npj | quantum information

www.nature.com/npjqi

ARTICLE OPEN

Check for updates

6-qubit optimal Clifford circuits

Sergey Bravyi[1], Joseph A. Latone[2] and Dmitri Maslov[1]✉

Clifford group lies at the core of quantum computation—it underlies quantum error correction, its elements can be used to perform magic state distillation and they form randomized benchmarking protocols, Clifford group is used to study quantum entanglement, and more. The ability to utilize Clifford group elements in practice relies heavily on the efficiency of their circuit-level implementation. Finding short circuits is a hard problem; despite Clifford group being finite, its size grows quickly with the number of qubits n, limiting known optimal implementations to $n = 4$ qubits. For $n = 6$, the number of Clifford group elements is about 2.1×10^{23}. In this paper, we report a set of algorithms, along with their C++ implementation, that implicitly synthesize optimal circuits for all 6-qubit Clifford group elements by storing a subset of the latter in a database of size 2.1TB (1kB = 1024B). We demonstrate how to extract arbitrary optimal 6-qubit Clifford circuit in 0.0009358 and 0.0006274 s using consumer- and enterprise-grade computers (hardware) respectively, while relying on this database. We use this implementation to establish a new example of quantum advantage by Clifford circuits over CNOT gate circuits and find optimal Clifford 2-designs for up to 4 qubits.

npj Quantum Information (2022)8:79 ; https://doi.org/10.1038/s41534-022-00583-7

INTRODUCTION

Quantum computations are studied for their promise to outperform classical counterparts for certain kinds of computations[1]. The Clifford group is an important finite subgroup of the full unitary group, describing the set of quantum computations. Despite being possible to simulate classically[2,3] by a low degree polynomial and having a simple structure[4] (admitting efficient parametrization and being possible to compute by linear depth circuits), the group is most famous for lying at the core of quantum error correction[1], which is believed to be necessary for scalable quantum computation. Restricted to the study of fault-tolerance, Clifford group plays multiple roles still. To illustrate, all (standard) encoding circuits are Clifford[1], and so are the circuits for state distillation[5,6], necessary for fault-tolerant implementation of non-Clifford gates. Clifford circuits lie at the core of randomized benchmarking protocols[7,8]. Other use cases include shadow tomography[9,10], the study of entanglement[1,11], and quantum data hiding[12]. It is perhaps fair to regard the Clifford group as one of the most visible and important subgroups of the group of all quantum computations.

Superconducting circuits and trapped ions are two technological frameworks that produced a stream of (universal prototype) programmable quantum computers, publicly available since the year 2016. Each technology comes in a range of flavors: e.g., superconducting circuits can be based on phase, charge, or flux qubits (or even hybrid kinds), and rely on various qubit coupling mechanisms, and trapped ions can be based on various ion species and rely on different approaches to the two-qubit gates (e.g., stationary vs mobile qubits). However, no matter the specific flavor, all prototype quantum computers based on these two approaches share one property[13,14]: the two-qubit gate has notably lower fidelity than a single-qubit gate. Thus, to the first degree of approximation, the fidelity of an entire quantum computation depends on the number of two-qubit gates it uses. To make a more subtle point, since the single-qubit gates are most frequently implemented by pulses with real-valued control parameters, the number of two-qubit gates in a circuit upper bounds the number of the single-qubit gates (up to a constant factor), meaning the reduction of the two-qubit gate count likely leads to the reduction in the number of single-qubit gates. We further note that the CNOT gates are available natively (i.e., requiring the minimal number of one two-qubit physical-level interaction) in both superconducting circuits and trapped ions technologies. Finally, recall that the physical-level entangling pulses frequently take the form of *XX*, *ZX*, and *ZZ*, requiring single-qubit corrections to turn those interactions into commonly used CNOT or CZ gates. This means that minimizing single-qubit gate count in an abstract circuit may not directly minimize the number of single-qubit physical pulses, since the single-qubit gates will be reshuffled during technology mapping. This justifies our focus on minimizing the CNOT gate count, selected as the optimization criterion in this paper.

In this paper, we study the problem of optimal synthesis of Clifford circuits. Since the problem of optimal circuit synthesis is hard, we restrict our attention to a small number of qubits, at most 6. The number of Clifford group elements over 6 qubits, 2.1×10^{23}, is still very large, and we employ a range of techniques to make the search tractable using modern computers. At the core of our approach is a mechanism to break down the set of Clifford unitaries into a set of classes containing unitaries sharing a similar optimal circuit structure, efficient computation of the canonical representative of each class, and efficient manipulation of class members and the database of canonical representatives.

We define the n-qubit Clifford group $\mathcal{C}_n$ as the group of $2n \times 2n$ symplectic matrices M over the two-element field $\mathbb{F}_2$, $\mathrm{Sp}(2n, \mathbb{F}_2) := \{M : M^T \Omega_n M = \Omega_n\}$, where M^T denotes the transpose matrix, Ω_n is the matrix $\begin{pmatrix} 0 & I_n \\ I_n & 0 \end{pmatrix}$, and I_n is the $n \times n$ identity matrix. Symplectic matrices are equivalent to and alternatively known as the tableaux[3]. The size of the symplectic group is $|\mathrm{Sp}(2n, \mathbb{F}_2)| = 2^{n^2} \prod_{j=1}^{n} (2^{2j} - 1)$, which for the purpose of this paper implies $|\mathcal{C}_6| = 208, 114, 637, 736, 580, 743, 168, 000 \approx 2.1 \times 10^{23}$ and assigns the numeric value to the size of the search space we are exploring.

[1]IBM Quantum, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA. [2]IBM Quantum, Almaden Research Center, San Jose, CA 95120, USA.
✉email: dmitri.maslov@ibm.com

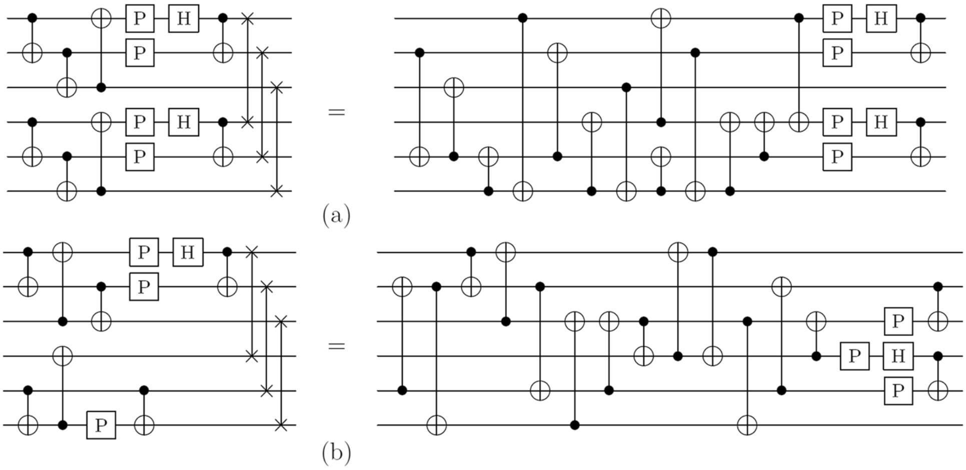


Fig. 1 All most expensive 6-qubit Clifford unitaries requiring 15 entangling gates (up to left and right multiplication by the single-qubit gates and qubit relabeling). a left: a compact representation in the form $(U \otimes U)$SWAP, right: its optimal implementation. **b** left: a compact representation in the form $(U' \otimes V')$ SWAP, right: its optimal implementation. Not illustrated is the cyclic SWAP of all 6 qubits, that also requires 15 entangling gates.

Table 1. The distribution of the number of 6-qubit Clifford unitaries across the entangling gate cost.

CNOT cost	Number of 6-qubit Clifford unitaries
0	46,656
1	6,298,560
2	554,273,280
3	39,045,473,280
4	2,365,081,986,240
5	126,526,140,927,360
6	5,998,793,185,860,480
7	249,378,588,704,827,008
8	8,870,235,256,471,637,952
9	255,646,483,904,239,690,752
10	5,278,109,585,506,533,785,088
11	58,697,087,161,047,579,538,560
12	135,876,260,385,953,644,020,480
13	7,998,401,853,543,422,302,848
14	6,525,042,824,342,016
15	13,308,157,440
	208,114,637,736,580,743,168,000

Tableau representation is particularly useful since it allows to define quantum gates and circuits directly without the need to resort to standard definitions in quantum information that employ $2^n \times 2^n$ unitary matrices[1]. Indeed,

- the Hadamard gate H on qubit k can be defined as the $2n \times 2n$ identity matrix with swapped columns k and $n+k$,
- the Phase gate P on qubit k can be defined as the addition of column k to column $n+k$ in the $2n \times 2n$ identity matrix,
- the CNOT gate with control qubit k and target j performs simultaneous addition of column k to column j and column $n+j$ to column $n+k$ in the $2n \times 2n$ identity matrix,

and circuits are matrix multiplications. The computational completeness of the {H, P, CNOT} library is readily exposed by the ability to apply Gaussian elimination to obtain arbitrary symplectic matrix as a product of gates. An additional advantage of such a definition of gates and circuits comes from displaying the capacity to implement transformations by Clifford gates efficiently by a computer program.

As a side note, we highlight that each element of the Clifford group $\mathcal{C}_n$ defines an equivalence class of $2^n \times 2^n$ unitary matrices realizable by the circuits over H, P, and CNOT gates (defined, in turn, via unitary matrices[1]). A pair of unitary matrices is considered equivalent if they can be mapped to each other by the left (or right) multiplication with single-qubit Pauli gates and overall phase factors. Since we focus on the minimization of the two-qubit gate count, Pauli gates and phase factors can be safely factored out. Had Pauli gates been included in the Clifford group, the search space size for $n = 6$ would read 8.5×10^{26}.

RESULTS

6-qubit optimal Clifford circuits

The distribution of the number of equivalence classes across CNOT gate costs is shown in Table 6. For the number of qubits 2 through 5 the most complex function to implement is unique (within the equivalence class definition), and it is equivalent to a cyclic permutation of qubits. For $n = 6$, the cyclic permutation is one of three such functions; the other two are illustrated in Fig. 1. The small number of equivalence classes for a small number of qubits implies an efficient formula (based on ReduceU) to compute the CNOT cost of a small Clifford unitary.

We ran a script to calculate the distribution of the number of Clifford group elements across optimal CNOT gate costs. Given the database, it took a few days to collect the data using an HPC system. This computation is highly parallelizable, and the runtime can be reduced significantly with many processors, e.g., GPUs; we have not pursued those reductions. The results are reported in Table 1.

We used the database to look for examples of quantum Clifford advantage over classical reversible CNOT circuits, meaning optimal CNOT circuits that can be implemented with fewer entangling gates as a Clifford circuit. We found one such example, illustrated in Fig. 2, that gives a reduction of 14 gates into 12, improving the 8 to 7 reduction seen earlier[4]$\left(\text{indeed, } \frac{14}{12} > \frac{8}{7}\right)$.

The compiler was benchmarked using both consumer-grade and enterprise-grade systems for a test set with 10, 000 elements of the Clifford group $\mathcal{C}_6$. Each element was generated by a Clifford circuit with 600 randomly chosen gates over the library {H, P, CNOT}. The number of gates was selected to be high enough to effect a close to random uniform distribution over the elements of

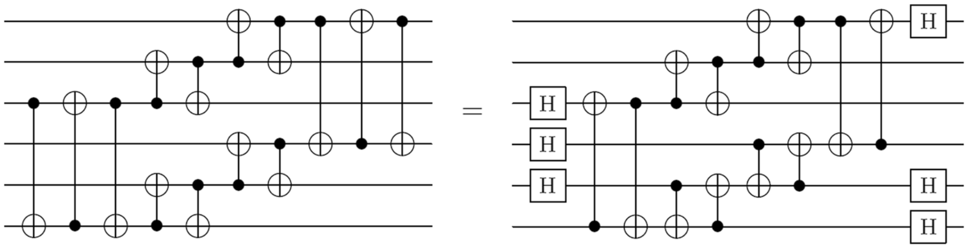


Fig. 2 Quantum advantage by Clifford circuits. An optimal CNOT gate circuit (left) can be implemented with fewer entangling gates as an optimal Clifford circuit (right).

