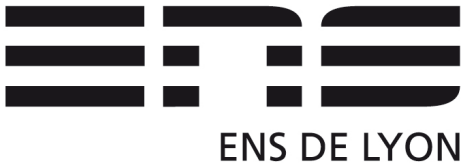
École Normale Supérieure de Lyon
Centre Inria de Lyon





M2 Internship Report

# Optimizing quantum measurements through Pauli sparsification

**Key words:** *Quantum, Measurement, Noise, Pauli Group, Clifford Group, Symplectic form, Graphs.*

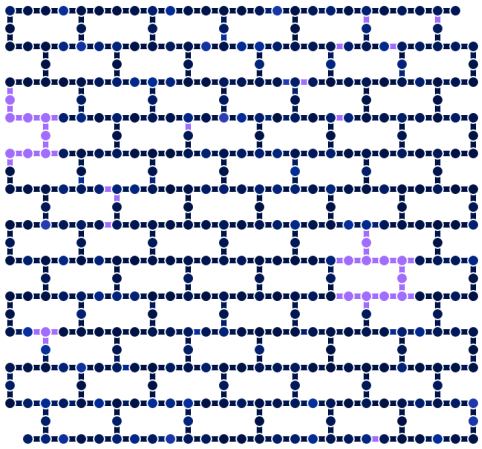


Figure 1: IBM 433 qubits quantum computer's network.

*Student:*
Maxime Cautrès

*Supervisor:*
Daniel Stilck França

February the 6th 2023 - July the 10th 2023

# Contents


**1 Introduction** **2**
- 1.1 Motivations . . . 2
- 1.2 State of the art . . . 4

**2 Preliminaries** **4**
- 2.1 The Pauli group . . . 4
- 2.2 The Clifford group . . . 6
- 2.3 Tableau representation of Clifford . . . 7

**3 The Study of Pauli Sparsification** **10**
- 3.1 The anticommutating structure . . . 11
- 3.2 A very specific Symplectic version of Gram-Schmidt . . . 13
- 3.3 Naive implementation of Pauli sparsification . . . 16

**4 A deeper study of cliques** **17**
- 4.1 The biggest click of $\mathcal{P}_n^w$. . . . 17
- 4.2 A construction of low-weight cliques . . . 19

**5 Conclusion** **20**

**6 Appendix** **22**

**A The behaviour of the product of Pauli matrices onto their tableau representations.** **22**

**B Detailed proof of the HCP round theorem by Aaronson and Gottesman** **25**

# 1 Introduction

**Context of the internship** As part of my second year of Master's at the École Normale Supérieure de Lyon, I completed a 5-months internship at a computer science laboratory. My previous internship at the University of Latvia, working on quantum query algorithms for recognizing Dyck words with Andris Ambainis and Kamil Khadiev, sparked my interest in continuing research in this area. With guidance from Omar Fawzi, I discussed with Daniel Stilck França about some interesting combinatorial problems in quantum computer science and we agreed on an internship under his supervision. My internship took place at the QInfo Inria team, hosted by the LIP at the ENS de Lyon. I am grateful to the team members for their warm welcome and insightful discussions. I also want to thank my fellow students and classmates for the great time together. The QInfo team's focus is on developing methods and algorithms to address the impact of noise on quantum information processing tasks. Consequently, my work on Pauli sparsification, and its application to noise reduction, aligns well with the team's objectives.

**The History of Quantum Computing** The history of quantum computing began in 1980 when physicist Paul Benioff proposed a new model for Turing machines based on quantum mechanics [Ben80]. This model utilized the unique properties of quantum physics discovered by physicists. It sparked interest among computer scientists who saw the potential of quantum computing surpassing classical computing. Shortly after, Richard Feynman introduced quantum circuits, marking a significant milestone in the field [Fey86]. In the mid-1990s, progress was made in constructing practical quantum computers. This led to increased funding and research, resulting in the growth of start-ups and companies dedicated to advancing quantum computing. Today, quantum computing is an active area of scientific exploration. A significant algorithmic breakthrough came in 1994 with Peter Shor's factorization algorithm [Sho94, p. 1994]. This algorithm showed exponential complexity improvement and posed a potential threat to cryptographic protocols once quantum computers reach sufficient power. Currently, the main challenges in achieving practical quantum computers revolve around addressing issues related to noise and error correction.

## 1.1 Motivations

The common denominator of any scientific experiment is the need for measurements and, more importantly, accurate measurements. This is also the case for scientists working in quantum physics or chemistry. One of these important measurements is to estimate the energy of a physical system. To do this, it is necessary to **measure an observable**, for example, the Hamiltonian of the hydrogen atom. Usually, the simplest way to make these measurements is to compute the expectation value of the operator associated with that observable. However, this is often not possible because the measurements have to be made on **the computational basis**. For this reason, operators are written as a weighted sum of Pauli operators, on which it is easier to perform measurements, but at the high cost of greatly increasing the number of measurements required. This intermediary role, in almost every measurement, underscores the importance of Pauli operators within the domain of quantum science. **Pauli operators** on a qubit are the matrices $I = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $Y = \left(\begin{smallmatrix} 0 & i \\ -i & 0 \end{smallmatrix}\right)$ and $Z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$, they form a basis for any quantum operator on a single qubit and are denoted by $\mathcal{P}_1$. They can be generalised to multiple qubits simply by taking $\mathcal{P}_1$ over $n$ qubits, more precisely $\mathcal{P}_n := \{\sigma_1 \otimes \cdots \otimes \sigma_n, \sigma_i \in \{I, X, Y, Z\}\}$.

Recent advances in quantum chemistry [Tra+18] provide examples of problems where scientists need to measure observables that decompose into Pauli operators of increasing weight. **The weight of a Pauli operator** is determined by the number of non-identity Pauli matrices present in its decomposition as a tensor product of Pauli matrices on a single qubit. The consideration of the weight is important due to its direct influence on the measurement noise.

Indeed, one of the main problems with quantum computers is that every **operation can fail**, with different probabilities and outcomes depending on the operator itself, but with a high enough probability to generate strong noise. The probability of failure of a 1-qubit gate is smaller than the one of a 2-qubit gate and finally than the one of a measurement. It is possible to verify these observations on any quantum computer that IBM has made available on its website. For example, their largest quantum computer, called IBM Seattle, has respective error rates of $p_1 =$

$6,349e-4, p_2 = 2,09e-2$ and finally $p_{readout\_error} = 5,520e-2$. This noise is built up through the quantum circuit, meaning for example that the number of qubits in the circuit has an exponential effect on the noise, as does the number of gates and measurements. More importantly for the problem of Pauli observables, there is also an exponential dependence between the final noise and the weight of the Pauli operator.

A way to reduce the noise in high-weight Pauli measurements would be interesting as it would improve the quality of the measurement and then the time it takes to get an accurate estimate of the expectation. In fact, the way people deal with noise is quite simple, they just repeat a lot of the same measurements to be confident enough in the approximation. **The weight has an exponential dependence on the noise**, so it also has an exponential dependence on the running time of the approximation. An interesting way to reduce this influence is to reduce the weight. In fact, by using a reasonable number of one and two-qubit gates, which have a lower probability of failure than the measurement, it may be possible to reduce the weight of the Pauli operator to be measured enough to reach a point where it is worth doing this precomputation from a global noise point of view.

For example, let's say one wants to measure the state of the density matrix $\rho$ with the Pauli operator $Z^{\otimes 433}$ on the best largest quantum computer from IBM as described above. With the noise, the expectation value measured is $(1 - p_{readout\_error})^{433}\langle Z^{\otimes 433}\rangle_\rho$ instead of $\langle Z^{\otimes 433}\rangle_\rho$ the expectation value we are looking for. This implies that the number of repetitions needed to get a $\varepsilon$ estimate is in $O(((1 - p_{readout\_error})^{433}\varepsilon)^{-2}) = O(2.27e21\varepsilon^{-2})$. Now, by applying a few rounds of $CNOT$, the state $\rho$ becomes $\rho'$, and measuring the observable $Z^{\otimes n}$ becomes measuring the observable $I^{\otimes 432} \otimes Z$. The expectation value being measured is now $(1 - p_{readout\_error})(1 - p_2)^{433}\langle I^{\otimes 432} \otimes Z\rangle_{\rho'} = (1 - p_{readout\_error})(1 - p_2)^{433}\langle Z^{\otimes 433}\rangle_\rho$. With the IBM value for noise, this implies that the number of repetitions for a $\varepsilon$ approximation is in $O(((1 - p_{readout\_error})(1 - p_2)^{433}\varepsilon)^{-2}) = O(9.842e7\varepsilon^{-2})$. If we compare the two numbers of repetitions, we can see that the first experiment does not run in a reasonable time, while the second is much better, even if it is still not reasonable. It is important to say that these simple calculations are a bad approximation because we assume that it is possible to do a CNOT on every pair of two qubits, which is not the case on real quantum computers. The Figure 2 illustrates this motivation.

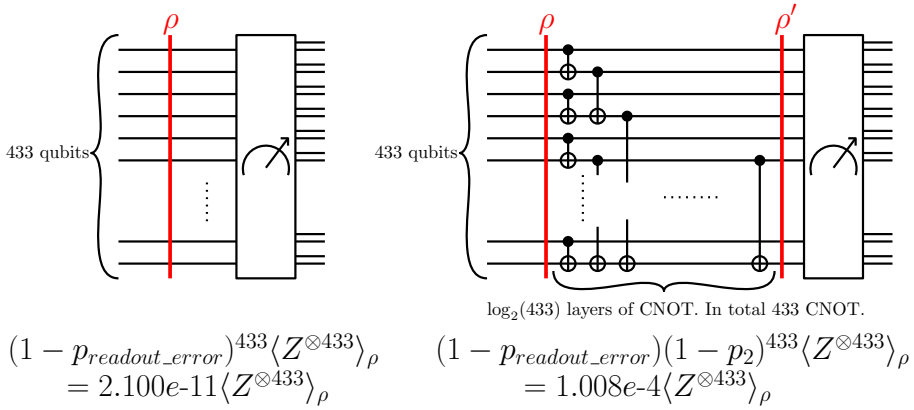



Figure 2: Illustration of the example where one wants to measure $\rho$ with $Z^{\otimes n}$.

In the last example, we interacted with the state using **CNOT** gates, but more generally one can use other types of gates such as the **Phase** and the **Hadamard** gates. Indeed, the 3 gates are generators of the group of unitaries called the **Clifford group**. This group has the specific property of stabilizing the Pauli operators, i.e. always sending one Pauli operator to another. This is an essential property since one still wants to measure Pauli operators. This action of reducing the weight of the Pauli observable before measuring it can be called **sparsification**. It can even be done simultaneously if the same Clifford operator can sparsify enough Pauli observables. Unfortunately, these Clifford operators are always bijective, so it is not always possible to reduce the maximum weight of a set of Pauli operators by applying a unique Clifford operator. For example, $\mathcal{P}_n$ is always sent to $\mathcal{P}_n$ by any Clifford operator. Therefore, it can be useful **to divide the set of Pauli** observables waiting to be measured into sets on which an individualising Clifford operator can

simultaneously sparsify the elements. The Pauli sparsification problem can then be written as follows:

**Problem 1** (**Pauli Sparsification**). *Let $A := \{Q_1, \ldots, Q_m\}$ be a subset of $\mathcal{P}_n$. The weight of a set $A$ of Paulis corresponds to the maximal weight of its elements. The sparsification problem is about finding pairs $(A_i, C_i)$ for $i$ in $[1, k]$ such that the $A_i$ are a partition of $A$ and $C_i$ a Clifford operator such that the weight of the set $C_i A_i C_i^\dagger$, the conjugation of each element of $A_i$ by the Clifford gate $C_i$, is significantly smaller than the original one.*

### 1.2 State of the art

On the one hand, many of the objects discussed in the previous paragraphs are well-known in the quantum computing community. For example, some optimisations for measuring with Pauli observables have already been studied. Jena, Genin, and Mosca in [JGM19] use simultaneous diagonalization to measure more efficiently some Pauli observables that commute together. Their work also makes nice connections with graph theory to find a minimal partition of the observables. Then the Clifford group is a well-known structure that appears in many different fields, from mathematics to chemistry and physics. Its properties are really interesting, for example, the Gottesman-Knill theorem [Got98a] states that every element of the Clifford group can be perfectly simulated on a classical probabilistic computer in polynomial time. More precisely, with Aaronson [AG04], they proved that every Clifford operator can be implemented using a polynomial number of gates $H$, $P$ and $CNOT$. Moreover, Clifford gates are important because they are sufficient to write some quantum algorithms for quantum error correction [KS06] and entanglement distillation [AG05]. This progress led Aaronson and Gottesman to write a high-performance simulator for circuits consisting only of Clifford gates [AG03]. However, Clifford gates are not sufficient to simulate every quantum algorithm, more precisely, circuits made only of $H$, $P$ and $CNOT$ can only compute the class of problems $\oplus L$ [AKH; AG04]. All these properties and applications have made the Clifford group a central object of quantum computing.

On the other hand, our Problem 1 has never been treated. However, a wide range of objects and properties more or less related to our sparsification problem have already been studied and will be presented with more details and rigour in the Section 2. Then, in Section 3, I will present in detail our attempts to implement an algorithm to solve the sparsification problem. In the last Section 4 I will discuss some properties about subsets of Paulis, where all elements anticommute, in order to find some interesting bounds on the best achievable weight of a sparsification. It is also important to note that this work is only a first step towards solving this question, and that many subproblems that are not considered need to be taken into account.

## 2 Preliminaries

In the introduction, I briefly defined certain notions; however, it is important to provide a more detailed exposition of the specific properties associated with these objects.

