arXiv:2009.03218v3 [quant-ph] 31 Jan 2024

Fast simulation of planar Clifford circuits

David Gosset[1,2,3], Daniel Grier[1,4,5], Alex Kerzner[1,2], and Luke Schaeffer[1,2,6]

[1]Institute for Quantum Computing, University of Waterloo, Canada

[2]Department of Combinatorics and Optimization, University of Waterloo, Canada

[3]Perimeter Institute for Theoretical Physics, Waterloo, Canada

[4]Cheriton School of Computer Science, University of Waterloo, Canada

[5]Department of Computer Science and Engineering and Department of Mathematics, University of California, San Diego, US

[6]Joint Center for Quantum Information and Computer Science, College Park, Maryland, US

A general quantum circuit can be simulated classically in exponential time. If it has a planar layout, then a tensor-network contraction algorithm due to Markov and Shi has a runtime exponential in the square root of its size, or more generally exponential in the treewidth of the underlying graph. Separately, Gottesman and Knill showed that if all gates are restricted to be Clifford, then there is a polynomial time simulation. We combine these two ideas and show that treewidth and planarity can be exploited to improve Clifford circuit simulation. Our main result is a classical algorithm with runtime scaling asymptotically as $n^{\omega/2} < n^{1.19}$ which samples from the output distribution obtained by measuring all n qubits of a *planar* graph state in given Pauli bases. Here ω is the matrix multiplication exponent. We also provide a classical algorithm with the same asymptotic runtime which samples from the output distribution of any constant-depth Clifford circuit in a planar geometry. Our work improves known classical algorithms with cubic runtime.

A key ingredient is a mapping which, given a tree decomposition of some graph G, produces a Clifford circuit with a structure that mirrors the tree decomposition and which emulates measurement of the corresponding graph state. We provide a classical simulation of this circuit with the runtime stated above for planar graphs and otherwise $nt^{\omega-1}$ where t is the width of the tree decomposition. Our algorithm incorporates two subroutines which may be of independent interest. The first is a matrix-multiplication-time version of the Gottesman-Knill simulation of multi-qubit measurement on stabilizer states. The second is a new classical algorithm for solving symmetric linear systems over $\mathbb{F}_2$ in a planar geometry, extending previous works which only applied to non-singular linear systems in the analogous setting.

David Gosset: dngosset@gmail.com

Daniel Grier: dgrier@ucsd.edu

Alex Kerzner: alexkerzner1@gmail.com

Luke Schaeffer: lrschaeffer@gmail.com

1 Introduction and Overview

Enormous resources are now being marshalled by quantum information scientists towards the goal of building a quantum computer: demonstration of a computation for which there are no known efficient classical algorithms [1]; quantum machines that are accessible to the public [2]; and validation of certain building blocks of quantum error correction [3, 4, 5]. A by-product of the progress towards building quantum machines has been a renewed interest in the classical simulation of quantum computers [6, 7, 8]. This is partly due to the fact that a direct comparison with classical simulation algorithms is an important measure of the performance of today's quantum computers [9, 10, 11]. A quantum computer in the real world is limited to a specific architecture, which may restrict both the native gate set and the number of qubits while also being subjected to noise from its environment. While in some cases it is possible to leverage quantum advantage from small quantum devices with limited capabilities, one can also exploit these limitations in classical simulation algorithms. As a result, the search for a quantum advantage with near-term quantum computers is grounded in our understanding of the power of restricted models of quantum computing. Such models include constant-depth quantum circuits [12], IQP circuits [13], BosonSampling [14], circuits with few non-Clifford gates [15, 16, 8], and others. Their study reveals bright spots of quantum advantage even in the presence of harsh restrictions on the operation of quantum devices.

Here we focus on the restricted family of Clifford circuits, which are expressed as a product of gates from the set Hadamard, $S = \mathrm{diag}(1, i)$, and $\mathrm{CZ} = \mathrm{diag}(1, 1, 1, -1)$. Gottesman and Knill showed that Clifford circuits can be simulated in polynomial time on a classical computer [15]. The quantum state at any point during such a Clifford circuit is a so-called *stabilizer state* that can be specified using a tableau of $O(n^2)$ classical bits, see Section 2.1 for details. Updating this representation after the action of a H, S, or CZ gate can be performed by modifying $O(n)$ bits of the tableau in a simple way. Measurements are more costly: the best known algorithm for simulating the outcome of a single-qubit measurement uses $O(n^2)$ time [17].

Despite the fact that they can be simulated in polynomial time, it has been shown that quantum computations based on Clifford circuits still enjoy a certain type of quantum advantage [18, 19, 20, 21]. In this paper we consider a problem which is closely connected to the tasks considered in Refs. [18, 19, 20], which demonstrate a quantum advantage for certain special cases. Those tasks can be viewed as special cases of the problem of simulating measurements on a *graph state*. For any graph $G = (V, E)$ the graph state $|G\rangle$ is defined as the $n := |V|$ qubit state given by

$$|G\rangle = \left(\prod_{\{u,v\} \in E} \mathrm{CZ}_{uv} \right) H^{\otimes n} |0^n\rangle.$$

Equivalently, $|G\rangle$ is the n-qubit state satisfying

$$\mathtt{X}_v \left(\prod_{j:\{v,j\} \in E} \mathtt{Z}_j \right) |G\rangle = |G\rangle \qquad \forall v \in V,$$

where $\mathtt{X}_v$ and $\mathtt{Z}_v$ denote the Pauli $\mathtt{X}$ and $\mathtt{Z}$ operators acting nontrivially on qubit v.

In this paper, we will focus on the computational problem of simulating measurement of graph states, which (as described later) can be used as the basis for simulating more

general types of Clifford circuits. In the simplest version of the problem, we are given a graph $G = (V, E)$ with $n = |V|$ vertices and a single-qubit Pauli measurement basis $P_v \in \{\mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$ for each vertex $v \in V$, and we are asked to sample from the probability distribution obtained by measuring each qubit of $|G\rangle$ in the given bases. In particular, letting U_v be a single-qubit Clifford unitary that maps the P_v basis to the computational ($\mathtt{Z}$) basis (i.e., $U_v P_v U_v^\dagger = \mathtt{Z}$ for each $v \in V$), we are asked to sample a string $z \in \{0,1\}^n$ from the probability distribution

$$p(z) = |\langle z| \bigotimes_{v \in V} U_v |G\rangle|^2. \tag{1}$$

Notice that because $|G\rangle$ is prepared by a Clifford circuit and each U_v is also Clifford, this problem is an instance of Clifford circuit simulation. Conversely, it is also well known that simulation of general Clifford circuits can be reduced to this problem using, e.g., techniques from measurement-based quantum computing [22], where the size of the graph state will increase with the depth of the circuit. We refer the reader to Section 1.1 for an example of this reduction, as well as a non-exhaustive list of applications of the problem of measuring graph states in Pauli bases.

In the general version of the problem, we also allow some of the qubits to be postselected. That is, in addition to the graph and measurement bases, we are given a subset of qubits $\mathcal{P} \subseteq V$ and measurement outcomes $m \in \{0,1\}^{\mathcal{P}}$ as input. The problem is then to sample the remaining qubits $\mathcal{S} := V \backslash \mathcal{P}$ after having postselected on measurement outcomes m for the qubits of $\mathcal{P}$. In particular, let

$$p_{\mathcal{P}}(m) = \langle G| \bigotimes_{v \in V} U_v^\dagger |m\rangle\langle m|_{\mathcal{P}} \bigotimes_{v \in V} U_v |G\rangle \tag{2}$$

be the probability of obtaining outcomes m for the qubits in $\mathcal{P}$. If $p_{\mathcal{P}}(m)$ is nonzero, then the output is a binary string $x \in \{0,1\}^{|\mathcal{S}|}$ sampled from the conditional distribution $p_{\mathcal{S}}(x|m)$ over measurement outcomes in $\mathcal{S}$ after postselecting on outcomes m for the qubits in $\mathcal{P}$:

$$p_{\mathcal{S}}(x|m) = \frac{1}{p_{\mathcal{P}}(m)} |\langle x|_{\mathcal{S}} \langle m|_{\mathcal{P}} \bigotimes_{v \in V} U_v |G\rangle|^2. \tag{3}$$

Otherwise, the output is a flag indicating that $p_{\mathcal{P}}(m) = 0$, since it is meaningless to postselect on an impossible outcome. To summarize, we now state the problem in full.

Graph state simulation problem. *The input to the problem is a graph $G = (V, E)$ with $n = |V|$ vertices, a measurement basis $P_v \in \{\mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$ for each vertex $v \in V$, a partition $[n] = \mathcal{S} \cup \mathcal{P}$ of the vertices, and a binary string $m \in \{0,1\}^{|\mathcal{P}|}$. If the marginal probability $p_{\mathcal{P}}(m)$ from Eq. (2) is zero, the output is an error flag. Otherwise, the output is a binary string $x \in \{0,1\}^{|\mathcal{S}|}$ sampled from the conditional distribution $p_{\mathcal{S}}(x|m)$ defined in Eq. (3).*

We shall refer to the special case $\mathcal{P} = \varnothing$ as the graph state simulation problem *without postselection*. We note that although this special case without postselection is of particular interest, the classical algorithms we develop in this paper are capable of solving the more general graph state simulation problem with the same asymptotic runtime.

Let us now compare quantum and classical algorithms for the graph state simulation problem. When there is no postselection, a quantum computer can solve the problem by

preparing the graph state $|G\rangle$ and then measuring all qubits in the given bases. The total runtime of this procedure is linear in the size of the graph (number of edges plus number of vertices) and is therefore upper bounded as $O(n^2)$ in the general case, and as $O(n)$ if G is sparse. The graph state simulation problem can also be solved efficiently on a classical computer via the Gottesman-Knill theorem since all unitaries and measurements to be simulated are Clifford operations. To our knowledge the best previously-known algorithm for the graph state simulation problem is based on the standard Gottesman-Aaronson simulation method [17] and has runtime $O(n^3)$. Our first contribution is to reduce the graph state simulation problem to matrix multiplication, improving the runtime in the general case to $O(n^\omega)$ where $\omega < 2.373$ [23] is the matrix multiplication exponent.

Theorem 1. *There is a classical algorithm with runtime $O(n^\omega)$ which solves the graph state simulation problem.*[1]

The proof of Theorem 1 is obtained from a general algorithm for simulating multi-qubit measurements on stabilizer states which may be of independent interest, see Section 5 for details.

Guan and Regan [24] have recently established a similar result for the problem of computing output probabilities; they show that given a binary string $z \in \{0,1\}^n$, there is a classical algorithm which computes the output probability $p(z)$ in Eq. (1) in $O(n^\omega)$ time. However, the result of Guan and Regan does not provide a classical algorithm for the graph state simulation problem considered here. An important distinction is that the latter problem provides a compelling candidate for a quantum speedup. Indeed, Theorem 1 leaves open the possibility that quantum computers may have a polynomial speedup in terms of the total runtime (number of one- and two-qubit gates) required to solve the graph state simulation problem without postselection. For sparse graphs the comparison is more dramatic—the linear time quantum algorithm *which bears no resemblance to classical linear algebraic methods* beats Theorem 1 by a quadratic factor even if the matrix multiplication exponent is shown to meet its lower bound $\omega \geq 2$. Unfortunately we lack a suitable technique for proving that classical algorithms are more costly in terms of total runtime. On the other hand, if we turn our attention to other metrics such as circuit depth, Refs. [18, 20] prove that quantum computers beat classical machines in solving the graph state simulation problem even for the special case of a (subgraph of a) 2D grid graph. These works establish a quantum advantage in the following sense:

Quantum advantage for relation tasks ([18, 19])

Let G be a subgraph of the $\ell \times \ell$ grid, and consider a relaxation of the graph state simulation problem (without postselection) in which it suffices to output *any* string of measurement outcomes $z \in \{0,1\}^n$ that occurs with nonzero probability. (Equivalently, the problem can be expressed in terms of properties of binary quadratic forms, without reference to quantum graph states, see Ref.[18] for details). This problem can be trivially solved by a constant-depth quantum circuit since since the graph has constant degree. In contrast, it cannot be solved by any constant-depth classical circuit composed of unbounded fan-in AND, OR, and NOT gates (i.e., AC^0 circuits) [18, 19].

[1] As with many problems solved with fast matrix multiplication, we incur an extra $O(\log n)$ factor if $\omega = 2$. For full details, see the proof of Theorem 1 in Section 5.2.

Quantum advantage for interactive tasks ([20]) Simulation of low-depth quantum circuits can be shown to be even harder for classical devices when they are required to answer the graph state simulation problem in two rounds. Again, here we consider some subgraph $G = (V, E)$ of the $\ell \times \ell$ grid. The first round input is a set of Pauli measurement bases for each of the qubits in some given subset $\mathcal{P}$ of the qubits, and the desired output is any measurement outcome $m \in \{0,1\}^{|\mathcal{P}|}$ such that $p_{\mathcal{P}}(m) \neq 0$. The second round input is a set of measurement bases for the remaining qubits in $\mathcal{S} = V \setminus \mathcal{P}$, and the desired outcome is any string x such that $p_{\mathcal{S}}(x|m) \neq 0$. Of course, this problem can be solved by a constant-depth quantum computation which prepares the graph state and in each round measures the corresponding subset of qubits in the given Pauli bases. Nevertheless, any classical algorithm which succeeds at this interactive task could be used to solve any problem in the class $\oplus\mathsf{L}$. This provides a strong lower bound on the type of classical circuits that can solve the interactive task, as the class $\oplus\mathsf{L}$ is known to be strictly more powerful than the constant-depth circuit family AC_0.[2]

Motivated by the above results, we first observe that Theorem 1 can be improved for the special case of 2D grid graphs.

Theorem 2 (Special case of Theorem 3)**.** *There is an $\widetilde{O}(n^{\omega/2})$-time classical algorithm which solves the graph state simulation problem for any subgraph of a $\sqrt{n} \times \sqrt{n}$ grid.*

The seemingly innocuous quadratic improvement for the graph state simulation problem on grid graphs provides a striking conclusion about the problems used for quantum advantage with low-depth circuits [18, 19, 20]. Although quantum circuits based on measuring grid graph states are superior in terms of depth, Theorem 2 leaves open the possibility that there exist classical circuits with a similar number of elementary gates (e.g., if $\omega = 2$, then a classical circuit of $\widetilde{O}(n)$ gates is possible, matching the quantum circuit with $O(n)$ gates). This leaves their advantage in terms of gate complexity on shakier footing.

Although Theorem 2 follows directly from Theorem 3 (discussed below), we now present a simpler, self-contained algorithm for grid graphs which serves as a warm-up for the more general case. Consider graph state simulation on the $\ell \times \ell$ grid. For simplicity we consider the case without postselection ($\mathcal{P} = \varnothing$) although the algorithms we consider below work in the more general case with only superficial modifications. We shall also ignore fast matrix multiplication for the moment. Our starting point is the very simple classical simulation of the process of preparing the graph state—by starting with the all-zeros computational basis state, applying Hadamards to each qubit, and applying CZ gates between qubits at vertices connected by an edge—and then sequentially measuring the ℓ^2 vertices in the given Pauli bases. Simulating all $O(\ell^2)$ gates in the circuit uses a runtime of $O(\ell^4)$. Since the state has $O(\ell^2)$ qubits, simulating each single-qubit measurement also uses a runtime of $O(\ell^4)$ [17], and the total runtime of this algorithm is $O(\ell^6)$. An improvement was noticed in Ref. [18], and is based on simulating a quantum algorithm that "sweeps" from the left to the right, processing at most 2ℓ qubits at a time. The key idea is that the measurements on any given column commute with the CZ gates two columns away. Concretely, one can prepare all qubits in the first two columns

[2]We note that the postselected graph state simulation problem was already known to be $\oplus\mathsf{L}$-hard [17]. While the interactive task described in Ref. [20] has a similar hardness guarantee, it can be solved by a constant-depth quantum computation.

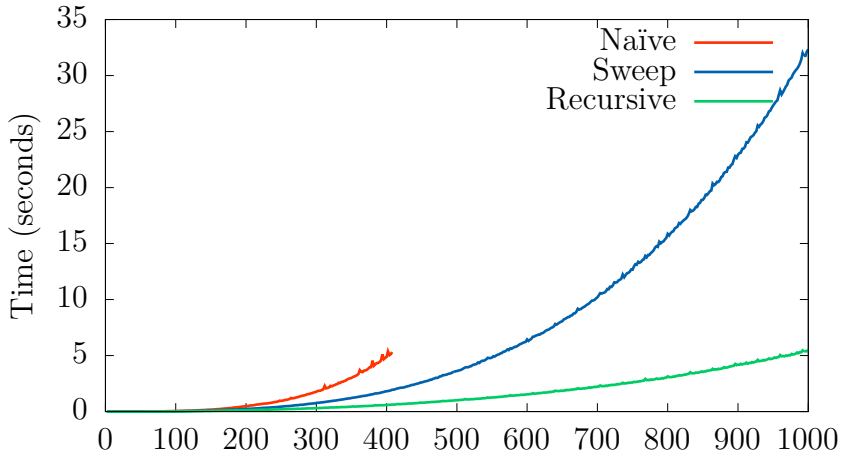


Figure 1: Simulation time in seconds vs. grid length ℓ of an $\ell \times \ell$ grid. The data is taken as an average over 7 random trials from $\ell = 2$ to $\ell = 1000$ by 2's. We cut off the naïve algorithm at $\ell = 408$ due to memory limitations. The simulations were conducted on a desktop computer using a modified version of the CHP Clifford simulator [17, 25]. Our simulations do not use fast matrix multiplication. See Appendix B for the implementation details.

of the grid, measure the qubits in the first column in the given Pauli basis, and then initialize the qubits in the third column and apply the CZ gates between the second and third column. Continuing in this way, the algorithm sweeps from left to right, storing an *active set* of at most 2ℓ qubits at any given time. Since we are now measuring a stabilizer state of only $O(\ell)$ qubits, the cost of each single-qubit measurement is reduced to $O(\ell^2)$. Thus, the classical simulation algorithm runs in $O(\ell^4)$ time. Perhaps surprisingly, this is *not* the ordering of CZ gates and measurements that leads to the most efficient classical algorithm. In the box containing Figure 2, we describe a recursive method for ordering the measurements in which only one active set of size $O(\ell)$ dominates the runtime, leading to the bound given by Theorem 2. We stress that these advantages are not merely theoretical. Figure 1 compares the runtimes of software implementations of the naïve, left-to-right sweep, and recursive algorithms. The results show a clear asymptotic advantage for the recursive algorithm, and an absolute advantage beyond a 50×50 grid (where the algorithm finished in under a hundredth of a second anyway). Using the recursive algorithm we were able to simulate a grid of 529 million qubits in under 11 hours on a desktop computer.

The recursive, divide-and-conquer algorithm for the grid has its roots in the *nested dissection* algorithm of George for solving positive semidefinite linear systems of equations, where the sparsity pattern of the matrix matches the adjacency of the grid graph [26]. This result was later generalized to obtain fast algorithms for solving nonsingular linear systems of equations on planar graphs [27, 28], and likewise our main result generalizes the graph state simulation algorithm from the grid to arbitrary planar graphs.

Theorem 3. *There is an $\widetilde{O}(n^{\omega/2})$-time classical algorithm which solves the graph state simulation problem for planar graphs.*

Our work shares inspiration and methodology with the literature on planar linear system solving algorithms. Moreover, one of the key subroutines from our algorithm

for graph state simulation provides a new method for solving linear systems in a planar geometry. In particular, unlike previous works [27, 28], our algorithm can be applied to singular linear systems.

Theorem 4 (Special case of Theorem 26)**.** *There exists an algorithm to find a uniformly random solution to* $Ay = b \pmod 2$ *(or report that no solution exists) in time* $\widetilde{O}(n^{\omega/2})$ *where* $A \in \mathbb{F}_2^{n\times n}$ *is the adjacency matrix of a planar graph.*

While the simulation algorithm from Theorem 3 does use a divide-and-conquer strategy, we stress that it is not simply obtained using the active set approach—based on a clever ordering of measurements and CZ gates—which we used in the case of the 2D grid.[3]

Instead, the divide-and-conquer algorithm from Theorem 3 is based on a data structure known as the *tree decomposition* of the graph. The tree decomposition consists of nodes–each representing a subset of vertices of the original graph–which are connected together in a certain way to form a tree. Loosely speaking, the more your original graph already resembled a tree, the smaller you can make the subsets at each node (for a formal definition of the tree decomposition, see Section 2.2). The size of the largest subset of vertices at a node in the tree decomposition is referred to as the *treewidth*. Planar graphs are special due to the fact that they admit tree decompositions of width $O(\sqrt{n})$ that can be computed in linear time using the planar separator theorem [29].

More generally, we ask how the runtime of algorithms for graph state simulation scale with the treewidth of the graph. Here we take our inspiration from pioneering work of Markov and Shi who showed that a general (universal) quantum circuit can be simulated on a classical computer in time exponential in the treewidth of a graph derived in a simple way from the circuit [6]. The simulation is based on viewing the quantum circuit as a tensor network and choosing an ordering to contract the tensor network based on a tree decomposition. Tensor-network contraction based algorithms along these lines are especially well-suited to the simulation of near-term quantum computers with a two-dimensional qubit architecture [30, 7, 31, 32]. Of course, our setting is somewhat different as we are interested in polynomial-time algorithms and the special case of Clifford circuits. We describe a parameterized algorithm for the graph state simulation problem for low-treewidth graphs with the following runtime guarantee.

Theorem 5. *There is a classical algorithm which, given an instance of the graph state simulation problem along with a tree decomposition* T *of* G *of treewidth* t *and at most* $O(n)$ *nodes, outputs a solution in* $\widetilde{O}(nt^{\omega-1})$ *time.*

We expect Theorem 5 to be most useful for families of graphs where a tree decomposition can be computed efficiently. Computing the treewidth of a graph is an NP-hard problem [33], and all known classical algorithms for finding a tree decomposition whose width is within a constant factor of optimal require exponential time in the worst case [34].[4]

[3] In fact, some features of our proof—such as the duplication of qubits—are mirrored in existing planar linear system solvers where the adjacency matrix is sparsified by adding additional rows and columns [28].

[4] Markov and Shi's algorithm does not require a tree decomposition to be given because its runtime is exponential in the treewidth, and constructing a suitable tree decomposition from scratch incurs no additional asymptotic cost [35]. In contrast, our algorithms are polynomial in the treewidth.

Theorem 3 and Theorem 5 are both obtained as special cases of a general classical simulation algorithm, which is described in detail in Section 3. At the core of this algorithm is a tree-decomposition-to-circuit mapping which takes a graph G along with a tree decomposition T and outputs a Clifford circuit that solves the graph state simulation problem on G and has a structure similar to that of T. This structure allows us to simulate the circuit in a lazy way, only initializing qubits and applying gates as needed. Just like in the active set approach described above, this reduces the (dominant) cost of measurements in the classical simulation algorithm, such that when we measure a given qubit, the stabilizer state we are measuring has as few qubits as possible. The circuit begins with a quantum state which is a tensor product of few-qubit graph states, one for each leaf node of the tree decomposition. Since a given vertex of the graph may appear in multiple nodes of its tree decomposition, this initial state may have several replica qubits corresponding to each data qubit of the graph state to be simulated. This replication of data qubits is an essential difference between our approach and the active set approach. We will exploit the fact that it is possible to replicate data qubits and operate on the replicas before merging them later in the simulation. During the course of the circuit, gates are applied, qubits are measured, and replica qubits are merged together as needed using a gadget composed of Clifford gates and postselection. We exhibit a classical simulation of this Clifford circuit with a runtime which is a simple function of the tree decomposition. We show that this runtime is upper bounded by Theorem 5 in the general case, and we also provide a sharper upper bound as stated in Theorem 3 for planar graphs.

There are several open questions related to our work. As far as we know, the graph state simulation problem without postselection on general bounded-degree graphs remains a compelling candidate for a polynomial quantum speedup. Can the classical algorithms for the latter problem be improved, or conversely is it possible to strengthen the evidence for a quantum speedup? Another direction for future work is to use our techniques in conjunction with stabilizer-rank based classical simulation algorithms for universal quantum circuits [36, 16, 8]. Roughly speaking, such algorithms are based on representing the quantum state of a given non-Clifford circuit as a superposition of stabilizer states which can each be simulated using the Gottesman-Knill theorem. For this application one requires a stronger form of classical simulation—going beyond the standard tableau representation [17]—which keeps track of the global phase of the stabilizer states, since they appear in superposition. Such phase-sensitive Clifford simulators are described in Refs. [37, 16, 8]. Recent work [38] shows that our results are compatible with stabilizer rank methods, however the need for phase-sensitivity precludes the use of the fast matrix multiplication methods in Section 5.

The remainder of the paper is organized as follows. In Section 1.1 we describe applications of Theorem 3 to the simulation of constant-depth Clifford circuits on planar graphs and an extension to more general Clifford tensor networks. In Section 2 we discuss subroutines for the classical simulation of stabilizer states using the standard tableau representation [17], and we describe improved subroutines for multi-qubit measurements using fast matrix multiplication. This section also reviews relevant background information regarding tree decompositions and treewidth. We describe the tree-decomposition-to-circuit mapping and prove Theorem 5 in Section 3. Finally, in Section 4 we give a sharper analysis of the runtime of our algorithm when applied to tree decompositions that are computed using the planar separator theorem [29] and we prove Theorem 3.

The proof of Theorem 1 is provided in Section 5.

Example: recursive algorithm for 2D grid graphs

Here we describe a simple recursive classical algorithm which attains the runtime bound claimed in Theorem 2 for the graph state simulation problem on an $\ell \times \ell$ grid graph. The *main subroutine* measures (or postselects) qubits in the interior of the subgrid in the given bases, and outputs the measurement outcomes along with the post-measurement state on an "active set" consisting of the $4\ell - 4$ perimeter vertices. The key insight is that the main subroutine can be performed by recursively computing the outcomes and post-measurement states for 4 copies of the $\lfloor \ell/2 \rfloor - 1 \times \lfloor \ell/2 \rfloor - 1$ grid which are completely contained in the interior of the $\ell \times \ell$ grid, separately preparing the stabilizer state on the $O(\ell)$ remaining qubits (including the perimeter), applying the $O(\ell)$ CZ gates which connect them together, and then measuring all qubits except those on the perimeter. The algorithm is illustrated schematically in Figure 2. Letting $f(\ell)$ denote the runtime of this algorithm, we have $f(1), f(2) = O(1)$ and $f(\ell) = 4f(\ell/2) + \widetilde{O}(\ell^\omega)$ for $\ell \geq 3$. Here the first term contains the runtime of the four recursive calls, while the second term contains the runtime required for the remaining steps of the algorithm which only involve preparation and measurement of stabilizer states of $O(\ell)$ qubits. The solution to the recurrence is $f(\ell) = \widetilde{O}(\ell^\omega) = \widetilde{O}(n^{\omega/2})$. In the very last stage of the algorithm the qubits on the perimeter are measured, incurring an additional $O(\ell^\omega)$ cost.

