Master thesis

Optimisation of approximate stabilizer state decomposition of arbitrary quantum states

Quentin Pitteloud

Supervisors

Prof. Vincenzo Savona
Dr. Sofia Vallecorsa

August 24, 2022

Laboratory of Theoretical Physics of Nanosystems
Institute of Physics
École Polytechnique Fédérale de Lausanne

Quantum Technology Initiative
CERN Openlab

Abstract

We present a ground state optimization algorithm based on the parallel tempering method and CH-form stabiliser state representation. The algorithm has been implemented and benchmarked in both Python and Julia and numerical optimizations such as multi-threading have been used. The quantum natural gradient method has been adapted to the framework, however issues in its implementation prevented its use. A study of the efficiency of using stabilizer decompositions to approximate arbitrary quantum states has been performed. These approximate decompositions show promise for improved performance on the representation of arbitrary states. Finally, an empirical relation for the approximate stabilizer rank required to achieve relative error ϵ on the energy of a state has been devised.

Contents

1 Introduction

In the NISQ (Noisy Intermediate-Scale Quantum computers) era, quantum computers (QC) have to prove their reliability and superiority over classical computers. Indeed, their noisiness constraint them to relatively shallow circuits and their intermediate-scale imply that only a small number of qubits are available. This makes most NISQ computers classically simulable on commercially available desktop computers using standard simulation techniques. Yet, progress on hardware is fast and already multiple quantum computers have reached scales that are out of reach for most widespread quantum computing frameworks [1] (for reference, Qiskit's default "state vector" simulator manages up to 30 qubits on a desktop computer [2]). Hence, developing state-of-the-art classical simulators to certify the accuracy of larger scale QC is paramount for quantum computer certification. Multiple approaches can be taken when creating a classical simulator to increase their efficiency. For instance, Julia's Yao package takes the computer science approach and makes extensive use of GPU/TPU accelerators and other optimizations to achieve incredible performance [3]. However, we will take a different approach by following in the steps of Brayvi *et al.* [4]. The idea is to improve performance over a standard approach by leveraging the efficiency of simulating stabilizer states. Once the simulator is completed, it will be used as the basis for a ground state (GS) optimisation algorithm based on stochastic algorithms.

In this work, we will first give the theory around the simulator and on the GS optimiser. The optimiser will be applied to the Transverse field Ising model (TIM) since it is analytically soluble and the ground state energy can be computed analytically. Then, the performance of the algorithm will be discussed. Finally, limited stabilizer decompositions will be studied to gauge the performance gain of using them as approximations of arbitrary states.

2 Theory & implementation

2.1 Transverse field Ising Model

Before presenting the theory behind the ground state optimiser itself, let us briefly talk about the model on which it will work. The model in question is the Transverse field Ising Model (TIM). This model is a quantum version of the Ising model and will be used to test and benchmark our algorithm since it is analytically soluble. The Hamiltonian of the (cyclic) one-dimensional TIM is very similar to its classical equivalent and is given by

$$\mathcal{H} = -J\sum_{j=1}^{N} Z_j Z_{j+1} - h\sum_{j=1}^{N} X_j, \quad Z_{N+1} = Z_1, \tag{1}$$

where X_j, Z_j are Pauli gates defined as $P_j := \left(\prod_{i=1}^{j-1} \mathbb{1}\right) P \left(\prod_{i=j+1}^{N} \mathbb{1}\right)$, J and h are the bond coefficient and external field intensity respectively, and N is the number of qubits. Just as the classical model, the TIM is a useful case study for phase transitions since it is a simple system that display a phase transition at $J/h = 1$ which can be analytically studied. However, we will not lay our interest on phase transitions, but rather on ground state optimization. The ground state energy can be analytically computed by mapping the model to a quadratic fermionic model [5, 6]:

$$\mathcal{H} = \frac{hN}{2} - h\sum_{i=1}^{N} c_i^\dagger c_i - \frac{J}{4}\sum_{i=1}^{N}(c_i^\dagger - c_i)(c_{i+1}^\dagger + c_{i+1}) + \frac{J}{4}(c_N^\dagger - c_N)(c_1^\dagger + c_1)\exp(i\pi L), \tag{2}$$

where c_i are fermionic operators whose form are not relevant to our purposes and $L = \sum_{i=1}^{N} c_i^\dagger c_i$. The last term in equation (2) is generated by the cyclicity of the chain. For large enough systems, this term can be neglected. Diagonalization of this Hamiltonian gives the dispersion relation [7]

$$\epsilon_k(J,h) = \sqrt{\left(\frac{h}{J} - \cos(k)\right)^2 + \sin^2(k)}, \tag{3}$$

where $k = 2\pi m/N$ and

$$m = \begin{cases} -N/2, \dots, 0, \dots, N/2 - 1 & \text{if N even} \\ -(N-1)/2, \dots, 0, \dots, (N-1)/2 & \text{if N odd.} \end{cases} \tag{4}$$

This relation can then be used to compute the ground state energy as

$$E_{GS} = \sum_k \epsilon_k. \tag{5}$$

2.2 CH-form representation

The classical simulator of quantum circuits uses a representation of states called the CH-form. This representation's theoretical foundation will first be explained in the following subsections, then theory regarding its evolution under quantum circuits and scalar products will be given in detail.

2.2.1 Clifford group

It is known that the set of gates $\{H, S, CX, T\}$, where

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \; S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \; CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \; T = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix}, \tag{6}$$

form a universal set of quantum gates [8]. That is, any quantum circuit can be approximated to any accuracy using only those four gates. By removing the T gate from the set, the resulting set $C(\mathcal{G}_N) = \{H, S, CX\}$ normalises the Pauli group $\mathcal{G}_N$, which means $\forall G \in C(\mathcal{G}_N)$ we have $G\mathcal{G}_N G^{-1} = \mathcal{G}_N$. $C(\mathcal{G}_N)$ is called the Clifford group. An important property of this group is that circuits constructed only from these gates are efficiently simulable by a classical computer [9]. Indeed, pure quantum states can be represented by their stabilizers. That is, an operator O is said to *stabilize* a pure quantum state $|\psi\rangle$ if this state is an eigenvector of O with eigenvalue 1 [10]. Following Gottesman-Knill's theorem [9], states obtained from Clifford circuits are stabilized by a polynomial number of Pauli operators, hence the efficient simulability.

In a similar way, we call a pure state $|\phi\rangle$ a stabilizer state if there exists a subgroup $S \subseteq \mathcal{G}_N$ such that $\forall P \in S \; P|\psi\rangle = |\psi\rangle$ [11, 12]. States generated by Clifford circuits are stabilizer states. These states form an over-complete basis of the Hilbert space and thus any quantum state can be represented as a linear combination of stabilizer states.

2.2.2 Definition

The CH-form is a representation of stabilizer states based on the Heisenberg representation of quantum mechanics and normal-form Clifford circuits. Normal form circuits are circuits composed of 3 blocks: first a block of Hadamard gates, followed by a block of X and CX gates and finally a block of phase and controlled-phase gates. A visual representation of such circuit can be viewed in figure 1.

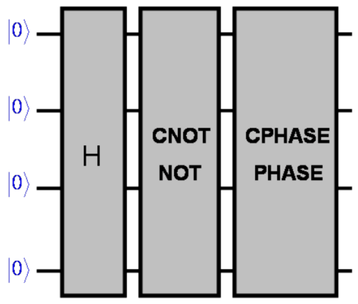


Figure 1: Normal form circuit [13]

It can be shown that any Clifford circuit is equivalent to a normal form circuit [13], where equivalence between circuits is defined here as having the same output probability for a chosen input basis state (usually $\left|0^{\otimes N}\right\rangle$). In other words, two circuits $\mathcal{C}$ and $\mathcal{C}'$ are equivalent if and only if $\mathcal{C}\left|0\right\rangle = \mathcal{C}'\left|0\right\rangle$. The CH-form representation makes use of this property of Clifford circuit to represent stabilizer states using normal form circuits,

$$\left|\phi\right\rangle = \omega U_C U_H \left|s\right\rangle, \tag{7}$$

where U_C is a operator composed of CX, S, and CZ gates, U_H is a block of Hadamard gates, $\left|s\right\rangle$ is an element of the computational basis, and ω is a complex phase. Since $H^2 = \mathbb{1}$, there is at maximum one H gate applied per qubit. Hence, the latter operator can be uniquely represented using a binary vector of N elements that we will denote by v.

As for the U_C operator, the most efficient representation for it is using its stabilizer tableau, similar to Aaronson and Gottesman's state representation in their CHP algorithm [10]. In the CHP algorithm, states are represented by their stabilizers to efficiently simulate them. In our case, we can represent the operators in a similar fashion: the operator is represented using its effect on Pauli operators. This is presented mathematically in equations (8),

$$U_C^{-1} X_p U_C = i^{\gamma_p} \prod_{j=1}^{N} X_j^{F_{p,j}} Z_j^{M_{p,j}}, \quad U_C^{-1} Z_p U_C = \prod_{j=1}^{N} Z_j^{G_{p,j}} \quad \forall p \in [N], \tag{8}$$

where F, M and G are $N \times N$ binary matrices and $\gamma \in \mathbb{Z}_4^N$. In total, the stabilizer state $\left|\phi\right\rangle$ can be uniquely represented from the set of data $\{F, M, G, \gamma, v, s, \omega\}$. To obtain a stabilizer state, this data can be updated iteratively in polynomial time for each gate of the appropriate Clifford circuit. In the following sections, we will shown the process of efficiently updating a CH-form's data when any gate $G \in \{H, S, CX, CZ\}$ is applied to the stabilizer state, as described in reference [4].

2.2.3 Evolution under a quantum circuit

In the first place, we will consider the update rules for the operator U_C, i.e. for gates $\{S, CX, CZ\}$. These can be computed using the Clifford algebra [9]. For brevity, we will only detail here the computation of the left-hand application of S_q to U_C. The computation of update rules for the left-hand application of CX and CZ is straightforward using the same method, and so are the computation of right-hand update rules.

$$\begin{aligned}(S_qU_C)^{-1}X_qS_qU_C &= (U_C)^{-1}S_q^{-1}X_qS_qU_C = (U_C)^{-1}(-iX_qZ_q)U_C = -i(U_C)^{-1}X_qU_CU_C^{-1}Z_qU_C \\ &= i^{\gamma_p-1}\prod_{j=1}^{N}X_j^{F_{p,j}}Z_j^{M_{p,j}}\prod_{j=1}^{N}Z_j^{G_{p,j}} = i^{\gamma_p-1}\prod_{j=1}^{N}X_j^{F_{p,j}}Z_j^{M_{p,j}+G_{p,j}}, \quad (9)\\ (S_qU_C)^{-1}Z_qS_qU_C &= (U_C)^{-1}S_q^{-1}Z_qS_qU_C = (U_C)^{-1}Z_qU_C, \quad (10)\end{aligned}$$

which give the rules

$$M_{q,j} \leftarrow M_{q,j} \oplus G_{q,j},\ \forall j \in [N] \tag{11}$$
$$\gamma_q \leftarrow \gamma_q - 1. \tag{12}$$

