizer circuit) can be simulated efficiently on a classical computer: (a) preparation of qubits in computational basis states, (b) quantum gates from the Clifford group, and (iii) measurements in the computational basis.

The proof of the theorem is simple after one introduces the notion of stabilizer states [21].

Definition 2. An N-qubit state $|\psi\rangle$ is called a stabilizer state if it is the unique eigenstate with eigenvalue +1 of N commuting multilocal Pauli operators P_a (called the stabilizer generators):

$$P_a|\psi\rangle = |\psi\rangle, \quad P_a \in \mathcal{P}_N, \quad a = 1,\ldots,N.$$

(These N operators generate an Abelian group, the *stabilizer*, of 2^N Pauli operators that all satisfy this stabilization equation.)

Computational basis states are stabilizer states. Furthermore, if a *Clifford* gate U acts on a stabilizer state $|\psi\rangle$, the new state $U|\psi\rangle$ is a stabilizer state with generators $UP_aU^\dagger \in \mathcal{P}_N$. Hence, the state in a stabilizer circuit can always be described by the *stabilizer tableau*, which is a matrix of $N \times N$ operators from $\{I,X,Y,Z\}$ (where each row is preceded by a sign factor). The effect of an n-qubit gate can then be determined by updating nN elements of the matrix, which is an efficient procedure.

Instead of on the stabilizer tableau, we shall base our state representation on graph states.

Definition 3. An N-qubit graph state $|G\rangle$ is a quantum state associated with a mathematical graph $G=(V,E)$, whose $|V| = N$ vertices correspond to the N qubits, while the edges E describe quantum correlations, in the sense that $|G\rangle$ is the unique state satisfying the N eigenvalue equations

$$K_G^{(a)}|G\rangle = |G\rangle, \quad a \in V,$$
$$\text{with } K_G^{(a)} = \sigma_x^{(a)} \prod_{b \in \mathrm{ngbh}\ a} \sigma_z^{(b)} =: X_a \prod_{b \in \mathrm{ngbh}\ a} Z_b, \tag{3}$$

where ngbh $a := \{b | \{a,b\} \in E\}$ is the set of vertices adjacent to a [14–16].

The following theorem states that the edges of the graph can be associated with phase gate interactions between the corresponding qubits.

Theorem 2. If one starts with the state $|+\rangle^{\otimes N} = \Pi_{a \in V} H_a |00\ldots0\rangle$, one can easily construct $|G\rangle$ by applying ΛZ on all pairs of neighboring qubits:

$$|G\rangle = \left(\prod_{\{a,b\} \in E} \Lambda Z_{ab}\right)\left(\prod_{a \in V} H_a\right)|0\rangle^{\otimes N}. \tag{4}$$

[Proof: insert Eq. (4) into Eq. (3) [20].]

As the operators $K_G^{(a)}$ belong to the Pauli group, all graph states are stabilizer states, and so are the states which we get by applying *local* Clifford operators $C \in \mathcal{C}_1$ to $|G\rangle$. For such states, we introduce the notation

$$|G;\underline{C}\rangle := |G;C_1,C_2,\ldots,C_N\rangle := \bigotimes_{i=1}^{N} C_i|G\rangle. \tag{5}$$

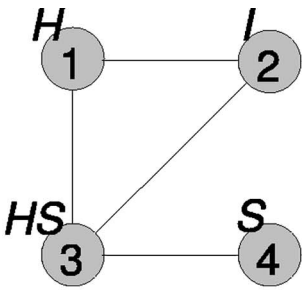
It has been shown that all stabilizer states can be brought into this form [17–19]; i.e., any stabilizer state is *LC* equivalent to a graph state. (We call two states *LC* equivalent if one can be transformed into the other by applying a tensor product of local Clifford operators.) Finding the graph state that is *LC* equivalent to a stabilizer state given by a tableau can be done by a sort of Gaussian elimination as explained in [17].

This is what we shall use to represent the current quantum state in the memory of our simulator. Figure 1 shows for an example state the tableau representation that is usually employed (and also used by Aaronson and Goldesman's CHP program, albeit in a modified form) and our representation. The tableau representation requires space of order $O(N^2)$. We store the graph in adjacency list form (i.e., for each vertex, a list of its neighbors is stored), which needs space of order $O(N\bar{d})$, where $\bar{d}$ is the average vertex degree (number of neighbors) in the graph. We also store a list of the N local Clifford operators $C_1,\ldots,C_N$, which transform the graph state $|G\rangle$ into the stabilizer state $|G;\underline{C}\rangle$. We call these operators the vertex operators. As there are only 24 elements in the

(a)

	1	2	3	4
+	Z	Z	X	I
+	X	X	X	I
−	X	Z	Y	Z
+	I	I	X	Y

(b)



(c)