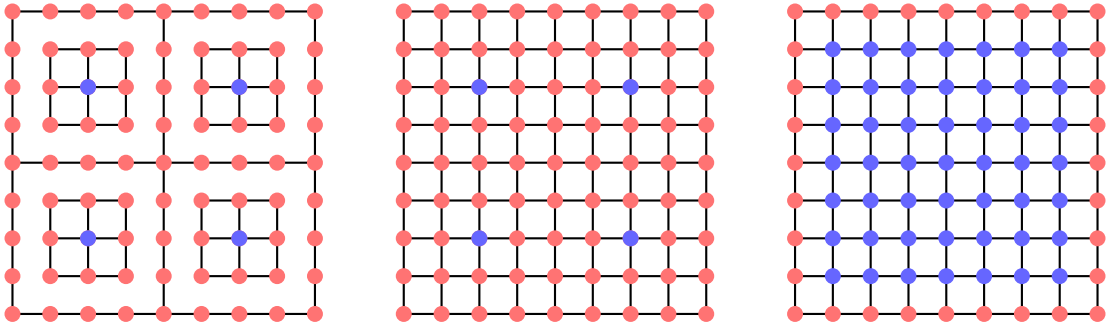


Figure 2: Depiction of the recursive algorithm for graph state simulation on a 2D grid. Blue vertices are measured and red are unmeasured. (a) Quantum graph states corresponding to the 5 connected components of this graph are prepared and the qubits at the blue vertices are measured. (b) CZ gates are applied to connect them together. (c) Single-qubit measurements are performed on all qubits corresponding to red vertices not on the perimeter.

1.1 Applications

In this section, we describe how our algorithm for the graph state simulation problem can be leveraged to solve other more traditional circuit simulation tasks (proofs and more

precise statements in Appendix A). In the most general setting, we prove an analog of the Markov and Shi tensor network contraction algorithm for Clifford tensor networks with planar geometry:

Theorem 6 (informal)**.** *A random non-zero element from a planar Clifford tensor network can be sampled in $\widetilde{O}(n^{\omega/2})$ time.*

Any n-qubit quantum state $|\psi\rangle$ can be represented by a multidimensional matrix (or *tensor*) indexed by $\{0,1\}^n$, whose $(x_1,\ldots,x_n)$th entry is $\langle\psi|x_1\ldots x_n\rangle$. Such tensors are sometimes written as a *tensor network*, where individual tensors are combined together by a generalization of matrix multiplication to describe the global tensor. Sampling from a tensor network refers to sampling from the state that the tensor network represents. A *Clifford tensor network* (see Appendix A for a formal definition) is a tensor network whose individual tensors all represent stabilizer states. The tensors making up a tensor network often have operational meanings such as applying a gate or projecting onto a subspace. These operational meanings illustrate the expressiveness of Clifford tensor networks, which capture unitary operations (such as composition of Clifford gates) and non-unitary operations (such as projection onto stabilizer states). In fact, our theorem for Clifford tensor networks immediately gives an algorithm for the simulation of planar Clifford circuits:

Theorem 7 (informal)**.** *Let C be an n-qubit, depth-d Clifford circuit with qubits located at the vertices of a planar graph and two-qubit gates acting on edges of the graph. A random n-bit measurement outcome of the state $C|0^n\rangle$ can be sampled in time $\widetilde{O}(n^{\omega/2}d^{\omega})$.*

While this result can be obtained as a corollary to Theorem 6, in Appendix A.2 we provide a proof which directly reduces the Clifford circuit simulation task to the graph state simulation problem *without postselection*. Although our algorithms for graph state simulation with and without postselection have the same asymptotic running time, their practical running times are likely to be quite different due to the increased complexity associated with postselection. Indeed, in Section 3 we will provide a self-contained algorithm for graph state simulation without postselection which avoids many of the linear algebra subroutines used in the more general case. This allows one to simulate constant-depth Clifford circuits in a planar geometry in a much more straightforward manner.

We note that the two theorems described above are not an exhaustive list of applications of graph state simulation. There are many possible variations of Clifford simulation problems one might want to solve that would follow from tweaking the proofs of the theorems above. For example, Theorem 7 applies when the qubits of a Clifford circuit are arranged on the vertices of a planar graph, but one could get a similar result for circuits where the *gates* are placed at the vertices of a planar graph with edges between gates if the output of one gate is fed into another gate (e.g., a Clifford circuit with gates acting only on neighboring qubits).[5] Furthermore, planar graphs are not the only graphs which admit efficiently computable tree decompositions. For example, the graph state sampling problem for the $n^{\frac{1}{3}}\times n^{\frac{1}{3}}\times n^{\frac{1}{3}}$ cube can be solved in $\widetilde{O}(n^{\frac{2}{3}\omega})$ time[6] using a tree

[5]In more detail, the underlying graph is as follows: each gate/wire of the circuit is a vertex; and there is an edge between two vertices when the wire is an input/output of the gate.

[6]In fact, such an algorithm follows from the same sort of recursive algorithm used in the proof of

decomposition for the cube of width $O(n^{2/3})$, and so we could obtain similar applications for Clifford circuits where the gates/qubits are arranged in a 3D grid.

Finally, we note that the graph state simulation algorithm for planar graphs still suffices when the graph is close to planar in some sense. To this end, we introduce the notion of coarse-graining a graph (see Figure 3). We say that a graph G' can be coarse-grained to another graph G if there exists a map $\varphi\colon V(G') \to V(G)$ such that for any edge $\{u, v\} \in E(G')$ we have either (a) $\{\varphi(u), \varphi(v)\} \in E(G)$, or (b) $\varphi(u) = \varphi(v)$.[7] We say that φ is an *r-coarse-graining* from G' to G if each vertex of G is the image of at most r vertices of G'.

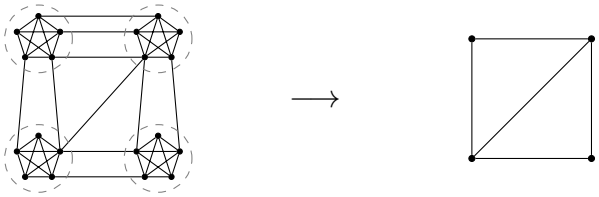


Figure 3: A 5-coarse-graining from a nonplanar graph to a planar graph.

The following theorem, which we prove in Section 4, shows that the graph state sampling algorithm can be extended to graphs which can be coarse-grained to planar.

Theorem 8 (Extension of Theorem 3)**.** *Let G' be a graph with n vertices and let G be a planar graph. Suppose we are given an r-coarse-graining from G' to G. There is a classical algorithm with runtime upper bounded as $\widetilde{O}(n^{\omega/2} r^{\omega})$ which solves the graph state simulation problem for G'.*

2 Preliminaries

In this section we define stabilizer states and tree decompositions and describe some of their properties. In Section 2.1 we review the tableau representation of stabilizer states and describe the runtime of classical algorithms for updating this representation under Clifford operations. In Section 2.2 we review the definition of a tree decomposition of a graph, and we describe simple algorithms which convert a given tree decomposition into one with certain desirable features.

2.1 Stabilizer states

For $a \in \{0,1\}^n$, let $\mathtt{X}^a$ be the n-qubit Pauli operator $\mathtt{X}^{a_1} \otimes \cdots \otimes \mathtt{X}^{a_n}$. Define $\mathtt{Z}^a$ similarly. A general n-qubit Pauli operator P is of the form $i^\alpha (-1)^\beta \mathtt{X}^a \mathtt{Z}^b$ for some n-bit strings a, b and bits $\alpha, \beta \in \{0,1\}$ describing the global phase.

Recall that an n-qubit stabilizer state ψ can be specified (up to a global phase) by a set of independent, commuting n-qubit Pauli operators $\{P_1, P_2, \ldots, P_n\}$ such that $P_j|\psi\rangle =$

Theorem 2: split the cube into 8 identical subcubes, recursively measure their interiors, and then measure the boundary. This leads to the recursion $f(n) = 8f(n/8) + \widetilde{O}(n^{\frac{2}{3}\omega})$. Once again, the computation at the root of the recursion dominates, so we get $f(n) = \widetilde{O}(n^{\frac{2}{3}\omega})$ runtime.

[7] In other words, a graph homomorphism modulo loops, since graph states are typically defined only on loopless graphs.

$|\psi\rangle$. Here independence means that none of the operators P_k is proportional to a product of the other ones. Equivalently, an n-qubit stabilizer state is of the form $|\psi\rangle = Q|0^n\rangle$ where Q is an n-qubit Clifford unitary, i.e., a unitary which can be expressed as a product of single-qubit Hadamard, phase $S = \mathrm{diag}(1, i)$, and two-qubit CZ $= \mathrm{diag}(1, 1, 1, -1)$ gates.

The Gottesman-Knill theorem establishes that stabilizer states can be stored and manipulated efficiently on a classical computer [15]. Throughout this work, we use a classical representation of stabilizer states and Clifford operations which is a variant of the tableau representation described by Aaronson and Gottesman [17]. In particular, we represent both the n-qubit stabilizer state $|\psi\rangle = Q|0^n\rangle$ (up to a global phase) and the Clifford unitary Q as a tableau that lists the images of $\mathtt{X}^{e_j}$ and $\mathtt{Z}^{e_j}$ under conjugation by Q for all j, where $e_j \in \{0,1\}^n$ is the n-bit vector with a 1 in position j and 0's everywhere else. Let

$$Q\mathtt{X}^{e_j}Q^\dagger = i^{\alpha_j}(-1)^{\beta_j}\mathtt{X}^{a_j}\mathtt{Z}^{b_j} \qquad \text{and} \qquad Q\mathtt{Z}^{e_j}Q^\dagger = i^{\delta_j}(-1)^{\epsilon_j}\mathtt{X}^{c_j}\mathtt{Z}^{d_j}.$$

We arrange this conjugation information into a $2n \times 2n$ matrix M, a phase vector $p \in \{0,1\}^{2n}$, and a sign vector $s \in \{0,1\}^{2n}$:

$$M = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \quad p = \left[\begin{array}{c} \alpha \\ \hline \delta \end{array}\right] \quad s = \left[\begin{array}{c} \beta \\ \hline \epsilon \end{array}\right] \tag{4}$$

where A, B, C, and D are the $n \times n$ binary matrices whose elements correspond to the Pauli decompositions. We will refer to this as the *tableau representation* of the state/circuit.

It is well known that the tableau representation can be updated efficiently if we apply a Clifford gate or measure a qubit of the stabilizer state. For example, one can easily see that the runtime of updating the tableau under the application of a 1- or 2-qubit Clifford gate is $O(n)$. Aaronson and Gottesman [17] provide an $O(n^2)$ algorithm for measuring a qubit in the computational basis, improving upon a naïve $O(n^3)$ algorithm. In this work we are interested in measuring batches of qubits all at once. An application of Aaronson-Gottesman would upper bound the runtime of measuring k qubits in the computational basis as $O(n^2k)$. We now describe how this can be improved.

The Aaronson-Gottesman simulation algorithm can be viewed as an application of linear algebra over $\mathbb{F}_2$. In fact, Aaronson and Gottesman show that a variant of the Clifford circuit simulation problem[8] is $\oplus\mathsf{L}$-complete, putting it in the same class as matrix inversion, iterated matrix multiplication, determinant, and other linear algebra tasks over $\mathbb{F}_2$ [39]. Since the discovery of fast matrix multiplication in the 1960s, many linear algebra tasks have been sped up via a reduction to matrix multiplication. The (asymptotically) fastest currently known algorithm for matrix multiplication has a runtime $O(n^\omega)$ for $\omega < 2.3728596$ [23]. It seems quite natural to conjecture that simulating a depth-n Clifford circuit on n qubits, and then measuring all n qubits could be accomplished in $O(n^\omega)$ time. Yet surprisingly, we find no proof of this fact in the literature. To this end, in Section 5 we will give algorithms for fast stabilizer circuit composition (Theorem 39) and state measurement (Theorem 41) using their tableau representations. These operations are described in more detail in Table 1, where we summarize all the subroutines for manipulating stabilizer states which are used in this work.

We will also need the following simple fact about stabilizer states:

[8]Specifically, they consider the task of determining whether measuring the first qubit of a given Clifford circuit's output state yields result $|1\rangle$ with probability 1.

Classical representation of stabilizer states

An n-qubit stabilizer state $|\psi\rangle$ is represented up to a global phase as a tableau Eq. (4) of $4n(n+1)$ bits. This representation can be manipulated as follows:

Append a qubit: $|\psi\rangle|0\rangle$ can be computed in time $O(n)$.

Apply a gate: Any 1- or 2-qubit Clifford gate can be applied to $|\psi\rangle$ in time $O(n)$.

Append: If $|\phi\rangle$ is an ℓ-qubit stabilizer state, compute $|\psi\rangle|\phi\rangle$ in time $O((n+\ell)^2)$.

Measure: Measure a given subset of k qubits (in the Z basis) in time $\widetilde{O}(k^{\omega-2}n^2)$.

Apply CZ gates: Apply an arbitrary collection of CZ gates in $O(n^\omega)$ time.

Table 1: Running times for basic Clifford operations. These follow from Aaronson and Gottesman [17] for naïve matrix multiplication (i.e., for $\omega = 3$). In Section 5, we show how to achieve these running times for $2 \leq \omega < 3$ assuming fast matrix multiplication in time $O(n^\omega)$. See Theorems 39 and 41 specifically.

Claim 9. *Suppose ψ is an n-qubit stabilizer state and $y \in \{0,1\}^n$ satisfies $\langle y|\psi\rangle \neq 0$. If R is a uniformly random stabilizer of ψ, then the bit string $z \in \{0,1\}^n$ such that $|z\rangle \propto R|y\rangle$ is drawn from the distribution $p(z) = |\langle z|\psi\rangle|^2$.*

Proof. Let $\mathrm{Stab}(\psi)$ denote the stabilizer group of ψ. The probability distribution of the random variable z is, by definition,

$$p(z) = \frac{|\Omega|}{|\mathrm{Stab}(\psi)|} = \frac{|\Omega|}{2^n} \qquad \Omega = \{P \in \mathrm{Stab}(\psi) : \langle z|P|y\rangle \neq 0\}. \tag{5}$$

Now suppose $P \in \Omega$, so $\langle y|P = \alpha_P\langle z|$ for some $\alpha_P \in \{\pm 1, \pm i\}$. Then $\langle y|P|\psi\rangle = \langle y|\psi\rangle = \alpha_P\langle z|\psi\rangle$. This shows that α_P does not in fact depend on P and in particular

$$\alpha_P = \alpha \equiv \frac{\langle y|\psi\rangle}{\langle z|\psi\rangle}. \tag{6}$$

Therefore

$$\langle y|\psi\rangle\langle\psi|z\rangle = \frac{1}{2^n}\sum_{Q\in\mathrm{Stab}(\psi)}\langle y|Q|z\rangle = \frac{1}{2^n}\sum_{P\in\Omega}\langle y|P|z\rangle = \frac{\alpha}{2^n}|\Omega|.$$

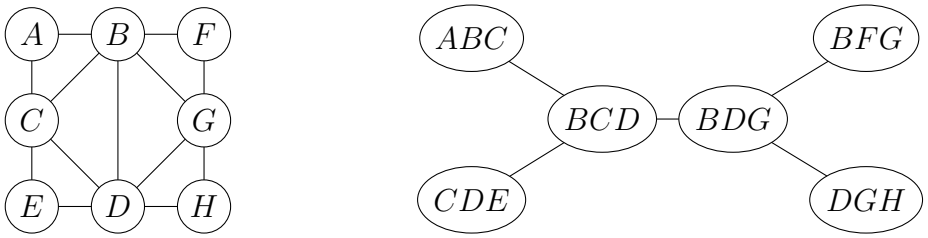
Using Eq. (6) gives $|\Omega| = 2^n|\langle z|\psi\rangle|^2$ and combining with Eq. (5) completes the proof. □

Finally, we note that Claim 9 has the following direct corollary:

Claim 10. *Suppose ψ is an n-qubit stabilizer state and $x, y \in \{0,1\}^n$ satisfy $\langle x|\psi\rangle \neq 0$ and $\langle y|\psi\rangle \neq 0$. Then there exists a stabilizer P of ψ such that $P|x\rangle \propto |y\rangle$.*

2.2 Tree Decompositions

One of our goals in this work is to give a parameterized algorithm for the graph state simulation problem, where the parameter is *treewidth*. Let us introduce tree decompositions and treewidth.



(a) (b)

Figure 4: (a) Planar graph on 8 vertices. (b) A tree decomposition of width 2 [40].

Definition 11. Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a tree T such that each of its nodes is associated with a *bag* of vertices $B_i \subseteq V$ such that the following hold:

1. For all $v \in V$, there exists B_i such that $v \in B_i$.
2. For all $\{u, v\} \in E$, there exists B_i such that $u, v \in B_i$.
3. For all $v \in V$, the nodes $\{B_i : v \in B_i\}$ form a connected subtree of T.

It is conventional to use the word "node" to describe vertices of the tree, to avoid confusion with vertices of the graph G. Similarly, we will use the term "link" to describe edges of the tree, to avoid confusion with the edges of G.

The *width* of a tree decomposition is defined as $t = \max_i |B_i| - 1$, the size of the largest bag minus one. The *treewidth* of a graph is the smallest t for which there exists a tree decomposition of width t. Figure 4 shows an example of a tree decomposition for a graph of treewidth 2.

Although the treewidth is a natural parameter for many graph algorithms, the performance of our algorithms are more precisely captured by the sum of the bag sizes raised to some power. We introduce the notation

$$\|T\|_p := \left(\sum_i |B_i|^p\right)^{1/p}$$

for the p-norm of the vector of bag sizes in a tree decomposition T for real $p > 0$. We also define the ∞-norm

$$\|T\|_\infty := \max_i |B_i|,$$

and note that $t = \|T\|_\infty - 1$.

We will often assume that the tree decomposition has a particularly nice form. We caution that while the definition below is convenient for our purposes, it may diverge from other works.

Definition 12. A *nice* tree decomposition [41] is a rooted tree decomposition where every node is either an *introduce* node, a *forget* node, or a *merge* node:

> *Introduce:* Nodes whose bag strictly contains its child's bag (or that have no children). These nodes are either leaves, or have one child.
>
> *Forget:* Nodes that have exactly one child and whose bag is strictly contained in their child's bag.

Merge: Nodes that have exactly two children and whose bag is the union of its children's bags.

The following theorem will allow us to work with nice tree decompositions containing relatively few nodes whenever it is convenient.

Theorem 13. *Let $G = (V, E)$ be a graph with $n = |V|$ vertices, and let T be a tree decomposition of G of width t. We can compute a nice tree decomposition T' of G with $O(n/t)$ nodes and width $O(t)$ in $O(\|T\|_1)$ time.*

We will prove this theorem from a series of lemmas. For a node B with parent A in a rooted tree decomposition, we say that the vertices $B \setminus A$ are *forgotten* on the link from A to B. If there is a link between nodes A and B, *contracting* the two nodes refers to the process of contracting the link between A and B and setting the combined node equal to $A \cup B$. If nodes B and S are siblings,[9] then *merging* them refers to the following process: delete the nodes B and S; create a new node $B \cup S$ with a link to parent shared by B and S, as well as links to any children of B and S. Note that the contraction and merge operations preserve the requirements for a tree decomposition set out in Definition 11.

Lemma 14. *Given a rooted tree decomposition of a graph, every vertex of the graph is either forgotten exactly once or belongs to the root node.*

Proof. Consider an arbitrary vertex v that is not in the root node. Suppose for the sake of contradiction that v is forgotten along two distinct links (i.e. v is in two distinct bags A and B, but not in their respective parent(s)). However, A and B are connected by a unique path which goes through one or both of their parents. However, by Definition 11, the nodes containing v are connected, contradicting our assumption that neither parent contains v. □

Lemma 15. *Consider a rooted tree decomposition. Let B be a node with child C, parent A, sibling S, and grandparent P.*[10] *Then:*

1. *If we contract A and B into a node $A \cup B$ then*

$$C \backslash (A \cup B) = C \backslash B, \qquad S \backslash (A \cup B) = S \backslash A, \qquad (A \cup B) \backslash P = (B \backslash A) \cup (A \backslash P).$$

2. *If we merge B and S into a new child $B \cup S$ of A then*

$$C \backslash (B \cup S) = C \backslash B, \qquad (B \cup S) \backslash A = (B \backslash A) \cup (S \backslash A).$$

In other words, contract and merge operations (i) preserve the nodes forgotten by their children or siblings; and (ii) forget the (disjoint) union of the vertices forgotten by their constituent parts.

Proof. The proof consists of repeated application of the requirement that vertices appear in connected subsets of a tree decomposition. For the first four equalities, we will show that the two sets are equal by showing that they contain one another.

[9] We say that two nodes are siblings if they share the same parent.

[10] We say that node P is a grandparent of B if P is the parent of B's parent.

Consider the parent-child contraction $A, B \mapsto A \cup B$ first. Clearly $C \backslash (A \cup B) \subseteq C \backslash B$. If there was some vertex $v \in C \backslash B$ but $v \notin C \backslash (A \cup B)$, then clearly $v \in C$ and $v \notin B$, but $v \in A \cup B$ and hence $v \in A$. This contradicts the fact that bags containing v are connected. Therefore we must also have $C \backslash (A \cup B) \supseteq C \backslash B$, so we conclude $C \backslash (A \cup B) = C \backslash B$. Similarly, $S \backslash (A \cup B)$ is contained in $S \backslash A$, and if the containment was strict we would have $v \in B$ and $v \in S$, but $v \notin A$, a contradiction.

Next, we have $(A \cup B) \backslash P = (B \backslash P) \cup (A \backslash P) \supseteq (B \backslash A) \cup (A \backslash P)$, since any $v \in B \backslash A$ is also in $B \backslash P$. (Otherwise, $v \in B$ and $v \in P$ implies $v \in A$ and hence $v \notin B \backslash A$, a contradiction.) In the other direction, take an arbitrary $v \in (B \backslash P) \cup (A \backslash P)$. Clearly $v \notin P$, but it must belong to either A or B. If $v \in A$ then $v \in (A \backslash P) \subseteq (B \backslash A) \cup (A \backslash P)$. Otherwise, $v \notin A$ and $v \in B$, so again $v \in (B \backslash A) \cup (A \backslash P)$. Thus, the two sets are equal.

Now consider the sibling merge $B, S \mapsto B \cup S$. Once again, clearly $C \backslash (B \cup S) \subseteq C \backslash B$. If $v \in C \backslash B$ then $v \in C$ and $v \notin B$. The path connecting C and S must pass through B, so using the fact that bags containing v form a connected subtree, we conclude that $v \notin S$, and thus $v \in C \backslash (B \cup S)$. The final equality follows from the distributive law:

$$(B \cup S) \backslash A = (B \cup S) \cap \overline{A} = (B \cap \overline{A}) \cup (S \cap \overline{A}) = (B \backslash A) \cup (S \backslash A).$$

□

Lemma 16. *Given a graph G, a rooted tree decomposition T for G of treewidth t, and an integer $c \geq 0$, we can construct a new rooted tree decomposition with treewidth at most $t + 2c$ and at most $\frac{n}{c+1} + 1$ nodes in $O(\|T\|_1)$ time. In particular for $c = \Theta(t)$ the tree decomposition has width $O(t)$ and $O(n/t)$ nodes.*

Proof. The algorithm proceeds recursively, moving from the leaf nodes up to the root node. For a bag B, we recursively process its children. Then we repeatedly merge pairs of children whenever there are two that both forget $\leq c$ vertices along the link to B. (This may result in more than two children being merged into the same node. For example, if B has three children that each forget $c/3$ vertices along the link to B, we would merge the first two, and then merge the new node with the third.) Once this process is complete, there might still be one such child. If this is the case, we contract it into B.

First, we prove that the resulting tree decomposition has at most $\frac{n}{c+1} + 1$ nodes. We do so by claiming that when this process is complete, all nodes (except the root) forget at least $c + 1$ vertices. Suppose to the contrary that some node B forgets $\leq c$ vertices. By Lemma 15, the merges and contractions above B in the tree do not change the set of vertices B forgets, so B forgot $\leq c$ vertices even at the recursive call on its parent. But then the algorithm would have merged it with a sibling, or if no sibling had $\leq c$ vertices, contracted it with its parent, which contradicts the fact that B exists in T. Thus, all links forget at least $c + 1$ vertices. By Lemma 14, every vertex is forgotten (somewhere in the tree) at most once, so there are at most $\frac{n}{c+1}$ links, and thus $\frac{n}{c+1} + 1$ nodes.

Next, we show that the resulting treewidth is at most $t + 2c$. Merging and contraction usually results in larger nodes, potentially beyond the original upper bound of $t + 1$ vertices. However, when we merge two siblings forgetting $\leq c$ vertices each, the resulting node forgets $\leq 2c$ vertices (by Lemma 15), and is thus at most $2c$ vertices larger than its parent. The parent is (at this point in the algorithm) untouched from the original tree, so it has at most $t + 1$ vertices, and thus the merged node has $\leq t + 1 + 2c$ vertices. Likewise, contraction only occurs when there is a child forgetting $\leq c$ vertices, so we add $\leq c$ vertices to an untouched parent with $\leq t + 1$ vertices for at total size of $\leq t + 1 + c$.

The size of a bag never exceeds $t + 2c + 1$ during transformation, and so the resulting treewidth is at most $t + 2c$.

Finally, we claim that the runtime can be made to be $O(\|T\|_1)$. The main hurdle is that if we merge and contract bags by explicitly computing the union of the two bags every time, then our runtime might not necessarily be $O(\|T\|_1)$. Instead, we will wait until the very end before explicitly computing the contents of each bag. Observe that each bag in the new tree decomposition is a union of bags from T, since that's all that the merge and contraction operations do to the contents of bags. Throughout the algorithm, for each bag we will store a set of pointers to the bags of T that that bag is the union of, and wait until the very end to compute all unions. In order to do so, we need to show that the number of vertices forgotten along a link can be computed using this pointer-based approach (rather than explicitly knowing the contents of the child's bag). Recall that by Lemma 15, when we merge two nodes B and S with common parent A, the number of vertices $|(B \cup S) \setminus A|$ forgotten along the new link to A is given by $|B \setminus A| + |S \setminus A|$. Lemma 15 also tells us that when contracting B and A, the number of vertices $|(A \cup B) \setminus P|$ forgotten along the new link to A's parent P is given by $|B \setminus A| + |A \setminus P|$. These simple rules allow us to keep a running tally of how many vertices are forgotten along each link, without having to explicitly compute bag contents every time. Using these rules, whenever we merge or contract two nodes, we can update the number of vertices forgotten along the link from the resulting node to its parent in time $O(1)$.

Before we begin processing T, we compute the number of vertices forgotten along each link. For a bag B with children $C_1, \ldots, C_k$, the number of vertices forgotten along each link to B can be computed in time $|B| + |C_1| + \cdots + |C_k|$ by enumerating vertices of $C_1, \ldots, C_k$ and checking each against B. Repeating this for each bag takes time $O(\|T\|_1)$, since each bag is used at most once as a parent and once as a child. Then, while performing the transformation on the tree decomposition, whenever we merge or contract two nodes, we can update the number of vertices forgotten along the link from the resulting node to its parent using the rules discussed above in time $O(1)$. Next, observe that each bag from T is represented in exactly one bag of the new tree decomposition. Therefore, if for each bag in the new tree decomposition, we have pointers to the bags of T that the new bag is the union of, then we can explicitly compute the contents of all bags in a total time of $O(\|T\|_1)$, since each bag of T will be processed by the "union" operation exactly once. This shows that the total runtime is $O(\|T\|_1)$. □

Lemma 17. *Given a width-t tree decomposition T with m nodes, we can construct a width-t nice tree decomposition T' with at most $\frac{7}{2}m$ nodes in $O(mt)$ time.*

Proof. We shall describe the transformation in two steps.

In the first step, for each node with bag B and children with bags $C_1, \ldots, C_k$, add nodes $B \cap (C_1 \cup \cdots \cup C_k)$ and $B \cap C_1, \ldots, B \cap C_k$ as shown in Figure 5. Contract links where the parent and child have the same bag.

It is clear that for each link in the tree, the bags of the endpoints are comparable with either $\subseteq$ or $\supseteq$. Classify nodes with zero or one children as introduce nodes or forget nodes by the direction of this containment.