### 2.1 The Pauli group

First, let's recall the definition of the Pauli group. On one qubit, the Pauli matrices ($\mathcal{P}_1$) are $I = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $Y = \left(\begin{smallmatrix} 0 & i \\ -i & 0 \end{smallmatrix}\right)$ and $Z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$. By adding a phase $\{1, i, -1, -i\}$ and the matrix product, as a binary operation, it forms a group structure with interesting properties. Indeed,

- Every element is its own inverse.
- $X$ and $Z$ generate $\mathcal{P}_1$, e.g. $XZ = -iY$.
- Two Pauli matrices either **commute or anticommute**.

$$XY = iZ = -YX \qquad YZ = iX = -ZY \qquad ZX = iY = -XZ. \tag{1}$$

- Multiplying two distinct non-identities returns the third non identity up to a phase.

Then, the Pauli group on $n$ qubits is obtained by bringing together $n$ one qubit Pauli groups. Lets name $\mathcal{P}_n$ the Pauli group on $n$ qubits, it is defined as

$$\mathcal{P}_n = \{e^{\frac{k\pi}{2}i} Q_1 \otimes \cdots \otimes Q_n, \quad k \in [0,3], \quad Q_i \in \{I, X, Y, Z\}\}. \tag{2}$$

The elements of $\mathcal{P}_n$ are called Pauli matrices and there are $\#\mathcal{P}_n = 4^{n+2}$ of them. The Pauli strings are the elements of $\mathcal{P}_n$ with a phase equal to 1 [GS22]. The set of Pauli strings on $n$ qubits $(\mathcal{P}_n/U_1, .)$ is isomorphic to the vector space $(\mathbb{F}_2^{2n}, \oplus)$. Indeed, the mapping

$$I \mapsto (0,0), \quad X \mapsto (1,0), \quad Y \mapsto (1,1), \quad Z \mapsto (0,1) \tag{3}$$

preserves the binary operation. For example, $X \otimes Y \otimes Z$ is mapped to $(1,0,0,1,1,1)$. The Pauli matrices of weight at most $w$ will be denoted by $\mathcal{P}_n^w$ while the Pauli matrices of weight exactly $w$ will be denoted by $\mathcal{P}_n^{=w}$.

**Notations:** Let $X_i$ ( resp. $Z_i$, $Y_i$) denote the Pauli matrices whose only non-identity operation is $X$ (resp. $Z$, $Y$) and is on the $i$-th qubit. The families $X_i$ and $Z_i$ together form a basis of $\mathcal{P}_n$, so they will be referred to as **generators of** $\mathcal{P}_n$. The coordinates of any Pauli string of $\mathcal{P}_n/U_1$ in any basis of $\mathcal{P}_n/U_1$ can only belong to $\{0,1\}^{2n}$ since each Pauli matrix is its own inverse. In addition, to represent the phase $\phi$ of the Pauli matrice $Q$, it is necessary to add two more values in $\{0,1\}$, the first to specify the type $\{1, i\}$ and the second for the sign $\{1, -1\}$. Thus it is possible to represent a Pauli matrix by a list in $\{0,1\}^{2n+2}$, where the first $2n$ columns represent the coordinates of $Q$ on the generators and the last two columns represent the phase. This list is called the **tableau representation of a Pauli matrix** $Q$ and is written $T'_Q$. More precisely, for $k$ in $[1,n]$, the $k$-th (or $(n+k)$-th) column of $T'_Q$ corresponds to the coefficient of $X_k$ (or $Z_k$). It is then possible to go from the tableau representation to the Pauli matrices with the following relation.

$$Q := i^{(T'_Q)_{2n+1}} (-1)^{(T'_Q)_{2n+2}} \prod_{k=1}^{n} X_k^{(T'_Q)_k \left(1-(T'_Q)_{n+k}\right)} Z_k^{(T'_Q)_{n+k}(1-(T'_Q)_k)} Y_k^{(T'_Q)_k (T'_Q)_{n+k}} \tag{4}$$

Thus, the multiplication of two Pauli matrices implies on their tableau representation an element-wise $\oplus$ on the first $2n$ columns and a more complicated operation on the last two columns whose outputs depend only on the two tableau representations in entry[1]. It is then possible to consider **the tableau representation of a set of Pauli matrices** $A \subseteq \mathcal{P}_n$ as a 2D tableau denoted $T'_A$, where the $k$-th row is exactly the tableau representation of the $k$-th Pauli of the set. With the same idea, it is possible to define the respective tableau representations $T_Q, T_A$ of a Pauli string $Q$ and a set of Pauli strings $A$ by simply removing the last two columns to drop the phase part.

**The anticommutation structure.** Pauli matrices have some interesting combinatorial properties. The fact that they either commute or anticommute brings a lot of structure to a subset of Pauli matrices. The generators of $\mathcal{P}_n$ are a nice example: for $k$ in $[1,n]$, $X_k$ anticommute with no other generators than $Z_k$ and vice versa. It implies that two Pauli matrices $p, q$ anticommute if and only if there is an odd number of $k$ such that $X_k$ ( resp. $Z_k$) belongs to $p$ ( resp. $q$) (i.e. $(T_p)_k (T_q)_{n+k} = 1$). Indeed, there exist $\phi_p, \phi_q$ in $\{1, i, -1, -i\}$ such that

$$\begin{aligned} pq &= (\phi_p X^{(T_p)_1} Z^{(T_p)_{n+1}} \cdots X^{(T_p)_n} Z^{(T_p)_{2n}})(\phi_q X^{(T_q)_1} Z^{(T_q)_{n+1}} \cdots X^{(T_q)_n} Z^{(T_q)_{2n}}) \\ &= \phi_p \phi_q X^{(T_p)_1} Z^{(T_p)_{n+1}} \cdots X^{(T_p)_n} Z^{(T_p)_{2n}} X^{(T_q)_1} Z^{(T_q)_{n+1}} \cdots X^{(T_q)_n} Z^{(T_q)_{2n}} \end{aligned}$$

Then, it is possible to bring $X^{(T_q)_1}$ next to $Z^{(T_p)_{n+1}}$ as it anticommutes only with $Z^{(T_p)_{n+1}}$

$$pq = \phi_p \phi_q X^{(T_p)_1} Z^{(T_p)_{n+1}} X^{(T_q)_1} \cdots X^{(T_p)_n} Z^{(T_p)_{2n}} Z^{(T_q)_{n+1}} \cdots X^{(T_q)_n} Z^{(T_q)_{2n}}$$

Thus, if both $X^{(T_q)_1}$ and $Z^{(T_p)_{n+1}}$ exists $((T_q)_1 (T_p)_{n+1} = 1)$ then there is anticommutation so a $-1$ factor appears when the permutation is performed.

$$pq = (-1)^{(T_q)_1 (T_p)_{n+1}} \phi_q X^{(T_q)_1} \phi_p X^{(T_p)_1} Z^{(T_p)_{n+1}} \cdots X^{(T_p)_n} Z^{(T_p)_{2n}} Z^{(T_q)_{n+1}} \cdots X^{(T_q)_n} Z^{(T_q)_{2n}}$$

[1]For proofs and more details about the product between two tableau representations, see Appendix A

Then, the same steps can be done for $Z^{(T_q)_{n+1}}$ and finally for all other factors of $q$.

$$\begin{aligned} pq &= (-1)^{(T_q)_1(T_p)_{n+1}} \phi_q X^{(T_q)_1} \phi_p X^{(T_p)_1} Z^{(T_q)_{n+1}} Z^{(T_p)_{n+1}} \cdots X^{(T_p)_n} Z^{(T_p)_{2n}} \cdots X^{(T_q)_n} Z^{(T_q)_{2n}} \\ &= (-1)^{(T_q)_1(T_p)_{n+1}+(T_q)_{n+1}(T_p)_1} \phi_q X^{(T_q)_1} Z^{(T_q)_{n+1}} \phi_p X^{(T_p)_1} Z^{(T_p)_{n+1}} \cdots X^{(T_p)_n} Z^{(T_p)_{2n}} \cdots X^{(T_q)_n} Z^{(T_q)_{2n}} \\ &\vdots \\ &= (-1)^{\underbrace{(T_q)_1(T_p)_{n+1}+(T_q)_{n+1}(T_p)_1+\cdots+(T_q)_n(T_p)_{2n}+(T_q)_{2n}(T_p)_n \mod [2]}_{\Omega(p,q)}} qp \end{aligned}$$

Thus, one can **characterize the anticommutation** of $Q_1$ and $Q_2$ by $\Omega(Q_1, Q_2) = 1$ with $\Omega$ define as in [AG04]

$$\Omega(p,q) := \sum_{k=1}^{n} ((T_p)_k (T_q)_{k+n} + (T_p)_{k+n}(T_q)_k) \mod [2]. \tag{5}$$

This function is a **symplectic form** as it can be rewritten $\Omega(p,q) := T_q S T_q^\dagger \mod [2]$ with $S := \left(\begin{smallmatrix} 0 & I_n \\ I_n & 0 \end{smallmatrix}\right)$ a non-singular, skew-symmetric matrix.

## 2.2 The Clifford group

To act on Pauli matrices, the set of unitary matrices whose conjugation stabilizes $\mathcal{P}_n$ is particularly interesting. This set is known as the **Clifford group on** $n$ **qubits** and is denoted by $\mathcal{C}_n$. Formally this group is defined as

$$\mathcal{C}_n := \left\{ C \in U_{2^n}, \quad \forall Q \in \pm(\mathcal{P}_n/U_1), CQC^\dagger \in \pm(\mathcal{P}_n/U_1) \right\}. \tag{6}$$

The elements of the Clifford group, named Clifford gates, have a lot of interesting properties. First, every Clifford gates $C$ is an **automorphism**. Indeed, $C$ is a morphism as it can be defined on its generators and is invertible

$$\forall Q_1, Q_2 \in \mathcal{P}_n, \; CQ_1Q_2C^\dagger = CQ_1C^\dagger CQ_2C^\dagger \quad \text{and} \quad Q = C^\dagger(CQC^\dagger)C.$$

Then, the Clifford gates **preserve the commutation** property of two Pauli matrices. Indeed, for $Q_1, Q_2$ belonging to $\mathcal{P}_n$,

$$CQ_1C^\dagger CQ_2C^\dagger = CQ_1Q_2C^\dagger = C(-1)^{\Omega(Q_1,Q_2)}Q_2Q_1C^\dagger = (-1)^{\Omega(Q_1,Q_2)}CQ_2C^\dagger CQ_1C^\dagger.$$

Thus, every Clifford gate can be represented by its images on the generators of $\mathcal{P}_n$ and some vertical lines between two Pauli matrices that anticommute. This representation is used by Ozols [Ozo08] and many others before[AG04; Got97].

$$\begin{array}{ccccc} X_1 \mapsto CX_1C^\dagger & & X_2 \mapsto CX_2C^\dagger & \cdots & X_n \mapsto CX_nC^\dagger \\ | & & | & & | \\ Z_1 \mapsto CZ_1C^\dagger & & Z_2 \mapsto CZ_2C^\dagger & \cdots & Z_n \mapsto CZ_nC^\dagger \end{array}$$

By observing this structure, we can wonder the following question:

**Question 1.** *Let $\sigma$ be the automorphism that maps $X_i$ and $Z_i$ to $Q_i$ and $Q_{n+i}$ in $\pm\mathcal{P}_n$. If*

- *$\{Q_i\}_{i\in[2n]}$ is a family of **distinct Pauli strings** on $n$ qubits.*
- *$\{Q_i\}_{i\in[2n]}$ satisfies the **commutation structure** ($Q_iQ_j = -Q_jQ_i \Leftrightarrow i = n + j \mod 2n$)*

***Does $\sigma$ belong to $\mathcal{C}_n$?***

*More precisely, is there a Clifford such that its conjugation stabilises $\mathcal{P}_n$ and its corresponding morphism is $\sigma$?*

A positive answer to this question can be found in [AG04] with their proof of the Theorem 4 of this report. This property is useful for Maris Ozol's straightforward proof of the cardinality of the Clifford group.

**Theorem 1.** ***Cardinality of*** $\mathcal{C}_n$ *The size of the Clifford group on* $n$ *qubit is*

$$\#\mathcal{C}_n = 2^{n^2+2n} \prod_{k=1}^{n} (4^k - 1). \tag{7}$$

Moreover, Clifford gates are interesting because they are easy to implement; indeed, Aaronson and Gottesman proved in [Got98b; Got97; AG04] the Theorem 2 on the generators of the Clifford group.

**Theorem 2.** ***Generators of*** $\mathcal{C}_n$**.** *Every Clifford gate can be computed with a polynomial-size circuit made of Phase* $P := \left(\begin{smallmatrix} 1 & 0 \\ 0 & i \end{smallmatrix}\right)$*, Hadamard* $H := \frac{1}{\sqrt{2}} \left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$*, and CNOT* $:= \left(\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{smallmatrix}\right)$ *gates.*

The proof given in [AG04] is difficult to understand. The steps of the proof require a lot of intuition and practice, which take a long time to build up. That's why, with the advice of my supervisor, I decided to write **an extension** to the Aaronson and Gottesman article in Section 2.3 that introduces the tools used in the article in a more reader-friendly form. Indeed, I believe that an important result of their work is this toolbox, which is not explained and may be useful to other scientists.