We denote $\mathcal{R}(C)$ the rule corresponding to the right-hand multiplication $U_C \leftarrow U_CC$ and $\mathcal{L}(C)$ for the left-hand multiplication $U_C \leftarrow CU_C$. The update rules for gates S, CX, and CZ are given by the following equations, where $j = 1, ..., N$, $\oplus$ is addition modulo 2, and operations on γ are performed modulo 4:

$$\mathcal{L}(S_q) : \begin{cases} M_{q,j} \leftarrow M_{q,j} \oplus G_{q,j} \\ \gamma_q \leftarrow \gamma_q - 1 \end{cases} \qquad \mathcal{R}(S_q) : \begin{cases} M_{j,q} \leftarrow M_{j,q} \oplus F_{j,q} \\ \gamma_j \leftarrow \gamma_j - F_{j,q} \end{cases} \tag{13}$$

$$\mathcal{L}(CZ_{c,t}) : \begin{cases} M_{c,j} \leftarrow M_{c,j} \oplus G_{t,j} \\ M_{t,j} \leftarrow M_{t,j} \oplus G_{c,j} \end{cases} \qquad \mathcal{R}(CZ_{c,t}) : \begin{cases} M_{j,c} \leftarrow M_{j,c} \oplus F_{j,t} \\ M_{j,t} \leftarrow M_{j,t} \oplus F_{j,c} \\ \gamma_j \leftarrow \gamma_j + 2F_{j,c}F_{j,t} \end{cases} \tag{14}$$

$$\mathcal{L}(CX_{c,t}) : \begin{cases} \gamma_c \leftarrow \gamma_c + \gamma_t + 2(M \cdot F^T)_{c,t} \\ F_{c,j} \leftarrow F_{c,j} \oplus F_{t,j} \\ M_{c,j} \leftarrow M_{c,j} \oplus M_{t,j} \\ G_{t,j} \leftarrow G_{t,j} \oplus G_{c,j} \end{cases} \qquad \mathcal{R}(CX_{c,t}) : \begin{cases} F_{j,t} \leftarrow F_{j,t} \oplus F_{j,c} \\ M_{j,c} \leftarrow M_{j,c} \oplus M_{j,t} \\ G_{j,c} \leftarrow G_{j,c} \oplus G_{j,t} \end{cases} \tag{15}$$

One can immediately see that the computational cost to update a CH-form from these gates is $\mathcal{O}(N)$.

As for the update rules for Hadamard gates, the process is more complicated and cannot be stated using simple equations in the same way as S, CZ, and CX. This update process had been written in a flowchart format in figure 2 to improve readability. The aim here is to compute operators U'_C and U'_H, $s' \in \{0,1\}^N$ and $\omega' \in \mathbb{C}$ such that

$$H_q |\phi\rangle = \omega' U'_C U'_H |s'\rangle . \tag{16}$$

The process of updating the CH-form of $|\phi\rangle$ for a gate H_q starts by noticing that $H_q = 2^{-1/2}(X_q + Z_q)$. Then using equation (8), one gets

$$H_q |\phi\rangle = 2^{-1/2}\omega U_C U_H[(-1)^{\alpha} |t\rangle + i^{\gamma_q}(-1)^{\beta} |u\rangle], \tag{17}$$

with $t, u \in \{0,1\}^N$ N-bit strings defined as

$$t := (s_j \oplus G_{q,j}v_j)_{j=1,...,N}, \quad u := (s_j \oplus F_{q,j}(1 \oplus v_j) \oplus M_{q,j}v_j)_{j=1,...,N}, \tag{18}$$

and

$$\alpha = \sum_{j=1}^{N} G_{q,j} s_j (1 \oplus v_j), \quad \beta \sum_{j=1}^{N} M_{q,j} s_j (1 \oplus v_j) + F_{q,j} v_j (M_{q,j} + s_j). \tag{19}$$

Now we have to consider two cases: $t = u$ and $t \neq u$. The first case gives the simple solution

$$H_q \left|\phi\right\rangle = 2^{-1/2}[(-1)^{\alpha} + i^{\gamma_p}(-1)^{\beta}]\omega U_C U_H \left|t\right\rangle, \tag{20}$$

which corresponds to $s' = t$ and $\omega' = 2^{-1/2}((-1)^{\alpha} + i^{\gamma_p}(-1)^{\beta})\omega$. The second case is less trivial. Our end goal is now to find the operators U_C' and U_H', s' and ω' such that they verify

$$U_H(\left|t\right\rangle + i^{\delta}\left|u\right\rangle) = \omega' U_C' U_H' \left|s'\right\rangle, \tag{21}$$

where δ is an integer modulo 4. To proceed we need the intermediary states $y, z \in \{0,1\}^N$ such that y and z differ by only one bit at index k and the operator V_C that verify

$$U_H(\left|t\right\rangle + i^{\delta}\left|u\right\rangle) = V_C U_H(\left|y\right\rangle + i^{\delta}\left|z\right\rangle), \tag{22}$$

where we identify $\delta \in \mathbb{Z}_4$ as

$$\delta := \gamma_k + 2(\alpha + \beta). \tag{23}$$

With y, z, we have the desired form from equation (21) for all but one bit. For bit k, we have

$$H^{v_k}(\left|y\right\rangle + i^{\delta}\left|z\right\rangle) = \omega'' S^a H^b \left|c\right\rangle, \tag{24}$$

for appropriate values of $a, b, c = 0, 1$. With this equation, we have found our desired state.

To compute the states y, z and operator V_C, we must first construct the following sets:

$$\mathcal{V}_0 = \{l \in [N] : t_l \neq u_l,\ v_l = 0\}, \quad \mathcal{V}_1 = \{l \in [N] : t_l \neq u_l,\ v_l = 1\}. \tag{25}$$

Depending on these two sets, the construction of V_C is different. Suppose first that $\mathcal{V}_0 \neq \emptyset$, then set k as the first element of $\mathcal{V}_0$. V_C is then defined as

$$V_C = \prod_{i \in \mathcal{V}_0 \backslash k} CX_{k,i} \prod_{i \in \mathcal{V}_1} CZ_{k,i}, \tag{26}$$

where if $\mathcal{V}_0 = \{k\}$, CX gates are skipped and likewise for CZ gates if $\mathcal{V}_1 = \emptyset$ In the other case, where $\mathcal{V}_0 = \emptyset$, V_C is defined as

$$V_C = \prod_{i \in \mathcal{V}_1 \backslash k} CX_{i,k}, \tag{27}$$

and $V_C = \mathbb{1}$ if $\mathcal{V}_1 = \{k\}$. In all cases, the bit strings y, z are defined by construction as

$$y = u \oplus e_k,\ z = u \text{ if } t_k = 1 \tag{28}$$

$$y = t,\ z = t \oplus e_k \text{ if } t_k = 0. \tag{29}$$

Now we have to compute the bits a, b, c, which depend on the values of $\delta,\ y_k,\ v_k$. Unfortunately, we have found no continous function to relate these variables together. Table 1 gives the output values a, b, c, ω_{alt} corresponding to each possible set of input values $\delta,\ y_k,\ v_k$.

Finally, we obtain the new state $\left|\phi'\right\rangle = H_q \left|\phi\right\rangle = \omega' U_C' U_H' \left|s'\right\rangle$, where

$$\omega' = (-1)^{\alpha}\omega_{alt}\omega \tag{30}$$

$$U_C' = U_C V_C S^a \tag{31}$$

$$U_H' = U_H H_k^{b \oplus v_k} \tag{32}$$

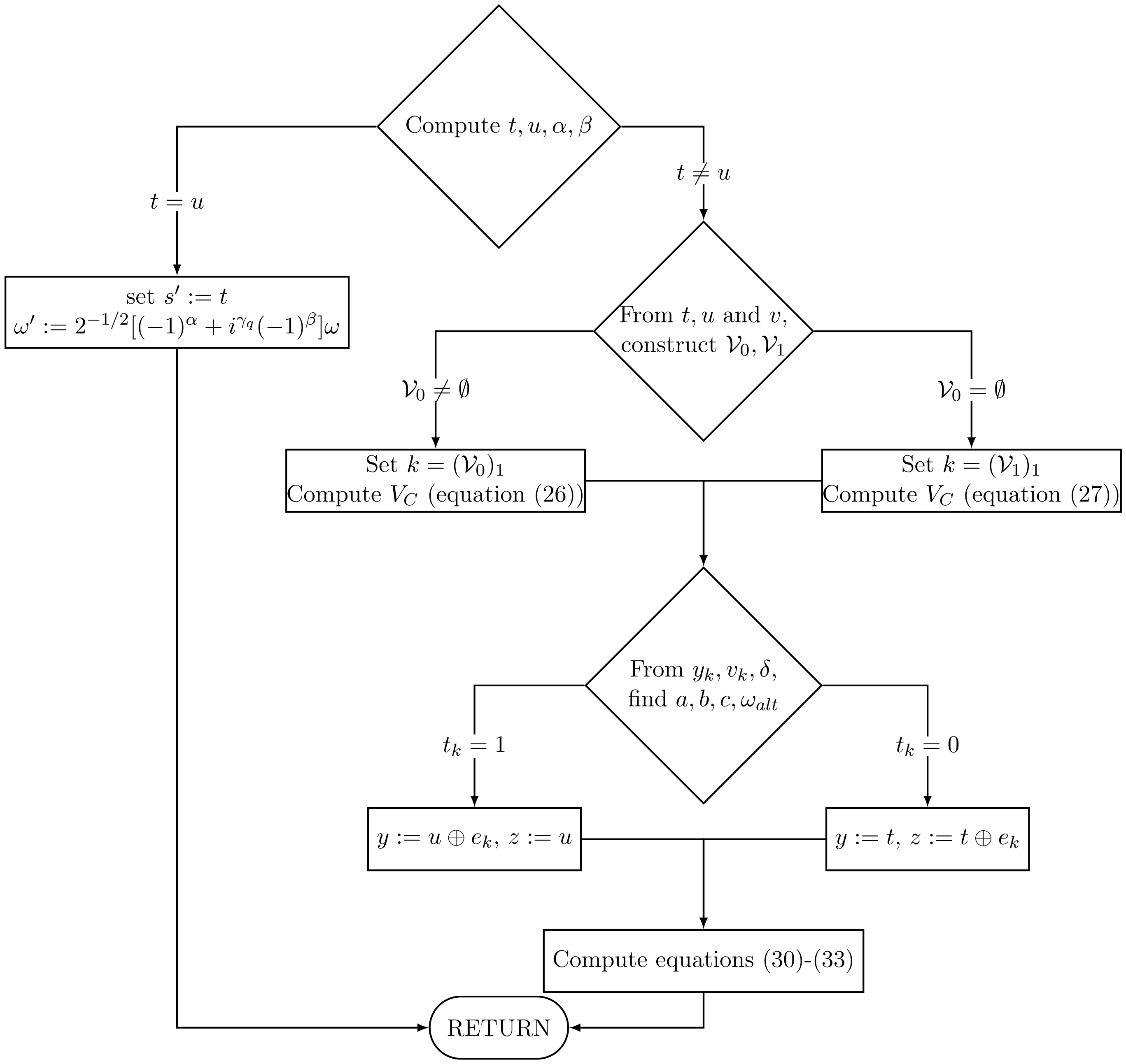
$$s_i' = y_i\ \forall i \in [N] \backslash k, \quad s_k' = c. \tag{33}$$

Unlike evolution for gates $\in \{S, CZ, CX\}$, Hadamard gates takes $\mathcal{O}(N^2)$ elementary operations to process.