Table 2. Average runtime for optimally compiling n-qubit Clifford operators with the full database of reduced elements loaded into RAM. The runtime was measured on MacBook Pro laptop (early 2015 model) with Intel® i7-5557U 3.1GHz CPU and 16GB RAM.

Qubits	Average runtime (s)	Database size (bytes)
$n = 5$	0.0002922	69,162,544
$n = 4$	0.0001928	37,808
$n = 3$	0.0001351	432
$n = 2$	0.00007968	64

the group $\mathcal{C}_6$. We observed that such random test set is dominated by the elements with costs 11 and 12. The compiler runtime reported below is the time required to obtain optimal circuits for all test set elements divided by the size of the test set. We observed the runtime of 0.0009358 s for a laptop with Intel® i7-1068NG7 2.3 GHz CPU and 16GB RAM with USB-C-attached consumer-grade SSD. The search relies on the database stored on SSD, and a 2.5GB index in RAM, see the section "Software tricks" for details. The time reported measures hot cache performance, cold cache performance reads 0.003708 s per an optimal circuit, on average. The compiler performance improves when the entire database can be stored in RAM. We observed the hot cache runtime of approximately 0.0006274 s for a server with Intel® Xeon® 128-CPU E7-4850 v4 @ 2.10GHz and 6TB RAM. The process of loading the full database into RAM took approximately 2 h.

This performance allows to use our implementation to obtain individual circuits and entire randomized benchmarking schedules in mere seconds using consumer-grade hardware as well as online via a web interface. For the use in demanding applications such as peep-hole optimization of large circuits, we suggest relying on large-RAM commercial-grade servers and note that it takes roughly half the time to look up the cost without computing the optimal circuit (the procedure that would likely get called most frequently during peep-holing).

The average runtime of our compiler for random n-qubit Clifford operators with $n \le 5$ is shown in Table 2.

Optimal 2-designs

Unitary designs[15] are probability distributions on the unitary group that reproduce low-order moments of the Haar (uniform) distribution. Of particular interest are unitary designs that can be efficiently implemented by quantum circuits[16]. Such designs can serve as a substitute for the Haar distribution in certain randomized quantum protocols such as data hiding[12], estimating fidelity of quantum operations[8,17], and quantum state tomography[10]. In this section, we leverage the database of reduced Clifford elements to construct optimal unitary designs that have the minimum average cost, subject to the constraint that all elements of the design are Clifford operators.

Let $U(2^n)$ be the group of unitary complex matrices of size $2^n \times 2^n$. Suppose $\mathcal{D} \subseteq U(2^n)$ is a finite subset and $\mu : \mathcal{D} \to \mathbb{R}_+$ is a probability distribution on $\mathcal{D}$. The pair $(\mathcal{D}, \mu)$ is called a unitary 2-design[18] if

$$\sum_{\hat{U} \in \mathcal{D}} \mu(\hat{U})(\hat{U}^\dagger \hat{A} \hat{U}) \otimes (\hat{U}^\dagger \hat{B} \hat{U}) = \int_{U(2^n)} (\hat{U}^\dagger \hat{A} \hat{U}) \otimes (\hat{U}^\dagger \hat{B} \hat{U}) \mathrm{d}U \tag{1}$$

for any complex matrices $\hat{A}$ and $\hat{B}$. Here the tensor product separates two n-qubit registers and the integral in the right-hand side of Eq. (1) is the average over the Haar distribution on the unitary group $U(2^n)$. We reserve the hat notation for complex unitary matrices to avoid confusion with binary symplectic matrices considered in the rest of the paper. Below we choose $\mathcal{D}$ to be the n-qubit Clifford group and construct a probability distribution μ that minimizes the average cost

$$\sum_{\hat{U} \in \mathcal{D}} \mu(\hat{U}) \cdot \mathrm{cost}(\hat{U}), \tag{2}$$

subject to the constraint that $(\mathcal{D}, \mu)$ is a unitary 2-design. Here $\mathrm{cost}(\hat{U})$ is the minimum number of the CNOT gates required to implement $\hat{U}$ by a quantum circuit composed of the Hadamard, Phase, and CNOT gates.

Since Pauli operators have zero cost, we can assume wlog that the optimal solution μ is Pauli-invariant, i.e., $\mu(\hat{U}) = \mu(\hat{U}\hat{O})$ for all n-qubit Pauli operators $\hat{O}$. As defined earlier, the unitary version of the n-qubit Clifford group is isomorphic to $\mathcal{C}_n \times \{I, X, Y, Z\}^n$. Here we ignore the overall phase factors. Define the probability distribution $\pi : \mathcal{C}_n \to \mathbb{R}_+$ such that $\pi(U) = 4^n \mu(U \times P)$ for all $U \in \mathcal{C}_n$ and $P \in \{I, X, Y, Z\}^n$. The distribution π is well-defined whenever μ is Pauli-invariant. In the section "Pauli mixing constraint", we show that μ is a Clifford 2-design iff π obeys the so-called Pauli mixing constraint[16]

$$\mathrm{Pr}_{U \sim \pi}[Ux = y] := \sum_{U \in \mathcal{C}_n :\, Ux = y} \pi(U) = \frac{1}{4^n - 1} \quad \text{for all non-zero vectors } x, y \in \{0,1\}^{2n}. \tag{3}$$

Furthermore, μ has the average cost

$$\sum_{U \in \mathcal{C}_n} \pi(U) \cdot \mathrm{cost}(U). \tag{4}$$

Thus it suffices to minimize the average cost Eq. (4) over variables $\pi(U) \ge 0$ subject to the normalization constraint $\sum_{U \in \mathcal{C}_n} \pi(U) = 1$ and the Pauli mixing constraint, Eq. (3). This gives a linear program with $|\mathcal{C}_n|$ variables.

The next step is to reduce the number of variables and the number of constraints in the linear program. Suppose π is a Pauli mixing distribution on $\mathcal{C}_n$, that is, π obeys Eq. (3). Define a symmetrized version of π as follows. First, sample $U \in \mathcal{C}_n$ from the distribution π. Second, sample $W \in S_n$ and $L, R \in \mathcal{C}_n^0$ from the uniform distribution on the respective groups. Finally, output $U' = LW^{-1}UWR$. The probability distribution of U' is given by

$$\pi'(U') = \frac{1}{6^{2n} n!} \sum_{L, R \in \mathcal{C}_n^0} \sum_{W \in S_n} \pi(WL^{-1}U'R^{-1}W^{-1}).$$

Since the cost is invariant under a qubit relabeling and left/right multiplications by the elements of local subgroup $\mathcal{C}_n^0$, the distributions π and π' have the same average cost. We claim that π' is Pauli mixing. Indeed, pick any non-zero vectors $x, y \in \{0,1\}^{2n}$, a qubit permutation $W \in S_n$, and local Cliffords $L, R \in \mathcal{C}_n^0$. Then

$$\mathrm{Pr}_{U\sim\pi}[LW^{-1}UWRx = y] = \mathrm{Pr}_{U\sim\pi}[Ux' = y'] = \frac{1}{4^n - 1}, \tag{5}$$

where $x' = WRx \neq 0$ and $y' = WL^{-1}y \neq 0$. The last equality in Eq. (5) follows from the assumption that π is Pauli mixing. Thus π' is a convex linear combination of Pauli mixing distributions, that is, π' itself is Pauli mixing.

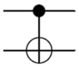
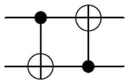
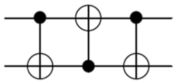
The above shows that an optimal Clifford 2-design can be found by minimizing the average cost Eq. (4) over symmetric Pauli mixing distributions π such that the probability $\pi(U)$ depends only on the equivalence class $[U]$ that contains U. Such distribution π can be compactly specified by considering the set of reduced elements

$$\mathcal{R}_n := \{ReduceU(U) : U \in \mathcal{C}_n\}.$$

Given a reduced element $U \in \mathcal{R}_n$, define the probability distribution

$$\eta(U) = \sum_{U' \in [U]} \pi(U') = \pi(U) \cdot |[U]|.$$

Table 3. Optimal two-qubit Clifford 2-design with the average cost 1.5. This coincides with the average cost of the full Clifford group $\mathcal{C}_2$.

circuit U_j	probability $\eta(U_j)$
	0.6
	0.3
	0.1

Note that η is a probability distribution on $\mathcal{R}_n$ since each equivalence class $[U]$ contains a unique reduced element, see the section "Computation of ReduceU". For brevity, we will refer to η as a reduced distribution. The average cost of the original distribution π depends only on η and can be computed using the formula

$$\sum_{U\in\mathcal{R}_n} \eta(U) \cdot \mathrm{cost}(U). \tag{6}$$

It remains to express the Pauli mixing constraint in terms of the reduced distribution η. Given a reduced element $U \in \mathcal{R}_n$ and non-zero vectors $x, y \in \{0,1\}^{2n}$, define the quantity

$$g(U,x,y) = \frac{\#\{U' \in [U] : U'x = y\}}{|[U]|}.$$

In words, $g(U,x,y)$ is the probability that a random uniformly distributed element of the equivalence class $[U]$ maps x to y. Then π is Pauli mixing iff

$$\sum_{U\in\mathcal{R}_n} \eta(U) g(U,x,y) = \frac{1}{4^n - 1} \tag{7}$$

for all non-zero vectors $x, y \in \{0,1\}^{2n}$. It remains to note that some constraints Eq. (7) are redundant. Indeed, since the equivalence class $[U]$ is invariant under the left/right multiplications of U by the elements of the local subgroup $\mathcal{C}_n^0$, one has $g(U,x,y) = g(U,Lx,Ry)$ for all $L, R \in \mathcal{C}_n^0$. Suppose $(x_j, x_{n+j}) \neq (0,0)$ for some qubit j. Then one can choose $L \in \mathcal{C}_n^0$ acting non-trivially only on the jth qubit such that $(Lx)_j = 0$ and $(Lx)_{n+j} = 1$, see the section "Computation of ReduceU". Applying this transformation to all qubits we conclude that the Pauli mixing constraint Eq. (7) has to be imposed only for vectors

$$x, y \in \{(0^n z) : z \in \{0,1\}^n \setminus 0^n\}. \tag{8}$$

Minimizing the average cost Eq. (6) over variables $\eta(U) \geq 0$ with $U \in \mathcal{R}_n$, subject to the normalization $\sum_{U\in\mathcal{R}_n} \eta(U) = 1$ and the Pauli mixing constraints Eqs. ((7), (8)), gives a linear program with $|\mathcal{R}_n|$ variables and $1 + (2^n - 1)^2$ equality constraints. We were able to find an optimal solution of this linear program numerically for $n = 2, 3, 4$ qubits. The optimal reduced distributions η presented in Table 3, Table 4, and Table 5 are compactly represented by a list of reduced elements $U_1, U_2, \ldots, U_m \in \mathcal{R}_n$ along with their probabilities $\eta(U_j)$. Only reduced elements that appear with non-zero probability are shown. The tables display an optimal circuit implementation of each reduced element U_j. To

Table 4. Optimal three-qubit Clifford 2-design with the average cost 3.12363.... For comparison, the full Clifford group $\mathcal{C}_3$ has the average cost 3.50937....

