Fast algorithms for classical specifications of stabiliser states and Clifford gates

Nadish de Silva[1], Wilfred Salmon[2], and Ming Yin[1]

[1]Department of Mathematics, Simon Fraser University
[2]Department of Applied Mathematics and Theoretical Physics, University of Cambridge

The stabiliser formalism plays a central role in quantum computing, error correction, and fault tolerance. Conversions between and verifications of different specifications of stabiliser states and Clifford gates are important components of many classical algorithms in quantum information, e.g. for gate synthesis, circuit optimisation, and simulating quantum circuits. These core functions are also used in the numerical experiments critical to formulating and testing mathematical conjectures on the stabiliser formalism.

We develop novel mathematical insights concerning stabiliser states and Clifford gates that significantly clarify their descriptions. We then utilise these to provide ten new fast algorithms which offer asymptotic advantages over any existing implementations. We show how to rapidly verify that a vector is a stabiliser state, and interconvert between its specification as amplitudes, a quadratic form, and a check matrix. These methods are leveraged to rapidly check if a given unitary matrix is a Clifford gate and to interconvert between the matrix of a Clifford gate and its compact specification as a stabiliser tableau.

For example, we extract the stabiliser tableau of a $2^n \times 2^n$ matrix, promised to be a Clifford gate, in $O(n2^n)$ time. Remarkably, it is not necessary to read all the elements of a Clifford gate matrix to extract its stabiliser tableau. This is an asymptotic speedup over the best-known method that is exponential in the number of qubits.

We provide implementations of our algorithms in `Python` and `C++` that exhibit vastly improved practical performance over existing algorithms in the cases where they exist.

Nadish de Silva: nadish_de_silva@sfu.ca

Contents

1 Introduction **3**
- 1.1 Summary of results . . . 5
- 1.2 Applications . . . 6

2 Background **8**
- 2.1 Stabiliser states . . . 8
- 2.2 Clifford gates . . . 10

3 Stabiliser state algorithms **10**
- 3.1 Amplitudes to a quadratic form triple . . . 10
- 3.2 Quadratic form triple to amplitudes . . . 12
- 3.3 Verifying a vector is a stabiliser state . . . 13
- 3.4 Check matrix to a quadratic form triple . . . 13
- 3.5 Quadratic form triple to a check matrix . . . 14

4 Clifford gate algorithms **15**
- 4.1 Matrix entries to a conjugate tuple . . . 15
- 4.2 Verifying a unitary matrix is a Clifford gate . . . 17
- 4.3 Conjugate tuple to a matrix . . . 18

5 Complexity analyses **18**
- 5.1 Stabiliser states . . . 18
- 5.2 Clifford gates . . . 20

6 Acknowledgments **20**

A Complexity comparisons to existing implementations **24**

B Timed benchmarks **26**

1 Introduction

The stabiliser formalism is an essential pillar of the theory of quantum computation. It consists of a particularly natural subset of quantum states, gates, and measurements: stabiliser states, Clifford gates, and Pauli measurements, respectively (see Section 2 for precise definitions). The stabiliser formalism has played a key role in quantum information from its earliest days. For example, communication protocols such as Wiesner's quantum money scheme [50], BB84 key distribution [10], superdense coding [11], quantum teleportation [13], and entanglement distillation [12] all utilise elements of the stabiliser formalism. The first proofs of nonlocality, an important topic in quantum foundations, used stabiliser states such as the Bell state [9] and the GHZ state [32]. Early quantum algorithms such as the Deutsch-Jozsa [26], Bernstein-Vazirani [14], and Simon algorithm [44] use only stabiliser operations on their input (given as an oracle).

The elements of the stabiliser formalism were first collectively defined in the context of quantum error-correction [30]. Nearly all quantum error-correcting codes presently under study are examples of *stabiliser codes*. These codes use stabiliser states to encode basic quantum data, i.e. logical basis states, and Pauli measurements to detect errors. The group of Clifford gates, which arise as symmetries of the group of Pauli measurements, represent operations that are easily (e.g. transversally) performed fault-tolerantly on encoded data [29].

The Eastin-Knill theorem [27] precludes universal fault-tolerant quantum computation using only a transversal set of quantum gates, which significantly complicates fault-tolerant frameworks. To circumvent this, most proposed fault-tolerant schemes use so-called magic states, which lead to universal quantum computation when taken with stabiliser operations [16]. Thus, most fault-tolerant quantum architectures rely on stabiliser operations and magic state generation, placing the stabiliser formalism at the heart of experimentally-realisable quantum computers. In particular, considerable attention is paid to leveraging the stabiliser formalism to minimise the use of costly magic states when synthesising gates or circuits. A deeper understanding of these subjects will hasten the arrival of practical quantum computers.

The stabiliser formalism has also played an central role in the study of classical simulation algorithms for quantum circuits since the introduction of the the Gottesman-Knill theorem [31, 2, 6]. The theorem now has many variants [34], all of which roughly state that adaptive circuits restricted to stabiliser operations can be classically efficiently simulated. At the core of the Gottesman-Knill theorem is the observation that elements of the stabiliser formalism, as a result of their defining symmetries, have classical descriptions that are much more compact than their standard Hilbert space descriptions.

Extensions of the Gottesman-Knill theorem are used to simulate universal quantum computation: e.g. the stabiliser rank algorithm [18] and the sum-over-Cliffords algorithm [20]. These algorithms reduce the case of simulating arbitrary circuits to the stabiliser case, with overhead scaling with how 'nonstabiliser' the circuit and input state are. Such algorithms have practical importance for testing quantum circuits, as well as theoretical importance in the study of quantum advantage.

Stabiliser operations are used across many other areas within quantum information. Some examples include resource states for measurement-based quantum computation [42], randomised benchmarking [37], and quantum learning algorithms [33]. There is substantial interest in, and active research on, finding good quantum protocols for the verification of stabiliser states [24] and Clifford gates [51].

Given the importance of the stabiliser formalism to so many areas of quantum in-

formation and computation, it is an urgent task to develop fast classical algorithms for working with its elements. Improvements to the core subroutines for computations using the stabiliser operations will find applications in, for example: classical simulation algorithms, circuit compilation, and gate synthesis. They will also enable better numerical experiments leading to the development of the mathematical theory of the stabiliser formalism.

In this article, we give ten novel fast algorithms for working with classical descriptions of elements of stabiliser theory. In particular, we consider three descriptions of n-qubit stabiliser states:

[$\mathrm{S_V}$] as a complex *vector* of amplitudes,

[$\mathrm{S_Q}$] as a *quadratic* form, a linear map, and an affine subspace of $\mathbb{Z}_2^n$,

[$\mathrm{S_P}$] as a check matrix, i.e. a compact list of *Pauli* gate generators for the stabiliser group.

We also consider two representations of n-qubit Clifford gates:

[$\mathrm{C_U}$] as a *unitary* matrix,

[$\mathrm{C_T}$] as a list of $2n$ Pauli gates representing the images of basic Pauli gates under conjugation, i.e. a *tableau*.

We give algorithms for interconverting between these descriptions, as well as algorithms for verifying whether a given state vector is a stabiliser state, or a given unitary is a Clifford gate. We achieve very large reductions in their runtimes with respect to existing implementations, by developing mathematical insights that conceptually clarify the elements of the stabiliser formalism.

Implementations of our algorithms in `Python` and `C++` can be found at https://github.com/ndesilva/stabiliser-tools.

- In Subsection 1.1, we summarise our novel algorithms and their complexity advantages.
- In Subsection 1.2, we suggest possible domains wherein applications of our algorithms can be useful.
- In Section 2, we provide the necessary definitions and give the precise definitions of the various inputs and outputs of our algorithms. We also describe the naïve brute force methods for verifying that a vector or matrix is a stabiliser state or a Clifford gate respectively and for interconverting between their descriptions.
- In Sections 3 and 4, we develop novel theory concerning stabiliser states and Clifford gates respectively. We use these developments to give much faster algorithms for verifying that a vector or matrix is a stabiliser state or a Clifford gate. We also give much faster algorithms for interconverting between their descriptions.
- In Section 5, we give detailed analyses of the worst-case asymptotic time complexity of our ten algorithms.
- In Appendix A, we give detailed analyses of the worst-case asymptotic time complexity of existing implementations of the functions we describe, for the cases where they exist.

- In Appendix B, we provide plots of timed benchmarks of our `C++` implementations. Wherever possible, we compare with existing implementations from `Stim` [28] and `Qiskit` [41]. We find absolute advantages of some orders of magnitude for small numbers of qubits; these advantages are guaranteed to become arbitrarily large as n grows.

1.1 Summary of results

In this work, we describe ten novel algorithms for interconverting between the above descriptions and for verifying that a candidate description is valid. In the table on the following page, we compare the worst-case asymptotic complexity of our algorithms to the state-of-the-art. The guide to interpreting these tables immediately follows them.

In all cases where algorithms performing the same functionality are known to exist, we find that our algorithm has a complexity advantage of a factor of at least n, the number of qubits in the system. The complexity of naïve brute force methods are analysed in Subsections 2.1 and 2.2. Improvements on brute force are given in [28] and [41]; see Appendix A for their complexity analyses.

Given that proposed error-correcting systems require components built of many-qubit subsystems [23], improvements of a factor of n are already highly consequential. In some cases, we find our algorithm has an advantage by a factor of at least $N = 2^n$: an exponential or higher improvement.

In practice, our algorithms exhibit excellent performance for any number of qubits. We achieve this by working directly with the mathematical description in question. See Appendix B for timed benchmarks of our `C++` implementations against existing algorithms [28, 41] in the cases where they exist.

In two cases, we give conversion algorithms for which no alternative implementations exist.

	$[\mathbf{S_V}]$	$[\mathbf{S_Q}]$	$[\mathbf{S_P}]$
$[\mathbf{S_V}]$	$Nn^2 \to N$	$? \to N$	$Nn^2 \to N$
$[\mathbf{S_Q}]$	$Nn^2 \to Nn$	n^2	$Nn^2 \to n^3$
$[\mathbf{S_P}]$	$N^4/Nn^2 \to Nn$	$? \to n^3$	n^3

	$[\mathbf{C_U}]$	$[\mathbf{C_T}]$
$[\mathbf{C_U}]$	$N^2n^2 \to N^2n$	$N^2n^2 \to Nn$
$[\mathbf{C_T}]$	$N^2n^2 \to N^2n$	n^3

Table 1: Complexity of existing methods vs. new methods.

- Grey cells correspond to simple tasks, all related to the verification of compact descriptions, for which there exists an obvious method that is nearly optimally fast.

- Diagonal entries of tables correspond to algorithms for verifying candidate descriptions. For example, the ($[\mathrm{S_V}]$,$[\mathrm{S_V}]$)-entry in the top-left of the table describes the problem of taking as input a vector in $\mathbb{C}^N$ and deciding whether it represents a valid stabiliser state.

  Off-diagonal entries of tables correspond to algorithms for converting from one valid description to another. For example, the ($[\mathrm{S_V}]$,$[\mathrm{S_P}]$)-entry in the top-right of the table describes the problem of taking as input a vector in $\mathbb{C}^N$ that represents a valid stabiliser state and gives as output its check matrix.

- An entry of the form $X \to Y$ indicates that the best currently-known technique requires time $\Omega(X)$ whereas our methods require only $O(Y)$ time.

  An entry of the form $X_1/X_2 \to Y$ indicates that the best currently-known technique that is guaranteed to succeed requires time $\Omega(X_1)$ and the best currently-known technique that succeeds with high probability requires time $\Omega(X_2)$ whereas our methods require only $O(Y)$ time.