To make visualisation of the update process for Hadamard easier, it can be seen in flowchart 2.

v_q	1				0			
δ	0	1	2	3	0	1	2	3
a	0	1	0	1	0	1	0	1
b	0	1	0	1	1	1	1	1
c	0	$y_k \oplus 1$	1	y_k	0	y_k	1	$y_k \oplus 1$
ω_{alt}	1	$(1+i)/\sqrt{2}$	$(-1)^{y_k}$	$(1-i)/\sqrt{2}$	i^δ			

Table 1: Relations needed to solve equation (24). The first two rows allows the selection of the input and the output is given in the lower four rows.



Compute t, u, α, β
$t = u$
$t \neq u$
set $s' := t$
$\omega' := 2^{-1/2}[(-1)^\alpha + i^{\gamma_q}(-1)^\beta]\omega$
From t, u and v, construct $\mathcal{V}_0, \mathcal{V}_1$
$\mathcal{V}_0 \neq \emptyset$
$\mathcal{V}_0 = \emptyset$
Set $k = (\mathcal{V}_0)_1$ Compute V_C (equation (26))
Set $k = (\mathcal{V}_1)_1$ Compute V_C (equation (27))
From y_k, v_k, δ, find a, b, c, ω_{alt}
$t_k = 1$
$t_k = 0$
$y := u \oplus e_k$, $z := u$
$y := t$, $z := t \oplus e_k$
Compute equations (30)-(33)
RETURN

Figure 2: Evolution of the CH-form for a Hadamard gate H_q

2.2.4 Scalar product

In this section, we will expand upon the work of reference [4] by introducing a routine that computes the scalar product between two CH-form stabilizer states. In the original paper by Bravyi *et al.*, a routine to compute scalar products between CH-form stabilizer and computational basis states was proposed. It will serve as a basis for our own scalar product method, hence it will be explained first.

The aim is to compute the scalar product $\langle x|\phi\rangle = \langle 0^N| X(x)\omega U_C U_H |s\rangle$ where $X(u) = \prod_{j=1}^N X_j^{u_j}$. For that, the idea is to construct an operator $Q = i^\mu Z(t)X(u) := i^\mu \prod_{j=1}^N Z_j^{t_j} X_j^{u_j}$ such that $\langle x|\phi\rangle = \omega \langle 0^N| QU_H |s\rangle$. In other words, we want to find the n-bit vectors u and t and the integer (modulo 4) μ as a function of x and the data contained in the CH-form of $|\phi\rangle$. The algebra to get these functions is relatively simple, and requires the definition of the stabilizer tableau of U_C (equation (8)). The complete computations are given in appendix A.1.

After some algebra, we find

$$\omega \langle 0^N| X(x)U_C U_H |s\rangle = \omega \langle 0^N| i^{\gamma\cdot x+2b} \prod_{j=1}^N Z_j^{t_j} X_j^{u_j} U_H |s\rangle . \tag{34}$$

where $\gamma \cdot x := \sum_{p=1}^N \gamma_p x_p$, $t_j := \sum_{p=1}^N x_p M_{p,j}$, $u_j := \sum_{p=1}^N x_p F_{p,j}$ and

$$b := \sum_{j=1}^N \sum_{l=1}^N x_l F_{l,j} \left(\sum_{k=l}^N x_k M_{k,j} \right) . \tag{35}$$

From these definitions we find the $\mathcal{O}(N^2)$ complexity advertised by Bravyi *et al.*.

Now that we have the operator Q, we can compute the scalar product. Indeed, we have

$$\begin{aligned}
\langle x|\phi\rangle &= \omega \langle 0^N| X(x)U_C U_H |s\rangle && (36)\\
&= \omega \langle 0^N| QU_H |s\rangle && (37)\\
&= \omega i^\mu \langle 0^N| X(u)U_H |s\rangle && (38)\\
&= \omega i^\mu 2^{-|v|/2} \prod_{j:\ v_j=1} (-1)^{s_j u_j} \prod_{j:\ v_j=0} \langle u_j|s_j\rangle , && (39)
\end{aligned}$$

where $|v| \equiv ||v||_1$.

Now we can consider scalar products between two CH-form stabilizer states. Note however that these scalar products can unfortunately not be reduced to a simple equation. This is a consequence of the very definition of CH-form representation: states represented by operators acting on a computational basis state (recall $|\phi\rangle := \omega U_C U_H |s\rangle$). The presence of these operators on both states in the product prevent the same method of constructing an operator Q.

Nonetheless, a conceptually simple algorithm can be devised to compute the scalar product $\langle \phi_2|\phi_1\rangle = \langle s_2| (U_H^2)^\dagger (U_C^2)^\dagger \omega_2^* \omega_1 U_C^1 U_H^1 |s_1\rangle = \langle s_2| U_H^2 (U_C^2)^{-1} \omega_2^* \omega_1 U_C^1 U_H^1 |s_1\rangle$.

- First, compute $U_C' = (U_C^2)^\dagger U_C^1 = (U_C^2)^{-1} U_C^1$.
- Second, apply U_H^2 to $U_C' U_H^1 |s_1\rangle$ to get $U_C'' U_H' |s'\rangle$, *i.e.* for each $j \in [N]$ such that $v_j^2 = 1$ apply H_j to $U_C' U_H^1 |s_1\rangle$.
- Finally, compute $\langle \phi_2|\phi_1\rangle = \omega_2^* \omega_1 \langle s_2| U_C'' U_H' |s'\rangle$

We can expect a complexity of around $\mathcal{O}(N^3)$ because of the second step in which we apply $\mathcal{O}(n)$ Hadamard gates whose evolution cost $\mathcal{O}(N^2)$ each.

To proceed at the first step of the algorithm, we need to compute the stabilizer tableau of $U_C := U_C^2 U_C^1$, in a circuit-agnostic way. The strategy here is the same as for computing update rules of individual gates. We want to find the stabilizer tableau of U_C', which is defined by

$$(U_C^2 U_C^1)^{-1} Z_p U_C^2 U_C^1 = \prod_{j=1}^{N} Z_j^{G'_{p,j}}, \quad (U_C^2 U_C^1)^{-1} X_p U_C^2 U_C^1 = i^{\gamma'_p} \prod_{j=1}^{N} X_j^{F'_{p,j}} Z_j^{M'_{p,j}}. \tag{40}$$

After some algebra (explicitly given in appendix A.2), we find

$$\begin{aligned} &F' = F^2 \cdot F^1, \quad M' = F^2 \cdot M^1 + M^2 \cdot G^1, \\ &\gamma'_q = \gamma_q^2 + (F^2 \cdot \gamma^1)_q + 2 \sum_{p=1}^{N} \sum_{j=2}^{N} a_{q,p}^j \left(\sum_{k=1}^{j-1} b_{q,p}^k \right) \ \forall q \in [N], \end{aligned} \tag{41}$$

where we defined $a_{q,p}^j := F_{q,j}^2 F_{j,p}^1$ and $b_{q,p}^j := F_{q,j}^2 M_{j,p}^1 + M_{q,j}^2 G_{j,p}^1$. As expected, the vectors s and v and the complex number ω are left unchanged in these computations, just as they are when a gate in {S,CX,CZ} is applied to the stabilizer state.

We now only have to compute the conjugate transpose of a CH-form representation and the explanation of our scalar product routine will be complete. We will proceed by considering the simplest case possible and identifying the relations from there. This simplest case is computing the norm of a stabilizer state:

$$1 = \langle \phi | \phi \rangle = |\omega|^2 \langle s | U_H (U_C)^{-1} U_C U_H | s \rangle . \tag{42}$$

We have $(U_C)^{-1} U_C = \mathbb{1} =: U_C'$ which corresponds to $F' = G' = \mathbb{1}$, $M' = 0, \gamma' = 0$. To simplify notations, we will write $A^* := A_{U_C^{-1}}$ and $A := A_{U_C}$ for $A \in \{F, M, G, \gamma\}$. Using the previously computed relations, we get

$$\mathbb{1} = F^* F \tag{43}$$
$$\mathbb{1} = G^* G \tag{44}$$
$$0 = F^* M + M^* G. \tag{45}$$
$$0 = \gamma_q^* + (F^* \gamma)_q + 2 \sum_{p=1}^{n} \sum_{j=2}^{n} F_{q,j}^* F_{j,p} \left[\sum_{k=1}^{j-1} F_{q,k}^* M_{k,p} + M_{q,k}^* G_{k,p} \right] \tag{46}$$

from which we can easily identify

$$F^* = (F)^{-1} \tag{47}$$
$$G^* = (G)^{-1} \tag{48}$$
$$M^* = -F^* M (G)^{-1} = -(F)^{-1} M (G)^{-1} \tag{49}$$
$$\gamma_q^* = - \left(((F)^{-1} \gamma)_q + 2 \sum_{p=1}^{n} \sum_{j=2}^{n} (F)_{q,j}^{-1} F_{j,p} \left[\sum_{k=1}^{j-1} (F)_{q,k}^{-1} M_{k,p} + (M)_{q,k}^{-1} G_{k,p} \right] \right). \tag{50}$$

For practical implementations of these algorithms, it is important to recall that all matrices above are binary matrices, which can lead to issues when using most common implementations of matrix inversion algorithms. When computed naïvely, the complexity of the scalar product routine is $\mathcal{O}(N^3)$. Indeed, the complex conjugation of one state includes the inversion of $N \times N$

binary matrices F and G which costs at most $\mathcal{O}(N^3)$. The computation of M^* takes at most $\mathcal{O}(N^2)$ (matrix multiplication) and γ^* takes $\mathcal{O}(N(N+1)/2) \approx \mathcal{O}(N^2)$. The multiplication $(U_C^2)^{-1}U_C^1$, is only composed of matrix multiplication and also takes $\mathcal{O}(N^2)$ operations. The computation of $\mathcal{O}(N)$ Hadamard gates (third step) take $\mathcal{O}(N^2)$ each, for a total of $\mathcal{O}(N^3)$. Finally, the computation of $\langle s|\phi''\rangle$ takes $\mathcal{O}(N^2)$. The highest scaling is thus $\mathcal{O}(N^3)$.

2.3 Metropolis-Hastings ground state optimiser

To find the best Clifford approximation of the ground state of an Hamiltonian, we have made use of the Metropolis-Hastings (MH) algorithm, which relies on a Markov chain to minimize a given cost function. In our case, this cost function is the TIM Hamiltonian presented in equation (1). While the M.-H. algorithm has previously been used in this context many times, we expand its range of use cases a little by adapting it to the optimisation of stabilizer state decompositions. However, the algorithm stays the same. For the sake of completeness, we will present the algorithm and discuss some important points for the implementation.

The M-H. algorithm can be seen as doing a random walk in the Hilbert space and imposing conditions on the selection of the steps. Steps are accepted if the energy of the new approximation is lower, but it can also be probabilistically accepted otherwise. Indeed, in the second case, one accepts the step with probability $P = \exp[(E_{current} - E_{new})/T]$, where T is a temperature. The algorithm will converge to the best approximation of the ground state as $t \to \infty$ if the temperature is decreasing sufficiently slowly, similar to an annealing process. At high temperature, the probability to accept a step which increases the energy is relatively high, which allows the process to avoid getting stuck in local minima. At lower temperature, this probability becomes small, which confines the walker to the ground state. For the last affirmation to be true, however, one needs to lower the temperature sufficiently slowly.