The second step of the transformation is as follows. For any node with more than two children, note that it is the union of its children. It will eventually become a merge node, but first we need to reduce the number of children. To do this, we pick two children arbitrarily, give them a new merge node as a parent, and make it a child of the original

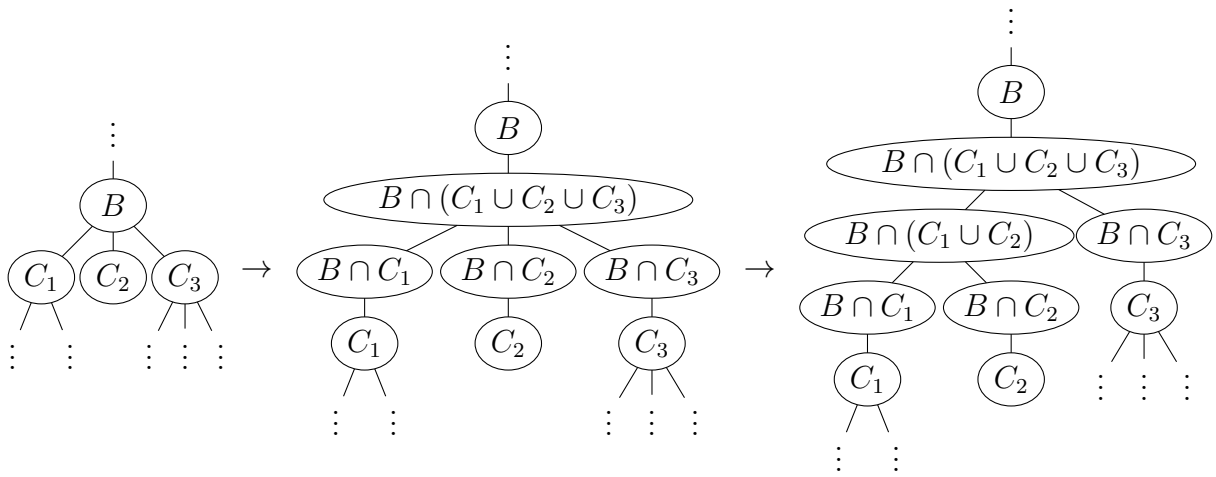


Figure 5: Depiction of the transformation used in Lemma 17 which maps a tree decomposition to a nice tree decomposition.

parent. Of course, the new merge node bag will be the union of its children's bags (as is required for a merge node). Repeat until all nodes have at most two children.

Figure 5 depicts an example of this transformation. Note that the transformation only creates nodes that are intersections of others, so the maximum bag size cannot increase. When we create merge nodes from two children, we take the union of two nodes, but both are already contained in their parent, so the new bag cannot be bigger than its parent. Thus, treewidth does not increase.

Observe that since T is a tree with m nodes, it has $m-1$ links. The first step of the transformation creates at most one node per link, and an additional node per multi-child internal node. There are at most $m-1$ links, and at most $\frac{m-1}{2}$ multi-child internal nodes (since each one is associated with at least two links), so we add at most $\frac{3}{2}m$ nodes in this step. For a bag B with children $C_1, \ldots, C_k$, this step takes time $O\left(|B| + \sum_{i=1}^{k} |C_i|\right)$, which leads to a total runtime of $O(||T||_1)$ for this first step, which is in $O(mt)$.

If a node of the tree has $d > 2$ children, then in the second step of the transformation we create new merge nodes which blow it up into a binary tree with d leaves. The leaves and root of the binary tree already exist, but we create $d-2$ additional nodes. The total number of children across all nodes is $m-1$, so we add at most $m-3$ additional nodes. Therefore in total, the process introduces at most $\frac{5}{2}m$ new nodes. Because each new node is obtained by taking unions and intersections of at most three nodes (whose sizes are at most $t+1$), the time to compute each new node is $O(t)$. Combining this with the fact that this transformation only creates $O(m)$ new nodes, we find that the runtime of this second step is $O(mt)$. □

We now prove Theorem 13.

Proof of Theorem 13. Set $c = \Theta(t)$ in Lemma 16 to contract/merge down to a tree decomposition T' with $O(n/t)$ nodes and width $O(t)$ in $O(\|T\|_1)$ time. Then use Lemma 17 on T' to get a *nice* tree decomposition T'' in $O(\frac{n}{t} \cdot t) = O(n)$ time, still with $O(n/t)$ nodes and width $O(t)$. Clearly $n = O(\|T\|_1)$ since T contains every vertex, so the result follows. □

Suppose that $\varphi\colon V(G) \to V(H)$ is a coarse-graining from a graph G to another graph H. The following Lemma allows us to compute a tree decomposition of G, given one of H. In particular, suppose T_H is a tree decomposition of H. Define a tree decomposition $T_G := \varphi^{-1}(T_H)$ to have the same tree as T_H but each bag B in T_H is replaced with its preimage

$$\varphi^{-1}(B) := \{v \in V : \varphi(v) \in B\}.$$

We now prove that T_G is indeed a tree decomposition of G.

Lemma 18 (Tree Decompositions are closed under coarse-graining)**.** *Let $\varphi\colon G \to H$ be a coarse-graining from a graph $G = (V, E)$ to graph H. Given a tree decomposition T_H of the graph H, the preimage $T_G := \varphi^{-1}(T_H)$ is a tree decomposition for T_G.*

Proof. We check the defining properties of a tree decomposition explicitly:

1. For any vertex $v \in V$, its image $\varphi(v)$ appears in some bag B of T_H, and therefore v appears in $\varphi^{-1}(B)$ in T_G.
2. For each edge $\{u, v\} \in E$, either $\varphi(u) = \varphi(v)$ appears in some bag B and therefore $\{u, v\}$ appears in $\varphi^{-1}(B)$, or $\varphi(u) \neq \varphi(v)$ but $\{\varphi(u), \varphi(v)\}$ is an edge appearing in some bag B of T_H, and thus $\{u, v\}$ appears in $\varphi^{-1}(B)$.
3. Take an arbitrary vertex $v \in V$. It appears in nodes of T_G corresponding to nodes of T_H that contain $\varphi(v)$. Since this is a connected subtree of T_H (by the tree decomposition property of T_H), it is also connected in T_G.

□

3 Algorithm for graph state simulation

In this section we describe our algorithm for the graph state simulation problem which makes use of a tree decomposition of the graph. We first recall the definition of the problem:

Graph state simulation problem. *The input to the problem is a graph $G = (V, E)$ with $n = |V|$ vertices, a measurement basis $P_v \in \{\mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$ for each vertex $v \in V$, a partition $[n] = \mathcal{S} \cup \mathcal{P}$ of the vertices, and a binary string $m \in \{0, 1\}^{|\mathcal{P}|}$. If the marginal probability $p_{\mathcal{P}}(m)$ from Eq. (2) is zero, the output is an error flag. Otherwise, the output is a binary string $x \in \{0, 1\}^{|\mathcal{S}|}$ from the conditional distribution $p_{\mathcal{S}}(x|m)$ defined in Eq. (3).*

Our goal in this section is to prove the following theorem:

Theorem 19. *There is a classical algorithm which takes as input an instance of the graph state simulation problem along with a nice tree decomposition T of the given graph G. The algorithm outputs a solution in time $\widetilde{O}(\|T\|_\omega^\omega)$.*

Theorem 5 from the introduction (restated below) follows immediately.

Theorem 5. *There is a classical algorithm which, given an instance of the graph state simulation problem along with a tree decomposition T of G of treewidth t and at most $O(n)$ nodes, outputs a solution in $\widetilde{O}(nt^{\omega-1})$ time.*

Proof. Theorem 13 allows us to map the given tree decomposition T to a nice tree decomposition T' with $O(n/t)$ nodes and treewidth $O(t)$ in time $O(nt)$. Clearly $\|T'\|_\omega^\omega = \sum_i |B_i|^\omega = O(nt^{\omega-1})$ since there are at most $O(n/t)$ bags in T' of size at most $O(t)$. Applying Theorem 19 to T' and G finishes the proof. □

Later, in Section 4 we will show how the graph state sampling algorithm for planar graphs (Theorem 3) also follows from Theorem 19. Thus, Theorem 19 can be viewed as our main result in this work.

Algorithm overview Let us first describe the high level structure of the algorithm described in Theorem 19. It is based on a quantum circuit which is constructed from the given tree decomposition. The circuit, which we denote by $\mathcal{C}$, is obtained by replacing each node of T with a gadget based on the node type, i.e., introduce, forget, or merge. Since the circuit has $n = |V|$ data qubits labelled by vertices of the graph, and one qubit for each of the n_a merge gadgets (we refer to these as *merge ancillas*), $\mathcal{C}$ acts on a total of $n_t := n + n_a$ qubits. All qubits are initially prepared in the state $|+\rangle \equiv 2^{-1/2}(|0\rangle + |1\rangle)$. We will see that, if we apply the circuit and then project all merge ancillas into the all-zeros state, the resulting state of the n data qubits is the graph state to be simulated:

$$(I \otimes \langle 0^{n_a}|)\mathcal{C}\,(|+^n\rangle \otimes |+^{n_a}\rangle) \propto |G\rangle. \tag{7}$$

Here the right-hand side is of course independent of the tree decomposition. To solve the graph state simulation problem it therefore suffices to classically simulate the following procedure: apply $\mathcal{C}$, project all merge ancillas onto the all-zeros state, project all data qubits in the set $\mathcal{P}$ according to the given measurement bases/outcomes, and measure all data qubits in the set $\mathcal{S}$ in the given Pauli bases. Our algorithm is a slight variant of this and is based on (a classical simulation of) of two steps: a sampling subroutine, followed by a correction subroutine.

In the first step, which we call the *sampling subroutine*, we apply $\mathcal{C}$, measure all merge ancillas in the computational basis, and measure all data qubits in the given Pauli bases $\{P_v\}_{v\in V}$. This produces a bit string $y \in \{0,1\}^{n_t}$ drawn from a distribution

$$p(y) = |\langle y|(U_{\text{bases}} \otimes I)\mathcal{C}|+^{n_t}\rangle|^2. \tag{8}$$

Here we write U_{bases} for the n-qubit Clifford unitary which locally changes the basis of each data qubit to the given Pauli bases $\{P_v\}$, such that

$$P_v U_{\text{bases}}|z\rangle = (-1)^{z_v} U_{\text{bases}}|z\rangle \qquad\qquad z \in \{0,1\}^n.$$

In particular, U_{bases} is a tensor product of single-qubit identity gates, Hadamard gates, and gates

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix}$$

which map the computational basis to itself, the X basis, and the Y basis respectively. A crucial feature of $\mathcal{C}$, as we will see in more detail below, is that its structure mirrors that of the tree decomposition. We show how to classically simulate the circuit in a "lazy" way whereby gates are applied from the leaves up to the root of the tree and qubits are measured as soon as possible (after all operations on them have been applied). At any point in the simulation the state is separable under a partition reflecting the structure of

the tree decomposition; we will show how this leads to a runtime of $\widetilde{O}(\|T\|_\omega^\omega)$ which can be far less than the naïve simulation based on the Gottesman-Knill theorem.

The output $y \in \{0,1\}^{n_t}$ of the sampling subroutine will typically fail to have all the merge ancillas in the all-zeros state, and thus will not directly provide a solution to the graph state simulation problem (since Eq. (7) will not be satisfied). In addition, y may disagree with the given measurement outcomes for the data qubits in the set $\mathcal{P}$ to be postselected. We will fix both issues in the *correction subroutine* of the algorithm. The goal of the correction subroutine is to sample a uniformly random n_t-qubit Pauli P_{cor} such that

$$P_{\text{cor}}(U_{\text{bases}} \otimes I)\mathcal{C}|+^{n_t}\rangle \propto (U_{\text{bases}} \otimes I)\mathcal{C}|+^{n_t}\rangle \tag{9}$$

and

$$P_{\text{cor}}|y\rangle \propto |z\rangle \otimes |0^{n_a}\rangle \qquad z|_{\mathcal{P}} = m. \tag{10}$$

Here the notation $z|_{\mathcal{P}} = m$ means that $z \in \{0,1\}^n$ matches the desired postselected outcomes (i.e., z agrees with m on the qubits in the set $\mathcal{P}$). The runtime used in our algorithm to compute P_{cor} is $O(\|T\|_\omega^\omega)$. For the special case where postselection on the data qubits is not needed (i.e., $\mathcal{P} = \varnothing$), we also provide a simplified algorithm with a faster $O(n_t + |E|)$ runtime.

Plugging Eqs. (9, 10) into Eq. (8) and using Eq. (7) we see that

$$p(y) = |\langle y|P_{\text{cor}}(U_{\text{bases}} \otimes I)\mathcal{C}|+^{n_t}\rangle|^2 \propto |\langle z|U_{\text{bases}}|G\rangle|^2. \tag{11}$$

Therefore, the binary string z defined by $P_{\text{cor}}|y\rangle \propto |z\rangle$ is a valid measurement outcome for the given graph state simulation instance, i.e., it occurs with nonzero probability if the graph state is measured in the given bases, and matches the specified postselection outcomes. To show that z is sampled from the correct probability distribution, which is uniform over all valid measurement outcomes, we appeal to Claim 10. Suppose z' is another valid measurement outcome for this given graph state simulation instance. Claim 10 states that there exists a Pauli P in the stabilizer group of $(U_{\text{bases}} \otimes I)\mathcal{C}|+^{n_t}\rangle$ such that $P|z\rangle \propto |z'\rangle$. Since P_{cor} is a uniformly random stabilizer subject to Eqs. (9, 10), the stabilizer PP_{cor} is also uniformly random subject to those constraints. By definition, sampling P_{cor} in the correction step leads to output z, whereas sampling PP_{cor} leads to output z'. Therefore any two valid measurement outcomes z, z' are equally likely and we have shown that the output of the correction step is uniformly sampled from the set of valid measurement outcomes.

In the remainder of this section we fill in the details of the algorithm described above. In Section 3.1 we describe the circuit $\mathcal{C}$ and establish Eq. (7). In Section 3.2 we describe the sampling subroutine, show that it outputs a bit string y sampled according to Eq. (8), and establish the runtime bound $\widetilde{O}(\|T\|_\omega^\omega)$. Finally, in Section 3.3 we describe the correction subroutine, and show that it outputs a random Pauli P_{cor} satisfying equations (9) and (10) using a runtime $\widetilde{O}(\|T\|_\omega^\omega)$.

3.1 The quantum circuit $\mathcal{C}$

Given a nice tree decomposition T of a graph G, let us consider a quantum computation obtained by replacing each node with a gadget based on its type—introduce, forget, or merge. The gadgets depend on both the node and its children, so let A be bag of the current node, and let B_1, B_2 be the bags of its children, or just B for a single child, or $B = \varnothing$ if no children. The gadgets are depicted in Figure 6 and described below:

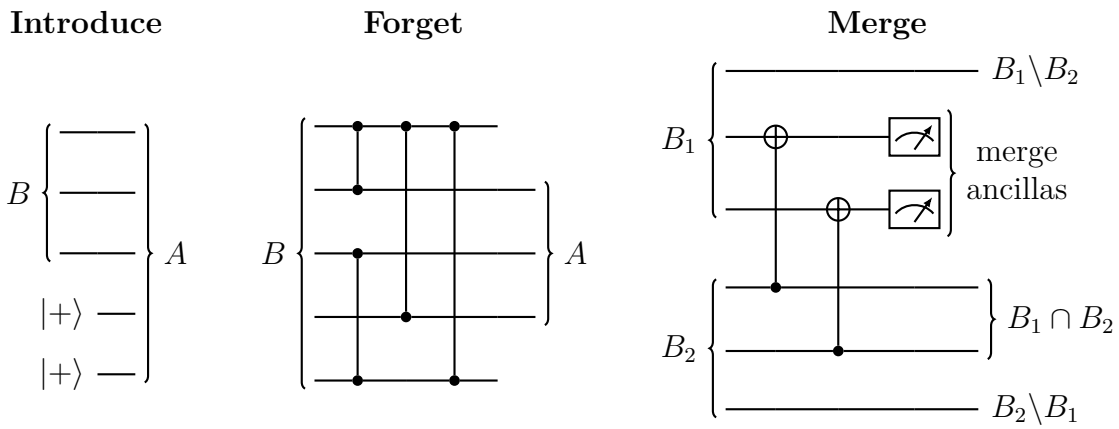


Figure 6: Examples of the three gadgets for translating tree decomposition T to circuit $\mathcal{C}$.

Introduce: Add a qubit in state $|+\rangle$ for each vertex in $A \backslash B$, and pass everything else through.

Forget: Set aside qubits $B \backslash A$ (i.e., do not pass them to the parent), but first, we apply CZ gates for any edges with at least one endpoint in $B \backslash A$ and both endpoints in B.

Merge: For each vertex $v \in B_1 \cap B_2$, apply a CNOT from the qubit $v \in B_2$ to qubit $v \in B_1$, then measure the latter qubit, a *merge ancilla*, in the Z-basis.

In other words, when a bag introduces a qubit, we add a new qubit in state $|+\rangle$. Just before a bag forgets a qubit, there is a "last call" to apply any CZ gates involving that qubit. Last, merge nodes apply the merge gadget to combine duplicate qubits.

See Figure 7 for an example of the entire construction. We define the quantum circuit $\mathcal{C}$ to be the *unitary* part of the circuit constructed in this way, which contains only CZ and CNOT gates and does not contain any measurements. So the quantum computation described above and depicted in Figure 7 consists of preparing all qubits in the $|+^{n+n_a}\rangle$ state, applying $\mathcal{C}$ and then measuring all merge ancillas in the computational basis.

Note that the total number of qubits in the circuit is upper bounded as

$$\text{Total number of qubits in } \mathcal{C} = n_t = n + n_a \leq \|T\|_1. \tag{12}$$

The circuit contains a CZ gate for every edge in the graph and a CNOT gate for each merge ancilla. Therefore the total number of gates in $\mathcal{C}$ is

$$\text{Total number of gates in } \mathcal{C} = |E| + n_a. \tag{13}$$

We now establish Eq. (7).

Theorem 20. *Let G be a graph with a nice tree decomposition T. If all the merge ancillas are measured in the state $|0\rangle$, then the state on the remaining qubits is precisely $|G\rangle$.*

Proof. We will prove the following more general statement:

Claim 21. *The state prepared by $\mathcal{C}$ is a graph state $|G'\rangle = \mathcal{C}|+^{n_t}\rangle$ where the graph $G' = (V', E')$ has the following properties:*

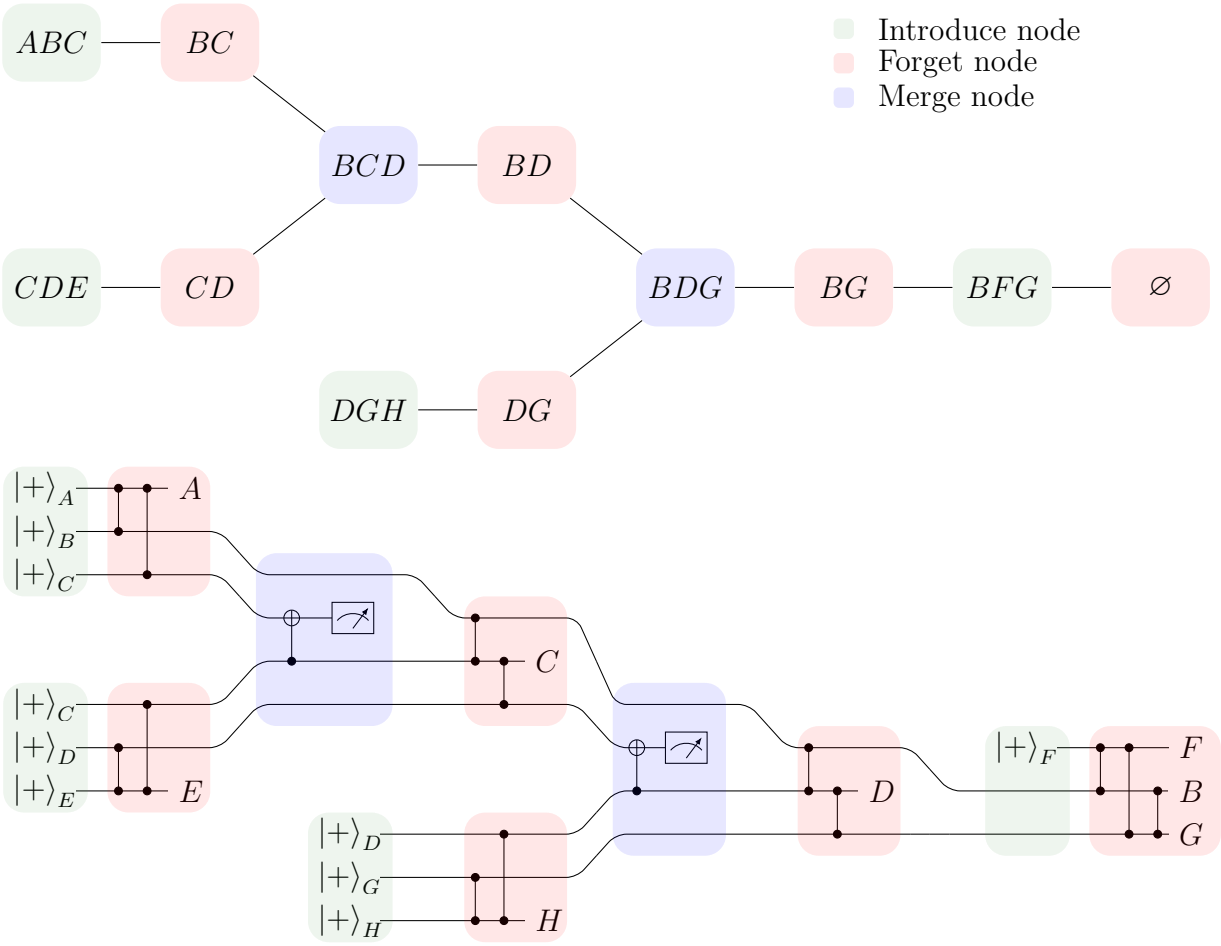


Figure 7: *Top:* Nice tree decomposition T of graph in Figure 4. *Bottom:* The quantum computation which prepares the state $\mathcal{C}|+^{n_t}\rangle$ and then measures all merge ancillas in the computational basis.

- *$V' = V \cup M$ where V is the vertex set of G and M contains a vertex for each merge ancilla in $\mathcal{C}$ (i.e., $|M| = n_a$).*
- *$E' = E \cup F$ where E is the edge set of G (with both endpoints in V) and F only contains edges with at least one endpoint in M.*

Theorem 20 follows directly from the claim, since (using an easily verified property of graph states) $(I_V \otimes \langle 0|_M)\,|G'\rangle$ is proportional to the graph state for the induced subgraph of G' on vertices in V. Claim 21 states that this induced subgraph is G.

It remains to establish the claim. Consider the two circuit identities shown in Fig. 8 describing how CNOT moves past CZ.

Figure 8: Two circuit identities.

Putting these together, we see that we can push CZ gates to the right of CNOT gates that appear in a merge gadget, as shown in Fig. 9.

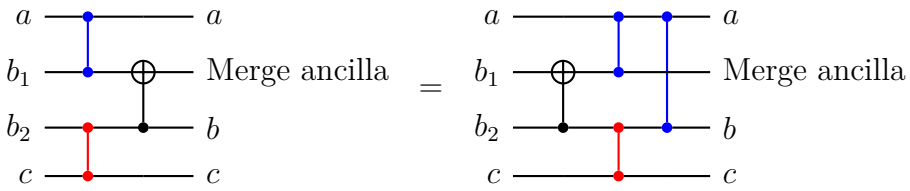


Figure 9: Pushing CZ gates to the right of CNOTS

We can use this rule to move all the CNOT gates in $\mathcal{C}$ to the left of all CZ gates. The CNOT gates can then be removed from the circuit altogether, as they have a trivial action on the initial state $|+^{n_t}\rangle$ (that is, CNOT $|{+}{+}\rangle = |{+}{+}\rangle$).

Once we have removed all the CNOTs from the circuit in this manner, we are left with a circuit composed of CZ gates acting on $|+^{n_t}\rangle$, that is, a graph state. The graph G' has $n + n_a$ vertices as claimed. Using the circuit identities above we can also verify that the edge set of G' has the desired properties. □

3.2 The sampling subroutine

The goal of the sampling subroutine is to (classically simulate) the following process: initialize $n_t = n + n_a$ qubits in the $|+\rangle$ state, apply the circuit $\mathcal{C}$ (described in the previous section), measure all n data qubits in the given Pauli bases $\{P_v\}_{v \in V}$, and measure all n_a merge ancillas in the computational basis. The output of the sampling subroutine is a binary string $y \in \{0,1\}^{n_t}$ sampled from the distribution Eq. (8).

The sampling subroutine is a recursive algorithm which is described in Algorithm 1. In particular, the desired output $y \in \{0,1\}^{n_t}$ is obtained by running Algorithm 1 with the given graph G, and tree decomposition T (recall it has an empty root bag). The algorithm should be interpreted as a *classical* algorithm in which all stabilizer states

are represented using tableaux and all operations are performed using the update rules summarized in Table 1.

Algorithm 1 Sampling subroutine

Input: Nice tree decomposition T

Global: Graph G and Pauli measurement bases $\{P_v\}_{v\in V(G)}$.

Output: Print measurement outcomes of $\mathcal{C}$ for merge ancillas and each vertex *not* in the root bag of T. Return post-measurement state of vertices in root bag.

function SampleGraph(T)
($\triangleright$) We denote the root bag of T by A, and its child bags by B_1 and B_2 (or just B if it only has one child). Similarly, let S_1 and S_2 be the left and right subtrees of T (or S if there is only one child).
    if T is a single node **then**
        return $\bigotimes_{v\in A} |+\rangle_v$
    if root(T).type = Introduce **then**
        $|\psi\rangle \leftarrow$ SampleGraph(S)
        return $|\psi\rangle \otimes \bigotimes_{v\in A\setminus B} |+\rangle_v$
    if root(T).type = Forget **then**
        $|\psi\rangle \leftarrow$ SampleGraph(S)
        Apply CZ to $|\psi\rangle$ for each edge $\{u, v\} \in E(G)$ with $u \in B$ and $v \in B\setminus A$
        For each $v \in B\setminus A$, measure corresponding qubit of $|\psi\rangle$ in P_v basis
        return post-measurement state
    if root(T).type = Merge **then**
        $|\psi_1\rangle \leftarrow$ SampleGraph(S_1)
        $|\psi_2\rangle \leftarrow$ SampleGraph(S_2)
        For each $v \in B_1 \cap B_2$, apply CNOT controlled by qubit v in $|\psi_2\rangle$, to qubit v in $|\psi_1\rangle$.
        For each such v, measure the corresponding qubit of $|\psi_1\rangle$ in Z basis
        return post-measurement state

Theorem 22. *Algorithm 1 runs in $\widetilde{O}(\|T\|_\omega^\omega)$ time.*

Proof. Algorithm 1 recursively simulates $\mathcal{C}$ gadget by gadget up the tree decomposition, so it suffices to upper bound the runtime for simulating each gadget using the stabilizer simulation algorithms summarized in Table 1. Let A be the bag of the current node, and B_1, B_2 be the bags of its children (or B if there is only one child).

Introduce node: Initializing $|A\setminus B|$ many qubits in the $|+\rangle$ state uses a runtime $O(|A|^2)$.

Forget node: First we apply all CZ gates for edges which touch forgotten vertices in B, which takes $O(|B|^\omega)$ time. We then measure qubits at vertices in $B\setminus A$, requiring $\widetilde{O}(|B\setminus A|^{\omega-2}|B|^2) \le \widetilde{O}(|B|^\omega)$ time.