Vertex	VOP	adjacency list
1	10	2, 3
2	0	1, 3
3	17	1, 2, 4
4	6	3

FIG. 1. A stabilizer state $|\psi\rangle$ represented in different ways: (a) as stabilizer tableau; i.e., the state is stabilized by the group of Pauli operators generated by the operators in the four rows. This representation needs space $O(N^2)$ for N qubits. (b), (c) as LC equivalence to a graph state. (b) shows the graph, with the VOP's given by their decomposition into the group generators $\{H,S\}$. (c) is the data structure that represents (b) in our algorithm. The VOP's are now specified using numbers between 0 and 23 (which enumerate the $|\mathcal{C}_1|=24$ LC operators). Here, we need space $O(N\bar{d})$, where $\bar{d}$ is the average vertex degree—i.e., the average length of the adjacency lists. Writing G for the graph in (b), we can use the notation of Eq. (5) and write $|\psi\rangle = |G;H,I,HS,S\rangle$.

local Clifford group, each VOP is represented as a number in 0,…, 23. The scheme to enumerate the 24 operators will be described in [22]. Note that we can disregard global phases of the VOP's as they only lead to a global phase of the full state of the simulator.

As we shall see later, we may typically assume that $\bar{d} = O(\log N)$. Hence, our representation needs considerably less space in memory than a tableau—namely, $O(N \log N)$, including $O(N)$ for the VOP list.

The Gaussian elimination needed to transform a stabilizer tableau to its graph-state representation is slow [time complexity $O(N^3)$], and so we should better not use it in our simulator. But usually, one starts with the initial state $|0\rangle^{\otimes N}$, and if we write this state already in graph-state form, the tableau representation is never used at all.

From Eq. (4), it is clear that the initial state can be written as a graph with no edges and Hadamard gates acting on all vertices:

$$|0\rangle^{\otimes N} = |(\{1,\ldots,N\},\{\});H,\ldots,H\rangle.$$

III. GATES

When the simulator is asked to simulate a Clifford gate, the current stabilizer state is changed and its graph representation has to be updated to correctly reflect the action of the gate. How to do this is the main technical result of this paper.

A. Single-qubit gates

In the graph representation, applying local (single-qubit) Clifford gates becomes trivial: if $C \in \mathcal{C}_1$ is applied to qubit a, we replace this qubit's VOP C_a by CC_a.

B. Two-qubit gates

It is sufficient if the simulator is capable of simulating a single multi-qubit gate: As the entire Clifford group is generated, e.g., by H, S, and ΛZ, all gates can be constructed by concatenating these. We chose to implement ΛZ, the phase gate, as this is [because of its role in Eq. (4)] most natural for the graph-state formalism.

In the following discussion, the two qubits onto which the phase gate acts are called the *operand vertices* and denoted with a and b. All other qubits are called *nonoperand vertices* and denoted c, d,… .

To solve the task, we have to distinguish several cases.

Case 1. The VOP's of both operand vertices are in $\mathcal{Z}$, where $\mathcal{Z} := \{I,Z,S,S^\dagger\}$ denotes the set of those four local Clifford operators that commute with ΛZ (the other 20 operators do not). In this case, applying the phase gate is simple: We use the fact that [due to Eq. (4)] applying a phase gate on a graph state just toggles an edge:

$$\Lambda Z_{ab}|(V,E)\rangle = |(V,E \triangle \{\{a,b\}\})\rangle,$$

where $\triangle$ denotes the symmetric set difference $A \triangle B := (A \cup B)\backslash(A \cap B)$; i.e., the edge $\{a,b\}$ is added to the graph if is was not present before; otherwise, it is removed.

Case 2. The VOP of at least one of the operand vertices is not in $\mathcal{Z}$. In this case, just toggling the edge is not allowed because the ΛZ_{ab} cannot be moved past the non-$\mathcal{Z}$ VOP. But there is a way to change the VOP's without changing the state, which works in the following case.

Case 2.2. Both operand vertices have nonoperand neighbors. Here, the following operation will help.

Definition 4. The operation of local complementation about a vertex a of a graph $G=(V,E)$, denoted L_a, is the operation that inverts the subgraph induced by the neighborhood of v:

$$L_a(V,E) = (V, E \triangle \{\{b,c\} | b,c \in \text{ngbh } a\}).$$

This operation transforms the state into a local-Clifford equivalent one, as the following theorem, taken from [17,20], asserts.

Theorem 3. Applying the local complementation L_a onto a graph G yields a state $|L_aG\rangle = U|G\rangle$, with the multilocal unitary

$$U = \sqrt{-iX_a} \prod_{b \in \text{ngbh } a} \sqrt{iZ_b} \propto \sqrt{K_G^{(a)}}.$$

Note that the operator $\sqrt{iZ}$ is related to the phase operator S of Eq. (2): $\sqrt{iZ} = e^{i\pi/4} S^\dagger$ and

$$\sqrt{iX} = \sqrt{-iX}^\dagger = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}.$$

An obvious consequence of theorem 3 is the following.