The process to generate and accept a step is shown in flowchart 3. S denotes the space of all possible steps, x, y are states in the Hilbert space and η is a small number (step size) which is defined constant.

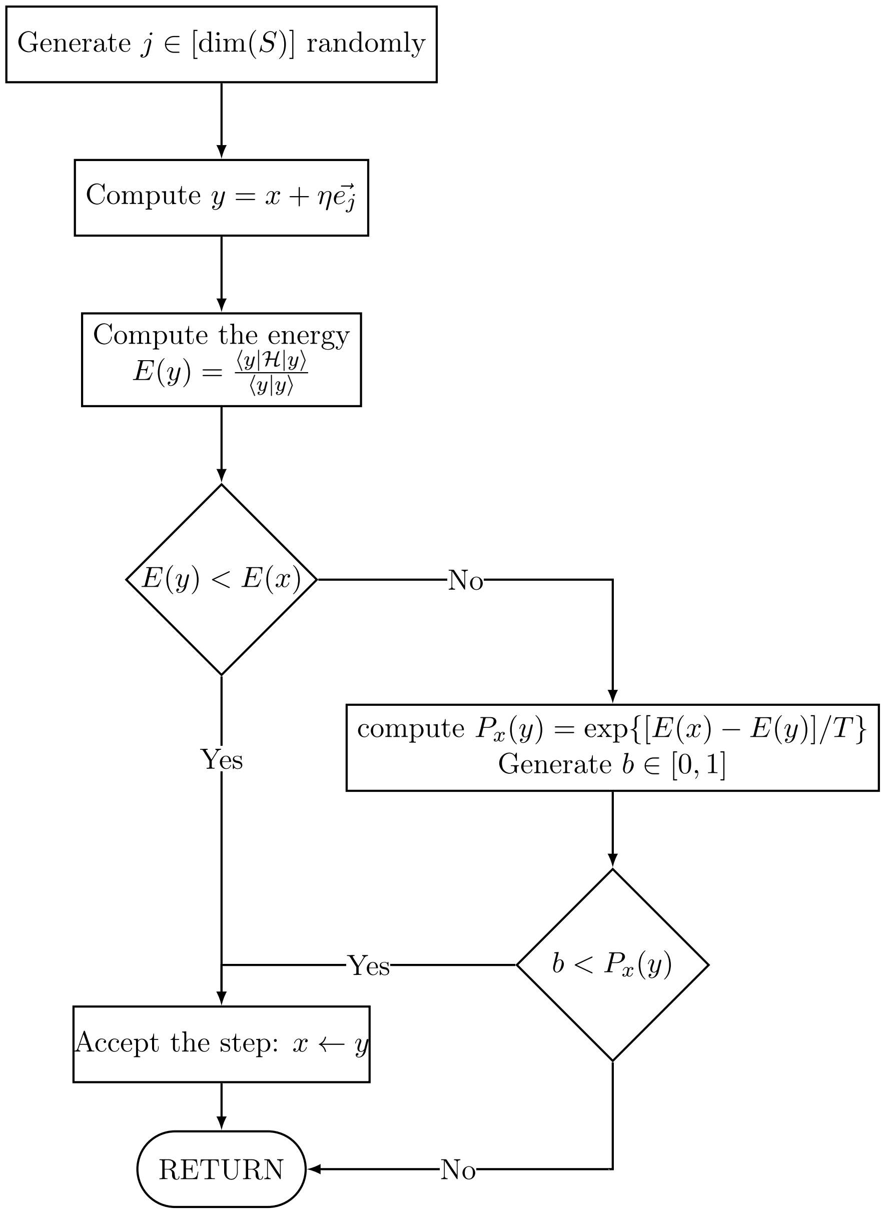

Generate $j \in [\dim(S)]$ randomly
Compute $y = x + \eta \vec{e_j}$
Compute the energy $E(y) = \frac{\langle y|\mathcal{H}|y\rangle}{\langle y|y\rangle}$
$E(y) < E(x)$
No
Yes
compute $P_x(y) = \exp\{[E(x) - E(y)]/T\}$ Generate $b \in [0, 1]$
$b < P_x(y)$
Yes
Accept the step: $x \leftarrow y$
RETURN
No

Figure 3: Metropolis-Hastings step process

In our practical case, the initial state is chosen to be the ground state of the Hamiltonian without external field ($h = 0$). This corresponds to a state $\left|0^{\otimes N}\right\rangle$ which has an energy of $E_{in} = -NJ$. We will refer to this state as the "Hartree-Fock state" (or HF state for short).

Note that the energy can be computed as solving a generalized eigenvalue problem, which is much easier to perform (numerically) in practice than solving the equation shown in the flowchart. Indeed, by defining the matrices $H_{i,j} = \langle\phi_i|\, \mathcal{H}\, |\phi_j\rangle$ and $P_{i,j} = \langle\phi_i|\phi_j\rangle$ (where $|\psi\rangle = \sum_k \alpha_k |\phi_k\rangle$), we have the equivalence,

$$E(y) = \frac{\langle\psi|\, \mathcal{H}\, |\psi\rangle}{\langle\psi|\psi\rangle} \iff H\vec{\alpha} = E\ P\vec{\alpha}, \tag{51}$$

which can be shown by simply rearranging the terms on the left expression.

In order to improve the consistency of the results, a restart condition can be implemented in the algorithm. At random, when a new energy is found lower or equal to the current energy, the temperature is reset to its initial value and the annealing restarts. This method allows the process to escape local minima but, as with every parameter of this algorithm, must be fine tuned as too many restarts will impede on the minimization. In the flowchart, this restart process would be placed between the "accept" box and the "return" box.

The choice of possible steps is very important as it directly impacts the speed of convergence. In the following subsections, we will discuss the different types of steps implemented.

2.3.1 Clifford- and Physical-based sampling

The obvious first choice of steps for a quantum M-H. algorithm would be single qubit gates, for instance Clifford Gates. In practice, this means that for each step a random Clifford gate is selected and applied to a random stabilizer state, then one checks how this change affects the ground state energy and decides to keep the step accordingly. It has the notable advantage of providing the most freedom in the movement of the random walker, since every stabilizer state can theoretically be accessed from any initial state. However, it seems relatively intuitive that this method has a low speed of convergence due to the fact that every step is really small, since it is defined by a single gate in the set {H, S, CX, CZ, X, Z}. Additionally, the low number of possible gates can also negatively affect the speed of convergence. This low speed can and does affect the results in practice, as the dropping temperature could trap the random walker in a local minimum instead of the ground state. Due to this drawback, this method has not been used beyond the testing of the algorithm.

However, the idea of representing steps by (sets of) gates does have its merits. For instance, it is both easy to implement and efficient computationally, as Clifford gates are efficiently implemented in the simulator. It is then natural to try to improve the speed of convergence by changing the gates - or circuits - representing the steps. The idea that was chosen is to build operators using elements of the Hamiltonian. This method will be referred to as the physical sampling. Considering a Transverse-field Ising Model Hamiltonian with periodic boundary conditions given in equation (1), one sees that all elements of the Hamiltonian are elements of the Pauli group $\mathcal{G}_N$. Using the property of the Pauli group that,

$$\exp(-i\theta P) = \cos\theta \mathbb{1} + i \sin\theta P, \quad \forall P \in \mathcal{G}_N, \tag{52}$$

and using reference [14] one can notice that for $\theta = k\pi/4,\ k = 0, 1, 2, 3$, the exponentials of Pauli elements are elements of the Clifford group. Reference [14] also gives a general method to build the quantum circuit computing the exponential of a Pauli that will be presented here for convenience. In cases of relevance to this work, the operator P can be written as

$$P = \prod_k i^{x_k \cdot z_k} X_k^{x_k} Z_k^{z_k}, \tag{53}$$

where $x_k, z_k = 0, 1$. Using the relations,

$$R_Y(\pi/2) Z\, R_Y(-\pi/2) = X, \tag{54}$$

$$R_X(\pi/2) Z\, R_X(-\pi/2) = -Y, \tag{55}$$

we can prove the following identity,

$$\exp(-i\delta P) = U_P(\pi/2) \cdot \prod_{j=1}^{N-1} CX_{j,j+1} \cdot R_{Z,N}(2\delta) \prod_{j=1}^{N-1} CX_{N-j,N-j+1} \cdot U_P(-\pi/2), \tag{56}$$

with

$$U_P(\theta) := \prod_{j=1}^{N} [R_{X,j}(\theta)]^{x_k \cdot z_k} [R_{Y,j}(\theta)]^{x_k \cdot (z_k+1)}, \tag{57}$$

where the addition in the last exponent is done modulo 2. Examples of equation (56) can be visualized as quantum circuits in figures 4 and 5.

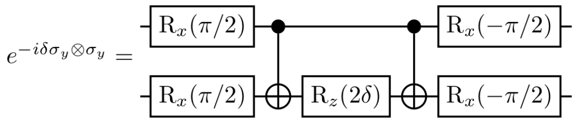


Figure 4: Example 1 [14]

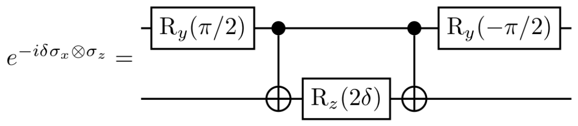


Figure 5: Example 2 [14]

To obtain a more diverse set of possible operators (steps), one should consider applying this method to Pauli operators in the set of commutators of the elements of the Hamiltonian. This set is defined as

$$\{Z_iZ_{i+1}, X_i, [Z_iZ_{i+1}, X_i], [Z_iZ_{i+1}, [Z_iZ_{i+1}, X_i]], [X_i, [Z_iZ_{i+1}, X_i]], \ldots\}. \tag{58}$$

In our testing, only considering the elements written above was sufficient to improve the convergence speed of the random walk compared to Clifford steps.

2.3.2 Parametric sampling

Another way to sample new states is by using parametric circuits and changing its parameters to sample a new state. We will only consider the simplest parametric circuits composed of multiple layers of single qubit rotation gate around x, rotations around z and finally CX between all adjacent sites (qubits). An example of a single-layer parametric circuit for 4 qubits can be seen in figure 6.

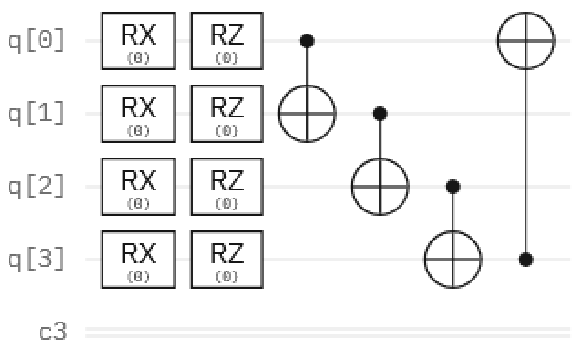


Figure 6: Example of a variational circuit for 4 qubits [15]

To initialize a state in the Hartree-Fock state, one must set all angles to 0. When choosing an initial approximation for the ground state, the first element is initialized to the HF state and subsequent elements are initialized to HF + one (or more) Metropolis step. This gives an initial approximation which can already be close to the true ground state, depending on the ratio J/h.