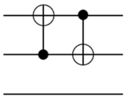
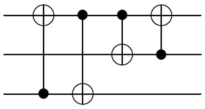
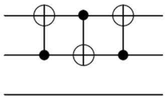
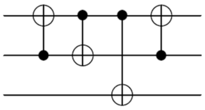
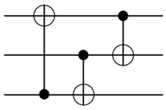
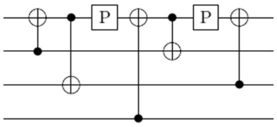
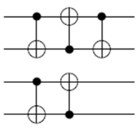
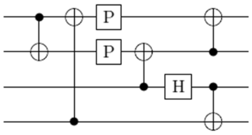
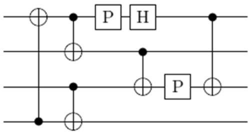
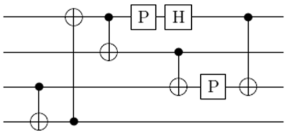
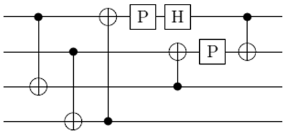
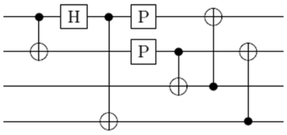
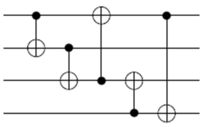
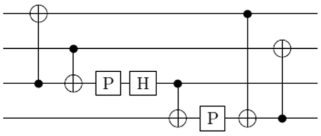
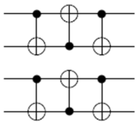
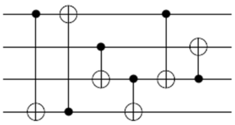
circuit U_j	probability $\eta(U_j)$	circuit U_j	probability $\eta(U_j)$
	0.074175		0.098901
	0.035715		0.098901
	0.692309		

Table 5. Optimal four-qubit Clifford 2-design with the average cost 5.08034…. For comparison, the full Clifford group $\mathcal{C}_4$ has the average cost 5.85856…. We note that all except for two circuits in this table have cost 5. The remaining pair of circuits have cost 6.

circuit U_j	$\eta(U_j)$	circuit U_j	$\eta(U_j)$
	0.141176		0.023663
	0.009893		0.013368
	0.146526		0.014572
	0.164572		0.206952
	0.198930		0.007353
	0.072994		

avoid clutter, we omit single-qubit gates on the left and on the right. The actual 2-design has the form $LW^{-1}U_jWR$, where the index $j \in \{1, 2, \ldots, m\}$ is sampled with the probability $\eta(U_j)$, the qubit permutation W is sampled uniformly from S_n, and L, R are sampled uniformly from the local subgroup C_n^0.

Comparison to prior work

Similar-spirited prior work includes the synthesis of 4-qubit optimal Clifford circuits[19], the synthesis of 4-bit optimal reversible circuits[20], and optimal solution of Rubik's cube puzzle[21,19] is most closely related to our work, given the focus on Clifford circuits; the difference is we chose to study the two-qubit gate cost, which better reflects the constraints of the existing quantum computers than the total gate count. The search space size comparison is 4.7×10^{10} in[19] to 2.1×10^{23} in our work—an almost 13 orders of magnitude difference[20] study reversible circuits, being a highly relevant type of computations. Their search space size is 2.1×10^{13}, meaning we solved a problem with 10 orders of magnitude higher search space size. Finally [21], studies Rubik's cube, which is also a finite group. Their search space size is 4.3×10^{19}, meaning ours is almost 4 orders of magnitude higher.

DISCUSSION

In this paper, we reported algorithms and their C++ implementation that compute all two-qubit gate count optimal 6-qubit Clifford circuits. There are about 2.1×10^{23} different Clifford functions. The large search space required us to employ server-class machines to make the computation possible. In particular, we used HPC to break down the set of canonical representatives of Clifford group elements sharing similar optimal circuit structure, and store them in a database of size 2.1TB. Given this database on an SSD and a 2.5GB index file in RAM, the time to extract an optimal circuit using a consumer-grade laptop is 0.0009358 s—10 times faster than the typical access time for a spindle drive. The time to extract an optimal circuit using an enterprise-level system while storing the database in RAM is 0.0006274 s—15 times faster than the typical HDD access time. We used the database to establish the maximal gate count needed to implement an arbitrary 6-qubit Clifford unitary and showed the distribution of the number of Clifford functions across their required gate counts. We established a new example of quantum advantage by Clifford circuits over CNOT gate circuits and found optimal Clifford 2-designs for the number of qubits up to, and including, 4.

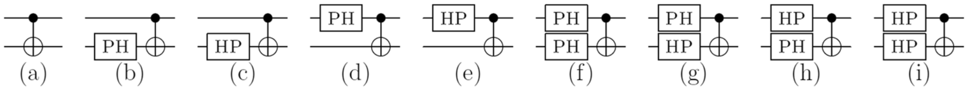


Fig. 3 CNOT gate equivalent entangling transformations that need to be applied to each of $\frac{n(n-1)}{2}$ pairs of qubits of a Clifford group element implementable with k entangling gates to explore the possibility of expanding it into a Clifford group element requiring $k+1$ gates. It suffices to apply these gates to a pair of qubits in an arbitrary fixed order, since the application of a gate in the other order is enabled by some other gate among those listed. For instance, the CNOT with flipped controls with respect to (**a**) is accomplished by (**h**), noting that the single-qubit gates on the right side do not matter due to the choice to work with equivalence classes.

METHODS

Algorithm and its implementation: an overview

Our approach relies on the use of pruned breadth-first search (BFS) to generate a number of databases containing Clifford unitaries that can be implemented by equal cost optimal circuits, and augment it by a set of tools that extract useful statistics (e.g., distribution of the number of unitaries by entangling gate cost, average cost, largest cost) as well as individual optimal circuits. BFS is a strategy that relies on taking optimal implementations of cost up to k, modifying them by applying cost-1 transformations to cost-k elements, and recording the result as a cost $k+1$ element if it is not yet found in the database. BFS is initiated with the identity operator costing zero and ends when all elements in the target set were explored. While our algorithm can be applied to obtain optimal 2-, 3-, 4-, 5-, and 6-qubit Clifford circuits using modern computers, we focus the rest of the description on the most difficult but still amenable to classical computers 6-qubit case.

Since the database we are synthesizing contains Clifford unitaries, the first order of business is to choose a suitable data structure to store those. The data structure must be both compact and allow quick application of gates; this is because BFS boils down to a series of gate applications and memory lookups. We start with the tableau, which is naturally suited for quick gate application, and modify it to remove two last rows corresponding to X and Z stabilizers each[3]. As described in the section "Data structure", these rows can be quickly restored. However, removing them allows to reduce the storage from $4n^2|_{n=6} = 144$ bits to $2\times 2n(n-1)|_{n=6} = 120$ bits. Each unitary is thus stored across two 64-bit machine words (each half corresponding to X and Z parts), with 4 bits per machine word of (yet) unused space. While information-theoretic minimum storage requirement, $\lceil \log_2(|\mathcal{C}_6|)\rceil = 78$, implies that more compact storage exists, BFS imposes the requirement of quick gate application and we furthermore rely on canonicity (discussed in next paragraph) to reduce the size of the database; thus, it is not obvious if more efficient storage is possible.

Should each Clifford element require storage, the search would not be possible to execute on modern computers since $|\mathcal{C}_6| \approx 2\times 10^{23}$. We, therefore, break Clifford group elements into classes of equivalence such that class members share the same optimal circuit structure, a canonical representative exists, and it is efficient to compute. In our approach, a class of equivalence can be thought of as containing unitaries with optimal circuits equivalent up to left- and right-hand multiplication by single-qubit Clifford unitaries, and qubit relabeling; the canonical representative is chosen to be the one with the least lexicographic order across all elements in its equivalence class. This means that we can pack up to $|\mathcal{C}_1|^{2n}\cdot |S_n||_{n=6} = 6^{12}\cdot 6! = 1,567,283,281,920$ unitaries into one class. More precisely, the number of unitaries contained in each equivalence class may vary between $|\mathcal{C}_1|^n$ and $|\mathcal{C}_1|^{2n}\cdot|S_n|$. The former case is realized for the identity operator which is invariant under all qubit relabelings and does not differentiate between left- and right-hand multiplications by single-qubit Clifford unitaries. The latter case is realized for a generic element of the Clifford group without any special symmetries. Here, $|\mathcal{C}_1|$ is the size of the single-qubit Clifford group $\mathcal{C}_1$ raised to the power $2n$ to represent one-qubit operators on each qubit in the beginning and end of the circuit, and S_n is the permutation group. However, the computation of canonical representative must be efficient, as otherwise, complexity moves from storage to computation. We utilized a Pareto-efficient definition of the equivalence class, as determined by ReduceU, the function computing the canonical representative, to be most practical. Our computationally-defined canonical representative is at most factor 14 storage inefficient, but it allows a quick computation of the canonical representative, taking on average 0.000003 s (using Intel Core i7-10700K processor). The computation of ReduceU turns out to be the runtime-level bottleneck of our implementation since other operations that are applied with a comparable frequency (such as tableau restoration and gate application) are faster. Further details about ReduceU may be found in the section "Computation of ReduceU".

The restriction to equivalence classes helps not only to dramatically reduce the storage requirement, but also to minimize the number of CNOT-equivalent transformations that we need to apply to a Clifford unitary requiring k gates to explore Clifford unitaries requiring $k+1$ entangling gates. Specifically, the number of transformations is only $9\frac{n(n-1)}{2}|_{n=6} = 135$, as illustrated in Fig. 3.

The 15-part (one part per a fixed gate count ranging from 1 to 15, with 15 turning out to be the maximum) sorted database with canonical representatives of equal cost is 2.1TB in size, and it took roughly 6 months to synthesize it on a small cluster of Intel® server-class machines. Since we made software updates as the search progressed, and improved the performance in doing so, we believe it may take about 2 months to rerun it from scratch. We store the database on an SSD ($2+$ TB RAM was expensive at the time of this writing). Given the database, an optimal circuit for a given 6-qubit Clifford unitary U may be found as follows: compute ReduceU(U), find it in part of the database containing size k unitaries, apply each of $9\frac{n(n-1)}{2}$ gates, compute the resulting canonical element and look it up in the size $k-1$ database; once found repeat for $k:=k-1$ until $k=0$. Our implementation of the above algorithm takes an average of 0.1 s to extract an optimal circuit. The bottleneck is the database search on the SSD, since the average number of times an element needs to be searched is at most $\frac{135}{2} = 67.5$, the databases for large k are large, and search needs to make multiple queries that add up quickly given SSD's limited access time. Instead, recall that $4+4=8$ bits of the original data structure are unused, and note that 8 bits suffice to store the gate information, since $\lceil\log_2(135)\rceil = 8$. We thus augment the database by loading these 8 bits with the last gate information, allowing to select the correct gate right away during the circuit restoration. This modification reduces the runtime by roughly a factor of 67.5. We further optimize the performance by storing an index with each 1024th element of the database in RAM. This allows finding an optimal circuit implementation of an arbitrary 6-qubit Clifford unitary in as little as 0.0009358 s on a MacBook Pro® (2.3 GHz Quad-Core Intel® Core i7-1068NG7 CPU, 16GB RAM) with a USB-C attached SSD (4TB VectoTech Rapid® 540MB/s 3D NAND Flash), and 0.0006274 s on a high-performance server (Quad Intel® Xeon E7-4850 v4 16-Core/2.1GHz, 6TB RAM). These performance figures were established by averaging out the time to synthesize optimal circuits for 10,000 random uniformly distributed Clifford unitaries while relying on kernel-owned memory to cache files with the use of *mmap* and using a supplementary index for the laptop version of the search.

