A graph-state based synthesis framework for Clifford isometries

Timothée Goubault de Brugière[1], Simon Martiel[2], and Christophe Vuillot[1]

[1]Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
[2]Atos Quantum Lab, Les Clayes-sous-Bois, France

We tackle the problem of Clifford isometry compilation, i.e, how to synthesize a Clifford isometry into an executable quantum circuit. We propose a simple framework for synthesis that only exploits the elementary properties of the Clifford group and one equation of the symplectic group. We highlight the versatility of our framework by showing that several normal forms of the literature are natural corollaries. We recover the state of the art two-qubit gate depth necessary for the execution of a Clifford circuit on an LNN architecture, concomitantly with another work. We also propose practical synthesis algorithms for Clifford isometries with a focus on Clifford operators, graph states and codiagonalization of Pauli rotations. Benchmarks show that in all three cases we improve the 2-qubit gate count and depth of random instances compared to the state-of-the-art methods. We also improve the execution of practical quantum chemistry experiments.

1 Introduction

Clifford operators represent one of the most useful and one of the most studied subclass of quantum operators. Clifford operators are involved in numerous applications such as quantum error correction [1], randomized benchmarking protocols [2, 3], quantum state distillation [4, 5], the study of entanglement [6]. It is known that Clifford operators can be generated by the Hadamard gates, the Phase gates and the CNOT gate [7]. Moreover the simulation of such circuits, the so-called Clifford circuits, can be done efficiently on a classical computer [8] and there is a one-to-one mapping between the Clifford operators and the $2n \times 2n$ binary matrices from the symplectic group $Sp(2n, \mathbb{F}_2)$.

Our work lies in the domain of quantum compilation for the quantum circuit model, where one wants to generate quantum circuits for a given hardware to execute a given quantum algorithm. Given the variety of quantum hardwares, each having its own set of universal gates, its own costly resources, and given the different approaches for quantum computation, namely a fault-tolerant approach [9, 10] or a NISQ approach [11], today's compilers have to be able to generate and optimize quantum circuits for several metrics (number of non Clifford gates, number of entangling gates, depth), several set of gates (Clifford+T, MS+SU(2), etc.) and several architectures (Rigetti [12], IBM [13,14], Google [15], etc.). While not universal for quantum computation, the Clifford group plays a significant role in quantum compilation for several types of hardware [16, 17, 18] in both a NISQ of a fault-tolerant setting and notably for the hardwares using the Clifford+T gate set, one of the most studied gate set for compilation problems. In this setting, the

available multi-qubit gates, generally the 2-qubit CNOT gate or the CZ gate, belong to the Clifford group and looking at a quantum circuit as a succession of Clifford operators separated by layers of T-gates is one way to express the problem of optimizing the 2-qubit gate cost as a Clifford circuit optimization problem. Given that the abstract and compact representation of Clifford operators makes it easier to design efficient synthesis algorithms the Clifford operator synthesis problem emerges naturally as an essential brick for quantum circuit optimization, and notably to reduce the 2-qubit gate cost.

The structure of the Clifford group has been deeply studied in the literature. Notably, many normal forms for Clifford circuits have been proposed [7,19,20,21,22]. Those normal forms express any Clifford operator as a series of stages containing one particular gate. For instance, the first normal form proposed by Aaronson and Gottesman is -H-CX-P-CX-P-CX-H-P-CX-P where -H- stands for a stage of Hadamard gates, -P- a stage of phase gates, and -CX- a stage of CNOT gates [7]. An important part of the work in Clifford circuits synthesis during the last decade was to propose shorter normal forms, resulting in shorter or shallower circuits with respect to the 2-qubit gates. Normal forms for the preparation of stabilizer states (states that can be prepared with a Clifford circuit) have also been proposed [7, 23, 24]. Those normal forms, while being universal, impose a strict structure on the produced circuits and may fail to find more efficient implementations for a given operator. Notably the most recent normal forms expose the role of a particular class of quantum operators, the so-called phase polynomials, in the implementation of a Clifford operator [19, 21]. Phase polynomials are now extensively studied in the literature and the best synthesis algorithms show that a stage by stage implementation of phase polynomials is irrelevant [25, 26]. We can expect that a more efficient approach for Clifford synthesis that does not rely on a stage by stage implementation can also be found.

Optimal synthesis of Clifford operators through brute-force search has been recently proposed [27,28,29], but for the moment the method is limited to small Clifford circuits, up to 26 qubits. Other works in the literature also focus on directly optimizing Clifford circuits through template matchings or peephole optimizations [30, 31]. These methods cannot handle the synthesis from an abstract representation nor leverage the global structure of these operators with potential efficiency and performance loss.

Overall, it lacks an efficient approach to synthesize Clifford operators other than by using normal forms or the unpractical tableau formalism. Furthermore, there are many cases in the literature where only a partial implementation of a Clifford operator is necessary. Obviously the synthesis of stabilizer states is one example, but more generally the synthesis of Clifford isometries where some of the input qubits are in the state $|0\rangle$ is of particular interest in some error-correction protocols such as magic state distillation [32, 33]. Again, to our knowledge, no work in the literature proposes to generalize the concept of normal forms and synthesis frameworks to Clifford isometries.

The main result of our paper is a generalized synthesis framework for Clifford isometries. Our framework is strongly inspired by the results obtained via the ZX-calculus for the optimization of quantum circuits [20] although the derivation of our framework is done without any use of the ZX-calculus. Indeed, our proof uses only the properties of the Clifford group and the symplectic matrices used to represent Clifford operators. Our framework shares also some similarities with the normal form proposed in [21]. Where our framework stands out is first of all in its simplicity: it only requires to reduce a symmetric boolean matrix into a suitable defined identity matrix with the use of a few elementary operations, each corresponding to one elementary Clifford gate. Notably it naturally includes the Hadamard gate in the available operations, which is the gate that is missing when considering a Clifford operator with stages of phase polynomials. We also

highlight its versatility by showing that many normal forms proposed in the literature can be trivially recovered. Similarly to [21], we also characterize more precisely the operators involved in the synthesis of one operator when the gate set is restricted to $\{CNOT, CZ, S\}$. We show that any Clifford operation on n qubits can be executed in two-qubit gate depth $7n-2$ on a Linear Nearest Neighbour (LNN) architecture. Our result is concomitant with the $7n+2$ depth (then improved to $7n-4$) of another article in the literature [34]. Overall, both work improve the previous best result of $9n$ [20, 21].

Finally we apply our framework to the design of practical synthesis algorithms for Clifford isometries. We propose two versions: one that optimizes the 2-qubit gate count and one that optimizes the 2-qubit gate depth. In our benchmarks we will focus on two special cases: the synthesis of graph states and the synthesis of Clifford operators. On random instances, we show that we outperform the state of the art. We also extend our framework to the codiagonalization of groups of commuting Pauli rotations, we apply our algorithms to a set of benchmarks from quantum chemistry and report improvements over the state of the art.

2 Background

2.1 Quantum isometries

An isometry is a generalization of quantum states and quantum operators. They map Hilbert spaces of possibly two different dimensions and they preserve the inner products, similarly to unitary matrices.

Definition 1. *Let $0 \leq k \leq n$ be two integers. A k to n isometry can be represented by a $2^n \times 2^k$ complex matrix V such that*

$$V^\dagger V = I_{2^k \times 2^k}.$$

In other words, isometries can be seen as the action of a quantum operator on a quantum memory where some of the qubits are in the state $|0\rangle$ — those qubits can be regarded as ancillary qubits. Generally the compilation of isometries result in shorter circuits than systematically synthesizing the full quantum operator because the ancillary qubits in a fixed input state offer some degrees of freedom in the synthesis. Clearly, a state preparation is usually less costly than a full operator compilation.

In this paper we focus on the synthesis of a particular subclass of isometries: Clifford isometries, i.e, isometries that can be generated by a Clifford operator.

2.2 Pauli matrices and the Clifford group

The Pauli matrices are defined by

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

and they obey the following equations:

$$X^2 = Y^2 = Z^2 = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$XY = iZ,$$

$$ZX = iY,$$
$$YZ = iX.$$

Therefore, the Pauli matrices generate a group where any member can be written as

$$i^k\{I, X, Y, Z\} \text{ for some } k \in \{0, 1, 2, 3\}.$$

This generalizes to several qubits: a Pauli operator on n qubits is any operator of the form

$$P = i^k \bigotimes_{j=1}^{n} G_j, \text{ where } G_j \in \{I, X, Z, Y\}, k \in \{0, 1, 2, 3\}.$$

We write $\mathcal{P}_n$ for the set of Pauli operators on n qubits. For conciseness any Pauli operator, up to the global phase, can be written as a Pauli word, for instance $IZXI = I \otimes Z \otimes X \otimes I$. We also write $G_j, G \in \{X, Z, Y\}$, the Pauli operator that applies solely the operator G on qubit j.

With the 2-bit encoding

$$I \to 00, X \to 01, Z \to 10, Y \to 11$$

we can always represent a Pauli operator with a boolean vector p of size $2n$ where $(p[j], p[j+n])$ encodes the Pauli matrix applied on qubit j plus an extra two bits to encode the value of k. We write P interchangeably for the Pauli operator or the associated boolean vector.

Example:

$$YZXI = \left[\begin{array}{c} 1 \\ 1 \\ 0 \\ 0 \\ \hline - - - \\ 1 \\ 0 \\ 1 \\ 0 \end{array}\right].$$

Given such a boolean vector P, we write P^Z for the first n values of P (encoding the Z components) and P^X for the last n values (encoding the X components). The product of two Pauli operators $(P, k) = (P_1, k_1) \times (P_2, k_2)$ is given by

$$P = P_1 \oplus P_2$$
$$k = k_1 + k_2 + \|P_2^X * P_1^Z\| - \|P_1^X * P_2^Z\| \mod 4$$

i.e, the boolean vector of the product of two Pauli operators is given by the bitwise XOR of the associated boolean vectors. Here $\|\cdot\|$ stands for the Hamming norm and $*$ stands for the element-wise product.

Definition 2. *The Clifford group on n qubits is defined by*

$$\mathcal{C}_n = \{U \in \mathcal{U}(2^n) \mid UPU^\dagger \in \mathcal{P}_n \ \ \forall P \in \mathcal{P}_n\}.$$

It is well known that any Clifford operator can be generated by a quantum circuit in the gate set $\{H, S, CNOT\}$. It is common to also add the CZ gate and the $\sqrt{X}$ gate in the usual gate set. A Clifford circuit is any circuit made of Clifford gates and uniquely defines a Clifford operator.

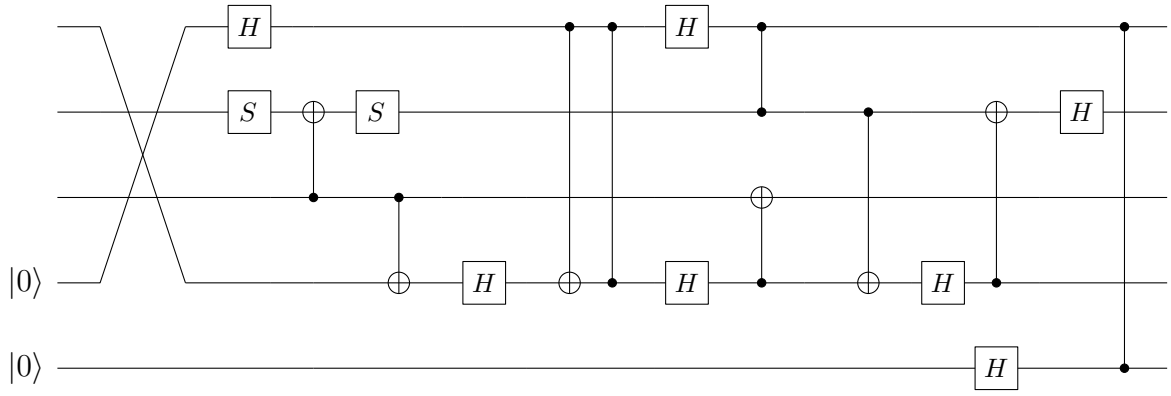


Figure 1: Example of a 5-qubit Clifford circuit with 2 qubits set to $|0\rangle$. This defines a 3 to 5 Clifford isometry.

Definition 3. *Let $0 \leq k \leq n$ be two integers. A k to n Clifford isometry V is defined as*

$$V = CI_{2^n \times 2^m}$$

where C belongs to the Clifford group on n qubits and $I_{2^n \times 2^m}$ denotes the first 2^m columns of the identity matrix.

Equivalently, one can see a k to n Clifford isometry as a Clifford operator where $n-k$ of the inputs are set to $|0\rangle$. The case $k = 0$ defines a stabilizer state and $k = n$ corresponds to a Clifford operator. An example of a 3 to 5 Clifford isometry defined by a Clifford circuit is given in Fig 1.

For the rest of the paper, we will solely focus on the conjugation (by Clifford operators) of Pauli operators of the form

$$\bigotimes_{j=1}^{n} G_j, \text{ where } G_j \in \{I, X, Z, Y\}$$

i.e, without a global phase.

Property 1. *The conjugations of any n-qubit Pauli word by a Clifford operator can be efficiently stored in a boolean vector of size $2n + 1$.*

Proof. The eigenvalues of the Pauli matrices without a global phase are $\{-1, 1\}$ and are unchanged after the conjugation by a Clifford. Therefore, the conjugations obtained can only be of the form

$$(-1)^k \bigotimes_{j=1}^{n} G_j, \text{ where } G_j \in \{I, X, Z, Y\}, k \in \{0, 1\}$$

and we only have to store the value of k on one bit.

□

2.3 Tableau representation of Clifford isometries

It is well-known that Clifford operators can be efficiently represented by a polynomial sized tableau with boolean entries and the tableau can also be efficiently updated when

multiplied by another Clifford operator. This relies on the fact that only the conjugations of the Pauli operators $Z_i, X_i, i = 1 \dots n$ are sufficient to fully characterize an n-qubit Clifford operator[1]. The tableau is of size $(2n+1) \times 2n$, one column for each conjugation, and the $2n$ rows without the overall sign define a boolean matrix that belongs to the symplectic group $Sp(2n, \mathbb{F}_2)$. A matrix M in the symplectic group satisfies the following relation:

$$M\Omega M^T = \Omega$$

where $\Omega = \begin{bmatrix} 0 & I_n \\ I_n & 0 \end{bmatrix}$. The symplectic product between two Pauli vectors is 0 if they commute and 1 otherwise. So matrices in the symplectic group are simply one case of commutation relations between a set of Pauli vectors. Given that the identity tableau is symplectic and that $CP_1C^\dagger$ and $CP_2C^\dagger$ have the same commutation relation as P_1 and P_2, the tableau of any Clifford operator C necessarily has the same commutation relations as the identity tableau and the tableau belongs to the symplectic group.

We write the tableau

$$M = \begin{bmatrix} Z \to Z & X \to Z \\ Z \to X & X \to X \end{bmatrix}$$

where the block $Z/X \to Z/X$ explicitly shows which part corresponds to the Z/X components of the images of the Z_i/X_i operators.

This efficient representation can be generalized to Clifford isometries and we give a constructive proof that a k to n Clifford isometry can be represented by a subset of $n+k$ columns of the tableau of a Clifford operator. We also show that this representation is not unique and we give a construction of the equivalence class.

Theorem 1. *Any k to n Clifford isometry V can be represented by $n+k$ columns of a Clifford tableau M. We write*

$$M_V = \begin{bmatrix} Z_1 \to Z & Z_2 \to Z & X \to Z \\ Z_1 \to X & Z_2 \to X & X \to X \end{bmatrix}$$

where

- $\begin{bmatrix} Z_1 \to Z \\ Z_1 \to X \end{bmatrix} \in \mathbb{F}_2^{(2n+1)\times k}$ *encodes the conjugations of $Z_i, i = 1 \dots k$,*
- $\begin{bmatrix} Z_2 \to Z \\ Z_2 \to X \end{bmatrix} \in \mathbb{F}_2^{(2n+1)\times (n-k)}$ *encodes the conjugations of $Z_i, i = k+1 \dots n$*
- $\begin{bmatrix} X \to Z \\ X \to X \end{bmatrix} \in \mathbb{F}_2^{(2n+1)\times k}$ *encodes the conjugations of $X_i, i = 1 \dots k$.*

Moreover, any of the $n-k$ columns of $\begin{bmatrix} Z_2 \to Z \\ Z_2 \to X \end{bmatrix}$ can be added to any other column other than itself such that the specification of the isometry is unchanged.

Proof. Let V be a k to n Clifford isometry. There exists a Clifford operator C such that V is characterized by the set

$$\{V |b\rangle = C |b\rangle |0\rangle^{\otimes n-k} \mid b \in \mathbb{F}_2^k.\}.$$

[1] By product we get the conjugation of any Pauli word, and the set of the 4^n Pauli words is a basis for the full set of complex matrices.

Let $|\psi\rangle = C|0\rangle^{\otimes n}$, we have

$$\begin{aligned}\forall b \in \mathbb{F}_2^k, C|b\rangle|0\rangle^{\otimes n-k} &= C\prod_{i=1}^{k} X_i^{b_i}|0\rangle^{\otimes n} \\ &= C\prod_{i=1}^{k} X_i^{b_i} C^\dagger C|0\rangle^{\otimes n} \\ &= \left(C\prod_{i=1}^{k} X_i^{b_i} C^\dagger\right)|\psi\rangle .\end{aligned}$$

Therefore, V is completely characterized by the values of

$$C\prod_{i=1}^{k} X_i^{b_i} C^\dagger |\psi\rangle , \forall b \in \mathbb{F}_2^k.$$

By product we only need the knowledge of $|\psi\rangle$ and $CX_iC^\dagger, i = 1 \dots k$. Equivalently we only need the states $VX_i|0\rangle^{\otimes k}, i = 1 \dots k$.

It is well-known that a Clifford state is completely determined by the knowledge of n Pauli operators that stabilize the state [7]. Given that

$$Z_i|0\rangle^{\otimes n} = |0\rangle^{\otimes n}, \ i = 1 \dots n,$$

we have

$$CZ_iC^\dagger|\psi\rangle = |\psi\rangle , \ i = 1 \dots n$$

that represent such a set of stabilizers for $|\psi\rangle$.