- An entry of the form $? \to Y$ indicates that there is neither an existing implementation nor an obvious method for the interconversion task in question; we give a method that requires time $O(Y)$.

1.2 Applications

The need for fast classical algorithms for verifying and converting descriptions of stabiliser elements is evidenced by online queries [47, 46] and their inclusion in popular software packages [28, 41]. This is no surprise given the ubiquity of the stabiliser formalism in quantum information. We identify a number of potential applications in the areas of classical simulation of quantum circuits, gate synthesis, and circuit optimisation; we are sure of the existence of other potential domains of applicability that we cannot anticipate. We conclude by suggesting how our methods can be used to formulate and test mathematical conjectures concerning the stabiliser formalism.

Classical simulation. Given the classical efficiency of the Gottesman-Knill algorithm, it is the standard method for simulating the stabiliser formalism. However, in some regimes, alternative simulation techniques can be faster. Our conversions are useful at the interfaces between these regimes. For example, recent work by de Beaudrap and Herbert [8] shows that stabiliser simulation can be made faster in certain cases using the *quadratic form expansion*, which is equivalent to what we term a quadratic form triple, of a stabiliser state. These cases include: performing stabiliser operations where the quantum state has a small computational basis expansion; simulating deterministic single-qubit Pauli measurements; and, particularly of interest, simulating local syndrome measurements for encoded stabiliser circuits. In contrast, Gottesman-Knill-type algorithms use the check matrix form of a stabiliser state. Thus our conversions to and from the quadratic form are useful for employing a hybrid algorithm that makes use of the strengths of both types of simulation.

Much work has been done on classical simulation of general quantum circuits by separating stabiliser subcircuits from nonstabiliser parts, e.g. recently by Smith et al. [45]. Since many popular quantum circuit simulators use state vectors (e.g. [49]), our conversions from the other specifications of a stabiliser state to the state vector will speed up the classical processing at these stabiliser-nonstabiliser interfaces.

Next, we note that stabiliser operations together with access to magic states are sufficient for universal quantum computation. Our tableau-to-unitary matrix conversion for Clifford gates can prove useful if one wishes to apply a Clifford, perhaps resulting from back-propagation of Pauli measurements, to initial magic states. As another example, in the sum-over-Cliffords method [20], a unitary is decomposed as a linear combination of Clifford gates which are easier to simulate. If this decomposition is found in unitary form (perhaps by recursively subtracting the closest Clifford gate from a unitary), our unitary matrix-to-tableau conversion would accelerate the algorithm.

Gate synthesis. Here we describe an example application of our methods that has already improved the efficiency of gate synthesis algorithms. Clifford isometries are a generalisation of standard Clifford gates which are of considerable interest in areas such as magic state distillation [17]. Recent work by Kliuchnikov, Beverland and Paetznick [35], followed by Kliuchnikov and Schonnenbeck [36], describes an efficient way to determine whether a unitary (in complex matrix form) is a Clifford isometry, and if so, to compile it over a chosen gateset [21]. The method uses the Choi stabiliser state of the operator, which must be converted from state vector form to check matrix form before the next steps can be used to find the tableau form of the Clifford isometry [7]. Our method of converting the Choi state to check matrix form is used to improve the efficiency of this method [36, p. 7].

Circuit synthesis and optimisation. General quantum circuits are commonly compiled in terms of Pauli exponentials [3], e.g. for T-count reduction algorithms [52] or in the context of quantum chemistry problems such as estimating ground state energies of Hamiltonians [22]. A Pauli exponential has the general form $R_P(\theta) = e^{-i\frac{\theta}{2}P}$, where P is a Pauli gate, and can be readily diagonalised into a form such as $R_Z(\varphi)$. The action of diagonal Pauli exponentials and CNOT gates [39] can be readily expressed in a phase polynomial form which is somewhat more general than the quadratic form triple of a stabiliser state. Phase polynomials have been extensively studied for circuit optimisation [5], e.g. in order to reduce the count of two-qubit gates. Thus, having fast algorithms for converting to and from the quadratic form representation of stabiliser states may aid the synthesis of quantum circuits, particularly if one wishes to use phase polynomials to

perform some kind of optimisation on a subcircuit. On a related note, it may help to use these conversions in cases where one wishes to use a Clifford to conjugate a fixed Pauli to another fixed Pauli [3, 4], since it may be useful to perform phase polynomial optimisation on a number of different such Cliffords to decide which one to choose.

Mathematical exploration. Our methods are also useful as research tools in quantum information by enabling faster and larger numerical experiments. As an example, the present work was initially motivated by the problem of searching for low-rank stabiliser decompositions [18] of magic states. Unlike previous approaches, which search over sets of stabiliser states, our approach is to generate promising small linear combinations of vectors and verify that they are indeed stabiliser states. As another example, in the context of searching for proofs of quantum advantage via boson sampling [1], one method under development [38] involves translating photonics circuits into unitary matrices and checking whether the result is a Clifford gate. More broadly, our methods can be used to formulate and test mathematical conjectures. This may take the form of numerical experiments that check whether the outcome of a randomised procedure is always a stabiliser state or Clifford gate. Such experiments could deploy our algorithms millions or billions of times, multiplying their performance advantages.

We have given above possible use cases for all ten of our novel algorithms. We anticipate that many more will be found by those with expertise in areas beyond our own.

2 Background

Let n be a positive integer and $N = 2^n$. We denote the computational basis states of an n-qubit system by $|\vec{z}\rangle \in \mathbb{C}^N$ for $\vec{z} \in \mathbb{Z}_2^n$. Given n unitaries $U_1, \ldots, U_n$ and a vector $\vec{p}$ of n integers, we denote by $U^{\vec{p}}$ the product $U_1^{p_1} \cdots U_n^{p_n}$. The basic Pauli gates are Z_i, X_i: i.e. Z, X respectively on the i-th qubit and identity on all others. Thus, $Z^{\vec{p}}$ and $X^{\vec{q}}$, with $\vec{p}, \vec{q} \in \mathbb{Z}_2^n$, are defined by

$$Z^{\vec{p}}\,|\vec{z}\rangle = (-1)^{\vec{p}\cdot\vec{z}}\,|\vec{z}\rangle \qquad X^{\vec{q}}\,|\vec{z}\rangle = |\vec{z}+\vec{q}\rangle \tag{1}$$

where the addition is entrywise and in $\mathbb{Z}_2$.

Definition 1. *The group of* Pauli gates *is the subgroup of $\mathcal{U}(2^n)$ generated by the basic Pauli gates and $i\mathbb{I}$:*

$$\mathcal{P}_n = \{(-1)^c (-i)^d X^{\vec{q}} Z^{\vec{p}} \mid (\vec{p}, \vec{q}) \in \mathbb{Z}_2^{2n},\ c, d \in \mathbb{Z}_2\}. \tag{2}$$

The Pauli gates $X^{\vec{q}_1} Z^{\vec{p}_1}$ and $X^{\vec{q}_2} Z^{\vec{p}_2}$ commute if and only if $[(\vec{p}_1, \vec{q}_1), (\vec{p}_2, \vec{q}_2)] \equiv \vec{p}_1 \cdot \vec{q}_2 - \vec{p}_2 \cdot \vec{q}_1 = 0$ over $\mathbb{Z}_2$. A set of Pauli gates is said to be independent if no nontrivial product of them is a multiple of the identity matrix.

2.1 Stabiliser states

Here, we define stabiliser states and the three ways they can be specified (in the first case, precisely, and in the latter two cases, up to phase).

Definition 2. *A* stabiliser subgroup *is a maximal abelian subgroup $\mathcal{S}$ of $\mathcal{P}_n$ that does not contain $-\mathbb{I}$.*

The stabiliser subgroups $\mathcal{S} \subset \mathcal{P}_n$ are all of size N. A stabiliser subgroup can be specified (nonuniquely) by a *check matrix*: a $n \times (2n+1)$ matrix over $\mathbb{Z}_2$

$$\begin{bmatrix} \vec{q}_1 & \vec{p}_1 & c_1 \\ \vdots & \vdots & \vdots \\ \vec{q}_n & \vec{p}_n & c_n \end{bmatrix} \tag{3}$$

that collates a set of n commuting Pauli gates $(-1)^{c_i}(-i)^{\vec{p}\cdot\vec{q}} Z^{\vec{p}} X^{\vec{q}}$ that generate $\mathcal{S}$.

Definition 3. *A* stabiliser state *is a state* $|s\rangle \in \mathbb{C}^N$ *such that:*

$$P\,|s\rangle = |s\rangle \tag{4}$$

for all $P \in \mathcal{S}$ *in some stabiliser subgroup* $\mathcal{S} \subset \mathcal{P}_n$.

A stabiliser state $|s\rangle$ is specified up to phase by a (nonunique) check matrix for the stabiliser subgroup $\mathcal{S}_{|s\rangle} = \{P \in \mathcal{P}_n \mid P\,|s\rangle = |s\rangle\}$.

Theorem 1 (Dehaene-De Moor [25], 2003)**.** *Every stabiliser state* $|s\rangle$ *(up to phase and normalisation) is specified by a triple* $(\mathcal{A}, Q, \ell)$ *where* $\mathcal{A} \subset \mathbb{Z}_2^n$ *is the affine subspace* $V + \vec{z}_0$ *for* $V \subset \mathbb{Z}_2^n$ *a vector subspace and* $\vec{z}_0 \in \mathbb{Z}_2^n$, $Q : V \to \mathbb{Z}_2$ *is a quadratic form, and* $\ell : V \to \mathbb{Z}_2$ *is a linear map:*

$$|s\rangle \propto \sum_{\vec{z} \in V} (-1)^{Q(\vec{z})} i^{\ell(\vec{z})}\,|\vec{z} + \vec{z}_0\rangle\,. \tag{5}$$

We will refer to this description as the *quadratic form triple* of a stabiliser state for brevity. Here, $\mathcal{A}$ is of dimension $k \leq n$ and can be specified by k basis vectors $\vec{z}_1, \ldots, \vec{z}_k$ for V, and a shift vector $\vec{z}_0$.

Theorem 1 has been further generalised to a more general class of states: see Ref. [40] for details.

We have thus described three ways of specifying a stabiliser state (up to phase):

[$\mathrm{S_V}$] as a complex vector of amplitudes: an element of $\mathbb{C}^N$,

[$\mathrm{S_Q}$] as an affine subspace, a linear map, and a quadratic form: an element of $(\mathbb{Z}_2^n)^{k+1} \times \mathbb{Z}_2^n \times \mathbb{Z}_2^{n(n+1)}$,

[$\mathrm{S_P}$] as a check matrix: an element of $(\mathbb{Z}_2^{2n+1})^n$.

The brute force test for whether $|\psi\rangle \in \mathbb{C}^N$ is a stabiliser state requires iterating through each of the N^2 Pauli gates P, computing $P\,|\psi\rangle$, and determining the number of Pauli gates that stabilise $|\psi\rangle$. Verifying [$\mathrm{S_Q}$] requires checking that the proposed basis vectors are linearly independent, taking time $O(n^2)$. Verifying [$\mathrm{S_P}$] requires checking that the n Pauli gates are independent and commute, which takes time $O(n^3)$.

The brute force method to convert a description from [$\mathrm{S_V}$] to [$\mathrm{S_P}$], is to iterate through the Pauli gates until a stabiliser group is constructed. To convert a description from [$\mathrm{S_P}$] to [$\mathrm{S_V}$], one constructs the projection onto the state by summing all the Pauli gates in its stabiliser group and finding the projector's +1-eigenvector. This is very costly as it involves multiplying and inverting matrices of size N. Converting from [$\mathrm{S_Q}$] to [$\mathrm{S_V}$] via brute force involves evaluating a quadratic form to find each amplitude; converting from [$\mathrm{S_Q}$] to [$\mathrm{S_P}$] would then require composing the result with the above [$\mathrm{S_P}$] to [$\mathrm{S_V}$] conversion. Methods of converting [$\mathrm{S_V}$] or [$\mathrm{S_P}$] to [$\mathrm{S_Q}$] are not immediately obvious as the work of Dehaene-De Moor does not readily admit implementation as an algorithm.