In the following subsections we report further details of our implementation.

Database generation

Let $\mathcal{C}_n^k \subseteq \mathcal{C}_n$ be the set of all Clifford group elements with the CNOT cost k. Here $k = 0, 1, \ldots, k_{max}(n)$ for some a-priori unknown maximum cost $k_{max}(n)$. For example, $\mathcal{C}_n^0$ is the local subgroup of $\mathcal{C}_n$, i.e., one generated by the single-qubit Clifford gates. Suppose *ReduceU* : $\mathcal{C}_n \to \mathcal{C}_n$ is a function such that ReduceU(U) = ReduceU(V) if and only if U and V are equivalent up to left and right multiplications by single-qubit gates and a qubit relabeling. In other words, ReduceU(U) is a canonical representative of the equivalence class

$$[U] := \{KW^{-1}UWL \ : \ K, L \in \mathcal{C}_n^0, \ W \in S_n\}. \qquad (9)$$

Here and below $S_n \subseteq \mathcal{C}_n$ is the subgroup of qubit permutations. A specific implementation of the function ReduceU, which we refer to the section "Computation of ReduceU", does not matter at this point. Let $\mathcal{R}_n^k$ be the set of all reduced cost-k Clifford group elements,

$$\mathcal{R}_n^k := \{ReduceU(U) \ : \ U \in \mathcal{C}_n^k\}.$$

Our database consists of $k_{max}(n) + 1$ parts, such that the k-th part contains all elements of $\mathcal{R}_n^k$. The elements are furthermore stored in the lexicographic order to enable binary search.

Let $I \in \mathcal{C}_n$ be the identity matrix and $\mathrm{CNOT}_{i,j}$ be the CNOT gate with the control qubit i and the target qubit j. Since any cost-0 and cost-1 element is equivalent to I and $\mathrm{CNOT}_{1,2}$ respectively, we have

$$\mathcal{R}_n^0 = \{ReduceU(I)\} \text{ and } \mathcal{R}_n^1 = \{ReduceU(\mathrm{CNOT}_{1,2})\}.$$

Suppose we have the sets $\mathcal{R}_n^0, \mathcal{R}_n^1, \ldots, \mathcal{R}_n^{k-1}$ for some $k \geq 2$ (initially $k = 2$). The rest of this section explains how to compute $\mathcal{R}_n^k$. First, we need to choose a set of cost-1 generators that obey certain technical conditions. Let $m = 9n(n-1)/2$ and $G_1, G_2, \ldots, G_m \in \mathcal{C}_n^1$ be the generators shown in Fig. 3. By definition, each generator has the form $\mathrm{A}_i\mathrm{B}_j\mathrm{CNOT}_{i,j}$ for some pair of qubits $i < j$ and $\mathrm{A}, \mathrm{B} \in \{\mathrm{I}, \mathrm{PH}, \mathrm{HP}\}$. We will use the following properties of the generator set.

Lemma 1. Any cost-k element $U \in \mathcal{C}_n^k$ can be written as $U = G_{a_1}G_{a_2}\cdots G_{a_k}L$ for some $L \in \mathcal{C}_n^0$ and some $a_1, a_2, \ldots, a_k \in \{1, 2, \ldots, m\}$.

The proof is deferred to the section "Proof of Lemma 1". This lemma has the following simple corollaries.

Corollary 1. Suppose $W \in S_n$ is a qubit permutation and $L \in \mathcal{C}_n^0$. For any generator G_a there exist a generator G_b and $M \in \mathcal{C}_n^0$ such that $WLG_a = G_bWM$.

Proof. Let $U = WLG_aW^{-1}$. Note that $U \in \mathcal{C}_n^1$ since U is equivalent to a cost-1 element G_a. Lemma 1 with $k = 1$ implies that $U = G_bM'$ for some generator G_b and some $M' \in \mathcal{C}_n^0$. Thus $WLG_a = G_bM'W = G_bWM$, where $M = W^{-1}M'W \in \mathcal{C}_n^0$.

Corollary 2. For any generator G_a and $L \in \mathcal{C}_n^0$ there exists a generator G_b such that $G_aLG_b \in \mathcal{C}_n^0$.

Proof. Let $U = (G_aL)^{-1}$. Note that $U \in \mathcal{C}_n^1$ since the cost is invariant under taking the inverse. Lemma 1 with $k = 1$ implies that $U = G_bM$ for some generator G_b and $M \in \mathcal{C}_n^0$. Thus $G_aLG_b = M^{-1} \in \mathcal{C}_n^0$.

We claim that the following algorithm outputs the set $S = \mathcal{R}_n^k$.

Algorithm 1.

```
 $S \leftarrow \emptyset$
 for $V \in \mathcal{R}_n^{k-1}$ do
     for $b \in \{1, 2, \ldots, m\}$ do
         $U \leftarrow$ ReduceU$(VG_b)$
         if $U \notin \mathcal{R}_n^{k-2} \cup \mathcal{R}_n^{k-1}$ then
           $S \leftarrow S \cup \{U\}$.
     end if
  end for
 end for
```

Let us first check that $\mathcal{R}_n^k \subseteq S$. Consider any element $U \in \mathcal{R}_n^k$. Then $U = ReduceU(\tilde{U})$ for some $\tilde{U} \in \mathcal{C}_n^k$. By Lemma 1, we can write $\tilde{U} = G_{a_1}G_{a_2}\cdots G_{a_k}M$ for some $M \in \mathcal{C}_n^0$. Define

$$\tilde{V} := G_{a_1}G_{a_2}\cdots G_{a_{k-1}} \text{ and } V := ReduceU(\tilde{V}).$$

Note that $\tilde{V} \in \mathcal{C}_n^{k-1}$ (if $\tilde{V} \in \mathcal{C}_n^\ell$ for some $\ell < k-1$ then $\tilde{U} = \tilde{V}G_{a_k}M$ would have cost less than k). Accordingly, $V \in \mathcal{R}_n^{k-1}$. By definition of the function ReduceU, we have $\tilde{V} = KW^{-1}VWL$ for some $K, L \in \mathcal{C}_n^0$ and some qubit relabeling $W \in S_n$. Thus

$$\tilde{U} = G_{a_1}G_{a_2}\cdots G_{a_k}M = \tilde{V}G_{a_k}M = KW^{-1}VWLG_{a_k}M.$$

Commuting G_{a_k} through WL next to V using Corollary 1 we obtain $\tilde{U} = KW^{-1}(VG_b)WM'$ for some generator G_b and some $M' \in \mathcal{C}_n^0$. This shows that $\tilde{U}$ is equivalent to VG_n and thus $Reduce(VG_b) = Reduce(\tilde{U}) = U$ for some $V \in \mathcal{R}_n^{k-1}$ and some generator G_b. Thus $U \in S$. We have proved that $\mathcal{R}_n^k \subseteq S$.

Table 6. The distribution of the number of equivalence classes across Clifford circuits over 2, 3, 4, 5, and 6 qubits.

CNOT count Qubits	2	3	4	5	6
0	1	1	1	1	1
1	1	1	1	1	1
2	1	3	4	4	4
3	1	8	20	22	23
4		10	112	183	198
5		3	525	1958	2549
6		1	1230	22,257	42,883
7			453	223,723	824,723
8			16	1,441,124	16,086,167
9			1	2,471,855	294,266,642
10				161,458	4,399,997,085
11				72	40,791,942,327
12				1	92,804,759,960
13					5,666,221,415
14					8,281
15					3
Total	4	27	2363	4,322,659	143,974,152,262

Conversely, suppose $U \in S$. Then U is a reduced element obtained from some cost-$(k-1)$ element V by adding a single generator, relabeling the qubits, and left/right multiplications by the single-qubit gates. Since adding a single generator can change the cost by at most one, we conclude that $U \in \mathcal{R}_n^{k-2} \cup \mathcal{R}_n^{k-1} \cup \mathcal{R}_n^k$. The cost cannot grow by more than 1 for an obvious reason. It cannot decline by $d > 1$ since this would imply that V can be implemented with cost $(k - 1 - d) + 1 = k - d < k - 1$ as the circuit $(Vg).g^{-1}$, where g is the generator, which contradicts the notion that V is a cost-$(k-1)$ element. Thus the algorithm adds U to S only if $U \in \mathcal{R}_n^k$. We have proved that $S \subseteq \mathcal{R}_n^k$.

By sorting the elements of each set $\mathcal{R}_n^\ell$ and using the binary search to check set membership, the above algorithm requires $\tilde{O}(|\mathcal{R}_n^{k-1}|m)$ calls to the function ReduceU, where the $\tilde{O}$ notation hides factors logarithmic in the size of $\mathcal{R}_n^{k-2}$, $\mathcal{R}_n^{k-1}$, and $\mathcal{R}_n^k$. The database generation terminates as soon as $\mathcal{R}_n^k = \emptyset$. This determines the maximum cost $k_{max}(n)$ as $k - 1$.

As discussed in the section "Methods", the generation of the 6-qubit database spans a few CPU months and involves manipulations with terabytes of data. How can we be confident that this computation is error-free? Our correctness tests included the verification that the size of the Clifford group inferred from the database agrees with the analytic formula $|\mathcal{C}_n| = 2^{n^2}\prod_{j=1}^n(4^j - 1)$. In more detail, the number of cost-k Clifford group elements can be inferred from the identity

$$|\mathcal{C}_n^k| = \sum_{U \in \mathcal{R}_n^k} |[U]|, \qquad (10)$$

where $|[U]|$ is the size of the equivalence class $[U]$ that contains U, see Eq. (9). Furthermore,

$$|[U]| = \frac{|\mathcal{C}_n^0|^2 \cdot |S_n|}{|\mathrm{Aut}(U)|} = \frac{6^{2n}n!}{|\mathrm{Aut}(U)|}, \qquad (11)$$

where $\mathrm{Aut}(U)$ is the automorphism group of U that consists of all triples $K \times L \times W \in \mathcal{C}_n^0 \times \mathcal{C}_n^0 \times S_n$ such that $U = KW^{-1}UWL$. We have checked that the counts $|\mathcal{C}_n^k|$ inferred from Eqs. ((10), (11)) indeed obey $\sum_{k=0}^{k_{max}(n)} |\mathcal{C}_n^k| = |\mathcal{C}_n|$. Thus our database passed the self-consistency test. Table 6 and Table 1 display the counts $|\mathcal{R}_n^k|$ and $|\mathcal{C}_n^k|$ can be found in the section "Results".

In order to speed up the synthesis of optimal circuits, we augmented each database entry $U \in \mathcal{R}_n^k$ with 8 auxiliary bits specifying a generator G_b that reduces the cost of U by one, such that $UG_b \in \mathcal{C}_n^{k-1}$. Here we assume $k \geq 1$. Let us prove that such cost-reducing generator G_b exists for any

$U \in \mathcal{R}_n^k$. Indeed, use Lemma 1 to write $U = G_{a_1}G_{a_2}\cdots G_{a_k}L$ for some $L \in \mathcal{C}_n^0$. By Corollary 2, there exists a generator G_b such that $F \equiv G_{a_k}LG_b \in \mathcal{C}_n^0$. Now $UG_b = G_{a_1}G_{a_2}\cdots G_{a_{k-1}}F$ for some $F \in \mathcal{C}_n^0$, that is, UG_b has cost $k-1$.