Overall, we need the values of

$$CZ_iC^\dagger, \ i = 1 \dots n$$

and

$$CX_iC^\dagger, \ i = 1 \dots k$$

to completely determine V. The tableau representation of C gives the conjugations of $Z_i, X_i, i = 1 \dots n$ by C so finally we only need the first $n + k$ columns of the tableau of C.

We write

$$M_V = \begin{bmatrix} Z_1 \to Z & Z_2 \to Z & X \to Z \\ Z_1 \to X & Z_2 \to X & X \to X \end{bmatrix}$$

where

- $\begin{bmatrix} Z_1 \to Z \\ Z_1 \to X \end{bmatrix} \in \mathbb{F}_2^{(2n+1)\times k}$ encodes the conjugations of $Z_i, i = 1 \dots k$,
- $\begin{bmatrix} Z_2 \to Z \\ Z_2 \to X \end{bmatrix} \in \mathbb{F}_2^{(2n+1)\times (n-k)}$ encodes the conjugations of $Z_i, i = k+1 \dots n$
- $\begin{bmatrix} X \to Z \\ X \to X \end{bmatrix} \in \mathbb{F}_2^{(2n+1)\times k}$ encodes the conjugations of $X_i, i = 1 \dots k$.

We now show that adding any column of $\begin{bmatrix} Z_2 \to Z \\ Z_2 \to X \end{bmatrix}$ to any column of $\begin{bmatrix} X \to Z \\ X \to X \end{bmatrix}$ does not modify the isometry specified.

This result relies on the following identities

$$\begin{aligned}
\forall i \in [\![1,k]\!], \forall a \in \mathbb{F}_2^{n-k},\ VX_i |0\rangle^{\otimes k} &= CX_i |0\rangle^{\otimes n} \\
&= CX_i \prod_{j=k+1}^{n} Z_j^{a_{j-k}} |0\rangle^{\otimes n} \\
&= C \left(X_i \prod_{j=k+1}^{n} Z_j^{a_{j-k}} \right) C^\dagger |\psi\rangle \\
&= \underbrace{\left(CX_iC^\dagger\right)}_{\text{one column of } \begin{bmatrix} X \to Z \\ X \to X \end{bmatrix}} \times \prod_{j=k+1}^{n} \underbrace{\left(CZ_j^{a_{j-k}}C^\dagger\right)}_{\text{one column of } \begin{bmatrix} Z_2 \to Z \\ Z_2 \to X \end{bmatrix}} |\psi\rangle ,
\end{aligned}$$

Normally, to reconstruct the states $VX_i |0\rangle^{\otimes k}$ from our tableau, we need the images of the Z_i's to reconstruct $|\psi\rangle$ and the images of the X_i. What the identity above shows is that if we know any image of a Pauli word of the form $X_i \prod_{j=k+1}^{n} Z_j^{a_{j-k}}, a \in \mathbb{F}_2^{n-k}$ then we can still reconstruct the desired state. This is equivalent to adding any column of $\begin{bmatrix} Z_2 \to Z \\ Z_2 \to X \end{bmatrix}$ to any column of $\begin{bmatrix} X \to Z \\ X \to X \end{bmatrix}$. Similarly, adding any column of $\begin{bmatrix} Z_2 \to Z \\ Z_2 \to X \end{bmatrix}$ to $\begin{bmatrix} Z_1 \to Z \\ Z_1 \to X \end{bmatrix}$ and $\begin{bmatrix} Z_2 \to Z \\ Z_2 \to X \end{bmatrix}$ (other than itself) only gives a new basis for the stabilizers of $|\psi\rangle$ but it does not change the specification of $|\psi\rangle$.

□

Note that the asymmetry between the Z and X operators in the tableau representation of an isometry comes from the fact that we choose the ancillary qubits to be in the $|0\rangle$ state. One could have chosen to have ancillas in the state $|+\rangle$ and we would need the images of all the X_i's and not all the Z_i's. Any intermediate mix of $|0\rangle$ and $|+\rangle$ states as ancillas is also allowed.

Following the rules for simulating Clifford circuits by tracking the images of the Pauli operators [7], we can compute the tableau of the isometry of Fig 1. We get

$$M_V = \begin{bmatrix} Z \to Z & X \to Z \\ Z \to X & X \to X \end{bmatrix} = \begin{bmatrix} A & C \\ B & D \end{bmatrix} = \left[\begin{array}{ccccc|ccc} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ & & & & & & & \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right].$$

2.4 The synthesis problem

We propose one formalization of the synthesis of Clifford isometries. We work in the gate set $\{H, S, CZ, CNOT\}$ which is known to be universal for Clifford computation. We suppose we are given the tableau representation M_V of a k to n isometry. We only have access to a black box representation of the isometry in a circuit, see Figure 2.

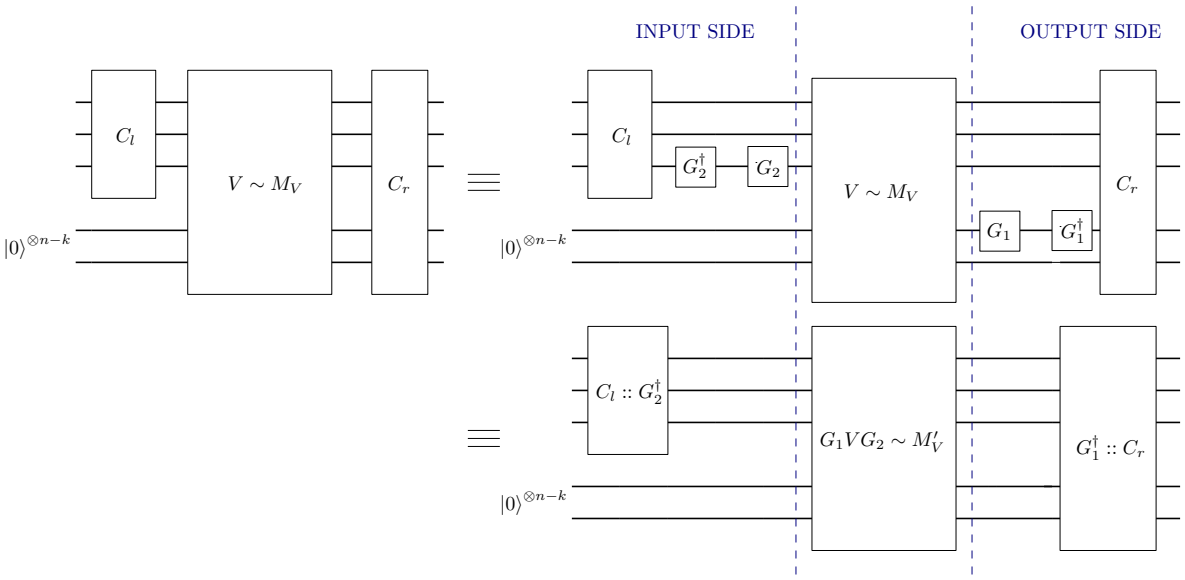


Figure 2: Illustration of the synthesis framework: gates and their inverses are applied on the left and right of the isometry. One gate is merged with the isometry while the other one is stacked into a left/right circuit. Once $M_V = I$, then $C_l :: C_r$ implements the desired isometry. Throughout the synthesis process, the circuit always implements the desired isometry.

2.4.1 Action of the elementary gates.

On the output side. Given a black box circuit implementing the isometry V, adding a gate G_1 from the right gives the operator $G_1 V$ to characterize and we therefore need to compute the values of

$$(G_1 V) \cdot Z_i \cdot (G_1 V)^\dagger = G_1 \left(V Z_i V^\dagger \right) G_1^\dagger, i = 1..n$$

and

$$(G_1 V) \cdot X_i \cdot (G_1 V)^\dagger = G_1 \left(V X_i V^\dagger \right) G_1^\dagger, i = 1..k.$$

The values of $VZ_iV^\dagger$ and $VX_iV^\dagger$ are given by M_V and by the rules of conjugation we know how to directly update M_V. For simplicity we write

$$M_V = \begin{bmatrix} T \\ r \end{bmatrix}$$

where r stands for the sign vector and T is the $(2n \times (n+k))$ matrix that specifies the conjugations without the sign. We also note $\wedge$ for the bitwise AND operation.

- Hadamard on qubit m: swap rows m and $n+m$, set

$$r = r \oplus (T[m,:] \wedge T[n+m,:]),$$

- S on qubits m: add row $n+m$ to row m, set

$$r = r \oplus (T[m,:] \wedge T[n+m,:]),$$

- CZ on qubits p, m: add row $n+p$ to row m and row $n+m$ to row p, set

$$r = r \oplus \big[T[n+m,:] \wedge T[n+p,:] \wedge (T[p,:] \oplus T[m,:])\big],$$

- CNOT with control p and target m: add row $n+p$ to row $n+m$ and row m to row p, set

$$r = r \oplus \big[T[n+p,:] \wedge T[m,:] \wedge (1 \oplus T[n+m,:] \oplus T[p,:])\big].$$

On the input side. When a gate G_2 is added from the left, we need the values of

$$(VG_2) \cdot Z_i \cdot (VG_2)^\dagger$$

and

$$(VG_2) \cdot X_i \cdot (VG_2)^\dagger.$$

Writing $G_2 Z_i G_2^\dagger = \bigoplus_i a_i Z_i \oplus \bigoplus_i b_i X_i, a_i \in \{0,1\}, b_i\{0,1\}$ we have

$$(VG_2) \cdot Z_i \cdot (VG_2)^\dagger = \bigoplus_i a_i \left(V Z_i V^\dagger\right) \oplus \bigoplus_i b_i \left(V X_i V^\dagger\right).$$

Therefore the values of $(VG_2) \cdot Z_i \cdot (VG_2)^\dagger$ are linear combinations of the columns of M_V. The same applies for the values of $(VG_2) \cdot X_i \cdot (VG_2)^\dagger$.

This gives:

- Hadamard on qubit m: swap columns m and $n+m$,
- S on qubits m: add column m to column $n+m$,
- CZ on qubits p, m: add column p to column $n+m$ and column m to column $n+p$,
- CNOT with control p and target m: add column p to column m and column $n+m$ to column $n+p$,

Similar updates for the sign vector can be derived.

For the rest of the paper we will now refer as the *input qubits* the k qubits on which a gate can be applied on the left. Similarly we refer as the *output qubits* the n qubits on which a gate can be applied on the right.

2.4.2 The synthesis workflow.

The synthesis process consists in adding a Clifford gate and its inverse side by side on the left or right of the black box isometry, as shown in Fig. 2. This will not change the overall functionality of the circuit but one can merge one Clifford gate to the black box and update M_V. At the same time the inverse gate is stacked to a list of left or right operations. When M_V is equal to the identity tableau, we have a valid sequence of operations implementing the Clifford isometry by stacking the left operations and the *reverse* of the right operations.

Note that we can only focus on reducing T to the identity because the sign vector can be treated separately at the end of the computation. Indeed, if $T = I$, adding twice an S gate on qubit m at the end of the circuit will simply do the operation

$$r[m] = r[m] \oplus 1.$$

In other words, keeping track of the value of r is sufficient because it can be ultimately zeroed with a layer of Z gates.

Designing synthesis algorithms directly from the tableau representation is clumsy. For the synthesis of Clifford states and Clifford operators the derivation of normal forms is common instead of proposing a direct synthesis algorithm [7, 19, 20, 21, 22, 23, 24].

We try to overcome both the difficulty of manipulating the Clifford tableaus and the necessity to use normal forms by proposing a simple framework for the synthesis of Clifford isometries. One advantage of our framework will be its versatility: it can be used to design either direct synthesis algorithms or to prove normal forms.

3 A graph-state formalism for Clifford isometries synthesis

We prove that any k to n Clifford isometry can be put in a graph-state form up to local Hadamard gates. Then we derive a synthesis framework by showing the behavior of the different elementary Clifford gates.

3.1 Graph-state structure of the tableau representation

First, we prove a normal form for symplectic matrices that exposes an underlying graph-state structure of Clifford operators.

Theorem 2. *Let* $M = \begin{bmatrix} A & C \\ B & D \end{bmatrix} = \begin{bmatrix} Z \to Z & X \to Z \\ Z \to X & X \to X \end{bmatrix} \in Sp(2n, \mathbb{F}_2)$. *If* B *is invertible, then there exist two boolean symmetric matrices* S, S' *such that*

$$M = \left[\begin{array}{c|c} S'B & (B^T)^{-1} \oplus S'BS \\ \hline B & BS \end{array}\right]. \tag{1}$$

Proof. If B is invertible, then we can define $S = B^{-1}D$ and $S' = AB^{-1}$ and $K = C \oplus (B^T)^{-1}$ such that

$$A = S'B,$$
$$D = BS,$$
$$C = (B^T)^{-1} \oplus K.$$

Matrices in the symplectic group verify three different equations. One of them will prove that $K = S'BS$ and the two others will show that S and S' are symmetric.

1. $B^T A \oplus A^T B = 0$ gives

$$B^T S'B \oplus B^T S'^T B = B^T (S' \oplus S'^T) B = 0$$

which is possible if and only if $S' = S'^T$.

2. $B^TC \oplus A^TD = I$ gives

$$B^T((B^T)^{-1} \oplus K) \oplus B^TS'^TBS = I \oplus B^T(K \oplus S'BS) = I.$$

So we have $B^T(K \oplus S'BS) = 0$ which is possible if and only if $K = S'BS$.

3. $D^TC \oplus C^TD = 0$ gives

$$\begin{aligned} S^TB^T(B^{-T} \oplus S'BS) \oplus ((B^{-T} \oplus S'BS))^TBS &= S^TB^TB^{-T} \oplus S^TB^TS'BS \oplus B^{-1}BS \oplus S^TB^TS'^TBS \\ &= S^T \oplus S = 0 \end{aligned}$$

which is possible if and only if $S = S^T$.

□

This trivially extends to Clifford isometries:

Corollary 1. *Let* $M_V = \begin{bmatrix} A & C \\ B & D \end{bmatrix} \in \mathbb{F}_2^{2n\times(n+k)}$ *be the tableau representation of a Clifford isometry. If* B *is invertible, then there exist two boolean symmetric matrices* $G_k \in F_2^{k\times k}, G_n \in F_2^{n\times n}$ *such that*

$$M_V = \left[\begin{array}{c|c} G_nB & (B^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_nB\begin{pmatrix} G_k \\ 0 \end{pmatrix} \\ \hline B & B\begin{pmatrix} G_k \\ 0 \end{pmatrix} \end{array}\right]. \tag{2}$$

Proof. From Eq. 1, by taking the first $n+k$ columns we have an operator of the form

$$M_V = \left[\begin{array}{c|c} G_nB & (B^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_nB\begin{pmatrix} G_k \\ F \end{pmatrix} \\ \hline B & B\begin{pmatrix} G_k \\ F \end{pmatrix} \end{array}\right]$$

where $F \in F_2^{(n-k)\times k}$ is an arbitrary rectangular boolean matrix, G_n is S' in Eq. 1 and G_k is the top block $k \times k$ of S.

We rewrite M_V as

$$M_V = \left[\begin{array}{c|c|c} G_nB\begin{pmatrix} I_k \\ 0_{(n-k)\times k} \end{pmatrix} & G_nB\begin{pmatrix} 0_{k\times(n-k)} \\ I_{n-k} \end{pmatrix} & (B^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_nB\begin{pmatrix} G_k \\ F \end{pmatrix} \\ \hline B\begin{pmatrix} I_k \\ 0_{(n-k)\times k} \end{pmatrix} & B\begin{pmatrix} 0_{k\times(n-k)} \\ I_{n-k} \end{pmatrix} & B\begin{pmatrix} G_k \\ F \end{pmatrix} \end{array}\right]$$

by splitting the first n columns of M_V and we recall from Theorem 1 that adding any column $k+1 \dots n$ to any column $n+1 \dots n+k$ of M_V is free and does not change the functionality of the Clifford isometry. So with suitable column operations we can zero F. Hence the result.

□

Theorem 3. *Any Clifford isometry, up to Hadamard gates on the output qubits, has a normal form given by Eq. 2.*

Proof. It is sufficient to prove that for any tableau representation of a Clifford isometry the action of Hadamard gates on the output qubits, i.e, specific row swaps on the tableau, is enough to make B invertible. This is a well-known result already proved in [7] for instance.

□

At first sight, it seems that we need to track the values of G_n, G_k, B to perform the synthesis of a Clifford isometry. In fact, the knowledge of G_n, G_k and a submatrix of B is sufficient:

Theorem 4. *Given a Clifford isometry of the form given by Eq. 2, if G_k, G_n are equal to the null matrix and if $B = \begin{bmatrix} I_k & 0_{k\times(n-k)} \\ B' & B'' \end{bmatrix}$ for some boolean matrices B', B'', then the Clifford isometry acts as the $H^{\otimes n}$ operator on the quantum memory.*

Proof. Replacing in Eq. 2 gives

$$M_V = \left[\begin{array}{c|c} 0 & (B^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \\ \hline B & 0 \end{array}\right].$$

Given that $B = \begin{bmatrix} I_k & 0_{k\times(n-k)} \\ B' & B'' \end{bmatrix}$, we also have $B^{-1} = \begin{bmatrix} I_k & 0_{k\times(n-k)} \\ B''' & B'''' \end{bmatrix}$ for some boolean matrices B''', B''''. Therefore, it is easy to check that

$$(B^T)^{-1}\begin{bmatrix} I_k \\ 0 \end{bmatrix} = \begin{bmatrix} I_k \\ 0 \end{bmatrix}$$

Overall, we have a new block structure for M_V,

$$M_V = \left[\begin{array}{c|c|c} 0 & 0 & \begin{pmatrix} I_k \\ 0 \end{pmatrix} \\ \hline I_k & 0_{k\times(n-k)} & 0 \\ B' & B'' & 0 \end{array}\right].$$

B is invertible, therefore B'' is also invertible. We recall that we can add any column of B'' on any other column of M_V, it does not change the functionality of the Clifford isometry. So with suitable column operations we can reduce B'' to the identity operator and zero B'. We end with the operator

$$M_V = \left[\begin{array}{c|c} 0_{k\times n} & I_k \\ 0_{(n-k)\times n} & 0_{(n-k)\times k} \\ \hline I_n & 0_{n\times k} \end{array}\right].$$

which correspond to the k to n isometry $H^{\otimes k} \otimes |+\rangle^{n-k}$.