2.2 Clifford gates

Definition 4. *The group of* Clifford gates *is the normaliser of the group of Pauli gates as a subgroup of the unitary matrices* $\mathcal{U}(N)$*:*

$$\mathcal{C}_n = \{C \in \mathcal{U}(N) \mid C\mathcal{P}_n C^* \subseteq \mathcal{C}_n\}. \tag{6}$$

The Clifford gates, up to phase, are in correspondence with *conjugate tuples* [43, Definition 3.6] of Pauli gates: these are n-tuples of pairs of Pauli gates $((U_1, V_1), \ldots, (U_n, V_n))$ such that $U_i^2 = V_i^2 = \mathbb{I}$, $U_i V_j = (-1)^{\delta_{ij}} V_j U_i$, $U_i U_j = U_j U_i$, and $V_i V_j = V_j V_i$ [43, Theorem 3.11]. A Clifford gate C, up to phase, corresponds to the conjugate tuple $((CZ_1C^*, CX_1C^*), \ldots, (CZ_nC^*, CX_nC^*))$. Each such Pauli, being of order 2, is of the form $(-1)^c(-i)^{\vec{p}\cdot\vec{q}} Z^{\vec{p}} X^{\vec{q}}$ and thus specified by $2n+1$ bits.

The conjugate tuple specification of a Clifford gate is thus vastly more compact ($4n^2 + 2n$ bits vs. N^2 complex numbers) and can be used more easily to e.g. act on stabiliser states in check matrix form or conjugate Pauli gates expressed as vectors in $\mathbb{Z}_2^{2n+1}$. It is well known in the literature as the *stabiliser tableau* of the Clifford gate.

We have thus described two ways of specifying a Clifford gate (up to phase):

[$\mathrm{C_U}$] as a complex matrix C: an element of $\mathbb{C}^{N\times N}$

[$\mathrm{C_T}$] as a conjugate tuple of Pauli gates $U_i = CZ_iC^*, V_i = CX_iC^*$: an element of $((\mathbb{Z}_2^{2n+1})^2)^n$.

The brute force method to verify that a matrix M is a Clifford gate is to conjugate the basic Pauli gates and verify that the results are also Pauli gates; this requires conjugating $2N$ matrices of size N; every matrix multiplication requires roughly N^3 steps. To verify [$\mathrm{C_T}$], one must check that the given Pauli gates satisfy the canonical commutation relations, which takes time $O(n^3)$.

The brute force method to convert a description from [$\mathrm{C_U}$] to [$\mathrm{C_T}$] is to conjugate the basic Pauli gates as above. The converse direction is described in [43, Theorem 3.9]; there, it requires a conversion between descriptions [$\mathrm{S_V}$] to [$\mathrm{S_P}$] of a stabiliser state. We can therefore use our new methods to improve the conversion [$\mathrm{C_T}$] to [$\mathrm{C_U}$], as detailed below.

3 Stabiliser state algorithms

Here, we describe a rapid method for extracting from a vector of amplitudes of a stabiliser state its quadratic form triple: that is, the conversion [$\mathrm{S_V}$] $\to$ [$\mathrm{S_Q}$]. We then provide a rapid method for the converse conversion [$\mathrm{S_Q}$] $\to$ [$\mathrm{S_V}$]. With some further checks, our algorithms also give a natural method for verifying that a complex vector is a stabiliser state. We also show that the conversions between a check matrix and a quadratic form triple and vice versa—[$\mathrm{S_P}$] $\to$ [$\mathrm{S_Q}$] and [$\mathrm{S_Q}$] $\to$ [$\mathrm{S_P}$]—can be done very quickly.

3.1 [$\mathrm{S_V}$] $\to$ [$\mathrm{S_Q}$]: Amplitudes to a quadratic form triple

Theorem 1 already imposes strong constraints on the amplitudes $\langle \vec{z}|s\rangle$ of a stabiliser state $|s\rangle$: up to a common factor, they must take values in $\{\pm 1, \pm i\}$ and the support $\mathrm{supp}(|s\rangle) = \{\vec{z} \in \mathbb{Z}_2^n \mid \langle \vec{z}|s\rangle \neq 0\}$ of $|s\rangle$ is an affine subspace $\mathcal{A} = V + \vec{z}_0$ of dimension $k \leq n$. We can break converting the amplitudes of $|s\rangle$ to a quadratic form triple into two stages:

1. Find a basis $\vec{z}_1, \ldots, \vec{z}_k$ for V and $\vec{z}_0$ such that the support $\mathrm{supp}(|s\rangle) = \mathcal{A} = V + \vec{z}_0$.

2. Find the coefficients of the quadratic form Q and the linear map ℓ as in Theorem 1.

To perform the first step, we convert $\mathrm{supp}(|s\rangle)$ into a vector space by taking $\vec{z}_0$ to be the first element (in the natural ordering on bitstrings) of $\mathrm{supp}(|s\rangle)$ and subtracting $\vec{z}_0$ from every element of $\mathrm{supp}(|s\rangle)$. Using the following lemma, we can conclude that this procedure leaves us with a *sorted* list of the vectors in V.

Lemma 1. *Let $\vec{v}, \vec{w}, \vec{z}_0 \in \mathbb{Z}_2^n$. Suppose that $\vec{v} < \vec{w}$ and $\vec{w} + \vec{z}_0 < \vec{v} + \vec{z}_0$. Then $(\vec{v} + \vec{w}) + \vec{z}_0 < \vec{z}_0$.*

Proof. Here, the notation $(\vec{a}, \vec{b})$ denotes the concatenation of bitstrings, i.e. a direct sum of vectors.

As $\vec{v} < \vec{w}$, we have that $\vec{v} = (\vec{v}', 0, \vec{c})$ and $\vec{w} = (\vec{w}', 1, \vec{c})$ for some (possibly empty) substrings $\vec{v}', \vec{w}', \vec{c}$.

Expressing $\vec{z}_0$ as $\vec{z}_0 = (\vec{d}, e, \vec{f})$, we have $\vec{v} + \vec{z}_0 = (\vec{v}' + \vec{d}, e, \vec{c} + \vec{f})$ and $\vec{w} + \vec{z}_0 = (\vec{w}' + \vec{d}, e + 1, \vec{c} + \vec{f})$. Therefore, $\vec{v} + \vec{z}_0 > \vec{w} + \vec{z}_0$ if and only if $e = 1$.

If $e = 1$, then $(\vec{v} + \vec{w}) + \vec{z}_0 = (\vec{v}' + \vec{w}' + \vec{d}, 0, \vec{f}) < \vec{z}_0$. □

Since $\vec{z}_0$ is the minimal element of $\mathcal{A}$, we can conclude that the process of subtracting $\vec{z}_0$ from every element of a sorted list of $\mathcal{A}$ results in a sorted list.

As we now have V as a sorted list, the basis vectors $\vec{z}_i$ are simply those whose position in the list is a power of 2. This is a consequence of the following lemma.

Lemma 2. *For any $m \in \mathbb{N}$ and any $\vec{x} = (x_m, \ldots, x_1) \in \mathbb{Z}_2^m$, denote by $I(\vec{x})$ its corresponding integer $\sum_{i=1}^m x_i 2^{i-1}$. Suppose $V \subseteq \mathbb{Z}_2^n$ is a subspace of dimension k ordered such that $\vec{v} < \vec{v}' \iff I(\vec{v}) < I(\vec{v}')$ and, in this order, $V = \{\vec{v}_0, \ldots, \vec{v}_{2^k-1}\}$. Then the map $T : \mathbb{Z}_2^k \to V$ given by $T(\vec{x}) = \vec{v}_{I(\vec{x})}$ is a linear isomorphism.*

Proof. Let $L : V \to \{0, \ldots, n\}$ return the position of the leftmost digit, counting from the right end; $L(\vec{v}_0) = 0$. This function is clearly nondecreasing.

For $j \in \{1, \ldots, k\}$: $\vec{w}_j = \vec{v}_{2^{j-1}}$ and $p_j = L(\vec{w}_j)$; $p_0 = 0$.

For $j \in \{0, \ldots, k\}$: define $V_j = \{\vec{v}_0, \ldots, \vec{v}_{2^j-1}\} = \{\vec{v} \in V \mid \vec{v} < \vec{w}_{j+1}\}$.

We will prove the following claim by induction: $L^{-1}(\{p_0, \ldots, p_j\}) = V_j$ for $j \in \{0, \ldots, k\}$.

This trivially holds for $j = 0$; assume it holds for j. Using the definition of V_j and the induction hypothesis to establish two containments respectively, we see that: $L^{-1}(p_{j+1}) = V_j + \vec{w}_{j+1}$. This set contains 2^j distinct, consecutive elements, including $\vec{w}_{j+1}$, and is disjoint from V_j. It is thus equal to $V_{j+1} \setminus V_j$ and our claim is proved.

Thus, $\{p_k, \ldots, p_1\}$ are distinct and so $(\vec{w}_k, \ldots, \vec{w}_1)$ is an ordered basis of V.

Let $T : \mathbb{Z}_2^k \to V$ be given by $T(\vec{x}) = \sum_{i=1}^k x_i \vec{w}_i$. We can prove that $\sum_{i=1}^k x_i \vec{w}_i = \vec{v}_{I(\vec{x})}$ for each V_j by induction on j. This requires the fact that adding $\vec{w}_{j+1}$ to elements of V_j is an order-preserving operation. Suppose that $\vec{v} < \vec{v}' \in V_j$. Then $\vec{v}' + \vec{w}_{j+1} < \vec{v} + \vec{w}_{j+1}$ only if $\vec{w}_{j+1}$ contains a 1 at the rightmost position in which $\vec{v}$ and $\vec{v}'$ differ. In this case, $\vec{v} + \vec{v}' + \vec{w}_{j+1} \in V_{j+1} \setminus V_j$ but is strictly less than $\vec{w}_{j+1}$; a contradiction. □

Having extracted $\vec{z}_0$ and transformed the support such that $\mathrm{supp}(|s\rangle) = V = \{\sum_i \alpha_i \vec{z}_i \mid \alpha_i \in \mathbb{Z}_2\}$, we can determine Q, ℓ as functions of the vectors $\vec{\alpha}$. From the amplitudes corresponding to members of the support with $\vec{\alpha}$ having Hamming weight 1, we can extract

ℓ. Similarly, from the amplitudes corresponding to members of the support with $\vec{\alpha}$ having Hamming weight 1 or 2, we can extract Q. Each such $\vec{\alpha}$ gives an equation:

$$\sum_{1\leq i\leq j\leq k} c_{ij}\alpha_i\alpha_j = Q\left(\sum_i \alpha_i \vec{z}_i\right). \tag{7}$$

We simply need to solve this inhomogenous linear system of $k(k+1)/2$ equations for the coefficients c_{ij}.

Algorithm 1: Convert the amplitudes of a stabiliser state to a quadratic form triple, i.e. $[\mathrm{S_V}] \to [\mathrm{S_Q}]$

1. Choose $\vec{z}_0$ to be the binary label for the first nonzero amplitude. Subtract $\vec{z}_0$ from each element of $\mathrm{supp}(|s\rangle) = V + \vec{z}_0$ to transform it to the vector space V.
2. Sort the binary labels of $\mathrm{supp}(|s\rangle) = V$. Taking those labels whose position is a power of 2 gives a basis $\{\vec{z}_i\}$ for this vector space.
3. Extract ℓ by inspecting the amplitudes corresponding to $\vec{z}_i$ and observing whether they are real or imaginary.
4. Extract Q by solving the linear system (7).