To augment a given element U of the cost-k database $\mathcal{R}_n^k$ we find the first cost-reducing generator $b \in \{1, 2, \ldots, m\}$ such that $ReduceU(UG_b) \in \mathcal{R}_n^{k-1}$. This requires at most m calls to ReduceU and binary searches in $\mathcal{R}_n^{k-1}$ (computing the group multiplication takes a negligible time). Once a cost-reducing generator G_b is found, its index b is recorded in the database using the unused bits of U. The augmentation step is applied to all $U \in \mathcal{R}_n^k$ and for all $k = 1, 2, \ldots, k_{\max}(n)$.

Synthesis of optimal circuits

The optimal compiler takes as input an element of the Clifford group $U \in \mathcal{C}_n$ and outputs a Clifford circuit (a list of the primitive gates H, P, and CNOT) implementing U with the smallest possible CNOT gate count, equal to the cost of U. The cost can be computed by making a single call to ReduceU and performing at most $k_{\max}(n)$ database searches. Below we assume that the database is augmented with the cost-reducing generators, as discussed in the section "Database generation". Thus the database search returns the cost k element V such that $V \equiv Reduce(U) \in \mathcal{R}_n^k$ and a cost-reducing generator G_a such that $VG_a \in \mathcal{C}_n^{k-1}$. The next step is to convert G_a into a cost-reducing generator for U. To this end, write $V = KW^{-1}UWL$ for some $K, L \in \mathcal{C}_n^0$ and some qubit permutation W. The group elements K, L, and W that transform U into the reduced form are readily available by adding appropriate bookkeeping steps to the implementation of ReduceU described in the section "Computation of ReduceU". At this point we have

$$KW^{-1}UWLG_a \in \mathcal{C}_n^{k-1}.$$

Commute G_a through WL next to U using Corollary 1. This gives

$$KW^{-1}UG_bWM \in \mathcal{C}_n^{k-1}$$

for some generator G_b and some $M \in \mathcal{C}_n^0$. The generator G_b can be computed in time $O(1)$ using the standard commutation rules of the Clifford group. Thus $UG_b \in \mathcal{C}_n^{k-1}$, that is, G_b is a cost-reducing generator for U. Replacing U by UG_b and applying the above step recursively, one constructs a k-tuple of generators such that $M = UG_{a_1}G_{a_2}\cdots G_{a_k} \in \mathcal{C}_n^0$ is a product of single-qubit gates. This gives $U^{-1} = G_{a_1}G_{a_2}\cdots G_{a_k}M^{-1}$. Decomposing each generator and M^{-1} into a product of primitive gates H, P, and CNOT gives an optimal circuit implementing U^{-1}. Since all primitive gates are self-inverse, an optimal circuit implementing U is obtained simply by reversing the order of gates. If needed, the number of single-qubit gates in the compiled circuit can be optimized by commuting single-qubit gates to the last time step (whenever possible) and merging them using optimal lookup of $\mathcal{C}_1$ elements.

Computation of ReduceU

In this section we introduce reduced forms of Clifford group elements and give algorithms for computing these forms. A given matrix $U \in \mathcal{C}_n$ is transformed into a reduced form by applying a sequence of elementary reductions from the following list:

1. Multiplication of U on the left by single-qubit Clifford gates.
2. Multiplication of U on the right by single-qubit Clifford gates.
3. Relabeling of qubits.

Depending on which type of reductions is considered, there are three different reduced forms: a left-reduced form (reductions of type 1 only), a locally reduced form (reductions of types 1 and 2), and a fully reduced form (reductions of types 1, 2, and 3). Each form comes with an algorithm specifying the sequence of reductions to be applied. We define the reduced forms inductively starting from the left-reduced form. The function ReduceU used in the sections "Database generation" and "Synthesis of optimal circuits" computes the fully reduced form.

We begin by defining convenient notations. Let $e^1, e^2, \ldots, e^{2n} \in \mathbb{F}_2^{2n}$ be the standard basis of $\mathbb{F}_2^{2n}$: the basis vector e^j has a single non-zero at the jth position. We consider e^j as column vectors. Let $e_j := (e^j)^T$ be the corresponding row vector. For example, if $n = 1$ then

$$e^1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad e^2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad e_1 = [1 \quad 0], \text{ and } e_2 = [0 \quad 1].$$

We write $u \oplus v$ to denote the addition of binary vectors u and v modulo 2. Elements of the Clifford group $U \in \mathcal{C}_n$ are treated as binary symplectic matrices of the size $2n \times 2n$. A matrix U has the jth column and the jth row Ue^j and e_jU, respectively.

Recall that $\mathcal{C}_n^0 \subseteq \mathcal{C}_n$ is the local subgroup generated by the single-qubit gates (H and P). Define a subgroup $\mathcal{C}_{n,j} \subseteq \mathcal{C}_n^0$ generated by the single-qubit gates acting on the jth qubit, where $j = 1, 2, \ldots, n$. Equivalently, $U \in \mathcal{C}_{n,j}$ iff $Ue^i = e^i$ for all $i \notin \{j, n+j\}$, whereas $Ue^j = ae^j \oplus be^{n+j}$ and $Ue^{n+j} = ce^j \oplus de^{n+j}$ for some coefficients $a, b, c, d \in \mathbb{F}_2$ such that

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} \in \mathrm{GL}(2, \mathbb{F}_2).$$

Note that the subgroups $\mathcal{C}_{n,j}$ pairwise commute.

A matrix $U \in \mathcal{C}_n$ is said to be *left-reduced* if

$$e_jU < e_{n+j}U < (e_j \oplus e_{n+j})U \text{ for all } j = 1, 2, \ldots, n. \tag{12}$$

Here and below the bit strings are compared using the lexicographic order (i.e., $00 < 01 < 10 < 11$ in the case $n = 1$). The following lemma shows that left-reduced elements of $\mathcal{C}_n$ can serve as canonical representatives of cosets $\mathcal{C}_n^0U$. In other words, $\mathcal{C}_n$ is a disjoint union of cosets $\mathcal{C}_n^0U$ and each coset contains a unique left-reduced element, which can be efficiently computed. We refer to the unique left-reduced element of a coset $\mathcal{C}_n^0U$ as the *left-reduced form* of U and denote it leftReduce(U). Our symplectic matrix data structure described in the section "Data structure" enables the computation of leftReduce(U) for a randomly picked matrix $U \in \mathcal{C}_n$ in time less than 2×10^{-8} s for any $n \leq 6$ on a server-class CPU, in this case an Intel® Xeon® CPU E7-4850 v4 @ 2.10GHz.

Lemma 2. *Each coset $\mathcal{C}_n^0U$ with $U \in \mathcal{C}_n$ contains a unique left-reduced element that can be computed in time $O(n^2)$, given symplectic matrix representation of U.*

Proof. First note that the rows of a symplectic matrix are linearly independent. Thus for each qubit j the bit strings $x_j := e_jU$, $z_j := e_{n+j}U$, and $y_j := (e_j \oplus e_{n+j})U$ are all distinct: $x_j \neq y_j \neq z_j$. It follows directly from the above definitions that multiplying U on the left by the elements of the subgroup $\mathcal{C}_{n,j}$ we can implement any permutation of the bit strings x_j, y_j, and z_j. For example, the Hadamard gate swaps x_j and z_j, the Phase gate swaps x_j and y_j. Since $|\mathcal{C}_{n,j}| = 6$, there is a one-to-one correspondence between elements of $\mathcal{C}_{n,j}$ and permutations of x_j, y_j, z_j. Multiply U on the left by the unique element of $\mathcal{C}_{n,j}$ that permutes the bit strings such that $x_j < z_j < y_j$. Now Eq. (12) is satisfied for the jth qubit. Repeating this for all n qubits and noting that $\mathcal{C}_n^0$ is generated by the subgroups $\mathcal{C}_{n,j}$ proves that the coset $\mathcal{C}_n^0U$ contains a unique left-reduced element. All above steps can be efficiently implemented. Indeed, given a matrix U, one can compute the bit strings x_j, y_j, and z_j and sort all three in time $O(n)$. Repeating this for all n qubits gives the total runtime of $O(n^2)$.

Given a matrix $U \in \mathcal{C}_n$ define a double coset

$$[U]^{loc} := \mathcal{C}_n^0U\mathcal{C}_n^0.$$

It includes all elements of the Clifford group obtained from U by adding single-qubit Clifford gates on the left and on the right. Clearly, the full Clifford group $\mathcal{C}_n$ is a disjoint union of double cosets $[U]^{loc}$ and the cost of the matrix U depends only on the double coset that contains U. The next step is to choose an efficiently computable canonical representative of each double coset. First define the map $\chi : \mathbb{F}_2^{2n} \to \mathbb{F}_2^n$ as

$$\chi(v) := [v_1 \vee v_{n+1}, v_2 \vee v_{n+2}, \ldots, v_n \vee v_{2n}],$$

where $\vee$ stands for the logical OR operation. The jth component of $\chi(v)$ is non-zero iff $v_j = 1$ or $v_{n+j} = 1$ (the bit string $\chi(v)$ can be interpreted as the support of an n-qubit Pauli operator parameterized by v, according to the standard binary parameterization of Pauli operators[3]). We claim that the map χ is invariant under left multiplications by the elements of the local subgroup, in the sense that

$$\chi(Lv) = \chi(v) \text{ for all } L \in \mathcal{C}_n^0 \text{ and } v \in \mathbb{F}_2^{2n}. \tag{13}$$

Indeed, it suffices to check Eq. (13) for the special case $L \in \mathcal{C}_{n,j}$ (since the local subgroup is generated by matrices $L \in \mathcal{C}_{n,j}$ with $j = 1, 2, \ldots, n$). As discussed above, the action of $L \in \mathcal{C}_{n,j}$ on v is equivalent to applying a

2×2 binary invertible matrix to the components v_j and v_{n+j} while all other components of v remain unchanged. Since an invertible matrix maps non-zero vectors to non-zero vectors, $(Lv)_j \vee (Lv)_{n+j} = 1$ iff $v_j \vee v_{n+j} = 1$. This implies Eq. (13).

A matrix $U \in \mathcal{C}_n$ is said to be *locally ordered* if U is left-reduced and

$$\chi(Ue^j) \le \chi(Ue^{n+j}) \le \chi(Ue^j \oplus Ue^{n+j}) \text{ for all } j = 1, 2, \ldots, n. \tag{14}$$

Here bit strings are compared using the lexicographic order. Let $\mathcal{L}(U) \subseteq [U]^{loc}$ be the set of all locally ordered elements of the double coset $[U]^{\mathrm{loc}}$. Define a *locally reduced* form of the matrix $U \in \mathcal{C}_n$, denoted localReduce(U), as the lexicographically smallest element of the set $\mathcal{L}(U)$. The following lemma shows that locally reduced elements of $\mathcal{C}_n$ can serve as canonical representatives of the double cosets $[U]^{\mathrm{loc}}$. In other words, $\mathcal{C}_n$ is a disjoint union of the double cosets $[U]^{\mathrm{loc}}$ and each double coset contains a unique locally reduced element that can be efficiently computed (albeit slightly less efficiently than leftReduce). The symplectic matrix data structure described in the section "Data structure" enables the computation of localReduce(U) for a randomly picked matrix $U \in \mathcal{C}_n$ in time less than 4×10^{-7} s for all $n \le 6$ on a server-class CPU, in this case an Intel® Xeon® CPU E7-4850 v4 @ 2.10GHz.

Lemma 3. Each double coset $[U]^{loc} = \mathcal{C}_n^0 U \mathcal{C}_n^0$ contains a unique locally reduced element that can be computed in time $O(n^2 6^n)$, given the symplectic matrix U.