## 2.3 Tableau representation of Clifford

In [AG04], Aaronson and Gottesman defined the notion of **tableau representation for Clifford gates**. Indeed, one has seen that a Clifford gate maps the generators of $\mathcal{P}_n$ onto a subset of Pauli matrices $Q_1, \ldots, Q_{2n}$, so one can almost represent a Clifford gate by the tableau representation of the set $Q_1, \ldots, Q_{2n}$. However, $Q_i$ belongs to $\pm 1$ times a Pauli string, because the eigenvalues of Pauli strings in $\pm 1$ are preserved by conjugation by a unitary [GS22]. This implies that it is possible to drop the columns corresponding to $\{1, i\}$ in the tableau representation. Thus,

**Definition 3.** ***Tableau representation of a*** $n$ ***qubit Clifford operator.*** *Let* $C \in \mathcal{C}_n$ *and* $\{Q_i\}_{i\in[2n]}$ *be the image of the generators of* $\mathcal{P}_n$ *by applying* $C$*. For all* $i$*,* $T'_{Q_i}$ *is the tableau representation of the Pauli matrix* $Q_i$ *as defined in subsection 2.1. Finally, the tableau representation* $T'_C$ *of* $C$ *is a* $2n \times (2n+1)$ *matrix such that*

$$\forall i \in [n], \quad \big(\forall j \in [2n], \quad (T'_C)_{i,j} := (T'_{Q_i})_j\big) \quad \textit{and} \quad (T'_C)_{i,2n+1} := (T'_{Q_i})_{2n+2}. \tag{8}$$

It can be represented as follows:

$$\begin{array}{c} \\ X_1 \\ \vdots \\ Z_n \end{array} \begin{array}{c} \begin{array}{cc} X_1 \ldots X_n | Z_1 \ldots Z_n | & \phi \end{array} \\ \begin{pmatrix} (T'_{Q_1})_{[1,2n]} & (T'_{Q_1})_{2n+2} \\ \vdots & \vdots \\ (T'_{Q_{2n}})_{[1,2n]} & (T'_{Q_{2n}})_{2n+2} \end{pmatrix} \end{array}. \tag{9}$$

For example, the tableau representation of the Clifford gate corresponding to the identity is $I_{2n,2n+1}$. In addition, the Clifford gate that flips every $X$ into $Z$ and vice versa has for tableau representation the matrix $S$ extended with a last row filled with zeros, i.e. $\left(\begin{smallmatrix} 0 & I_n & 0 \\ I_n & 0 & 0 \end{smallmatrix}\right)$.

**Linear Algebra interpretation of H, P CNOT.** In subsection 2.2 I presented that the Clifford group is generated by Hadamard, phase and CNOT gates. In [AG04], to prove the Theorem 2, Aaronson and Gottesman quickly detailed the linear interpretations of these three gates on the tableau representation. Here I'm going to give more details on the linear algebra interpretations of the generators and the tools that can be built with them.

- **The Hadamard gate:** First, $HXH^\dagger = Z$, $HZH^\dagger = X$, and $HYH^\dagger = -Y$ so the Hadamard gate swaps $Z$ and $X$ Pauli matrices. In the tableau representation of sets of Pauli matrices, it implies that applying a Hadamard gate on the $k$-th qubit swaps the coefficients in front of $X_k$ and $Z_k$ for every row. Thus, it swaps both the $k$-th and $(n+k)$-th columns of the tableau representation. The phase is updated only if the Hadamard gate is applied onto a $Y$. So, the $2n+1$ column with be updated by adding the elementwise product of both $k$-th and $(n+k)$-th columns.

- **The Phase gate:** On one qubit, the phase gate acts with $PXP^\dagger = Y$, $PZP^\dagger = Z$, and $PYP^\dagger = -X$. It implies that the column of $T$ corresponding to $X_k$ stays unchanged as the one corresponding to $Z_k$ takes the value of $X_k \oplus Z_k$. The phase is affected as with the Hadamard so it is sufficient to update the last column with the elementwise product of the $k$-th and the $(n+k)$-th columns.
- **The CNOT gate:** This gate is on two qubits, from $a$ to target $b$, and acts such that the columns corresponding to $X_a$ and $Z_b$ stay unchanged but the column corresponding to $X_b$ (resp. $Z_a$) takes the value of the elementwise operation on the columns $X_b \oplus X_a$ (resp. $Z_b \oplus Z_a$). The update of the phase here is more complicated but it is still possible to track. Indeed, Aaronson proved that the $(2n+1)$-th column is updated with the addition of the following elementwise expression $X_a Z_b (X_b \oplus Z_a \oplus 1)$.

We have seen that the tableau representation is useful for representing Clifford gates and allowing the phase to be taken into account. However, our problem is to reduce the weight of the Pauli strings, so the **useful information does not include the phase**. Later, when performing a simulation, the phase will be needed again, adding the phase would be easy because it will be sufficient to execute the calculated steps in the formalism, but with the phase. That's why we **will no longer consider the phase** for Pauli matrices and Clifford gates. Since there is no more ambiguity, the name Paulis will be used for Pauli matrices and Clifford for Clifford gates. This implies that the tableau representation of Paulis $Q$ and Clifford $C$ for the rest of the report will be represented by the tableau $T_Q$ and $T_C$ of the respective dimensions $1 \times 2n$ and $2n \times 2n$. On this tableau representation, it is possible to interpret $P$, $H$ and $CNOT$ as simple linear algebra operators. In fact, the swap performed by the Hadamard is simply done by multiplying the tableau representation by the matrix $T_H := I_{2n} - e_{k,k} - e_{n+k,n+k} + e_{n+k,k} + e_{k,n+k}$. It is the same for the phase which is a product on the right by $T_P := I_{2n} + e_{k,n+k}$. Finally, $CNOT$ is simply a multiplication on the right by $T_{CNOT} := I_{2n} + e_{k_1,k_2} + e_{n+k_2,n+k_1}$.

One can summarize the previous interpretation in Figure 3

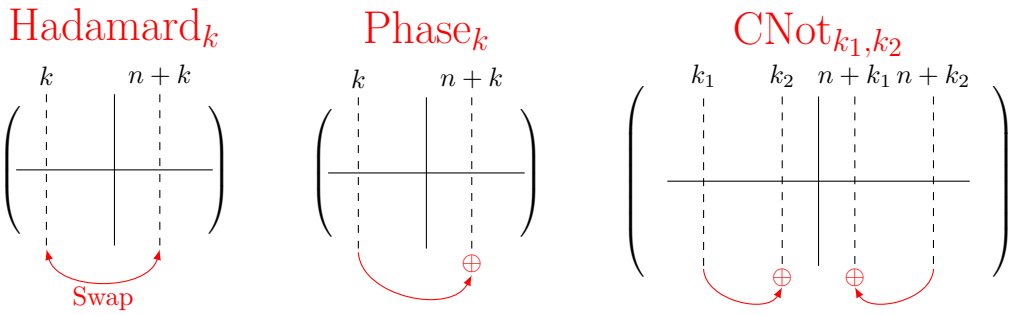



Figure 3: **Interpretation of the action of $H$, $P$ and $CNOT$** on the tableau representation of a Clifford.

**The tableau interpretation of Clifford composition.** Any Clifford $C$ can be represented as a tableau $T$. In the previous paragraph, we saw that the product of $T$ on its right side with $T_H, T_P$ and $T_{CNOT}$ implements the compositions $H \cdot C, P \cdot C$ and $CNOT \cdot C$ respectively. Since every Clifford can be implemented with a circuit composed of $P$, $H$ and $CNOT$ gates, there exists $S_1, \ldots, S_k$ a sequence of gates implementing $C$. Then $T_C = T_{S_1} \cdots T_{S_k}$.

**Proposition 3.1.** ***(Grier and Schaeffer [GS22])*** *If $C_1, C_2$ are Clifford operators on $n$ qubits then*

$$T_{C_2 C_1} = T_{C_1} T_{C_2}.$$

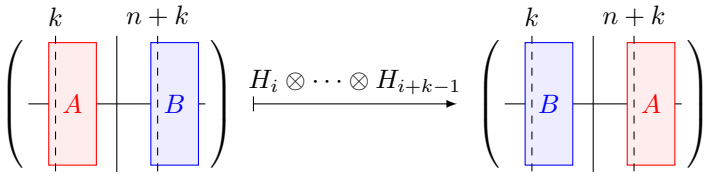
*Proof.* Both $C_1$ and $C_2$ can be computed with a sequence of $H, P$ and $CNOT$ gates $S_1$ and $S_2$. Then, $S_1$ concatenated to $S_2$ computes $C_2 C_1$ and by associativity of the matrix product

$$T_{C_2 C_1} = \prod_{c \in S_1 S_2} T_c = \left( \prod_{c \in S_1} T_c \right) \left( \prod_{c \in S_2} T_c \right) = T_{C_1} T_{C_2}.$$

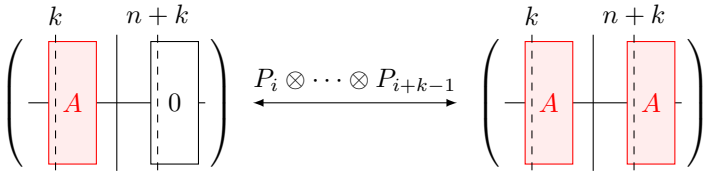
□

**The linear algebra point of view is important to explore.** Indeed, the previous explanations have helped us to understand how $H$, $P$ and $CNOT$ act on the tableau representation of a Clifford. However, they are not sufficient to easily understand the tools used by Aaronson and Gottesman in [AG04]. These tools are used directly, and it is difficult for the reader to fully understand what is going on without going through the details themselves. More precisely, **these tools are some linear algebra routines and tricks** that can be useful to any scientist who wants to work directly on some tableau representations of Cliffords or sets of Paulis.

**Proposition 3.2.** ***Swapping:*** *Let $T$ be any tableau representation, this tableau can be partitioned such that $A, B$ are two $2n \times i$ matrices starting respectively at index $k$ and $n+k$. The application of $i$ Hadamard gates from qubits $k$ to $k+i-1$ swaps submatrices $A$ and $B$.*




**Proposition 3.3.** ***Duplication:*** *Let $T$ be any tableau representation where $Z$ is not used for qubits from $k$ to $k+i-1$. The application of Phase gates on the qubit from $k$ to $k+i-1$ will copy and paste the corresponding $X$ submatrix onto the $Z$ one.*




**Proposition 3.4.** ***Gaussian elimination:*** *Let $T$ be a tableau representation of a $n$ qubits set of $2n$ Paulis. $T$ can be written $T = \left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right)$ for $A, B, C$, and $D$ some $n \times n$ matrices. It is possible to perform Gaussian elimination on one of these four matrices. However, the three other matrices will also be updated.*

First, without loss of generality, let's consider one wants to diagonalize $C$. First, $C$ has to be full rank else the algorithm fails. The Gaussian elimination algorithm, on matrices whose values are in $\mathbb{F}_2$, consists in doing a sequence of $C_j \leftarrow C_j \oplus C_i$ with $C_i$ the $i$-th column of the matrix. So, each of these operations can be performed by a call to $CNOT_{i,j}$. Indeed, $CNOT_{i,j}$ performs $T_{\_,j} \leftarrow T_{\_,j} \oplus T_{\_,i}$ and $T_{\_,n+i} \leftarrow T_{\_,n+i} \oplus T_{\_,n+j}$. These operations on the whole tableau representation not only multiply $C$ by $C^{-}1$, but also have an action on $A, B$, and $D$ so it also has to be tracked.

$$T = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \mapsto \begin{pmatrix} A' & B' \\ I & D' \end{pmatrix}$$

More precisely, this map is performed using a sequence $S_k$ of $CNOT$ gates whose tableau representations are of the form

$$\begin{pmatrix} I + e_{i_k,j_k} & 0 \\ 0 & I + e_{j_k,i_k} \end{pmatrix} \quad \text{with} \quad \left((I + e_{i_k,j_k})^T\right)^{-1} = I + e_{j_k,i_k}.$$

Let $M_i$ denotes $I + e_{i_k,j_k}$, the tableau representation of the $CNOT$ round is

$$\begin{aligned} S_1 \cdots S_k &= \begin{pmatrix} M_1 & 0 \\ 0 & (M_1^T)^{-1} \end{pmatrix} \cdots \begin{pmatrix} M_k & 0 \\ 0 & (M_k^T)^{-1} \end{pmatrix} \\ &= \begin{pmatrix} M_1 \cdots M_k & 0 \\ 0 & (M_1^T)^{-1} \cdots (M_k^T)^{-1} \end{pmatrix} \\ &= \begin{pmatrix} M_1 \cdots M_k & 0 \\ 0 & \left((M_1 \cdots M_k)^T\right)^{-1} \end{pmatrix}. \end{aligned}$$

Here, $M_1 \dots M_k$ is equal to $C^{-1}$ as it is how has been define the $S_i$.

$$= \begin{pmatrix} C^{-1} & 0 \\ 0 & ((C^{-1})^T)^{-1} \end{pmatrix} = \begin{pmatrix} C^{-1} & 0 \\ 0 & C^T \end{pmatrix}$$

Then, it implies that

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} C^{-1} & 0 \\ 0 & C^T \end{pmatrix} = \begin{pmatrix} AC^{-1} & BC^T \\ CC^{-1} & DC^T \end{pmatrix}.$$

Finally,

$$= \begin{pmatrix} A' & B' \\ I & D' \end{pmatrix}$$

**Proposition 3.5.** ***A symmetry property.*** *Let $T$ be a tableau representation of an $n$ qubits Clifford. If $T$ is equal to $\begin{pmatrix} A & B \\ I & D \end{pmatrix}$ then $D$ is symmetric.*

Indeed the $n$ bottom rows of $T$ commute together, since they are the image of the $Z_i$ which commute. This implies that for all $i, j$ belonging to $\{n+1, \dots, 2n\}$, the symplectic form $\Omega(T_i, T_j)$ is equal to 0. Finally, $D$ is symmetric, since

$$0 = \Omega(T_i, T_j) = \bigoplus_{k=1}^{n} \left( \underbrace{T_{i,k}}_{=1 \text{ if } i=j} \overbrace{T_{j,n+k}}^{D_{j,i}} \right) \oplus \left( \overbrace{T_{i,n+k}}^{D_{i,j}} \underbrace{T_{j,k}}_{=1 \text{ if } i=j} \right) = D_{i,j} \oplus D_{j,i} = 0. \tag{10}$$

These ways of using $CNOT$, $H$ and $P$ to build some linear algebra tools are really interesting in themselves, and writing the manual on how to use them helped me to build my intuition on Clifford's tableau representations. Then it is possible to find the rewritten proofs of Lemma 10.3 and Theorem 4 (Aaronson and Gottesman) in Appendix B to have some nice examples of applications with more explanation than in the original proof.