3.2 $[\mathrm{S_Q}] \to [\mathrm{S_V}]$: Quadratic form triple to amplitudes

The obvious conversion method is to evaluate the quadratic form for each element of the support of the state vector. Indeed, it is surprising that one can improve upon this method. We obtain an asymptotic advantage by carefully choosing the order in which we iterate through the support. With this order, we save time by avoiding a fresh evaluation of the quadratic form; instead we update the result of the previous evaluation.

To be precise, let a vector $\vec{\alpha} \in \mathbb{Z}_2^k$ represent the element of the support: $\vec{v}(\vec{\alpha}) = \sum_i \alpha_i \vec{z}_i$. Then, the nonzero elements of the state vector of the stabiliser state are given by $(-1)^{Q(\vec{\alpha})} i^{\ell\vec{\alpha}} |\vec{v}(\vec{\alpha}) + \vec{z}_0\rangle$. Thus, to construct the state vector, we iterate through $\mathbb{Z}_2^k$, setting the nonzero elements of the state vector.

Suppose that one iterates through $\mathbb{Z}_2^k$ in some order $\vec{\alpha}_0, \ldots \vec{\alpha}_{2^k-1} \in \mathbb{Z}_2^k$. Instead of calculating $\vec{v}(\vec{\alpha}_i) + \vec{z}_0$ for every i, from scratch, we can keep track of the 'current' $\vec{\beta}_i = \vec{v}(\vec{\alpha}_i) + \vec{z}_0$. Then, by linearity,

$$\vec{\beta}_{i+1} = \vec{\beta}_i + \vec{v}(\vec{\alpha}_i + \vec{\alpha}_{i+1}). \tag{8}$$

In particular, we iterate through Gray code [15] (i.e. $\vec{\alpha}_i$ is the i-th codeword of the Gray code). This has the useful property that $\vec{\alpha}_i + \vec{\alpha}_{i+1}$ always has Hamming weight 1, and therefore we only need to add a single basis vector to $\vec{\beta}_i$ to construct $\vec{\beta}_{i+1}$. Similarly, for $\vec{e}_i \in \mathbb{Z}_2^k$, the i-th standard basis vector,

$$\ell(\vec{\alpha} + \vec{e}_i) + \ell(\vec{\alpha}) = \ell(\vec{e}_i), \tag{9}$$

$$Q(\vec{\alpha} + \vec{e}_i) + Q(\vec{\alpha}) = \sum_{j<i} (Q_{ij} + Q_{ji})\vec{\alpha}_j. \tag{10}$$

Thus, if we keep track of the overall phase $(-1)^{Q(\vec{\alpha})} i^{\ell\vec{\alpha})}$, it takes time linear in n to perform each update as we iterate through the Gray code.

Algorithm 2: Convert a quadratic form triple to the amplitudes of a stabiliser state, i.e. $[\mathrm{S_Q}] \to [\mathrm{S_V}]$

1. Initialise an array of complex numbers $\Psi \in \mathbb{C}^N$, meant to be our state vector, to be all zeroes. Set $\vec{\beta}_0 = \vec{z}_0$ and $t_0 = 1$.

2. Let $\vec{\alpha}_i$ be the i-th codeword of the Gray code (on k bits). Iterate through the Gray code, starting with $i = 1$ (recall that the first codeword has $i = 0$). Supposing that $\vec{\alpha}_i + \vec{\alpha}_{i-1} = \vec{e}_j$ for some $j \in \mathbb{Z}_k$, use Equations (8)–(10) to find $\vec{\beta}_i$ and t_i. Set $\Psi[\beta_i] = t_i$.

3.3 $[\mathrm{S_V}]$: Verifying a vector is a stabiliser state

One can extend Algorithm 1 to verify that a given vector is a stabiliser state. First, we run Algorithm 1, and reject if any of the steps of the algorithm fail. Second, we must check whether the support of the state has size 2^k. Finally, we must also check that the remaining nonzero entries of the state vector are consistent with ℓ and Q. To do this, one runs Algorithm 2 ($[\mathrm{S_Q}] \to [\mathrm{S_V}]$), and checks that the resulting state vector is equal to the input.

3.4 $[\mathrm{S_P}] \to [\mathrm{S_Q}]$: Check matrix to a quadratic form triple

We will show that with minimal calculations, one can effectively read off the quadratic form triple data of a stabiliser state from its check matrix. We begin by establishing some required relations between a stabiliser state expressed as a quadratic form triple and the Pauli gates that stabilise it. Recall the expression of the amplitudes of a stabiliser state from Theorem 1. An arbitrary stabiliser state $|s\rangle$ can be expressed in the form:

$$|s\rangle \propto \sum_{\vec{z} \in V} (-1)^{Q(\vec{z})} i^{\ell(\vec{z})} |\vec{z} + \vec{z}_0\rangle \tag{5}$$

where $V \subset \mathbb{Z}_2^n$ is a vector subspace of dimension $k \leq n$, $\vec{z}_0 \in \mathbb{Z}_2^n$, and $Q, \ell : V \to \mathbb{Z}_2$.

Suppose $P = (-1)^c (-i)^{\vec{p}\cdot\vec{q}} X^{\vec{q}} Z^{\vec{p}}$ stabilises $|s\rangle$: $P|s\rangle = |s\rangle$. We can see immediately that $\vec{q} \in V$ or else P changes the support of $|s\rangle$. Moreover, by comparing the amplitudes of $|s\rangle$ and $P|s\rangle$, we find that

$$\ell(\vec{q}) = \vec{p} \cdot \vec{q} \tag{11}$$

$$\text{For all } \vec{z} \in V: \quad Q(\vec{z}) = c + Q(\vec{z} + \vec{q}) + \vec{p} \cdot (\vec{z} + \vec{q} + \vec{z}_0) + (\vec{p} \cdot \vec{q})\ell(\vec{z}), \text{ over } \mathbb{Z}_2. \tag{12}$$

Choose B to be a $\mathbb{Z}_2$-bilinear form such that $Q(\vec{z}) = B(\vec{z}, \vec{z})$; for example if $Q(\vec{z}) = \vec{z}^T \tilde{Q} \vec{z}$ for the upper triangular matrix $\tilde{Q}$, then we can take $B(\vec{z}, \vec{z}') = \vec{z}^T \tilde{Q}\, \vec{z}'$. We can then express Equation (12) as

$$[B(\vec{q}, \vec{z}) + {}^{\mathrm{T}}B(\vec{q}, \vec{z}) + (\vec{p} \cdot \vec{q})\ell(\vec{z}) + \vec{p} \cdot \vec{z}] + [B(\vec{q}, \vec{q}) + \vec{p} \cdot (\vec{q} + \vec{z}_0) + c] = 0 \tag{13}$$

where the first term is a linear function of $\vec{z} \in V$ and the second term is a constant. This can hold only if both terms are identically zero. Therefore:

$$c = \vec{p} \cdot (\vec{q} + \vec{z}_0) + B(\vec{q}, \vec{q}) \tag{14}$$

$$\vec{p} \cdot \vec{z} = B(\vec{q}, \vec{z}) + {}^{\mathrm{T}}B(\vec{q}, \vec{z}) + (\vec{p} \cdot \vec{q})\ell(\vec{z}) \tag{15}$$

where the latter equality is as linear functions from V to $\mathbb{Z}_2$. For each $\vec{q} \in V$, there is precisely one choice of c and one choice of $\vec{p}$, up to adding an element of $V^\perp$, such that $P|s\rangle = |s\rangle$.

Now, suppose we are given a check matrix of a stabiliser state. Without loss of generality, we may perform row reduction and assume that the first $2n$ columns are in reduced echelon form:

$$\begin{bmatrix} \vec{q}_1 & \vec{p}_1 & c_1 \\ \vdots & \vdots & \vdots \\ \vec{q}_k & \vec{p}_k & c_k \\ \vec{0} & \vec{\rho}_1 & \gamma_1 \\ \vdots & \vdots & \vdots \\ \vec{0} & \vec{\rho}_{n-k} & \gamma_{n-k} \end{bmatrix} \tag{16}$$

We can immediately take the $\vec{q}_i$ as a basis for V. The shift $\vec{z}_0$ is any solution to the system of $n-k$ equations $\vec{\rho}_i \cdot \vec{z}_0 = \gamma_i$. By Equation (11) and the fact that all the stabilising Pauli gates must have order 2, we can extract ℓ and define it on the basis for V via the equations $\ell(\vec{q}_i) = \vec{p}_i \cdot \vec{q}_i$.

We can extract the diagonal entries of $\tilde{Q}$ by rearranging Equation (14):

$$\tilde{Q}_{ii} = B(\vec{q}_i, \vec{q}_i) = c_i + \vec{p}_i \cdot (\vec{q}_i + \vec{z}_0). \tag{17}$$

We can extract the strictly upper triangular entries of $\tilde{Q}$, i.e. $\tilde{Q}_{ij}$ for $1 \leq i < j \leq k$, by rearranging Equation (15):

$$\tilde{Q}_{ij} = B(\vec{q}_i, \vec{q}_j) = B(\vec{q}_i, \vec{q}_j) + {}^{\mathrm{T}}B(\vec{q}_i, \vec{q}_j) = \vec{p}_i \cdot \vec{q}_j + (\vec{p}_i \cdot \vec{q}_i)(\vec{p}_j \cdot \vec{q}_j). \tag{18}$$

Algorithm 3: Convert the check matrix of a stabiliser state to a quadratic form triple, i.e. $[\mathrm{S_P}] \to [\mathrm{S_Q}]$

1. Row reduce the first two columns whilst updating the sign bits consistently to ensure that the resulting check matrix represents the same stabiliser state up to phase. Label the result as in Equation (16).
2. Take the $\vec{q}_i$ as a basis for V and a solution to $\vec{\rho}_i \cdot \vec{z}_0 = \gamma_i$ for the shift $\vec{z}_0$.
3. Compute $\ell(\vec{q}_i) = \vec{q}_i \cdot \vec{p}_i$ and store these as a vector representing ℓ in the ordered basis $\{\vec{q}_1, \ldots, \vec{q}_k\}$ of V.
4. Compute a matrix representing Q relative to the same ordered basis using Equations (17) and (18).

3.5 $[\mathrm{S_Q}] \to [\mathrm{S_P}]$: Quadratic form triple to a check matrix

We have shown above that it is relatively straightforward to read off the quadratic form triple data from a check matrix. Here, we reverse this process. Suppose we are given a basis $\{\vec{v}_1, \ldots, \vec{v}_k\}$ for $V \subset \mathbb{Z}_2^n$, a shift vector $\vec{z}_0 \in \mathbb{Z}_2^n$, a $k \times k$ upper triangular matrix $\tilde{Q}$ over $\mathbb{Z}_2$, and a vector $\ell \in \mathbb{Z}_2^k$.

As in Equation (16), we can generate the $\vec{q}_i$ by row reducing the basis $\vec{v}_i$; we store the resulting change of basis matrix of $\mathbb{Z}_2^k$ and use it to update $\tilde{Q}, \ell$. The updated $\tilde{Q}$ may no longer be upper triangular but can made so by taking $\tilde{Q} + \tilde{Q}_l + (\tilde{Q}_l)^{\mathrm{T}}$ where $\tilde{Q}_l$ is the strictly lower triangular part of $\tilde{Q}$.