□

This means that the last $n-k$ rows of B or B^{-1} do not influence the specification of the identity isometry and it is useless to keep track of their values in the synthesis process. Instead, we set

$$G = \begin{bmatrix} G_k & B_{k,n} \\ B_{k,n}^T & G_n \end{bmatrix}$$

where $B_{k,n} = B^{-1}[1:k,:] \in F_2^{k\times n}$. This is the *graph-state form* of the Clifford isometry because G can be seen as an adjacency matrix and as we will see the CZ gates directly add or remove an edge from G, similarly to the graph-state formalism. We illustrate the computational process on the example of the isometry in Fig 1. We recall that we computed its tableau

$$M_V = \begin{bmatrix} A & C \\ B & D \end{bmatrix} = \left[\begin{array}{ccccc|ccc} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ & & & & & & & \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right].$$

One can check that B is already full rank. We have

$$B^{-1} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and from it follows the values of G_n, G_k and B_{kn}:

$$G_n = AB^{-1} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$B_{kn} = \begin{pmatrix} I_k & 0_{k\times(n-k)} \end{pmatrix} B^{-1} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

$$G_k = B_{kn}D = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

The goal is to reduce G to

$$\left[\begin{array}{c|cc} 0_k & I_k & 0_{k\times(n-k)} \\ \hline I_k & \multicolumn{2}{c}{\multirow{2}{*}{$0_{n\times n}$}} \\ 0_{n-k\times k} & & \end{array}\right].$$

Indeed, when $G_n = 0, G_k = 0$ and $B_{k,n} = \begin{bmatrix} I_k \\ 0_{(n-k)\times k} \end{bmatrix}$ we have seen from the proof above that M_V acts as the $H^{\otimes n}$ operator. Adding n Hadamard gates on the output side in the framework illustrated in Fig. 2 reduces M_V to the identity isometry and this finishes the synthesis.

3.2 Available operations

We now list the action of the authorized operations on G. We write e_i the i-th canonical vector, e_{ij} the matrix with all zero except on entry (i,j) and $E_{ij} = I \oplus e_{ij}$. For clarity we also note $\tilde{e}_{ij} = e_{ij} \oplus e_{ji}$ if $i \neq j$ and $\tilde{e}_{ii} = e_{ii}$. The proofs are given in the Appendix.

- S on the output qubit i $(1 \leq i \leq n)$:

$$G \leftarrow G \oplus \tilde{e}_{i+k,i+k}$$

- S on the input qubit i $(1 \leq i \leq k)$:

$$G \leftarrow G \oplus \tilde{e}_{i,i}$$

- CZ on the output qubits i, j $(1 \leq i < j \leq n)$:

$$G \leftarrow G \oplus \tilde{e}_{i+k,j+k}$$

- CZ on the input qubits i, j $(1 \leq i < j \leq n)$:

$$G \leftarrow G \oplus \tilde{e}_{ij}$$

- $CNOT$ on the output qubits with control i, target j $(1 \leq i < j \leq n)$:

$$G \leftarrow E_{i+k,j+k} G E_{j+k,i+k}.$$

- $CNOT$ on the input qubits with control i, target j $(1 \leq i < j \leq k)$:

$$G \leftarrow E_{ij} G E_{ji}.$$

Contrary to the other gates that preserve the structure of Eq. 2, the Hadamard gate and the $R_x(\pi/2)$ gate do not guarantee it because the matrix B can become non invertible. We need to separate two cases.

- if $G_n[i,i] = 0$, then we can apply an $R_x(\pi/2)$ on the output qubit i:

$$G \leftarrow G \oplus G[:, k+i] G[:, k+i]^T.$$

- if $G_n[i,i] = 1$, then we apply an Hadamard gate on the output qubit i, or equivalently we apply $SR_x(\pi/2)S$, and the action on G is

$$G \leftarrow G \oplus g_i g_i^T$$

  where $g_i = G[:, k+i] \oplus e_i$.

- if $G_k[i,i] = 0$, then we can apply an $R_x(\pi/2)$ on the input qubit i, we get

$$G \leftarrow G \oplus G[:, i] G[:, i]^T.$$

- if $G_k[i,i] = 1$, then we apply an Hadamard gate on the input qubit i, or equivalently we apply $SR_x(\pi/2)S$, and the action on G is

$$G \leftarrow G \oplus g_i g_i^T$$

  where where $g_i = G[:, i] \oplus e_i$.

Clifford gate	Action on G	Matrix interpretation	Graph interpretation	Range of validity
S_i	$G[i,i] \leftarrow G[i,i] \oplus 1$	Flips diagonal entry	Complements loop of node i	$1 \leq i \leq n+k$
$CZ_{i,j}$	$G[i,j] \leftarrow G[i,j] \oplus 1$ $G[j,i] \leftarrow G[j,i] \oplus 1$	Flips two symmetric entries	Complements edge (i,j)	$1 \leq i,j \leq k$ or $k+1 \leq i,j \leq n+k$
$CNOT_{i,j}$	$G[i,:] \leftarrow G[i,:] \oplus G[j,:]$ $G[:,i] \leftarrow G[:,i] \oplus G[:,j]$	Elementary row and column operation	Complements the neighborhood of node j for node i	$1 \leq i,j \leq k$ or $k+1 \leq i,j \leq n+k$
H_i (if $G[i,i]=1$)	$G \leftarrow G \oplus gg^T$ $g = G[:,i] \oplus e_i$	Rank-one update	Local complementation on node i (with loops except on node i)	$1 \leq i \leq n+k$
$(R_x)_i$ (if $G[i,i]=0$)	$G \leftarrow G \oplus gg^T$ $g = G[:,i]$	Rank-one update	Local complementation on node i (with loops)	$1 \leq i \leq n+k$

Table 1: Available operations for the synthesis of a k to n Clifford isometry G.

The available operations are summarized in Table 1. For practical reasons, we also give the available operations as actions on the submatrices G_n, G_k and $B_{k,n}$ in Table 2. We will mostly design algorithms in the gate set {CNOT, CZ, S}, without the Hadamard gate, and it is easier to give intuitions on their behavior by looking at the action of the gates on G_n, G_k and $B_{k,n}$ separately rather than on the full graph matrix G. Notably, we want to emphasize the following key points:

- CZ gates and S gates only flip an entry of either G_k or G_n, depending on which side those gates are applied,
- a CNOT circuit on the output side will apply an operator C on $B_{k,n}^T$, giving the new operator $CB_{k,n}^T$ and will conjugate G_n by the same operator, giving CG_nC^T,
- similarly a CNOT circuit on the input side will apply an operator C on $B_{k,n}^T$, giving the new operator $B_{k,n}^T C$ and will conjugate G_k by the same operator, giving $C^T G_k C$.

3.3 Summary: the synthesis framework

We have shown that any k to n Clifford isometry, up to Hadamard gates, is completely determined by a symmetric boolean matrix $G \in F_2^{(n+k)\times(n+k)}$. G has the following block structure

$$G = \begin{bmatrix} G_k & B_{k,n} \\ B_{k,n}^T & G_n \end{bmatrix}$$

where $G_k \in F_2^{k\times k}, G_n \in F_2^{n\times n}$ are symmetric and $B_{k,n} \in F_2^{k\times n}$ is full rank. Synthesizing the isometry is equivalent to reducing G to the identity operator

$$\left[\begin{array}{c|cc} 0_k & I_k & 0_{k\times n-k} \\ \hline I_k & \multicolumn{2}{c}{\multirow{2}{*}{$0_{n\times n}$}} \\ 0_{n-k\times k} & & \end{array}\right]$$

with the available operations given in Tables 1 and 2.

Overall, everything acts as if we have a graph state on $n+k$ qubits on which only a subset of operations are available: two-qubit gates are only possible among one set of k qubits or the complementary set of n qubits. Contrary to the graph state synthesis where one wants to remove every edge of the graph, we look for an "identity" layout of the edges where k qubits of the first set are solely connected to one qubit of the other set. This is

Clifford gate	Side	Qubits	Acts on	Action
S	Input	i	G_k	$G_k[i,i] \leftarrow G_k[i,i] \oplus 1$
S	Output	i	G_n	$G_n[i,i] \leftarrow G_n[i,i] \oplus 1$
CZ	Input	i,j	G_k	$G_k[i,j] \leftarrow G_k[i,j] \oplus 1$ $G_k[j,i] \leftarrow G_k[j,i] \oplus 1$
CZ	Output	i,j	G_n	$G_n[i,j] \leftarrow G_n[i,j] \oplus 1$ $G_n[j,i] \leftarrow G_n[j,i] \oplus 1$
$CNOT$	Input	i,j	$G_k, B_{k,n}$	$G_k[i,:] \leftarrow G_k[j,:] \oplus G_k[i,:]$ $G_k[:,i] \leftarrow G_k[:,j] \oplus G_k[:,i]$ $B_{k,n}[i,:] \leftarrow B_{k,n}[j,:] \oplus B[i,:]$
$CNOT$	Output	i,j	$G_n, B_{k,n}$	$G_n[i,:] \leftarrow G_n[j,:] \oplus G_n[i,:]$ $G_n[:,i] \leftarrow G_n[:,j] \oplus G_n[:,i]$ $B_{k,n}[:,i] \leftarrow B_{k,n}[:,j] \oplus B[:,i]$
H/R_x	Input	i	$G_k, G_n, B_{k,n}$	$G_k \leftarrow G_k \oplus gg^T$ $G_n \leftarrow G_n \oplus B_{k,n}[i,:]^T B_{k,n}[i,:]$ $B_{k,n} \leftarrow B_{k,n} \oplus gB_{k,n}[i,:]$ ($g = G_k[:,i]$ if $G_k[i,i] = 0$, $g = G_k[:,i] \oplus e_i$ otherwise)
H/R_x	Output	i	$G_k, G_n, B_{k,n}$	$G_n \leftarrow G_n \oplus gg^T$ $G_k \leftarrow G_k \oplus B_{k,n}[:,i]^T B_{k,n}[:,i]$ $B_{k,n} \leftarrow B_{k,n} \oplus B_{k,n}[:,i]g^T$ ($g = G_n[:,i]$ if $G_n[i,i] = 0$, $g = G_n[:,i] \oplus e_i$ otherwise)

Table 2: An alternative way to present the available operations for the synthesis of a k to n Clifford isometry $G = [G_k\ B_{k,n}; B_{k,n}^T\ G_n]$.

illustrated in Figure 3 with the application of one CZ and one CNOT gate. This form is a generalization of the form derived in [20] but we did not use the ZX-calculus in our proof and we provide a more in-depth characterization of the synthesis framework implied by this graph-state form.

We remind that our framework is not universal in the sense that we are not authorized to apply any Clifford gate, therefore we cannot generate any Clifford circuit. Solving optimally an instance of our framework does not guarantee that we have an optimal implementation of the Clifford isometry.

We now review the possibilities offered by this new framework and propose some algorithms to perform the synthesis of Clifford isometries.

4 Synthesis algorithms

4.1 Normal forms

One advantage of this framework is that normal forms can be derived quite easily with short proofs. We now propose some of them. Notably, we show that we can recover similar structures to some previous normal forms found in the literature. By similar structures we mean that the number and properties of the 2-qubit gate layers are the same, but our own normal forms may differ in the 1-qubit gate layers. Given that the main cost in the implementation of Clifford isometries lie in the execution of the 2-qubit gate layers, we believe that our own normal forms can be assimilated as the ones of the literature despite some differences in the 1-qubit gate layers.

We use the following terminology: -S- corresponds to a layer of S gates, -C- corresponds to a layer of CNOT gates, -CZ- corresponds to a layer of CZ gates and -H- corresponds to a layer of H gates. Given that the gates on the input side are only applied to the first k qubits, and that the gates in the output side can be applied to the whole n qubits, we will note the layers with a subscript to emphasize when it concerns only k or n qubits.

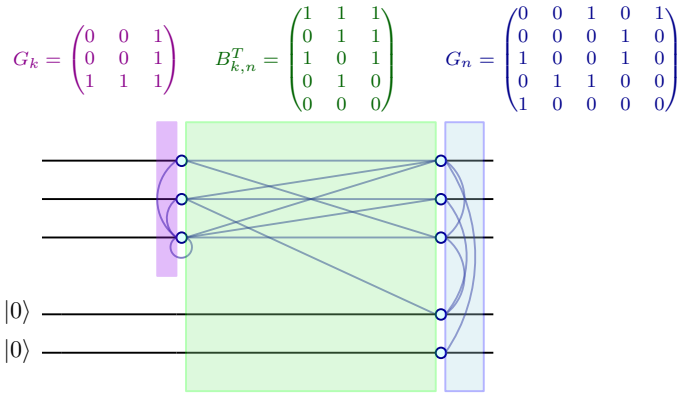
Note that our normal forms always end with a layer -H_n- of Hadamard gates: this layer transforms any isometry into its graph-state form. For each qubit, there may or may not be an Hadamard gate applied. We also always have a full layer -H- of n Hadamard gates "in the middle" of our normal forms that ends the synthesis of our graph-state framework. This layer draws a boundary between the operations on the input qubits and the operations on the output qubits. For clarity we omit the subscript for this layer.

Note also that the layers applied on the output side have to be read in reverse order to understand their action in our graph-state framework.

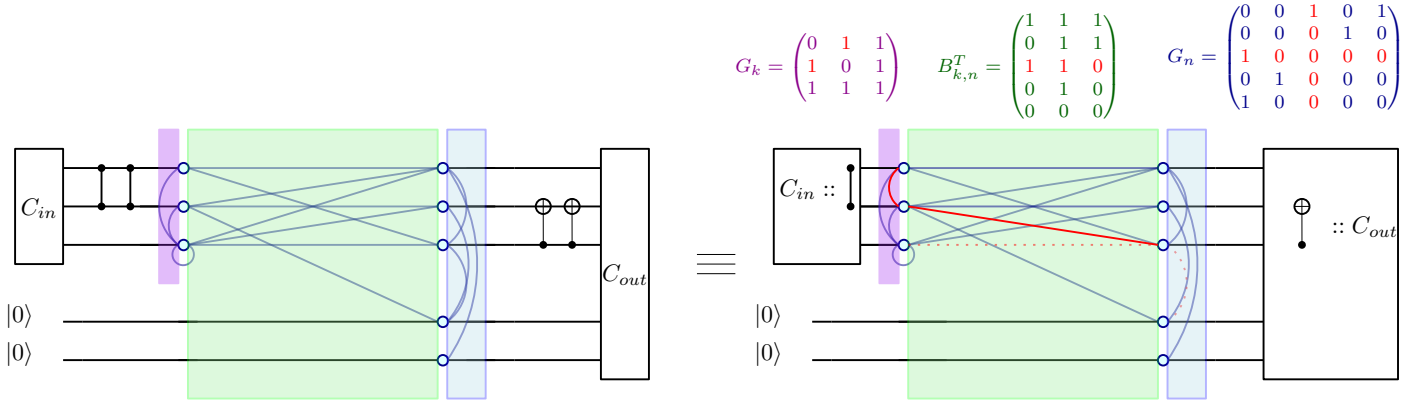
4.1.1 S_k-CZ_k-H-C_n-CZ_n-S_n-H_n.

The layers of -S- and -CZ- gates on the input side zero G_k. The layers -S- and -CZ- on the output side zero G_n. Finally the -C- layer reduces B to the identity. Note that we are free to place these three last layers (CNOT, S and CZ) as we like: this will not change the -C- layer but we have to adjust the -S- and -CZ- layers depending on the value of G_n that can be changed by the action of the CNOT layer. In fact, as long as a suitable phase polynomial is implemented we have a valid implementation of the Clifford isometry.

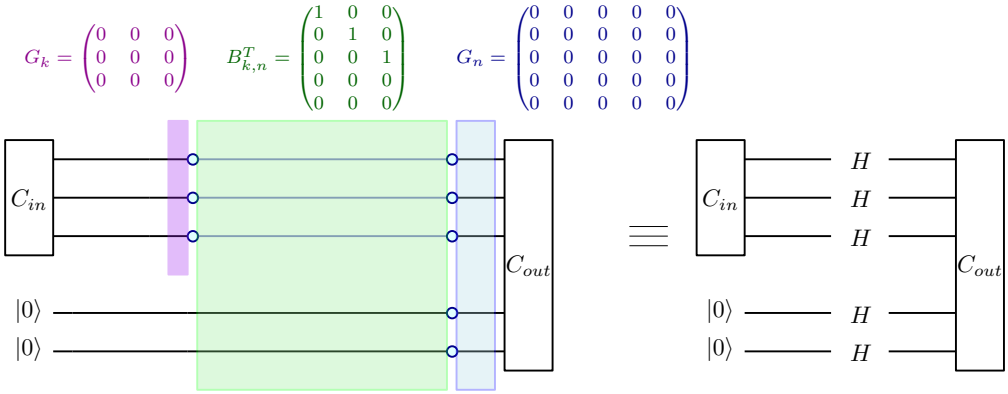
With a full Clifford operator, i.e when $k = n$, we recover the normal form -H-S-CZ-C-H-CZ-S-H- given in [20]. Note that, in this particular case, the -C- layer can be placed before or after the -H- layer: the only difference is that we will act on the B or B^T part of G. We also have one -H- layer less.



(a) Each submatrix $G_n, G_k, B_{k,n}$ corresponds to adjacency relationships between nodes of the graph.



(b) Similarly to the workflow explained in Fig. 2, one can add a gate and its inverse on the input or output side without changing the functionality of the global circuit. One gate is merged to a circuit and the other is absorbed by the graph state which is modified accordingly. In red the entries, rows or columns that are modified by the gate. The corresponding added edges are also in red, removed edges are red and dotted.