**Theorem 4** (**HCP rounds**). *Any Clifford can be implemented with a circuit consisting of 11 rounds of Hadamard, Phase and $CNOT$ gates in the sequence H-C-P-C-P-C-H-C-P-C.*

Moreover, the main motivation for the study of these linear algebra tools comes from the implementation side of a real algorithm. Indeed, in order to optimise the main algorithm, the tools will no longer be at the level of the Clifford group, but at the level of the generator and what can be directly implemented with them. We will discuss this issue in more detail in the future work section of the conclusion.

# 3 The Study of Pauli Sparsification

Let's recall, Problem 1, the problem we want to tackle.

**Problem 1** (**Pauli Sparsification**). *Let $A := \{Q_1, \dots, Q_m\}$ be a subset of $\mathcal{P}_n$. The weight of a set $A$ of Paulis corresponds to the maximal weight of its elements. The sparsification problem is about finding pairs $(A_i, C_i)$ for $i$ in $[1, k]$ such that the $A_i$ are a partition of $A$ and $C_i$ a Clifford operator such that the weight of the set $C_i A_i C_i^\dagger$, the conjugation of each element of $A_i$ by the Clifford gate $C_i$, is significantly smaller than the original one.*

The first thing is to check that the **Problem 1 is well defined**. For this, we consider the following example, where we show that for each Pauli there exists a Clifford mapping it to a weight one Pauli.

**Example 1.** *Let $Q$ belong to $\mathcal{P}_n$. There exists $C$ belonging to $\mathcal{C}_n$ such that $w(CQC^\dagger) = 1$.*

*Proof.* Let $C' \in \mathcal{C}_n$ be a Clifford that map the generator $X_1$ to $Q$. For other generators, it is possible to find elements of $\mathcal{P}_n$ that satisfy the (anti)commutativity relation. Then, $C := C'^\dagger$ implies that $CQC^\dagger = C'^\dagger Q (C'^\dagger)^\dagger = C'^\dagger Q C' = X_1$. □

It implies that any set of $m$ Paulis can be brought to weight one with partitions into $m$ sets of size one. The first step in studying the problem is to understand how the basic properties of Clifford can help us.

## 3.1 The anticommutating structure

A central property of Clifford is the preservation of the Pauli anticommutativity relations given in the input. On the scale of a set of Pauli strings, every internal pair either commutes or anticommutes. We have chosen to define **the anticommutation structure**, or the **canonical form** because we have to work with sets of Paulis and the anticommutation relation seems to play an important role in the achievable weight by sparsification. More precisely, the canonical form is a graph defined as follows

**Definition 5.** *Let $A = Q_1, \ldots, Q_k$ be a set of Paulis. **The canonical form** $\Delta(A)$ of the set $A$ is the graph whose vertices are elements of the set $A$ and where there is an edge between $(Q_i, Q_j)$ if and only if they anticommute together.*

Let $A = Q_1, \ldots, Q_k$ be a set of Pauli strings and $T_A$ it's tableau representation. The adjacency matrix of the canonical form is directly equal to $\Omega(T_A, T_A) = T_A S T_A^\dagger$ where $S$ is equal to $\left(\begin{smallmatrix} 0 & I_n \\ I_n & 0 \end{smallmatrix}\right)$. More precisely, it implies that the adjacency matrix belongs to $\mathcal{M}_k(\mathbb{F}_2)$ and has its element in row $i$ and column $j$ equals to $\Omega(Q_i, Q_j)$. It follows that the **conjugation by a Clifford $C$ induces a graph isomorphism** onto the canonical forms of $A$ and $CAC^\dagger$.

**Proposition 5.1.** *For all $C$ in $\mathcal{C}_n$, for all $A \subseteq \mathcal{P}_n$, $\Delta(A)$ is isomorphic to $\Delta(CAC^\dagger)$*

*Proof.* First, $(Q_i, Q_j)$ is an edge of $\Delta(A)$ if and only if $\Omega(Q_i, Q_j) = 1$. Then, by the anticommutativity preservation of Clifford ($\star$),

$$\forall i, j \in [1, m], \quad \Delta(CAC^\dagger)_{i,j} = \Omega(CQ_iC^\dagger, CQ_jC^\dagger) \overset{\star}{=} \Omega(Q_i, Q_j) = \Delta(A)_{i,j}.$$

□

We can now wonder if the converse holds.

**Question 2.** *Let $A_1$ and $A_2$ be two subsets of $\mathcal{P}_n$. Does $\Delta(A_1) \equiv \Delta(A_2)$ implies that it exists $C$ a Clifford operator such that $CA_1C^\dagger = A_2$?*

First, this theorem is a symplectic version of [HJ91, Theorem 7.3.11], which is a result of matrix analysis. It states that for every pair of matrices $A, B \in M_{k,2n}$ with the same Gram matrix (the Gram matrix of $A$ is $A^\dagger A$), there exists a matrix $O$ with orthonormal columns such that $A = OB$. Unfortunately, the symplectic version (Question 2) isn't as general as the orthogonal one. Indeed, $A_1 := \{XXI, YYI, ZZI\}$ and $A_2 := \{XII, IXI, IIX\}$ have the same canonical form since each pair commutes. However, there isn't a Clifford mapping $A_1$ and $A_2$ because $A_2$ is linearly independent, which is not the case for $A_1$. So the resulting Theorem 6 must be more restrictive.

**Theorem 6.** *Let $B_1$ and $B_2$ be **two linearly independent** sets of $\mathcal{P}_n$ with isomorphic canonical forms. Then, there exists $C$ such that $CB_1C^\dagger = B_2$.*

Before proving the Theorem 6, some notions have to be introduced. First, $(\mathcal{P}_n, .)$ with its symplectic form $\Omega$ can be seen as a symplectic vector space[2]. In addition, as for Hilbert spaces with their orthonormal bases (i.e. a basis $B$ where $\langle BI|B\rangle = I$), it is possible to define a symplectic-normal base $B$ such that $\Omega(B, B)$ is almost equal to $S := \left(\begin{smallmatrix} 0 & I_n \\ I_n & 0 \end{smallmatrix}\right)$.

**Definition 7.** *A **Symplectic-normal basis** $B$ of $\mathcal{P}_n$ is a basis of $P_n$ such that **each vertex in** $\Delta(B)$ **is in exactly one edge**, or equivalently it exists $S_\sigma$ a permutation of rows and columns of $S$ such that $\Omega(B, B) = S_\sigma$. The last characterization can be written also $\Delta(B) \equiv S$ with $\equiv$ the graph isomorphism relation. The canonical symplectic-normal basis $\mathcal{B}$ is defined as $\{X_1, Z_1, \ldots, X_n, Z_n\}$ and has for tableau representation $T_\mathcal{B} = I_{2n}$.*

These symplectic-normal bases are relevant to our formalism. Indeed, in each basis of $\mathcal{P}_n$ every element anticommutes with at least one other[3], it implies that every vertex belongs to at least one edge and finally that **symplectic-normal bases are minimal for the number of edges** (Each vertex is covered once). Moreover, every valid choice of $Q_i$ used to set up a Clifford corresponds to a symplectic-normal basis and vice versa. Thus,

**Lemma 7.1.** *For all symplectic-normal basis $B$ of $\mathcal{P}_n$ it exists $C_B$ in $\mathcal{C}_n$ such that $T_\mathcal{B} T_{C_B} = T_B$. Observe that $T_B = I_{2n}$ and so $T_B = T_{C_B}$.*

The notation $C_B$ will be used to denote the Clifford mapping the canonical symplectic-normal basis $\mathcal{B}$ to $B$. It directly implies a Corollary.

**Corollary 7.1.** *Let $B_1$ and $B_2$ be two symplectic-normal bases of $\mathcal{P}_n$. It exists a Clifford $C$ in $\mathcal{C}_n$ such that $CB_1C^\dagger = B_2$.*

*Proof.* Let $C := C_{B_2} C_{B_1}^\dagger$, then

$$T_{B_1} T_C = T_{B_1} T_{C_{B_2} C_{B_1}^\dagger} = T_{B_1} T_{C_{B_1}^\dagger} T_{C_{B_2}} = T_\mathcal{B} T_{C_{B_2}} = T_{B_2}$$

□

Here, Corollary 7.1 is a restriction of Theorem 6 as two **symplectic-normal bases have always isomorphic canonical forms**. In Theorem 6, the entries $A_1$ and $A_2$ are linearly independent subsets of $\mathcal{P}_n$. These sets can be expressed in any symplectic-normal basis $B$ as $A_1 = M_{A_1} T_B$ and $A_2 = M_{A_2} T_B$ with $M_{A_i}$ the coordinate of $A_i$ in $B$. Here, for the purposes of notation, $A_i$ designates $T_{A_i}$. Doing the same change of basis does not bring any information to build a Clifford to go from one to the other. However, if it exists two symplectic-normal bases $B_1$ and $B_2$, such that $A_1 = MT_{B_1}$ and $A_2 = MT_{B_2}$ (i.e. have the same coordinates) then by Corollary 7.1, it exists $C$ to go from $B_1$ to $B_2$ and finally

$$A_1 T_C = MT_{B_1} T_C = MT_{B_2} = A_2. \tag{11}$$

The difficulty is now moved to the **search of two symplectic-normal bases such that two linearly independent sets with isomorphic canonical forms have the same coordinates**. As this property holds for every pair of linearly independent sets having isomorphic canonical form, it implies that the coordinates depend only on the equivalence class by graph isomorphism of the canonical form. So, one needs to find a symplectic-normalization algorithm (Simplectic Gram-Schmidt) whose execution steps depend only on the canonical form. Before presenting our algorithm, let's recall and define some operations on graphs.

[2] A symplectic vector space is a vector space $V$ over a field $F$. It is also equipped with a symplectic bilinear form which is defined as a mapping of a pair of elements from the vector space to the field that satisfied: bilinearity, alternation $\Omega(w, w) = 0$ and non-degeneration

[3] Let suppose it exists an element in $B$ that commutes with all the others. Then, this element commutes with every element of the span of the basis so with every element of $P_n$ which is not possible as every element anticommutes with half of $P_n$. Finally, this element does not exist.

## 3.2 A very specific Symplectic version of Gram-Schmidt

Let $G$ be a graph, $V(G)$ (resp. $E(G)$) returns the list of vertices (resp. edges) of $G$. For any vertex, $Adj(v)$ is the list of vertices adjacent to $v$ in $G$. We define the $\nleftrightarrow$ binary operation between two vertices as follows:

**Definition 8.** *Let $v_1, v_2$ belong to $V(G)$. These two vertices have a value in $\mathcal{P}_n$. The $\nleftrightarrow$ operation from $v_2$ onto $v_1$, noted $v_1 \nleftrightarrow v_2$ is in two steps:*

- *Update the Pauli in $v_1$ with the product of the Paulis in $v_1$ and $v_2$.*
- *Update the edges according to the new anticommutation relation.*

This $\nleftrightarrow$ operation **translates the anticommutation relations of $v_2$ onto the already existing ones of $v_1$**. For example, let's take three Paulis $Q_1$, $Q_2$ and $Q_3$, the product $Q_1Q_2$ anticommutes with $Q_3$ if and only if one of $Q_1$ and $Q_2$ anticommutes with $Q_3$. Otherwise, no anticommutation with $Q_3$ is a trivial case and two anticommutations produce two -1 that cancel out. Also, $Q_1Q_2$ anticommutes with $Q_2$ if and only if $Q_1$ anticommutes with $Q_2$. It implies that $Q_1Q_2$ anticommutes with exactly the set of Paulis that were anticommuting with only one of $Q_1$ and $Q_2$. Now, as the operation $v_1 \nleftrightarrow v_2$ replace $v_1 = Q_1$ by $v_1 = Q_1Q_2$, the new adjacency list of $v_1$ is exactly $Adj(v_1) \oplus Adj(v_2) \setminus \{v_1\}$, with the set minus to ensure there is no loop. The Figure 4 shows an example of the operation on a graph.

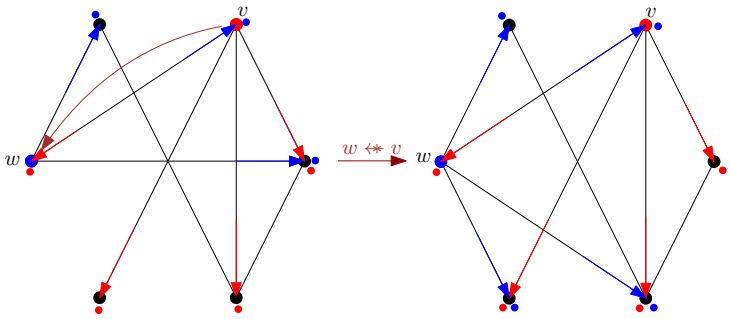



Figure 4: Illustration of $v_1 \nleftrightarrow v_2$ on a graph. On the left, small blue (resp. red) dots represent the adjacency of $w$ (resp. $v$). So, $w \nleftrightarrow v$ does nothing to the adjacency of $v$, however, it updates the adjacency of $w$ by setting the neighbours of $w$ as the vertices (except $w$) which are covered exactly once by blue and red dots.

After this little recap, one can finally detail the Algorithm 1 which corresponds to our implementation of a **symplectic Gram-Schmidt**. To prove its correctness, we will prove that

**Theorem 9.** *The Algorithm 1 takes in entry a linearly independent family $B$ of $\mathcal{P}_n$ and returns a symplectic-normal basis $S$ and the coordinates matrix $M$ such that $T_B = MT_{C_S}$ (or equivalently $B = C_S M C_S^\dagger$) with $M$* ***depending only on the equivalence class of $\Delta(B)$ by graph isomorphism****.*

*Proof.* This proof by induction follows Algorithm 1's steps and proves the algorithm's correctness. First, the algorithm works with three objects: $G$ is the canonical form associated with the entry $B$ and $Q$ is an empty stack of queries updating the value of $C$ which is set up as the canonical symplectic-normal basis. One wants to prove that vertices of $G$ are always generators of $B$.