Merge node: We apply CNOT gates between $O(|B_1\cap B_2|)$ pairs of qubits, and a similar number of measurements. We bound both as costing at most

$$\widetilde{O}((|B_1| + |B_2|)^\omega) = \widetilde{O}(|A|^\omega).$$

In each of the above cases, we can conservatively upper bound the cost as the sum of the ωth power of all the bags involved. Since each bag is involved in at most two gadgets (once as a child, once as a parent), the cost for the entire circuit is at most $\widetilde{O}(\|T\|_\omega^\omega)$ as claimed. □

3.3 The correction subroutine

The sampling subroutine from the previous section outputs a binary string $y \in \{0,1\}^{n_t}$ sampled from the distribution Eq. (8). The correction subroutine takes this y as input and computes a uniformly random n_t-qubit Pauli P_{cor} such that Eqs. (9,10) hold, or reports that no such Pauli exists. In the former case, the output of the correction subroutine is the binary string $z \in \{0,1\}^n$ from Eq.(10) which solves the graph state simulation problem, i.e., it is sampled from the conditional distribution obtained by measuring all qubits of the graph state $|G\rangle$ in the given Pauli bases $\{P_v\}_{v\in V}$ and postselecting on the given outcomes m for qubits in $\mathcal{P}$.

In Section 3.3.1 we provide a description of the correction subroutine, which completes our description of the classical algorithm for graph state simulation and the proof of Theorem 19. In some applications, such as the Clifford circuit simulation task described in Appendix A.2, one may be interested in the special case of graph state simulation without postselection. For this special case we provide a simplified correction subroutine with an improved runtime, in Section 3.3.2.

3.3.1 Correction subroutine: general case

There are two types of qubits for which our correction algorithm has strict requirements: postselected qubits (i.e., those qubits in the set $\mathcal{P}$) and merge ancillas. For the merge ancillas, we enforce that their measurement outcomes are 0. For postselected qubits, we enforce that their measurement outcomes coincide with some desired values specified by the input to the problem. Nevertheless, in our analysis below we will treat each type of qubit equally since our correction subroutine will not need any properties that distinguish the two cases. Our goal is to compute a uniformly random Pauli P_{cor} that stabilizes the state $(U_{\text{bases}} \otimes I)\mathcal{C}|+^{n_t}\rangle$ (up to a sign) and whose $\mathtt{X}$-component either flips or leaves unchanged all bits of y corresponding to merge ancillas and postselected qubits, depending on its desired measurement result (see Eq. (10)).

Let us represent n-qubit Pauli operators using the homomorphism mapping $\alpha\mathtt{X}^a\mathtt{Z}^b$ to the $2n$-bit vector (a,b). The phase α is not needed because it will not affect measurement. Under this homomorphism, the Pauli group maps to $\mathbb{F}_2^{2n}$, a subgroup of the Paulis maps to a linear subspace of $\mathbb{F}_2^{2n}$, and a coset of a Pauli subgroup maps to an *affine* subspace of $\mathbb{F}_2^{2n}$.

When searching for a Pauli stabilizer with particular $\mathtt{X}$- and $\mathtt{Z}$-components, we represent the desired components by a *pattern* $\pi = (\pi^{(X)}, \pi^{(Z)}) \in \{0,1,*\}^{2n}$, where $*$ indicates that the $\mathtt{X}$- or $\mathtt{Z}$-component can be either 0 or 1. We associate the pattern π with the set of all allowable outcomes

$$\Pi := \{(a,b) \in \{0,1\}^{2n} : a_i = \pi_i^{(X)} \text{ if } \pi_i^{(X)} \in \{0,1\} \text{ and } b_j = \pi_j^{(Z)} \text{ if } \pi_j^{(Z)} \in \{0,1\}\},$$

which forms an affine subspace of $\{0,1\}^{2n}$. We say that a Pauli $\mathtt{X}^a\mathtt{Z}^b$ *respects* π if $(a,b) \in \Pi$. The rest of this section is devoted to proving the following theorem:

Theorem 23. *Let a nice tree decomposition T of some n-vertex graph G be given. Let* $\mathrm{Stab} \subseteq \mathbb{F}_2^{2n_t}$ *be the linear subspace describing the stabilizer group of* $(U_{\text{bases}} \otimes I)\mathcal{C}|+^{n_t}\rangle$*, and let* $\Pi \subseteq \mathbb{F}_2^{2n_t}$ *be the affine subspace describing a given pattern* π*. There is an* $O(\|T\|_\omega^\omega)$ *time algorithm that samples a uniformly random element of the intersection* $\mathrm{Stab} \cap \Pi$*, or reports that no such element exists.*

While our algorithm will make use of the structure of the tree decomposition, most of the required operations are in fact independent of T. Consider then any n-qubit stabilizer state $|\psi\rangle$. We will define subsets of the stabilizer group of $|\psi\rangle$ that are given by affine subspaces $\mathcal{A} = \{\binom{C}{D}x + \binom{c}{d} : x \in \{0,1\}^N\} \subseteq \{0,1\}^{2n}$ for matrices $C, D \in \{0,1\}^{n\times N}$, vectors $c, d \in \{0,1\}^n$, and $N \leq 2n$ (note that if the latter condition did not hold then some of the columns of $\binom{C}{D}$ would be linearly dependent). Here, C and c correspond to the `X`-component of the stabilizer, while D and d correspond to the `Z`-component. For example, the affine subspace for the entire stabilizer group of $|\psi\rangle$ is such that each column of $\binom{C}{D}$ is the bit representation of a stabilizer generator of $|\psi\rangle$ and c, d are both zero. We say that $\mathcal{A}$ is an *affine stabilizer subspace*. For convenience, we sometimes refer to the elements of $\mathcal{A}$ directly as Pauli operators (modulo signs) using the correspondence described above.

To find the elements of an affine stabilizer subspace $\mathcal{A}$ that also respect the pattern π we must find all solutions x to

$$\binom{C}{D} x + \binom{c}{d} \in \Pi.$$

We can write this more conventionally as the rectangular system of linear equations

$$\binom{C'}{D'} x = \binom{c'}{d'} + \pi'$$

where C', D', c', d' and π' are obtained by removing row i from $\binom{C}{D}$, $\binom{c}{d}$, and π whenever $\pi_i = *$. This system can be solved using the following lemma:

Lemma 24. *Let* $E \in \{0,1\}^{k\times\ell}$ *and* $\zeta \in \{0,1\}^k$ *be given for some* k *and* ℓ*. There is an* $O(k^\omega + \ell^\omega)$*-time algorithm which determines if the system of linear equations* $Ex = \zeta$ *has a solution. If so, the algorithm reports its affine solution space, that is, a matrix* $E' \in \{0,1\}^{\ell\times\ell}$ *and vector* $\zeta' \in \{0,1\}^\ell$ *such that*

$$\{x \in \{0,1\}^\ell : Ex = \zeta\} = \{E'z + \zeta' : z \in \{0,1\}^\ell\}.$$

Proof. First construct a generalized inverse[11] E^g of E in $O(k^{\omega-1}\ell + k\ell^{\omega-1})$ time using an algorithm of Ibarra, Moran, and Hui [42]. This algorithm can be used directly if $k \leq \ell$; otherwise, if $k > \ell$, we may find a generalized inverse of E^T and then use the fact that $E^T(E^T)^g E^T = E^T$ implies $E((E^T)^g)^T E = E$. We have that $Ex = \zeta$ has a solution iff $EE^g\zeta = \zeta$, and furthermore, if a solution exists, then the solution space is given by [43]

$$\{(I - E^gE)z + E^g\zeta : z \in \{0,1\}^\ell\}. \tag{14}$$

Setting $E' := I - E^gE$ and $\zeta' := E^g\zeta$, which can be computed in time $O(k^\omega + \ell^\omega)$ and $O(k\ell)$, respectively, will give the lemma. All that remains to show is that E^gE can be

[11] A generalized inverse of a $k \times \ell$ matrix E is an $\ell \times k$ matrix E^g such that $EE^gE = E$.

computed in $O(k^\omega + \ell^\omega)$ time. If $\ell \leq k$, then the matrices E^g and E can be partitioned into $\lceil k/\ell \rceil$ $\ell \times \ell$ matrices (if ℓ doesn't evenly divide k, just pad the matrix). To compute $E^g E$, we must then compute $\lceil k/\ell \rceil$ square $\ell \times \ell$ matrix multiplications, taking time $\lceil k/\ell \rceil \ell^\omega = O(k\ell^{\omega-1})$. However, since we assumed $\ell \leq k$ this running time is also $O(k^\omega)$. A similar analysis shows that when $\ell \geq k$, the running time is $O(\ell^\omega)$. □

We arrive at the following important fact: enforcing the pattern π onto an affine stabilizer subspace returns another affine stabilizer subspace. In the following lemma, we summarize other important operations that share this property.

Lemma 25. *Let $|\psi\rangle$ be an n-qubit stabilizer state. Let $\mathcal{A} \subseteq \{0,1\}^{2n}$ be an affine stabilizer subspace representing a subset of stabilizers of $|\psi\rangle$. There exist algorithms to calculate the affine stabilizer subspace $\mathcal{A}'$ arising from each of the following operations:*

- Enforcing pattern $\pi \in \{0,1,*\}^{2n}$ on $\mathcal{A}$: *$\mathcal{A}' \subseteq \mathcal{A}$ contains exactly those Paulis that are both in $\mathcal{A}$ and respect pattern π. (Runtime $O(n^\omega)$.)*
- Applying a Clifford unitary U to $|\psi\rangle$: *$\mathcal{A}' = U\mathcal{A}U^\dagger$. (Runtime $O(n^\omega)$.)*
- Restricting $\mathcal{A}$ to qubits in some subset $S \subseteq [n]$: *$\mathcal{A}'$ is the set of Paulis $\bigotimes_{i\in S} P_i$ for which there exists a Pauli $\bigotimes_{i\in[n]} P_i$ in $\mathcal{A}$. (Runtime $O(n^\omega)$.)*

Letting $|\psi_1\rangle$ and $|\psi_2\rangle$ be n_1- and n_2-qubit stabilizer states with affine stabilizer subspaces $\mathcal{A}_1 \subseteq \{0,1\}^{2n_1}$ and $\mathcal{A}_2 \subseteq \{0,1\}^{2n_2}$, we have

- Taking tensor products: *$\mathcal{A}' = \mathcal{A}_1 \oplus \mathcal{A}_2$ contains exactly those Paulis which are tensor products of elements of $\mathcal{A}_1$ and $\mathcal{A}_2$. (Runtime $O((n_1+n_2)^2)$.)*

Proof. Let $\mathcal{A} = \{\left(\begin{smallmatrix} C \\ D \end{smallmatrix}\right)x + \left(\begin{smallmatrix} c \\ d \end{smallmatrix}\right) : x \in \{0,1\}^N\} \subseteq \{0,1\}^{2n}$ be the affine stabilizer subspace for $|\psi\rangle$. We'll consider the four operations starting with pattern enforcement. Supposing the pattern π has k non-$*$ entries, this requires solving a system of linear equations $Ex = \zeta$ where $E \in \{0,1\}^{k\times N}$ and $\zeta \in \{0,1\}^k$. By Lemma 24, the solution space is affine and can be computed in $O(n^\omega)$ time since $N \leq 2n$.

For unitary application, recall that stabilizers of $|\psi\rangle$ are mapped to stabilizers of $U|\psi\rangle$ under conjugation by U:

$$P|\psi\rangle = |\psi\rangle \iff (UPU^\dagger)U|\psi\rangle = U|\psi\rangle$$

Therefore, the affine stabilizer subspace $\mathcal{A}$ is mapped to the affine stabilizer subspace $U\mathcal{A}U^\dagger$ for the state $U|\psi\rangle$ by "conjugating" the columns of $\left(\begin{smallmatrix} C \\ D \end{smallmatrix}\right)$ and the displacement vector $\left(\begin{smallmatrix} c \\ d \end{smallmatrix}\right)$, i.e., associating each vector with a Pauli element, conjugating by U, and then reverting back to the bit representation. To do this, we simply multiply $\left(\begin{smallmatrix} C \\ D \end{smallmatrix}\right)^T$ and $\left(\begin{smallmatrix} c \\ d \end{smallmatrix}\right)^T$ by the tableau for U, which takes $O(n^\omega)$ time (this is equivalent modulo signs to the procedure described in Theorem 38).

Suppose now that we want to restrict $\mathcal{A}$ to some subset of the qubits of $|\psi\rangle$. We do this by tracing out the unwanted qubits. To trace out the ith qubit from $\mathcal{A}$, we simply remove the ith row of C, D, c, and d. After tracing out m qubits, we obtain the affine space:

$$\mathcal{A}' = \{\begin{pmatrix} C' \\ D' \end{pmatrix}x + \begin{pmatrix} c' \\ d' \end{pmatrix} : x \in \{0,1\}^N\} \subseteq \{0,1\}^{2(n-m)}$$

for matrices $C', D' \in \{0,1\}^{(n-m)\times N}$ and vectors $c', d' \in \{0,1\}^{n-m}$. Unfortunately, N may now be quite large in comparison to the total number of qubits $n-m$. If $N > 2(n-m)$

we apply an algorithm of Ibarra, Moran, and Hui to find a maximal set of independent columns of $\binom{C'}{D'}$ in time $O(n^\omega)$, so that our representation of $\mathcal{A}'$ becomes manageable once again [42].

Finally, consider the tensor product of two states $|\psi_1\rangle$ and $|\psi_2\rangle$ with affine stabilizer subspaces $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Notice that Pauli operator $P_1 \otimes P_2$ stabilizes $|\psi_1\rangle \otimes |\psi_2\rangle$ (up to signs) iff $P_1 \in \mathcal{A}_1$ and $P_2 \in \mathcal{A}_2$. In other words, the new stabilizer affine space is given by the direct product of the two previous spaces:

$$\mathcal{A}' = \mathcal{A}_1 \oplus \mathcal{A}_2 = \{ \begin{pmatrix} C_1 & 0 \\ D_1 & 0 \\ 0 & C_2 \\ 0 & D_2 \end{pmatrix} x + \begin{pmatrix} c_1 \\ d_1 \\ c_2 \\ d_2 \end{pmatrix} : x \in \{0,1\}^{N_1+N_2} \}. \tag{15}$$

□

Lemma 25 shows that tracing out qubits of a stabilizer state still results in an affine subspace. Importantly, one can check that all of the other operations (applying gates, enforcing patterns, and taking tensor products) in the lemma can also be performed on subsets of qubits of a stabilizer state using only the affine space for that restricted subset. This is due to the fact that these operations commute with the partial trace operator. Because of this, we will continue to use the phrase "affine stabilizer subspace" to refer to these restricted subspaces to emphasize their connection to stabilizers of some larger state.

We are now ready to prove the main theorem for this section (restated below):

Theorem 23. *Let a nice tree decomposition T of some n-vertex graph G be given. Let* $\mathrm{Stab} \subseteq \mathbb{F}_2^{2n_t}$ *be the linear subspace describing the stabilizer group of* $(U_{\text{bases}} \otimes I)\mathcal{C}|+^{n_t}\rangle$*, and let* $\Pi \subseteq \mathbb{F}_2^{2n_t}$ *be the affine subspace describing a given pattern π. There is an* $O(\|T\|_\omega^\omega)$ *time algorithm that samples a uniformly random element of the intersection* $\mathrm{Stab} \cap \Pi$*, or reports that no such element exists.*

Proof. Let $\mathcal{C}' = (U_{\text{bases}} \otimes I)\mathcal{C}$ for ease of notation. A naïve way to do this would be to compute Stab in its entirety, enforce π using Lemma 24, and randomly sample an element of the solution space. This $O(n_t^\omega)$-time approach is far too slow. Instead, we use the tree structure of $\mathcal{C}'$ to compute the stabilizers of $\mathcal{C}'|+^{n_t}\rangle$ while simultaneously enforcing π. In Appendix C we work through an example of the algorithm described below.

We begin by defining an affine space $\mathcal{A}_B$ for each bag B of the given tree decomposition T. Consider the subtree T_B of T rooted at B. Let us write $\mathcal{C}|_B$ and $\mathcal{C}|_{T_B}$ to denote the portions of $\mathcal{C}$ corresponding to B and T_B, respectively. For example, for the root node R of the tree decomposition in Figure 7, $\mathcal{C}|_R$ consists of three wires and three CZ gates; and for the leftmost merge node $M = \{B, C, D\}$, the circuit $\mathcal{C}|_{T_M}$ consists of six wires, four CZ gates, and one CNOT gate. Define $\mathcal{C}'|_B$ and $\mathcal{C}'_{T_B}$ analogously. Let $|\psi_B\rangle = \mathcal{C}'|_{T_B}|+^{n_{T_B}}\rangle$, where n_{T_B} is the number of qubits in $\mathcal{C}'|_{T_B}$. First consider an affine space $\mathcal{Y}_B$ which describes the stabilizers of $|\psi_B\rangle$ whose X- and Z-components agree with π on all qubits that appear *only* in $\mathcal{C}'|_{T_B}$. These are the qubits which are merged or forgotten in $\mathcal{C}'|_{T_B}$ and are therefore untouched by all operations that occur later in the circuit $\mathcal{C}'$. Now define $\mathcal{A}_B$ to be the restriction of $\mathcal{Y}_B$ to coordinates of qubits in $\mathcal{C}'|_B$. For example, $\mathcal{Y}_R = \mathrm{Stab} \cap \Pi$ and $\mathcal{A}_R \subseteq \mathbb{F}_2^6$.

We construct $\mathcal{A}_B$ inductively, working our way up the tree. Each node inherits an affine subspace from its children (unless it is a leaf node), and the first step will always be to restrict that affine subspace to the qubits of $\mathcal{C}'|_B$. For an introduce node in which

N qubits are initialized, we create the affine space $\mathcal{I} = \{\left(\begin{smallmatrix} I \\ 0 \end{smallmatrix}\right)x + \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right) : x \in \{0,1\}^N\}$ for the space of stabilizers of the $|+^N\rangle$ state. If B has a child B', then $\mathcal{A}_B = \mathcal{A}_{B'} \oplus \mathcal{I}$; otherwise, $\mathcal{A}_B = \mathcal{I}$. For a forget node, we apply the CZ gates and local operators of $\mathcal{C}'|_B$, and then enforce π on all qubits being forgotten in B. For a merge node we take the direct product of the affine spaces for each child, apply the CNOT gates of $\mathcal{C}'|_B$, and enforce π on the merged qubits.

We continue up the tree until we reach the root node R. If at any point during this process we encounter an affine space that is empty, we report that a stabilizer matching π does not exist. Otherwise by definition $\mathcal{A}_R$ contains the restriction of such a stabilizer to coordinates of qubits in $\mathcal{C}'|_R$. All that remains is to reconstruct the remaining coordinates using the other affine spaces. To do so, we process the bags in reverse order.

Let R' be the child of R. By definition, a stabilizer P_{cor} of $\mathcal{C}'\,|+^{n_t}\rangle$ respects π iff its restriction to qubits in $\mathcal{C}'|_R$ is in $\mathcal{A}_R$. With this in mind, we choose a uniformly random element (a, b) from $\mathcal{A}_R$ and set $\mathtt{X}^a\mathtt{Z}^b$ to be the tensor element of P_{cor} corresponding to qubits of R. To reconstruct the remaining components of P_{cor}, we push (a, b) backward through $\mathcal{C}'|_R$ by computing $\mathtt{X}^{a'}\mathtt{Z}^{b'} \propto (\mathcal{C}'|_R)^\dagger \mathtt{X}^a\mathtt{Z}^b(\mathcal{C}'|_R)$; randomly sample an element (a'', b'') of $\mathcal{A}_{R'}$ respecting the pattern (a', b'); and set the jth tensor element of P_{cor} to be $\mathtt{X}^{a''_j}\mathtt{Z}^{b''_j}$, for each qubit j being merged or forgotten in $\mathcal{C}'|_{R'}$. We continue in this way down the tree, always randomly sampling an element of the affine stabilizer subspace of the child node that preserves choices made higher up in the tree.

The number of qubits that we handle at any given moment is never more than the size of the bag being operated upon plus that of its children. The only operations needed to construct $\mathcal{A}_B$ are those in Lemma 25. The time needed for each of these operations is at most the size(s) of the bag(s) involved raised to the ωth power. All bags contribute to the runtime at most twice. Therefore all affine spaces can be constructed in time $O(\|T\|_\omega^\omega)$. When reconstructing P_{cor} from (a, b), the only operations needed are those from Lemma 25 together with the ability to sample from an affine subspace. The latter operation can be done in time at most the square of the corresponding bag's size, by randomly sampling a bit string of length N, where N is the subspace dimension, and then multiplying that string by a rank-N matrix and adding a displacement vector corresponding to the subspace. Each bag is processed just once during the reconstruction of P_{cor} from (a, b), and therefore this process also takes time $O(\|T\|_\omega^\omega)$.

Finally, we must show that P_{cor} is chosen uniformly from $\text{Stab} \cap \Pi$. Let us start with a general fact about sampling affine spaces. Consider some affine space $\mathcal{B} \subseteq \mathbb{F}_2^n$ and let $\mathcal{B}_S \subseteq \mathbb{F}_2^{|S|}$ be the restriction of $\mathcal{B}$ onto the coordinates in $S \subseteq [n]$. Now consider the following iterative process over subsets $S_1, S_2, \ldots$ such that $\bigcup_i S_i = [n]$: sample β_1 uniformly at random from $\mathcal{B}_{S_1}$; define the affine space $\mathcal{B}^{(1)} := \{x \in \mathcal{B} : x \text{ agrees with } \beta_1 \text{ on } S_1\}$; then sample β_2 uniformly from $\mathcal{B}^{(1)}_{S_2}$; and so on. Let β be obtained by combining all the β_i into a single string of length n. Since each β_i term was chosen such that it never conflicted with β_j for $j < i$, the choice of β is unambiguous.

We claim that β is a uniformly random element of $\mathcal{B}$. After the first iteration, we only want to sample strings of $\mathcal{B}$ where the coordinates in S_1 are restricted to β_1. As we've seen before, this imposes a linear constraint on $\mathcal{B}$, and so this new space ($\mathcal{B}^{(1)}$) is still affine by Lemma 24. In fact, we see from that lemma that the size of the solution space is independent of our choice of β_1 (provided, of course, that $\beta_1 \in \mathcal{B}_{S_1}$). In the second iteration, we sample uniformly from $\mathcal{B}_1$ restricted to the coordinates in S_2. And so on. It follows that the number of different β_j that occur with nonzero probability in the jth

step is independent of the specific choices of $\beta_1, \ldots, \beta_{j-1}$. Therefore, the final string β chosen from $\mathcal{B}$ has the same probability of being chosen as any other string $\beta' \in \mathcal{B}$.

We now only have left to show that this is actually what is occurring in our correction protocol. At the root node, we sample uniformly from the affine space $\mathcal{A}_R$ which, by construction, is equal to Stab $\cap\, \Pi$ restricted to qubits which are forgotten in the root bag. Further down in the tree, when nodes are forgotten or merged, we only sample strings which are consistent with measurements made higher up in the tree. Recall that the qubits that are measured in a particular bag are never touched again higher in the tree. Therefore, the affine space for a bag B (i.e., the space $\mathcal{A}_B$) restricted to the qubits that are measured in that bag is equal to the affine space for the entire circuit Stab $\cap\, \Pi$ restricted to the qubits that are measured in that bag. For the qubits in $\mathcal{A}_B$ that are not measured, we restrict them to the measurement results sampled higher in the tree. In some cases, these measurement results are obtained *after* applying some unitary. However, the application of that unitary is equivalent to applying an invertible linear transformation on the coordinates of $\mathcal{A}_B$ corresponding to unmeasured qubits. Applying the inverse of this transformation to the outcome sampled higher in the tree gives the unique string over the unmeasured qubits of $\mathcal{A}_B$ consistent with higher measurements. Since we sample uniformly from $\mathcal{A}_B$ restricted to this string, the theorem follows. □

We close this subsection with the observation that Theorem 23 can be used to solve symmetric linear systems modulo 2. That is, suppose we are given a system $Ay = b$ over $\mathbb{F}_2$ where $A \in \mathbb{F}_2^{n \times n}$ is the (symmetric) adjacency matrix of a graph G, and we are given a tree decomposition T for the graph G. We claim that $Ay = b$ has a solution if and only if $\mathcal{C}\,|{+}^{n_t}\rangle$ has a stabilizer Pauli respecting the pattern $(*^n\, 0^{n_a}, b\, *^{n_a})$, and moreover the solution y can be read off of the stabilizer. This yields a fast algorithm for planar or limited-treewidth linear systems.

Theorem 26. *There exists an algorithm to find a uniformly random solution to $Ay = b$ (or report that no solution exists) in time $O(\|T\|_\omega^\omega)$ given a graph G with adjacency matrix $A \in \mathbb{F}_2^{n \times n}$, a vector $b \in \mathbb{F}_2^n$, and a nice tree decomposition T for G.*

Conversely, we also claim such an algorithm for solving linear systems implies the correction procedure in Theorem 23. See Appendix D for the proof of Theorem 26 and the full details of this equivalence.

In Section 4 we show that, given a planar graph, we can compute a nice tree decomposition T in $O(n \log n)$ time, such that $\|T\|_\omega^\omega = \tilde{O}(n^{\omega/2})$ (see Lemmas 31, 32). Thus we obtain Theorem 4 from the introduction as a corollary.

We believe the new linear system solver stated in Theorem 26 may be of independent interest because, to our knowledge, existing solvers exploiting planarity and/or treewidth–such as Alon and Yuster [28] or Lipton, Rose, and Tarjan [27]–require A to have full rank. Our algorithm makes no such assumption; in fact, it can sample a uniformly random solution when the solution is not unique.

3.3.2 Simplified correction subroutine for graph state simulation without postselection

The correction procedure for the graph state simulation problem *without postselection* can be simplified to pushing Paulis backwards and forwards through the circuit, removing the need for explicit linear system solving. This improves the runtime to linear in the size of the circuit (i.e., $O(|E| + n_t)$).

Theorem 27. *The state $\mathcal{C}|+^{n_t}\rangle$ has stabilizers with arbitrary $\mathtt{X}$-component. Moreover, given a desired $\mathtt{X}$-component vector $a \in \{0,1\}^{n_t}$, there is an $O(|E|+n_t)$-time algorithm that outputs a stabilizer $Q(a) = \mathtt{X}^a\mathtt{Z}^b$ (up to sign) for some $b \in \{0,1\}^{n_t}$.*

Proof. Since $Q(a) = \mathtt{X}^a\mathtt{Z}^b$ is a stabilizer for $\mathcal{C}\,|+^{n_t}\rangle$ (up to sign), it follows that $\mathcal{C}^\dagger\mathtt{X}^a\mathtt{Z}^b\mathcal{C}$ stabilizes $|+^{n_t}\rangle$ (up to sign), and is therefore of the form $\pm\mathtt{X}^{a'}$. That is,

$$\mathcal{C}^\dagger Q(a)\mathcal{C} = \mathcal{C}^\dagger\mathtt{X}^a\mathtt{Z}^b\mathcal{C} = \pm\mathtt{X}^{a'} \tag{16}$$

for some $a' \in \{0,1\}^{n_t}$.