(c) Once the identity graph is obtained, the isometry is equivalent to a layer of Hadamard gates and this ends the synthesis.

Figure 3: Illustration of the graph-state framework.

Corollary 2. *Given a k to n Clifford isometry V under graph-state form* $\begin{bmatrix} G_k & B_{k,n} \\ B_{k,n}^T & G_n \end{bmatrix}$, *then, up to Pauli operators,*

$$V = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^k} \sum_{y\in\{0,1\}^n} e^{i\frac{\pi}{2}\left(x^T G_k x + y^T G_n y\right) + i\pi x^T B_{k,n} y} |y\rangle \langle x| \tag{3}$$

Proof. We start from the identity isometry and we check that the normal form gives the expected formula. We write $I_{n,k}$ the $n \times n$ matrix with the first k columns equal to the identity, and 0 everywhere else.

$$\begin{aligned}
\sum_{x\in\{0,1\}^k} |x\rangle |0\rangle^{\otimes n-k} \langle x| &\xrightarrow{CZ+S} \sum_{x\in\{0,1\}^k} e^{i\frac{\pi}{2}x^T G_k x} |x\rangle |0\rangle^{\otimes n-k} \langle x| \\
&\xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^k} \sum_{y\in\{0,1\}^n} e^{i\frac{\pi}{2}x^T G_k x + i\pi \sum_{j=1}^k x_j y_j} |y\rangle \langle x| \\
&\xrightarrow{CNOT} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^k} \sum_{y\in\{0,1\}^n} e^{i\frac{\pi}{2}x^T G_k x + i\pi x^T I_{n,k} y} |By\rangle \langle x| \\
&\xrightarrow{CZ+S} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^k} \sum_{y\in\{0,1\}^n} e^{i\frac{\pi}{2}\left(x^T G_k x + y^T B^T G_n B y\right) + i\pi x^T I_{n,k} y} |By\rangle \langle x| \\
&\xrightarrow{y\leftarrow By} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^k} \sum_{y\in\{0,1\}^n} e^{i\frac{\pi}{2}\left(x^T G_k x + y^T G_n y\right) + i\pi x^T I_{n,k} B^{-1} y} |y\rangle \langle x| \\
&\xrightarrow{B_{k,n}=B^{-1}[1:k,:]=I_{n,k}B^{-1}} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^k} \sum_{y\in\{0,1\}^n} e^{i\frac{\pi}{2}\left(x^T G_k x + y^T G_n y\right) + i\pi x^T B_{k,n} y} |y\rangle \langle x|
\end{aligned}$$

□

4.1.2 S_k-C_k-S_k-H-C_n-S_n-C_n-S_n-H_n.

We can remove any use of -CZ- layers with the use of CNOT gates. Any symmetric boolean matrix M can be written $M = LDL^T \oplus D'$ where L is lower triangular and D, D' are diagonal [7]. This applies to G_k and G_n. Write $G_k = LDL^T \oplus D'$. The first layer of -S- gates on the input side adds D' to G_k such that we get $G_k \oplus D' = LDL^T$. Then, referring to the action of CNOT circuits in Section 3.2, the -C- layer only needs to implement the triangular operator L^{-1} on the output side to reduce $G_k \oplus D'$ to D which is zeroed with a second layer of -S- gates. We have a similar argument to treat the zeroing of G_n. The last layer -C- on the output side ultimately reduces B to the identity.

For the full Clifford operator case, we recover a normal form similar to -C-S-C-S-H-C-S-C-S- given in [19]. In this case each -C- stage implements a triangular operator. This is also true in our own normal form as two -C- blocks are already triangular and the -C- block that reduces B to the identity can be written as a product of two triangular matrices.

4.1.3 Example

We illustrate the two normal forms given above with the concrete example of Fig. 3. The exact circuits are given and for each layer we show its effect on the graph matrix G and how we eventually recover the identity. This is presented in Figures 4 and 5.

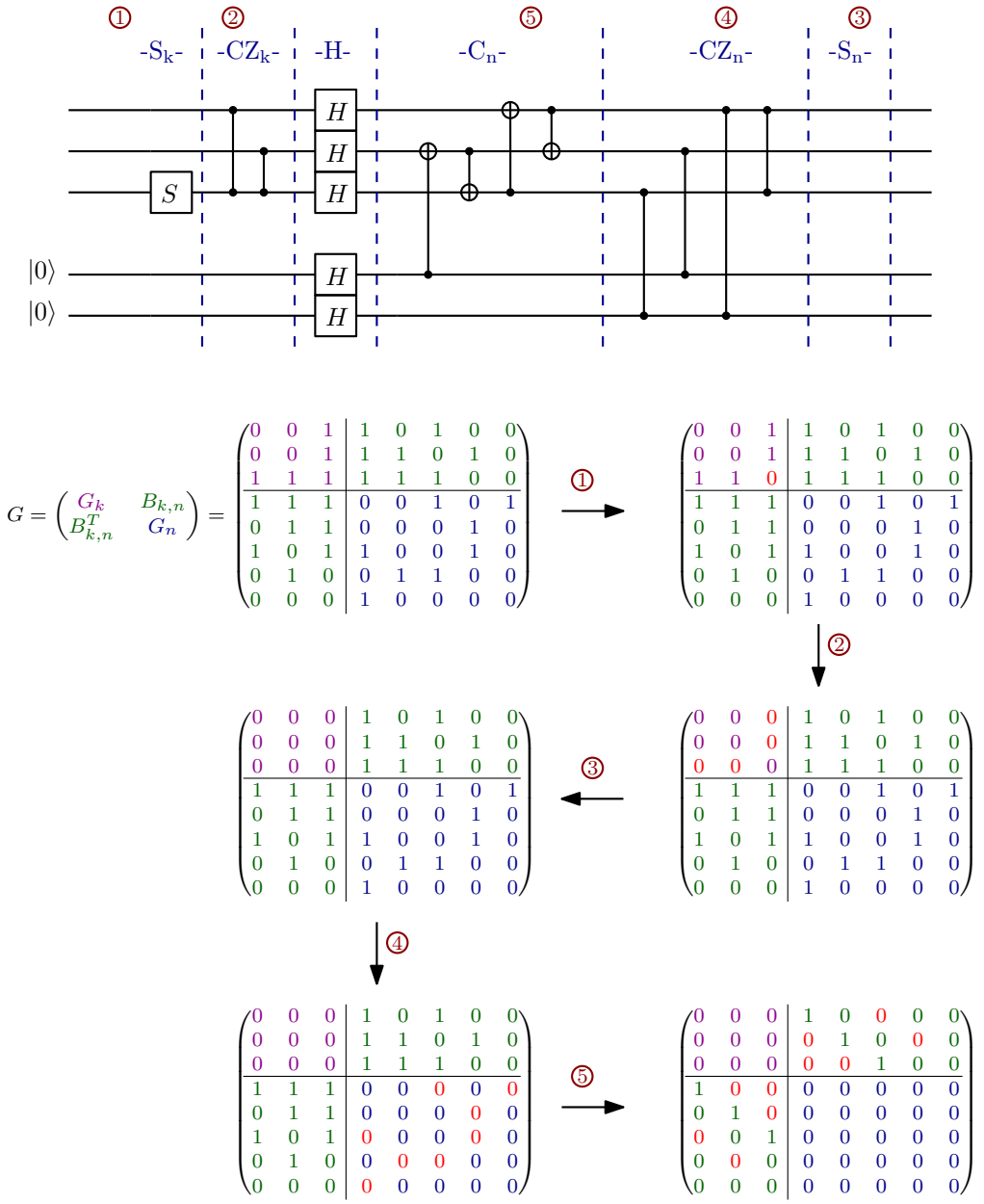


Figure 4: Normal form S_k-CZ_k-H-C_n-CZ_n-S_n-H_n for the Clifford isometry given in Fig. 3. We show how each layer acts on the original graph state matrix to eventually reduce it to the identity. To do so, we recall that we read the circuit on the input side from left to right and the circuit on the output side from right to left, as explained in Figures 2 and 3.

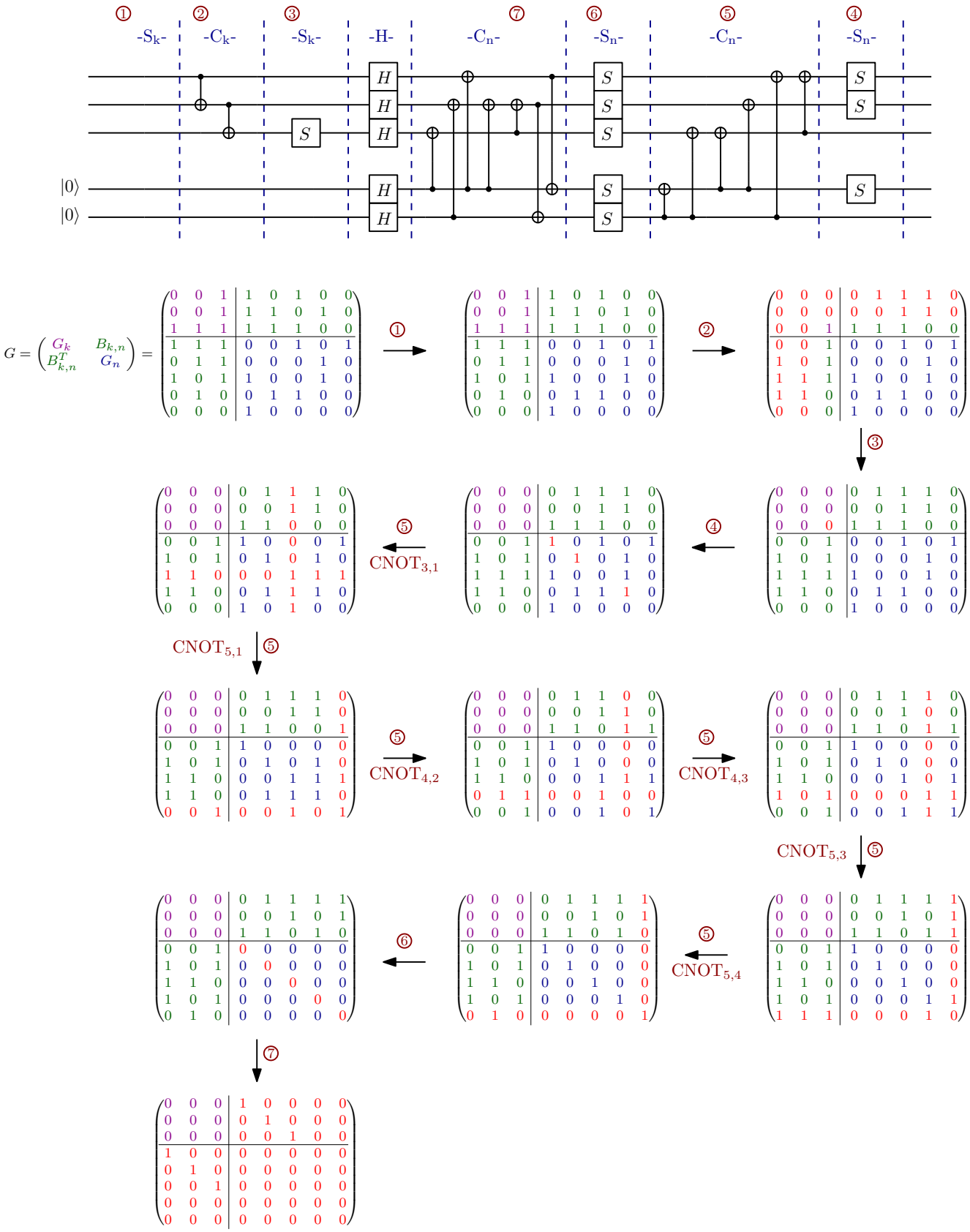


Figure 5: Normal form S_k-C_k-S_k-H-C_n-S_n-C_n-S_n-H_n for the Clifford isometry given in Fig. 3. We show how each layer acts on the original graph state matrix to eventually reduce it to the identity. To do so, we recall that we read the circuit on the input side from left to right and the circuit on the output side from right to left, as explained in Figures 2 and 3. Two -C- layers implement triangular operators. We explicit a step-by-step simulation of the right-most -C- layer to show the effect on both $B_{k,n}$ and G_n.

4.1.4 Other normal forms for Clifford operators.

We show how other normal forms in the literature for Clifford operators can also be derived from our framework.

-C-CZ-S-H-S-CZ-C- [19] This normal form consists in an application of a stage of CNOT gates, then a stage of CZ gates and finally a stage of S stages on both the input and the output side (remember that the operations done on the output side are reversed). Our normal form also has an additional -H- layer at an ultimate layer.

It is straightforward to define suitable operators for each stage:

- the -C- stage on the input side can implement any operator C_1 and the -C- stage on the output side can implement any operator C_2, as long as $C_1 B C_2 = I$,
- the -CZ- and -S- stages on the input side are defined such that it implements the graph state $C_1 G_k C_1^T$,
- the -CZ- and -S- stages on the output side are defined such that it implements the graph state $C_2^T G_n C_2$.

Naturally, as explained in [19], the order of the -C-, -CZ- and -S- stages are free. In fact, as long as a suitable phase polynomial is implemented, then we have a valid implementation of the Clifford operator.

-C-CZ-S-H-CZ-S-H- [22] Similar to the normal form -C-CZ-S-H-S-CZ-C-, this normal form highlights the fact that only one -C- stage is sufficient on one of the two sides. Again, here is a viable construction in our framework:

- reduce B to the identity with the -C- stage,
- reduce the new G_k and G_n to zero with the -CZ- and -S- stages on both input and output sides.

4.1.5 Normal forms for stabilizer states

We explicit some normal forms for the special case of stabilizer states. In the case $k = 0$ the normal forms greatly simplify because we only need to reduce a symmetric matrix G_n to 0 with no input side as all input qubits are in the initial state $|0\rangle$.

-H-$\mathbf{CZ_n}$-$\mathbf{S_n}$-$\mathbf{H_n}$ Stabilizer states are equivalent to graph states up to local Clifford, see e.g [35] for a proof. Given the definition of a graph state the normal form follows. In our framework, the -CZ- and -S- layers zero G_n component by component.

-H-$\mathbf{S_n}$-$\mathbf{C_n}$-$\mathbf{S_n}$-$\mathbf{H_n}$ Using a similar technique: writing $G_n = LDL^T \oplus D'$, we zero G_n by first adding D' with a layer of S gates, then the CNOT layer implements L^{-1} and the second layer of S implements D.

4.2 Synthesis in {CNOT, S, CZ}

Overall, the structure of the normal forms given in Section 4.1 are very similar: two phase polynomials are implemented from either side of the Hadamard layer, the differences being in the way those phase polynomials are implemented. If we restrict the synthesis of G in the gate set {CNOT, S, CZ}, we necessarily implement phase polynomials and we show that only a partial specification of the phase polynomials is sufficient.

Theorem 5. *Let V be a k to n Clifford isometry V under graph-state form specified by its graph matrix $G = \begin{bmatrix} G_k & B_{k,n} \\ B_{k,n}^T & G_n \end{bmatrix}$. An operator of the form*

$$P_1 H^{\otimes n} P_2 \tag{4}$$

where P_1, P_2 are phase polynomials implements V if and only if

$$P_1^\dagger = \sum_{x \in \{0,1\}^n} e^{i\frac{\pi}{2} x^T G_n x} |A_1 x\rangle \langle x|$$

$$P_2 = \sum_{x \in \{0,1\}^k} e^{i\frac{\pi}{2} x^T G_k x} |A_2 x\rangle |0\rangle^{\otimes n-k} \langle x|$$

such that

$$\begin{bmatrix} A_2^T & 0 \end{bmatrix} A_1 = B_{k,n}$$

where A_1 is $n \times n$, A_2 is $k \times k$.

Proof. $P_1 H^{\otimes n} P_2$ is a valid implementation of V if and only if

$$H^{\otimes n} P_2 = P_1^\dagger V.$$

We write

$$P_1^\dagger = \left(\sum_{x \in \{0,1\}^n} e^{i\frac{\pi}{2} x^T \Gamma_1 x} |A_1 x\rangle \langle x| \right),$$

$$P_2 = \left(\sum_{x \in \{0,1\}^k} e^{i\frac{\pi}{2} x^T \Gamma_2 x} |A_2 x\rangle |0\rangle^{\otimes n-k} \langle x| \right),$$

and we expand the two terms:

$$\begin{aligned} P_1^\dagger V &= \left(\sum_{y \in \{0,1\}^n} e^{i\frac{\pi}{2} y^T \Gamma_1 y} |A_1 y\rangle \langle y| \right) \left(\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^k} \sum_{y \in \{0,1\}^n} e^{i\frac{\pi}{2}\left(x^T G_k x + y^T G_n y\right) + i\pi x^T B_{k,n} y} |y\rangle \langle x| \right) \\ &= \frac{1}{\sqrt{2^n}} \left(\sum_{x \in \{0,1\}^k} \sum_{y \in \{0,1\}^n} e^{i\frac{\pi}{2}\left(x^T G_k x + y^T (G_n \oplus \Gamma_1) y\right) + i\pi x^T B_{k,n} y} |A_1 y\rangle \langle x| \right) \\ &= \frac{1}{\sqrt{2^n}} \left(\sum_{x \in \{0,1\}^k} \sum_{y \in \{0,1\}^n} e^{i\frac{\pi}{2}\left(x^T G_k x + y^T A_1^{-1} (G_n \oplus \Gamma_1) A_1^{-1} y\right) + i\pi x^T B_{k,n} A_1^{-1} y} |y\rangle \langle x| \right) \end{aligned}$$

$$\begin{aligned} H^{\otimes n} P_2 &= \frac{1}{\sqrt{2^n}} \left(\sum_{x \in \{0,1\}^n} \sum_{y \in \{0,1\}^n} e^{i\pi x^T y} |y\rangle \langle x| \right) \left(\sum_{x \in \{0,1\}^k} e^{i\frac{\pi}{2} x^T \Gamma_2 x} |A_2 x\rangle |0\rangle^{\otimes n-k} \langle x| \right) \\ &= \frac{1}{\sqrt{2^n}} \left(\sum_{x \in \{0,1\}^k} \sum_{x' \in \{0,1\}^{n-k}} \sum_{y \in \{0,1\}^n} e^{i\pi [A_2 x; x']^T y} |y\rangle \langle A_2 x| \langle x'| \right) \left(\sum_{x \in \{0,1\}^k} e^{i\frac{\pi}{2} x^T \Gamma_2 x} |A_2 x\rangle |0\rangle^{\otimes n-k} \langle x| \right) \\ &= \frac{1}{\sqrt{2^n}} \left(\sum_{x \in \{0,1\}^k} \sum_{y \in \{0,1\}^n} e^{i\pi x^T A_2^T E_k y} e^{i\frac{\pi}{2} x^T \Gamma_2 x} |y\rangle \langle x| \right) \end{aligned}$$

We have the equality if and only if

$$\forall x \in \{0,1\}^k, \forall y \in \{0,1\}^n, e^{i\frac{\pi}{2}\left(x^T G_k x + y^T A_1^{-1}(G_n \oplus \Gamma_1)A_1^{-1}y\right) + i\pi x^T B_{k,n} A_1^{-1} y} = e^{i\pi x^T A_2^T E_k y} e^{i\frac{\pi}{2}x^T \Gamma_2 x}$$

i.e

$$G_k = \Gamma_2,$$
$$G_n = \Gamma_1,$$

and

$$\begin{bmatrix} A_2^T & 0 \end{bmatrix} A_1 = B_{k,n}.$$

□

Corollary 3. *Let $|G\rangle$ be a graph state. Then we have*

$$|G\rangle = \left(\sum_x e^{i\frac{\pi}{2}\cdot x^T G x} |Ax\rangle \langle x|\right)^\dagger H^{\otimes n} |0\rangle^{\otimes n}$$

for any invertible boolean matrix A. In other words, the synthesis of a graph state in the gate set $\{CNOT, CZ, S\}$ is equivalent to the synthesis of a Clifford phase of a phase polynomial given by the graph matrix G without any restriction on the final linear reversible operator applied.