**Initialisation** Before entering the while loop ($k = 0$), vertices of $G$ are exactly $B$.

**Induction** Suppose before step $k$ that vertices of $G$ generate $B$. **There are two possibilities**:

First, there is **a vertex $e_k$ with at least two neighbours**. Choose $(e_k, f_k)$ in $E(G)$ (i), the goal is to disconnect the edge from the rest of the graph. For every vertex $v$ of the adjacency of $e_k$ distinct from $f_k$, perform $v \nleftrightarrow f_k$ (iii) and store in the stack $S$ the update "$M_v \leftarrow M_v M_{f_k}$". The operation $v \nleftrightarrow f_k$ cancels the edge between $e_k$ and $v$ by updating the adjacency of $v$ with the

one from $f_k$. The store of "$M_v \leftarrow M_v M_{f_k}$" will be useful to revert the process to compute $B$'s coordinates in $S$ later. After that, the vertex $e_k$ is connected only to $f_k$.

As $(v \leftrightarrow\!\!* \ f_k) \leftrightarrow\!\!* \ f_k$ let the graph $v$ unchanged, the effect of $v \leftrightarrow\!\!* \ f_k$ can be reversed by $v \leftrightarrow\!\!* \ f_k$ and so **the $\leftrightarrow\!\!*$ operation does not affect the linearly independence** of vertices of the graph and $V(G)$ is still a basis of the span of $B$.

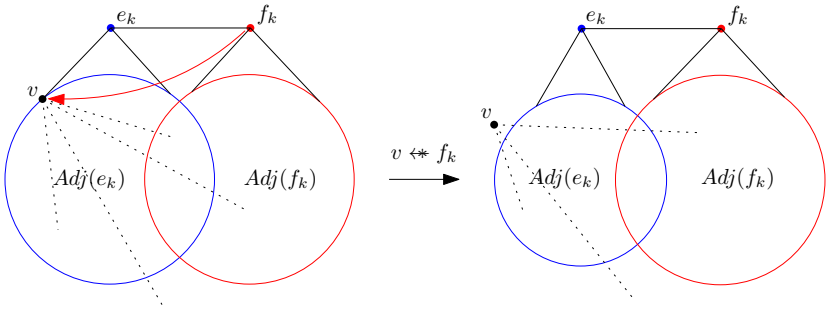



Figure 5: Illustration of the $\leftrightarrow\!\!*$ to isolate an edge.

After, **if $Adj(f_k)$ has more than one element** (iv) then for all vertices $v$ in the adjacency of $f_k$ distinct from $e_k$, perform $v \leftrightarrow\!\!* \ e_k$ (v) and store the update $M_v \leftarrow M_v M_{e_k}$ on the stack $S$ for later. $v \leftrightarrow\!\!* \ e_k$ removes the edge between $v$ and $f_k$ as $e_k$ belongs to the adjacency of $f_k$. Finally, the vertex $f_k$ is connected only to $e_k$ and the edge $(e_k, f_k)$ is disconnected from other vertices of the graph. As in the previous paragraph, the process is reversible so $V(G)$ is still a basis of the span of $B$.

**Otherwise**, there is no vertex with at least 2 neighbours. Then, it implies that vertices of $G$ form a basis of the span of $B$ and each of them is covered at most once. Finally, it implies that vertices of $G$ can be completed into a symplectic-normal basis of $\mathcal{P}_n$ on denote by $S$.

**Conclusion** Each step of the while ensures that 2 more vertices are covered at most once, so there are at most $\#B/2$ steps before exiting the while loop. Indeed, a step of the while loop can only affect the adjacency of vertices connected to the edge $(e_k, f_k)$. Thus, the adjacencies of previously disconnected vertex stay unchanged.

Now, to get the coordinates of $B$ in $S$, it is sufficient to execute the updates of $M$ specified by the elements of the stack popped one after the other. Indeed, for all graphs $G$ and for all vertices $v$ and $w$ of $G$, $(v \leftrightarrow\!\!* \ w) \leftrightarrow\!\!* \ w$ let $v$ unchanged and two updates, $(((v_1 \leftrightarrow\!\!* \ v_2) \leftrightarrow\!\!* \ v_3) \leftrightarrow\!\!* \ v_3) \leftrightarrow\!\!* \ v_2 =$ let also $v_1$ unchanged. Then it implies that the updates applied in reversed order on the graph $S$ compute $B$. In addition, all these queries are linear algebra operations on the rows of $S$. In fact, the $i$-th query $M_v \leftarrow M_v M_w$ can be represented by matrix multiplication of $S$ on the left by $Q_i := I_{\#B} + e_{v,w}$. Then,

$$T_B = Q_1 Q_2 \cdots Q_{\#Q} \underbrace{Q_{\#Q} \cdots Q_1 T_B}_{T_S} = \underbrace{Q_1 Q_2 \cdots Q_{\#Q} I_{\#B,2n}}_{M} T_S = M T_S. \tag{12}$$

Finally, the updates of $M$ added onto the stack depend only on the edge of the graph in the entry so on the equivalence class by graph isomorphism of the canonical form of the entry. Then, these updates are performed on $M$ whose initial value is independent of $B$. It directly implies that the value of coordinates $M$ only depends on the graph isomorphism class. □

Now, it is finally possible to state that Theorem 6 is proved. Indeed, let's recall the scheme of the proof.

*Proof.* Let $B_1$ and $B_2$ be two bases whose canonical form are isomorphic, then it exists $S_1$, $S_2$ some symplectic-normal basis and $M$ some coordinates (computed by Algorithm 1), such that $T_{B_i} = M T_{S_i}$ (Theorem 9). By Corollary 7.1, it exists $C$ a Clifford such that $T_{S_1} T_C = T_{S_2}$ which implies that $T_{B_1} T_C = M T_{S_1} T_C = M T_{S_2} = T_{B_2}$. □

**Algorithm 1** Simplectic Gram-Schmidt

**Input** $B$ a linearly independent set of $\mathcal{P}_n$
**Output** $M, S$ with $S$ a symplectic-normal basis where $M$ are $B$'s coordinate in $S$.
$G \leftarrow \Delta(B)$ /*The canonical form of $B$*/
$Q \leftarrow [\,]$ /*An empty query stack*/
$M \leftarrow I_{\#B,2n}$

/* Symplectic-normalization of vertices with at least 2 neighbours */
***while*** $\exists v \in V(G), \quad |Adj(v)| > 1$ ***do***
  ***choose*** $(e_k, f_k)$ *in* $E(G)$ *such that* $|Adj(e_k)| > 1$ (i)
  ***for*** $v \in Adj(e_k) \setminus \{f_k\}$ ***do*** (ii)
    $v \leftrightarrow\!\!* \ f_k$ (iii)
    ***push*** *the query "*$M_v \leftarrow M_v M_{f_k}$*"* ***on top of*** $Q$
  ***end for***
  ***if*** $\#Adj(f_k) > 1$ ***then*** (iv)
    ***for*** $v \in Adj(f_k) \setminus \{e_k\}$ ***do***
      $v \leftrightarrow\!\!* \ e_k$ (v)
      ***push*** *the query "*$M_v \leftarrow M_v M_{e_k}$*"* ***on top of*** $Q$
    ***end for***
  ***end if***
***end while***

$S \leftarrow V(G)$ *completed into a symplectic-normal basis*

/* Every vertex of $G$ is covered at most once,
let's compute $M$, coordinates of $B$ in $V(G)$ */ (vi)
***while*** $Q$ *is not empty* ***do***
  $q \leftarrow pop(Q)$ /* where $pop()$ returns and removes the top element of the stack */
  ***Execute*** $q$ (vii)
***end while***
***return*** $M, S$

In Algorithm 1, the way $M$ is built implies interesting results on its weight. Indeed, these coordinates satisfy $T_B T_{C_S^\dagger} = M$, and the traceability of the process to compute $M$ helps to bound the weight of $M$ in case of a sparsification using the Clifford $C_S^\dagger$.

**Lemma 9.1.** *$M_{v,w} = 1$ if and only if $v = w$ or the algorithm does the query $v \leftrightarrow\!\!* \ w$.*

*Proof.* First, $M$ is of the form $(I_{\#B}|0_{\#B,2n-\#B})$. Without loss of generality, let's suppose that the $2k$ first rows of $C$ are indexed by $e_i, f_i$ for $1 \leq i \leq k$ and $k$ the number of iterations of the while loop. Let's do the proof by induction on the number of queries applied in the reversed order.

**Initialization:** For 0 query applied in reverse order, there are ones only on the diagonal so for $v = w$.

**Induction:** Let suppose that after the $j$-th query applied in reverse order, there are ones in $M_{v,w}$ if and only if $v = w$ or a query $v \star w$ has been done. The $(j+1)$-th query is $v_1 \star e_i$. The row $e_i$ has still only one 1 since there is no query of the form $e_i \star v$ in the already applied query. Indeed, such queries imply that at some point $e_i$ is connected to $e_j$ of $f_j$ for some $j > i$ which is not possible. Thus, the line corresponding to $e_i$ is of weight one with the one in coordinate $e_i \times e_i$. Moreover, the algorithm never generates two times the same query. So it implies that $v_1 \star e_i$ was not already existing and that there is no 1 at coordinates $v_1, e_i$. So the addition of row $e_i$ on row $v_1$ just adds the one at coordinate $v_1, e_i$. Which was the 1 missing to have the lemma true at $j+1$.

**Conclusion** Each query $v \star w$ will exactly add a 1 at position $v \star w$ by adding rows only when they are unchanged. With the already existing ones on the diagonal, it proves the Lemma. □

This Lemma 9.1 implies that the **number of ones in the coordinate matrix is exactly equals to the size of the linearly independent family in entry plus the number of queries performed by**

**the algorithm**. More precisely, the number of ones in the $i$-th row corresponds to the number of updates targetting the $i-th$ row plus one. On the weight side, it implies that the weight of this row is bounded by the number of updates plus one, indeed two updates on two elements that anticommute implies 2 ones in the coordinate matrix but only a weight increase of one. So finally,

$$W := 1 + \max_{i\in[1,\#B]} \#\{q \in Q, q \text{ target the } i^{\text{th}} \text{ row}\} \geq w(M) \geq W/2. \tag{13}$$

This result has some interesting implications. First, the steps of the algorithm depend only on the graph isomorphism class of the canonical form so it is the same for the coordinate matrix and then for the weight. It directly implies that **our algorithm may not be optimal** as every element of the same equivalence classes will be sent to the same weight. Indeed, let $B$ be the equivalence class of the linearly independent set $A$ of Pauli string on $n$ qubits. For every $b \in B$ it exists $C \in \mathcal{C}_n$ such that $CAC^\dagger = b$. Thus, let's call $D$ the set of Paulis of minimal weight from $B$ and $w_D$ the minimal weight. The Algorithm 1 returns a set optimal for $A$ if the output belongs to $D$. If it does not append, it implies that the output weight $w$ is bigger than $w_D$ and so every element from $B$ with a weight lower than $w$ will be mapped to a set of higher weights. However, if the output has a weight **not optimal but still small**, as there are more high-weight than low-weight elements in $B$ then the **average weight in the output should be lower**.

There are other limitations to this algorithm. The first is that it only deals with linearly independent families of size at most $2n$, whereas the goal is to deal with all types of Pauli families. Another is that cliques are an example where the Algorithm 1 does not reduce the weight significantly. In fact, in a clique, the last pair of Paulis separated from each other has been updated by all the previous vertices. This implies that there are $2n-1$ ones(Lemma 9.1) in their rows in the coordinate matrix and so they are of weight $n$. It implies that **cliques and linear dependencies are bottlenecks** to the weight reduction capability of our algorithm. More generally, one can ask whether cliques are a bottleneck for all sparsification algorithms or just for Algorithm 1. We will try to answer this question in Section 4.

## 3.3 Naive implementation of Pauli sparsification

We have seen in the previous paragraph that the Algorithm 1 is not optimal. But one can ask whether it does sparsify on average. The first thing to consider here is the partitioning process. In fact, this aspect has been forgotten for the moment, as the work has focused on minimising the output weight for a given partition. A first preliminary result is a lower bound on the size of the partition required.

**Proposition 9.1.** *The number of partitions of $Q_1, \dots, Q_m$ needed is at least $\frac{m}{\#(\mathcal{P}_n^w)-1}$ with*

$$\#\mathcal{P}_n^w = \sum_{i=0}^{w} 3^i \binom{n}{i}$$

*Proof.* One can use the bijection property of Clifford operators. Indeed, if the cardinality of $\mathcal{P}_n^w$ is $r$, then a Clifford sends $r-1$ distinct elements (distinct from the identity) to $\mathcal{P}_n^w$. If there are $m$ Pauli strings in the entry then the size of the partition is at least $\lceil m/(r-1) \rceil$. Then, the cardinality of $\mathcal{P}_n^w$ is easy to compute. Indeed, $\mathcal{P}_n^w$ can be partitioned into $\mathcal{P}_n^{=i}$ for $i \in [0, w]$. Then, the cardinality of $\mathcal{P}_n^{=i}$ can be computed easily. The first step is to choose the $i$ indices where the non-identities are placed, so $\binom{n}{i}$ choices and the second step is to instantiate the indices with $X, Y$, or $Z$ so 3 choices per indice that turn to $3^i$ choices for the whole word. Thus $\#\mathcal{P}_n^{=i} = 3^i\binom{n}{i}$. Finally, by the partition property, $\#\mathcal{P}_n^w = \sum_{i\in[0,w]} \mathcal{P}_n^{=i} = \sum_{i\in[0,w]} 3^i\binom{n}{i}$. □

This result shows that **the more severe the weight constraint** in the output, **the more the number of sets in the partitions must increase**. For the naive algorithm, however, the problem considered will be simpler. In fact, the goal is to use the Algorithm 1 to do the sparsification and observe whether, on average, the maximum weight of a set of Paulis is reduced. This implies that **the partitioning process must return linearly independent families that minimise the size of the largest cliques from the input set**. For this reason, we have chosen a naive partitioning process. First, we consider in entry only linearly independent families, then the sparsification

consists in reducing the size of the largest connected components by dividing each of the connected components into $k$ parts to get a partition of size $k$. Then the Algorithm 1 is called on each of these $k$ sets. Finally, we can average the difference between the original maximum weight of the input and the maximum weight of the output. The results of some simulations are detailed in Figure 6. They help us to understand that on average the Algorithm 1 sparsifies the linearly independent sets given in the input. It also turns out that the arbitrarily chosen heuristic for partitioning the linearly independent set works and helps a lot in sparsifying with the Symplectic Gram Schmidt algorithm.