Recall that $\mathcal{C}$ is composed of CZ and CNOT gates, both of which normalize the group generated by $\mathtt{Z}$-Paulis,

$$\mathcal{Z} := \{\mathtt{Z}^{b'} : b' \in \{0,1\}^{n_t}\}.$$

It follows that we can mod out Eq. (16) by $\mathcal{Z}$ to get $\pm\mathtt{X}^{a'} \equiv \mathcal{C}^\dagger\mathtt{X}^a\mathcal{C} \pmod{\mathcal{Z}}$. We compute $\mathcal{C}^\dagger\mathtt{X}^a\mathcal{C}$ by commuting $\mathtt{X}^a$ backwards through $\mathcal{C}$, one gate at a time. This gives us a Pauli with an irrelevant $\mathtt{Z}$-component (and sign) times $\mathtt{X}^{a'}$. Finally, we push $\mathtt{X}^{a'}$ *forwards* through $\mathcal{C}$ to obtain $\mathcal{C}\mathtt{X}^{a'}\mathcal{C}^\dagger$ which is the desired stabilizer $Q(a)$ up to an overall ± 1 sign.

The cost of the algorithm is in commuting $\mathtt{X}^a$ and $\mathtt{X}^{a'}$ through the circuit, which involves updating a $2n_t$-bit string once for each of the $|E|+n_a$ gates, leading to an overall runtime of $O(|E|+n_t)$. □

In particular, the Pauli P_{cor} that corrects the output $y \in \{0,1\}^{n_t}$ of the sampling routine is computed as

$$P_{\text{cor}} = (U_{\text{bases}} \otimes I)\, Q(y) \left(U_{\text{bases}}^\dagger \otimes I\right). \tag{17}$$

Since $Q(y)$ stabilizes $\mathcal{C}|+^{n_t}\rangle$, we get that P_{cor} stabilizes $(U_{\text{bases}} \otimes I)\mathcal{C}|+^{n_t}\rangle$, that is, P_{cor} satisfies Eq. (9). Furthermore, notice that the $\mathtt{X}$-component of P_{cor} on the merge ancillas is equal to y on the merge ancillas, that is, P_{cor} satisfies Eq. (10). Therefore, P_{cor} is a valid correcting Pauli, computable in $O(|E|+n_t)$ time.

Unlike the more general postselection procedure, we have not generated a uniformly random correcting Pauli. Of course, the measurement outcome $z \in \{0,1\}^n$ obtained in this way is still valid. So, by Claim 9, we can obtain a uniformly random sample in $O(n)$ time by choosing a random stabilizer R of the state $|\psi\rangle = U_{\text{bases}}|G\rangle$. A random stabilizer R of ψ can be computed straightforwardly in time $O(n+|E|)$.[12]

4 Improved algorithm for planar graphs

In this section we describe an improved algorithm for the graph state simulation problem in the special case of planar graphs. In particular, we prove Theorem 3 and its slight generalization Theorem 8, restated below.

Theorem 3. *There is an $\widetilde{O}(n^{\omega/2})$-time classical algorithm which solves the graph state simulation problem for planar graphs.*

[12]Note that ψ has a stabilizer generator for each vertex v of G, with Pauli weight d_v+1. We obtain R by choosing a random subset of vertices and multiplying the corresponding generators, which takes time $O(\sum_{v\in V}(d_v+1)) = O(n+|E|)$.

Theorem 8 (Extension of Theorem 3)**.** *Let G' be a graph with n vertices and let G be a planar graph. Suppose we are given an r-coarse-graining from G' to G. There is a classical algorithm with runtime upper bounded as $\widetilde{O}(n^{\omega/2}r^{\omega})$ which solves the graph state simulation problem for G'.*

The above results are obtained as a consequence of our more general algorithm for graph state simulation (Theorem 19). That result allows us to solve the graph state simulation problem for G, given a tree decomposition T of G, using a runtime $\widetilde{O}(\|T\|_{\omega}^{\omega})$. It is known that planar graphs have treewidth $t = O(\sqrt{n})$, and there are linear-time algorithms to construct a width-$O(\sqrt{n})$ tree decomposition of a planar graph. This fact, combined with Theorem 19, leads directly to a worst-case $\widetilde{O}(nt^{\omega-1}) = \widetilde{O}(n^{(\omega+1)/2})$ algorithm for planar graph state simulation. In this section we present a more careful analysis which shows that on the tree decompositions that we construct (recursively), the algorithm from Theorem 19 actually runs in $O(n^{w'/2})$ for any $w' > w$. The intuition is that any subgraph of a planar graph is again planar, so an m-vertex subgraph of an n-vertex graph has treewidth $O(\sqrt{m})$, not $O(\sqrt{n})$. Thus, we expect bags to get smaller as we descend the tree. This leads to a better upper bound on the runtime $\widetilde{O}(\|T\|_{\omega}^{\omega})$.

We shall use tree decompositions which are constructed recursively from planar graph *separators*.

Definition 28. Given a graph $G = (V, E)$, a *separator* is a subset of the vertices $S \subseteq V$ which divides the remaining vertices $V \backslash S$ into two disconnected parts, A and B.

It is known that any planar graph has a separator of size $|S| = O(\sqrt{n})$, and Lipton and Tarjan showed how to find one in linear time.

Theorem 29 (Lipton and Tarjan [29])**.** *Given an n-vertex planar graph G, there is an $O(n)$-time deterministic algorithm to compute a separator S of size $|S| = \beta\sqrt{n}$ that divides the remaining vertices into two disconnected parts A and B, of size $|A|, |B| \leq \alpha n$, where $\alpha = \frac{2}{3}$ and $\beta = 2\sqrt{2}$.*

It is well known that one can compute a tree decomposition of width $O(\sqrt{n})$ using a runtime $O(n \log n)$ by recursively applying these separators, as in Algorithm 2 (ComputeTD). This can be improved to linear time with clever data structures [44], though we will not use this improvement here.

The rest of this section is devoted to analyzing the runtime of ComputeTD, the properties of the tree it creates, and the performance of our graph state simulation algorithm when given this tree.

We begin by verifying that ComputeTD is correct, and upper bounding the width of its output.

Lemma 30. *Given a planar graph G and a subset U of its vertices, the tree T computed by ComputeTD(G, U) is a tree decomposition of G with root bag U and width $O(|U| + \sqrt{n})$.*

Proof. We claim by induction that ComputeTD(G, U) produces a valid tree decomposition where the root bag is U. We leave the base case for the reader to check. In the recursive case, the tree decomposition has the form depicted in Figure 10 and we see that

- every vertex appears in some bag, since every vertex belongs to $A \cup S$ or $B \cup S$, and by induction the two subtrees contain those vertices,

Algorithm 2 Separator to Tree Decomposition Algorithm.
Makes use of a subroutine SEPARATOR which takes as input a planar graph and outputs a planar separator with parameters α, β.

Input: Graph G, subset $U \subseteq V(G)$
Output: A tree decomposition of G with U contained in the root bag

$n_0 \leftarrow \left\lceil \frac{\beta^2}{(1-\alpha)^2} \right\rceil$
function COMPUTETD(G, U)
    if $|V(G)| \leq n_0$ **then**
        return introduce node $V(G)$ below a forget node U
    else
        $(A, S, B) \leftarrow$ SEPARATOR(G)
        $T_A \leftarrow$ COMPUTETD($G|_{A \cup S}$, $S \cup (A \cap U)$)
        $T_B \leftarrow$ COMPUTETD($G|_{B \cup S}$, $S \cup (B \cap U)$)
        return forget node U above a merge node $S \cup U$ with children T_A and T_B

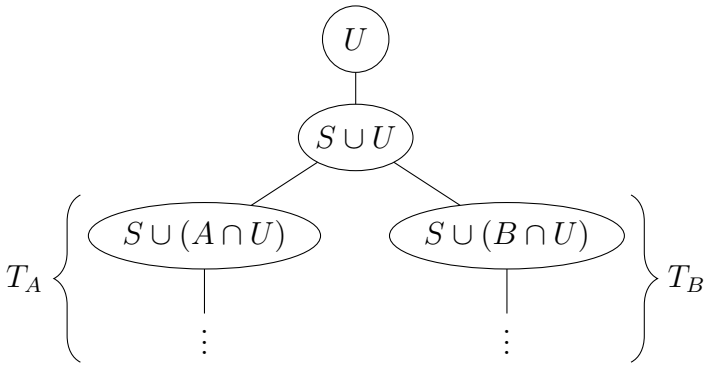


Figure 10: The tree decomposition constructed by COMPUTETD(G,U) in the case where $|V(G)| \geq n_0$.

- every edge appears in some bag, since all the edges in $A \cup S$ appear in T_A, all the edges in $B \cup S$ appear in T_B, and there are no edges between A and B because of the separator property,
- nodes containing $v \in V$ form a connected component because
  - they are connected in T_A and T_B,
  - if v appears in both T_A and T_B, then it follows that $v \in S$ and appears in the merge node above T_A and T_B, and
  - if v belongs to the merge node $S \cup U$, then it also appears at the root of T_A (or T_B) if it appears in that subtree at all.

It is also clear by construction that U is the root node of the new tree, finishing the induction.

Notice that the recursive calls are on graphs $G|_{A\cup S}$ and $G|_{B\cup S}$ of size

$$|A \cup S|, |B \cup S| \leq \alpha n + \beta\sqrt{n} \leq \gamma n,$$

for some γ such that

$$\gamma < \alpha + \frac{\beta}{\sqrt{n_0}} \leq 1, \tag{18}$$

where $n_0 = \frac{\beta^2}{(1-\alpha)^2}$ (as specified in the algorithm) guarantees $\gamma < 1$. It follows that the graph input to a recursive call at depth ℓ will have at most $\gamma^\ell n$ vertices.

The bag size can only increase from parent to child by at most $|S|$, so a bag in the ℓth level of recursion has size which has increased by at most the sizes of ℓ planar separators. Since the size of each planar separator is at most β times the square root of the number of vertices in the graph, the number of elements gained from ℓ levels of separators is therefore at most

$$|\mathrm{Bag}\backslash U| \leq \sum_{j=0}^{\ell} \beta\sqrt{n}\gamma^{j/2} \leq \beta\sqrt{n}\sum_{j=0}^{\infty}\gamma^{j/2} = \frac{\beta\sqrt{n}}{1-\sqrt{\gamma}} = O(\sqrt{n}).$$

We conclude that every bag has at most $O(|U| + \sqrt{n})$ vertices. □

We now bound the runtime $\widetilde{O}(\|T\|_\omega^\omega)$ which appears in Theorem 19, for tree decompositions T produced by ComputeTD.

Lemma 31. *Let G be a planar graph with n vertices, let $U \subseteq V(G)$, and let T be the tree decomposition computed by ComputeTD(G, U). For all $c > 2$ we have*

$$\|T\|_c^c = \widetilde{O}(n^{c/2} + |U|^c).$$

Proof. It is a property of p-norms that $\|T\|_p \leq \|T\|_q$ whenever $p \geq q > 0$. In particular, since $c > 2$ we have $\|T\|_c \leq \|T\|_2$. It suffices to show that $\|T\|_2^2 = \widetilde{O}((\sqrt{n} + |U|)^2)$, because then

$$\|T\|_c^c \leq \|T\|_2^c = \left(\|T\|_2^2\right)^{c/2} = \widetilde{O}((\sqrt{n} + |U|)^c) = \widetilde{O}(n^{c/2} + |U|^c),$$

as desired.

The crux of the proof is to reduce to the case where $U = \varnothing$. That is, consider the tree $T' \leftarrow \textsc{ComputeTD}(G, \varnothing)$ that we get by passing $\varnothing$ instead of U as the second argument. The tree structure T and T' will be identical except for the content of the nodes, since they will get the same separators, recurse on the same graphs, and stop at the same base cases (in other words, the control flow of ComputeTD does not depend on U). However, each node in T' is contained in the corresponding node of T, i.e., some vertices of nodes in T are missing from some nodes T'. We say T and T' *disagree* on these vertices at these nodes.

We will upper bound $\|T\|_2^2 - \|T'\|_2^2$ as

$$\begin{aligned}\|T\|_2^2 - \|T'\|_2^2 = \sum_i |B_i|^2 - |B_i'|^2 &= \sum_i (|B_i| - |B_i'|)\,(|B_i| + |B_i'|) \\ &\leq \left(\sum_i |B_i| - |B_i'|\right)(\max_j |B_j| + \max_j |B_j'|).\end{aligned}$$

Recall from Lemma 30 that the size of each bag is at most $O(|U| + \sqrt{n})$, which upper bounds the second factor in the product above. It remains to bound the first factor (essentially the total number of missing vertices across all nodes), and use the resulting bound on $\|T\|_2^2 - \|T'\|_2^2$ to draw a conclusion about $\|T\|_2^2$.

Note that T and T' will disagree on a vertex $u \in U$ at the root, and continue to disagree as we descend the tree until either

- the separator divides u from T, i.e., the current subtree corresponds to a subgraph of G that does not include u anymore, or
- u is in the separator at some level of the recursion, and since T and T' have the same separators, it will appear in both trees.

In either case, if T and T' agree on a vertex u at node X, then they also agree on u in all descendants of X. Furthermore, T and T' cannot disagree about u in both children, Y_1 and Y_2, of some node X. Either u belongs to the separator at node X, and thus appears in both children in both T and T', *or* u is in one half of the partition and thus absent from the other half in both T and T'. The take away is that $u \in U$ can be missing from at most one node per level of the tree. Otherwise, if u is missing from nodes Z_1 and Z_2 in the same level, then it is also missing from their ancestors (since if T and T' agree on an ancestor of Z_i, they agree on all its descendants, including Z_i), and in particular from their least common ancestor Z and its two children. Since T and T' cannot disagree on both children, this is a contradiction.

We conclude that any particular $u \in U$ is missing from at most $O(\text{depth}) = O(\log n)$ nodes, and so there are a total of $O(|U| \log n)$ missing nodes across the whole tree. That is, $\sum_i |B_i| - |B_i'| = O(|U| \log n)$, and thus

$$\|T\|_2^2 - \|T'\|_2^2 = O(|U|(|U| + \sqrt{n}) \log n).$$

We will now show that $\|T'\|_2^2 = \widetilde{O}(n)$ to complete the proof.

Let $f(n)$ be the maximum value of $\|T'\|_2^2$ for a tree $T' \leftarrow \textsc{ComputeTD}(G, \varnothing)$ over all graphs G on at most n vertices. In the recursive case (where $n > n_0$), we have two recursive calls which are unfortunately *not* of the form $\textsc{ComputeTD}(G', \varnothing)$; they have non-empty second arguments. However, by the same argument as above, setting the

second argument from U' to $\varnothing$ changes the squared 2-norm by at most $O(|U'|(|U'| + \sqrt{n})\log n) = O(n \log n)$, since here $|U'| \leq |S| = O(\sqrt{n})$. It follows that we can upper bound $f(n)$ by $f(n_1) + O(n \log n)$ and $f(n_2) + O(n \log n)$ for the two subtrees, plus $O(|S|^2) = O(n)$ for the root node, giving

$$f(n) \leq f(n_1) + f(n_2) + O(n \log n), \tag{19}$$

where $n_1, n_2 \leq \alpha n + \beta\sqrt{n}$ and $n_1 + n_2 \leq n + \beta\sqrt{n}$. We show by induction that $f(n) \leq kn \log^2 n$ for all $n \geq 2$, where the constant k is to be determined.[13] As long as $n \geq 2/\alpha$, both parts will be size $\geq \alpha n \geq 2$, so we can apply the induction hypothesis. Let us take k large enough that the induction hypothesis holds for the remainder, $2 \leq n < 2/\alpha$. Then taking γ to be the constant from the $O(n \log n)$ term in Equation (19), we have

$$\begin{aligned} f(n) &\leq f(n_1) + f(n_2) + \gamma n \log n \\ &\leq k(n_1 \log^2 n_1 + n_2 \log^2 n_2) + \gamma n \log n \\ &\leq k(n_1 + n_2) \log^2 \max(n_1, n_2) + \gamma n \log n \\ &\leq k(n + \beta\sqrt{n}) \log^2 \max(n_1, n_2) + \gamma n \log n \\ &= kn \log^2 \max(n_1, n_2) + k\beta\sqrt{n} \log^2 \max(n_1, n_2) + \gamma n \log n. \end{aligned}$$

Clearly $\max(n_1, n_2) \leq n$, and $\sqrt{n} \log^2 n = o(n \log n)$, so for any $\beta > 0$ and $\epsilon > 0$ we can take N large enough that

$$\beta\sqrt{n} \log^2 \max(n_1, n_2) \leq \epsilon n \log n$$

for all $n > N$. For the other term, we note that $n_1, n_2 \leq \alpha n + \beta\sqrt{n} \leq \alpha' n$ for some constant $\alpha < \alpha' < 1$ when n is sufficiently large.

$$\begin{aligned} f(n) &\leq kn(\log n)(\log n + \log \alpha') + (\gamma + k\epsilon)n \log n \\ &= kn \log^2 n + (\gamma + k\epsilon + k \log \alpha')n \log n \end{aligned}$$

We wish to show this is at most $kn \log^2 n$. At this point we fix α', assume N is large enough that $\epsilon < \log(1/\alpha')$, and then take any $k > \frac{\gamma}{\log(1/\alpha) - \epsilon}$ so that the second term is negative, and hence $f(n) \leq kn \log^2 n$. This completes the induction, and we conclude that $f(n) = O(n \log^2 n)$. □

The running time for the algorithm follows immediately, as long as ComputeTD is not too expensive. Let us bound that next.

Lemma 32. *Let G be a planar graph with n vertices. ComputeTD(G, U) runs in time $O(n \log n)$ given that Separator runs in linear time.*

Proof. Let $F(n)$ be the worst-case running time of ComputeTD on graphs of at most n vertices. Using the recursive structure of the algorithm, we see that for sufficiently large n,

$$f(n) \leq f(n_1) + f(n_2) + O(n)$$

for some n_1, n_2 such that $n_1 + n_2 \leq n + \beta\sqrt{n}$ and $n_1, n_2 \leq \alpha n + \beta\sqrt{n}$. Similar to the previous lemma, an inductive argument shows this recurrence is solved by $f(n) = O(n \log n)$. □

[13] The base of the logarithm is irrelevant since it can be absorbed into k.

We are now ready to prove Theorem 3.

Proof of Theorem 3. We will apply the algorithm from Theorem 19 to the tree T output by $\textsc{ComputeTD}(G, \varnothing)$. The graph state simulation algorithm runs in $\widetilde{O}(\|T\|_\omega^\omega)$ by Theorem 19, and $\|T\|_\omega^\omega$ is $\widetilde{O}(n^{\omega/2})$ by Lemma 31. The cost of constructing the tree decomposition is $O(n \log n)$ by Lemma 32. Thus, the total running time is $\widetilde{O}(n^{\omega/2})$. □

We also immediately get Theorem 8.

Proof of Theorem 8. Let $\varphi\colon G' \to G$ be the mapping witnessing that G' is r-coarse-grained planar. That is, G is planar and φ maps at most r vertices in G' to one vertex in G. To solve an instance of the graph state simulation problem for G', we run the following steps:

1. Construct $T_G \leftarrow \textsc{ComputeTD}(G, \varnothing)$, using the planar tree decomposition.
2. Take the preimage $T_{G'} \leftarrow \varphi^{-1}(T_G)$. That is, let $T_{G'}$ have the same tree structure, and replace a bag $B \subseteq V(G)$ with the preimage set $\varphi^{-1}(B) := \{v \in V(G') : \varphi(v) \in B\}$.
3. Run the graph state simulation algorithm from Theorem 19 for G' with tree decomposition $T_{G'}$.

Note that $T_{G'}$ is a tree decomposition for G' by Lemma 18, and therefore the algorithm is correct.

The running time will be dominated by Step 3 since constructing T_G is $O(n \log n)$ time by Lemma 32, and computing preimages is $O(n)$ time. Recall that the graph state simulation algorithm runs in time $O(\|T_{G'}\|_\omega^\omega)$ by Theorem 19. Each bag in $T_{G'}$ is at most r times bigger than T_G, so it is clear that $\|T_{G'}\|_\omega^\omega \leq r^\omega \|T_G\|_\omega^\omega$. Finally, $\|T_G\|_\omega^\omega = \widetilde{O}(n^{\omega/2})$ by Lemma 31. Putting these facts together, we see that the sampling algorithm is $\widetilde{O}(n^{\omega/2} r^\omega)$ time. □

For example, consider the $k \times k \times k$ cubic grid G'. There is clearly a k-coarse graining onto the $k \times k$ grid G. Since the $k \times k$ grid is only $n = k^2$ qubits, sampling only takes $\widetilde{O}((k^2)^{\omega/2} k^\omega) = \widetilde{O}(k^{2\omega})$ time. This matches (up to log factors) the natural analogue of the recursive 2D grid algorithm, and beats the $\widetilde{O}(k^{3\omega})$ runtime of direct Clifford simulation.

5 Stabilizer operations in matrix multiplication time

The goal of this section is to show that fast matrix multiplication techniques can be used in a wide range of stabilizer operations. These speedups are shown in Table 1, where $\omega < 2.372859$ [23] is the matrix multiplication constant. In particular, we will show that given the tableaux for Clifford operations on n and m qubits, the tableau for their composition can be computed in $O(\min(nm^{\omega-1}, mn^{\omega-1}))$ time. Furthermore, given the tableau for a Clifford state on n qubits, we simulate a measurement on k qubits in $O(k^{\omega-2} n^2 \log(2n/k))$ time.

Most of the results in this section follow from the work of Dehaene and De Moor [45] which describes the composition of Clifford operations as the product of binary matrices.

We begin with a short summary of their techniques. First, recall that every Pauli P can be represented uniquely as[14]

$$i^{\alpha}(-1)^{\beta}\mathtt{X}^{a}\mathtt{Z}^{b}$$

for $\alpha, \beta \in \{0,1\}$ and $a, b \in \{0,1\}^n$.

Lemma 33 (Dehaene, De Moor [45])**.** *Let* $P_1 = i^{\alpha_1}(-1)^{\beta_1}\mathtt{X}^{a_1}\mathtt{Z}^{b_1}$ *and* $P_2 = i^{\alpha_2}(-1)^{\beta_2}\mathtt{X}^{a_2}\mathtt{Z}^{b_2}$. *Then* $P_1P_2 = i^{\alpha_{12}}(-1)^{\beta_{12}}\mathtt{X}^{a_{12}}\mathtt{Z}^{b_{12}}$ *where*

$$\begin{aligned} \alpha_{12} &= \alpha_1 + \alpha_2 & a_{12} &= a_1 + a_2 \\ \beta_{12} &= \beta_1 + \beta_2 + \alpha_1\alpha_2 + b_1 \cdot a_2 & b_{12} &= b_1 + b_2 \end{aligned}$$

where all operations are over $\mathbb{F}_2$.

Proof. We start by noticing that $i^{\alpha_1+\alpha_2} = i^{\alpha_1 \oplus \alpha_2}(-1)^{\alpha_1\alpha_2}$ when $\alpha_1, \alpha_2 \in \{0,1\}$. Then

$$\begin{aligned} P_1P_2 &= i^{\alpha_1\oplus\alpha_2}(-1)^{\beta_1+\beta_2+\alpha_1\alpha_2}\mathtt{X}^{a_1}\mathtt{Z}^{b_1}\mathtt{X}^{a_2}\mathtt{Z}^{b_2} \\ &= i^{\alpha_1\oplus\alpha_2}(-1)^{\beta_1+\beta_2+\alpha_1\alpha_2+b_1\cdot a_2}\mathtt{X}^{a_1}\mathtt{X}^{a_2}\mathtt{Z}^{b_1}\mathtt{Z}^{b_2} \\ &= i^{\alpha_1\oplus\alpha_2}(-1)^{\beta_1+\beta_2+\alpha_1\alpha_2+b_1\cdot a_2}\mathtt{X}^{a_1\oplus a_2}\mathtt{Z}^{b_1\oplus b_2}, \end{aligned}$$

which gives the desired result. □

Corollary 34. *Let* $P_i = i^{\alpha_i}(-1)^{\beta_i}\mathtt{X}^{a_i}\mathtt{Z}^{b_i}$ *for* $i \in \{1, \ldots, \ell\}$. *Then,* $P_1 \cdots P_\ell = i^{\alpha}(-1)^{\beta}\mathtt{X}^{a}\mathtt{Z}^{b}$ *where*

$$\begin{aligned} \alpha &= \sum_{i=1}^{\ell} \alpha_i & a &= \sum_{i=1}^{\ell} a_i \\ \beta &= \sum_{i=1}^{\ell} \beta_i + \sum_{i<j}^{\ell} \alpha_i\alpha_j + \sum_{i<j}^{\ell} b_i \cdot a_j & b &= \sum_{i=1}^{\ell} b_i \end{aligned}$$

where all operations are over $\mathbb{F}_2$.

Proof. Repeatedly apply Lemma 33. □

Let us now fix our tableau representation for an n-qubit Clifford circuit Q.[15] The tableau is simply a list of the images of $\mathtt{X}^{e_j}$ and $\mathtt{Z}^{e_j}$ under conjugation by Q for all j, where $e_j \in \{0,1\}^n$ is the n-bit vector with a 1 in position j and 0's everywhere else. Let $Q\mathtt{X}^{e_j}Q^\dagger = i^{\alpha_j}(-1)^{\beta_j}\mathtt{X}^{a_j}\mathtt{Z}^{b_j}$ and $Q\mathtt{Z}^{e_j}Q^\dagger = i^{\delta_j}(-1)^{\epsilon_j}\mathtt{X}^{c_j}\mathtt{Z}^{d_j}$. We arrange this conjugation information into a $2n \times 2n$ binary matrix M, a phase vector $p \in \{0,1\}^{2n}$, and a sign vector $s \in \{0,1\}^{2n}$:

$$M = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \quad p = \left[\begin{array}{c} \alpha \\ \hline \delta \end{array}\right] \quad s = \left[\begin{array}{c} \beta \\ \hline \epsilon \end{array}\right]$$

where $A = \{a_{i,j}\}$, $B = \{b_{i,j}\}$, $C = \{c_{i,j}\}$, and $D = \{d_{i,j}\}$ are the $n \times n$ binary matrices whose elements correspond to the Pauli decompositions. That is, a_i is the ith row of A, and so on.

[14] For the purposes of this section, we will use a separate font to write $\mathtt{X}$, $\mathtt{Y}$, and $\mathtt{Z}$ Pauli operators to avoid confusion with the many other matrices that will appear.

[15] We will mostly adopt the conventions of the Aaronson-Gottesman tableau representation [17], but will keep track of the phase bits according to Dehaene and De Moor [45].

Fact 35 (e.g., [45])**.** *M, p, and s correspond to a valid Clifford unitary iff*

- *M is an element of the binary symplectic group: $M^T \Omega M = \Omega$ where $\Omega = \left(\begin{smallmatrix} 0 & I \\ I & 0 \end{smallmatrix}\right)$;*
- *$p = \mathrm{diag}(M \Lambda M^T)$ where $\Lambda = \left(\begin{smallmatrix} 0 & I \\ 0 & 0 \end{smallmatrix}\right)$.*

As you can see from the above fact, we don't actually need to store p since it is encoded into M, but it will helpful to keep it around anyway.

Let us now turn to the computational question of how much time is required to perform various Clifford operations. Since an n-qubit state is represented by $O(n^2)$ many bits, it will often be computationally inefficient to write out an entirely new tableau resulting from a Clifford operation. For this reason, we will adopt the conventional view of Clifford operations as updating the existing tableau (e.g. modifying entries and appending rows/columns). Therefore, operations such as composing a single-qubit gate with an n-qubit Clifford circuit only require $O(n)$ time if the tableau for the n-qubit Clifford is given as input (see Section 5.1 for a detailed explanation of composition).