4.3 Synthesis for an LNN architecture

Theorem 6. *Any k to n Clifford isometry can be executed in two-qubit depth $4n + 3k - 2$ on an LNN architecture.*

Proof. Our proof mainly relies on the synthesis of CNOT circuits in depth $5n$ from [36] and CZ circuits in depth $2n - 2$ from [19]. First, we can reduce G to

$$G = \begin{bmatrix} G'_k & B'^T \\ B' & 0 \end{bmatrix}$$

with a CZ circuit of depth $2n - 2$ using the technique from [19]. Initially the circuit in [19] is of depth $2n + 2$ but the last four layers are purely CNOT gates which do not affect the zeroing of G_n, therefore we are free to remove them and keep a circuit of depth $2n - 2$. Then we need to take a look at how the algorithm from [36] works. Given a linear reversible operator A on n qubits they apply two circuits:

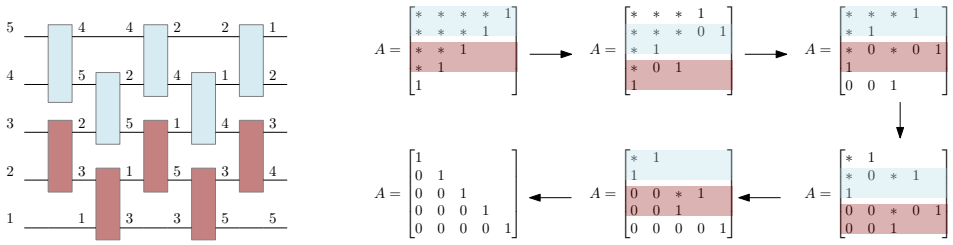
- the first one of depth $2n$, applying an operator C_1 such that $A' = C_1 A$ is northwest triangular. For instance on 5 qubits,

$$A' = C_1 A = \begin{bmatrix} * & * & * & * & 1 \\ * & * & * & 1 & \\ * & * & 1 & & \\ * & 1 & & & \\ 1 & & & & \end{bmatrix}$$

  where $*$ stands for any boolean value, and the entries non specified are 0.

- the second circuit, of depth $3n$, implements an operator C_2 that synthesizes the triangular operator:

$$C_2 A' = I.$$



(a) The sorting network applied to the operator, as a box-based circuit.

(b) The action of the sorting network on the operator for each layer.

Figure 6: Illustration of the second part of the algorithm from [36] to reduce a northwest triangular linear reversible operator to the identity.

Overall, we get $A = C_1^{-1}C_2^{-1}$ as a CNOT circuit of depth $5n$. For the rest of the proof, we need to go into the details of the generation of C_2. An example on 5 qubits is given in Fig. 6. The general idea is to give a label $a_i = n - i + 1, i = 1 \dots n$ to each row of A' and to sort those labels with a LNN compliant sorting network as shown in Fig. 6a. Each box will swap the rows (and the labels) and may perform a row operation between the two swapped rows. Namely if a row with label a_i is swapped with a row of label a_j, $i < j$, the row with label a_j may have a nonzero i-th component and if so we perform the operation

$$a_j \leftarrow a_j \oplus a_i$$

to zero that component. Given the initial label positions, a label a_j will have to "encounter" all labels a_i, $i < j$ during the sorting process and we are ensured that the j-th row will eventually be the j-th canonical vector.

Each box will therefore consist of either a SWAP operation or an operation of the form:

$$(u, v) \leftarrow (u \oplus v, u)$$

or

$$(u, v) \leftarrow (v, u \oplus v).$$

The SWAP operation requires 3 CNOT gates while the SWAP+CNOT requires in fact only 2 CNOT gates [36]. Therefore each box will be of depth at most 3 and the total circuit C_2, of depth n as a box-based circuit, will be of depth $3n$ as a CNOT-based circuit. Note also that, in any case, we always have at some point during the execution of a box on two rows the sum of the values of the two rows that appear on one row of A'. This will be an important property in our proof.

Coming back to our proof, with the first half of the construction from [36] we are able to reduce the CNOT part B' to a northwest triangular matrix B'' in depth $2n$ as well. At this point we have an operator of the form

$$G = \begin{bmatrix} G''_k & \begin{pmatrix} B''^T & 0 \end{pmatrix} \\ \begin{pmatrix} B'' \\ 0 \end{pmatrix} & 0 \end{bmatrix}$$

where B'' is $k \times k$ and northwest triangular.

We now show that we can reduce B'' to the identity and G_k to zero in depth $3k$. In fact, we will show that we can solely insert suitable $R_x(\pi/2)$ gates in the CNOT circuit reducing B'' to the identity matrix (taken from [36]) to reduce G_k to 0 as well.

With G in the form $\begin{bmatrix} G_k & D^T \\ D & 0 \end{bmatrix}$ with arbitrary D, the action of an $R_x(\pi/2)$ gate on the output side is equivalent to a rank-one modification of G_k by a vector u given by one of the row of D. In other words, if we apply an $R_x(\pi/2)$ on qubit j, then

$$G_k \leftarrow G_k \oplus D[j,:]^T D[j,:].$$

Therefore in our case we can reduce G_k to 0 if the set

$$S = \{u^T u | u \text{ appearing in } B'' \text{ during the synthesis } \}$$

is a basis of the set of symmetric boolean matrices.

We prove by induction that S generates the set

$$P_n = \{e_i e_i^T \mid i \in [\![1,n]\!]\} \cup \{e_i e_j^T \oplus e_j e_i^T \mid i,j \in [\![1,n]\!], i \neq j\}$$

and therefore generates the set of symmetric boolean matrices.

This is trivial for $n = 1$. Now suppose this is true for some n. We introduce similar notations to the ones in [36]:

- each line i of B'' has a value c_i, i.e a boolean vector of size $n+1$,
- each line i of B'' carries a label $a_i \in [\![1, n+1]\!]$, independent of c_i.

For simplicity, at any moment during the algorithm from [36], we now refer as parity k the line of B'' that carries the label a_k, and by extension the value of the line carrying the label a_k. The key is to see the algorithm from [36] as moving parities in B'' and during the SWAP of the parities we add or not some parities to the others. For simplicity we also consider the reverse circuit, i.e, the one that transforms the identity operator I to B''. This does not change the set S but this will simplify the reasoning. We still refer as B'' the matrix at any point during the algorithm.

The first point is to notice that the $(n+1)$-th parity will never be added to the other parities. So we can conclude that the first n parities, apart from their interactions with the $(n+1)$-th parity, will behave exactly as in the case with n qubits, therefore they satisfy the property and generates the set P_n.

We only need to show that, with the new values obtained on the last parity, we can generate P_{n+1}, i.e, we need to generate the set $\{e_{n+1}e_{n+1}^T\} \cup \{e_{n+1}e_j^T \oplus e_j e_{n+1}^T \mid j \in [\![1,n]\!]\}$.

Every value carried by the $(n+1)$-th parity is of the form

$$e_{n+1} \oplus u_k$$

for a family of boolean vectors $(u_k)_k$ of size $n+1$ with the $(n+1)$-th component always equal to 0. Applying an R_x in two occasions gives the transformation

$$A \leftarrow A \oplus (e_{n+1} \oplus u_i)(e_{n+1} \oplus u_i)^T \oplus (e_{n+1} \oplus u_j)(e_{n+1} \oplus u_j)^T$$

$$A \leftarrow A \oplus u_i u_i^T \oplus u_i e_{n+1}^T \oplus e_{n+1} u_i^T \oplus u_j u_j^T \oplus u_j e_{n+1}^T \oplus e_{n+1} u_j^T$$

$$A \leftarrow A \oplus \underbrace{u_i u_i^T \oplus u_j u_j^T}_{\in P_n} \oplus (u_i \oplus u_j) e_{n+1}^T \oplus e_{n+1}(u_i \oplus u_j)^T$$

This means that, up to terms in P_n that can be removed with suitable R_x gates on the first n parities, we can generate the set $e_{n+1}(\oplus_k \alpha_k u_k)^T \oplus (\oplus_k \alpha_k u_k) e_{n+1}^T$ for any $\alpha_k \in \{0,1\}$, i.e, any linear combination of the (u_k)'s. What remains is to show that the family of vectors $(u_k)_k$ is a basis such that we can generate all the e_i's and this concludes the proof.

We show that any new u_p appearing during the process is not a linear combination of the previous $(u_k)_{k=1\ldots p-1}$, given that we have n such u_k we get a basis $(u_k)_{k=1\ldots n}$. By construction of the algorithm the $(n+1)$-th parity is initially on the last row of B'' and is equal to e_{n+1}. During the synthesis this parity will be swapped, and sometimes added, with the parity "above".

In other words, if the $(n+1)$-th parity is currently on line k of B'', we have

$$c_k = e_{n+1} \oplus v_{k-1}$$

where v_{k-1} is necessarily a linear combination of the set of parities located lines $k+1 \ldots n$, i.e, the lines "below". The family $(u_j)_{j=1\ldots k-1}$ also consists in linear combinations of the set of parities located lines $k+1 \ldots n$. During the swap/addition with the parity above we necessarily create the value

$$e_{n+1} \oplus v_{k-1} \oplus c_{k-1}$$

and we set $u_k = v_{k-1} \oplus c_{k-1}$. u_k is necessarily linearly independent with the previous $(u_j)_{j=1\ldots k-1}$ otherwise the lines $k+1$ to n and the $k-1$-th line of B'' are not linearly independent which is not possible by invertibility of B''.

We set $v_k = v_{k-1}$ or $v_k = v_{k-1} \oplus c_{k+1}$, depending on the value of the $(n+1)$-th parity of B'' and we pursue the algorithm. □

Corollary 4. *Any n-qubit Clifford operator can be executed in two-qubit depth $7n-2$ on an LNN architecture.*

A similar result was independently found in [34]. They first provided a circuit of depth $7n+2$ which was then improved to $7n-4$. Note that in our proof we showed it is possible to work exclusively in the output side by using {CNOT+S} (to zero G_n) then {CNOT+Rx} circuits (to reduce $B_{k,n}$ to the identity while zeroing G_k). We slightly deviated from the need to implement two phase polynomials on both sides which gives, in our opinion, good insights about the variety of techniques that can be used in Clifford circuits synthesis.

5 Practical algorithms for graph state synthesis and application to the codiagonalization of Pauli rotations

We give two algorithms for the synthesis of graph states, a special case of Clifford isometry where $k = 0$. One algorithm optimizes the number of two-qubit gates while the second one optimizes the depth of the circuit, both assuming an all-to-all connectivity between the physical qubits. Then we use our algorithms in quantum compilers that synthesize a sequence of Pauli rotations. Usually such sequences are implemented by packing the rotations into groups of commuting operators, each group is then codiagonalized with a Clifford circuit and implemented with the synthesis of a phase polynomial. An efficient graph-state synthesis algorithm will improve the codiagonalization part of this process. Then in Section 6 we will extend the algorithms for graph state synthesis to deal with the case of any arbitrary Clifford isometry.

5.1 Practical algorithms: a variant of the syndrome decoding based algorithm and a depth-optimized greedy algorithm

Following our framework, G_k is a 0×0 matrix and $B_{k,n}$ a $0 \times n$ matrix. Thus, the synthesis only consists in reducing a symmetric $n \times n$ boolean matrix G_n to the zero matrix with the following operations:

- flip entries $G_n[i, j]$ and $G_n[j, i]$,
- add row i to row j and column i to column j, $i \neq j$,
- rank-one updates as described in Table 1.

We will restrict ourselves to the gate set $\{CNOT, CZ, S\}$ which means that the rank-one updates are not available. The algorithms are variants of two algorithms designed by some of the authors for the synthesis of CNOT circuits [37, 38, 39]. We present a sketch of how the algorithms work by highlighting the differences with CNOT circuits but more details can be found in the original articles.

5.1.1 The syndrome decoding based algorithm [37, 38].

This algorithm aims at optimizing the total number of two-qubit gates in the circuit. The algorithm is recursive, suppose we managed to reduce G_n to

$$G_n = \begin{bmatrix} 0_{n-1} & s^T \\ s & \star \end{bmatrix}$$

with a sequence of operations C acting on the first $n-1$ qubits where 0_{n-1} is the $n-1 \times n-1$ zero matrix, s is an arbitary vector of size $n-1$ and $\star$ can be 0 or 1. We show how to efficiently add elementary operations to C to create a new sequence C' such that its execution will zero s while keeping the $n-1 \times n-1$ submatrix equal to 0. $G[n, n]$ can be ultimately zeroed with an S gate and does not participate in the increase of the 2-qubit gate count. Overall the algorithm consists in:

- a recursive call on a submatrix of G_n of size $n-1$, giving a sequence C,
- and a call to the procedure that will treat separately the last line and column of G_n.

One key of the algorithm is to notice that at any time during the execution of C we are free to do the following operations:

- flip any entry of s with a CZ gate, by symmetry this will flip the same entry in s^T,
- add any row $1 \leq i \leq n-1$ of the current value of G_n to s with a CNOT gate, by symmetry the column operation will also modify s^T,
- flip any diagonal entry i of $G[1 : n-1, 1 : n-1]$ with an S gate, then add row i to s with a CNOT gate and flip again the $i-th$ diagonal entry.

These operations will only modify s in G_n and keep $G_n[1 : n-1, 1 : n-1]$ unchanged and therefore ultimately 0. All these operations have a two-qubit cost of 1. Overall any of these three operations we can be seen as adding a vector to s. Therefore, the procedure to zero s consists in two steps:

1. scan the matrix $G[1 : n-1, 1 : n-1]$ during its zeroing and store all the vectors that can be added to s in a matrix H. H is eventually $m \times (n-1)$ where m is the number of different parities encountered,

2. solve $xH = s$ with x of smallest Hamming weight possible. Each vector to add costs one two-qubit gate so the smallest the Hamming weight of x, the smallest the number of extra two-qubit gates.

Step 2 is also known as the syndrome decoding problem and we use a greedy method to solve it: we choose the vector v in H that minimizes the Hamming weight $v \oplus s$, we set $s \leftarrow s \oplus v$ and we repeat this until s is 0. The presence of the canonical vectors guarantees that the procedure eventually terminates. To improve the efficiency of the greedy method, it is possible to solve $xHP = sP$ for different invertible matrices P and keep the best result. This method is known as the Information Set Decoding strategy and significant improvements can be obtained if sufficient iterations are done. The only constraint on P is that the canonical vectors must be present in HP.

Once the syndrome decoding problem is solved, we get a sequence of vectors, each corresponding to a specific operation (i.e, its type – CNOT, CZ or S+CNOT+S – and the index where it should be inserted inside C to perform the desired operation). To create the new sequence of C' we just need to update C by inserting accordingly the operations.

5.1.2 A depth-optimized greedy algorithm.

We use the following greedy algorithm to optimize the depth. Until G is zero repeat the following procedure:

1. initialize an empty list L of used qubits and an empty set of operations O.

2. At each iteration choose the row or column operations among the non-used qubits that reduces the Hamming weight of G_n the most. If the decrease in the Hamming weight is strictly larger than 1, then perform the operation, add it to O and add the two qubits to L.

3. When no such operation is available, compute the set

$$Q = [\![1, n]\!] \setminus L,$$

   and consider the submatrix $G_n[Q, Q]$: this is the submatrix on which entries can still be flipped without increasing the depth of O. To compute the largest number of entries that can be flipped in depth 1 see $G_n[Q, Q]$ as the adjacency matrix of a graph. Each 1 entry in $G[Q, Q]$ is an edge in the graph. Then a maximum matching in this graph gives a set of maximum size of entries that can be flipped.

4. One step of this procedure produces a set O of depth 1. Add O to the full set of operations done on G_n.

We could have avoided the use of O in the description of our algorithm but we wanted to highlight the principle of the algorithm: at each step we compute a set of operations that reduces the most the number of 1 in G_n and that can be executed in depth 1. The algorithm always terminates as at each step we strictly reduce the number of nonzero elements in G_n.