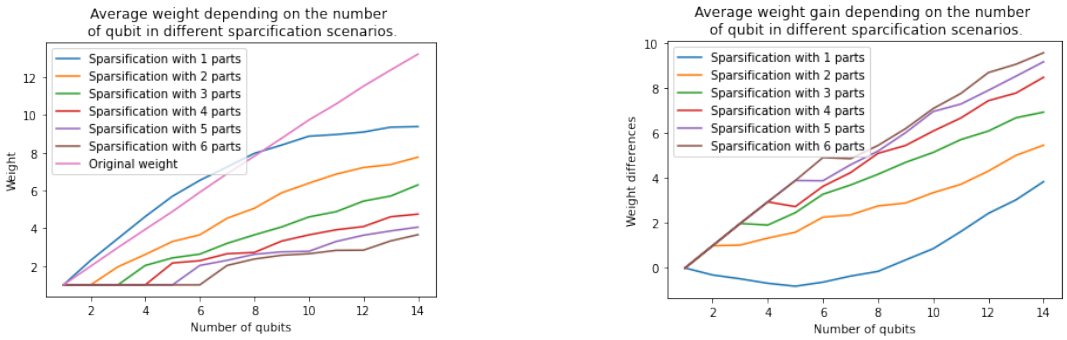



Figure 6: **On the left:** The diagram represents the average maximum weight of a linearly independent set of $2n$ Paulis for $n \in [1, 14]$ before and after sparsification with the number of sets of the partition varying into $[1, 6]$. **On the right:** The diagram represents the average gain on the maximum weight between a linearly independent set of $2n$ Pauli strings for $n \in [1, 14]$ before and after sparsification with the number of sets of the partition varying into $[1, 6]$.

# 4 A deeper study of cliques

As said in the previous paragraph, the algorithm described before has a bottleneck for the lower weight achievable in the size of the largest clique of the canonical form in entry. One may wonder whether this bottleneck is a **property of our algorithm or a more general property of cliques.** This section will aim to detail our research to establish such a property.

## 4.1 The biggest click of $\mathcal{P}_n^w$.

As mentioned before, the property of Clifford to be graph isomorphic implies that they don't affect the size of the largest click in the canonical forms. Thus, knowing the maximum size a clique can have in $\mathcal{P}_n^w$ is important to give a lower bound on the best achievable weight of a sparsification. For weights equal to $n$ the answer is already known to be $2n + 1$ [SB19; Hru14; BBO20], but for lower weights the problem is still open. A first thought would be to adapt the arguments of [Hru14; BBO20] to our lower weight scenarios, but all these proofs use some algebraic properties of the group structure of $\mathcal{P}_n$ that no longer exist with $\mathcal{P}_n^w$. In fact, the product is no longer an inner product for $\mathcal{P}_n^w$. This means that **a new formalism has to be found**.

**The pattern formalism of weight at most $w$.** Let's consider words on $\{0, 1\}$, where 1 stands for $X, Y$ or $Z$ and 0 stands only for $I$. These words are called patterns of weight at most $w$ if there are at most $w$ 1. Two patterns anticommute if and only if there exists an instantiation of both that makes them anticommute. It implies that two patterns anticommute if they have at least one 1 in common. The notions of **canonical forms and cliques can be generalised to the pattern formalism** with the previous definition of anticommutativity. It is then possible to ask the same question, what is the maximum size of a click in this formalism for a given weight? This problem is relevant to our question in $\mathcal{P}_n^w$, as it gives the maximum number of patterns that can be instantiated to find a largest clique of $\mathcal{P}_n$.

A first observation is that it does not seem possible to construct an infinite number of anticommuting patterns with a given weight. Indeed, it is possible to have at most $2k+1$ elements in a clique on $k$ qubits, which implies that if $2k+2$ patterns share the same $k$ ones, and if there is no other sharing between two of the patterns, then there cannot exist an instantiation of them that anticommute. For example, $11000, 10100, 10010, 10001$ cannot be instantiated into an anticommutative family of Paulis, since their anticommutativity depends only on the first qubits. This observation is explained by the following Theorem 10.

**Theorem 10.** *There is no anticommuting family of patterns of weight at most 2 that uses more than 4 different qubits for the placement of its ones.*

*Proof.* First, without loss of generality, the first choice is 1100, then the second has to anticommute with 1100, so 1010 is valid. Here, to anticommute with both and be different, there are 2 possibilities: First, the third can be 0110, but the fourth pattern should have a one on the 4-th qubit and so cannot anticommute with the first three. Second, the third can be 1001. To satisfy anticommutation and be distinct, the fourth pattern should be 10001. However, there are 4 Paulis whose anticommuting properties only come from the first qubit, which is not possible because the largest clique on a qubit is of size three. Here these two cases cover all cases up to some permutations of qubits. Then it implies that it is not possible to generate more than 3 different patterns that anticommute, moreover, all their 1s are grouped on at most the same four qubits. □

Then some simulations in Python let us think that this result about the number of qubits occupied by a clique of maximal size might generalise:

**Conjecture 1.** *For $w$ fixed, the amount of qubit used by a maximal clique of patterns* ***is bounded and becomes a constant*** *for $n$ greater than $N_w$ with $N_w$ being the indices where it starts to be constant given $w$.*

This result would be really interesting to prove, as it would help us to compute directly the maximum size of a clique in $\mathcal{P}_n^w$. Indeed, if one knows that for $n$ greater than $N_w$, the number of qubits used in the largest cliques of $\mathcal{P}_n^w$ is a constant, then it implies that the size of the largest click is also a constant equal to the size of the largest clique of $\mathcal{P}_{N_w}^w$. This would imply that computing the size of the largest clique of $\mathcal{P}_n^w$ for all $n$ depends only on $w$. With the Theorem 10 it is then possible to prove the following Corollary 10.1.

**Corollary 10.1.** *For $n \geq 4$ and $w = 2$, the size of the largest clique in $\Delta(\mathcal{P}_n^w)$ is a constant.*

*Proof.* A largest clique of $\Delta(\mathcal{P}_n^w)$ can only be an instantiation of a largest clique of patterns of size $n$ and weight at most $w$. Since all its ones are grouped on the same four qubits it implies that all the non-identities of the element of $S$ are on the same four qubits. It implies that every largest clique on $n$ qubits is also a largest clique on the four first qubits up to a permutation. It directly implies that the largest cliques on more than 4 qubits have the same size as the largest cliques on four qubits. □

**Smaller result on cliques.** The study of cliques has also led to the proof of some other **minor lemmas** that might be useful someday. A first one, for example, is that elements within cliques are almost linearly independent.

**Lemma 10.1.** *Let $Q_1, \ldots, Q_k$ be a set of anticommuting Paulis, then it is sufficient* ***to remove at most one element*** *in order to get a linearly independent family.*

*Proof.* Let $A = Q_1, \ldots, Q_k$ be a clique and let's suppose that $A$ is not a linearly independent family. Without loss of generality, there exists $m < k$ such that $Q_1, \ldots, Q_m$ is a basis. This implies that $Q_{m+1}$ can be written as $\prod_{1=i}^{m} Q_i^{\alpha_i}$ with $\alpha_i \in \{0, 1\}$. There are at least 2 values of $i$ such that $\alpha_i = 1$, otherwise it is an element of the basis or identity. Now there are two cases: First, if the set of indices $I$ where $\alpha_i = 1$ is of odd size, then it implies that $Q_{m+1}$ does not anticommute with every $Q_j$ for $j$ in $I$, so it is not possible. So it implies that the cardinality of $I$ is even, but then $Q_{m+1}$ doesn't anticommute with every element in $Q_j$ for $j$ in $[1, m] \setminus I$. The only way it works is if there are no elements in the family other than those in $I$, so $I = [0, k]$ with $k$ even. In this

case, there is only one element which is not in the basis and which is equal to the product of all elements of the basis. In summary, the family can be linearly independent or linearly dependent, in which case its size is odd and each element is equal to the product of all the others. □

Also, it is quite difficult to find a nice induction to build a maximum click of weight at most $w$ because of the following Lemma 10.2:

**Lemma 10.2.** *Let $A = Q_1, \ldots, Q_k$ be a biggest clique of $\mathcal{P}_n^w$. Then, we denote by $A_I$ (resp. $A_X, A_Y, A_Z$) the subset of $A$ of Paulis ending by $I$ (resp. $X, I, Z$) without their last gates. None of $A_q$ for $q$ in $X, Y, Z$ can be a biggest clique of $\mathcal{P}_{n-1}^w$ if $|A_I| \neq 0$ and vice versa.*

*Proof.* Without loss of generality, the next proof also works with X replaced by $Y$ and $Z$. Let's consider the case where $|A_I|, |A_X| > 0$. The anticommutation relation between elements of $A_I$ and $A_X$ before the restriction to $n-1$ first qubits comes from the same qubits as $I$ commutes with $X$, so it implies that every element of $A_I$ anticommutes with every element of $A_X$ and finally that $A_I \cup A_X$ is a larger click in $\mathcal{P}_{n-1}^w$. Since every element of $A_X$ anticommutes together because they share the same $n$-th qubit, the same proof still works and allows us to conclude that $A_X$ and $A_I$ don't belong to the largest cliques of $\mathcal{P}_{n-1}^w$. □

## 4.2 A construction of low-weight cliques

Another way to answer the question is to ask directly what is the smallest possible weight for a clique of size $k$. Indeed, having this information might give us a lower bound on the minimum weight achievable without partitioning, and might also motivate heuristics to compute partitions. For example, there is a nice way to **build low-weight cliques** of size 2n+1 with the iterative Algorithm 2.

**Algorithm 2** Low weight clique computation

**Input** $n$ the number of qubits
**Output** $S$ a set of Paulis
$S \leftarrow [X_1, Y_1, Z_1]$
**for** $i = 2$ to $n$ **do**
  $S \leftarrow S$ concatenates with $[S[0] * X_i, S[0] * Y_i, S[0] * Z_i]$
  Remove the first element of $S$
**end for**
**return** $S$

First, on one qubit a clique can only be of weight one so $\{X, Y, Z\}$ is the biggest one. Then the algorithm satisfies the following invariant after and before each loop step.

**Invariant step $k$:** It exists $l$ and $w$ such that the set $S = S_1, \ldots, S_k$ satisfy $w(S_i) = w$ for $i < l$ and $w(S_i) = w + 1$ otherwise. Moreover, $S$ is a clique.

**Checking the invariant** is easy. First, it is true before the loop. Then let's assume it's true after the $k^{\text{th}}$ iteration of the for loop, then the algorithm picks the first element $Q$ of $S$. Remove it from the set and add the same element at the end with another non-identity on a qubit never used before in the algorithm ($Q \times \{X_k, Y_k, Z_k\}$). This implies that the weight of the newly added Paulis at the end of the set is $w + 1$ and also that the 3 added Paulis are still anticommuting with all other elements of the set. To sum up, the invariant is still valid at the end of the step. It implies that at the end of the algorithm, the set $S$ is a clique of size $2n + 1 = 3 + (n-1)(-1+3)$.

It is also possible through this invariant to compute the weight of the clique for a given $n$.

**Proposition 10.1.** *The **weight** of the output of Algorithm 2 equals $\lceil \log_3(2n+1) \rceil$.*

*Proof.* For $n = 1$ the set consists of 3 Pauli strings of weight 1. It takes 3 steps to replace them all with 9 Pauli strings of weight 2, and another 9 steps to replace them all with 27 Pauli strings of weight 3. By repeating the pattern, we can deduce that if $\sum_{i=0}^{w-2} 3^i < n \leq \sum_{i=0}^{w-1} 3^i$ then the weight of the output is exactly $w$. So we have

$$\sum_{i=0}^{w-2} 3^i < n \leq \sum_{i=0}^{w-1} 3^i \implies w - 1 < \log_3(2n+1) \leq w \implies w = \lceil \log_3(2n+1) \rceil.$$

□

This result is important because it shows that the **Algorithm 1 gives some results that are far from optimal when applied to cliques**. Moreover, it shows that cliques, in general, are not a bottleneck for the achievable weight, which is important because **it does not refute the existence of efficient Pauli sparsification algorithms**.

# 5 Conclusion

First, our work on the Pauli sparsification problem was interesting and stimulating because of its diversity and implications. As a combinatorist at heart, I was very happy to discover Paulis and Cliffords. Indeed, the (anti)commutativity property of Paulis and its preservation by Clifford gates brings a really interesting combinatorial aspect to these well-studied algebraic objects. Moreover, the addition of weight makes these combinatorial structures even more interesting as everything is to be done. For example, many questions about cliques, such as the size of the largest clique in $\mathcal{P}_n^w$, stay open despite the multiple approaches we tried. Moreover, the Pauli sparsification problem itself is far from being solved. Through motivations, we have shown that sparsification can be useful to reduce the noise when doing Pauli measurements, and in Section 3, our constructive proof of Theorem 6 has helped to develop a simple algorithm to sparsify some linearly independent sets of Paulis. Simulations have then shown that, despite its simplicity, it sparsifies well on average, which is encouraging for the future. Furthermore, Section 4 has shown that the clique bottleneck of our algorithm is not intrinsic to the Cliffords operators, so some future algorithms may no longer suffer from it.