We conclude this section with a simple fact that tensor products of Clifford operations are represented by the direct sum of their tableau:

Fact 36. *Let (M_1, p_1, s_1) and (M_2, p_2, s_2) be the tableaux for Clifford operations Q_1 and Q_2, respectively. Then $Q_1 \otimes Q_2$ is represented by tableau*

$$M_{12} = \left[\begin{array}{cc|cc} A_1 & 0 & B_1 & 0 \\ 0 & A_2 & 0 & B_2 \\ \hline C_1 & 0 & D_1 & 0 \\ 0 & C_2 & 0 & D_2 \end{array}\right] \qquad p_{12} = \left[\begin{array}{c} \alpha_1 \\ \alpha_2 \\ \hline \delta_1 \\ \delta_2 \end{array}\right] \qquad s_{12} = \left[\begin{array}{c} \beta_1 \\ \beta_2 \\ \hline \epsilon_1 \\ \epsilon_2 \end{array}\right]$$

Proof. For Pauli operators P_1 and P_2, we have

$$(Q_1 \otimes Q_2)(P_1 \otimes P_2)(Q_1^\dagger \otimes Q_2^\dagger) = Q_1 P_1 Q_1^\dagger \otimes Q_2 P_2 Q_2^\dagger.$$

The fact follows by setting $P_1 \otimes P_2$ to be $\mathtt{X}^{e_j}$ (for the destabilizers) and $\mathtt{Z}^{e_j}$ (for the stabilizers). For example, the first row of the tableau comes from setting $P_1 = \mathtt{X}^{e_1}$ and $P_2 = I$. □

Therefore, we get that appending a single-qubit stabilizer state to an n-qubit stabilizer state takes time $O(n)$ and appending an ℓ-qubit state takes time $O((n+\ell)^2)$.

5.1 Composition

This section is dedicated to showing fast matrix multiplication speedups for the composition of Clifford operations. Before we do so, let us first describe the action of a single Clifford operation on a Pauli string. For any square matrix A, we will define $\mathrm{lower}(A)$ to be the strictly lower triangular matrix of A (i.e., with the diagonal and above diagonal entries replaced with 0).

Lemma 37 (Dehaene, De Moor [45])**.** *Let Q be an n-qubit Clifford operation represented by tableau (M, p, s). Let $P := i^\alpha (-1)^\beta \mathtt{X}^a \mathtt{Z}^b$ be a Pauli where $a \in \{0,1\}^n$ and $b \in \{0,1\}^n$ form the $2n$-bit row vector $c = [a \;\; b]$. We have $QPQ^\dagger = i^{\alpha'} (-1)^{\beta'} X^{a'} Z^{b'}$ with $c' = [a' \;\; b'] = cM$, $\alpha' = \alpha + c \cdot p$, and*

$$\beta' = \beta + \alpha c \cdot p + c \cdot s + c\,\mathrm{lower}(pp^T + M \Lambda M^T) c^T$$

Proof. Once again let $M = \left[\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right]$. We will write M_i for the ith row of matrix M (and similarly for the submatrices A, B, C, and D). Notice that the inner products between the $\mathtt{X}$ and $\mathtt{Z}$ parts of M can be expressed with the $\Lambda = \left(\begin{smallmatrix} 0 & I \\ 0 & 0 \end{smallmatrix}\right)$ operator. For example, $A_i \cdot B_j = M_i \Lambda M_j^T$ for $i, j \in \{1, \ldots, n\}$. Ignoring the phase for the moment, we have

$$\begin{aligned} Q\mathtt{X}^a\mathtt{Z}^bQ^\dagger &= \prod_{j:a_j=1} Q\mathtt{X}^{e_j}Q^\dagger \prod_{j:b_j=1} Q\mathtt{Z}^{e_j}Q^\dagger \\ &= \prod_{j:a_j=1} i^{p_j}(-1)^{s_j}\mathtt{X}^{A_j}\mathtt{Z}^{B_j} \prod_{j:b_j=1} i^{p_{j+n}}(-1)^{s_{j+n}}\mathtt{X}^{C_j}\mathtt{Z}^{D_j} \\ &= i^{\alpha'}(-1)^{\beta'}\mathtt{X}^{a'}\mathtt{Z}^{b'}. \end{aligned}$$

That is, $Q\mathtt{X}^a\mathtt{Z}^bQ^\dagger$ is simply the product of the Pauli elements indexed by the rows of the tableau, so by Corollary 34 (there are $\ell := 2n$ such Pauli operators) we get

$$\begin{aligned} \alpha' &= \sum_{j=1}^{n} a_j p_j + \sum_{j=1}^{n} b_j p_{j+n} = \sum_{j=1}^{n} c_j p_j + \sum_{j=1}^{n} c_{j+n} p_{j+n} = c \cdot p \\ \beta' &= \sum_{j=1}^{2n} c_j s_j + \sum_{i<j}^{2n} c_i c_j p_i p_j + \sum_{i<j}^{2n} c_i c_j M_i \Lambda M_j^T \\ &= c \cdot s + c\,\mathrm{lower}(pp^T)c^T + c\,\mathrm{lower}(M\Lambda M^T)c^T \end{aligned}$$

and $c' = [a' \;\; b'] = cM$. Adding the global phase back in gives the lemma. □

Theorem 38 (Tableau composition [45])**.** *Let Q_1, Q_2 be n-qubit Clifford unitaries represented by tableaux $(M_1,\ p_1,\ s_1)$ and $(M_2,\ p_2,\ s_2)$ respectively. The composition Q_2Q_1 has tableau (M_{12}, p_{12}, s_{12}) with $M_{12} = M_1M_2$, $p_{12} = p_1 + M_1p_2$, and*

$$s_{12} = s_1 + p_1 \circ M_1p_2 + M_1s_2 + \mathrm{diag}(M_1\,\mathrm{lower}(p_2p_2^T + M_2\Lambda M_2^T)M_1^T)$$

where $\circ$ denotes the Hadamard product[16] *and "diag" selects the elements along the diagonal.*

Proof. Notice that $Q_2Q_1\mathtt{X}^{e_j}Q_1^\dagger Q_2^\dagger = Q_2(i^{(p_1)_j}(-1)^{(s_1)_j}\mathtt{X}^{(A_1)_j}\mathtt{Z}^{(B_1)_j})Q_2^\dagger$, and similarly for the images of $\mathtt{Z}^{e_j}$. Thus, we arrive at the theorem statement by applying Lemma 37 with Q_2 and every Pauli specified by the rows of the tableau for Q_1. □

From the tableaux composition theorem, it should be clear that the tableau for the product of two Clifford operations on n-qubits can be computed in time $O(n^\omega)$. Furthermore, when the two Clifford operations are of different sizes, it may be possible to obtain a significant savings. We know from standard techniques that if Q_2 is a one or two-qubit gate, then the tableau for Q_1 can be updated in $O(n)$ time.

The theorem below interpolates between these extremes.

Concretely, composing an n-qubit operation Q_1 with an m-qubit operation Q_2 for $m \leq n$, means computing the product $(Q_2 \otimes I^{n-m})Q_1$, and similarly $Q_2(Q_1 \otimes I^{m-n})$ when $m > n$.

Theorem 39. *Let Q_1, Q_2 be n and m-qubit Clifford operators, respectively, given by their tableaux. The tableau of their product can be computed in time $O(\min(m^{\omega-1}n, n^{\omega-1}m))$.*

[16]The Hadamard product of two $n \times m$ matrices $A = \{a_{i,j}\}_{i,j}$ and $B = \{b_{i,j}\}_{i,j}$ is their entrywise product $A \circ B = \{a_{i,j}b_{i,j}\}_{i,j}$.

Proof. When $m = \Theta(n)$, the theorem is trivial. Let's consider the case where $m = o(n)$. Without loss of generality, let's assume that Q_2 is applied to the first m qubits of Q_1. On n qubits, the tableau of Q_2 is

$$M_2 = \left[\begin{array}{cc|cc} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ \hline C & 0 & D & 0 \\ 0 & 0 & 0 & I \end{array}\right] \qquad p_2 = \left[\begin{array}{c} \alpha \\ 0 \\ \hline \delta \\ 0 \end{array}\right] \qquad s_2 = \left[\begin{array}{c} \beta \\ 0 \\ \hline \epsilon \\ 0 \end{array}\right]$$

where $M_2 \in \{0,1\}^{2n\times 2n}$ and $A, B, C, D \in \{0,1\}^{m\times m}$. Matching the dimensions of the matrices above, let's write the tableau of Q_1 as follows:

$$M_1 = \left[\begin{array}{cc|cc} A_{11} & A_{12} & B_{11} & B_{12} \\ A_{21} & A_{22} & B_{21} & B_{22} \\ \hline C_{11} & C_{12} & D_{11} & D_{12} \\ C_{21} & C_{22} & D_{21} & D_{22} \end{array}\right] \qquad p_1 = \left[\begin{array}{c} \alpha_1 \\ \alpha_2 \\ \hline \delta_1 \\ \delta_2 \end{array}\right] \qquad s_1 = \left[\begin{array}{c} \beta_1 \\ \beta_2 \\ \hline \epsilon_1 \\ \epsilon_2 \end{array}\right]$$

Therefore, according to Theorem 38, we have that the matrix M_{12} for the tableaux of Q_2Q_1 is

$$\left[\begin{array}{cc|cc} A_{11} & A_{12} & B_{11} & B_{12} \\ A_{21} & A_{22} & B_{21} & B_{22} \\ \hline C_{11} & C_{12} & D_{11} & D_{12} \\ C_{21} & C_{22} & D_{21} & D_{22} \end{array}\right] \left[\begin{array}{cc|cc} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ \hline C & 0 & D & 0 \\ 0 & 0 & 0 & I \end{array}\right] = \left[\begin{array}{cc|cc} A_{11}A + B_{11}C & A_{12} & A_{11}B + B_{11}D & B_{12} \\ A_{21}A + B_{21}C & A_{22} & A_{21}B + B_{21}D & B_{22} \\ \hline C_{11}A + D_{11}C & C_{12} & C_{11}B + D_{11}D & D_{12} \\ C_{21}A + D_{21}C & C_{22} & C_{21}D + D_{21}D & D_{22} \end{array}\right]$$

There are two types of matrix products we require. Terms such as $A_{11}A + B_{11}C$ consist entirely of $m \times m$ matrix products, so can be computed in time $O(m^\omega)$. Terms such as $A_{21}A + B_{21}C$ require multiplication of an $(n-m)\times m$ matrix by an $m \times m$ matrix. Dividing these matrices into blocks of size m, we get new block matrices of dimensions roughly $n/m \times 1$ and 1×1 where multiplication of a single element (i.e., block) takes time $O(m^\omega)$. There are n/m many block multiplications in the product, so it can be computed in time $O((n/m)m^\omega)$. Therefore, the entire procedure costs $O(nm^{\omega-1})$ time.

The calculation for the phase vector is similar, so we omit it. Let us now turn to the calculation of the sign vector. First, notice that the diagonal of $M_1 \operatorname{lower}(p_2 p_2^T + M_2 \Lambda M_2^T) M_1^T$ is equal to

$$\operatorname{diag}(M_1 \operatorname{lower}(p_2 p_2^T) M_1^T) + \operatorname{diag}(M_1 \operatorname{lower}(M_2 \Lambda M_2^T) M_1^T),$$

so we will focus on computing the second more complicated term. We have

$$M_2 \Lambda M_2^T = \left[\begin{array}{cc|cc} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ \hline C & 0 & D & 0 \\ 0 & 0 & 0 & I \end{array}\right] \left[\begin{array}{cc|cc} 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{cc|cc} A^T & 0 & C^T & 0 \\ 0 & I & 0 & 0 \\ \hline B^T & 0 & D^T & 0 \\ 0 & 0 & 0 & I \end{array}\right] = \left[\begin{array}{cc|cc} AB^T & 0 & AD^T & 0 \\ 0 & 0 & 0 & I \\ \hline CB^T & 0 & CD^T & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

from which, we get

$$\operatorname{lower}(M_2 \Lambda M_2^T) = \left[\begin{array}{cc|cc} \operatorname{lower}(AB^T) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline CB^T & 0 & \operatorname{lower}(CD^T) & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

from which we can derive the condition along the diagonal—that is, we have that the diagonal of $M_1 \operatorname{lower}(M_2 \Lambda M_2^T) M_1^T$ is equal to

$$\begin{aligned} & \operatorname{diag}(A_{11} \operatorname{lower}(AB^T) A_{11}^T + B_{11} C B^T A_{11}^T + B_{11} \operatorname{lower}(CD^T) C_{11}^T) \\ \oplus\ & \operatorname{diag}(A_{21} \operatorname{lower}(AB^T) A_{21}^T + B_{21} C B^T A_{21}^T + B_{21} \operatorname{lower}(CD^T) C_{21}^T) \\ \oplus\ & \operatorname{diag}(C_{11} \operatorname{lower}(AB^T) B_{11}^T + D_{11} C B^T B_{11}^T + D_{11} \operatorname{lower}(CD^T) D_{11}^T) \\ \oplus\ & \operatorname{diag}(C_{21} \operatorname{lower}(AB^T) B_{21}^T + D_{21} C B^T B_{21}^T + D_{21} \operatorname{lower}(CD^T) D_{21}^T) \end{aligned}$$

where $\oplus$ is used here to indicate direct sum. Since A, B, C, D are $m \times m$ matrices, we can explicitly compute $\operatorname{lower}(AB^T)$ and $\operatorname{lower}(CD^T)$ in $O(m^\omega)$ time. Consider a term such as $A_{21} \operatorname{lower}(AB^T) A_{21}^T$ where we are multiplying matrices with dimension $(n-m) \times m$, $m \times m$, and $m \times (n-m)$. Dividing each matrix into blocks of size m, we get matrices of dimensions roughly $n/m \times 1$, 1×1, and $1 \times n/m$ where multiplication of blocks takes time $O(m^\omega)$. Since we only need to calculate the n/m blocks along the diagonal of the product matrix, we can compute the entire diagonal in time $O((n/m)m^\omega)$. Since all other terms follow in a similar manner, we can calculate the entire sign vector in time $O(nm^{\omega-1})$. Thus, for $m = o(n)$, the entire composition takes time $O(nm^{\omega-1})$.

When $n = o(m)$, the calculation is extremely similar to the one above, so we omit it. □

The above theorem is more general than needed for the purposes of this paper. Nevertheless, we can use it to show how to multiply any n-qubit stabilizer state by an arbitrary collection of CZ gates in time $O(n^\omega)$ as stated in Table 1. First, let us assume that no two CZ gates in our collection are identical since they would simply cancel. Then, let A be the $n \times n$ binary symmetric matrix whose (i, j) entry is 1 iff there is a CZ gate between qubits i and j in our collection. The tableau for the entire collection of CZ gates is

$$M = \left[\begin{array}{c|c} I & A \\ \hline 0 & I \end{array}\right] \quad p = \left[\begin{array}{c} 0 \\ \hline 0 \end{array}\right] \quad s = \left[\begin{array}{c} 0 \\ \hline 0 \end{array}\right]$$

Since we can explicitly write down the tableau for the collection of CZ gates, we can straightforwardly apply Theorem 39.

5.2 Measurement

The previous sections describe tableaux as representing Clifford unitary operations. However, tableaux are equally suitable for representing states. Given tableau $M = \left[\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right]$, $p = \left[\begin{smallmatrix} \alpha \\ \delta \end{smallmatrix}\right]$, and $s = \left[\begin{smallmatrix} \beta \\ \epsilon \end{smallmatrix}\right]$, the stabilizer state defined by the tableau is the unique state $|\psi\rangle$ such that

$$i^{\delta_j} (-1)^{\epsilon_j} \mathtt{X}^{C_j} \mathtt{Z}^{D_j} |\psi\rangle = |\psi\rangle$$

for all $j \in \{1, \ldots, n\}$. That is, the lower half of the tableau describes the Pauli operations that fix or "stabilize" the state, which is why these Pauli operations are often referred to as stabilizers. Nevertheless, the upper half of the tableau (consisting of the "destabilizers" of the state $|\psi\rangle$) is still useful in speeding up measurement computations [17]. In this section, we show how to combine those techniques with fast matrix multiplication.

An important primitive for the measurement protocol is choosing new stabilizer generators for the tableau by multiplying existing stabilizer generators together. Therefore, we will need a fast way to multiply several Pauli operators at once. We will encounter two situations in particular:

1. There are several Pauli operators, but each operator has few non-identity elements;
2. There are few Pauli operators, but each operator has many non-identity elements.

The following technical lemma handles both cases.

Lemma 40. *Let Q be an n-qubit Clifford unitary represented by tableau (M, p, s). Suppose we wish to compute $QP_jQ^\dagger$ for ℓ Pauli operators $\{P_j\}_{j=1}^{\ell}$ that only act non-trivially on the first k qubits. Precisely, we define $P_j := \mathtt{X}^{[A_j\ 0]}\mathtt{Z}^{[B_j\ 0]}$ to be an n-qubit Pauli where $A_j, B_j \in \{0,1\}^k$ form the rows of the $\ell \times 2n$ matrix $C = [A\ 0 \mid B\ 0]$. We have $QP_jQ^\dagger = i^{\alpha'_j}(-1)^{\beta'_j}\mathtt{X}^{A'_j}\mathtt{Z}^{B'_j}$ with $C' = [A' \mid B'] = CM$, $\alpha' = Cp$, and*

$$\beta' = Cs + \operatorname{diag}(C \operatorname{lower}(pp^T + M\Lambda M^T)C^T)$$

Furthermore, C', α', and β' can be computed in time

- *$O(n^2k^{\omega-2})$ when $\ell = n$, i.e., several Pauli operators with few non-identity elements.*
- *$O(n^2\ell^{\omega-2}\log(2n/\ell))$ when $\ell \le k = n$, i.e., few Pauli operators with many non-identity elements*

Proof. Correctness follows from Lemma 37, so we focus only on running time. Once again, we will only show how to compute the most complicated term—$\operatorname{diag}(C \operatorname{lower}(M\Lambda M^T)C^T)$—of the sign vector β', and the rest of the terms follow by similar (or simpler) arguments. We will write M in terms of the blocks $A_{11} \in \{0,1\}^{k\times k}$, $A_{12} \in \{0,1\}^{k\times(n-k)}$, $A_{22} \in \{0,1\}^{(n-k)\times(n-k)}$, and so on as in Theorem 39.

First, let's consider the case with many Pauli operators (i.e., $\ell = n$). Following the same algebra as in Theorem 39, we arrive at the following expression for $C \operatorname{lower}(M\Lambda M^T)C^T$:

$$A \operatorname{lower}(A_{11}B_{11}^T + A_{12}B_{12}^T)A^T + B(C_{11}B_{11}^T + C_{12}B_{12}^T)A^T + B \operatorname{lower}(C_{11}D_{11}^T + C_{12}D_{12}^T)B^T.$$

To compute $\operatorname{lower}(\cdot)$, we need to compute a $k \times (n-k)$ by $(n-k) \times k$ matrix product. Dividing into $k \times k$ blocks, such a multiplication takes time $O(nk^{\omega-1})$, after which we can trivially compute the lower triangle matrix in $O(k^2)$ time. Finally, we must multiply matrices with dimensions $n \times k$, $k \times k$, and $k \times n$. Once again, we divide into $k \times k$ blocks, to compute the entire product in $O((n/k)^2k^\omega)$ time. This concludes the first case.

Now consider the case where the Pauli operators can act non-trivially on all qubits (i.e., $k = n$). Since $k = n$, we can eliminate many of the terms from the expression we are trying to compute:

$$A \operatorname{lower}(A_{11}B_{11}^T)A^T + B(C_{11}B_{11}^T)A^T + B \operatorname{lower}(C_{11}D_{11}^T)B^T.$$

Nevertheless, we unfortunately encounter a new problem. Now notice that computing $\operatorname{lower}(\cdot)$ now requires $n \times n$ matrix products, so there is not obvious way to compute it in $O(n^2\ell^{\omega-2})$ time. For the $B(C_{11}B_{11}^T)A^T$ term, we can choose the order in which we compose the matrices: first compute BC_{11} which takes time $O(n^2\ell^{\omega-2})$ and leaves another $\ell \times n$ matrix, and so on. Unfortunately, such a strategy does not automatically work for the other terms.

Since the two $\operatorname{lower}(\cdot)$ terms have identical structure, let's focus on $A \operatorname{lower}(A_{11}B_{11}^T)A^T$. To reiterate, A is an $\ell \times n$ matrix, and A_{11} and B_{11}^T are $n \times n$ matrices. Let $T(n, \ell)$ be the time it takes to compute $A \operatorname{lower}(A_{11}B_{11}^T)A^T$ for dimensions n and ℓ.

We give a recursive strategy to speed up the brute force calculation of terms of this type. Let's decompose these matrices further, writing $A = [A_1 A_2]$ for $A_1, A_2 \in \{0,1\}^{\ell \times n/2}$, $A_{11} = \left[\begin{smallmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{smallmatrix}\right]$, with $X_{ij} \in \{0,1\}^{n/2 \times n/2}$, and $B_{11}^T = \left[\begin{smallmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{smallmatrix}\right]$, with $Z_{ij} \in \{0,1\}^{n/2 \times n/2}$. Together, we can expand $A \operatorname{lower}(A_{11} B_{11}^T) A^T$ as

$$A_1 \operatorname{lower}(X_{11}Z_{11} + X_{12}Z_{21}) A_1^T + A_2(X_{21}Z_{11} + X_{22}Z_{21}) A_1^T + A_2 \operatorname{lower}(X_{21}Z_{12} + X_{22}Z_{22}) A_2^T.$$

By the above expression, we have

$$T(n, \ell) \leq 4T(n/2, \ell) + c_1 n^2 \ell^{\omega-2}$$

where c_1 is some constant. We now claim that $T(n, \ell) \leq c_2 n^2 \ell^{\omega-2} \log(2n/\ell)$. This is clearly true when $\ell = n$, since we can compute $\ell \times \ell$ matrix products in $O(\ell^\omega)$ time. Otherwise, by induction we have

$$\begin{aligned} T(n, \ell) &\leq 4c_2 (n/2)^2 \ell^{\omega-2} \log(n/\ell) + c_1 n^2 \ell^{\omega-2} \\ &= c_2 n^2 \ell^{\omega-2} \log(2n/\ell) - c_2 n^2 \ell^{\omega-2} + c_1 n^2 \ell^{\omega-2} \\ &\leq c_2 n^2 \ell^{\omega-2} \log(2n/\ell) \end{aligned}$$

for $c_2 \geq c_1$. □

We are now ready to prove the main measurement theorem. We state the theorem in terms of a Z-basis measurement on k qubits. That said, the cost of this measurement dominates the cost of applying an arbitrary k-qubit Clifford unitary, so the theorem still holds for any choice of basis.

Theorem 41. *Let any n-qubit stabilizer state (represented by its tableau) and a subset of k qubits be given. There exists an $O(\min(n^2 k^{\omega-2} \log(2n/k), kn^2))$ time algorithm that returns*

1. *Z-basis measurement outcomes for each qubit in the subset, and*
2. *The Clifford tableau for the $(n-k)$-qubit post-measurement state.*

Furthermore, the same algorithm can be used to postselect[17] *on a particular k-bit outcome.*

Proof. Without loss of generality, let us assume we measure the first k qubits. We will also only be concerned with the $O(n^2 k^{\omega-2} \log(2n/k))$ measurement algorithm since one can trivially obtain a $O(kn^2)$ algorithm by applying the single-qubit measurement scheme of Aaronson and Gottesman k times [17].

Let us write the tableau for our state represented by Clifford Q as

$$M = \left[\begin{array}{cc|cc} A_{11} & A_{12} & B_{11} & B_{12} \\ A_{21} & A_{22} & B_{21} & B_{22} \\ \hline C_{11} & C_{12} & D_{11} & D_{12} \\ C_{21} & C_{22} & D_{21} & D_{22} \end{array}\right] \qquad p = \left[\begin{array}{c} \alpha_1 \\ \alpha_2 \\ \hline \delta_1 \\ \delta_2 \end{array}\right] \qquad s = \left[\begin{array}{c} \beta_1 \\ \beta_2 \\ \hline \epsilon_1 \\ \epsilon_2 \end{array}\right]$$

where $A_{11} \in \{0,1\}^{k \times k}$, $A_{22} \in \{0,1\}^{(n-k) \times (n-k)}$, and so on. We will perform the measurement in two steps, following the Aaronson-Gottesman measurement procedure [17]—first

[17] In addition to the subset of k qubits to be measured, a string of k preferred measurement outcomes is given as input. The algorithm either returns that the postselected event has probability 0 or returns the $(n-k)$-qubit state assuming the postselection was successful.

focusing on those outcomes which are random and then on those which are determinate.[18] We will assume correctness from the Aaronson-Gottesman procedure and will focus primarily on running times.

Step 1: Random measurement outcomes

To determine the random measurements, we first need the Pauli-X parts of the stabilizers, i.e., the $n \times k$ submatrix $C := \left[\begin{smallmatrix} C_{11} \\ C_{21} \end{smallmatrix}\right]$, to be in reduced row echelon form. To do this, we compute the LSP factorization (see Ibarra, Moran, and Hui [42]) of C^T in $O(k^{\omega-1}n)$ time. That is, C^T can be written as the product of three matrices LSP, where L is an $k \times k$ lower triangular matrix with 1's on the main diagonal, S is an $k \times n$ matrix which reduces to an upper triangular matrix with 1's along the diagonal when the zero rows are deleted, and P is an $n \times n$ permutation matrix. Therefore, we get $C = P^T S^T L^T$.

We assume without loss of generality that P is the identity transformation, since we can rearrange of the rows of the tableau according to P without affecting the underlying state.

Now, let us rearrange the columns of S^T by permutation Π such that only the first $r := \text{rank}(C)$ columns are non-zero. That is, we write $C = (S^T\Pi)(\Pi^T L^T \Pi)\Pi^T$.

The final matrix in this decomposition, Π^T, is a permutation on the first k qubits of the state, so let us drop it in the following discussion since we can simply permute those columns of the tableau accordingly. We can now write $C = S'U'$, where S' is the $n \times r$ matrix obtained by removing the last $k - r$ columns from $S^T\Pi$ and U' is the $r \times k$ matrix obtained by removing the last $k - r$ rows from $\Pi^T L^T \Pi$. Notice that the first $r \times r$ submatrix of U' is upper triangular with 1's on the main diagonal. Let's also write $S' = \left[\begin{smallmatrix} L' \\ A' \end{smallmatrix}\right]$ where L' is an $r \times r$ lower triangular matrix with 1's on the diagonal, and A' is arbitrary.

Before proceeding, let us recap the current state of the tableau. Until this point, we have only swapped its rows and columns, so nothing has fundamentally changed about its structure. We can either think of these swaps as occurring virtually, by keeping track of the permutation of the rows and columns, or explicitly, by physically moving the data around in the tableau as each row/column is only swapped constantly-many times. Because of this procedure, we can assume our original $n \times k$ matrix C is of the form

$$C = S'U' = \begin{bmatrix} L'U' \\ A'U' \end{bmatrix}$$

We now define $\mathcal{L}$ to be the $n \times n$ matrix that reduces C to row echelon form, i.e., $\mathcal{L}C = \left[\begin{smallmatrix} U' \\ 0 \end{smallmatrix}\right]$. One can check that $\mathcal{L} = \left(\begin{smallmatrix} (L')^{-1} & 0 \\ A'(L')^{-1} & I \end{smallmatrix}\right)$ has the desired behavior.

First notice that we can construct $\mathcal{L}$ efficiently since computing the inverse of the $r \times r$ matrix L' takes time $O(r^\omega)$ and computing the product between the $(n - r) \times r$ matrix A' and L' takes time at most $O(nr^{\omega-1})$.