5.2 Application to the codiagonalization of Pauli rotations

Given a set of k Pauli operators on n-qubit $\{P_1, P_2, \ldots, P_k\}$, if they all commute then there exists a Clifford operator C such that

$$CP_iC^\dagger = Z^{a_i} = Z_1^{a_{i,1}} \otimes Z_2^{a_{i,2}} \otimes \ldots \otimes Z_n^{a_{i,n}} \forall i \in [\![1, k]\!], a_i \in \mathbb{F}_2^n.$$

C is not unique and we can look for an optimized one using our graph synthesis framework. If we write our Pauli operators as a tableau

$$\begin{bmatrix} Z \\ X \end{bmatrix} \in \mathbb{F}_2^{n \times k},$$

the codiagonalization problem is then equivalent to finding a Clifford operator that will zero the X part. We propose the following procedure:

1. considering the following block structure of our tableau

$$\begin{bmatrix} Z_k \\ Z_{n-k} \\ X_k \\ X_{n-k} \end{bmatrix} \in \mathbb{F}_2^{n \times k}$$

   where Z_k, X_k are $k \times k$ and Z_{n-k}, X_{n-k} are $(n-k) \times k$. Find a circuit of Hadamard gates and SWAP gates such that X_k is full rank.

2. find a Clifford operator that zeroes X_{n-k}.

3. with free column operations reduce X_k to the identity operator,

4. use the graph-state synthesis framework to zero Z_k,

5. use Hadamard gates on the first k qubits to commute X_k with the zero block

At the end of the procedure, we eventually have $X = 0$. The main complexity of this procedure lies in steps 2 and 4. The Hadamard layers do not increase either the 2-qubit gate count or the 2-qubit gate depth. The SWAP gates can be transfered at the end of the circuit and post-processed. Although the column operations during step 3 are free, they change the final set of diagonalized Pauli rotations. This does not change the fact that the resulting Clifford circuit will codiagonalize our initial set of Pauli rotations but, once such a Clifford is computed, we need to reverse the column operations to get back the actual diagonal Pauli operators. Alternatively, one can execute the Clifford operator again on the initial set of Pauli rotations.

We already have algorithms for step 4, we now give methods to do step 2 as well. Those methods are variants of the ones we already used for graph state synthesis so we will address them quickly.

5.2.1 2-qubit count optimization with the syndrome decoding problem.

By assumption X_k is full rank. With suitable column operations we can have $X_k = I$. We zero each row of X_{n-k} one by one starting from row 1 to row $n-k$. During the i-th step we compute the rows that appeared in the zeroing of $X_{n-k}[1 : i-1]$ that we gather in a matrix H. We also add the canonical vectors given in X_k. We similarly solve the syndrome decoding problem $Hx = X_{n-k}[i, :]$ to have the smallest set of rows from H one

has to add to $X_{n-k}[i,:]$ to zero the row. Those row operations can be performed with suitable CNOT gates inserted in the current Clifford circuit. Note that if we apply an Hadamard gate, resp. an $R_x(\pi/2)$ gate, on the $k+i$-th qubit, the row to zero becomes $Z_{n-k}[i,:]$, resp. $X_{n-k}[i,:] \oplus Z_{n-k}[i,:]$: we can solve the syndrome decoding problem with these three possibilities and keep the best result at the cost of adding one-qubit gates.

5.2.2 Depth optimization with a greedy procedure.

This case is even closer to the CNOT case from [39] than the graph-state case. Consider the matrix

$$B = X_{n-k}X_k^{-1}.$$

Any row operation on X_{n-k} is equivalent to a row operation on B, any row operation on X_k is equivalent to a column operation on B and adding a row of X_k to a row of X_{n-k} is equivalent to shifting an entry of B. Until B is zero repeat the following procedure:

1. initialize two sets of non-used qubits $L_1 = [\![1,k]\!], L_2 = [\![k+1,k+n]\!]$,
2. compute the row or column operation that reduces at most the number of 1 in B if such row or column operation involves only qubits in L_1 or L_2, perform this operation and remove the two used qubits from either L_1 or L_2,
3. when no more row or column operation can be done consider the submatrix $B[L_2, L_1]$, this is the adjacency matrix of a bipartite graph G. Compute a maximum matching in G that will give you the set of row operations to perform between X_k and X_{n-k}.

One iteration of the three steps above gives a CNOT circuit of depth 1 that tries to reduce at most the number of one entries in B. The procedure always terminates as we can always remove the entry one by one during step 3 of the procedure above.

5.2.3 Extra-optimization.

Note that if the number of rotations k is close to n, it might be preferable to complete the set of Pauli operators into n operators that commute and directly apply the graph state synthesis on n qubits. In the following benchmarks we will always try the two approaches and keep the best results. We also do the codiagonalization multiple times with random qubits permutation at the beginning of the algorithm, again we keep the best result and we post-processed the selected permutation.

6 Practical algorithms for Clifford isometry synthesis

We reuse the methods used in Section 5 to deal with the case of general Clifford isometries given by matrices of the form $\begin{pmatrix} G_k & B_k \\ B_k^T & G_n \end{pmatrix}$. Again we restrict ourselves to the gate set {CNOT,CZ,S}. Essentially, we show how to zero both G_n and G_k using the graph state synthesis techniques while reducing B_k to the identity.

6.1 Count optimization with the syndrome decoding problem

For any $n \times k$ matrix A, there exists an $n \times n$ permutation matrix P, an $n \times k$ lower triangular operator L and a $k \times k$ upper triangular matrix U such that

$$A = PLU.$$

We use this so-called LU decomposition on B_k^T to split our synthesis in three steps:

1. first, deal with the permutation matrix P such that we get a matrix of the form
$$\begin{pmatrix} G_k & (LU)^T \\ LU & G_n \end{pmatrix},$$
2. then, reduce G_n to 0 while reducing L to the identity with operations on the output side, this would give us an intermediate matrix of the form
$$\begin{pmatrix} G_k & U^T \\ U & 0 \end{pmatrix},$$
3. finally, reduce G_k to 0 while reducing U to the identity with operations on the input side to get the identity
$$\begin{pmatrix} 0 & I_k^T \\ I_k & 0 \end{pmatrix}.$$

Once this is done we have a valid implementation of the isometry.

Dealing with the permutation. To deal with P, one can post-process it to the end of the circuit. If for some reason the permutation cannot be postponed, one can find a CNOT circuit on the output side such that the resulting B_k has an LU decomposition with $P = I$. This is possible if and only if all its leading principal minors are nonzero (i.e, $B_k[:i, :i]$ is invertible for every $i = 1...k$). Therefore, for all $i = 1...k$, assuming all $B_k[:j, :j]$ are invertible for $j < i$, one can look for a CNOT gate between qubit i and a qubit $i_2 > i$ to make $B_k[:i, :i]$ invertible. This is always possible as B_k is of full rank. This procedure only requires at most k additional CNOT gates.

Graph state + linear reversible triangular operator synthesis. The last two steps essentially reduce to the same problem. In the case of step 2 we are given a $n \times (n + k)$ matrix of the form
$$\begin{pmatrix} L & G_n \end{pmatrix}$$
where L is lower triangular and G_n symmetric. The goal is to reduce it to
$$\begin{pmatrix} I_k & 0 \end{pmatrix}$$
with the following available operations:

- CZ and S gates will flip any entry of G_n,
- CNOT gates will act as row and column operations on G_n while only acting as row operations on L.

Step 3 is dealt in a similar way on $k \times 2k$ matrices of the form
$$\begin{pmatrix} G_k & L \end{pmatrix}$$
where L is $k \times k$.

We only show how to deal with step 2. A recursive formulation similar to the one presented in Section 5 works. If we managed to deal with the first $n-1$ lines we end up with a matrix of the form

$$\begin{pmatrix} I_{k-1} & 0_{(n-1)} \\ s' & s \end{pmatrix}$$

where s' is of size $\min(k, n)$ and s is of size n. To complete the synthesis, we need to zero $s[1:n-1]$ and s' (unless $k = n$, in this case s' needs to be reduced to the canonical vector e_n therefore only $s'[1:n-1]$ needs to be zeroed). We can just stack s and s' and get a larger syndrome decoding problem instance $xH = [s's]$ to solve where the vectors in H are also concatenation of rows in L and rows in G_n. We are guaranteed that the $[s's]$ can be zeroed by a greedy process:

- the canonical vectors for the s part are available through the CZ gates,
- the final matrix form $\begin{pmatrix} I_{k-1} & 0_{(n-1)} \\ s' & s \end{pmatrix}$ that we get after having executed all the elementary operations on the first $n-1$ lines ensure that the canonical vectors for the s' part are accessible through CNOT gates.

Again, we can improve the performance of the syndrome decoding solver by solving it with different changes of basis.

6.2 Depth optimization

The algorithm for depth optimization proceeds in a similar fashion that the count optimization one. Again, we apply an LU decomposition to B_k^T and we follow the same three steps as explained in Section 6.1.

Dealing with the permutation. It is known that any permutation can be implemented in depth $d = 6$ in an all-to-all architecture [40].

Graph state + linear reversible triangular operator synthesis. To reduce a $n \times (n+k)$ matrix of the form

$$\begin{pmatrix} L & G_n \end{pmatrix}$$

with L lower triangular to

$$\begin{pmatrix} I_k & 0 \end{pmatrix}$$

we follow the same greedy procedure as in Section 5.1.2 for regular graph state synthesis. The only difference is that we have to keep the triangular structure of L in order to reduce it to the identity matrix. Essentially this reduces to only selecting CNOT gates between qubits such that row operations $L[i,:] \leftarrow L[i,:] \oplus L[j,:]$ with $i > j$ are performed if $i \leq k$. Note that there are no constraints on the rows $k+1 \dots n$ of L as this part is not triangular.

We are ensured that the greedy process can terminate because CZ and S gates can always be used to strictly decrease the Hamming weight of G_n, then CNOT gates will eventually reduce L to I_k because of the triangular structure of L. The first column of L can always be reduced to 0 using the first line of L, the second column with the second row and so forth.

The reduction of the $k \times 2k$ matrix

$$\begin{pmatrix} G_k & L \end{pmatrix}$$

is done in a similar way. Here the constraints apply to all the rows of the $k \times k$ matrix L.

7 Benchmarks

We implemented the different algorithms of Section 6 in a python-binded Rust crate called *Rustiq*[2]. The library offers a straightforward python interface to call the various synthesis algorithms (and more) via basic python types. In order to evaluate the performances of the algorithms introduced in section 5, we ran benchmarks over both random and structured instances. Results are compared with the co-diagonalization heuristics described in [41] and with depth and count oriented Clifford synthesis heuristics from [31, 42].

7.1 Co-diagonalization and stabilizer state synthesis

Random instances: In the first benchmark, we simply pick some Clifford Tableaux uniformily at random and co-diagonalize their Z outputs. In other words, given some Clifford operator C, we use our heuristics to co-diagonalize operators $\{CZ_iC^\dagger, \forall i \in [1, n]\}$. We then compare the entangling gate count and entangling depth of the output circuits. Results are reported in Figures 7, 8, 9, 10. For full stabilizer state synthesis, we can observe that the Syndrome algorithm asymptotically outperforms the state-of-the-art method in both entangling gate count and entangling depth. Interestingly, so does the Matching algorithm, while it only aims at limiting the circuit's depth without being conservative in the entangling gate count. When using our heuristics to co-diagonalize incomplete set of rotations (Figure 9 and 10), the Syndrome algorithm performs similarly to the state-of-the-art heuristic until some break-point where it stagnates and thus starts outperforming it. This behavior can be observed in both entangling count and depth. The Matching algorithm is outperformed by the other two in entangling count until it also starts stagnating and ends up outperforming the state-of-the-art.

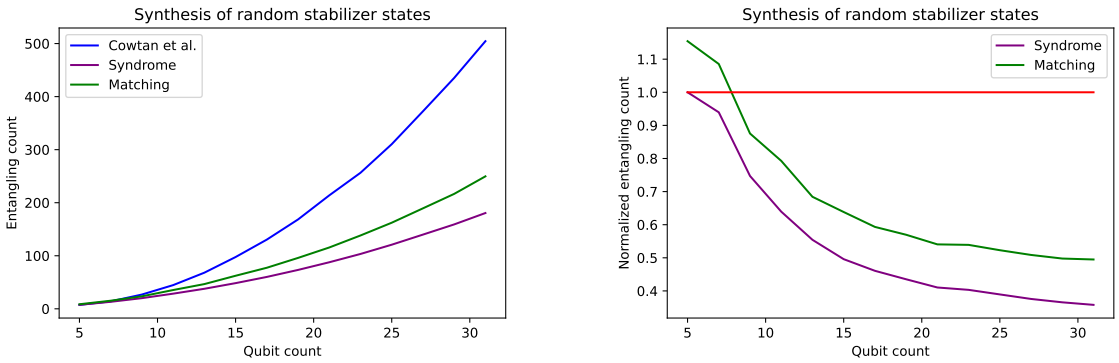


Figure 7: Left: Entangling count (as in CNOT equivalent count) as a function of the number of qubits. Each point is generated by averaging the entangling count of 40 random Clifford tableaux. Right: the same data, but normalized by the entangling count obtained via the state-of-the-art method of [41].

Structured instances: In order to assess the performances of these algorithms over interesting use cases, we embedded them into the meta-heuristics presented in [41] that tackles synthesis of sequences of Pauli rotations (up to arbitrary commutation of these rotations).

[2] https://github.com/smartiel/rustiq

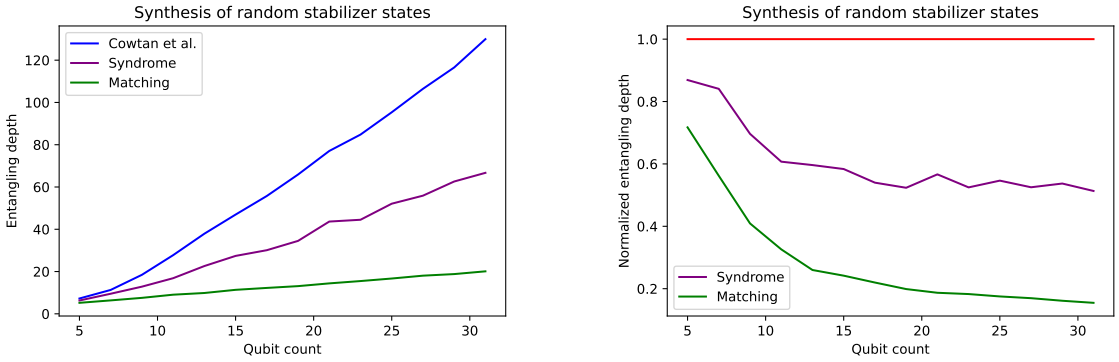


Figure 8: Left: Entangling depth (as in CNOT equivalent depth) as a function of the number of qubits. Each point is generated by averaging the entangling depth of 40 random Clifford tableaux. Right: the same data, but normalized by the entangling depth obtained via the state-of-the-art method of [41].

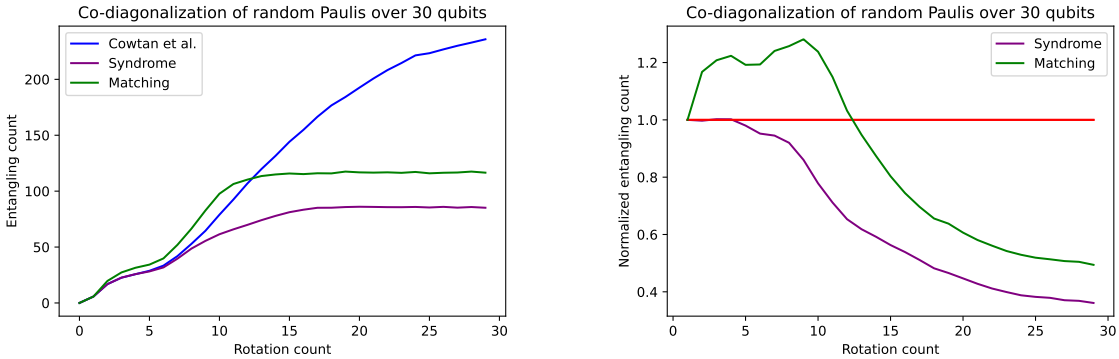


Figure 9: Left: Entangling count (as in CNOT equivalent count) as a function of the number of rotations in a commuting group of rotations over 30 qubits. Each point is generated by averaging the entangling count of 40 random rotation groups. Right: the same data, but normalized by the entangling count obtained via the state-of-the-art method of [41].

In this meta-heuristic, Pauli rotations are first into buckets of pairwise commuting rotations. Then, rotations in each bucket are co-diagonalized (using any co-diagonalization routine), and finally synthesized via a phase polynomial synthesis routine (GraySynth from [25] in the formulation of [41]). Our two algorithms, together with the co-diagonalization routine from [41] effectively gives us three different algorithms to synthesize sequences of Pauli rotations. We ran these algorithm on the UCCSD (Unitary Coupled-Cluster Singles and Doubles) instances provided in [41]. Once again, we measured the entangling gate count and depth of the resulting co-diagonalization circuits produced during the synthesis, and reported them in Tables 3,4,5.

As expected, the Syndrome algorithm systematically outfperforms the Cowtan et al. approach. For large molecules, it reaches up to a 10% CNOT count reduction. Interestingly, on these particularly structures instances, the Matching algorithm manages to significantly reduce the circuit depth (up to 71% depth reduction) while only adding 1% CNOT gates. This opens up the possibility of generating UCCSD Ansätze with a significatively lower CNOT depth than the one proposed in [41], as long as one can also improve the entangling depth of the phase polynomials in between the co-diagonalization circuits.