Applying Equation (15) to $(\vec{p},\vec{q}) = (\vec{\rho}_i, \vec{0})$, we see that the $\vec{\rho}_i$ can be taken to be a basis of the null space of the matrix whose rows are $\vec{q}_i$, i.e. of $V^\perp$. We can then extract γ_i using $\gamma_i = \vec{\rho}_i \cdot \vec{z}_0$.

Using Equations (11) and (15) we can use our given $\tilde{Q}$ and ℓ to give linear systems for each $\vec{p}_i$:

$$\vec{p}_i \cdot \vec{q}_j = \tilde{Q}_{ij} + \ell(\vec{q}_i)\ell(\vec{q}_j). \tag{19}$$

These determine each $\vec{p}_i$ in the basis $\{\vec{q}_1, \ldots, \vec{q}_k\}$ up to an element of $V^\perp$.

Finally, we compute the c_i using Equation (14):

$$c_i = \vec{p}_i \cdot (\vec{q}_i + \vec{z}_0) + \tilde{Q}_{ii}. \tag{20}$$

Algorithm 4: Convert the quadratic form triple of a stabiliser state to a check matrix, i.e. $[\mathsf{S}_\mathsf{Q}] \to [\mathsf{S}_\mathsf{P}]$

1. Row reduce the basis $\{\vec{v}_1, \ldots, \vec{v}_k\}$ of V to find $\{\vec{q}_1, \ldots, \vec{q}_k\}$ and a basis $\{\vec{\rho}_1, \ldots, \vec{\rho}_{n-k}\}$ of $V^\perp$.
2. Update $\tilde{Q}, \ell$ using the change of basis matrix of the previous Step. Convert $\tilde{Q}$ to an upper triangular matrix representing the same quadratic form.
3. Compute $\gamma_i = \vec{\rho}_i \cdot \vec{z}_0$.
4. Find each of the $\vec{p}_i$ by solving the underdetermined linear systems of Equation (19).
5. Compute $c_i = \vec{p}_i \cdot (\vec{q}_i + \vec{z}_0) + \tilde{Q}_{ii}$.

---

4 Clifford gate algorithms

In this section, we describe a highly efficient algorithm for extracting from a Clifford gate C, given by its matrix elements, its conjugate tuple: the $2n$ Pauli gates $U_i = CZ_iC^*, V_i = CX_iC^*$. It can also be used to check if a given unitary matrix U is a Clifford gate. It works by building upon our above algorithm for extracting generators for the stabiliser group of a stabiliser vector.

4.1 $[\mathsf{C}_\mathsf{U}] \to [\mathsf{C}_\mathsf{T}]$: Matrix entries to a conjugate tuple

The conjugate tuple $U_i = CZ_iC^*, V_i = CX_iC^*$, expressed as $2n$ elements of $\mathbb{Z}_2^{n+1}$, of a Clifford gate can be found in two steps.

We begin by extracting the U_i by examining the stabiliser group of the first column of C. To justify this calculation, we require some lemmata.

Lemma 3. *For a Clifford gate C, the stabiliser group of $C|\vec{0}\rangle$ is generated by $U_i = CZ_iC^*$.*

Proof. $U_i(C|\vec{0}\rangle) = CZ_i|\vec{0}\rangle = C|\vec{0}\rangle$. The U_i are independent since the Z_i are. □

We can rapidly extract a set of generators P_i for this stabiliser subgroup using the techniques of the previous section. Since P_i and U_i generate the same stabiliser subgroup, we have that $P_i = U^{\vec{\rho}_i}$ for some $\vec{\rho}_i \in \mathbb{Z}_2^n$. Thus,

$$P_i C|\vec{z}\rangle = U^{\vec{\rho}_i} C|\vec{z}\rangle = CZ^{\vec{\rho}_i}|\vec{z}\rangle = (-1)^{\vec{\rho}_i \cdot \vec{z}} C|\vec{z}\rangle. \tag{21}$$

We can thus extract the vectors $\vec{\rho}_i$ by computing $P_i\, C\, |\vec{z}\rangle$ for $\vec{z}$ having Hamming weight 1.

Note that Pauli-vector multiplication can be done more quickly than by using standard matrix-vector multiplication. For a vector $|v\rangle = \sum_{\vec{z}\in\mathbb{Z}_2^n} v_{\vec{z}}\,|\vec{z}\rangle$, and a Pauli gate $P = (-1)^c(-i)^{\vec{p}\cdot\vec{q}}Z^{\vec{p}}X^{\vec{q}}$, we have

$$P\,|v\rangle = (-1)^{c+\vec{p}\cdot\vec{q}}(-i)^{\vec{p}\cdot\vec{q}}\sum_{\vec{z}\in\mathbb{Z}_2^n}(-1)^{\vec{p}\cdot\vec{z}}v_{\vec{z}+\vec{q}}\,|\vec{z}\rangle\,.$$

We can compute $P\,|v\rangle$ by instead reindexing the vector $|v\rangle$ and introducing both a global phase change and, for each index $\vec{z}\in\mathbb{Z}_2^n$, the sign change determined by $\vec{p}\cdot\vec{z}$. This requires $O(Nn)$ time rather than the $O(N^2)$ time required for standard matrix-vector multiplication.

We know that the $\vec{\rho}_i$ are linearly independent since the P_i are independent. We may thus invert the matrix whose columns are $\vec{\rho}_i$; we call the rows of this inverse $\vec{\mu}_i$. It is a consequence of the following lemma that $U_i = P_i^{\vec{\mu}_i}$.

Lemma 4. *Suppose that M, U_i are matrices of size $N\times N$, with M unitary, such that for all $\vec{z}\in\mathbb{Z}_2^n$*

$$U_iM\,|\vec{z}\rangle = (-1)^{z_i}M\,|\vec{z}\rangle\,. \tag{22}$$

Then M conjugates the Z_i to U_i; that is, $MZ_iM^* = U_i$.

Proof. Multiplying (22) by $\langle\vec{z}_2|\,M^*$ we see that $\langle\vec{z}_2|\,M^*U_iM\,|\vec{z}_1\rangle = (-1)^{z_i}\delta_{\vec{z}_1,\vec{z}_2}$ and thus $M^*U_iM = Z_i$. □

Having found the U_i, we must now find the V_i. Our strategy will be to find Pauli gates W_i of order 2 such that U_i and W_j anticommute if and only if $i=j$. We will then correct these to get the V_i by querying as few entries of C as possible.

Constructing the W_i is straightforward. If $U_i \propto X^{\vec{q}_i}Z^{\vec{p}_i}$, define U to be the $n\times 2n$ matrix over $\mathbb{Z}_2$ whose i-th row is $(\vec{q}_i,\vec{p}_i)$. As these rows are linearly independent, $U^+ = U^T(UU^T)^{-1}$ is a right inverse: $UU^+ = \mathbb{I}$. We may thus take W_i to be $(-i)^{\vec{\alpha}_i\cdot\vec{\beta}_i}X^{\vec{\beta}_i}Z^{\vec{\alpha}_i}$ where $(\vec{\alpha}_i,\vec{\beta}_i)$ is the i-th column of U^+.

Each V_i differs from W_i by a product of the U_j as a result of the following lemma.

Lemma 5. *Suppose the n Pauli gates U_i generate a stabiliser group, the Pauli gates V_i, W_i are of order 2, and that V_i commutes (or anticommutes) with U_i if and only if W_i does. Then $V_i = (-1)^{c_i}(-i)^{d_i}W_iU^{\vec{v}_i}$ for some $\vec{v}_i\in\mathbb{Z}_2^n$ and $c_i,d_i\in\mathbb{Z}_2$.*

Proof. Define the Pauli gates $T_i = i^{d_i}W_iV_i$ where d_i is 0 if W_i and V_i commute and 1 if they anticommute. They commute with every U_j: $T_iU_j = i^{d_i}W_iV_iU_j = i^{d_i}U_jW_iV_i = U_jT_i$. Further, the T_i are of order 2. Therefore, by Definition 2, for each i, either T_i or $-T_i$ is a member of the stabiliser group generated by U_i. □

Therefore, our final task is to find the V_i by determining the appropriate $\vec{v}_i\in\mathbb{Z}_2^n$ and $c_i,d_i\in\mathbb{Z}_2$.

$$\begin{aligned}
C\,|\vec{z}+\vec{e}_i\rangle &= CX_i\,|\vec{z}\rangle && (23)\\
&= V_iC\,|\vec{z}\rangle && (24)\\
&= (-1)^{c_i}(-i)^{d_i}W_iU^{\vec{v}_i}C\,|\vec{z}\rangle && (25)\\
&= (-1)^{c_i}(-i)^{d_i}W_iCZ^{\vec{v}_i}\,|\vec{z}\rangle && (26)\\
&= (-1)^{\vec{v}_i\cdot\vec{z}+c_i}(-i)^{d_i}W_i\;C\,|\vec{z}\rangle && (27)
\end{aligned}$$

where $\vec{e}_i$ is the i-th unit vector of $\mathbb{Z}_2^n$.

By comparing the relative phases of the columns of C corresponding to $\vec{z}$ and $\vec{z}+\vec{e}_i$, we gain information about the required variables $\vec{v}_i, c_i, d_i$. Choosing $\vec{z}=\vec{0}$, we determine the c_i, d_i. Choosing $\vec{z}$ to be of Hamming weight 1, we determine the components of the $\vec{v}_i$.

To determine the relative phases of two columns, we need only check two nonzero entries: one from each column. To avoid potentially having to perform a high number of checks for columns that contain many zeros, we can first find an index in the support of each column. We use the fact that the support of $C|\vec{z}\rangle$ is the same as that of $W^{\vec{z}}C|\vec{0}\rangle$, as each U_i either stabilises or antistabilises every column. We find the stabiliser state $C|\vec{0}\rangle$ in $[\mathrm{S_Q}]$ form, which then allows us to use fast Pauli-vector multiplication to apply $W^{\vec{z}}$ and find an index in the support of $C|\vec{z}\rangle$. By subsequently applying W_i, we can find an index in the support of $C|\vec{z}+\vec{e}_i\rangle$.

Algorithm 5: Extract the conjugate tuple of a Clifford gate C, i.e. $[\mathrm{C_U}] \to [\mathrm{C_T}]$

1. Compose Algorithms 1 and 4 to extract generators P_i for the stabiliser group of $C|\vec{0}\rangle$.
2. For each $\vec{z}$ with Hamming weight 1, find a nonzero entry of $C|\vec{z}\rangle$. For each i, use these nonzero entries to determine $\vec{\rho}_i$ satisfying $P_iC|\vec{z}\rangle = (-1)^{\vec{\rho}_i\cdot\vec{z}}|\vec{z}\rangle$. Note we do not have to verify that this equation holds; it is sufficient to check the action of P_i^* on the nonzero entries of the columns to set $\vec{\rho}_i$.
3. Invert the $n\times n$ matrix over $\mathbb{Z}_2$ whose columns are $\vec{\rho}_i$ to find one with rows $\vec{\mu}_i$. Conclude that $U_i = P^{\vec{\mu}_i}$.
4. Compute the bitstrings specifying the W_i (up to sign) by taking the pseudoinverse of the matrix whose rows are the bitstrings specifying the U_i.
5. Correct the W_i to V_i using Equations (23) and (27) by comparing two nonzero entries of the $\vec{z}$-th and $\vec{z}+\vec{e}_i$-th columns of C for $\vec{z}$ being $\vec{0}$ or having Hamming weight 1.