Proof. For each qubit j define the bit strings $x_j := \chi(Ue^j)$, $z_j := \chi(Ue^{n+j})$, and $y_j := \chi(Ue^j \oplus Ue^{n+j})$. Same as before, multiplying U on the right by the elements of the subgroup $\mathcal{C}_{n,j}$ one can implement any permutation of the bit strings x_j, y_j, and z_j. Define a subset $\mathcal{S}_j \subseteq \mathcal{C}_{n,j}$ as the one including all elements $R_j \in \mathcal{C}_{n,j}$ such that the right multiplication $U \leftarrow UR_j$ permutes the bit strings x_j, y_j, and z_j into the non-decreasing order $x_j \le z_j \le y_j$. Note that $\mathcal{S}_j$ is non-empty since the right multiplication by the elements of $\mathcal{C}_{n,j}$ can implement any permutation of x_j, y_j, and z_j. Recall that the set $\mathcal{L}(U)$ includes all locally ordered elements of the double coset $[U]^{\mathrm{loc}}$. We claim that

$$\mathcal{L}(U) = \{leftReduce(UR_1R_2\cdots R_n) \; : \; R_1 \in \mathcal{S}_1, R_2 \in \mathcal{S}_2, \ldots, R_n \in \mathcal{S}_n\}. \tag{15}$$

Indeed, $\mathcal{L}(U) \subseteq [U]^{loc}$ since any matrix $W \in \mathcal{L}(U)$ has the form $W = LUR$ for some $L, R \in \mathcal{C}_n^0$. Furthermore, $\mathcal{L}(U)$ is non-empty since each subset $\mathcal{S}_j$ is non-empty. Let us check that any element $W \in \mathcal{L}(U)$ is locally ordered. Indeed, pick any matrices $R_j \in \mathcal{S}_j$ and let $R = R_1R_2\cdots R_n$. By construction, the matrix $V = UR$ satisfies Eq. (14) with U replaced by V. Let $W =$ leftReduce(V). Then $W = LV$ for some $L \in \mathcal{C}_n^0$. The invariance of the map χ under left multiplications by the elements of the local subgroup, see Eq. (13), implies that W satisfies Eq. (14) with U replaced by W. Thus W is locally ordered. Conversely, suppose $W \in [U]^{\mathrm{loc}}$ is locally ordered. Then $W = LUR$ for some $L, R \in \mathcal{C}_n^0$ and leftReduce(W) $= W$. The invariance of the map χ under left multiplications by the elements of the local subgroup and the local ordering condition imply that the matrix $V = UR$ satisfies Eq. (14) with U replaced by V. Thus $R = R_1R_2\cdots R_n$ for some $R_j \in \mathcal{S}_j$. This proves that $W \in \mathcal{L}(U)$. The uniqueness follows from the ability to encode the elements of the sets considered by distinct integers and the existence of the smallest integer in any finite set of integers.

It remains to check that the set $\mathcal{L}(U)$ can be computed in time $O(n^2 6^n)$. Indeed, for any given qubit j one can compute the bit strings x_j, y_j, and z_j and the subset $S_j \subseteq \mathcal{C}_{n,j}$ in time $O(n)$. Note that $|\mathcal{S}_j| \le |\mathcal{C}_{n,j}| = 6$. Thus the number of matrices $R = R_1R_2\cdots R_n$ with $R_j \in \mathcal{S}_j$ is at most 6^n. Since the right multiplication by the elements of the subgroup $\mathcal{C}_{n,j}$ changes at most two rows of a matrix, we can compute UR in time $O(n^2)$. By Lemma 2, computing the left reduced form of UR takes time $O(n^2)$. Thus the overall runtime of computing $\mathcal{L}(U)$ is $O(n^2 6^n)$. Once the set $\mathcal{L}(U)$ is computed, finding its lexicographically smallest element takes time $O(n|\mathcal{L}(U)|) = O(n6^n)$.

Comment 1: Our implementation of localReduce(U) relies on a streamlined version of the above algorithm with a modified definition of the subsets $\mathcal{S}_j$. Namely, we define $\mathcal{S}_j$ as a set of all elements $R_j \in \mathcal{C}_{n,j}$ such that the right multiplication $U \leftarrow UR_j$ permutes the bit strings x_j, y_j, and z_j into the non-decreasing order and leftReduce(UR_j) ≠ leftReduce(U). The last condition rules out the possibility that the right multiplication of U by R_j is equivalent to a left multiplication of U by some element of the local subgroup (for example, this is the case if U is the identity matrix). Since leftReduce(U) depends only on the coset $\mathcal{C}_n^0 U$, the left multiplication of U by any element of the local subgroup does not change leftReduce(U). Thus the set of locally ordered elements $\mathcal{L}(U)$ can be computed using Eq. (15) with the modified definition of $\mathcal{S}_j$.

Comment 2: We empirically observed that the average-case runtime of the above algorithm is much better than the worst case upper bound of $O(n^2 6^n)$. Indeed, a direct inspection shows that the runtime scales as $O(n^2 M)$, where $M = |\mathcal{S}_1| \cdot |\mathcal{S}_2| \cdot \ldots \cdot |\mathcal{S}_n|$. For randomly picked matrices $U \in \mathcal{C}_6$ we observed that $M \approx 5$ on average even though $M = |\mathcal{C}_6^0| = 6^6 = 46,656$ in the worst case. We leave it as an open question whether the average-case runtime of the above algorithm scales polynomially with n.

Recall that we consider the symmetric group S_n that includes all qubit permutations as a subgroup of $\mathcal{C}_n$. If w is a permutation of integers $\{1, 2, \ldots, n\}$, then the corresponding symplectic matrix $W \in S_n$ acts on the basis vectors as $We^j = e^{w(j)}$ and $We^{n+j} = e^{n+w(j)}$ for all $j = 1, 2, \ldots, n$. Given a matrix $U \in \mathcal{C}_n$, define the equivalence class

$$[U] := \{LW^{-1}UWR \; : \; L, R \in \mathcal{C}_n^0, W \in S_n\}.$$

The rest of this section is devoted to choosing an efficiently computable canonical representative of each class $[U]$. Let $\mathbb{Z}^{n\times n}$ be the set of $n\times n$ matrices with integer entries. Define the map $\kappa : \mathcal{C}_n \to \mathbb{Z}^{n\times n}$ such that the matrix element of $\kappa(U)$ located at the ith row and the jth column is the rank of the 2×2 submatrix of U formed by the intersection of rows i and $i+n$ and columns j and $j+n$. The rank is computed over the binary field $\mathbb{F}_2$. In other words, each matrix element of $\kappa(U)$ has the form

$$\kappa(U)_{i,j} = \mathrm{rank}_{\mathbb{F}_2}\begin{bmatrix} U_{i,j} & U_{i,n+j} \\ U_{n+i,j} & U_{n+i,n+j} \end{bmatrix}.$$

By definition, $\kappa(U)$ contains entries from the set $\{0, 1, 2\}$ and the full matrix $\kappa(U)$ can be computed in time $O(n^2)$. We claim that the left and right multiplications of U by the single-qubit Clifford gates leave $\kappa(U)$ invariant, that is,

$$\kappa(LUR) = \kappa(U) \text{ for all } L, R \in \mathcal{C}_n^0. \tag{16}$$

Indeed, suppose first that $L = I$ and $R \in \mathcal{C}_{n,j}$. Right multiplication $U \leftarrow UR$ applies an invertible linear transformation to the pair of columns Ue^j and Ue^{n+j}, and acts trivially on the remaining columns. Since the matrix rank is invariant under applying an invertible linear transformation, we conclude that $\kappa(UR) = \kappa(U)$ for all $R \in \mathcal{C}_{n,j}$. Same argument shows that $\kappa(LU) = \kappa(U)$ for all $L \in \mathcal{C}_{n,j}$. This proves Eq. (16) since the local subgroup $\mathcal{C}_n^0$ is generated by the subgroups $\mathcal{C}_{n,j}$.

Let $\kappa_{min}(U)$ be the lexicographically smallest matrix in the set of matrices $\{\kappa(W^{-1}UW) : W \in S_n\}$. Define a set of qubit permutations

$$\mathcal{S}(U) := \{W \in S_n \; : \; \kappa(W^{-1}UW) = \kappa_{min}(U)\}$$

and a set of matrices

$$\mathcal{R}(U) := \{localReduce(W^{-1}UW) \; : \; W \in \mathcal{S}(U)\}.$$

Note that $\mathcal{R}(U) \subseteq [U]$ since

$$localReduce(W^{-1}UW) = LW^{-1}UWR \in [U]$$

for some $L, R \in \mathcal{C}_n^0$. Define a *fully reduced* form of a matrix $U \in \mathcal{C}_n$, denoted ReduceU(U), as the lexicographically smallest element of the set $\mathcal{R}(U)$. The following lemma shows that the fully reduced elements of $\mathcal{C}_n$ can serve as canonical representatives of the equivalence classes $[U]$. In other words, $\mathcal{C}_n$ is a disjoint union of the equivalence classes $[U]$ and each class contains a unique fully reduced element that can be efficiently computed (albeit slightly less efficiently than localReduce). The symplectic matrix data structure enables the computation of ReduceU(U) for a randomly picked matrix $U \in \mathcal{C}_n$ in time less than 3×10^{-6} s for $n = 6$ and time less than 10^{-6} s for all $n \le 5$ on a server-class CPU, in this case an Intel® Xeon® CPU E7-4850 v4 @ 2.10GHz.

Lemma 4. Each equivalence class $[U]$ with $U \in \mathcal{C}_n$ contains a unique fully reduced element that can be computed in time $O(n^2 \cdot n! + t_n \cdot |\mathcal{S}(U)|)$, given the symplectic matrix representation of U. Here t_n is the runtime of localReduce for elements of $\mathcal{C}_n$.

Proof. Consider a matrix $U \in \mathcal{C}_n$. It follows directly from the definitions that ReduceU(U) $\in [U]$. Thus it suffices to check that

$$\mathcal{R}(U') = \mathcal{R}(U) \text{ for all } U' \in [U]. \tag{17}$$

Indeed, this equation implies *ReduceU*(U) = *ReduceU*(U') for all $U' \in [U]$, that is, the equivalence class $[U]$ contains a unique reduced element. Let us prove Eq. (17). Write $U' = LW^{-1}UWR$ for some $L, R \in \mathcal{C}_n^0$ and $W \in S_n$. Then

$$\begin{aligned}\mathcal{R}(U') &= \{localReduce(\tilde{W}^{-1}LW^{-1}UWR\tilde{W}) : \tilde{W} \in \mathcal{S}(U')\}\\ &= \{localReduce(L'\tilde{W}^{-1}W^{-1}UW\tilde{W}R') : \tilde{W} \in \mathcal{S}(U')\}\\ &= \{localReduce(\tilde{W}^{-1}W^{-1}UW\tilde{W}) : \tilde{W} \in \mathcal{S}(U')\}.\end{aligned} \tag{18}$$

Here $L' := \tilde{W}^{-1}L\tilde{W} \in \mathcal{C}_n^0$ and $R' := \tilde{W}^{-1}R\tilde{W} \in \mathcal{C}_n^0$. In the third equality we noted that localReduce is invariant under left/right multiplications by the elements of the local subgroup $\mathcal{C}_n^0$, see Lemma 3. Finally, the invariance of the map κ under the left and right multiplications by the elements of the local subgroup, see Eq. (16), implies $\kappa_{\min}(U') = \kappa_{\min}(U)$. Thus $\tilde{W} \in \mathcal{S}(U')$ iff $W\tilde{W} \in \mathcal{S}(U)$. Combining this and Eq. (18) gives $\mathcal{R}(U') = \mathcal{R}(U)$, as claimed.