The sampling process works by uniformly randomly selecting one (or more) rotation gate(s) and changing their angles to a random value. In order to only sample stabilizer states, the angles must be integer multiples of $\pi/4$. Indeed, note that,

$$R_\sigma(\theta) = \exp(-i\theta\sigma/2), \quad \sigma = X, Y, Z, \tag{59}$$

which can then be translated as products of Clifford gates to make to prove the affirmation.

$$R_Z(k\pi/2) = \left(\frac{(1-i)S}{\sqrt{2}}\right)^k, \; R_X(k\pi/2) = \begin{cases} \mathbb{1} & \text{if } k=0 \\ SZHZS & \text{if } k=1 \\ -iX & \text{if } k=2 \\ -SHS & \text{if } k=3. \end{cases} \tag{60}$$

Note that only the angles $\{k\pi/4\}_{k=0,1,2,3}$ are non-redundant (up to a global phase).

It has been observed empirically that the convergence of the Metropolis algorithm is faster when a step is done by changing both rotation gates of a chosen qubit. Similarly, in our testing, using more than a single layer would decrease convergence speed.Using this method, it has been observed that this method of sampling stabilizer states allows the Metropolis algorithm to converge faster than other tested sampling methods.

2.3.3 Parallel tempering

The Metropolis-Hastings algorithm is a rather simple algorithm which fell out of favour due to its limitations and relatively low convergence speed. A way to improve the performance of the algorithm is the parallel tempering (PT) method. It consists of running multiple Markov chains in parallel at different temperatures and swapping pairs of chains at random. As Markov chains have no memory of their pasts, swapping the temperature is an equivalent alternative [16]. This method can offer better consistency and performance than a single chain with restart offers.

The probability to swap must be symmetrical between all chains, hence it must be independent of the chain's variables. For our purposes, a uniform probability distribution was chosen to dictate the swaps.

Note however that this step is merely the initiator of a swap. If the process goes forward, two Markov Chains are then chosen at random. The probability P_{swap} is then computed, where

$$P_{swap} = \min(1, \exp[(E_i - E_j)(\beta_i - \beta_j)]), \tag{61}$$

where we renormalized the temperature to satisfy $\beta = 1/T$ and i, j indicate different Markov chains. Next, with probability P_{swap}, swap the temperatures of the two chains. If any probability check fails, the Markov chains are updated as usual, including the ones not selected for swapping.

This algorithm is best explained by flowchart 7:

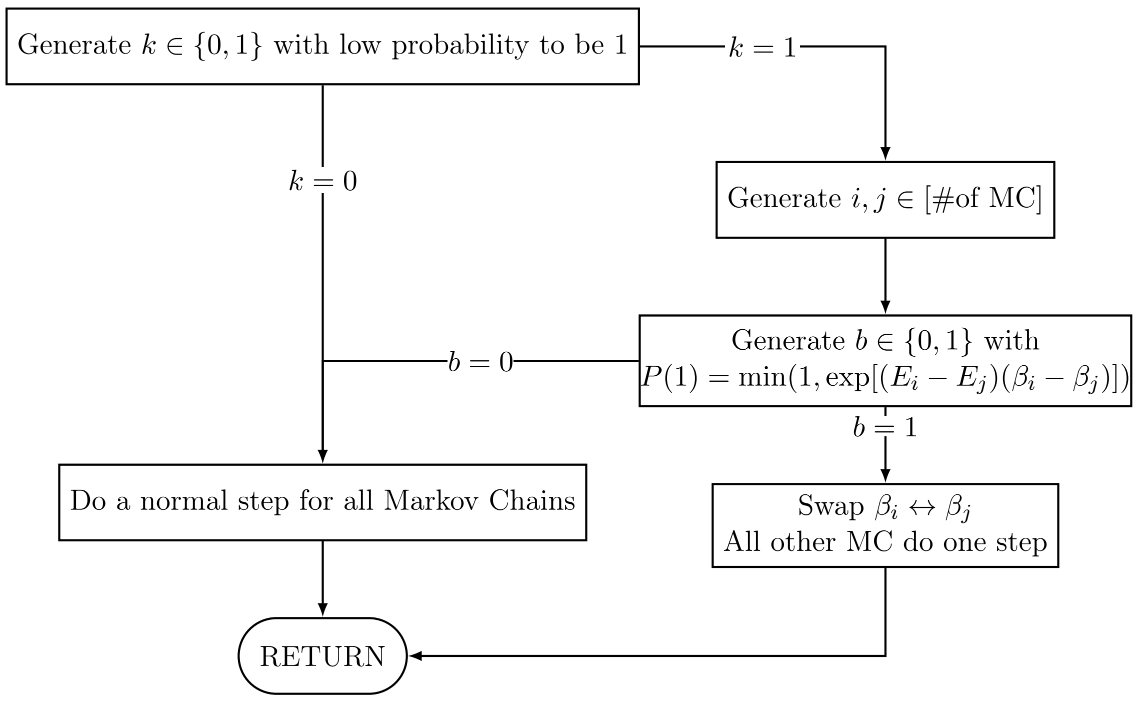

Generate $k \in \{0,1\}$ with low probability to be 1
$k = 1$
$k = 0$
Generate $i, j \in$ [#of MC]
Generate $b \in \{0,1\}$ with $P(1) = \min(1, \exp[(E_i - E_j)(\beta_i - \beta_j)])$
$b = 0$
$b = 1$
Do a normal step for all Markov Chains
Swap $\beta_i \leftrightarrow \beta_j$
All other MC do one step
RETURN

Figure 7: Parallel tempering step process

2.4 Quantum natural gradient

Instead of using the Metropolis algorithm, one could instead use more efficient and consistent methods of obtaining the ground state. Indeed, the Metropolis algorithm is stochastic algorithm, which limits its consistency. More importantly, it requires the costly computation of the energy of the system at every step. To avoid the MH algorithm, one could try optimizing the ground state using the well-known Variational Quantum Eigensolver (VQE) which uses the steepest descent algorithm to optimize the parameters. However, from equations (52) and (59), directly using a VQE with our framework would be highly inefficient: It would require an exponential number of stabilizer states to accurately represent the state given by an Ansatz with $\mathcal{O}(N)$ rotation gates. Directly truncating the decomposition to only have a polynomial number of stabilizer states is equivalent (up to a global phase) to restricting some parameters to values that are integer multiples of π. In fact, it would make the rotation gates act as single qubit Clifford gates just as angles of $m\pi$ would. This limitation could be slightly relaxed by making use of the properties of single qubit rotation gates shown in equation (60).

Nevertheless, we will implement a slightly more refined version of the steepest descent, which is the natural gradient. More precisely, we will implement the quantum natural gradient (QNG) method pioneered by J.Stokes *et al.* [17]. It is as its name imply a quantum generalization of the natural gradient (NG) method, which in turn is related to the steepest descent algorithm.

2.4.1 Natural gradient

The steepest descent tries to find the global minimum of a given cost function by optimizing a set of parameters defining a state, by going down the steepest gradient, and does so under the assumption that the parameter space is flat in the Euclidean sense [18]. The explicit parameter update rule is

$$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta_t), \tag{62}$$

where η is the step size and $\mathcal{L}$ is a cost function we want to optimise. However, the parameter space is not flat in most cases - including the one of interest to us - as different values of the parameters can give the same energy. The natural gradient method takes into account this property of the parameter space by incorporating a Riemann metric tensor F into equation (62):

$$\theta_{t+1} = \theta_t - \eta F^{-1}(\theta_t) \nabla \mathcal{L}(\theta_t). \tag{63}$$

2.4.2 Quantum analogue

In the quantum analogue of the natural gradient method, the evolution of the parameters is still dictated by equation (63) with a slight but important change. The metric in this case is the Fubini-Study metric tensor, which is defined as

$$g_{ij}(\theta) = \mathrm{Re}\left(\left\langle \frac{\partial \psi_\theta}{\partial \theta^i} \middle| \frac{\partial \psi_\theta}{\partial \theta^j} \right\rangle - \left\langle \frac{\partial \psi_\theta}{\partial \theta^i} \middle| \psi_\theta \right\rangle \left\langle \psi_\theta \middle| \frac{\partial \psi_\theta}{\partial \theta^j} \right\rangle \right), \tag{64}$$

where $|\psi_\theta\rangle = U(\theta)|0\rangle$ is the image of the parametric circuit applied on the computational basis state $|0\rangle$ (To simplify the notation, let us agree that $|0\rangle \equiv |0^{\otimes N}\rangle$). This changes the update rule to

$$\theta_{t+1} = \theta_t - \eta g^{+}(\theta_t) \nabla \mathcal{L}(\theta_t), \tag{65}$$

where the $^{+}$ operator designates the Moore-Penrose pseudo-inverse. One can avoid having to compute this pseudo-inverse by rearranging the terms into

$$g(\theta_t)(\theta_{t+1} - \theta_t) = -\eta \nabla \mathcal{L}(\theta_t). \tag{66}$$

In the quantum case, the cost function $\mathcal{L}$ is the Lagrangian, which gives a measure of the energy of the system. It takes the form $\mathcal{L}(\theta) = \frac{1}{2}\langle\psi_\theta|\mathcal{H}|\psi_\theta\rangle$ with $\mathcal{H}$ the system's Hamiltonian. We can then compute the gradient in equation (65), however since we work with superposition of stabilizer states there are some differences to reference [17] in the computation of the gradient. While normally working with a superposition of states would simply make the change $|\psi_\theta\rangle \rightarrow |\psi_\theta\rangle = \sum_j \alpha_j |\phi_\theta\rangle$, we will be treating each stabilizer as an independent state instead. Indeed, this follows from the idea in reference [17] to only consider the block-diagonal elements of the tensor relating to some layer and neglecting the rest of the matrix. This provides an approximation of the full tensor which could be sufficient for the QNG to work, provided the stabilizer states are not linearly dependent. By construction, this last requirement is fulfilled. Hence, the approximation of the tensor that we will compute has the form (for R stabilizer states and L layers)

$$g = \left(\begin{array}{c|c|c|c} g_1 & 0 & \dots & 0 \\ \hline 0 & g_2 & & \vdots \\ \hline \vdots & & \ddots & 0 \\ \hline 0 & \dots & 0 & g_R \end{array}\right), \text{ where } g_k = \mathrm{Re}\left(\begin{array}{c|c|c|c} G_k^{(1)} & 0 & \dots & 0 \\ \hline 0 & G_k^{(2)} & & \vdots \\ \hline \vdots & & \ddots & 0 \\ \hline 0 & \dots & 0 & G_k^{(L)} \end{array}\right), \tag{67}$$

and

$$(G_k^{(l)})_{i,j} = \left\langle \frac{\partial \phi_\theta}{\partial \theta^i} \middle| \frac{\partial \phi_\theta}{\partial \theta^j} \right\rangle - \left\langle \frac{\partial \phi_\theta}{\partial \theta^i} \middle| \phi_\theta \right\rangle \left\langle \phi_\theta \middle| \frac{\partial \phi_\theta}{\partial \theta^j} \right\rangle. \tag{68}$$