4.2 $[\mathrm{C_U}]$: Verifying a unitary matrix is a Clifford gate

One can use Algorithm 5 to verify that a given matrix M is a Clifford gate by accepting only at the point that a conversion is made successfully. Since we are not assuming that M is a Clifford gate, we must add a few additional checks.

First, when we extract the P_i, we may not assume that $M|0\rangle$ is a stabiliser state; we explain how to verify this in Subsection 3.3. We find the U_i and V_i as in Algorithm 5; the U_i stabilise $M|0\rangle$ and $U_iV_j = (-1)^{\delta_{ij}}V_jU_i$ by construction. Note that by Lemma 5, the V_i will pairwise commute. We then make use of the following lemmata:

Lemma 6. *Let M be an $N\times N$ unitary matrix. Suppose there exist Pauli gates $U_i, V_i \in \mathcal{P}_n$ for $i\in[n]$ such that:*

i) $MX_iM^* = V_i$,

ii) $U_iV_j = -V_jU_i$ *if and only if* $i=j$,

iii) $U_iM|0\rangle = M|0\rangle$.

Then $MZ_iM^* = U_i$ and, hence, M is a Clifford gate.

Proof. $U_i M |\vec{z}\rangle \overset{i)}{=} U_i V^{\vec{z}} M |\vec{0}\rangle \overset{ii)}{=} (-1)^{z_i} V^{\vec{z}} U_i M |\vec{0}\rangle \overset{iii)}{=} (-1)^{z_i} V^{\vec{z}} M |\vec{0}\rangle \overset{i)}{=} (-1)^{z_i} M |\vec{z}\rangle$. The result then follows from Lemma 4. □

Lemma 7. *Suppose that M, V_i are matrices of size $N \times N$, with M unitary, such that for all $\vec{z} \in \mathbb{Z}_2^n$*

$$V_i M |\vec{z}\rangle = M |\vec{z} + \vec{e}_i\rangle . \tag{28}$$

Then M conjugates the X_i to V_i; that is, $M X_i M^* = V_i$.

Proof. Similar to Lemma 4. □

By Lemma 6, it is sufficient to check that $M X_i M^* = V_i$. By Lemma 7, it is sufficient to check that $M |\vec{z}\rangle = V^{\vec{z}} M |0\rangle$. To do this, we iterate through the Gray code [15]. Let $\vec{\alpha}_i$ be the i-th codeword. We check that $M |\vec{\alpha}_i\rangle = V^{\vec{\alpha}_i + \vec{\alpha}_{i-1}} M |\vec{\alpha}_{i-1}\rangle$. The Gray code has the useful property that any two consecutive codewords differ only at a single bit, so that we only need to multiply our previously checked column by a single V_i at each step.

4.3 $[\mathsf{C_T}] \to [\mathsf{C_U}]$: Conjugate tuple to a matrix

To rapidly construct the matrix of a Clifford gate (up to phase) given its conjugate tuple, we employ Theorem 3.9 of [43].

Theorem 2. *The unitary C that carries each (Z_i, X_i) to (U_i, V_i) under conjugation is given by:*

$$C |\vec{z}\rangle = V^{\vec{z}} |u_0\rangle \tag{29}$$

where $\vec{z} \in \mathbb{Z}_2^n$ and $|u_0\rangle = C |0\rangle$ is a simultaneous eigenvector of the $U_1, \ldots, U_n$ with eigenvalue 1.

The simultaneous eigenvector of the $U_1, \ldots, U_n$ with eigenvalue 1 can be quickly computed by combining Algorithms 3 and 2 above, giving the first column of the matrix. Then, to find the remaining columns, we iterate through the Gray code, using Theorem 2. Every column will differ from the previous one by the application of a single V_i that corresponds to the bit that was just flipped. Here, we again make use of the fast method for Pauli-vector multiplication.

5 Complexity analyses

We explicitly consider the complexity of Algorithms 1 through 5 as well as the other interconversion and verification algorithms based on them. Recall that n denotes the number of qubits and $N = 2^n$.

5.1 Stabiliser states

Algorithm 1: $[\mathbf{S_V}] \to [\mathbf{S_Q}]$

1. Finding the nonzero amplitudes takes $O(N)$ time; applying a bitwise `XOR` between $\vec{z}_0$ and each binary label in the support then takes $O(2^k)$ operations, where k is the dimension of the support. We are modelling bitwise `XOR` as a constant-time operation as it is fully parallelisable.
2. Extracting ℓ takes time $O(n)$.
3. Extracting Q takes time $O(n^2)$.

The dominant step of the algorithm is Step 1 yielding a runtime of $O(N)$.

Algorithm 2: $[\mathbf{S_Q}] \to [\mathbf{S_V}]$

1. Initialising an empty state vector takes time $O(N)$.
2. As noted in the main text, at each step of the Gray code, it takes time $O(n)$ to update t_i and $\vec{\beta}_i$. Thus, this step runs in time $O(2^k n)$.

In the case that k is very small, the dominant step of the algorithm is Step 1. However, if $k = \Omega(n)$, Step 2 will dominate and Algorithm 2 runs in time $O(Nn)$.

Algorithm 3: $[\mathbf{S_P}] \to [\mathbf{S_Q}]$

1. Gaussian elimination takes time $O(n^3)$.
2. Finding a solution to the row reduced equation takes time $O(n)$.
3. Computing ℓ takes time $O(k)$, where k is the dimension of the affine subspace.
4. Computing Q takes time $O(k^2)$.

Thus, Algorithm 3 runs in time $O(n^3)$.

Algorithm 4: $[\mathbf{S_Q}] \to [\mathbf{S_P}]$

1. Row reduction takes time $O(nk^2)$, where k is the dimension of V. Finding a null basis for the null space then takes time $O((n-k)k)$.
2. Computing Q and ℓ can be done in parallel with Step 1, altering after each row reduction. Each update takes constant time (as it corresponds to swapping a row or adding two rows together) and thus this does not affect the complexity of Step 1.
3. This takes time $O((n-k)n)$.
4. This takes time $O(kn)$.

In general, when $k = \Omega(n)$, Algorithm 4 runs in time $O(n^3)$.

$[\mathbf{S_V}] \to [\mathbf{S_P}]$:
Our algorithm is the composition of Algorithms 1 and 4, and thus requires time $O(N)$.

$[\mathbf{S_P}] \to [\mathbf{S_V}]$:
Our algorithm is the composition of Algorithms 3 and 2, and thus requires time $O(Nn)$.

Verifying $[\mathbf{S_V}]$:
Our algorithm is the composition of Algorithm 1 and Algorithm 2, followed by a consistency check between the original and reconsituted state vectors, and thus requires time $O(N)$. Note that to verify a vector, one must look at every element at least once, giving an $\Omega(N)$ lower bound.

5.2 Clifford gates

Algorithm 5: $[\mathbf{C_U}] \to [\mathbf{C_T}]$

1. The conversion $[\mathrm{S_V}]$ to $[\mathrm{S_P}]$ takes $O(N)$ time as discussed above.
2. In the worst case, we search for time $O(N)$ to find a nonzero element of a given column. Once we have found one for each column of Hamming weight 1, taking time $O(Nn)$, we can extract all the $\vec{\rho}_i$ vectors in time $O(n^2)$.
3. It takes time $O(n^3)$ to find the pseudoinverse of the U. In fact, the output of our $[\mathrm{S_V}]$ to $[\mathrm{S_P}]$ conversion gave us the row-reduced version of U; by keeping track of the row operations when constructing the U_i from the P_i, we can construct the pseudoinverse simultaneously. We then require time $O(n^2)$ to find each of the W_i.
4. Comparing the relative phase of the columns corresponding to Hamming weight 1 and 2 takes time $O(n^3)$ as there are n^2 such columns and finding a nonzero element takes time $O(n)$. Then, finding the V_i involves multiplying each W_i by $O(n)$ Pauli gates, where each Pauli multiplication takes time $O(n)$. This step thus also runs in time $O(n^3)$.

The algorithm is dominated by Step 2 and thus takes time $O(Nn)$.

$[\mathbf{C_T}] \to [\mathbf{C_U}]$:
The $[\mathrm{S_P}] \to [\mathrm{S_V}]$ conversion to find the first column requires time $O(Nn)$. Computing the remaining columns using the Gray code takes time $O(N^2 n)$, giving a total time of $O(N^2 n)$.

Verifying $[\mathbf{C_U}]$:
Compared to Algorithm 5, we must also verify that the first column of the unitary matrix is a stabiliser state and check the consistency of the remaining columns (using the Gray code). Verifying the first column takes time $O(N)$. At every step of the Gray code, the two consecutive codewords differ at a single bit, so we multiply our previously checked column by a single Pauli gate which takes time $O(Nn)$. We check every column, so the total time is $O(N^2 n)$.

Note that to verify a matrix, one must look at every element at least once, giving an $\Omega(N^2)$ lower bound.

6 Acknowledgments

ND acknowledges support from the Canada Research Chair program, NSERC Discovery Grant RGPIN-2022-03103, the NSERC-European Commission project FoQaCiA, and the Faculty of Science of Simon Fraser University. WS acknowledges support from the EPSRC and Hitachi. We thank Matthew Amy and Shane Mansfield for helpful comments.

References

[1] Scott Aaronson and Alex Arkhipov. "The computational complexity of linear optics". In: *STOC '11*. 2011, pp. 333–342. DOI: 10.1145/1993636.1993682.

[2] Scott Aaronson and Daniel Gottesman. "Improved simulation of stabilizer circuits". In: *Physical Review A* 70.5 (2004), p. 052328. DOI: 10.1103/physreva.70.052328.

[3] Matthew Amy. Personal communication. 2024.

[4] Matthew Amy, Owen Bennett-Gibbs, and Neil J. Ross. “Symbolic Synthesis of Clifford Circuits and Beyond”. In: *EPTCS* 394 (2023), pp. 343–362. DOI: 10.4204/EPTCS.394.17. arXiv: 2204.14205 [quant-ph].

[5] Matthew Amy, Dmitri Maslov, and Michele Mosca. “Polynomial-time T-depth optimization of Clifford+ T circuits via matroid partitioning”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.10 (2014), pp. 1476–1489. DOI: 10.1109/tcad.2014.2341953.

[6] Simon Anders and Hans J Briegel. “Fast simulation of stabilizer circuits using a graph-state representation”. In: *Physical Review A* 73.2 (2006), p. 022334. DOI: 10.1103/physreva.73.022334.

[7] Koenraad MR Audenaert and Martin B Plenio. “Entanglement on mixed stabilizer states: normal forms and reduction procedures”. In: *New Journal of Physics* 7.1 (2005), p. 170. DOI: 10.1088/1367-2630/7/1/170.

[8] Niel de Beaudrap and Steven Herbert. “Fast stabiliser simulation with quadratic form expansions”. In: *Quantum* 6 (2022), p. 803. DOI: 10.22331/q-2022-09-15-803.

[9] John S Bell. “On the Einstein-Podolsky-Rosen paradox”. In: *Physics Physique Fizika* 1.3 (1964), p. 195. DOI: 10.1103/PhysicsPhysiqueFizika.1.195.

[10] Charles H Bennett and Gilles Brassard. “Quantum cryptography: Public key distribution and coin tossing”. In: *IEEE ICSSP* (1984). DOI: 10.1016/j.tcs.2014.05.025.

[11] Charles H Bennett and Stephen J Wiesner. “Communication via one-and two-particle operators on Einstein-Podolsky-Rosen states”. In: *Physical Review Letters* 69.20 (1992), p. 2881. DOI: 10.1103/PhysRevLett.69.2881.

[12] Charles H Bennett et al. “Concentrating partial entanglement by local operations”. In: *Physical Review A* 53.4 (1996), p. 2046. DOI: 10.1103/PhysRevA.53.2046.