Corollary 1. A state $|G;\underline{C}\rangle$ is invariant under application of L_a to G, followed by an updating of C according to

$$C_b \mapsto \begin{cases} C_b\sqrt{iX} & \text{for } b = a, \\ C_b\sqrt{-iZ} & \text{for } b \in \text{ngbh } a, \\ C_b & \text{otherwise.} \end{cases} \quad (6)$$

Now note that the local Clifford group is generated not only by S and H but also by $\sqrt{-iX}$ and $\sqrt{iZ}$, the Hermitian adjoints of the operators right multiplied by the VOP's in Eq. (6). Our simulator has a look-up table that spells out every local Clifford operator as a product of—as it turns out, at most five—of these two operators times a disregarded global phase. For example, the table's line for H reads

$$H \propto \sqrt{-iX}\sqrt{iZ}\sqrt{iZ}\sqrt{iZ}\sqrt{-iX}. \quad (7)$$

This allows us now to reduce the VOP C_a of any *nonisolated* vertex a to the identity I by proceeding as follows: The decomposition of C_a taken from the look-up table is read from right to left. When a factor $\sqrt{-iX}$ is read we do a local complementation about a. This does not change the state if the correction of Eq. (6) is applied, which right-multiplies a factor $\sqrt{iX}$ to C_a. This factor $\sqrt{iX}$ cancels with the factor $\sqrt{-iX}$ at the right-hand end of C_a's decomposition, so that we now have a VOP with a shorter decomposition.

If the rightmost operator of the decomposition is $\sqrt{iZ}$, we do a local complementation about an arbitrarily chosen neighbor of a, called a's "swapping partner." Now, the correction operation will lead to a factor S being right multiplied by C_a, again shortening the decomposition.

Note that a local complementation about a never changes the edges incident on a and hence, if a was nonisolated in the beginning of the procedure, it will stay so. This is important, as only a nonisolated vertex can have a swapping partner. Hence, the procedure can be iterated, and (as the decompositions have a maximum length of 5) after at most five iterations, we are left with the identity I as VOP.

We apply the described "VOP reduction procedure" to both operand vertices. After that, both vertices are the identity, and we can proceed as in case 1.

One might wonder, however, whether the use of the VOP reduction procedure on the second operand vertex b spoils the reduction of the VOP of the first operand a. After all, a could be a neighbor of b or of the swapping partner c of b. Then, if a local complementation L_b or L_c is performed, the compensation according to Eq. (6) changes the neighborhood of b and c (which include a). But note that a neighbor of the inversion center only gets a factor $\sqrt{-iZ} \propto S^\dagger$. As $S^\dagger$ generates $\mathcal{Z}$, this means that after the reduction of b, the VOP of a might be no longer the identity but it is still an element of $\mathcal{Z}$, and we are allowed to go on with case 1.

But what happens if one of the vertices does not have a nonoperand neighbor that could serve as swapping partner? This is the next case.

Case 2.2. At least one of the operand vertices is isolated or only connected to the other operand vertex. We first assume that the other vertex is nonconnected in the same sense.

Case 2.2.1. Both operand vertices are either completely isolated or only connected with each other. Then, we can ignore all other vertices and have to study only a finite, rather small number of possible states.

Let us denote by • • the two-vertex graph with no edges and by •—• the two-vertex graph with one edge. There are only very few possible two-qubit stabilizer states: namely, those in

$$\mathcal{S}_2 := \{|G;C_1,C_2\rangle | G \in \{\bullet\ \bullet, \bullet - \bullet\}, C_1, C_2 \in \mathcal{C}_1\}. \quad (8)$$

Of course, many of the assignments on the right-hand side describe the same state, such that $|\mathcal{S}_2| < 2 \times 24^2$. Remember that the phase gate $\Lambda Z_{1,2}$ (being a Clifford operator) maps $\mathcal{S}_2$ bijectively onto itself.

The function table of $\Lambda Z_{1,2}|_{\mathcal{S}_2}: |G;C_1,C_2\rangle \mapsto |G';C_1',C_2'\rangle$ can easily be computed in advance (we did it with Mathematica) and hard coded into the simulator as a look-up table. This table contains 2×24^2 lines such as

$$|\bullet\ \bullet, C_{[13]}, C_{[2]}\rangle \mapsto |\bullet - \bullet, C_{[0]}, C_{[2]}\rangle, \quad (9)$$

where the $C_{[i]} (i=0,\ldots,23)$ are the Clifford operators in the enumeration detailed in [22] (e.g., $C_{[0]}=I$, $C_{[2]}=Y$).