Furthermore, the gradient of $\mathcal{L}$ is itself approximated as being composed of blocks corresponding to each layer of the parametric circuits defining each stabilizer. We will denote the block of the gradient corresponding to layer l and stabilizer r by $\nabla\mathcal{L}_r^l$ and simplify the notation by omitting the explicit θ dependency of $U(\theta)$ and ϕ_θ. The index j denotes the parameter, which does not necessarily match the index of the qubit upon which the corresponding parameterized gate acts. Using $\nabla\mathcal{L}_r^{(l)}(\theta) = \frac{1}{2}\left\langle \phi_r^{(l)} \middle| \mathcal{H} \middle| \phi_r^{(l)} \right\rangle$ and $\left|\phi_r^{(l)}\right\rangle = U^l_{[1:l],r}\left|0\right\rangle$ where $U^l_{[1:l],r} = U^l_{1,r}\cdots U^l_{l,r}$, we can compute a more explicit form for the gradient as

$$\begin{aligned}
\partial_j\mathcal{L}_r^l(\theta) &= \frac{1}{2}\partial_j \left\langle 0\right| U^\dagger_{[1:l],r}\mathcal{H}U_{[1:l],r}\left|0\right\rangle && (69)\\
&= \frac{1}{2}\left\langle 0\right| \partial_j U^\dagger_{[1:l],r}\mathcal{H}U_{[1:l],r}\left|0\right\rangle + \frac{1}{2}\left\langle 0\right| U^\dagger_{[1:l],r}\mathcal{H}\partial_j U_{[1:l],r}\left|0\right\rangle && (70)\\
&= \frac{1}{2}\left\langle 0\right| U^\dagger_{[1:l],r} iK_j^\dagger\mathcal{H}U_{[1:l],r}\left|0\right\rangle + \frac{1}{2}\left\langle 0\right| U^\dagger_{[1:l],r}\mathcal{H}(-i)K_j U_{[1:l],r}\left|0\right\rangle && (71)\\
&= \frac{i}{2}\left\langle 0\right| U^\dagger_{[1:l],r}(K_j^\dagger\mathcal{H} - \mathcal{H}K_j)U_{[1:l],r}\left|0\right\rangle && (72)
\end{aligned}$$

where we used

$$\partial_j U_{[1:l],r} = -iK_j U_{[1:l],r} \tag{73}$$

with K_j one of the Hermitian generators or the parametric circuit. Using a simple Ansatz with single-qubit rotation gates, for instance the one given in figure 6, these generators have the form

$$K_j = \frac{P_j}{2}, P \in \mathcal{G}_N. \tag{74}$$

Indeed, recall that the rotation gates are given by $R_{P_j}(\theta) = \exp(-i\theta P_j/2)$.

The quantum geometric tensor (QGT) g can also be computed using the property in equation (73). Indeed, it gives

$$\left\langle \partial_i\phi_r^{(l)} \middle| \partial_j\phi_r^{(l)} \right\rangle = \left\langle \phi_r^{(l)}\right| K_iK_j \left|\phi_r^{(l)}\right\rangle, \; \left\langle \phi_r^{(l)} \middle| \partial_j\phi_r^{(l)} \right\rangle = -i\left\langle \phi_r^{(l)}\right| K_j \left|\phi_r^{(l)}\right\rangle, \tag{75}$$

which becomes, when inserted in equation (68),

$$(G_k^{(l)})_{i,j} = \left\langle \phi_r^{(l)}\right| K_iK_j \left|\phi_r^{(l)}\right\rangle - \left\langle \phi_r^{(l)}\right| K_j \left|\phi_r^{(l)}\right\rangle \left\langle \phi_r^{(l)}\right| K_j \left|\phi_r^{(l)}\right\rangle. \tag{76}$$

Using these explicit forms for $\nabla\mathcal{L}$ and the QGT, it is straightforward to compute them using the quantum simulator framework detailed in section 2.2. The system of equations (66) can then be solved to compute the new vector of parameters θ_{t+1} and the process re-done until the ground state is found. Similarly to a steepest descent, the process could be further optimised using a decreasing step size for each iteration to avoid circling around the minimum, which increase convergence speed.

3 Results and Discussion

3.1 Quantum natural gradient

At a glance, using the QNG would be a great improvement over the Metropolis-Hastings algorithm (or even the parallel tempering). It avoids the relatively costly process of computing the energy at every step and improves the consistency of the process as it is not stochastic. However, the main attribute of this work, which is the stabilizer state decomposition, brings its own limitations that impact the applicability of the QNG. One limitation is the use of stabilizer states, as it limits the parameters to only integer multiples of $\pi/2$. The steps done by the QNG are supposed to be small, but with this limitation only arguably large integer steps in the parameters are possible. However, this could not be determined to be a problem in practice, as other issues were much more predominant.

The main issues encountered were on the computation of the gradient of the Lagrangian $\nabla\mathcal{L}$. This computation, when performed on stabilizer states, would always result on a null gradient. We believe this is caused by a bad choice of both Ansatz and stabilizer states in the initial decomposition. Ultimately, time constraints would prevent this issue to be fully investigated and the QNG method was scrapped in favour of improved stochastic algorithms such as parallel tempering.

Another hurdle encountered when adapting the QNG to our framework has come from the singular nature of the Quantum Geometric Tensor g when built from simple Ansätze and stabilizer states. One way to solve this issue is by the computation of the pseudo-inverse, which could negatively affect results if used haphazardly (for instance by computing the pseudo-inverse by inverting all singular values, regardless of their measure). Another way is by lifting the singularity using a small diagonal value: $g \leftarrow g + \epsilon\mathbb{1}$, with for instance $\epsilon \approx 10^{-4}$.

3.2 Parallel tempering

In this section, we will display an example of the output of the parallel tempering method to better visualize the way this algorithm works. This example has been done with a system of $N = 10$ qubits, with a decomposition of $R = 10$ stabilizer states, and at $J/h = 2$. This also serves as a sanity test before exploring other results. The performance of the algorithm will be studied and discuss in section 3.4.2.

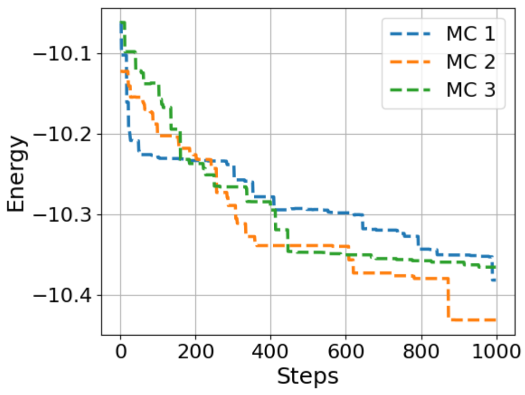


Figure 8: Evolution of the energy of each Markov chains during the execution of the parallel tempering method for $N = 10$, $R = 10$, $J/h = 0.5$

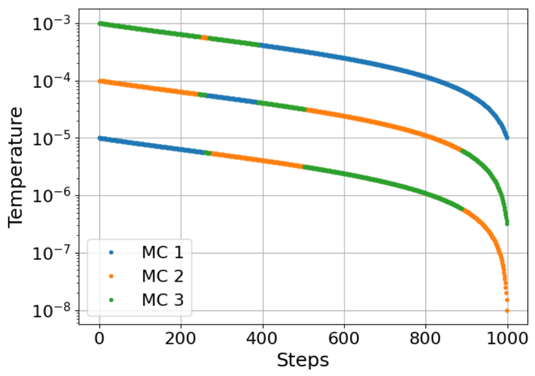


Figure 9: Evolution of the temperature of each Markov chains during the execution of the parallel tempering method. Note that the temperatures are swapped, represented by the changing color of the lines

3.3 Stabilizer state- and computational basis state decomposition

The heart of this work is the use of stabilizer states superpositions to approximate arbitrary states. However, a conceptually simpler and more intuitive way of decomposing states is through basis elements, like the computational basis states. In this section, we will show that approximate stabilizer decomposition offers better precision than approximate computational basis decompositions. This result is shown in figure 10, where we see that the stabilizer state decomposition gives a slight improvement over computational basis decompositions. Note that this result has been computed with only one run of the PT method with no restart, which means that final states may not be the best approximation possible. Nonetheless, we still observe that stabilizer states improve accuracy over computational basis states.

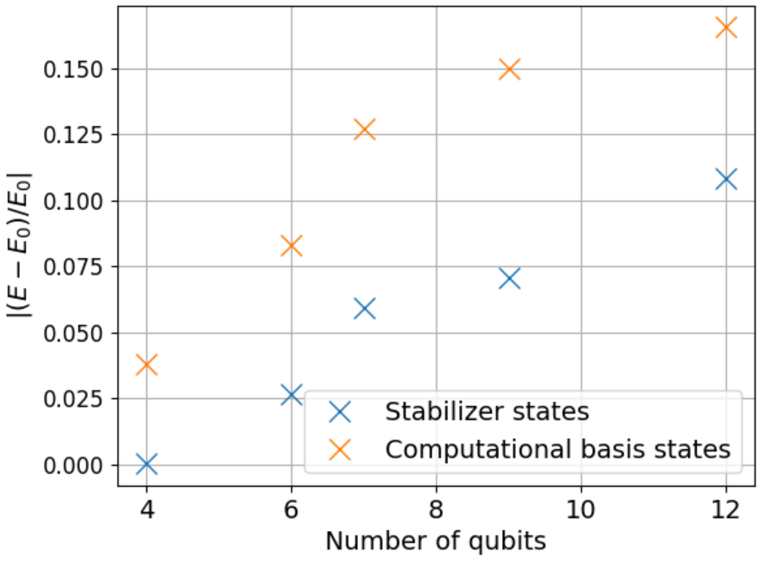


Figure 10: Stabilizer state decomposition and computational basis state decomposition of TIM ground state at $J/h = 1$ and $R = 4$

The advantage of stabilizer states can be intuitively understood for two main reasons. First, every computational basis element is itself a stabilizer state. Hence, basis state decompositions can be constructed from fewer different states. The second reason is that the computational basis is an orthonormal basis, while stabilizer states form an overcomplete basis. Approximate basis state decompositions cannot, in general, represent an arbitrary state perfectly by the very definition of a complete basis, which the computational basis is. In contrast, due to being overcomplete there is a higher chance that the basis of stabilizer states can approach an arbitrary state closer than basis states.

3.4 Performance optimizations and benchmarks

3.4.1 Optimisations: Julia, multithreading and code parallelisation

At first, the algorithm was implemented in Python. It is an interpreted, general-purpose programming language which can prove a powerful and easy to use tool. Being interpreted, however, hinders its execution speed when performing large-scale array operations compared to purpose-built compiled language. It does have access to purpose-built packages for array operations, such as NumPy. Most of these libraries circle around the speed issue by implementing their functions in C. As such, using NumPy drastically improves the computational efficiency of array operations in Python. Unfortunately, this project relies heavily on custom data sets (such as CH-forms) and functions, which restricts the use of NumPy functions for almost anything other than the base CH-form evolution. This problem could be solved by adapting theys functions to work with the custom classes, which in effect would mean coding the implementation in C. However, being low level, coding in C is more time-consuming. A more time-efficient middle ground was struck by recreating the framework and algorithm in Julia, for its good performance and ease of use.

A comparison of the performance between the two implementations is given in figure 11. The chosen metric to compare them is the computation of the expectation value $\langle\phi_1|\,\mathcal{H}\,|\phi_2\rangle$ for two arbitrary stabilizer states $|\phi_i\rangle$. This metric was selected as it essentially amounts to compute $O(N)$ scalar products of stabilizer states, which we know take $O(N^3)$ operations each to compute, which then takes $\mathcal{O}(N^4)$ to perform. The advantage of measuring this rather than a single scalar product is two-fold: First, the larger computational cost improves the accuracy of the time measurement as the overhead coming from other processes running simultaneously is comparatively lower. Furthermore, the precision also increases as the computation of scalar products is repeated $O(N)$ times. Second, the effect of multithreading is easily noticeable on this process. Indeed, up to about 20 qubits, the computations are effectively completed in time $O(N^2)$ also thanks to code parallelisation. For bigger systems, this effect becomes less noticeable and the process computational costs grows as $O(N^4)$. To better visualise this, lines representing $O(N^2)$ and $O(N^4)$ evolution have been added for both implementations in the figure 11.