The runtime stated in the lemma consists of two terms. The term $O(n^2 \cdot n!)$ is the time needed to compute the set of permutations $\mathcal{S}(U)$. The term $O(t_n \cdot |\mathcal{S}(U)|)$ is the time needed to compute the set of matrices $\mathcal{R}(U)$ and pick the lexicographically smallest element of $\mathcal{R}(U)$.

Comment 3: Our implementation of ReduceU(U) relies on a streamlined version of the above algorithm with a modified definition of the set $\mathcal{S}(U)$. Namely, we define $\mathcal{S}(U)$ as the set of all permutations $W \in S_n$ such that $\kappa(W^{-1}UW) = \kappa_{\min}(U)$ and leftReduce($W^{-1}UW$) $\neq$ leftReduce(U). The last condition rules out the possibility that the conjugation of U by W is equivalent to a left multiplication of U by some element of the local subgroup (for example, this is the case if U is the identity matrix). Since localReduce(U) depends only on the double coset $\mathcal{C}_n^0 U \mathcal{C}_n^0$, a left multiplication of U by any element of the local subgroup does not change localReduce(U). Thus one can compute the set $\mathcal{R}(U)$ using the modified definition of $\mathcal{S}(U)$.

Comment 4: We empirically observed that $|\mathcal{S}(U)| = 1$ for typical a element of the Clifford group and the maximal value of $|\mathcal{S}(U)|$ is 14. The mean value of $|\mathcal{S}(U)|$ is approximately 1.03 for a randomly picked $U \in \mathcal{C}_6$.

By a slight abuse of terminology, we refer to the computationally-defined fully reduced elements of the Clifford group as the reduced elements in the remainder of the paper. This should not lead to confusion since the left-reduced and the locally reduced forms are used only in this subsection.

Data structure

By definition, any element of the Clifford group $U \in \mathcal{C}_n$ can be represented by a binary matrix of size $2n \times 2n$. However, if we only care about the reduced form of U, a slightly more efficient representation is possible, as given by the following lemma.

Lemma 5. Let U' be the matrix obtained from $U \in \mathcal{C}_n$ by removing the n-th and the $2n$-th rows from it. Then U is uniquely determined by U' up to left multiplication by the single-qubit Clifford gates acting on the n-th qubit.

Proof. Let $\mathcal{L} \subseteq \mathbb{F}_2^{2n}$ be the linear subspace spanned by the jth row of U with $j \notin \{n, 2n\}$ and let $\mathcal{L}^\perp \subseteq \mathbb{F}_2^{2n}$ be the linear subspace spanned by the vectors orthogonal to $\mathcal{L}$ with respect to the symplectic inner product. Note that $\mathcal{L}$ depends only on U'. The condition that U is a symplectic matrix implies $\text{span}_{\mathbb{F}_2}(e_nU, e_{2n}U) = \mathcal{L}^\perp$. Here we use the notations from the section "Computation of ReduceU". The missing pair of rows e_nU and $e_{2n}U$ is uniquely defined by $\mathcal{L}$ up to an invertible linear transformation $e_nU \leftarrow ae_nU \oplus be_{2n}U$ and $e_{2n}U \leftarrow ce_nU \oplus de_{2n}U$ for some

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} \in \text{GL}(2, \mathbb{F}_2).$$

As discussed in the section "Computation of ReduceU", there is a one-to-one correspondence between such transformations and left multiplications $U \leftarrow LU$, where $L \in \mathcal{C}_n^0$ acts non-trivially only on the nth qubit.

We refer to the matrix U' obtained from $U \in \mathcal{C}_n$ by removing the pair of rows n and $2n$ as a *thin matrix* representation of U. Our C++ implementation adopts the thin matrix data format for all intermediate steps of the algorithm. The thin matrix spans $4n(n-1)$ bits and can be conveniently distributed over two machine words, each of length 64 bits. The first word stores the rows $e_1U, e_2U, \ldots, e_{n-1}U$ and the second word stores the rows $e_{n+1}U, e_{n+2}U, \ldots, e_{2n-1}U$. This leaves $128 - 4n(n-1)|_{n\le 6} \ge 8$ free bits that can be conveniently used to specify the cost-reducing generator in the augmented database, see the section "Database generation". Recall that the number of generators is $m = 9n(n-1)/2|_{n\le 6} \le 135$. Thus the generator can be specified using only 8 bits. Note also that storing the full matrix $U \in \mathcal{C}_n$ using only two machine words is impossible for $n = 6$, as it requires $4n^2|_{n=6} = 144$ bits.

The thin matrix format enables fast left and right multiplication by the single-qubit and two-qubit Clifford gates, that require at most 24 CPU instructions per gate for all $n \le 6$ (each instruction implements a bitwise operation on a single machine word). When needed, the thin matrix U' can be expanded into the full symplectic matrix $U \in \mathcal{C}_n$ by calculating the missing pair of rows e_nU and $e_{2n}U$ using the symplectic version of Gram–Schmidt orthogonalization. Our implementation converts the thin matrix to the full matrix in time less than 2×10^{-7} s for any $n \le 6$ on a server-class CPU, in this case an Intel® Xeon® CPU E7-4850 v4 @ 2.10GHz, which is negligible compared with the time it takes to compute the reduced form.

Software tricks

Database generation. The calculation of the reduced cost-k Clifford group set $\mathcal{R}_n^k$, as described in the section "Database generation", lends itself to parallel processing. Specifically, each element of the set $\mathcal{R}_n^k$ can be calculated concurrently from its own data on its own processor. The implementation considerations for this run-once parallel processing job depended on factors such as:

i. the cost and availability of scaled-up/scaled-out hardware, and
ii. the cost-benefit for implementing, measuring, and tuning for different data-level parallel processing options, including shared memory versus distributed memory (e.g., OpenMP/MPI) and specialized processors (e.g., vector processors, GPUs, FPGAs),

not to mention the multiple software options with each, from programming languages to libraries[22].

Using Flynn's taxonomy[23], the *Single Program, Multiple Data* (SPMD) *streams* model was implemented using the C++ *concurrent-set* template class; specifically, each reduced cost-k Clifford group set $\mathcal{R}_n^k$ is an instance of `set<pair<uint64, uint64>>`. This is a good choice for programmer productivity, i.e., letting the container's semantics deal with the requirements of maintaining distinct and efficiently-searchable elements of a multi-terabyte set on SMP hardware, in this case an Intel® Xeon® 128-CPU E7-4850 v4 @ 2.10GHz with 6TB RAM.

Runtime was extrapolated to take about 100 days to complete the full database generation on a single machine, amounting to approximately $100 \cdot 24 \cdot 128 = 307,200$ CPU-hours that can be effectively divided among as many machines as there are available. Hardware and software measurements during database generation, using performance analysis tools such as *vmstat* to VTune™, exposed heavy "NUMA thrashing," i.e., soft page faults[24]. To alleviate this for the final half of the run, C's most basic systems programming mechanisms were more readily and easily used to replace the C++ *set* template in order to allocate, position, and search raw memory, resulting in a 5x speed-up; namely, *malloc*, *bsearch*, and *qsort*, along with *read/write* and *uint128*.

Synthesis of optimal circuits. With the one-time generation of the database complete and saved on secondary storage (Solid State Disk), similar systems programming mechanisms in C were exploited to optimize performance and scalability in order to read/search what is now effectively a lookup table (LUT), with the expensive runtime calculation of an optimal 6-qubit Clifford circuit completed and replaceable by a simple array indexing operation. The database can be memory-mapped with *mmap*[25] for a greater degree of

i. programmer productivity, i.e., the database can be easily referenced as memory using pointers, with no explicit file IO, and
ii. operational flexibility, i.e., the database can be effectively used by any type of hardware, ranging from a single laptop to a cluster of server-class machines, with scaling solely dependent on the choice of hardware,

all without changing the code; while the OS kernel and *mmap* transparently and efficiently take care of

i. demand paging, and
ii. maintaining only a single copy of data in memory, as opposed to copies in both the file cache and user space.

In addition, to reduce the number of SSD queries, being the most time-consuming operation our search relies on, we employed the following strategy:

i. we store the databases of Clifford circuits requiring 1–8, 14, and 15 gates in RAM,
ii. we store an index consisting of each 1024th element of Clifford unitaries implementable with 9–13 gates in RAM, and
iii. when the length-1024 chunk containing the desired element is found by the binary search, we make one long query to extract all 2048 64-bit integers in this chunk.

The above modification limits the number of SSD queries required to synthesize an optimal circuit to at most 10 (at most two queries per searches over the gate counts of 9, 10, 11, 12, and 13) at the cost of RAM memory usage of 2.5GB.

A machine with enough RAM to fit the entire database in will get the best performance as the complete database fills the file cache, and a machine with little-to-no available RAM will get the worst performance as every pointer access to a memory-mapped region (e.g., *bsearch*) will touch the secondary storage. A commodity machine with typical RAM sizes will get near-best performance as the "hot" parts of the database—the internal nodes of *bsearch*—will tend to remain in the cache hierarchy (L1-L3, file cache) and result in minimal access to secondary storage. OS-specific parameters were not explored but can also be benchmarked and tuned independently of the database and code, including page sizes and pinned memory.

Proof of Lemma 1

We need to show that any element $U \in \mathcal{C}_n^k$ can be written as $U = G_{a_1} G_{a_2} \cdots G_{a_k} L$ for some $L \in \mathcal{C}_n^0$ and some k-tuple of generators. We use the induction in k. The base of induction is $k = 0$, in which case the statement is trivial. Suppose $k \geq 1$ and $U \in \mathcal{C}_n^k$. By definition, U can be implemented by a circuit composed of k CNOT gates and some number of single-qubit gates. Let $\mathrm{CNOT}_{i,j}$ be the last CNOT gate in this circuit. Then

$$U = M\mathrm{CNOT}_{i,j}V$$

for some $M \in \mathcal{C}_n^0$ and $V \in \mathcal{C}_n^{k-1}$. We can assume without loss of generality that $i < j$. Indeed, if $i > j$, use the identity $\mathrm{CNOT}_{j,i} = \mathrm{H}_i\mathrm{H}_j\mathrm{CNOT}_{i,j}\mathrm{H}_i\mathrm{H}_j$ to flip the control and the target qubits of the last CNOT gate. The extra H gates can be absorbed into M and V layers. By the induction hypothesis, $V = G_{a_2} \cdots G_{a_k} L$ for some $L \in \mathcal{C}_n^0$. Furthermore, we can assume without loss of generality that $M = \mathrm{A}_i\mathrm{B}_j$ for some $\mathrm{A}, \mathrm{B} \in \mathcal{C}_1$. Indeed, all single-qubit gates in M that act on qubits $\ell \notin \{i, j\}$ can be commuted through $\mathrm{CNOT}_{i,j}$ and absorbed into V. If $\mathrm{A}, \mathrm{B} \in \{\mathrm{I}, \mathrm{HP}, \mathrm{PH}\}$, we are done. Indeed, in this case $\mathrm{A}_i\mathrm{B}_j\mathrm{CNOT}_{i,j} = G_{a_1}$ is a generator and $U = G_{a_1}V = G_{a_1}G_{a_2} \cdots G_{a_k}L$ with $L \in \mathcal{C}_n^0$. Otherwise, transform A and B into the desired form by "borrowing" the missing single-qubit gates from V and commuting them through $\mathrm{CNOT}_{i,j}$ using the Clifford group identities:

$$\mathrm{P}^2 = \mathrm{H}^2 = (\mathrm{PHP})^2 = \mathrm{I}, \quad \mathrm{PHP} = \mathrm{HPH},$$

$$\mathrm{P}_i\mathrm{CNOT}_{i,j} = \mathrm{CNOT}_{i,j}\mathrm{P}_i, \quad \mathrm{H}_i\mathrm{CNOT}_{i,j} = (\mathrm{HP})_i\mathrm{CNOT}_{i,j}\mathrm{P}_i,$$
$$(\mathrm{PHP})_i\mathrm{CNOT}_{i,j} = (\mathrm{PH})_i\mathrm{CNOT}_{i,j}\mathrm{P}_i,$$

$$\mathrm{P}_j\mathrm{CNOT}_{i,j} = (\mathrm{HP})_j\mathrm{CNOT}_{i,j}(\mathrm{PHP})_j, \mathrm{H}_j\mathrm{CNOT}_{i,j} = (\mathrm{PH})_j\mathrm{CNOT}_{i,j}(\mathrm{PHP})_j,$$
$$\text{and } (\mathrm{PHP})_j\mathrm{CNOT}_{i,j} = \mathrm{CNOT}_{i,j}(\mathrm{PHP})_j.$$

Recall that these identities only apply to elements of the binary symplectic group; the corresponding identities for unitary Clifford operators may include some extra phase factors and Pauli gates. This completes the proof.