[13] Charles H Bennett et al. “Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels”. In: *Physical Review Letters* 70.13 (1993), p. 1895. DOI: 10.1103/PhysRevLett.70.1895.

[14] Ethan Bernstein and Umesh Vazirani. “Quantum complexity theory”. In: *STOC ’25*. 1993, pp. 11–20. DOI: 10.1145/167088.167097.

[15] James R. Bitner, Gideon Ehrlich, and Edward M. Reingold. “Efficient generation of the binary reflected Gray code and its applications”. In: *Commun. ACM* 19.9 (Sept. 1976), pp. 517–521. ISSN: 0001-0782. DOI: 10.1145/360336.360343.

[16] Sergey Bravyi and Jeongwan Haah. “Magic-state distillation with low overhead”. In: *Physical Review A* 86.5 (2012), p. 052329. DOI: 10.1103/physreva.86.052329.

[17] Sergey Bravyi and Alexei Kitaev. “Universal quantum computation with ideal Clifford gates and noisy ancillas”. In: *Physical Review A* 71.2 (2005), p. 022316. DOI: 10.1103/PhysRevA.71.022316.

[18] Sergey Bravyi, Graeme Smith, and John A Smolin. “Trading classical and quantum computational resources”. In: *Physical Review X* 6.2 (2016), p. 021043. DOI: 10.1103/PhysRevX.6.021043.

[19] Sergey Bravyi et al. “Clifford circuit optimization with templates and symbolic Pauli gates”. In: *Quantum* 5 (2021), p. 580. DOI: 10.22331/q-2021-11-16-580.

[20] Sergey Bravyi et al. “Simulation of quantum circuits by low-rank stabilizer decompositions”. In: *Quantum* 3 (2019), p. 181. DOI: 10.22331/q-2019-09-02-181.

[21] Timothée Goubault de Brugière, Simon Martiel, and Christophe Vuillot. "A graph-state based synthesis framework for Clifford isometries". In: *arXiv preprint* (2022). DOI: 10.48550/arXiv.2212.06928. arXiv: 2212.06928 [quant-ph].

[22] Alexander Cowtan, Will Simmons, and Ross Duncan. *A Generic Compilation Strategy for the Unitary Coupled Cluster Ansatz*. 2020. DOI: 10.48550/arXiv.2007.10515. arXiv: 2007.10515 [quant-ph].

[23] Alexander M. Dalzell et al. "Quantum algorithms: A survey of applications and end-to-end complexities". In: *arXiv preprint* (2023). DOI: 10.48550/arXiv.2310.03011. arXiv: 2310.03011 [quant-ph].

[24] Ninnat Dangniam, Yun-Guang Han, and Huangjun Zhu. "Optimal verification of stabilizer states". In: *Physical Review Research* 2.4 (Dec. 2020). ISSN: 2643-1564. DOI: 10.1103/physrevresearch.2.043323.

[25] Jeroen Dehaene and Bart De Moor. "Clifford group, stabilizer states, and linear and quadratic operations over GF(2)". In: *Physical Review A* 68.4 (2003), p. 042318. DOI: 10.1103/PhysRevA.68.042318.

[26] David Deutsch and Richard Jozsa. "Rapid solution of problems by quantum computation". In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (1992), pp. 553–558. DOI: 10.1098/rspa.1992.0167.

[27] Bryan Eastin and Emanuel Knill. "Restrictions on transversal encoded quantum gate sets". In: *Physical Review Letters* 102.11 (2009), p. 110502. DOI: 10.1103/physrevlett.102.110502.

[28] Craig Gidney. "Stim: a fast stabilizer circuit simulator". In: *Quantum* 5 (July 2021), p. 497. ISSN: 2521-327X. DOI: 10.22331/q-2021-07-06-497.

[29] Daniel Gottesman. "An introduction to quantum error correction and fault-tolerant quantum computation". In: *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*. Vol. 68. 2010, pp. 13–58. DOI: 10.1090/psapm/068/2762145.

[30] Daniel Gottesman. "Stabilizer Codes and Quantum Error Correction". PhD thesis. Pasadena, CA, USA: California Institute of Technology, 1997. DOI: 10.48550/arXiv.quant-ph/9705052.

[31] Daniel Gottesman. "The Heisenberg representation of quantum computers". In: *22nd International Colloquium on Group Theoretical Methods in Physics*. July 1998, pp. 32–43. arXiv: quant-ph/9807006.

[32] Daniel M Greenberger et al. "Bell's theorem without inequalities". In: *American Journal of Physics* 58.12 (1990), pp. 1131–1143. DOI: 10.1119/1.16243.

[33] Hsin-Yuan Huang, Richard Kueng, and John Preskill. "Predicting many properties of a quantum system from very few measurements". In: *Nature Physics* 16.10 (2020), pp. 1050–1057. DOI: 10.1038/s41567-020-0932-7.

[34] Richard Jozsa and Maarten Van den Nest. "Classical simulation complexity of extended Clifford circuits". In: *Quantum Information and Computation* 14.7, 8 (May 2014), pp. 633–648. ISSN: 1533-7146. DOI: 10.26421/qic14.7-8-7.

[35] Vadym Kliuchnikov, Michael Beverland, and Adam Paetznick. "Stabilizer circuit verification". In: *arXiv preprint* (2023). DOI: 10.48550/arXiv.2309.08676. arXiv: 2309.08676 [quant-ph].

[36] Vadym Kliuchnikov and Sebastian Schönnenbeck. "Stabilizer operators and Barnes-Wall lattices". In: *arXiv preprint* (2024). DOI: 10.48550/arXiv.2404.17677. arXiv: 2404.17677 [quant-ph].

[37] Emanuel Knill et al. "Randomized benchmarking of quantum gates". In: *Physical Review A* 77.1 (2008), p. 012307. DOI: 10.1103/PhysRevA.77.012307.

[38] Shane Mansfield. Personal communication. 2023.

[39] Yunseong Nam et al. "Automated optimization of large quantum circuits with continuous parameters". In: *npj Quantum Information* 4.1 (May 2018). ISSN: 2056-6387. DOI: 10.1038/s41534-018-0072-4.

[40] Xiaotong Ni, Oliver Buerschaper, and Maarten Van den Nest. "A non-commuting stabilizer formalism". In: *Journal of Mathematical Physics* 56.5 (2015). DOI: 10.1063/1.4920923.

[41] Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing*. 2023. DOI: 10.5281/zenodo.2573505.

[42] Robert Raussendorf and Hans J Briegel. "Quantum computing via measurements only". In: *arXiv preprint quant-ph/0010033* (2000). DOI: 10.48550/arXiv.quant-ph/0010033.

[43] Nadish de Silva. "Efficient quantum gate teleportation in higher dimensions". In: *Proceedings of the Royal Society A* 477.2251 (2021), p. 20200865. DOI: 10.1098/rspa.2020.0865.

[44] Daniel R Simon. "On the power of quantum computation". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1474–1483. DOI: 10.1137/S0097539796298637.

[45] Kaitlin N Smith et al. "Clifford-based circuit cutting for quantum simulation". In: *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 2023, pp. 1–13. DOI: 10.1145/3579371.3589352.

[46] StackExchange. *How do I check if a gate represented by unitary U is a Clifford Gate?* https://quantumcomputing.stackexchange.com/questions/13157/how-do-i-check-if-a-gate-represented-by-unitary-u-is-a-clifford-gate. 2020.

[47] StackExchange. *How to verify whether a state is a stabilizer state?* https://quantumcomputing.stackexchange.com/questions/3861/how-to-verify-whether-a-state-is-a-stabilizer-state. 2018.

[48] G.I. Struchalin et al. "Experimental Estimation of Quantum State Properties from Classical Shadows". In: *PRX Quantum* 2 (1 Jan. 2021), p. 010307. DOI: 10.1103/PRXQuantum.2.010307.

[49] Quantum AI team and collaborators. *qsim*. Version v0.10.3-dev+20211001. Oct. 2021. DOI: 10.5281/zenodo.5544365.

[50] Stephen Wiesner. "Conjugate coding". In: *ACM Sigact News* 15.1 (1983), pp. 78–88. DOI: 10.1145/1008908.1008920.

[51] Pei Zeng, You Zhou, and Zhenhuan Liu. "Quantum gate verification and its application in property testing". In: *Phys. Rev. Res.* 2 (2 June 2020), p. 023306. DOI: 10.1103/PhysRevResearch.2.023306.

[52] Fang Zhang and Jianxin Chen. *Optimizing T gates in Clifford+T circuit as $\pi/4$ rotations around Paulis*. 2019. DOI: 10.48550/arXiv.1903.12456. arXiv: 1903.12456 [quant-ph].

A Complexity comparisons to existing implementations

In this appendix, we compare our algorithms to existing implementations in popular libraries: IBM's `Qiskit` [41] package and Google's `Stim` package due to Gidney [28]. We note that worst-case asymptotic complexity is a very coarse metric that does not fully capture the scope of an algorithm's advantage; it ignores differences in prefactors and lower order terms in their runtime costs. Establishing a difference is sufficient, however, to guarantee that one algorithm will outperform another for all sufficiently large inputs.

$[\mathbf{S_V}] \to$ a succinct description of a stabiliser state and verifying $[\mathbf{S_V}]$:
To our knowledge, `Qiskit` does not implement a method for converting the amplitudes of a stabiliser state into a succinct description of that state.

`Stim` converts the amplitudes of a stabiliser state into a succinct representation in the form of a tableau using the `from_state_vector` function in the `Tableau` class. A key subroutine is the `stabiliser_state_vector_to_circuit` function in the file `circuit_vs_amplitudes.cc`, which finds a Clifford circuit that transforms $|\vec{0}\rangle$ into the given stabliser state. On input $|\psi\rangle \in \mathbb{C}^N$ the algorithm runs as follows.

1. By applying a series of X gates, move the element with the largest amplitude to the first entry of the state vector.

2. Find a nonzero element in the vector which is not the first entry; suppose it has index $\vec{k} \in \{0,1\}^n$. If no such entry exists, terminate the algorithm.

3. Let $i \in [n]$ be the smallest index such that $\vec{k}_i = 1$. Apply a series of CNOT gates between qubit i and qubit j, for every $j \neq i$ such that $\vec{k}_j = 1$. Finally, apply a single-qubit Clifford gate to qubit i. If $|\psi\rangle$ was a stabiliser state, this will halve the size of the support of the state vector. If this was not the case then output that the input was not a stabiliser state.

4. Return to Step 2.

Since the support of the state vector is halved after every iteration, the algorithm will run Step 3 at most k times, where k is the dimension of the support of $|\psi\rangle$. Each iteration of Step 3 involves applying at most $\Omega(n)$ CNOT gates in the worse case. Each application of a CNOT gate to $|\psi\rangle$ requires $\Omega(N)$ steps. Thus, in the worst case, `Stim`'s algorithm runs in time $\Omega(Nn^2)$ and produces a circuit of size $\Omega(n^2)$. Note that it also verifies whether the input state is a stabiliser state.

$[\mathbf{S_P}] \to [\mathbf{S_V}]$:
We do not believe this is implemented directly in `Qiskit`. This is implemented in `Stim` by the function `to_state_vector` of the `Tableau` class. The conversion works by:

1. Generate a random vector in $\mathbb{C}^N$. It will have overlap with all stabiliser states with probability 1 in the ideal setting and probability very close to one in practice.

2. For every Pauli gate P_i in the check matrix, apply the projector $(P_i + \mathbb{I})/2$ to the current state vector.

3. Renormalise the state.

Note that applying a projector $(\mathbb{I} + P)/2$ takes times $\Omega(Nn)$. This is done for each Pauli gate, giving a total runtime of $\Omega(Nn^2)$. This algorithm is probabilistic and is not

guaranteed to succeed on every run.