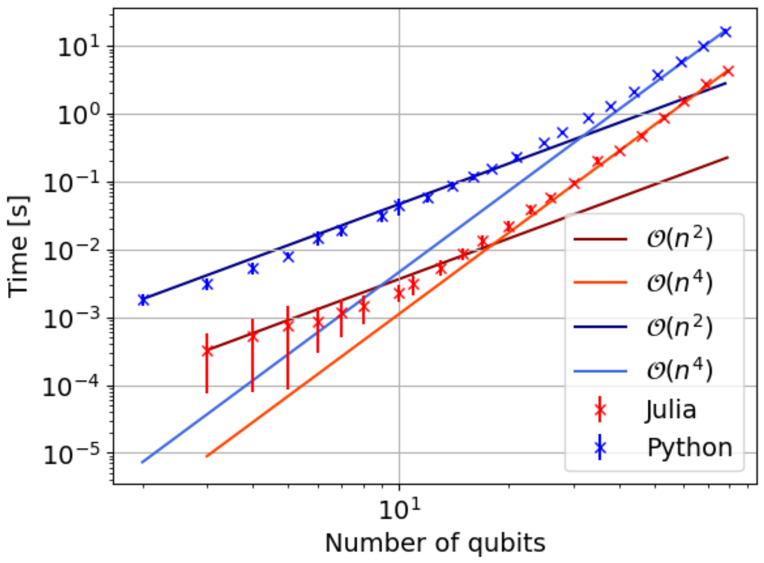


Figure 11: Computation time of the expectation value $\langle\phi_1|\,\mathcal{H}\,|\phi_2\rangle$ for two arbitrary stabilizer states

3.4.2 Comparison of Metropolis-Hastings Vs. parallel tempering

The parallel tempering method can offer better performance than a simple Metropolis algorithm, despite the added complexity of the chain swap. We will attempt to verify this in our implementation. The method to compare the two was as follows: First, the parallel tempering was run on a moderate amount of steps (for instance $\mathcal{O}(10^3)$), during which the execution time and allocated memory was logged. Second, an M.-H. algorithm was run one step at a time until either the energy obtained is lower than the one obtained by PT or until the chain had run a selected number of steps. This amount of steps would be typically the number of PT steps multiplied by the number of parallel chains.

The results obtained from this study are given in figures 12-14. We can observe that both algorithms show similar performance in execution speed for small systems, but the parallel tempering method display a slight improvement over larger ones. Additionally, total memory allocation of PT becomes comparable to Metropolis for larger systems. Figure 14 shows that the energies do match except for larger systems, where the PT method gives a considerably better result than MH. However, we observe only a marginal improvement in execution speed. This is a consequence of the methodology used to compare the two algorithms. Indeed, for smaller systems the number of steps might be too large for the PT algorithm, effectively wasting resources and allowing the Metropolis algorithm to catch up.This corroborates with the memory allocation; for larger systems, the MH algorithm does as many steps as the parallel chains do combined, hence the total memory allocation is similar. However for smaller systems, the MH algo does less steps, showing better performance than it would in a fairer comparison. As the system size increase, this flaw in the methodology becomes less present and the results are more trustworthy.

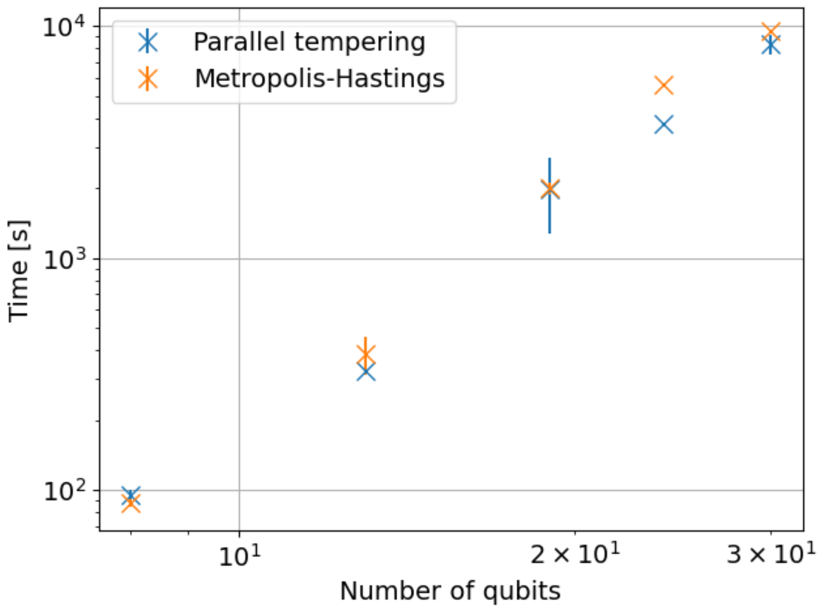


Figure 12: Execution time of parallel tempering and Metropolis-Hastings methods for decompositions of rank $R = 12$ and 3 parallel Markov chains

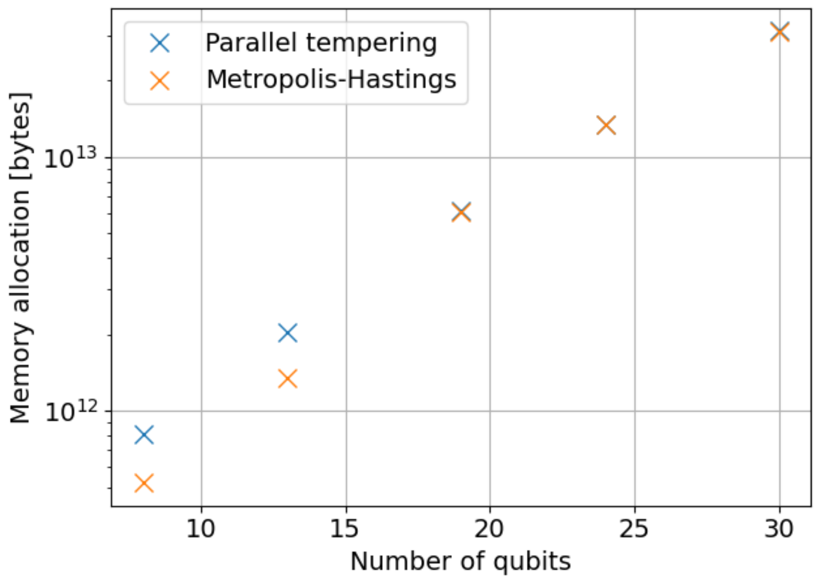


Figure 13: Total memory allocation PT and M.-H. algorithms for rank $R = 12$ decompositions and 3 parallel MCs

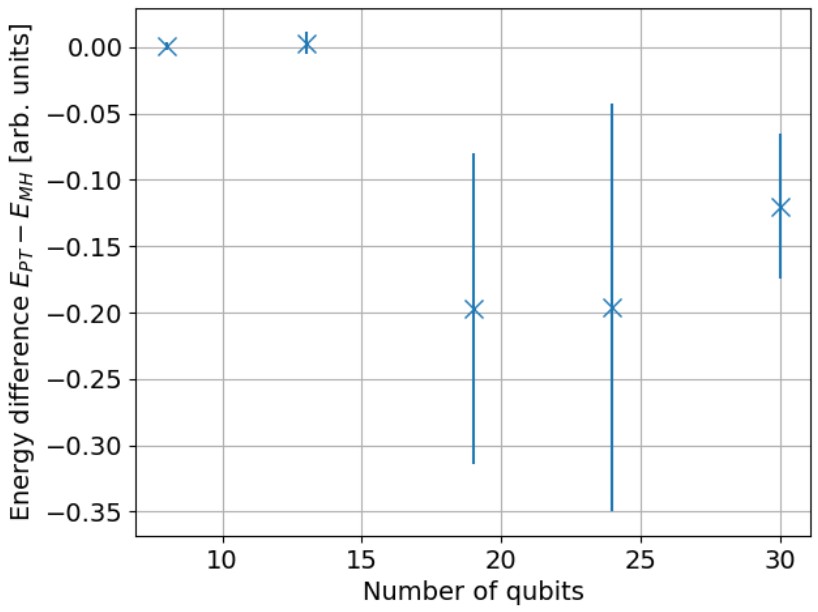


Figure 14: Ground state energy obtained by PT and MH methods for rank $R = 12$ decompositions and 3 parallel MCs

3.5 Approximation of arbitrary states by stabilizer state superposition

As explained in section 2.2.1, arbitrary states can be approximated as superpositions of stabilizer states. For fixed stabilizer rank decomposition, it is likely that some states are better approximated than others, as there are many more arbitrary states than stabilizer states. In the case of the transverse Ising model, we know that in the weak external field limit ($h = 0$) the ground state is given by the state $\left|0^{\otimes N}\right\rangle$ and in the decoupled spin limit ($J = 0$) is given by the state $\left|-^{\otimes N}\right\rangle$, $|-\rangle = 2^{-1/2}(|0\rangle - |1\rangle)$. Since $J/h = 1$ there is a phase transition, we expect the relative error on the energy of the best stabilizer approximation to peak. We do indeed observe this behaviour, as shown in figure 15. Note that the error curve flattens with increasing decomposition rank. This can be expected, since energy is a continuous function in the Hilbert space. Hence, the relative fidelity error $\epsilon := |(E - E_0)/E_0|$ is continous in the stabilizer rank R of a decomposition.

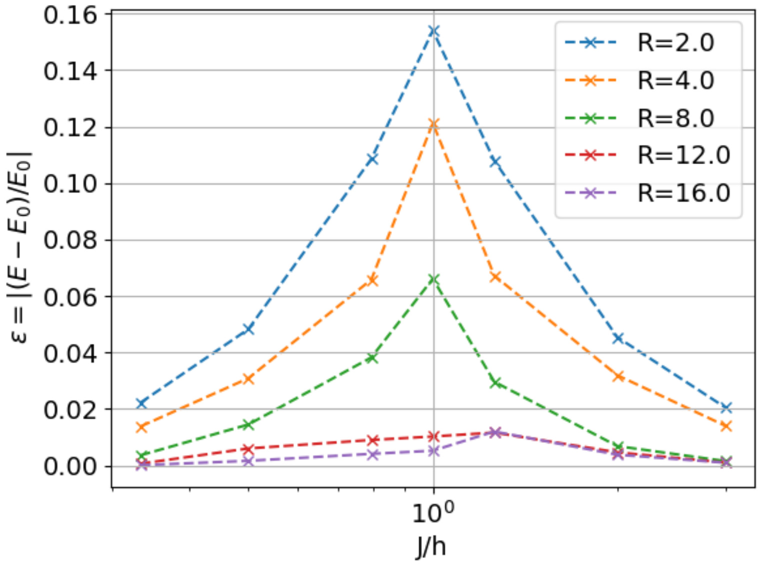


Figure 15: Relative error on the ground state energy for various rank decompositions for 8 qubits

Another way of understanding this concept is seeing it as the ground state at the phase transition depicts more prominently a non-Clifford behaviour, *i.e.* the decomposition requires a higher stabilizer rank to obtain a similar precision. The fact that the error is a continuous function of the stabilizer rank motivates us to extrapolate the exact stabilizer rank of each ground states shown in figure 15. We observe that the relative error on the ground state energy at $J/h = 1$ decreases seemingly linearly with the stabilizer rank in figure 16. From this figure, we see that the stabilizer rank of this state is around $R = 13$. This study has been performed at the phase transition point $J/h = 1$ in order to maximise the diversity of decomposition. What is meant by diversity here is that the larger stabilizer rank of this particular G.S. allows us to consider cases in both the $N > R$ and $N < R$ ranges. The approximate ground state at $R = 16$ has not been considered in the linear regression due to the way the optimization algorithm works: To maximise the convergence speed, the algorithm only considers state decompositions of linearly independent stabilizer states; if a sampled state is dependent on the other states, it is discarded and another state is sampled. This works well for approximate decomposition, but prevents the true ground state to be found for over-rank decompositions. The slight bump for $R = 16$ at the right of $J/h = 1$ is probably due in part to this property of the optimizer and carries no physical meaning.