Note that many of the assignments to C_1 and C_2 in Eq. (8) describe the same state. Hence, we have a choice in the operators C_1', C_2' with which we represent the results of the phase gate in the look-up table. It turns out (by inspection of all the possibilities) that we can always choose the operators such that the following constraint is fulfilled.

Constraint 1. If $C_1(C_2) \in \mathcal{Z}$, choose C_1', C_2' such that again $C_1'(C_2') \in \mathcal{Z}$.

The use of this will become clear soon.

Case 2.2.2. We are left with one last case—namely, that one vertex, let it be a, is connected with nonoperand neighbors, but the other vertex b is not—i.e., has either no neighbors or only a as neighbor. Then, we proceed as follows: We use iterated local complementations to reduce C_a to I. After that, we may use the look-up table as in case 2.2.1. That this is allowed even though a is connected to a nonoperand vertex is shown in the following: First note that the state after the reduction of C_a to I can be written [following Eq. (5)]= as

$$|(V,E);\underline{C}\rangle = \prod_{c\in V} C_c \prod_{\{c,d\}\in E} \Lambda Z_{cd} |+\,+\cdots+\rangle = \underbrace{\prod_{\substack{c\in\\ V\setminus\{a,b\}}} C_c \prod_{\substack{\{c,d\}\in\\ E\setminus\{\{a,b\}\}}} \Lambda Z_{cd}}_{\substack{C_b \text{ and } \Lambda Z_{ab}\\ \text{commute with this}}} \underbrace{C_b(\Lambda Z_{ab})^{\zeta}|+\,+\cdots+\rangle}_{\substack{=|+\rangle^{\otimes N-2}\otimes|\varphi\rangle_{ab}\\ \text{with } |\varphi\rangle\in\mathcal{S}_2\\ (*)}}$$

(where $\zeta=0,1$ indicates whether $\{a,b\}\in E$). Observe that C_b has been moved past the operators ΛZ_{cd}. This is allowed because none of the ΛZ_{cd} acts on b.

We now apply ΛZ_{ab} to this state. ΛZ_{ab} can be moved through all the phase gates and vertex operators above the left brace so that it stands right in front of the $\mathcal{S}_2$ state $|\varphi\rangle_{ab}$ which is separated from the rest. Thus, the table (9) from case 2.2.1 may be used. [This would not be the case if, in the state above the brace marked with (*), the two operand vertices were still entangled with other qubits.] The look-up in the table will give new operators C_a', C_b' and a new ζ', so that the new state has the following form:

$$\Lambda Z_{ab}|(V,E);\underline{C}\rangle = \prod_{c\in V\setminus\{a,b\}} C_c \prod_{\{c,d\}\in E\setminus\{\{a,b\}\}} \Lambda Z_{cd} \times C_a' C_b' (\Lambda Z_{ab})^{\zeta'} |+\,+\cdots+\rangle. \quad (10)$$

For this to be a state in our usual $|G;\underline{C}\rangle$ form (5), the two operators C_a' and C_b' have to moved to the left, through the ΛZ_{cd}. For C_b', this is no problem, as b was assumed to be either isolated or connected only to a, so that C_b' commutes with $\Pi_{\{c,d\}\in E\setminus\{\{a,b\}\}}\Lambda Z_{cd}$, as the latter operator does not act on b. The vertex a, however, has connections to nonoperand neighbors, so that some of the ΛZ_{cd} act on it. We may move it only if $C_a'\in\mathcal{Z}$ (as this means that it commutes with ΛZ). Luckily, due to constraint 1 imposed above, we can be sure that $C_a'\in\mathcal{Z}$, because $C_a=I\in\mathcal{Z}$.

Figure 2 shows as listing in pseudocode how these results can be used to actually implement the controlled phase gate ΛZ.

IV. MEASUREMENTS

In a stabilizer circuit, the simulator may be asked at any point to simulate the measurement of a qubit in the computational basis. How the outcome of the measurement is determined and how the graph representation has to be updated in order to then represent the post-measurement state will be explained in the following.

To measure a qubit a of a state $|G,\underline{C}\rangle$ in the computational basis means to measure the qubit in the underlying graph state $|G\rangle$ in one of the three Pauli bases. Writing the measurement outcome as ζ, this means

$$\frac{I+(-1)^{\zeta}Z_a}{2}|G,\underline{C}\rangle = \left(\prod_{b\in V\setminus\{a\}} C_b\right)\frac{I+(-1)^{\zeta}Z_a}{2}C_a|G\rangle = \left(\prod_{b\in V\setminus\{a\}} C_b\right)C_a\frac{I+(-1)^{\zeta}C_a^{\dagger}Z_aC_a}{2}|G\rangle. \quad (11)$$

As C_a is a Clifford operator, $P_a := C_a^{\dagger}Z_aC_a \in \{X_a,Y_a,Z_a,-X_a,-Y_a,-Z_a\}$. Thus, in order to measure qubit a of $|G,\underline{C}\rangle$ in the *computational* basis, we measure the observable P_a on $|G\rangle$. Note that in case that P_a is the negative of a Pauli operator, the measurement result ζ to be reported by the simulator is the complement of $\tilde{\zeta}$, the result given by the X, Y, or Z measurement on the underlying graph state $|G\rangle$.