We will view the application of $\mathcal{L}$ as directly multiplying the stabilizer generators of Q. For example, if the ith row of $\mathcal{L}$ has 1's in the first, second, and fourth columns, we will combine the first, second, and fourth stabilizers in the tableau (i.e., compute $Q\mathsf{Z}^{e_1}\mathsf{Z}^{e_2}\mathsf{Z}^{e_4}Q^\dagger$). Since $\mathcal{L}$ is only dense in the first r columns, we can use Lemma 40 to construct the resulting tableau in $O(n^2 r^{\omega-2})$ time (i.e., the variables "ℓ" and "k" in the

[18] Note that the choice of which measurements are random and which are determinate is not unique. For example, consider the Bell state $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$. Measuring the first qubit, determines the second, and vice versa.

statement of Lemma 40 are set to n and r, respectively). Since we are only taking linear combinations of generators, the underlying state does not change. To preserve the fact that the tableau is symplectic, we can apply Lemma 40 again with $\mathcal{L}^{-T} = \left(\begin{smallmatrix} (L')^T & (A')^T \\ 0 & I \end{smallmatrix}\right)$ applied to the destabilizers.[19] Because $\mathcal{L}^{-T}$ is now upper triangular, we must use the other setting of Lemma 40 (i.e., with the variable "ℓ" set to r and "k" set to n), which takes $O(n^2 r^{\omega-2} \log(2n/r))$ time.

To recap, the original submatrix C of our tableau is now $\left[\begin{smallmatrix} U' \\ 0 \end{smallmatrix}\right]$. Also recall that $U' = \left[\begin{smallmatrix} U'' A'' \end{smallmatrix}\right]$ where U'' is an $r \times r$ upper triangular matrix with ones along the diagonal. Therefore, we can compute $(U'')^{-1}$ explicitly in time $O(r^\omega)$, and once again apply Lemma 40 so that each of the first r stabilizers has exactly one Pauli-X component (i.e., the submatrix is in reduced row echelon form). We then apply $(U'')^T$ to the destabilizers in the same fashion.

In the next step, we multiply the stabilizer generators into the destabilizer generators, so that no destabilizer has a Pauli-X component within the first r qubits.[20] Once again, this can be accomplished by Lemma 40. Then, we replace the first r destabilizers with the first r stabilizers.

Since these first r qubits will be the qubits for which the measurement outcome is random, we can now output a uniformly random measurement for each (or, in the case of postselection, the specified outcome). Let this measurement be $z \in \{0,1\}^r$. For $i \in \{1, \ldots, r\}$, we replace the ith stabilizer with $(-1)^{z_i} \mathtt{Z}^{e_i}$.

Unfortunately, the tableau we just constructed is only valid for the post-measurement state in tensor product with the measured qubits. Since we would like a tableau for just the $(n-k)$-qubit post-measurement state, additional work is required. To recap, the state of the tableau is now

$$M = \left[\begin{array}{cc|cc} I & A_{12} & B_{11} & B_{12} \\ 0 & A_{22} & B_{21} & B_{22} \\ \hline 0 & 0 & I & 0 \\ 0 & C_{22} & D_{21} & D_{22} \end{array}\right] \qquad p = \left[\begin{array}{c} \alpha_1 \\ \alpha_2 \\ \hline 0 \\ \delta_2 \end{array}\right] \qquad s = \left[\begin{array}{c} \beta_1 \\ \beta_2 \\ \hline z \\ \epsilon_2 \end{array}\right]$$

where $A_{12} \in \{0,1\}^{n \times n-r}$, and so on. To clarify, this is a slight abuse of notation since we've subdivided the matrix in terms of r instead of k (and we've chosen to redefine the submatrices instead of giving them new names). In order to safely remove the first r qubits from the tableau, we need to ensure that the tableau will still be symplectic afterwards. In some sense, we need $A_{12}, B_{11}, B_{12}, B_{21}$ and D_{21} all to be 0. As before, we can zero out D_{21} using Lemma 40 with the transformation $L = \left(\begin{smallmatrix} I & 0 \\ D_{21} & I \end{smallmatrix}\right)$ on the stabilizers

[19] In other words, the row operations on the stabilizers/destabilizers result from multiplying the following matrix on the entire tableau: $\left[\begin{array}{c|c} \mathcal{L}^{-T} & 0 \\ \hline 0 & \mathcal{L} \end{array}\right]$. Since this matrix is symplectic, it corresponds to a valid Clifford tableau with 0's for phase bits (Fact 35). Therefore, the resulting product is also a valid Clifford tableau. Here and elsewhere, we use $-T$ to denote both inverse and transpose.

[20] A note of warning: this is the first operation that we apply to the tableau which is not symplectic. For a more in-depth justification of this procedure, we refer the reader to the work of Aaronson/Gottesman [17].

and $L^{-T} = \left(\begin{smallmatrix} I & D_{21}^T \\ 0 & I \end{smallmatrix}\right)$ on the destabilizers. This results in the tableau:

$$M' = \left[\begin{array}{cc|cc} I & A'_{12} & B'_{11} & B'_{12} \\ 0 & A'_{22} & B'_{21} & B'_{22} \\ \hline 0 & 0 & I & 0 \\ 0 & C_{22} & 0 & D_{22} \end{array}\right] \qquad p' = \left[\begin{array}{c} \alpha'_1 \\ \alpha'_2 \\ \hline 0 \\ \delta_2 \end{array}\right] \qquad s' = \left[\begin{array}{c} \beta'_1 \\ \beta'_2 \\ \hline z \\ \epsilon'_2 \end{array}\right]$$

We can now construct the tableau for the state on $n - r$ qubits by simply removing the first r rows/columns from each submatrix:

$$M'' = \left[\begin{array}{c|c} A'_{22} & B'_{22} \\ \hline C_{22} & D_{22} \end{array}\right] \qquad p'' = \left[\begin{array}{c} \alpha'_2 \\ \hline \delta_2 \end{array}\right] \qquad s'' = \left[\begin{array}{c} \beta'_2 \\ \hline \epsilon'_2 \end{array}\right]$$

It may not be the case that $A'_{12}, B'_{11}, B'_{12}$, and B'_{21} are all zero, but one can check that the tableau is symplectic using the fact that it was symplectic before the rows/columns were removed.

Step 2: Determinate measurement outcomes

Since we have already removed the r qubits which had random outcomes in Step 1, let us assume without loss of generality that we are measuring the first k qubits, all of which have determinate outcomes. That is, for $i \in \{1, \ldots, k\}$, we have $\epsilon_i \mathrm{Z}^{e_i}$ is a stabilizer of the state for some $\epsilon_i \in \{\pm 1\}$. The goal is to determine the ϵ_i.

To start, let us put the Pauli `X` part of the destabilizers—i.e., the submatrix $\left[\begin{smallmatrix} A_{11} \\ A_{21} \end{smallmatrix}\right]$—into reduced row echelon form. Once again, this will require a fast LSP decomposition coupled with several applications of Lemma 40. Recall that their are no Pauli-`X` terms in the stabilizers, so the tableau of the resulting transformation is (once again, redefining the submatrices instead of giving them new names)

$$M = \left[\begin{array}{cc|cc} I & A_{12} & B_{11} & B_{12} \\ 0 & A_{22} & B_{21} & B_{22} \\ \hline 0 & C_{12} & D_{11} & D_{12} \\ 0 & C_{22} & D_{21} & D_{22} \end{array}\right] \qquad p = \left[\begin{array}{c} \alpha_1 \\ \alpha_2 \\ \hline \delta_1 \\ \delta_2 \end{array}\right] \qquad s = \left[\begin{array}{c} \beta_1 \\ \beta_2 \\ \hline \epsilon_1 \\ \epsilon_2 \end{array}\right].$$

In fact, because M is symplectic, we have that $C_{12} = 0$, $D_{11} = I$, and $D_{12} = 0$, so we can immediately report the measurement outcome of the ith qubit as $(\epsilon_1)_i$. In the case of postselection, we check if these outcomes match the specified outcomes. If so, then there is nothing to be done; otherwise, we report that the postselected event has probability 0.

Finally, we repeat the procedure from the end of Step 1: row reducing so that $D_{21} = 0$, and then returning the tableau with the first k rows/columns removed from each submatrix. This completes the proof. □

Theorem 1. *Let $|\psi\rangle$ be any n-qubit stabilizer state given by its tableau, and let $z = z_1 \ldots z_k$ be any k-bit string of postselected outcomes. There is an $O(n^\omega)$ algorithm to obtain a uniformly random measurement of the state $(\langle z| \otimes I)|\psi\rangle$ in the computational basis.*[21]

Proof. The idea will be to invoke Theorem 41 twice—first, for the postselected qubits, and then for the remaining qubits. In other words, if we let $f(n, k) := n^2 k^{\omega-2} \log(2n/k)$

[21] If $\omega = 2$, we pick up an additional log factor in the solution of the recurrence, i.e., a running time of $O(n^2 \log n)$.

denote the bound given for the measurement protocol in Theorem 41, then the running time of the proposed algorithm is bounded by

$$f(n,k) + f(n-k, n-k) = O(n^2 k^{\omega-2} \log(2n/k) + (n-k)^\omega).$$

It remains to show that gives a final bound of $O(n^\omega)$ for all k. We separate into two cases based on how k grows with n. When $k = \Theta(n)$, we have $c_1 n \leq k \leq c_2 n$ for some constants $c_1, c_2 > 0$, and so

$$n^2 k^{\omega-2} \log(2n/k) + (n-k)^\omega \leq c_2^{\omega-2} n^\omega \log(2/c_1) + n^\omega = O(n^\omega).$$

When $k = o(n)$, we have $k = n/g(n)$ for some $g(n) = \omega(1)$, and so

$$n^2 k^{\omega-2} \log(2n/k) + (n-k)^\omega \leq \frac{n^\omega}{g(n)^{\omega-2}} \log(2g(n)) + n^\omega = O(n^\omega)$$

where we use that $\log(2g(n))/g(n)^\epsilon = o(1)$ for all constants $\epsilon > 0$. Therefore, in the case of $\omega = 2$, we actually get a bound of $O(n^2 \log n)$. □

We note again that the choice of measurement bases in Theorem 1 is arbitrary since single-qubit gates can be applied in $O(n)$ time.

6 Acknowledgments

D. Gosset and D. Grier are supported in part by IBM Research. We also acknowledge the support of the Natural Sciences and Engineering Research Council of Canada and the Canadian Institute for Advanced Research. We thank Sergey Bravyi and Robert König for helpful discussions. We also thank an anonymous reviewer for detailed feedback on presentation and for identifying an incorrect statement regarding runtime in Claim 21 in the previous version.

References

[1] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. "Quantum supremacy using a programmable superconducting processor". Nature **574**, 505–510 (2019).

[2] "Ibm quantum documentation". https://docs.quantum.ibm.com/run.

[3] Matthew D Reed, Leonardo DiCarlo, Simon E Nigg, Luyan Sun, Luigi Frunzio, Steven M Girvin, and Robert J Schoelkopf. "Realization of three-qubit quantum error correction with superconducting circuits". Nature **482**, 382–385 (2012).

[4] Antonio D Córcoles, Easwar Magesan, Srikanth J Srinivasan, Andrew W Cross, Matthias Steffen, Jay M Gambetta, and Jerry M Chow. "Demonstration of a quantum error detection code using a square lattice of four superconducting qubits". Nature communications **6**, 1–10 (2015).

[5] Nissim Ofek, Andrei Petrenko, Reinier Heeres, Philip Reinhold, Zaki Leghtas, Brian Vlastakis, Yehan Liu, Luigi Frunzio, SM Girvin, Liang Jiang, et al. "Extending the lifetime of a quantum bit with error correction in superconducting circuits". Nature **536**, 441–445 (2016).

[6] Igor L Markov and Yaoyun Shi. "Simulating quantum computation by contracting tensor networks". SIAM Journal on Computing **38**, 963–981 (2008).
[7] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, and Hartmut Neven. "Simulation of low-depth quantum circuits as complex undirected graphical models" (2017).
[8] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. "Simulation of quantum circuits by low-rank stabilizer decompositions". Quantum **3**, 181 (2019).
[9] Edwin Pednault, John A Gunnels, Giacomo Nannicini, Lior Horesh, Thomas Magerlein, Edgar Solomonik, Erik W Draeger, Eric T Holland, and Robert Wisnieff. "Breaking the 49-qubit barrier in the simulation of quantum circuits" (2017).
[10] Edwin Pednault, John A Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. "Leveraging secondary storage to simulate deep 54-qubit Sycamore circuits" (2019).
[11] Boaz Barak, Chi-Ning Chou, and Xun Gao. "Spoofing linear cross-entropy benchmarking in shallow quantum circuits" (2020).
[12] Barbara M Terhal and David P DiVincenzo. "Adaptive quantum computation, constant depth quantum circuits and Arthur-Merlin games" (2002).
[13] Michael J Bremner, Richard Jozsa, and Dan J Shepherd. "Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy". Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **467**, 459–472 (2011).
[14] Scott Aaronson and Alex Arkhipov. "The computational complexity of linear optics". In Proceedings of the forty-third annual ACM symposium on Theory of computing. Pages 333–342. (2011).
[15] Daniel Gottesman. "The Heisenberg representation of quantum computers" (1998).
[16] Sergey Bravyi and David Gosset. "Improved classical simulation of quantum circuits dominated by Clifford gates". Physical review letters **116**, 250501 (2016).
[17] Scott Aaronson and Daniel Gottesman. "Improved simulation of stabilizer circuits". Physical Review A **70**, 052328 (2004).
[18] Sergey Bravyi, David Gosset, and Robert König. "Quantum advantage with shallow circuits". Science **362**, 308–311 (2018).
[19] Adam Bene Watts, Robin Kothari, Luke Schaeffer, and Avishay Tal. "Exponential separation between shallow quantum circuits and unbounded fan-in shallow classical circuits". In Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing. Pages 515–526. (2019).
[20] Daniel Grier and Luke Schaeffer. "Interactive shallow Clifford circuits: quantum advantage against NC^1 and beyond". In Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing. Pages 875–888. (2020).
[21] Sergey Bravyi, David Gosset, Robert Koenig, and Marco Tomamichel. "Quantum advantage with noisy shallow circuits". Nature PhysicsPages 1–6 (2020).
[22] Robert Raussendorf and Hans J Briegel. "A one-way quantum computer". Physical Review Letters **86**, 5188 (2001).
[23] Josh Alman and Virginia Vassilevska Williams. "A refined laser method and faster matrix multiplication" (2020).
[24] Chaowen Guan and Kenneth W Regan. "Stabilizer circuits, quadratic forms, and computing matrix rank" (2019).

[25] Daniel Grier and Luke Schaeffer. "gridCHP++, Apache license version 2.0". https://github.com/danielgrier/gridCHPpp/tree/v1.0.0.
[26] Alan George. "Nested dissection of a regular finite element mesh". SIAM Journal on Numerical Analysis **10**, 345–363 (1973).
[27] Richard J Lipton, Donald J Rose, and Robert Endre Tarjan. "Generalized nested dissection". SIAM journal on numerical analysis **16**, 346–358 (1979).
[28] Noga Alon and Raphael Yuster. "Solving linear systems through nested dissection". In 2010 IEEE 51st Annual Symposium on Foundations of Computer Science. Pages 225–234. IEEE (2010).
[29] Richard J. Lipton and Robert Endre Tarjan. "A separator theorem for planar graphs". SIAM Journal on Applied Mathematics **36**, 177–189 (1979).
[30] Scott Aaronson and Lijie Chen. "Complexity-theoretic foundations of quantum supremacy experiments". In 32nd Computational Complexity Conference (CCC 2017). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2017).
[31] Jianxin Chen, Fang Zhang, Cupjin Huang, Michael Newman, and Yaoyun Shi. "Classical simulation of intermediate-size quantum circuits" (2018).
[32] Benjamin Villalonga, Dmitry Lyakh, Sergio Boixo, Hartmut Neven, Travis S Humble, Rupak Biswas, Eleanor G Rieffel, Alan Ho, and Salvatore Mandrà. "Establishing the quantum supremacy frontier with a 281 pflop/s simulation". Quantum Science and Technology **5**, 034003 (2020).
[33] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. "Complexity of finding embeddings in a k-tree". SIAM Journal on Algebraic Discrete Methods **8**, 277–284 (1987).
[34] HL Bodlaender. "A tourist guide through treewidth". Acta Cybernetica **11**, 1–21 (1993).
[35] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. "A $c^k n$ 5-approximation algorithm for treewidth". SIAM Journal on Computing **45**, 317–378 (2016).
[36] Sergey Bravyi, Graeme Smith, and John A Smolin. "Trading classical and quantum computational resources". Physical Review X **6**, 021043 (2016).
[37] M Van den Nest. "Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond" (2008).
[38] Alex Kerzner. "Clifford simulation: Techniques and applications". Master's thesis. University of Waterloo. (2021).
[39] Carsten Damm. "Problems complete for $\oplus$L". In Jürgen Dassow and Jozef Kelemen, editors, Aspects and Prospects of Theoretical Computer Science. Pages 130–137. Berlin, Heidelberg (1990). Springer Berlin Heidelberg.
[40] David Eppstein (2007). commons.wikimedia.org/wiki/File:Tree_decomposition.svg, accessed 08/31/2020.
[41] Hans L Bodlaender and Ton Kloks. "Better algorithms for the pathwidth and treewidth of graphs". In Automata, Languages and Programming: 18th International Colloquium Madrid, Spain, July 8–12, 1991 Proceedings 18. Pages 544–555. Springer (1991).
[42] Oscar H Ibarra, Shlomo Moran, and Roger Hui. "A generalization of the fast LUP matrix decomposition algorithm and applications". Journal of Algorithms **3**, 45–56 (1982).

[43] Adi Ben-Israel and Thomas NE Greville. "Generalized inverses: theory and applications". Volume 15. Springer Science & Business Media. (2003).
[44] Michael T Goodrich. "Planar separators and parallel polygon triangulation". Journal of Computer and System Sciences **51**, 374–389 (1995).
[45] Jeroen Dehaene and Bart De Moor. "Clifford group, stabilizer states, and linear and quadratic operations over GF(2)". Physical Review A **68**, 042318 (2003).
[46] Simon Anders and Hans J Briegel. "Fast simulation of stabilizer circuits using a graph-state representation". Physical Review A **73**, 022334 (2006).
[47] Sergey Bravyi. Personal communication, 2017 (2017).
[48] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. "Graphical description of the action of local Clifford transformations on graph states". Physical Review A **69**, 022316 (2004).

A Applications

In this section, we formally state and prove Theorems 6 and 7 from the introduction concerning the application of our graph state sampling problem to more traditional circuit simulation tasks. We will need the well-known fact that measurements of quantum states can induce certain unitary transformations on the remaining qubits. This is the basis of measurement-based quantum computation [22] and the heuristic algorithm for Clifford simulation due to Anders and Briegel [46]. Bravyi has established a linear-time reduction from constant-depth circuit simulation to graph state simulation using the tableau representation of stabilizer states [47]. To keep things relatively self-contained, here we shall use a "Hadamard gadget" [13] which allows us to implement Hadamard gates using X-basis measurements (see Figure 11). We note that a similar technique was used in the recent paper of Guan and Regan [24].

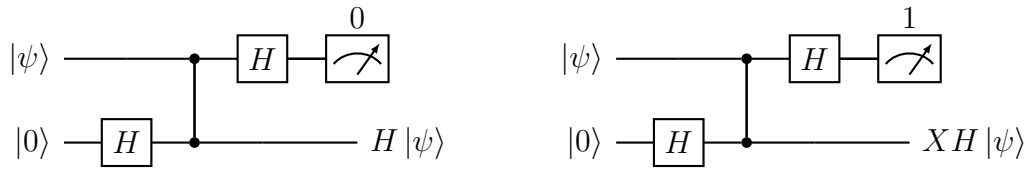


Figure 11: The Hadamard gadget implements either H or XH on the single-qubit input state ψ depending on the measurement outcome. By linearity, the gadget can also be used to implement a Hadamard gate on a single-qubit of a multi-qubit state.

A.1 Clifford tensor network simulation

A *Clifford tensor* T of rank k is an array of 2^k complex numbers $T_{i_1 i_2 \ldots i_k} \in \mathbb{C}$ indexed by k binary variables $i_1, i_2, \ldots, i_k \in \{0, 1\}$, with the additional property that the k-qubit state

$$|T\rangle \equiv \sum_{i_1 i_2 \ldots i_k} T_{i_1 i_2 \ldots i_k} |i_1 i_2 \ldots i_k\rangle \tag{20}$$

is proportional to a stabilizer state. Such a tensor can be represented pictorially as in Figure 12a where each "leg" represents one of the k binary variables. A leg of tensor $T^{(1)}$

and a leg of tensor $T^{(2)}$ can be contracted by summing the corresponding binary variable as depicted in Figure 12b.

A Clifford tensor network consists of a set $T^{(1)}, T^{(2)}, \ldots, T^{(n)}$ of "elementary" Clifford tensors along with a set $\mathcal{E}$ of pairs of legs that are to be contracted. Write $\mathcal{T}$ for the tensor obtained from the given collection of tensors by contracting all pairs of legs in $\mathcal{E}$. We shall abuse notation and use $\mathcal{T}$ to refer both to the tensor network and the tensor that it evaluates to. In particular, $\mathcal{T}$ is a Clifford tensor with rank equal to the number of uncontracted legs. If it is nonzero, then it defines a stabilizer state (up to normalization) as in Eq. (20) and we consider the problem of simulating measurement of all qubits of this state.

The *underlying graph* of the tensor network $\mathcal{T}$ is the bipartite graph with a vertex for each elementary tensor $T^{(i)}$, a vertex for each pair of contracted legs $e = \{\ell_1, \ell_2\} \in \mathcal{E}$ (i.e., an edge/wire of the network), and edges $\{e, \ell_1\}$ and $\{e, \ell_2\}$ between tensor vertices and contracted leg vertices which are incident in the network.

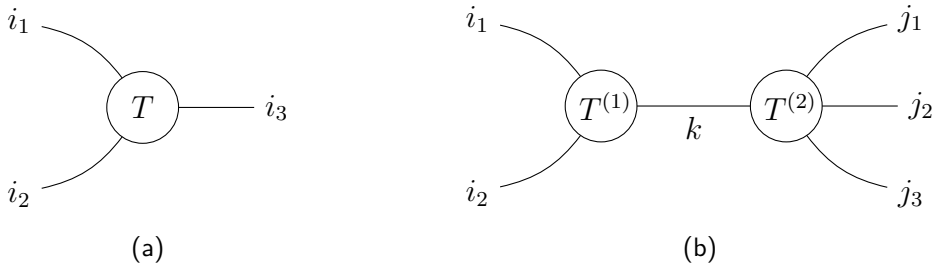


Figure 12: (a) Graphical representation of rank-3 tensor T with indices i_1, i_2, and i_3. (b) Contracting rank-3 tensor $T^{(1)}$ with rank-4 tensor $T^{(2)}$ on index k gives a rank-5 tensor $T_{i_1 i_2 j_1 j_2 j_3} := \sum_{k \in \{0,1\}} T^{(1)}_{i_1 i_2 k} T^{(2)}_{j_1 j_2 j_3 k}$.

Planar Clifford tensor network simulation problem. *The input is a Clifford tensor network $\mathcal{T}$ which has a planar underlying graph, and each elementary tensor has constant rank. If $\mathcal{T}$ is the zero tensor then the output is a flag indicating this; otherwise the output is a binary string sampled from the probability distribution obtained by measuring each qubit of $|\mathcal{T}\rangle$ in the standard basis.*

Theorem 6. *There is a classical algorithm which solves the planar Clifford tensor network simulation problem with runtime $\widetilde{O}(n^{\omega/2})$, where n is the number of elementary tensors.*

Proof. Let $\mathcal{T}$ be specified by elementary tensors $T^{(1)}, T^{(2)}, \ldots, T^{(n)}$ and pairs of contracted legs $\mathcal{E}$. Each elementary tensor defines a stabilizer state $|T^{(i)}\rangle$ via Eq. (20). The state defined by the tensor network $|\mathcal{T}\rangle$ is then given by

$$|\mathcal{T}\rangle = \prod_{e \in \mathcal{E}} (\langle 00| + \langle 11|)_e \, |T^{(1)}\rangle \otimes |T^{(2)}\rangle \otimes \ldots \otimes |T^{(n)}\rangle$$

The state $|T^{(1)}\rangle \otimes \ldots \otimes |T^{(n)}\rangle$ appearing on the right hand side has one qubit for each leg of each elementary tensor. It is an m-qubit state, where $m = O(n)$ is the total number of legs in the tensor network. We have then used the fact that contraction of two legs is equivalent to projecting the two corresponding qubits onto the Bell pair $\langle 00| + \langle 11|$.

We now use the following lemma to express each of the elementary tensors $|T^{(i)}\rangle$ as a graph state with local unitaries:

Lemma 42 (Theorem 1 from Ref. [48])**.** *A k-qubit stabilizer state $|\phi\rangle$ can be expressed as $|\phi\rangle = C_1 \otimes C_2 \otimes \ldots \otimes C_k |G\rangle$ where G is a graph state and $C_1, C_2, \ldots, C_k$ are single-qubit Clifford unitaries.*

Each elementary tensor has constant rank and therefore we may compute the single-qubit Cliffords and graph state for each elementary tensor in constant time on a classical computer. In this way, we compute graphs $G_1, G_2, \ldots, G_n$ and single-qubit Cliffords $C_1, C_2, \ldots, C_m$ such that

$$|\mathcal{T}\rangle = \prod_{e \in \mathcal{E}} (\langle 00| + \langle 11|)_e \, C_1 \otimes C_2 \otimes \ldots \otimes C_m |G_1\rangle |G_2\rangle \ldots |G_n\rangle. \tag{21}$$

Our goal is to sample a binary string from the probability distribution obtained by measuring this state in the computational basis (or determine if $\mathcal{T} = 0$). Below we show that this can be recast as an instance of the graph state simulation problem with a graph G that admits an $O(1)$-coarse-graining to a planar graph, and then Theorem 8 completes the proof.

Consider a pair of qubits $e = \{i, j\} \in \mathcal{E}$ and the corresponding state

$$C_i^\dagger \otimes C_j^\dagger \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle)$$

which appears (up to normalization) in Eq. (21). Noting that

$$\frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) = (I \otimes H) \operatorname{CZ} |{+}{+}\rangle \,,$$

we get

$$C_i^\dagger \otimes C_j^\dagger \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) = I \otimes C_j^\dagger \overline{C}_i \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) = (I \otimes D) \operatorname{CZ} |{+}{+}\rangle \,. \tag{22}$$

where $D = C_j^\dagger \overline{C}_i H$ is a single-qubit Clifford, and therefore can be written as a product of Hadamard and phase gates [37]

$$D \propto S^a H^b S^c,$$

for some $a, c \in \{0, 1, 2, 3\}$ and $b \in \{0, 1\}$. If $b = 0$ then

$$C_i^\dagger \otimes C_j^\dagger \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) \propto \operatorname{CZ} |\phi_i \otimes \phi_j\rangle \tag{23}$$

for single-qubit stabilizer states $|\phi_i\rangle = |+\rangle$ and $|\phi_j\rangle = S^{a+c}|+\rangle$. On the other hand, if $b = 1$ then we can use the Hadamard gadget from Figure 11 to get

$$\begin{aligned} C_i^\dagger \otimes C_j^\dagger \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) &\propto (I \otimes S^a H S^c) \operatorname{CZ} |{+}{+}\rangle \\ &= (I \otimes \langle +| \otimes I) \operatorname{CZ}_{ie} \operatorname{CZ}_{ej} |\phi_i \otimes \phi_e \otimes \phi_j\rangle, \end{aligned} \tag{24}$$

where $|\phi_i\rangle = |+\rangle, |\phi_j\rangle = S^a|+\rangle, |\phi_e\rangle = S^c|+\rangle$ are single-qubit stabilizer states.