Then, many important and interesting aspects of Pauli sparsification weren't developed in this work. For example, the partitioning part of sparsification has been discussed briefly and needs to be explored in more depth in order to have better coherence between the sparsification algorithm and the set of entries. However, the main challenges for Pauli sparsification will come from its implementation on real quantum computers. Indeed, having a Clifford that sparsifies a set of Paulis is useless if the number of gates needed for the implemention it is huge. This means that future research may need to think of sparsification in terms of the generators of the Clifford group rather than the group itself. This is also where the preliminary section on tableau representation takes on its full meaning, as the tools presented will be used directly to compute the sparsification unitaries. Furthermore, the current network architecture of quantum computers makes 2 qubit gates expensive to use. First, it is expensive because a polynomial number of $SWAP$ gates must be used to connect the two qubits to which the gate is to be applied in the qubit network. Thus, a random quantum circuit can have a polynomially increasing depth in the number of qubits. Second, any of the gates can also fail, increasing the noise in the output. Future work should therefore address this challenge by keeping the qubits that need to interact dynamically close to each other during the execution of the quantum circuit.

Finally, I would like to thank Daniel Stilck França for this internship where I discovered many different topics and researchers related to quantum computing. It has also convinced me that quantum computing and its combinatorial problem are really interesting and worth working on for more than 3 years.

## References


[Ben80] Paul Benioff. "The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines". In: *Journal of Statistical Physics* 22(5):653-591 (1980).

[Fey86] Richard Feymann. "Quantum mechanical computers". In: *Foundamentale Physics* 16, 507-531 (1986).

[HJ91] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991.

[Sho94] P.W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134.

[Got97] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. 1997. arXiv: quant-ph/9705052 [quant-ph].

[Got98a] Daniel Gottesman. *The Heisenberg Representation of Quantum Computers*. 1998. arXiv: quant-ph/9807006 [quant-ph].

[Got98b] Daniel Gottesman. "Theory of fault-tolerant quantum computation". In: *Physical Review A* 57.1 (Jan. 1998), pp. 127–137.

[AG03] Scott Aaronson and Daniel Gottesman. *CHP: CNOT-Hadamard-Phase*. 2003.

[AG04] Scott Aaronson and Daniel Gottesman. "Improved simulation of stabilizer circuits". In: *Physical Review A* 70.5 (Nov. 2004).

[AG05] Andris Ambainis and Daniel Gottesman. *The Minimum Distance Problem for Two-Way Entanglement Purification*. 2005. arXiv: quant-ph/0310097 [quant-ph].

[KS06] Andreas Klappenecker and Pradeep Kiran Sarvepalli. *Clifford Code Constructions of Operator Quantum Error Correcting Codes*. 2006. arXiv: quant-ph/0604161 [quant-ph].

[Ozo08] Maris Ozols. "Clifford group". In: (2008).

[Hru14] Pavel Hrubeš. *On families of anticommuting matrices*. 2014. arXiv: 1412.5893 [cs.CC].

[Tra+18] Andrew Tranter et al. "A Comparison of the Bravyi–Kitaev and Jordan–Wigner Transformations for the Quantum Simulation of Quantum Chemistry". In: *Journal of Chemical Theory and Computation* 14.11 (Sept. 2018), pp. 5617–5630.

[JGM19] Andrew Jena, Scott Genin, and Michele Mosca. *Pauli Partitioning with Respect to Gate Sets*. 2019. arXiv: 1907.07859 [quant-ph].

[SB19] Rahul Sarkar and Ewout van den Berg. *On sets of commuting and anticommuting Paulis*. 2019. arXiv: 1909.08123 [quant-ph].

[BBO20] Xavier Bonet-Monroig, Ryan Babbush, and Thomas E. O'Brien. "Nearly Optimal Measurement Scheduling for Partial Tomography of Quantum States". In: *Physical Review X* 10.3 (Sept. 2020).

[GS22] Daniel Grier and Luke Schaeffer. "The Classification of Clifford Gates over Qubits". In: *Quantum* 6 (June 2022), p. 734.

[AKH] Scott Aaronson, Greg Kuperberg, and Olivier Habryka. *Check https://complexityzoo.net for more detail.*

# 6 Appendix

## List of Figures


1 IBM 433 qubits quantum computer's network . . . . . 1
2 Illustration of the example where one wants to measure $\rho$ with $Z^{\otimes n}$. . . . . 3
3 Linear algebra interpretation of $H$, $P$ and $CNOT$ action. . . . . 8
4 Illustration of $v_1 \nleftrightarrow v_2$ . . . . . 13
5 Illustration of the $\nleftrightarrow$ to isolate an edge. . . . . 14
6 Experimental results for the naive Sparsification algorithm . . . . . 17


## List of Algorithms


1 Simplectic Gram-Schmidt . . . . . 15
2 Low weight clique computation . . . . . 19


## Github repository with code of the simulations.

https://github.com/MaximeCautres/Optimizing-quantum-measurements-through-Pauli-sparsification

# A The behaviour of the product of Pauli matrices onto their tableau representations.

$$Q_1Q_2 = i^{(T_{Q_1})_{2n+1}}(-1)^{(T_{Q_1})_{2n+2}}\prod_{k=1}^{n} X_k^{(T_{Q_1})_k(1-(T_{Q_1})_{n+k})} Z_k^{(T_{Q_1})_{n+k}(1-(T_{Q_1})_k)} Y_k^{(T_{Q_1})_k(T_{Q_1})_{n+k}}$$
$$i^{(T_{Q_2})_{2n+2}}(-1)^{(T_{Q_2})_{2n+2}}\prod_{k=1}^{n} X_k^{(T_{Q_2})_k(1-(T_{Q_2})_{n+k})} Z_k^{(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)} Y_k^{(T_{Q_2})_k(T_{Q_2})_{n+k}}$$

They are commuting for different $k$ so it is possible to bring the same $k$ together.

$$= i^{(T_{Q_1})_{2n+1}+(T_{Q_2})_{2n+2}}(-1)^{(T_{Q_1})_{2n+2}+(T_{Q_2})_{2n+2}}\prod_{k=1}^{n}\Big[X_k^{(T_{Q_1})_k(1-(T_{Q_1})_{n+k})} Z_k^{(T_{Q_1})_{n+k}(1-(T_{Q_1})_k)}$$
$$Y_k^{(T_{Q_1})_k(T_{Q_1})_{n+k}} X_k^{(T_{Q_2})_k(1-(T_{Q_2})_{n+k})} Z_k^{(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)} Y_k^{(T_{Q_2})_k(T_{Q_2})_{n+k}}\Big]$$

anticommutation induces a certain amount of $-1$

$$= i^{(T_{Q_1})_{2n+1}\oplus(T_{Q_2})_{2n+2}}(-1)^{(T_{Q_1})_{2n+2}+(T_{Q_2})_{2n+2}+(T_{Q_1})_{2n+1}(T_{Q_2})_{2n+2}}$$
$$\prod_{k=1}^{n}\Big[(-1)^{(T_{Q_2})_k(1-(T_{Q_2})_{n+k})\left((T_{Q_1})_k(T_{Q_1})_{n+k}+(T_{Q_1})_{n+k}(1-(T_{Q_1})_k)\right)}$$
$$X_k^{(T_{Q_1})_k(1-(T_{Q_1})_{n+k})+(T_{Q_2})_k(1-(T_{Q_2})_{n+k})}(-1)^{(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)(T_{Q_1})_k(T_{Q_1})_{n+k}}$$
$$Z_k^{(T_{Q_1})_{n+k}(1-(T_{Q_1})_k)+(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)} Y_k^{(T_{Q_1})_k(T_{Q_1})_{n+k}+(T_{Q_2})_k(T_{Q_2})_{n+k}}\Big]$$

Adding even values to rewrite it in the form of the tableau of the product does change the value modulo 2.

$$
\begin{aligned}
= i^{(T_{Q_1})_{2n+1} \oplus (T_{Q_2})_{2n+2}} (-1)^{(T_{Q_1})_{2n+2} + (T_{Q_2})_{2n+2} + (T_{Q_1})_{2n+1}(T_{Q_2})_{2n+2}} \\
\prod_{k=1}^{n} \Big[ (-1)^{(T_{Q_2})_k (1-(T_{Q_2})_{n+k}) \left( (T_{Q_1})_k (T_{Q_1})_{n+k} + (T_{Q_1})_{n+k} (1-(T_{Q_1})_k) \right)} \\
(-1)^{(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)(T_{Q_1})_k (T_{Q_1})_{n+k}} \\
X_k^{(T_{Q_1})_k (1-(T_{Q_1})_{n+k}) + (T_{Q_2})_k (1-(T_{Q_2})_{n+k}) + 2(T_{Q_1})_k (T_{Q_2})_{n+k} + 2(T_{Q_1})_{n+k} (T_{Q_2})_k} \\
Z_k^{(T_{Q_1})_{n+k} (1-(T_{Q_1})_k) + (T_{Q_2})_{n+k} (1-(T_{Q_2})_k) + 2(T_{Q_1})_k (T_{Q_2})_{n+k} + 2(T_{Q_1})_{n+k} (T_{Q_2})_k} \\
Y_k^{(T_{Q_1})_k (T_{Q_1})_{n+k} + (T_{Q_2})_k (T_{Q_2})_{n+k}} \Big]
\end{aligned}
$$

Bringing them together implies a certain amount of anticommutations.

$$
\begin{aligned}
= i^{(T_{Q_1})_{2n+1} \oplus (T_{Q_2})_{2n+2}} (-1)^{(T_{Q_1})_{2n+2} + (T_{Q_2})_{2n+2} + (T_{Q_1})_{2n+1}(T_{Q_2})_{2n+2}} \\
\prod_{k=1}^{n} \Big[ (-1)^{(T_{Q_2})_k (1-(T_{Q_2})_{n+k}) \left( (T_{Q_1})_k (T_{Q_1})_{n+k} + (T_{Q_1})_{n+k} (1-(T_{Q_1})_k) \right)} \\
(-1)^{(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)(T_{Q_1})_k (T_{Q_1})_{n+k}} \\
X_k^{(T_{Q_1})_k (1-(T_{Q_1})_{n+k}) + (T_{Q_2})_k (1-(T_{Q_2})_{n+k}) + (T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k} \\
Z_k^{(T_{Q_1})_{n+k} (1-(T_{Q_1})_k) + (T_{Q_2})_{n+k} (1-(T_{Q_2})_k) + (T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k} \\
(-1)^{\left( (T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k \right) \left( (T_{Q_1})_k (1-(T_{Q_1})_{n+k}) + (T_{Q_2})_k (1-(T_{Q_2})_{n+k}) \right)} \\
X_k^{(T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k} Z_k^{(T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k} \\
Y_k^{(T_{Q_1})_k (T_{Q_1})_{n+k} + (T_{Q_2})_k (T_{Q_2})_{n+k}} \Big]
\end{aligned}
$$

Reorder this $X_k$ and $Z_k$ to form some $Y_k$ may induces $-1$.

$$
\begin{aligned}
= i^{(T_{Q_1})_{2n+1} \oplus (T_{Q_2})_{2n+2}} (-1)^{(T_{Q_1})_{2n+2} + (T_{Q_2})_{2n+2} + (T_{Q_1})_{2n+1}(T_{Q_2})_{2n+2}} \\
\prod_{k=1}^{n} \Big[ (-1)^{(T_{Q_2})_k (1-(T_{Q_2})_{n+k}) \left( (T_{Q_1})_k (T_{Q_1})_{n+k} + (T_{Q_1})_{n+k} (1-(T_{Q_1})_k) \right)} \\
(-1)^{(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)(T_{Q_1})_k (T_{Q_1})_{n+k}} \\
(-1)^{\left( (T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k \right) \left( (T_{Q_1})_k (1-(T_{Q_1})_{n+k}) + (T_{Q_2})_k (1-(T_{Q_2})_{n+k}) \right)} \\
X_k^{(T_{Q_1})_k (1-(T_{Q_1})_{n+k}) + (T_{Q_2})_k (1-(T_{Q_2})_{n+k}) + (T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k} \\
Z_k^{(T_{Q_1})_{n+k} (1-(T_{Q_1})_k) + (T_{Q_2})_{n+k} (1-(T_{Q_2})_k) + (T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k} \\
(-1)^{\frac{((T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k)((T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k - 1)}{2}} \\
(X_k Z_k)^{(T_{Q_1})_k (T_{Q_2})_{n+k} + (T_{Q_1})_{n+k} (T_{Q_2})_k} \\
Y_k^{(T_{Q_1})_k (T_{Q_1})_{n+k} + (T_{Q_2})_k (T_{Q_2})_{n+k}} \Big]
\end{aligned}
$$

$Y_k$ is not directly $XZ$ as a phase is showing.