How is the graph G changed and how do the vertex operators have to be modified if the measurement $[(I\pm P_a)/2]|G\rangle$ is carried out? This has been worked out in detail in Ref. [20], which we now briefly review for the present purpose.

The simplest case is that of $P=\pm Z$. Here, the state changes as follows:

$$\frac{I+(-1)^{\tilde{\zeta}}Z_a}{2}|(V,E)\rangle = \underbrace{\left(X_a\prod_{b\in \mathrm{ngbh}a} Z_b\right)^{\tilde{\zeta}}H_a}_{(*)} \quad |(V,E\setminus\{\{a,b\}|b\in \mathrm{ngbh}a\})\rangle. \quad (12)$$

```
1  cphase (vertex a, vertex b):
2      if ngbh a\{b} ≠ {}:
3          remove_VOP (a, b)
4      end if
5      if ngbh b\{a} ≠ {}:
6          remove_VOP (b, a)
7      end if
8      [It may happen that the condition in line 2
       has not been fulfilled then, but is now due to
       the effect of line 5. So we check again:]
9      if ngbh a\{b} ≠ {}:
10         remove_VOP (a, b)
11     end if
12     [Now we can be sure that the the condition
       (ngbh c\{a, b} = {} or VOP[c] ∈ Z) is fulfilled
       for c = a, b and we may use the lookup table
       (cf. Eq. (9)).]
13     if {a, b} ∈ E :
14         edge ← true
15     else:
16         edge ← false
17     end if
18     (edge, VOP[a], VOP[b]) ←
       cphase_table[edge, VOP[a], VOP[b]]
19
20 remove_VOP (vertex a, vertex b):
21 [This reduces VOP[a] to I, avoiding (if possible) to
   use b as swapping partner.]
22     [First, we choose a swapping partner c.]
23     if ngbh a\{b} ≠ {}:
24         c ← any element of ngbh a\{b}
25     else:
26         c ← b
27     end if
28     d ← decomposition_lookup_table [a]
29     [c contains now a decomposition such as Eq.
       (7)]
30     for v from last factor of d to first factor of d
31         if v = √(−iX):
32             local_complementation (a)
33         else: (this means that v = √(iZ))
34             local_complementation (b)
35         end if
36     [Now, VOP[a] = I.]
37
38 local_complementation (vertex a)
39 [performs the operation specified in Definition 4]
40     n_v ← ngbh v
41     for i ∈ n_v:
42         for j ∈ n_v:
43             if i < j:
44                 if (i, j) ∈ E:
45                     remove edge (i, j)
46                 else:
47                     add edge (i, j)
48                 end if
49             end if
50         end for
51         VOP[i] ← VOP[i]√(−iZ)
52     VOP[v] ← VOP[v]√(iX)
53     end for
```

FIG. 2. Pseudocode for controlled phase gate (ΛZ) acting on vertices a and b ("cphase") and for the two auxiliary routines "remove VOP" and "local complementation".

The value of $\tilde{\zeta}$ is chosen at random (using a pseudo-random-number generator). To update the simulator state, the VOP's are right multiplied by the underbraced operators (*) and the edges incident on a are deleted as indicated in the ket.

A measurement of the Y observable ($P=\pm Y$) requires a complementation of the edges set according to

$$E \mapsto E \triangle \{\{b,c\}|b,c \in \mathrm{ngbh}a\}$$

and a change in the VOP's as follows:

$$C_b \mapsto C_b\sqrt{-iZ}^{(\dagger)} \text{ for } b \in \mathrm{ngbh}a \cup \{a\},$$

where the dagger in parentheses is to be read only for measurement result $\tilde{\zeta}=1$.