				Cowtan et al.		Syndrome				Matching			
mol.	orbitals	enc.	basis	count	depth	count	rel. count	depth	rel. depth	count	rel. count	depth	rel. depth
H2	cmplt	P	sto3g	0	0	0	0%	0	0%	0	0%	0	0%
H2	cmplt	BK	sto3g	0	0	0	0%	0	0%	0	0%	0	0%
H2	cmplt	JW	sto3g	8	4	8	-0.00%	4	-0.00%	8	-0.00%	4	-0.00%
H2	cmplt	P	631g	50	46	46	-8.00%	38	-17.39%	50	-0.00%	38	-17.39%
H2	cmplt	BK	631g	50	48	50	-0.00%	48	-0.00%	52	4.00%	40	-16.67%
H4	cmplt	BK	sto3g	66	38	64	-3.03%	40	5.26%	68	3.03%	40	5.26%
H2	cmplt	JW	631g	68	64	68	-0.00%	64	-0.00%	68	-0.00%	46	-28.12%
H4	cmplt	P	sto3g	76	50	76	-0.00%	50	-0.00%	88	15.79%	62	24.00%
LiH	frz	P	sto3g	98	90	92	-6.12%	82	-8.89%	98	-0.00%	74	-17.78%
H4	cmplt	JW	sto3g	98	50	98	-0.00%	48	-4.00%	98	-0.00%	36	-28.00%
LiH	frz	JW	sto3g	120	114	118	-1.67%	110	-3.51%	120	-0.00%	80	-29.82%
LiH	frz	BK	sto3g	134	122	128	-4.48%	106	-13.11%	142	5.97%	90	-26.23%
NH	frz	JW	sto3g	276	176	276	-0.00%	160	-9.09%	276	-0.00%	112	-36.36%
NH	frz	P	sto3g	334	242	330	-1.20%	206	-14.88%	368	10.18%	178	-26.45%
H2O	frz	JW	sto3g	428	236	428	-0.00%	218	-7.63%	428	-0.00%	164	-30.51%
NH	cmplt	JW	sto3g	428	236	428	-0.00%	218	-7.63%	428	-0.00%	164	-30.51%
LiH	cmplt	JW	sto3g	440	220	440	-0.00%	208	-5.45%	440	-0.00%	160	-27.27%
LiH	cmplt	P	sto3g	480	302	470	-2.08%	266	-11.92%	532	10.83%	248	-17.88%
NH	frz	BK	sto3g	486	426	452	-7.00%	274	-35.68%	524	7.82%	196	-53.99%
CH2	frz	JW	sto3g	496	232	496	-0.00%	206	-11.21%	516	4.03%	136	-41.38%
H2	cmplt	P	ccpvdz	520	510	484	-6.92%	446	-12.55%	522	0.38%	374	-26.67%
LiH	frz	P	631g	520	510	496	-4.62%	464	-9.02%	522	0.38%	374	-26.67%
H2O	frz	P	sto3g	526	368	516	-1.90%	312	-15.22%	582	10.65%	274	-25.54%
NH	cmplt	P	sto3g	526	368	512	-2.66%	298	-19.02%	582	10.65%	278	-24.46%
LiH	frz	JW	631g	554	528	554	-0.00%	526	-0.38%	556	0.36%	376	-28.79%
H2	cmplt	JW	ccpvdz	554	528	554	-0.00%	528	-0.00%	556	0.36%	376	-28.79%

Table 3: Entangling count and depth of the co-diagonalization circuits produced by our algorithms (Syndrome and Matching) and the state-of-the-art routine (Cowtan et al.). For the Syndrome and Matching algorithms, we additionally display the improvement (or degradation) relative to the state of the art.

				Cowtan et al.		Syndrome				Matching			
mol.	orbitals	enc.	basis	count	depth	count	rel. count	depth	rel. depth	count	rel. count	depth	rel. depth
H2	cmplt	BK	ccpvdz	646	596	622	-3.72%	540	-9.40%	654	1.24%	398	-33.22%
LiH	frz	BK	631g	646	596	620	-4.02%	534	-10.40%	656	1.55%	400	-32.89%
H2O	cmplt	JW	sto3g	776	490	774	-0.26%	452	-7.76%	790	1.80%	322	-34.29%
CH2	frz	P	sto3g	778	512	766	-1.54%	432	-15.62%	870	11.83%	332	-35.16%
NH	cmplt	BK	sto3g	898	658	846	-5.79%	484	-26.44%	978	8.91%	380	-42.25%
H2O	frz	BK	sto3g	898	658	846	-5.79%	500	-24.01%	978	8.91%	378	-42.55%
LiH	cmplt	BK	sto3g	948	652	904	-4.64%	518	-20.55%	1074	13.29%	380	-41.72%
CH2	cmplt	JW	sto3g	1010	550	1004	-0.59%	464	-15.64%	1082	7.13%	336	-38.91%
H4	cmplt	JW	631g	1024	524	1024	-0.00%	488	-6.87%	1024	-0.00%	376	-28.24%
H2O	cmplt	P	sto3g	1084	734	1030	-4.98%	594	-19.07%	1188	9.59%	496	-32.43%
CH2	frz	BK	sto3g	1094	710	1040	-4.94%	540	-23.94%	1188	8.59%	362	-49.01%
H4	cmplt	P	631g	1134	690	1118	-1.41%	606	-12.17%	1228	8.29%	552	-20.00%
H4	cmplt	BK	631g	1334	848	1302	-2.40%	690	-18.63%	1566	17.39%	668	-21.23%
H2O	cmplt	BK	sto3g	1406	972	1278	-9.10%	736	-24.28%	1470	4.55%	502	-48.35%
CH2	cmplt	P	sto3g	1412	900	1356	-3.97%	728	-19.11%	1558	10.34%	506	-43.78%
H8	cmplt	JW	sto3g	1534	710	1518	-1.04%	574	-19.15%	1680	9.52%	420	-40.85%
LiH	frz	P	ccpvdz	1808	1798	1748	-3.32%	1690	-6.01%	1810	0.11%	1252	-30.37%
H8	cmplt	P	sto3g	1858	1074	1812	-2.48%	856	-20.30%	2014	8.40%	572	-46.74%
LiH	frz	JW	ccpvdz	1866	1808	1866	-0.00%	1806	-0.11%	1868	0.11%	1256	-30.53%
H8	cmplt	BK	sto3g	1894	1096	1812	-4.33%	908	-17.15%	2176	14.89%	616	-43.80%
LiH	frz	BK	ccpvdz	2036	1952	1982	-2.65%	1828	-6.35%	2064	1.38%	1326	-32.07%
CH2	cmplt	BK	sto3g	2372	1486	2142	-9.70%	1154	-22.34%	2442	2.95%	664	-55.32%
LiH	cmplt	JW	631g	2482	1410	2474	-0.32%	1270	-9.93%	2526	1.77%	958	-32.06%
LiH	cmplt	P	631g	3682	2402	3572	-2.99%	1862	-22.48%	3986	8.26%	1468	-38.88%
NH	frz	JW	631g	4208	2636	4156	-1.24%	1966	-25.42%	4598	9.27%	1368	-48.10%
H2	cmplt	P	ccpvtz	4498	4488	4416	-1.82%	4328	-3.57%	4500	0.04%	3074	-31.51%

Table 4: See Table 3.

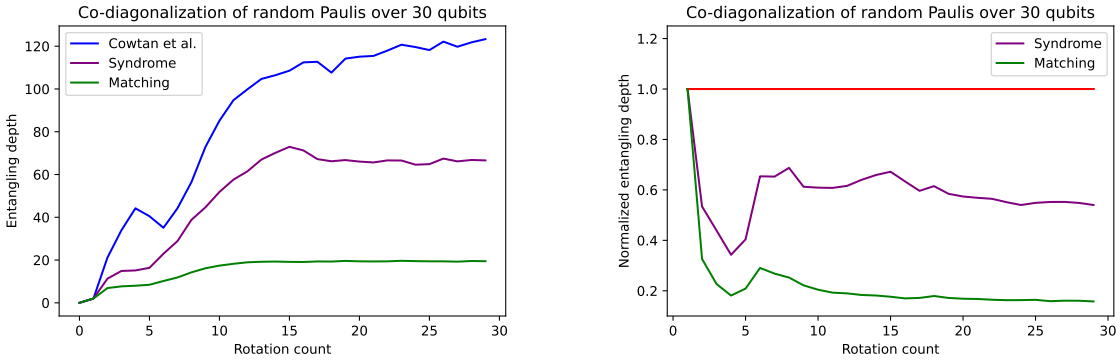


Figure 10: Left: Entangling depth (as in CNOT equivalent depth) as a function of the number of rotations in a commuting group of rotations over 30 qubits. Each point is generated by averaging the entangling depth of 40 random rotation groups. Right: the same data, but normalized by the entangling depth obtained via the state-of-the-art method of [41].

7.2 Clifford operator synthesis

We also compared our Clifford isometry synthesis algorithms with state-of-art heuristics for Clifford operator synthesis. We ran seven different synthesis algorithms:

- our count-optimized synthesis algorithm, using different number of syndrome decoding iterations (1, 10, 100, 500), (key **Syndrome** n, for n iterations)
- our depth-optimized synthesis algorithm (key **Depth**).
- the count-optimized heuristic from [31] (key **bshm**),
- and the depth-optimized algorithm from [42] (key **mz**).

We generated random Clifford operators over different qubit counts and averaged the entangling count and depth over 20 instances for each input size. The results are presented in Figure 11.

Our syndrome decoding based heuristic seems to perform quite well for small number of qubits. Calling the syndrome decoding greedy solver 10 times per decoding instance seems enough to beat the state-of-the-art methods up to 60 qubits. Increasing that number only improves performances. However a quick numerical extrapolation show that even using 500 calls to the solver is not enough to reach state-the-art performances for a few hundred qubits. This leads us to believe that for any fixed number of calls to the decoding solver, the resulting heuristic will eventually be worse than the state-of-the-art heuristic of [31]. This behavior was already observed in [37, 38] for linear operator synthesis. Nevertheless, we think that our heuristic remains competitive for practical applications where one might be ready to spend a large amount of time/resources in order to optimize a single Clifford portion in a larger circuit.

Our **Depth** method seems to outperform the algorithm of [42] even asymptotically. Of course, the main interest of the method presented in [42] lies in the derivation of an upper bound on the produced circuit's depth, which we fail to provide in our work.

				Cowtan et al.		Syndrome				Matching			
mol.	orbitals	enc.	basis	count	depth	count	rel. count	depth	rel. depth	count	rel. count	depth	rel. depth
H2	cmplt	JW	ccpvtz	4586	4488	4586	-0.00%	4488	-0.00%	4588	0.04%	3076	-31.46%
NH	frz	P	631g	4838	2894	4710	-2.65%	2216	-23.43%	5282	9.18%	1508	-47.89%
H2	cmplt	BK	ccpvtz	5056	4808	4932	-2.45%	4602	-4.28%	5066	0.20%	3154	-34.40%
LiH	cmplt	BK	631g	6600	4344	5954	-9.79%	2954	-32.00%	7170	8.64%	1810	-58.33%
NH	frz	BK	631g	6946	4380	6524	-6.08%	3132	-28.49%	7646	10.08%	1832	-58.17%
NH	cmplt	JW	631g	7014	3748	6904	-1.57%	2868	-23.48%	7880	12.35%	2004	-46.53%
H2O	frz	JW	631g	7436	3684	7318	-1.59%	2928	-20.52%	8546	14.93%	2134	-42.07%
NH	cmplt	P	631g	7522	4104	7212	-4.12%	3120	-23.98%	8128	8.06%	1884	-54.09%
H2O	frz	P	631g	8700	4834	8408	-3.36%	3666	-24.16%	9584	10.16%	2236	-53.74%
C2H4	frz	JW	sto3g	8854	3912	8564	-3.28%	3004	-23.21%	10472	18.27%	1990	-49.13%
LiH	cmplt	JW	ccpvdz	9684	5480	9504	-1.86%	4748	-13.36%	10156	4.87%	3748	-31.61%
H4	cmplt	JW	ccpvdz	10162	5334	10148	-0.14%	4774	-10.50%	10766	5.94%	3980	-25.38%
H2O	frz	BK	631g	10746	6002	10040	-6.57%	4184	-30.29%	12168	13.23%	2402	-59.98%
NH	cmplt	BK	631g	11956	7418	10486	-12.30%	4762	-35.80%	12526	4.77%	2546	-65.68%
H2O	cmplt	JW	631g	12398	5948	12052	-2.79%	4674	-21.42%	14400	16.15%	3082	-48.18%
C2H4	frz	P	sto3g	12660	6820	11646	-8.01%	4620	-32.26%	13404	5.88%	2556	-62.52%
H4	cmplt	P	ccpvdz	13594	8446	13158	-3.21%	6580	-22.09%	14702	8.15%	5198	-38.46%
LiH	cmplt	P	ccpvdz	13992	8630	13418	-4.10%	6434	-25.45%	15102	7.93%	4738	-45.10%
H4	cmplt	BK	ccpvdz	17058	10284	16328	-4.28%	7802	-24.13%	19360	13.50%	6072	-40.96%
H2O	cmplt	P	631g	18242	9642	16742	-8.22%	6584	-31.72%	19458	6.67%	3674	-61.90%
C2H4	cmplt	JW	sto3g	18734	9008	17538	-6.38%	6734	-25.24%	21654	15.59%	4114	-54.33%
LiH	cmplt	BK	ccpvdz	18906	11342	17460	-7.65%	7942	-29.98%	21090	11.55%	5410	-52.30%
C2H4	frz	BK	sto3g	20720	9718	17730	-14.43%	6904	-28.96%	21914	5.76%	3422	-64.79%
C2H4	cmplt	P	sto3g	22402	11642	20142	-10.09%	7826	-32.78%	23576	5.24%	4082	-64.94%
H2O	cmplt	BK	631g	28008	14542	23914	-14.62%	9718	-33.17%	29076	3.81%	4548	-68.73%
C2H4	cmplt	BK	sto3g	41642	19966	34076	-18.17%	12622	-36.78%	42030	0.93%	5774	-71.08%

Table 5: See Table 3.

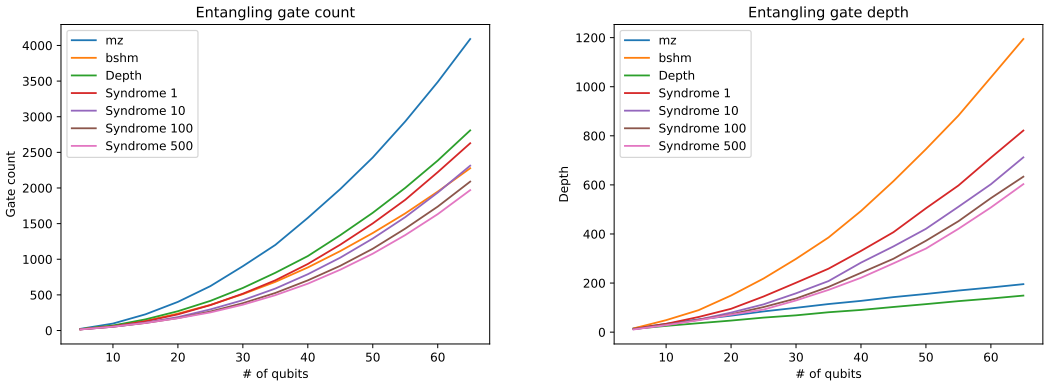


Figure 11: Entangling gate count and depth as a function of the number of qubits averaged over 20 random Clifford operators.

8 Conclusion

This article provides a generic compilation method for Clifford isometries. The framework exploits the graph-state structure of Clifford operators already exposed in previous works in the literature but with a minimal set of theoretical requirements. Overall, the synthesis of Clifford isometries is encoded as the reduction of a symmetric boolean matrix into a purposely defined identity matrix. The available operations are elementary: row and column operations, entries flip, and suitable rank-one updates. The versatility of our framework is highlighted by its capability of rediscovering almost every normal forms of the literature for Clifford operators and stabilizer states. Our framework also highlights deeper understandings about the role of phase polynomials and equivalent structures (namely

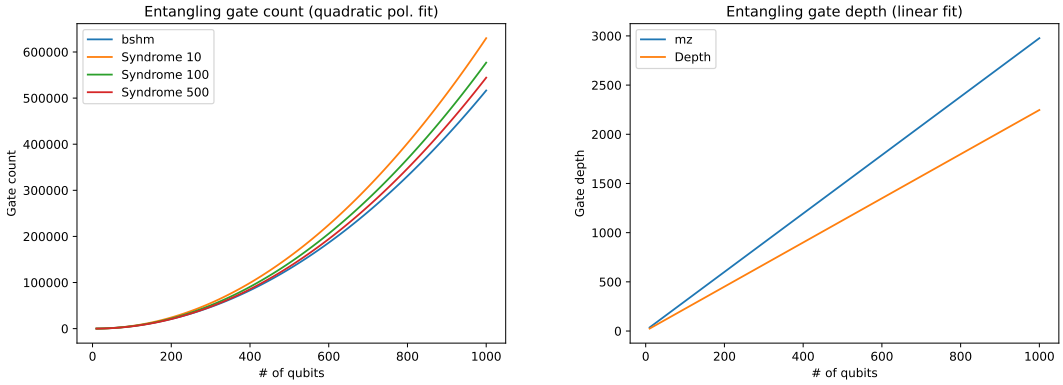


Figure 12: Extrapolated entangling gate count and depth for relevant methods. The fit are degree 2 polynomials for count and linear functions for depth. The fits were derived from the data presented in Figure 11.

CNOT + Rx circuits) in the construction of Clifford circuits. One application is the proof that any n-qubit Clifford circuit can be executed in two-qubit depth $7n$ on an LNN architecture. We also practically improve the codiagonalization of Pauli rotations and use it to optimize quantum circuits from quantum chemistry.

Still, while being simple and accessible, our framework lacks some universality. The Hadamard gate can only be applied in specific cases, and we are constrained to keep our Clifford isometries in a graph-state form. Therefore our framework can fail at offering the possibility to find optimal Clifford circuits for one given operator. As a future work it would be interesting to extend the graph-state formalism and see if a simple framework as universal as the tableau formalism and as simple as the graph-state approach can be derived.

Acknowledgments

The authors thank Simon Perdrix for comments on an earlier version of this article. This work has been supported by the French state through the ANR as a part of *Plan France 2030*, projects NISQ2LSQ (ANR-22-PETQ-0006) and EPiQ (ANR-22-PETQ-0007), as well as the ANR project SoftQPro (ANR-17-CE25-0009).