Other stabiliser state conversions:
We are not aware of any implementations of algorithms that convert to or from $[\mathrm{S_Q}]$ or from $[\mathrm{S_P}]$ to $[\mathrm{S_Q}]$ other than ours.

$[\mathbf{C_U}] \to [\mathbf{C_T}]$ and verifying $[\mathbf{C_U}]$:
Both `Qiskit` and `Stim` implement $[\mathrm{C_U}] \to [\mathrm{C_T}]$ and additionally verify that the input matrix is a Clifford gate.

In `Qiskit`, the conversion is the `from_matrix` method of the `Clifford` class. It is implemented via the brute force approach: conjugating each basic Pauli gate via matrix multiplication, which takes time $\Omega(N^3 n)$.

In `Stim`, the conversion is the `from_unitary_matrix` method of the `Tableau` class. Its implementation is more complex; on input matrix C the algorithm runs as:

1. Find a Clifford gate U (and its circuit representation) such that $U|\vec{0}\rangle = C|\vec{0}\rangle$ (i.e. run the conversion described above on the first column of C).
2. Calculate $M = U^\dagger C$. If C is a Clifford gate, then M will be a Clifford gate that permutes the computational basis vectors with phases.
3. Find a circuit implementing M. In this step, M is verified to be a Clifford gate which in turn verifies that C is a Clifford gate.
4. Concatenate the circuits for U and M to find a circuit representation of C.
5. Convert this circuit representation into a stabiliser tableau.

In particular, Step 2 involves multiplying $U^\dagger$ and M. The circuit for U has size $\Omega(n^2)$ in the worst case (see above), and multiplying C by a one- or two-qubit gate takes time $\Omega(N^2)$. Thus Step 2, and hence `Stim`'s algorithm, runs in time $\Omega(N^2 n^2)$.

$[\mathbf{C_T}] \to [\mathbf{C_U}]$:
Both `Qiskit` and `Stim` implement $[\mathrm{C_T}] \to [\mathrm{C_U}]$. In `Qiskit`, the conversion is the `to_operator` method of the `Clifford` class. The algorithm first decomposes the Clifford gate as a circuit, using the method of [19], and then finds the matrix of that circuit. If the circuit is size s, this compilation takes time $\Omega(sN^2)$. In the worst case, $s = \Omega(n^2)$ and thus `Qiskit`'s algorithm runs in time $\Omega(N^2 n^2)$. In `Stim`, the conversion is the `to_unitary_matrix` method of the `Tableau` class. It takes $O(N^2 n^2)$ time; as it relies on its implementation of the $[\mathrm{S_P}]$ to $[\mathrm{S_V}]$ conversion, it is also probabilistic and not guaranteed to succeed on every run.

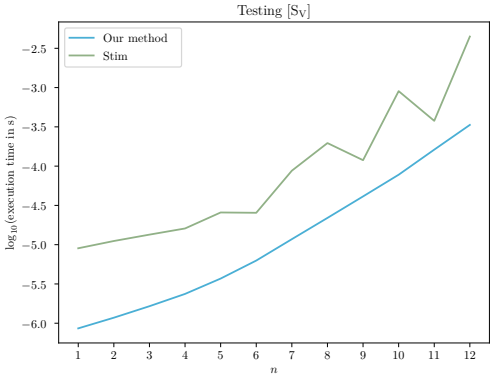
B Timed benchmarks

We benchmarked our `C++` implementations of the ten algorithms for small numbers of qubits n and, wherever possible, compared our implementations with existing ones from `Stim` [28] and `Qiskit` [41]. Uniformly random stabiliser states were generated using the techniques of [48]; uniformly random Clifford gates were generated using the `qiskit.quantum_info.random_clifford` function. The benchmarking was done on a ThinkPad P16 Gen 1 PC running Ubuntu 22.04.5 LTS, with a 12th Gen Intel Core i7-12800HX CPU @ 2 GHz and 16 GiB of RAM. Our code was compiled using `gcc` version 11.4.0, and the wrapper code run using `Python` 3.10.12.

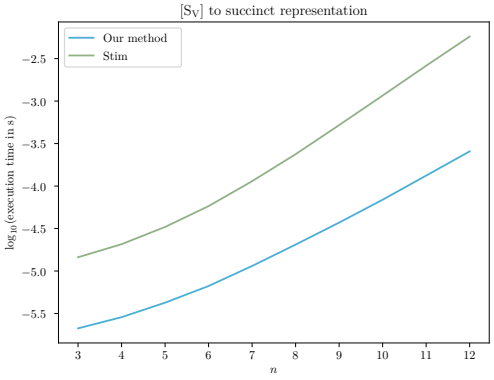
Our data is collected in Figure 1 below; note the log scale y-axis. Each test (a)–(j) was run with 1000 iterations. Wherever 'succinct representation' appears in the title of a graph, we mean the standard way of storing a stabiliser state or Clifford gate in each implementation. In our code, stabiliser states are represented using the `Stabiliser_State` class, which stores the $[\mathrm{S_Q}]$ description, and Cliffords are represented using the `Clifford` class, which stores the $[\mathrm{C_T}]$ description. `Stim` represents both stabiliser states and Cliffords using the `stim.Tableau` class, which is equivalent to the $[\mathrm{S_P}]$ and $[\mathrm{C_T}]$ descriptions. `Qiskit` similarly represents Cliffords in the $[\mathrm{C_T}]$ description using the `qiskit.quantum_info.Clifford` class.

We found absolute timing advantages of some orders of magnitude for small numbers of qubits; our asymptotic complexity analyses guarantee arbitrarily large absolute advantages with sufficiently large n.

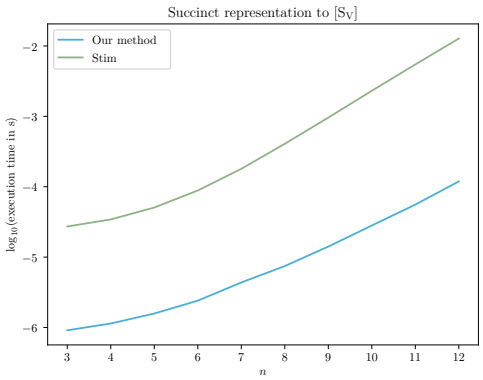
Figure 1: Benchmarking results for our code vs. `Stim` and `Qiskit`. Each test (a)–(j) was run with 1000 iterations.



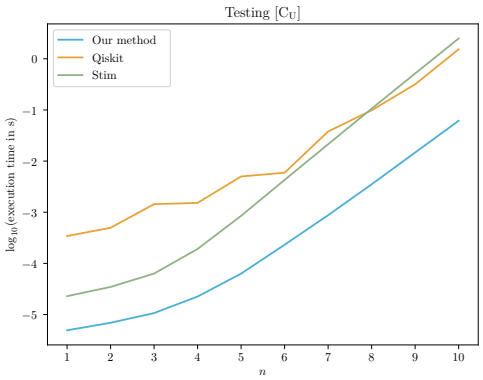
(a) Testing a random mixture of valid stabiliser states and stabiliser states with one entry modified.



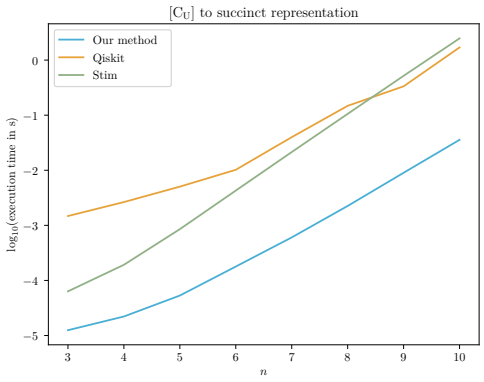
(b) Converting random stabiliser statevectors (amplitudes) into a succinct representation.



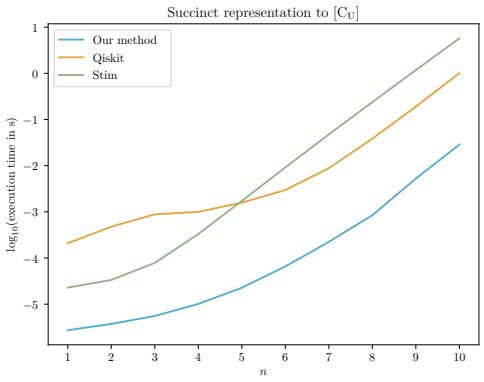
(c) Performing the reverse conversion to (b) on random stabiliser states, starting with the succinct representation used by each library.



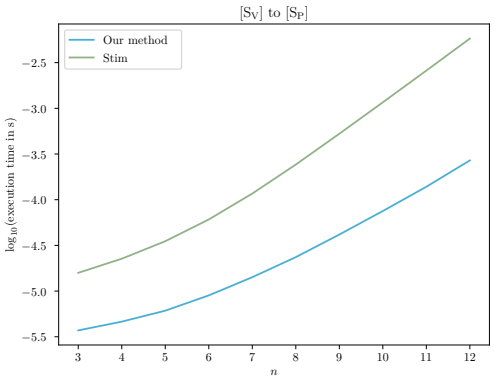
(d) Testing a random mixture of valid Clifford gate matrices and Clifford gate matrices with one entry modified.



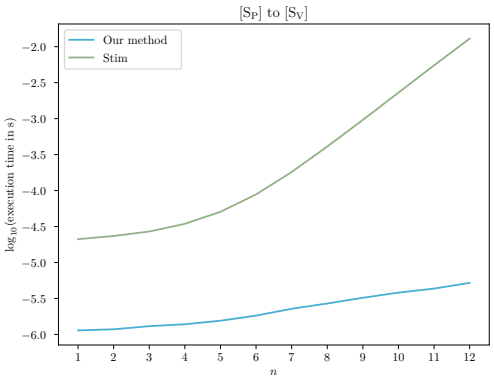
(e) Converting random Clifford gate matrices into a succinct representation.



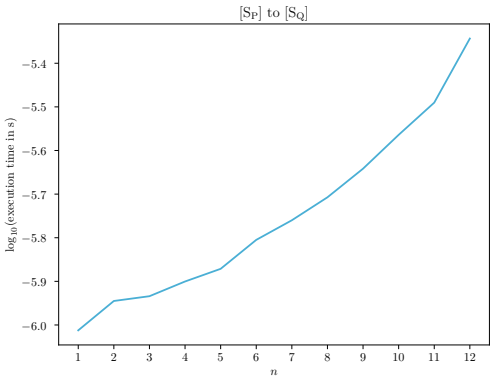
(f) Performing the reverse conversion to (e) on random Clifford gate matrices, starting with the succinct representation used by each library.



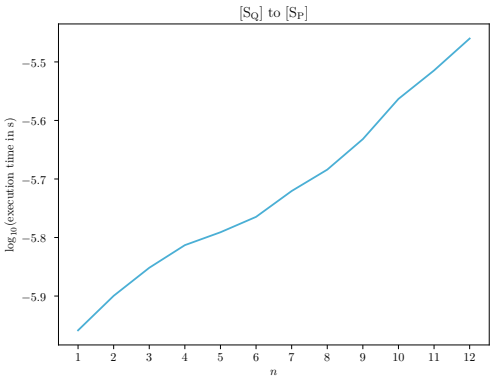
(g) Converting random stabiliser statevectors into check matrix form.



(h) Converting random stabiliser check matrices into statevectors.



(i) Converting random stabiliser check matrices into our succinct representation, $[S_Q]$.



(j) Converting random stabiliser states from our succinct representation to check matrix form.