The most complicated case is the X measurement which requires an update of edges and VOP's as follows:

$$\begin{aligned} E \mapsto E &\triangle \{\{c,d\}|c \in \mathrm{ngbh}b, d \in \mathrm{ngbh}a\} \\ &\triangle \{\{c,d\}|c,d \in \mathrm{ngbh}b \cap \mathrm{ngbh}a\} \\ &\triangle \{\{b,d\}|d \in \mathrm{ngbh}a \setminus \{b\}\}, \end{aligned}$$

$$C_c \mapsto \begin{cases} C_c Z^{\tilde{\zeta}} & \text{for } c = a, \\ C_c\sqrt{iY}^{(\dagger)} & \text{for } c = b \text{ (read `` † '' only for } \tilde{\zeta}=1), \\ C_c Z & \text{for } c \in \begin{cases} \mathrm{ngbh}\ a \setminus \mathrm{ngbh}\ b \setminus \{b\} \\ \qquad (\text{for } \tilde{\zeta}=0), \\ \mathrm{ngbh}\ b \setminus \mathrm{ngbh}\ a \setminus \{a\} \\ \qquad (\text{for } \tilde{\zeta}=1), \end{cases} \\ C_c & \text{otherwise.} \end{cases} \tag{13}$$

Here, b is a vertex chosen arbitrarily from ngbh a and

$$\sqrt{iY} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}.$$

In all these cases the measurement result is chosen at random. Only in the case of the measurement of $P_a=\pm X$ an isolated vertexis the result always $\tilde{\zeta}=0$ (which means an actual result of $\zeta=0$ for $P_a=X$ and $\zeta=1$ for $P_a=-X$).

V. IMPLEMENTATION

The algorithm described above has been implemented in C++ in object-oriented programming style. We have used the GNU Compiler Collection (GCC) [24] under Linux, but it should be easy to compile the program on other platforms as well [28]. The implementation is done as a library to allow for easy integration into other projects. We also offer bindings to PYTHON [25], so that the library can be used by PYTHON programs as well. (This was achieved using SWIG [26].)

The simulator, called "GRAPHSIM," can be downloaded from [23].

A detailed documentation of the library is supplied with it. To demonstrate the usage here at least briefly, we give Fig. 3

```
1  import random
2  import graphsim
3
4  gr = graphsim.GraphRegister(8)
5
6  gr.hadamard(4)
7  gr.hadamard(5)
8  gr.hadamard(6)
9  gr.cnot(6, 3)
10 gr.cnot(6, 1)
11 gr.cnot(6, 0)
12 gr.cnot(5, 3)
13 gr.cnot(5, 2)
14 gr.cnot(5, 0)
15 gr.cnot(4, 3)
16 gr.cnot(4, 2)
17 gr.cnot(4, 1)
18
19 for i in xrange(7):
20    gr.cnot(i, 7)
21
22 print gr.measure(7)
23
24 gr.print_stabilizer()
```

FIG. 3. A simple example in PYTHON.

as a simple toy example. It is written in PYTHON and a complete program.

In the example, we start by loading the GraphSim library (line 2) and then initialize a register of eight qubits (line 4), which are then all in $|0\rangle$ state. We get an object called "gr" of class GraphRegister, which represents the register of qubits. For all following operations, we use the methods of gr to access its functionality. In our example, we simply build up an encoded "0" state in the well-known seven-qubit Steane code, which we then measure.

First, we apply Hadamard and CNOT gates onto the qubits with numbers 0–6 in order to build up the Steane-encoded "0" (lines 6–17). To check that we did so, we measure the encoded qubit, which is done by using CNOT gates to sum up their parity in the eighth qubit ("qubit 7") (lines 19, 20). Measuring qubit 7 then gives "0," as it should (line 22).

For further details on using of the GRAPHSIM library from a C++ or PYTHON program, see the documentation supplied with the source code [23].

With approximately 1400 lines, GRAPHSIM is complex enough that one cannot take for granted that it faithfully implements the described algorithm without bugs, and testing is necessary. Fortunately, this can be done very conveniently by comparing with Aaronson and Gottesman's "CHP" simulator. As these two programs use quite different algorithms to do the same task, it is very unlikely that any bugs, which they might have, produce the *same* false results. Hence, if both programs give the same result, they can reasonably be considered both to be correct.

We set up a script to do random gates and measurements on a set of qubits for millions of iterations. All operations were performed simultaneously with CHP and GRAPHSIM. For measurements whose outcome was chosen at random by CHP, a facility of GRAPHSIM was used that overrides the random choice of measurement outcomes and instead uses a supplied value. For measurements with determined outcome, however, it was checked whether both programs output the same result. Also, every 1000 steps, the stabilizer tableau of GRAPHSIM's state was calculated from its graph representation and compared to CHP's tableau [29].

After simulation 4×10^6 operations on 200 qubits in 18 h and 2×10^8 operations on 20 qubits in 19.7 h without seeing discrepancies, we are confident that we have exhausted all special cases, so that the two programs can be assumed to always give the same output. As they are based on very different algorithm, this reasonably allows to conclude that they both operate correctly.

VI. PERFORMANCE

We now show that our simulator yields the promised performance—i.e., performs a simulation of M steps in time of order $O(NdM)$, where N is the number of qubits and d the maximum vertex degree that is encountered during the calculation. Let us go through the different possible simulation steps in order to assess their respective time requirements.