References

[1] Daniel Gottesman. "Stabilizer codes and quantum error correction". PhD thesis. Caltech. (1997).

[2] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland. "Randomized benchmarking of quantum gates". Phys. Rev. A **77**, 012307 (2008).

[3] Easwar Magesan, J. M. Gambetta, and Joseph Emerson. "Scalable and robust randomized benchmarking of quantum processes". Phys. Rev. Lett. **106**, 180504 (2011).

[4] Sergey Bravyi and Alexei Kitaev. "Universal quantum computation with ideal clifford gates and noisy ancillas". Phys. Rev. A **71**, 022316 (2005). url: https://doi.org/10.1103/PhysRevA.71.022316.

[5] Emanuel Knill. "Quantum computing with realistically noisy devices". Nature **434**, 39–44 (2005).

[6] Charles H. Bennett, David P. DiVincenzo, John A. Smolin, and William K. Wootters. "Mixed-state entanglement and quantum error correction". Phys. Rev. A **54**, 3824–3851 (1996).

[7] Scott Aaronson and Daniel Gottesman. "Improved simulation of stabilizer circuits". Phys. Rev. A **70**, 052328 (2004).

[8] Daniel Gottesman. "The heisenberg representation of quantum computers" (1998). url: https://arxiv.org/abs/quant-ph/9807006.

[9] P.W. Shor. "Fault-tolerant quantum computation". In Proceedings of 37th Conference on Foundations of Computer Science. Pages 56–65. (1996).

[10] John Preskill. "Fault-tolerant quantum computation". In Introduction to quantum computation and information. Pages 213–269. World Scientific (1998).

[11] John Preskill. "Quantum Computing in the NISQ era and beyond". Quantum **2**, 79 (2018).

[12] Riccardo Manenti, Eyob A Sete, Angela Q Chen, Shobhan Kulshreshtha, Jen-Hao Yeh, Feyza Oruc, Andrew Bestwick, Mark Field, Keith Jackson, and Stefano Poletto. "Full control of superconducting qubits with combined on-chip microwave and flux lines". Applied Physics Letters **119**, 144001 (2021).

[13] Jerry M. Chow, Jay M. Gambetta, Easwar Magesan, David W. Abraham, Andrew W. Cross, B. R. Johnson, Nicholas A. Masluk, Colm A. Ryan, John A. Smolin, Srikanth J. Srinivasan, and M. Steffen. "Implementing a strand of a scalable fault-tolerant quantum computing fabric". Nature Communications **5**, 4015 (2014).

[14] Christopher Chamberland, Guanyu Zhu, Theodore J. Yoder, Jared B. Hertzberg, and Andrew W. Cross. "Topological and Subsystem Codes on Low-Degree Graphs with Flag Qubits". Physical Review X **10**, 011022 (2020).

[15] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. "Quantum supremacy using a programmable superconducting processor". Nature **574**, 505–510 (2019).

[16] John van de Wetering. "Constructing quantum circuits with global gates". New Journal of Physics **23**, 043015 (2021).

[17] Dmitri Maslov and Yunseong Nam. "Use of global interactions in efficient quantum circuit constructions". New Journal of Physics **20**, 033018 (2018).

[18] Simon Martiel and Timothée Goubault de Brugière. "Architecture aware compilation of quantum circuits via lazy synthesis". Quantum **6**, 729 (2022).

[19] Dmitri Maslov and Martin Roetteler. "Shorter stabilizer circuits via bruhat decomposition and quantum circuit transformations". IEEE Trans. Inf. Theory **64**, 4729–4738 (2018).

[20] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John Van De Wetering. "Graph-theoretic simplification of quantum circuits with the zx-calculus". Quantum **4**, 279 (2020).

[21] Sergey Bravyi and Dmitri Maslov. "Hadamard-free circuits expose the structure of the clifford group". IEEE Transactions on Information Theory **67**, 4546–4563 (2021).

[22] Marc Bataille. "Reduced quantum circuits for stabilizer states and graph states" (2021). url: https://arxiv.org/abs/2107.00885.

[23] Maarten Van den Nest. "Classical simulation of quantum computation, the gottesman-knill theorem, and slightly beyond". Quantum Inf. Comput. **10**, 258–271 (2010).

[24] Héctor J. García, Igor L. Markov, and Andrew W. Cross. "On the geometry of stabilizer states". Quantum Inf. Comput. **14**, 683–720 (2014).

[25] Matthew Amy, Parsiad Azimzadeh, and Michele Mosca. "On the controlled-NOT complexity of controlled-NOT–phase circuits". Quantum Science and Technology **4**, 015002 (2018).

[26] Vivien Vandaele, Simon Martiel, and Timothée Goubault de Brugière. "Phase polynomials synthesis algorithms for nisq architectures and beyond". Quantum Science and Technology **7**, 045027 (2022).

[27] Sergey Bravyi, Joseph A Latone, and Dmitri Maslov. "6-qubit optimal clifford circuits". npj Quantum Information **8**, 79 (2022).

[28] Sarah Schneider, Lukas Burgholzer, and Robert Wille. "A SAT encoding for optimal clifford circuit synthesis". In Atsushi Takahashi, editor, Proceedings of the 28th Asia and South Pacific Design Automation Conference, ASPDAC 2023, Tokyo, Japan, January 16-19, 2023. Pages 190–195. ACM (2023).

[29] Tom Peham, Nina Brandl, Richard Kueng, Robert Wille, and Lukas Burgholzer. "Depth-optimal synthesis of clifford circuits with SAT solvers". In Brian La Cour, Lia Yeh, and Marek Osinski, editors, IEEE International Conference on Quantum Computing and Engineering, QCE 2023, Bellevue, WA, USA, September 17-22, 2023. Pages 802–813. IEEE (2023).

[30] Vadym Kliuchnikov and Dmitri Maslov. "Optimization of clifford circuits". Phys. Rev. A **88**, 052307 (2013).

[31] Sergey Bravyi, Ruslan Shaydulin, Shaohan Hu, and Dmitri Maslov. "Clifford Circuit Optimization with Templates and Symbolic Pauli Gates". Quantum **5**, 580 (2021).

[32] Sergey Bravyi and Alexei Kitaev. "Universal quantum computation with ideal Clifford gates and noisy ancillas". Physical Review A **71**, 022316 (2005).

[33] Sergey Bravyi and Jeongwan Haah. "Magic-state distillation with low overhead". Physical Review A **86**, 052329 (2012).

[34] Dmitri Maslov and Willers Yang. "Cnot circuits need little help to implement arbitrary hadamard-free clifford transformations they generate". npj Quantum Information **9**, 96 (2023).

[35] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. "Graphical description of the action of local clifford transformations on graph states". Phys. Rev. A **69**, 022316 (2004).

[36] Samuel A. Kutin, David Petrie Moulton, and Lawren Smithline. "Computation at a distance". Chicago J. Theor. Comput. Sci.**2007** (2007).

[37] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Quantum CNOT circuits synthesis for NISQ architectures using the syndrome decoding problem". In Ivan Lanese and Mariusz Rawski, editors, Reversible Computation - 12th International Conference, RC 2020, Oslo, Norway, July 9-10, 2020, Proceedings. Volume 12227 of Lecture Notes in Computer Science, pages 189–205. Springer (2020).

[38] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Decoding techniques applied to the compilation of CNOT circuits for NISQ architectures". Sci. Comput. Program. **214**, 102726 (2022).

[39] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Reducing the depth of linear reversible quantum circuits". IEEE Transactions on Quantum Engineering **2**, 1–22 (2021).

[40] Cristopher Moore and Martin Nilsson. "Parallel quantum computation and quantum codes". SIAM J. Comput. **31**, 799–815 (2001).

[41] Alexander Cowtan, Will Simmons, and Ross Duncan. "A generic compilation strategy for the unitary coupled cluster ansatz" (2020). url: `https://arxiv.org/abs/2007.10515`.

[42] Dmitri Maslov and Ben Zindorf. "Depth optimization of cz, cnot, and clifford circuits". IEEE Transactions on Quantum Engineering **3**, 1–8 (2022).

Action of the Clifford gates in the graph-state formalism

1. S on the output qubit i $(1 \leq i \leq n)$
   We perform the row operation $M[i,:] = M[i,:] \oplus M[i+n,:]$. We get

$$M = \left[\begin{array}{c|c} G_n B \oplus \tilde{e}_{ii} B & (B^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_n B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus \tilde{e}_{ii} B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \\ \hline B & B\begin{pmatrix} G_k \\ 0 \end{pmatrix} \end{array}\right]$$

   and we have $G_n = G_n \oplus \tilde{e}_{ii}$. Equivalently we have

$$G = G \oplus \tilde{e}_{i+k,i+k}$$

2. S on the input qubit i $(1 \leq i \leq k)$
   We perform the column operation $M[:,i+n] = M[:,i+n] \oplus M[:,i]$. We get

$$M = \left[\begin{array}{c|c} G_n B & (B^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_n B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus G_n B \tilde{e}_{ii} \\ \hline B & B\begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus B\tilde{e}_{ii} \end{array}\right]$$

and we have $G_k = G_k \oplus \tilde{e}_{ii}$. Equivalently we have

$$G = G \oplus \tilde{e}_{i,i}$$

3. CZ on the output qubits i, j $(1 \leq i < j \leq n)$
   We perform the row operations $M[i,:] = M[i,:] \oplus M[j+n,:]$ and $M[j,:] = M[j,:] \oplus M[i+n,:]$. We get

$$M = \left[\begin{array}{c|c} G_n B \oplus \tilde{e}_{ij} B & (B^T)^{-1} \begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_n B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus \tilde{e}_{ij} B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \\ \hline B & B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \end{array}\right]$$

   and we have $G_n = G_n \oplus \tilde{e}_{ij}$. Equivalently we have

$$G = G \oplus \tilde{e}_{i+k,j+k}$$

4. CZ on the input qubits i, j $(1 \leq i < j \leq n)$
   We perform the column operations $M[:, i+n] = M[:, i+n] \oplus M[:, j]$ and $M[:, j+n] = M[:, j+n] \oplus M[:, i]$. We get

$$M = \left[\begin{array}{c|c} G_n B & (B^T)^{-1} \begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_n B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus G_n B \tilde{e}_{ij} \\ \hline B & B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus B \tilde{e}_{ij} \end{array}\right]$$

   and we have $G_k = G_k \oplus \tilde{e}_{ij}$. Equivalently we have

$$G = G \oplus \tilde{e}_{ij}$$

5. $CNOT$ on the output qubits with control i, target j $(1 \leq i < j \leq n)$
   We perform the row operations $M[i,:] = M[i,:] \oplus M[j,:]$ and $M[j+n,:] = M[j+n,:] \oplus M[i+n,:]$. We get

$$M = \left[\begin{array}{c|c} E_{ij} G_n B & E_{ij} (B^T)^{-1} \begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus E_{ij} G_n B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \\ \hline E_{ji} B & E_{ji} B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \end{array}\right].$$

   We write

$$M = \left[\begin{array}{c|c} E_{ij} G_n E_{ji} \times E_{ji} B & (E_{ji} B)^{-T} \begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus E_{ij} G_n E_{ji} \times E_{ji} B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \\ \hline E_{ji} B & E_{ji} B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \end{array}\right]$$

   and we get $G_n = E_{ij} G_n E_{ji}$ and $B = E_{ji} B$, i.e, $B_{k,n} = B_{k,n} E_{ji}$ or $B^T_{k,n} = E_{ij} B^T_{k,n}$. Equivalently we have

$$G = E_{i+k,j+k} G E_{j+k,i+k}.$$

6. $CNOT$ on the input qubits with control i, target j $(1 \leq i < j \leq k)$
We perform the column operations $M[:,j] = M[:,j] \oplus M[:,i]$ and $M[:,i+n] = M[:,i+n] \oplus M[:,j+n]$. We get

$$M = \left[\begin{array}{c|c} G_n B E_{ij} & (B^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} E_{ji} \oplus G_n B \begin{pmatrix} G_k \\ 0 \end{pmatrix} E_{ji} \\ \hline BE_{ij} & B\begin{pmatrix} G_k \\ 0 \end{pmatrix} E_{ji} \end{array}\right].$$

We write

$$M = \left[\begin{array}{c|c} G_n B E_{ij} & (BE_{ij})^{-T})\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_n BE_{ij} \begin{pmatrix} E_{ij}G_kE_{ji} \\ 0 \end{pmatrix} \\ \hline BE_{ij} & BE_{ij}\begin{pmatrix} E_{ij}G_kE_{ji} \\ 0 \end{pmatrix} \end{array}\right]$$

and we get $G_k = E_{ij}G_kE_{ji}$ and $B = BE_{ij}$, i.e, $B_{k,n} = E_{ij}B_{k,n}$. Equivalently we have

$$G = E_{ij}GE_{ji}.$$

7. if $G_n[i,i] = 0$, then we can apply an $R_x(\pi/2)$ on the output qubit i, we get

$$M = \left[\begin{array}{c|c} G_n B & (B^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_n B \begin{pmatrix} G_k \\ 0 \end{pmatrix} \\ \hline (I + e_{ii}G_n)B & (I + e_{ii}G_n)B\begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus e_{ii}(B^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \end{array}\right].$$

One can check that $(I + e_{ii}G)^2 = I$, therefore:

$$M = \left[\begin{array}{c|c} G_n(I + e_{ii}G_n)(I + e_{ii}G_n)B & (((I + e_{ii}G_n)(I + e_{ii}G_n)B)^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_n(I + e_{ii}G_n)(I + e_{ii}G_n)B\begin{pmatrix} G_k \\ 0 \end{pmatrix} \\ \hline (I + e_{ii}G_n)B & (I + e_{ii}G_n)B\begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus e_{ii}(((I + e_{ii}G_n)(I + e_{ii}G_n)B)^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \end{array}\right]$$

It is easy to see that we will have

$$G_n \leftarrow G_n(I + e_{ii}G_n),$$

$$B \leftarrow (I + e_{ii}G_n)B,$$

but it is trickier to see the effect on G_k. First, let's simplify the formula by replacing by the new values of B and G_n:

$$M = \left[\begin{array}{c|c} G'_nB' & (I + e_{ii}G_n)^T(B'^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G'_nB'\begin{pmatrix} G_k \\ 0 \end{pmatrix} \\ \hline B' & B'\begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus e_{ii}(I + e_{ii}G_n)^T(B'^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \end{array}\right].$$

We have $e_{ii}(I + e_{ii}G_n)^T = e_{ii}(I + G_ne_{ii}) = e_{ii}$ because $e_{ii}G_ne_{ii} = G_n[i,i]e_{ii} = 0$, then

$$M = \left[\begin{array}{c|c} G'_nB' & (B'^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G_n e_{ii} (B'^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G'_nB'\begin{pmatrix} G_k \\ 0 \end{pmatrix} \\ \hline B' & B'\begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus e_{ii}(B'^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \end{array}\right].$$

We remind that $G_nB = G'_nB'$, therefore $G_n = G'_nB'B^{-1}$:

$$M = \left[\begin{array}{c|c} G'_nB' & (B'^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G'_nB'B^{-1} e_{ii} (B'^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G'_nB'\begin{pmatrix} G_k \\ 0 \end{pmatrix} \\ \hline B' & B'\begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus B'B'^{-1}e_{ii}(B'^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \end{array}\right].$$

We need to compute two last quantities:

$$B^{-1}e_{ii}(B'^T)^{-1}$$

and

$$B'^{-1}e_{ii}(B'^T)^{-1}$$

We recall that $e_{ii}(I + e_{ii}G_n)^T = e_{ii}$, and similarly $(I + e_{ii}G_n)e_{ii} = e_{ii}$. So

$$B^{-1}e_{ii}(B'^T)^{-1} = B^{-1}e_{ii}(I + e_{ii}G_n)^T(B^{-1})^T = B^{-1}e_{ii}(B^{-1})^T,$$
$$B'^{-1}e_{ii}(B'^T)^{-1} = B^{-1}(I + e_{ii}G_n)e_{ii}(I + e_{ii}G_n)^T(B^{-1})^T = B^{-1}e_{ii}(B^{-1})^T.$$

Writing $b_i = B^{-1}[:, i]$, $B^{-1}e_{ii}(B^{-1})^T = b_ib_i^T$ and finally we have

$$M = \left[\begin{array}{c|c} G'_nB' & (B'^T)^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus G'_nB'\left(b_ib_i^T\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus \begin{pmatrix} G_k \\ 0 \end{pmatrix}\right) \\ \hline B' & B'\left(b_ib_i^T\begin{pmatrix} I_k \\ 0 \end{pmatrix} \oplus \begin{pmatrix} G_k \\ 0 \end{pmatrix}\right) \end{array}\right].$$

i.e

$$\begin{pmatrix} G_k \\ 0 \end{pmatrix} \leftarrow \begin{pmatrix} G_k \\ 0 \end{pmatrix} \oplus b_ib_i^T\begin{pmatrix} I_k \\ 0 \end{pmatrix}$$

and

$$G_k \leftarrow G_k \oplus b_i[1:k]b_i[1:k]^T = G_k \oplus B_{k,n}[:, i]B_{k,n}[:, i]^T.$$

We do not care about the block below G_k being modified as we can always zero it with free column operations.

To summarize, we have

$$G_n \leftarrow G_n(I + e_{ii}G_n), \text{ equivalently, } G_n \leftarrow G_n \oplus G_n[:, i]G_n[:, i]^T$$
$$G_k \leftarrow G_k \oplus B_{k,n}[:, i]B_{k,n}[:, i]^T,$$
$$B^{-1} \leftarrow B^{-1}(I + e_{ii}G_n), \text{ equivalently, } B_{k,n} \leftarrow B_{k,n} \oplus B_{k,n}[:, i]G_n[:, i]^T.$$

One can check that this is equivalent to

$$G \leftarrow G \oplus G[:, k + i]G[:, k + i]^T.$$

8. A similar proof can be given to show the effect of an $R_x(\pi/2)$ gate on the input qubits.

9. For the Hadamard gates, simply write $H \sim SR_x(\pi/2)S$ and apply successively the three gates to have the results.