Pauli mixing constraint

In this section, we prove that a Pauli-invariant probability distribution μ on the n-qubit Clifford group is a unitary 2-design iff μ is Pauli mixing. The fact that Pauli-invariance and Pauli mixing are sufficient for being a 2-design is known[16,Appendix D]. Thus it suffices to prove that any Pauli-invariant Clifford 2-design is Pauli mixing.

The Haar integeral in Eq. (1) can be computed explicitly using Weingarten functions[26],

$$\int_{U(2^n)} (\hat{U}^\dagger \hat{A}\hat{U}) \otimes (\hat{U}^\dagger \hat{B}\hat{U}) \mathrm{d}U = \mathrm{SWAP}\left[\frac{\mathrm{Tr}(\hat{A}\hat{B})}{4^n - 1} - \frac{\mathrm{Tr}(\hat{A})\mathrm{Tr}(\hat{B})}{2^n(4^n - 1)}\right]$$
$$+\hat{I} \otimes \hat{I}\left[\frac{\mathrm{Tr}(\hat{A})\mathrm{Tr}(\hat{B})}{4^n - 1} - \frac{\mathrm{Tr}(\hat{A}\hat{B})}{2^n(4^n - 1)}\right].$$

Here SWAP is a unitary operator that swaps the two n-qubit registers separated by the tensor product. It is well-known that any complex matrix of size $2^n \times 2^n$ can be expanded in the Pauli basis

$$\mathcal{P}_n = \{\hat{I}, \hat{X}, \hat{Y}, \hat{Z}\}^{\otimes n}.$$

Thus it suffices to impose Eq. (1) only for $\hat{A}, \hat{B} \in \mathcal{P}_n$. Noting that the Pauli basis is orthonormal with respect to the inner product $\mathrm{Tr}(\hat{A}^\dagger\hat{B})/2^n$ one concludes that a pair $(\mathcal{D}, \mu)$ is a unitary 2-design iff

$$\sum_{\hat{U} \in \mathcal{D}} \mu(\hat{U})(\hat{U}^\dagger \hat{A}\hat{U}) \otimes (\hat{U}^\dagger \hat{B}\hat{U}) = \begin{cases} 0 & \text{if} \quad \hat{A} \neq \hat{B}, \\ \hat{\Lambda} & \text{if} \quad \hat{A} = \hat{B} \neq \hat{I} \end{cases} \quad \text{for all } \hat{A}, \hat{B} \in \mathcal{P}_n \tag{19}$$

where

$$\hat{\Lambda} = \frac{1}{4^n - 1}(2^n\mathrm{SWAP} - \hat{I} \otimes \hat{I}) = \frac{1}{4^n - 1} \sum_{\hat{O} \in \mathcal{P}_n \setminus \{\hat{I}\}} \hat{O} \otimes \hat{O}.$$

A Pauli operator $\hat{O} \in \mathcal{P}_n$ can be parameterized by a bit string $v \in \{0, 1\}^{2n}$ such that

$$\hat{O}(v) \equiv \hat{O}(v_1 v_{n+1}) \otimes \hat{O}(v_2 v_{n+2}) \otimes \cdots \otimes \hat{O}(v_n v_{2n}),$$

where $\hat{O}(00) \equiv \hat{I}$, $\hat{O}(10) \equiv \hat{X}$, $\hat{O}(01) \equiv \hat{Z}$, and $\hat{O}(11) \equiv \hat{Y}$. The unitary version of the Clifford group, which we denote $\mathfrak{C}_n$, is a group of complex matrices $\hat{U} \in U(2^n)$ that map Pauli operators to Pauli operators under conjugation. More formally, $\hat{U} \in \mathfrak{C}_n$ iff there exists a symplectic matrix $U \in \mathcal{C}_n$ such that

$$\hat{U}\hat{O}(v)\hat{U}^\dagger = \pm\hat{O}(Uv) \tag{20}$$

for all $v \in \{0, 1\}^{2n}$. Here the sign may depend on v. The symplectic matrix $U \in \mathcal{C}_n$ in Eq. (20) is uniquely determined by $\hat{U}$. Conversely, $\hat{U}$ is uniquely determined by U up to (right) multiplications by Pauli operators and the overall phase. In other words, $\mathfrak{C}_n$ is isomorphic (as a set) to $\mathcal{C}_n \times \mathcal{P}_n$ if one ignores the overall phase of unitary matrices.

Suppose $\mu : \mathfrak{C}_n \to \mathbb{R}_+$ is a Pauli-invariant probability distribution, that is, $\mu(\hat{U}) = \mu(\hat{U}\hat{O})$ for all $\hat{O} \in \mathcal{P}_n$ and $\hat{U} \in \mathfrak{C}_n$. Using the isomorphism $\mathfrak{C}_n \cong \mathcal{C}_n \times \mathcal{P}_n$, define a distribution $\pi : \mathcal{C}_n \to \mathbb{R}_+$ such that $\mu(U \times P) = \pi(U)/4^n$ for all $U \in \mathcal{C}_n$ and $P \in \mathcal{P}_n$. Suppose $(\mathfrak{C}_n, \mu)$ is a 2-design, that is, μ obeys Eq. (19) with $\mathcal{D} = \mathfrak{C}_n$. Consider the second case of Eq. (19) such that $\hat{A} = \hat{B} = \hat{O}(x)$ for some non-zero vector $x \in \{0, 1\}^{2n}$. Then it is equivalent to

$$\sum_{U \in \mathcal{C}_n} \pi(U)\hat{O}(Ux) \otimes \hat{O}(Ux) = \frac{1}{4^n - 1} \sum_{y \in \{0,1\}^{2n} \setminus 0^{2n}} \hat{O}(y) \otimes \hat{O}(y).$$

Since Pauli operators are linearly independent, this is possible only if a random vector Ux with U sampled from $\pi(U)$ is distributed uniformly on the set of all non-zero vectors $\{0, 1\}^{2n} \setminus 0^{2n}$. This gives the Pauli mixing condition Eq. (3).

DATA AVAILABILITY

A Python implementation of the described algorithms will be available at: https://github.com/qiskit-community/prototype-clifford-optimizer.

Received: 12 April 2021; Accepted: 4 June 2022;
Published online: 05 July 2022

REFERENCES

1. Nielsen, M. A. & Chuang, I. *Quantum Computation and Quantum Information* (Cambridge University Press, 2002).
2. Gottesman, D. The Heisenberg representation of quantum computers. Preprint at https://arxiv.org/abs/quant-ph/9807006 (1998).
3. Aaronson, S. & Gottesman, D. Improved simulation of stabilizer circuits. *Phys. Rev. A* **70**, 052328 (2004).

4. Bravyi, S. & Maslov, D. Hadamard-free circuits expose the structure of the Clifford group. *IEEE Trans. Inform. Theory* **67**, 4546–4563 (2021).
5. Bravyi, S. & Kitaev, A. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A* **71**, 022316 (2005).
6. Knill, E. Quantum computing with realistically noisy devices. *Nature* **434**, 39–44 (2005).
7. Knill, E. et al. Randomized benchmarking of quantum gates. *Phys. Rev. A* **77**, 012307 (2008).
8. Magesan, E., Gambetta, J. M. & Emerson, J. Scalable and robust randomized benchmarking of quantum processes. *Phys. Rev. Lett.* **106**, 180504 (2011).
9. Aaronson, S. Shadow tomography of quantum states. *SIAM J. Computing* (0):STOC18–368–STOC18–394, (2020).
10. Huang, Hsin-Yuan, Kueng, R. & Preskill, J. Predicting many properties of a quantum system from very few measurements. *Nat. Phys.* **16**, 1050—1057 (2020).
11. Bennett, C. H., DiVincenzo, D. P., Smolin, J. A. & Wootters, W. K. Mixed-state entanglement and quantum error correction. *Phys. Rev. A* **54**, 3824 (1996).
12. DiVincenzo, D. P., Leung, D. W. & Terhal, B. M. Quantum data hiding. *IEEE Trans. Inform. Theory* **48**, 580–598 (2002).
13. IBM. IBM Quantum Experience. https://quantum-computing.ibm.com/, last accessed 10/5/2020.
14. Amazon Web Services. Amazon Bracket. https://aws.amazon.com/braket/, last accessed 10/5/2020.
15. R. A., Low. Pseudo-randomness and learning in quantum computation. PhD Thesis, University of Bristol, UK (2010).
16. Cleve, R., Leung, D. W., Liu, L. & Wang, C. Near-linear constructions of exact unitary 2-designs. *Quantum Inform. Comput.* **16**, 721–756 (2016).
17. Emerson, J., Alicki, R. & Życzkowski, K. Scalable noise estimation with random unitary operators. *J. Opt. B: Quantum Semiclassical Opt* **7**, S347 (2005).
18. Dankert, C., Cleve, R., Emerson, J. & Livine, E. Exact and approximate unitary 2-designs and their application to fidelity estimation. *Phys. Rev. A* **80**, 012304 (2009).
19. Kliuchnikov, V. & Maslov, D. Optimization of Clifford circuits. *Phys. Rev. A* **88**, 052307 (2013).
20. Golubitsky, O. & Maslov, D. A study of optimal 4-bit reversible Toffoli circuits and their synthesis. *IEEE Trans. Comput.* **61**, 1341–1353 (2011).
21. Rokicki, T., Kociemba, H., Davidson, M. & Dethridge, J. The diameter of the Rubik's cube group is twenty. *SIAM Rev.* **56**, 645–670 (2014).
22. Clang project. Clang version 9.0.0.
23. Wikipedia contributors. Flynn's taxonomy. https://en.wikipedia.org/wiki/Flynn's_taxonomy (2020) (accessed 20 October 2020).
24. Wikipedia contributors. Page fault. https://en.wikipedia.org/wiki/Page_fault (2020).
25. Wikipedia contributors. mmap. https://en.wikipedia.org/wiki/Mmap (2020). See Further reading for the Windows® mmap equivalent (accessed 20 October 2020).
26. Collins, B. & Śniady, P. Integration with respect to the Haar measure on unitary, orthogonal and symplectic group. *Commun. Math. Phys.* **264**, 773–795 (2006).

AUTHOR CONTRIBUTIONS

All authors contributed equally to this work.

COMPETING INTERESTS

The authors declare no Competing Non-Financial Interests but the following Competing Financial Interests. A provisional patent application US20220114468A1 covering this work was filed by IBM.

ADDITIONAL INFORMATION

Correspondence and requests for materials should be addressed to Dmitri Maslov.

Reprints and permission information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.