Single-qubit gates are the fastest: they only need one look-up in the multiplication table of the local Clifford group (which is hard coded into the simulator) and are hence of time complexity $\Theta(1)$.

Measurements have a complexity depending on the basis in which they have to be carried out. For a Z measurement, we have to remove the deg a edges of the measured vertex a. As d is the maximum vertex degree that is to be expected within the studied problem, the complexity of a Z measurement is $O(d)\leqslant O(N)$ (as $d\leqslant N$).

For a Y and X measurement, we have to do local complementation, which requires dealing with up to $d(d-1)/2$ edges, and hence, the overall complexity of measurements is $O(d^2)$.

For the phase gate, the same holds. Here, we need a fixed number (up to 5) of local complementations. Thus, measurements and two-qubit gates take $O(d^2)$ time.

This would be no improvement to Aaronson and Gottesman's algorithm if we had $d=O(N)$. The latter is indeed the case if one applies randomly chosen operations as we did to demonstrate GRAPHSIM's correctness. There, we indeed did not observe any superiority in the run time of GRAPHSIM.

In practice, however, this is quite different. For example, when simulating quantum error correction, one can reasonable assume $d=O(\log N)$. This is because all quantum error correction (QEC) schemes avoid doing many operations on one and the same qubit in a row, as this would spread errors. So vertex degrees remain small. The same reasoning applies to entanglement purification schemes and, more generally, to all circuits which are designed to be robust against noise.

The space complexity is dominated by the space needed to store the quantum-state representation. As argued in Sec. II, this requires only space of $O(N\bar{d})$, where $\bar{d}$ is the average vertex degree. As explained above, we may expect $\bar{d}$ (as d) to scale sublinearly with N in typical applications, in many applications as $O(N\ln N)$. This is what allows us to handle

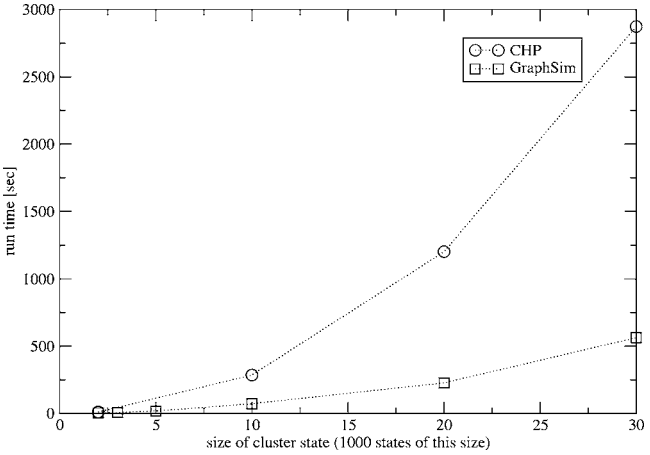


FIG. 4. Comparison of the performance of CHP and GRAPHSIM. A simulation of entanglement purification was used as sample application. The register has 1000 times the size of the states to hold an ensemble of 1000 states.

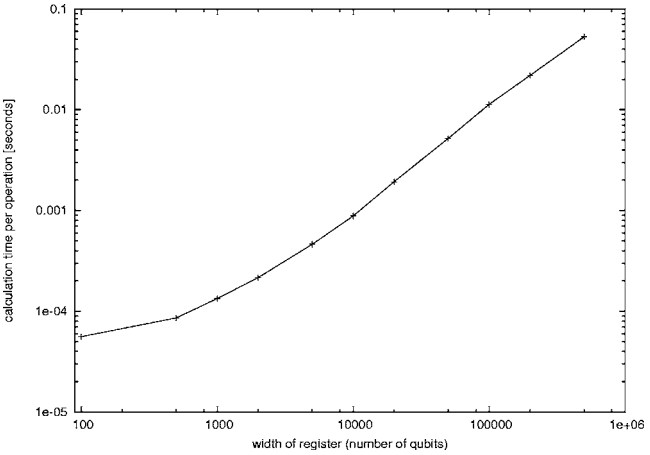


FIG. 5. Benchmark of GRAPHSIM for very large registers. Entanglement purification—specifically, the purification of ten-qubit cluster states with the protocol of Ref. [7]—was used as sample problem. The register was filled up with cluster states to make a large ensemble, and two protocol steps were simulated. The average time per operation was obtained from the total run time [27].

substantially more qubits than is possible with the $O(N^2)$ tableau representation.

As a first practical test, we used GRAPHSIM to simulate entanglement purification of cluster states with the protocol of Ref. [7]. This has been a starting point of a detailed analysis of the communication costs of establishing multipartite entanglement states via noisy channels [13]. Figure 4 demonstrates that GRAPHSIM is indeed suitable for this purpose. Note that for the rightmost data points, the register holds 30 000 qubits.