For every pair of contracted legs $e = \{i, j\} \in \mathcal{E}$ in the tensor network we can compute $b \in \{0, 1\}$ and the decomposition given by either Eq. (23) or Eq. (24). Let us partition the elements of $\mathcal{E} = \mathcal{E}_0 \cup \mathcal{E}_1$ where $\mathcal{E}_0, \mathcal{E}_1$ have $b = 0, 1$ respectively. We obtain

$$|\mathcal{T}\rangle \propto \prod_{e=\{i,j\}\in\mathcal{E}_1} (\langle\phi_i \otimes \phi_e \otimes \phi_j| \operatorname{CZ}_{ie} \operatorname{CZ}_{ej} |+\rangle_e) \prod_{e=\{i,j\}\in\mathcal{E}_0} (\langle\phi_i \otimes \phi_j| \operatorname{CZ})_{ij} |G_1\rangle|G_2\rangle \dots |G_n\rangle. \tag{25}$$

In Eq. (25) we have a single-qubit stabilizer state $\langle\phi_i|$ for each contracted leg and one $\langle\phi_e|$ for each $e \in \mathcal{E}_1$. Letting $\mathcal{L} \subseteq [m]$ be the set of all contracted legs, we have

$$|\mathcal{T}\rangle = \left(\prod_{i\in\mathcal{L}} \langle\phi_i| \prod_{e\in\mathcal{E}_1} \langle\phi_e|\right) |G_{\mathcal{T}}\rangle \tag{26}$$

where $G_{\mathcal{T}}$ is a graph with $m + |\mathcal{E}_1|$ vertices, defined as follows. Start with a graph with m vertices that consists of a copy of each of the graphs $G_1, G_2, \dots, G_n$. Next add an edge between every pair $e = \{i, j\} \in \mathcal{E}_0$. Finally, add a new vertex labeled e for each $e = \{i, j\} \in \mathcal{E}_1$ and edges $\{i, e\}$ and $\{e, j\}$.

In light of Eq. (26), to solve the planar Clifford tensor network simulation problem defined by $\mathcal{T}$, it suffices to simulate single-qubit Pauli measurements on the graph state $|G_{\mathcal{T}}\rangle$ and postselect on the subset of measurement outcomes corresponding to the qubits in $\mathcal{L} \cup \mathcal{E}_1$. This is an instance of the graph state simulation problem with the graph $G_{\mathcal{T}}$.

To complete the proof, we construct an r-coarse-graining from $G_{\mathcal{T}}$ to a planar graph, derived from the underlying planar graph of the tensor network. For all $1 \le i \le n$, let the coarse-graining map all vertices of G_i to the vertex for $T^{(i)}$ in the underlying graph. Map the additional vertex created for a pair of contracted legs $e = \{k, \ell\} \in \mathcal{E}_1$ to the vertex labelled e in the underlying graph of the tensor network. If $G_{\mathcal{T}}$ has no vertex for a pair of contracted legs $e \in \mathcal{E}$, then let us remove that vertex from the underlying graph by contracting either edge incident to it; clearly the modified graph is still planar. At worst, the coarse-graining collapses all of G_i to a single vertex (corresponding to $T^{(i)}$), but this is at most the rank of the tensors $T^{(i)}$, which is constant. □

A.2 Clifford circuit simulation

Because Clifford circuit simulation is a such a common task, this section is devoted to proving an independent reduction from graph state simulation that does not require the use of postselection. We now state a formal version of the theorem we'd like to prove:

Theorem 7. *Consider n qubits arranged at the vertices of a planar graph G. Let C be an n-qubit, depth d quantum circuit composed of one- and two-qubit Clifford gates such that each two-qubit gate acts nontrivially on an edge of G. There is a classical algorithm with runtime upper bounded as $\tilde{O}(n^{\omega/2}d^{\omega})$ which produces a sample $z \in \{0, 1\}^n$ drawn from the output distribution of C.*

Before we give a new proof of this theorem, let's sketch why it follows from Theorem 6. Notice that an n-qubit Clifford circuit can be viewed as a Clifford tensor network obtained by contracting n rank-1 tensors representing the initial single-qubit states $|0\rangle$, with rank-2 and rank-4 Clifford tensors corresponding to 1- and 2-qubit Clifford gates. The underlying graph of this tensor network is not necessarily planar, but is $O(d)$-coarse-grained planar as can be seen by grouping all the gates acting on the same qubit. This suffices to prove the claimed bound on the running time.

Proof of Theorem 7. Suppose without loss of generality that all gates are either H, S, or CZ, since any one- or two-qubit Clifford gate can be rewritten as a circuit of size $O(1)$ using only these three gates. We will massage the circuit into a certain canonical form along the lines of Ref. [13]. Consider a circuit C' which is obtained from C by inserting two Hadamard gates acting on each qubit at the very beginning of the circuit, and also at the very end of the circuit. The unitaries $U_C, U_{C'}$ implemented by these circuits are the same, since

$$U_{C'} = (H^{\otimes n})^2 U_C (H^{\otimes n})^2 = U_C.$$

Moreover, the depth of C' is $d+4$. Let us say that a Hadamard gate in C' is a *middle Hadamard gate* if it is in the subcircuit $H^{\otimes n} U_C H^{\otimes n}$ containing all gates except the initial and final layer of Hadamards. It will be convenient to write h for the total number of middle Hadamard gates in C'. For any binary string $z \in \{0,1\}^h$ we also define $C'(z)$ to be the quantum circuit obtained from C' by replacing the jth middle Hadamard gate by $X^{z_j} H$, for each $j \in [h]$ (here we fix some arbitrary ordering of the middle Hadamards).

Now the circuit C' is composed of $O(nd)$ one- and two-qubit Clifford gates, and the circuit $C'(z)$ is obtained from C' by inserting $O(nd)$ Pauli X gates in locations determined by z. It follows that, given any binary string $z \in \{0,1\}^h$ we may in $O(nd)$-time compute an n-qubit Pauli $P(z)$ such that

$$U_{C'(z)} = P(z) U_{C'} = P(z) U_C. \tag{27}$$

In particular, $P(z)$ is obtained by pushing all Pauli corrections to the end of the circuit using the fact that every gate in the circuit is Clifford and therefore maps Paulis to Paulis under conjugation.

We now construct a new quantum circuit by replacing all middle Hadamard gates in C' with the Hadamard gadget depicted in Figure 11. Each Hadamard gadget introduces one ancilla qubit, so the resulting circuit has $n+h$ qubits in total. Let us now partition these $n+h$ qubits as $[n+h] = A_1 \cup A_2 \cup \ldots \cup A_n$ where subsets $A_j \subseteq [n+h]$ are defined as follows. For each $j \in [n]$, the set A_j contains qubit j as well each ancilla qubit from a gadget replacing a Hadamard gate acting on the jth qubit in C'. Note that $|A_j| \leq d+4$ for all $j \in [n]$ since C' has depth at most $d+4$.

Since all gates in C' were either H, S or CZ gates, after inserting the Hadamard gadgets we are left with a circuit of the form

$$V = H^{\otimes n+h} \prod_{j \in M} S_j \prod_{\{i,j\} \in E(G')} \mathrm{CZ}_{ij}\, H^{\otimes n+h},$$

for some subset $M \subseteq [n+h]$ and graph G' with vertices labeled by $[n+h]$. Moreover, there exists a $(d+4)$-coarse-graining from G' to G. Indeed, such a coarse-graining is defined by $\varphi\colon [n+h] \to [n]$ where $\varphi(k)$ is the unique $j \in [n]$ satisfying $k \in A_j$. As noted above, $|A_j| \leq d+4$ for all j. One can also straightforwardly verify that $\{u,v\} \in E(G')$ only if either (a) $u, v \in A_j$ for some j, or (b) $u \in A_j$ and $v \in A_i$ for some $\{i,j\} \in E(G)$.

We may therefore use the algorithm from Theorem 8 to obtain $x \in \{0,1\}^n$ and $z \in \{0,1\}^h$ such that $\langle x| \otimes \langle z| V |0^{n+h}\rangle \neq 0$. (Note that here we only need to solve an instance of the graph state simulation problem with $\mathcal{P} = \varnothing$.) Using the identities in Fig. 11 we then have

$$\langle x| \otimes \langle z| V |0^{n+h}\rangle = \langle x| U_{C'(z)} |0^n\rangle$$

Finally, as noted above, we may compute (in $O(nd)$ time) an n-qubit Pauli $P(z)$ such that

$$\langle x|U_{C'(z)}|0^n\rangle = \langle x|P(z)U_C|0^n\rangle \neq 0.$$

Letting $g \in \{0,1\}^n$ be such that $P(z)|x\rangle \propto |g\rangle$ we see that $\langle g|U_C|0^n\rangle \neq 0$. Finally, we select a random $s \in \{0,1\}^n$ and let

$$Q(s) = U_C Z(s) U_C^\dagger.$$

Note that $Q(s)$ is a random stabilizer of $U_C|0^n\rangle$ and can be computed in time $O(nd)$, since U_C has $O(nd)$ gates. By Claim 9, the binary string y such that $|y\rangle \propto Q(s)|g\rangle$ is a random sample from the output distribution of C. This y is the output of the algorithm. The total runtime is $\widetilde{O}(n^{\omega/2}d^\omega)$ since all steps except for the graph state simulation take time $O(nd)$. □

B Software Implementation

This section is about our software implementation of the graph state simulation problem for the $n^{\frac{1}{2}} \times n^{\frac{1}{2}}$ grid graph. The goal was to show that the asymptotic speedup we predict for the grid in Theorem 2 is actually practical, i.e., that it is feasible to implement and that the increased complexity of the algorithm does not hurt the runtime too much for small grids. We do not implement fast matrix multiplication for simplicity, so in the theorem we may set $\omega \to 3$ and avoid some extra log factors to obtain an $O(n^{3/2})$-time algorithm.

Recall that this running time is achieved by a divide-and-conquer recursive algorithm described in the example on page 9. We compare the running time of this algorithm with the naïve algorithm and the left-to-right sweep algorithm in Figure 1. In each case, for a given n, the algorithm was translated into a circuit (on n qubits for the naïve algorithm, $O(\sqrt{n})$ for the others), and then fed to the same tableau-based Clifford simulator (CHP++ [25]) running on a modern desktop computer.[22] We make one concession to translate the divide-and-conquer algorithm to a circuit: each gate includes information about which qubits (represented as a interval) could be currently entangled with its inputs. In this way, we can represent the multiple subtableaux (demanded by a divide-and-conquer strategy) as separate intervals of qubits within the full tableau, without the performance hit that would come from scanning the entire tableau for each operation. Finally, in our tests, the measurement basis (`X` or `Y`) for each qubit is chosen at random; we expect (and conjecture) that this is induces near worst-case performance in the three algorithms.

In the log-log plot shown in Figure 13, the line of best fit for all three algorithms suggests a better runtime than we predict. Namely, the naïve algorithm scales as $n^{1.7}$ whereas the theoretical guarantee is $O(n^3)$; the left-to-right sweep algorithm scales as $n^{1.5}$ with theoretical guarantee only $O(n^2)$; and the recursive algorithm scales as $n^{1.2}$ with theoretical guarantee only $O(n^{1.5})$. Nevertheless, we believe this is simply due to small-n effects.

In an effort to improve the performance of the recursive algorithm as much as possible, our software implementation differs from that given in the example in two important ways:

[22] The experiments were conducted on a desktop computer with an AMD Ryzen 5 3600 processor and 16GB of RAM.

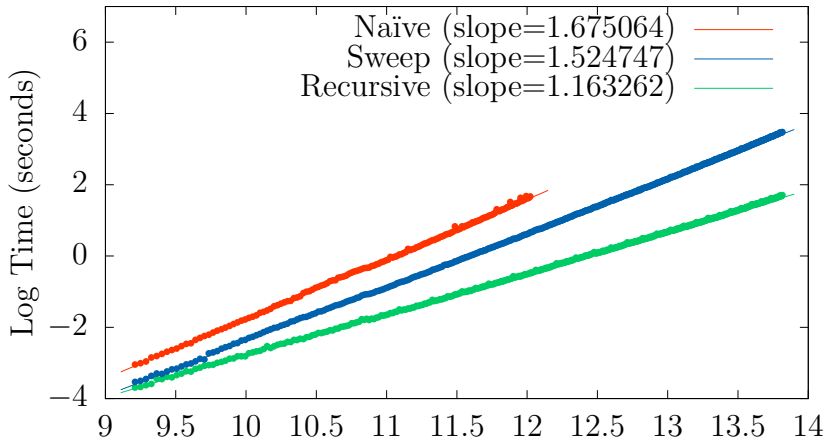


Figure 13: Log-log plot of the simulation time (seconds) vs. $\ln(n)$ (i.e., log the total number of qubits in the $\sqrt{n} \times \sqrt{n}$ grid). The slope is calculated as a linear regression of the data points, and shows the empirical scaling with number of qubits. The data depicted is the same as that in Figure 1 except only for $\sqrt{n}$ at least 100.

1. In each recursive step, we split the grid into two (almost) identical halves rather than four quarters as in the example. Of course, in the next step we split those halves again in the opposite direction (the rule is to split the longer dimension), so we end up with four quarters anyway.
2. We eliminate the frame qubits between the two halves. The recursive calls return the state of the perimeter qubits for each half, connect the two perimeters directly with CZs at the seam (rather than to a separating frame), and measure qubits that are no longer part of the perimeter. This leads to fewer active qubits at the top-most levels of the recursion, which is important because that part of the computation dominates the runtime.

Similarly, we use a tableau-based Clifford simulator loosely based on Aaronson's CHP [17], but it differs in two important ways:

1. When a qubit is measured, we update the tableau to be a direct sum, to reflect that the qubit is no longer entangled with the rest. This requires more work, but helps clearly separate the past interactions of the qubit from the future, facilitating reuse.
2. Additional gates have been added: CZ because it is essential to graph states, single-qubit gates to translate Y-basis measurement into standard basis measurement, and SWAP gates.

C Correction Subroutine Example

In this section we work through an explicit example of the correction subroutine described in Section 3.3.1. The input is shown in Figure 14. We shall consider the case in which all four data qubits are to be measured in the `X` basis (i.e., $U_{\text{bases}} = H^{\otimes 4}$). The quantum circuit $\mathcal{C}'$ is shown in Figure 15. Note that vertices of the graph are labelled by uppercase letters while qubits of $\mathcal{C}'$ are labelled by lowercase letters, with the data qubit and merge

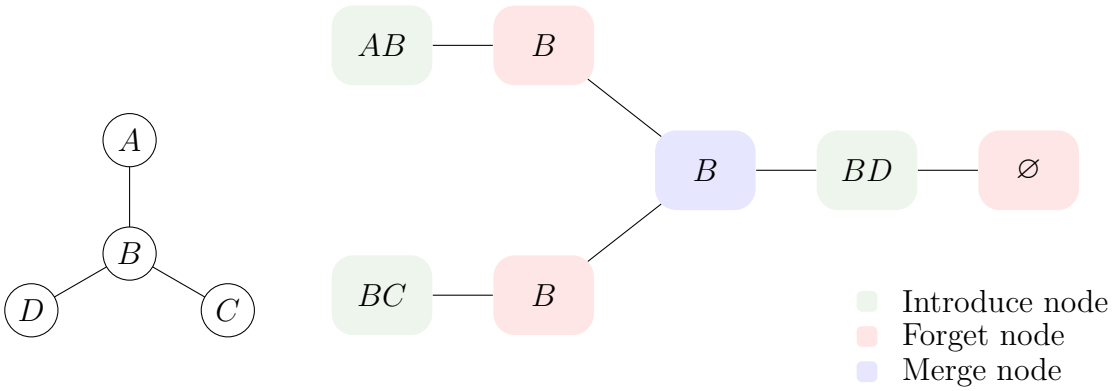


Figure 14: *Left:* Input graph to the graph state simulation problem. *Right:* A nice tree decomposition.

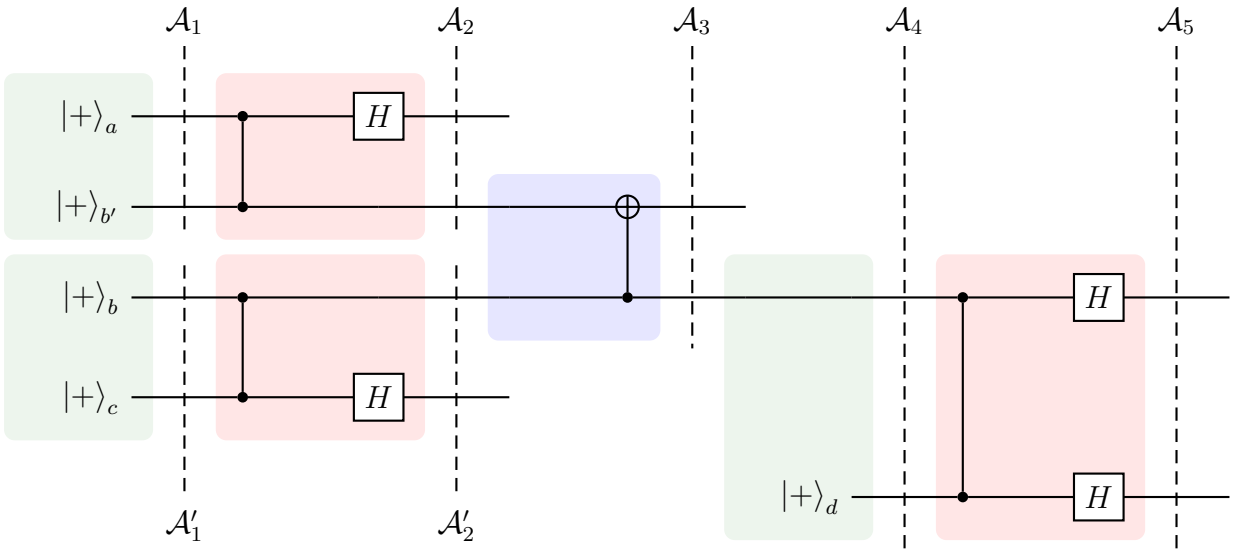


Figure 15: Example circuit $\mathcal{C}'$ constructed from the graph and tree decomposition in Figure 14. Each affine space is associated with the bag immediately to the left of its dashed line. Qubits are ordered in the same way as π, from top to bottom.

ancilla for vertex B being labelled b and b', respectively. Let us take $\pi = (*1{*}{*}1, {*}{*}{*}{*}{*})$ as the desired pattern, where qubits are ordered $ab'bcd$. That is, we would like P_{cor} to have nonzero $\mathtt{X}$-components acting on qubits b' and d.

We first compute the affine spaces $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_5$, which correspond to sections of $\mathcal{C}$ shown with dashed lines in Figure 15. We will omit the computation of $\mathcal{A}'_1$ and $\mathcal{A}'_2$ (which are also shown in Figure 15), because by symmetry they are isomorphic to $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. The leaf node containing A and B is an introduce node, so we set

$$\mathcal{A}_1 = \left\{ \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} x : x \in \mathbb{F}_2^2 \right\},$$

corresponding to the stabilizer generators $\mathtt{X}_a$ and $\mathtt{X}_{b'}$ of $|+\rangle_a \otimes |+\rangle_{b'}$. Throughout this example, we will order the rows of matrices first by grouping $\mathtt{X}$- and $\mathtt{Z}$-components together, and then in the same descending order as in π and $\mathcal{C}'$. In the matrix above, the first and third rows correspond to qubit a while the second and fourth correspond to b'.

Next we conjugate $\mathcal{A}_1$ by $(H \otimes I)\mathrm{CZ}$. Since qubit a is forgotten, we would in general need to enforce $\pi_a^{(X)}$ and $\pi_a^{(Z)}$, but since $\pi_a^{(X)} = \pi_a^{(Z)} = *$, there is nothing to be done. Therefore we have

$$\mathcal{A}_2 = ((H \otimes I)\mathrm{CZ})\mathcal{A}_1((H \otimes I)\mathrm{CZ})^\dagger = \left\{ \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} x : x \in \mathbb{F}_2^2 \right\}.$$

To compute $\mathcal{A}_3$, we first remove the rows of the matrix in $\mathcal{A}_2$ corresponding to a (i.e. the first and third) to obtain $\{\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)x : x \in \mathbb{F}_2^2\}$. By symmetry, we would obtain the same set after removing qubit c from $\mathcal{A}_2'$. We then take the direct product

$$\left\{ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} x : x \in \mathbb{F}_2^2 \right\} \oplus \left\{ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} x : x \in \mathbb{F}_2^2 \right\} \cong \left\{ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} x : x \in \mathbb{F}_2^4 \right\}.$$

We have rearranged the right hand side matrix so that the X-components form the first two rows and the Z-components the last two, in contrast with Eq. (15). Next, we apply the CNOT gate to obtain

$$\left\{ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} x : x \in \mathbb{F}_2^4 \right\}. \tag{28}$$

Finally, we must enforce the pattern on the merge ancilla b'. Since $\pi_{b'}^{(X)} = 1$, this means finding the subset of elements in Eq. (28) that have the form $(1, *, *, *)^T$. Basic linear algebra tells us that x must take the form $x = e_4 + y_1 e_1 + y_2(e_2 + e_4) + y_3 e_3$, where e_j is the usual standard basis vector and $y_1, y_2, y_3 \in \{0, 1\}$ are free. The desired subset of Eq. (28) therefore is

$$\mathcal{A}_3 = \left\{ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} y + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right) : y \in \mathbb{F}_2^3 \right\} = \left\{ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} y + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} : y \in \mathbb{F}_2^3 \right\}.$$

We then restrict $\mathcal{A}_3$ to coordinates corresponding to b to obtain $\{\left(\begin{smallmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{smallmatrix}\right)x + \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right) : x \in \mathbb{F}_2^3\}$. Since the matrix has more columns than rows, we search for a maximum linearly independent subset of columns. By inspection, we can just remove the third column to get $\{\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)x + \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right) : x \in \mathbb{F}_2^2\}$. Taking a direct product with the affine space $\mathcal{I}$ for the stabilizer generator of $|+\rangle_d$ gives us

$$\mathcal{A}_4 = \left\{ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} : x \in \mathbb{F}_2^3 \right\}.$$

To compute $\mathcal{A}_5$, we conjugate the affine space by the CZ and Hadamard gates to obtain

$$\left\{ \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} : x \in \mathbb{F}_2^3 \right\},$$

and since $\pi_d^{(X)} = 1$, we search for all elements in this space of the form $(*, 1, *, *)^T$. Using the same approach as in the computation of $\mathcal{A}_3$ we arrive at

$$\mathcal{A}_5 = \left\{ \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} y + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} : y \in \mathbb{F}_2^2 \right\}.$$

We are now ready to construct P_{cor}. We first choose a random element from $\mathcal{A}_5$, say, $(0, 1, 1, 0)^T$. This fixes two tensor elements of the Pauli correction: $P_{\mathrm{cor}} = ? \otimes ? \otimes \mathtt{Z} \otimes ? \otimes \mathtt{X}$ (with the question marks to be determined below).

Since $((H \otimes H)CZ)^\dagger \mathtt{Z} \otimes \mathtt{X}((H \otimes H)CZ) = \mathtt{X} \otimes I$, pushing the chosen element of $\mathcal{A}_5$ backward through the H and CZ gates gives us $(1,0,0,0)^T \in \mathcal{A}_4$. We then remove qubit d to obtain $(1,0)^T$ and select a random element in $\mathcal{A}_3$ of the form $(*,1,*,0)^T$, say, $(1,1,0,0)^T$. Since b' is measured immediately after the merge node, we can fill in another tensor element in the Pauli correction: $P_{\text{cor}} = ? \otimes \mathtt{X} \otimes \mathtt{Z} \otimes ? \otimes \mathtt{X}$.

Pushing $(1,1,0,0)^T$ back through the CNOT gate gives $(0,1,0,0)^T$. Removing b gives us $(0,0)^T$, so we search for an element of $\mathcal{A}_2$ of the form $(*,0,*,0)^T$. The only such element is $(0,0,0,0)^T$, so we now have $P_{\text{cor}} = I \otimes \mathtt{X} \otimes \mathtt{Z} \otimes ? \otimes \mathtt{X}$. Similarly, we can reconstruct the tensor element corresponding to qubit c and find that $P_{\text{cor}} = I \otimes \mathtt{X} \otimes \mathtt{Z} \otimes \mathtt{X} \otimes \mathtt{X}$ is a valid choice; indeed, we have $(01011, 00100) \in \Pi$. We do not need to use $\mathcal{A}_1$ or $\mathcal{A}_1'$ as we now have all tensor elements of P_{cor}.

D Linear System Solving

Theorem 26. *There exists an algorithm to find a uniformly random solution to $Ay = b$ (or report that no solution exists) in time $O(\|T\|_\omega^\omega)$ given a graph G with adjacency matrix $A \in \mathbb{F}_2^{n\times n}$, a vector $b \in \mathbb{F}_2^n$, and a nice tree decomposition T for G.*

Proof. Construct the circuit $\mathcal{C}$ associated with G and its tree decomposition (in time linear in $\|T\|_1$). Claim 21 says that $\mathcal{C}\,|+^{n_t}\rangle$ is a graph state $|G'\rangle$ for $G' = (V', E')$ which extends G in the sense that $V' = V \cup M$ (where M are the new vertices associated with the merge ancillas, i.e., $|M| = n_a$) and the adjacency matrix is

$$A' = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix} \tag{29}$$

where A is the adjacency matrix of G, and for some $n \times n_a$ matrix B, $n_a \times n_a$ matrix C.

Suppose $x = x_V x_M, z = z_V z_M \in \mathbb{F}_2^{|V'|}$ are the $\mathtt{X}$- and $\mathtt{Z}$-components of a Pauli stabilizer for $|G'\rangle$. Since $|G'\rangle$ is a graph state, the components are related by the adjacency matrix: $z = A'x$. By Eq. (29),

$$\begin{pmatrix} z_V \\ z_M \end{pmatrix} = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix} \begin{pmatrix} x_V \\ x_M \end{pmatrix} = \begin{pmatrix} Ax_V + Bx_M \\ B^T x_V + Cx_M \end{pmatrix}. \tag{30}$$

We claim there is a bijection between the set of solutions to $Ay = b$ and the set of solutions to Eq. (30) which satisfy $z_V = b$ and $x_M = 0$ (i.e., stabilizers of $|G'\rangle$ of the form $(*_V 0_M, b_V *_M)$). We will prove this claim by describing injective maps from each set to the other.

First, suppose we are given y such that $Ay = b$. Then setting $x_V = y$, $x_M = 0$, $z_V = b$, and $z_M = B^T y$ satisfies Eq. (30). Because we have set $x_V = y$, for two distinct solutions y, y' we will find distinct stabilizers, and so this map is injective. Conversely, if we have a solution to Eq. (30), then the first equation, $z_V = Ax_V + Bx_M$ immediately implies $Ax_V = b$ since $z_V = b$ and $x_M = 0$. If $(x_V 0_M, b_V z_M)$ and $(x_V 0_M, b_V z'_M)$ represent stabilizers of $|G'\rangle$, then their product, represented by $(0_V 0_M, 0_V(z_M \oplus z'_M))$, is also a stabilizer. However because $|G'\rangle$ is a graph state, its only stabilizer with trivial $\mathtt{X}$-component is the identity, which implies that $z_M = z'_M$, and thus this map is also injective.

Therefore, we may apply Theorem 23 to sample a uniformly random stabilizer of $|G'\rangle = \mathcal{C}\,|+^{n_t}\rangle$ respecting the pattern $(*_V 0_M, b_V *_M)$ (or detect that no such stabilizer exists) in $O(\|T\|_\omega^\omega)$ time. If a solution exists, then by the previous paragraph, it immediately yields a uniformly random solution to $Ay = b$. □