$$
\begin{aligned}
Q_1Q_2 = i^{(T_{Q_1})_{2n+1}\oplus(T_{Q_2})_{2n+2}}(-1)^{(T_{Q_1})_{2n+2}+(T_{Q_2})_{2n+2}+(T_{Q_1})_{2n+1}(T_{Q_2})_{2n+2}} \\
&\prod_{k=1}^{n}\Big[(-1)^{(T_{Q_2})_k(1-(T_{Q_2})_{n+k})\left((T_{Q_1})_k(T_{Q_1})_{n+k}+(T_{Q_1})_{n+k}(1-(T_{Q_1})_k)\right)} \\
&(-1)^{(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)(T_{Q_1})_k(T_{Q_1})_{n+k}} \\
&(-1)^{\left((T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k\right)\left((T_{Q_1})_k(1-(T_{Q_1})_{n+k})+(T_{Q_2})_k(1-(T_{Q_2})_{n+k})\right)} \\
&(-1)^{\frac{((T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k)((T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k-1)}{2}} \\
&X_k^{(T_{Q_1})_k(1-(T_{Q_1})_{n+k})+(T_{Q_2})_k(1-(T_{Q_2})_{n+k})+(T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k} \\
&Z_k^{(T_{Q_1})_{n+k}(1-(T_{Q_1})_k)+(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)+(T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k} \\
&(-i)^{(T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k}(Y_k)^{(T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k} \\
&Y_k^{(T_{Q_1})_k(T_{Q_1})_{n+k}+(T_{Q_2})_k(T_{Q_2})_{n+k}}\Big] \\
= i^{(T_{Q_1})_{2n+1}\oplus(T_{Q_2})_{2n+2}}(-1)^{(T_{Q_1})_{2n+2}+(T_{Q_2})_{2n+2}+(T_{Q_1})_{2n+1}(T_{Q_2})_{2n+2}} \\
&\prod_{k=1}^{n}\Big[(-1)^{(T_{Q_2})_k(1-(T_{Q_2})_{n+k})\left((T_{Q_1})_k(T_{Q_1})_{n+k}+(T_{Q_1})_{n+k}(1-(T_{Q_1})_k)\right)} \\
&(-1)^{(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)(T_{Q_1})_k(T_{Q_1})_{n+k}} \\
&(-1)^{\left((T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k\right)\left((T_{Q_1})_k(1-(T_{Q_1})_{n+k})+(T_{Q_2})_k(1-(T_{Q_2})_{n+k})\right)} \\
&(-1)^{\frac{((T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k)((T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k-1)}{2}} \\
&(-i)^{(T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k} \\
&X_k^{(T_{Q_1})_k(1-(T_{Q_1})_{n+k})+(T_{Q_2})_k(1-(T_{Q_2})_{n+k})+(T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k} \\
&Z_k^{(T_{Q_1})_{n+k}(1-(T_{Q_1})_k)+(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)+(T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k} \\
&Y_k^{(T_{Q_1})_k(T_{Q_1})_{n+k}+(T_{Q_2})_k(T_{Q_2})_{n+k}+(T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k}\Big]
\end{aligned}
$$

It is finally possible to factorize.

$$
\begin{aligned}
= i^{(T_{Q_1})_{2n+1}\oplus(T_{Q_2})_{2n+2}}(-1)^{(T_{Q_1})_{2n+2}+(T_{Q_2})_{2n+2}+(T_{Q_1})_{2n+1}(T_{Q_2})_{2n+2}} \\
&\prod_{k=1}^{n}\Big[(-1)^{(T_{Q_2})_k(1-(T_{Q_2})_{n+k})\left((T_{Q_1})_k(T_{Q_1})_{n+k}+(T_{Q_1})_{n+k}(1-(T_{Q_1})_k)\right)} \\
&(-1)^{(T_{Q_2})_{n+k}(1-(T_{Q_2})_k)(T_{Q_1})_k(T_{Q_1})_{n+k}} \\
&(-1)^{\left((T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k\right)\left((T_{Q_1})_k(1-(T_{Q_1})_{n+k})+(T_{Q_2})_k(1-(T_{Q_2})_{n+k})\right)} \\
&(-1)^{\frac{((T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k)((T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k-1)}{2}} \\
&(-i)^{(T_{Q_1})_k(T_{Q_2})_{n+k}+(T_{Q_1})_{n+k}(T_{Q_2})_k} \\
&X_k^{((T_{Q_1})_k+(T_{Q_2})_k)(1-((T_{Q_1})_{n+k}+(T_{Q_2})_{n+k}))} \\
&Z_k^{((T_{Q_1})_{n+k}+(T_{Q_2})_{n+k})(1-((T_{Q_1})_k+(T_{Q_2})_k))} \\
&Y_k^{((T_{Q_1})_k+(T_{Q_2})_k)((T_{Q_1})_{n+k}+(T_{Q_2})_{n+k})}\Big]
\end{aligned}
$$

And to find by identification the formula satisfied by the tableau representation of $Q_1Q_2$. Let $f_i$ and $f_{-1}$ be two function of $T_{Q_1}$ and $T_{Q_2}$ that compute

$$
Q_1Q_2 = i^{f_i(T_{Q_1},T_{Q_2})}(-1)^{f_{-1}(T_{Q_1},T_{Q_2})}\prod_{k=1}^{n}X_k^{(T_{Q_1Q_2})_k(1-(T_{Q_1Q_2})_{n+k})}Z_k^{(T_{Q_1Q_2})_{n+k}(1-(T_{Q_1Q_2})_k)}Y_k^{(T_{Q_1Q_2})_k(T_{Q_1Q_2})_{n+k}}
$$

## B Detailed proof of the HCP round theorem by Aaronson and Gottesman

First, let's prove a technical lemma used in the main proof later.

**Lemma 10.3.** *Let $T$ be a tableau representation only for $Z_1$ to $Z_n$. The application of some Hadamard gates can make the left half of the tableau representation full rank.*

*Proof.* First, it exists $k$ such that the left half of the tableau representation is of rank $k$. So, it is possible to perform a Gaussian elimination on the rows to obtain the tableau in the form

$$\begin{pmatrix} A & B \\ 0 & C \end{pmatrix}$$

where $A$ is a $k \times n$ matrix of rank $k$. This Gaussian elimination can be performed by applying a change of basis on the input. This change of basis has a matrix representation $P$ so the current tableau is $PT$. Then, $T$ is of rank $n$, $C$ is of rank $n-k$ as it is linearly independent of $A$. It implies that it exists a permutation of columns $P_2$ such $C_2$ is full rank.

$$P_1 T \begin{pmatrix} P_2 & 0 \\ 0 & P_2 \end{pmatrix} = \begin{pmatrix} A_1 & A_2 & B_1 & B_2 \\ 0 & 0 & C_1 & C_2 \end{pmatrix}$$

By another Gaussian elimination by a left base change $P_3$, it is possible to map $C_2$ to $I_{n-k}$.

$$P_3 P_1 T \begin{pmatrix} P_2 & 0 \\ 0 & P_2 \end{pmatrix} = \begin{pmatrix} A_1 & A_2 & B_1 & B_2 \\ 0 & 0 & D & I_{n-k} \end{pmatrix}$$

Now, the commutativity of the $Z$ generators implies that $\langle \overline{A_1 A_2 B_1 B_2} | S\overline{00DI_{n-k}} \rangle = 0$ and so that $A_1 D^T = A_2$. As $A_2$ is a linear combination of the columns of $A_1$. Thus, $\overline{A_1 A_2}$ of column rank $k$ implies that $A_1$ is of rank $k$. Next, the right most columns are moved to the left by the application of $n-k$ Hadamard gates. Finally, the final table looks like

$$P_3 P_1 T \begin{pmatrix} P_2 & 0 \\ 0 & P_2 \end{pmatrix} H_{[n-k+1,n]} = \begin{pmatrix} A_1 & B_2 & B_1 & A_2 \\ 0 & I_{n-k} & D & 0 \end{pmatrix}$$

In addition, the left half tableau is composed of two linear independent sub-matrices of rank $k$ and $n-k$ so the global rank is $n$. However, this no more corresponds to the tableau formalism from the beginning. Indeed, the left basis is no more $Z_1$ to $Z_n$ and the output basis has seen a permutation. Let reverse the operations that have been done.

$$P_1^{-1} P_3^{-1} P_3 P_1 T \begin{pmatrix} P_2 & 0 \\ 0 & P_2 \end{pmatrix} H_{[n-k+1,n]} \begin{pmatrix} P_2 & 0 \\ 0 & P_2 \end{pmatrix}^{-1} = P_1^{-1} P_3^{-1} \begin{pmatrix} A_1 & B_2 & B_1 & A_2 \\ 0 & I_{n-k} & D & 0 \end{pmatrix} \begin{pmatrix} P_2 & 0 \\ 0 & P_2 \end{pmatrix}^{-1}$$

The reversal does a change of basis on the left so it does not change the rank. It does also two times the same permutation of columns for left and right half so it does not affect the rank of both half tableau. This tableau can be obtained from the original one because of the changes of basis on the left cancel out and the permutations on the right side simply modify the qubit on which the Hadamard gates were applied. Finally, the $TH_{\sigma_{P_2}([n-k+1,n])}$'s left half sub tableau is full rank. □

Another intermediate is required to prove their main result.

**Lemma 10.4.** ***(Aaronson and Gottesman [AG04])*** *For any symmetric matrix $A \in Z_2^{n \times n}$ , there exists a diagonal matrix $\Lambda$ such that $A + \Lambda = MM^T$, with $M$ some invertible binary matrix.*

Finally, the second one is the theorem about the efficient construction of any Clifford with a polynomial number of $CNOT$, Hadamard and Phase gates.

**Theorem 4** (**HCP rounds**)**.** *Any Clifford can be implemented with a circuit consisting of 11 rounds of Hadamard, Phase and $CNOT$ gates in the sequence H-C-P-C-P-C-H-C-P-C.*

*Proof.* Let T be the tableau representation of a Clifford on $n$ qubits. $T$ can be written $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ for $A, B, C$, and $D$ some $n \times n$ matrices on $\mathbb{F}_2$. The goal is to find a sequence of $H$, $P$ and $CNOT$ such that $T$ became $I_{2n}$. It is then sufficient to reverse the sequence in order to implement $T$. The algorithm is as follows.

1. The matrix $C$ have to become full rank. By Lemma 10.3, the use of $H$ gates on some qubits allow ending with $C$ full rank. This round of Hadamard gates also affects other sub matrices but to keep it simple names stay the same and this simplification hold for the rest of the proof.
$$\begin{pmatrix} A & B \\ C' & D \end{pmatrix}$$

2. The matrix $C'$ is mapped to the identity by the use of $CNOT$ gates to perform a Gaussian elimination as $C'$ is now full rank (Proposition 3.4). $A, B$, and $D$ are updated by each $CNOT$ but they do not have important property so there are no issues.
$$\begin{pmatrix} A & B \\ I & D \end{pmatrix}$$

3. By Proposition 3.5, $D$ is symmetric because of the commutativity of $Z_1$ to $Z_n$. So by Lemma 10.4, it exists $\Lambda$ a diagonal matrix and $M$ a $2n \times 2n$ lower triangular matrix such that $D + \Lambda = MM^T$.

   Then, Proposition 3.3 implies that it is possible to add any diagonal matrix to $D$ as the bottom left one is $I$ with a round of Phase gates.

   In addition, $M$ is full rank so a round of $CNOT$ gates can do a reverse Gaussian elimination on the bottom left matrices (Proposition 3.4). Here, Proposition 3.4 is applied in reverse so, in the property result, $C^{-1}$ is replaced by $M$ and
$$\begin{pmatrix} A & B \\ I & D+\Lambda \end{pmatrix} \begin{pmatrix} M & 0 \\ 0 & \left(M^T\right)^{-1} \end{pmatrix} = \begin{pmatrix} AM & B\left(M^T\right)^{-1} \\ M & (D+\Lambda)\left(M^T\right)^{-1} \end{pmatrix}.$$
   As $D + \Lambda = MM^T$,
$$\begin{pmatrix} AM & B\left(M^T\right)^{-1} \\ M & (D+\Lambda)\left(M^T\right)^{-1} \end{pmatrix} = \begin{pmatrix} AM & B\left(M^T\right)^{-1} \\ M & MM^T\left(M^T\right)^{-1} \end{pmatrix} = \begin{pmatrix} A' & B' \\ M & M \end{pmatrix}.$$

4. By Proposition 3.3, it is possible to remove $M$ from the bottom right by duplicating the bottom left $M$ using Phase gates on every qubit.
$$\begin{pmatrix} A & B \\ M & 0 \end{pmatrix}$$

5. $M$ is full rank so by Proposition 3.4, some $CNOT$ gates can perform a Gaussian elimination and map $M$ to $I$.
$$\begin{pmatrix} A & B \\ I & 0 \end{pmatrix}$$

6. Proposition 3.2 implies that applying $H$ on every qubit swaps the left and right columns.

   At the same time, the commutation properties imply that $\Omega(Q_i, Q_{n+j}) = 1$ if and only if $i = j$, thus $\Omega(\overline{AB}, \overline{I0}) = I$. It implies that $A * 0 + I * B = I$ so $B = I$. The tableau at this point can be written
$$\begin{pmatrix} I & A \\ 0 & I \end{pmatrix}.$$

7. By the same commutativity argument used in Item 3, it exists $N$ and $\Lambda'$ such that $A + \Lambda' = NN^T$. It implies that a round of phases followed by a round of $CNOT$ gates can perform the $+\Lambda'$ and the reverse Gaussian elimination in order to get the tableau

$$\begin{pmatrix} N & N \\ 0 & C \end{pmatrix}.$$

More precisely, by Proposition 3.4, $C$ is equal to $I * \left(N^T\right)^{-1}$

$$\begin{pmatrix} N & N \\ 0 & \left(N^T\right)^{-1} \end{pmatrix}.$$

8. By Proposition 3.3, applying Phase gates on each qubit remove the top right $N$ and let the bottom right $C$ unchanged.

$$\begin{pmatrix} N & 0 \\ 0 & \left(N^T\right)^{-1} \end{pmatrix}$$

9. Finally, doing a last Gaussian elimination on the top left corner, with a round of $CNOT$ gates, made the matrix becomes the identity. With the Proposition 3.4, where $C$ here equals $N$, the table becomes

$$\begin{pmatrix} N & 0 \\ 0 & \left(N^T\right)^{-1} \end{pmatrix} \begin{pmatrix} N^{-1} & 0 \\ 0 & N^T \end{pmatrix} = \begin{pmatrix} NN^{-1} & 0 \\ 0 & \left(N^T\right)^{-1} N^T \end{pmatrix} = I_{2n}$$

□