As we did a Monte Carlo simulation, we had to loop the calculation very often and still got an output within a few hours. For simulations involving several millions of qubits and a large number of runs, we waited about a week for the results when using eight processors in parallel. We redid some of these calculations in a more controlled testing environment as a benchmark for GRAPHSIM. Figure 5 shows the results in a log-log plot.

VII. CONCLUSION

To summarize, we have used recent results on graph states to find a very space-efficient representation of stabilizer states and determined how this representation changes under the action of Clifford gates. This can be used to simulate stabilizer circuits more efficiently than previously possible. The gain is not only in simulation speed, but also in the number of manageable qubits. In the latter, at least two orders of magnitude are gained. We have presented an implementation of our simulation algorithm and will soon publish results about entanglement purification which makes use of our technique.

ACKNOWLEDGMENTS

We would like to thank Marc Hein for most helpful discussions. This work was supported in part by the Austrian Science Foundation (FWF), the Deutsche Forschungsgemeinschaft (DFG), and the European Union (Grant Nos. IST-2001-38877 and IST-2001-39227, OLAQUI, SCALA).

[1] D. Gottesman, e-print quant-ph/9807006.
[2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, England, 2000).
[3] C. H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J. A. Smolin, and W. K. Wootters, Phys. Rev. Lett. **76**, 722 (1996).
[4] D. Deutsch, A. Ekert, R. Jozsa, C. Macchiavello, S. Popescu, and A. Sanpera, Phys. Rev. Lett. **77**, 2818 (1996).
[5] M. Murao, M. B. Plenio, S. Popescu, V. Vedral, and P. L. Knight, Phys. Rev. A **57**, R4075 (1998).
[6] E. N. Maneva and J. A. Smolin, in *Quantum Computation and Quantum Information*, edited by J. S. J. Lomonaco (AMS, Providence, RI, 2002); also e-print quant-ph/0003099.
[7] W. Dür, H. Aschauer, and H. J. Briegel, Phys. Rev. Lett. **91**, 107903 (2003).
[8] P. W. Shor, Phys. Rev. A **52**, R2493 (1995).
[9] A. M. Steane, Phys. Rev. Lett. **77**, 793 (1996).
[10] A. R. Calderbank and P. W. Shor, Phys. Rev. A **54**, 1098 (1996).
[11] A. M. Steane, Proc. R. Soc. London, Ser. A **452**, 2551 (1996).
[12] S. Aaronson and D. Gottesman, Phys. Rev. A **70**, 052328 (2004).
[13] C. Kruszynska, S. Anders, W. Dür, and H. J. Briegel, e-print quant-ph/0512218
[14] H. J. Briegel and R. Raußendorf, Phys. Rev. Lett. **86**, 910 (2001).

[15] R. Raußendorf, D. E. Browne, and H. J. Briegel, Phys. Rev. A **68**, 022312 (2003).

[16] D. Schlingemann and R. F. Werner, Phys. Rev. A **65**, 012308 (2002).

[17] M. Van den Nest, J. Dehaene, and B. De Moor, Phys. Rev. A **69**, 022316 (2004).

[18] M. Grassl, A. Klappenecker, and M. Rötteler, in *Proceedings of the 2002 IEEE International Symposium on Information Theory*, (IEEE, New York, 2002), p. 45.

[19] D. Schlingemann, e-print quant-ph/0111080.

[20] M. Hein, J. Eisert, and H. J. Briegel, Phys. Rev. A **69**, 062311 (2004).

[21] D. Gottesman, Ph.D. thesis, California Institute of Technology, 1997, e-print quant-ph/9705052.

[22] S. Anders (unpublished).

[23] The described software can be found at http://homepage.uibk.ac.at/homepage/c705/c705213/work/graphsim.html

[24] The GCC Team, The GNU Compiler Collection, software at http://gcc.gnu.org

[25] PYTHON, programming language developped by Guido van Rossum *et al.*, http://www.python.org

[26] SWIG *(simplified wrapper and interface generator)*, software developed by David M. Beazley *et al.*, http://www.swig.org

[27] Giving the time per operation in seconds is of use only when one specifies the machine which has run the code: We used Linux computers with AMD Opteron processors, clocked with 2.2 GHz. Only one the machine's several processors was dedicated to our computation task. The code was compiled using the GNU C++ compiler (version 3.2.3) with 64-bit target and "O3" optimization.

[28] We use only ISO Standard C++ with one exception: The hash set template is used, which is, though, not part of the standard, supplied by most modern compilers.

[29] This was done with a Mathematica subroutine which tries to find a row adding and swapping arrangement to transform one tableau into the other.