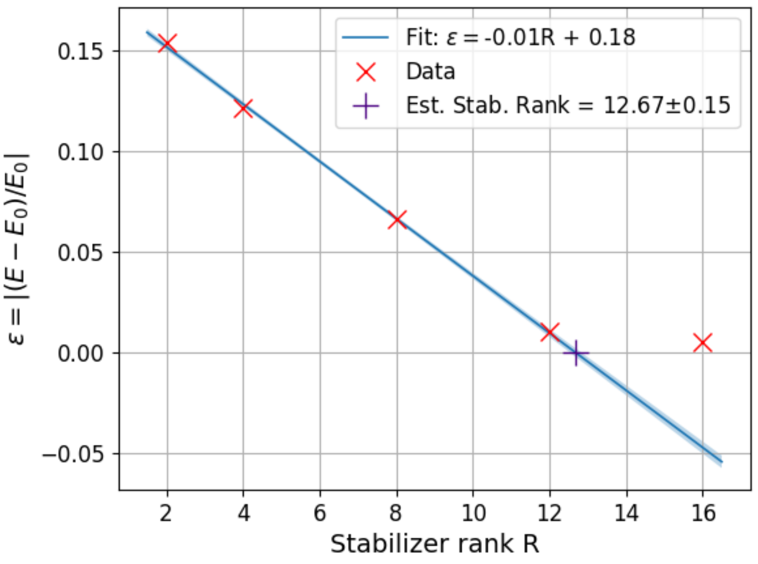


Figure 16: Relative error on the ground state energy at the phase transition $J/h = 1$ for 8 qubits

It is worthy of note that this linear regression is but an approximation, valid only for decompositions at low R/N. Were this linearity true for all stabilizer rank, it would imply that there is a polynomial relation between the full stabilizer rank of an arbitrary state and the size of the system (number of qubits). The latter is unlikely, as it would in turn imply that any quantum state is efficiently simulable by a classical computer. We will see shortly (figure 17) that for larger systems this linearity does not hold well at higher R/N.

The regression shown in figure 16 can now be done for greater number of qubits in order to study the relation between the number of qubits N and the stabilizer rank R. While only an approximation, this study would give us a some insight of the potential stabilizer decompositions have for arbitrary state approximation. The full stabilizer rank extrapolated in this way may be underestimated, since only low rank decompositions can be used. However, this first approximation still allows us to extrapolate insightful results. The number of stabilizer states grows faster than the number of dimensions, as there are $\mathcal{O}(2^{N^2})$ stabilizer states for a given N [19]. In general, this would mean that fewer stabilizer states are required to approximate states, as there is a higher chance to have one already close to a given state. Hence, we expect the full stabilizer rank to give an edge over usual (state vector) simulation in the form of a system size dependency of $\mathcal{O}(2^{\alpha N})$ with $\alpha < 1$. The expanded study is shown in figure 17.

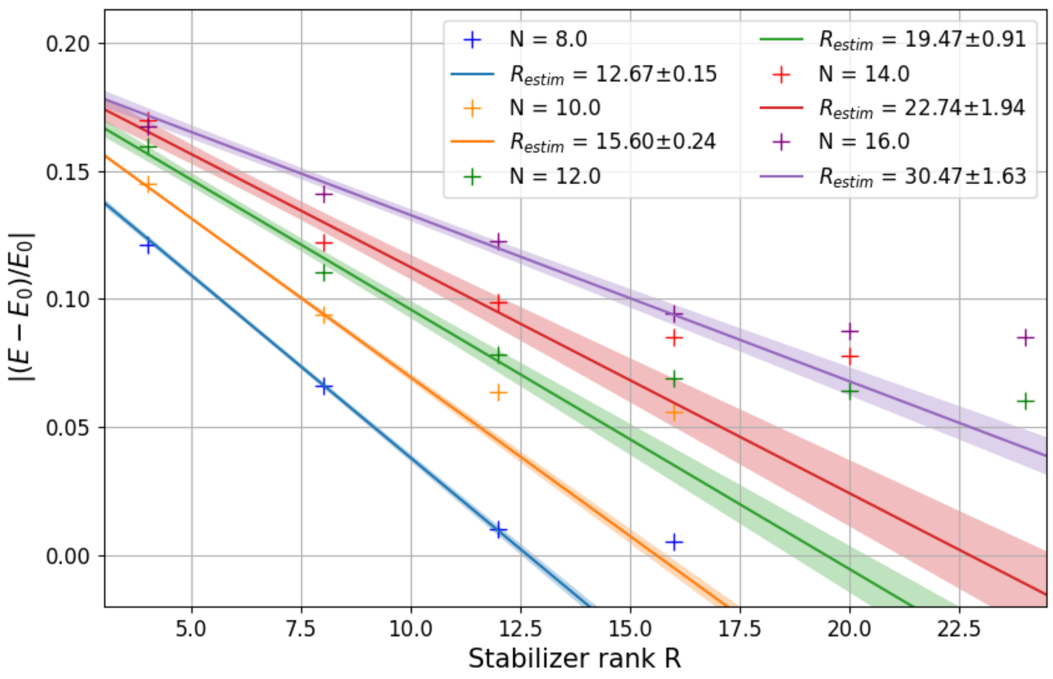


Figure 17: Relative error on the ground state energy at the phase transition $J/h = 1$

As predicted, the linear behaviour of the relative error in figure 17 does not hold at larger R/N for $N > 8$. Note that for 10 qubits, this law does not hold even at a lower ratio, however it is likely a statistical outlier. Due to the stochastic nature of the optimisation algorithm, it is likely that result for $N = 10$, $R = 12$ is not the best approximation of the ground state. Thus, this result has not been considered for the linear fit, which had the adverse effect of arbitrarily lowering the incertitude on the corresponding extrapolated stabilizer rank. However, we will motivate our decision from hindsight as we will see shortly that this particular result matches the rank evolution against N. This evolution is shown in figure 18. As expected, the exponent α is lower than one, though it is underestimated due to the way the stabilizer rank has been extrapolated. Nonetheless, this shows the performance improvement potential of using small stabilizer state decompositions to represent arbitrary quantum states.

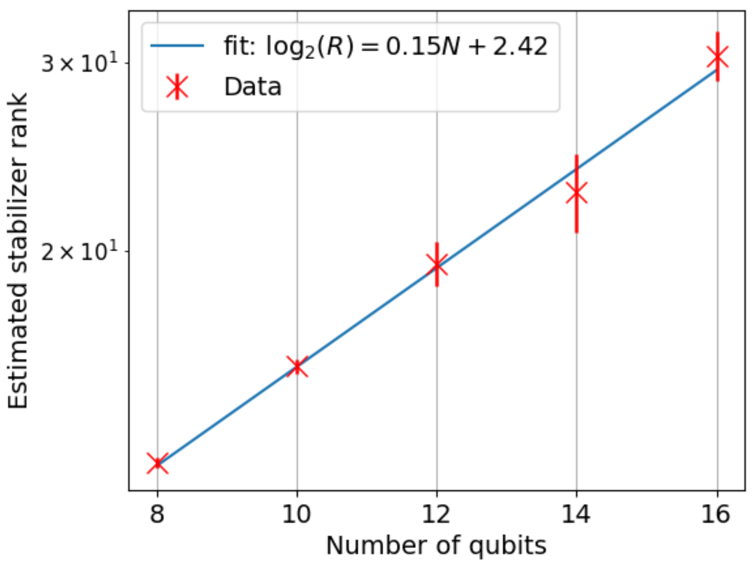


Figure 18: Extrapolated stabilizer rank of TIM ground states at $J/h = 1$

The evolution of the ratio R/N can also be visualised directly from the relative error on the ground state energy on figure 19. In order to avoid extremely long simulations, this study has been performed at $J/h = 2$. This brings the initial state closer to the true ground state energy-wise. Thus, shorter simulations still can reliably reach the latter state. Additionally, the study was performed on a relatively narrow range of stabilizer ranks and number of qubits in order to avoid the issue of over-rank decompositions (recall figure 15 for $R = 16$). The increase in the stabilizer rank can be observed by the increase in relative error on the diagonal $N = R$. This effect is less intense in this figure due to the true ground state being closer to the Hartee-Fock initial state.

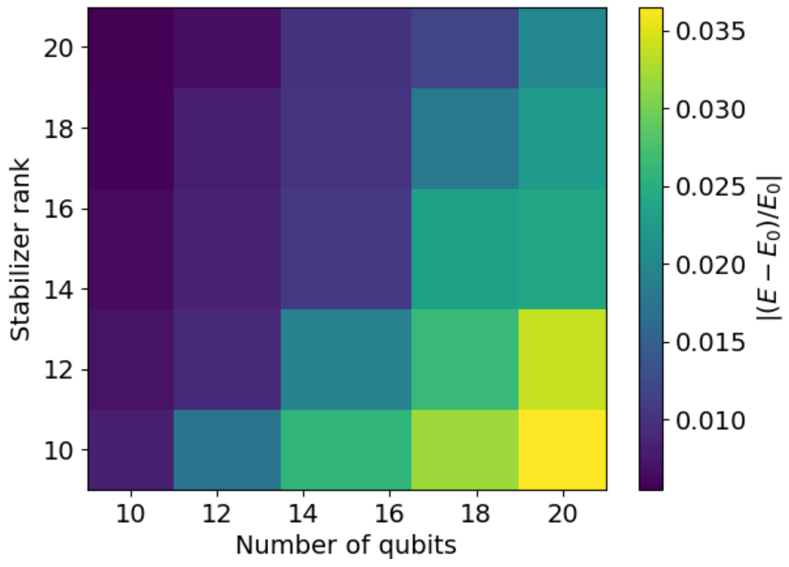


Figure 19: Colourmap of the relative error on the energy of TIM ground state for different approximate stabilizer rank at $J/h = 2$

Finally, from the data shown in figure 17, we can devise an empirical formula for the stabilizer rank R necessary to approximate a state of N qubits with error ϵ on its energy. This formula is obtained from the fit shown in figure 20, and reads:

$$R \approx \frac{0.20 - \epsilon}{0.15} N. \tag{77}$$

Note that this relation is in general only valid for R/N relatively small.

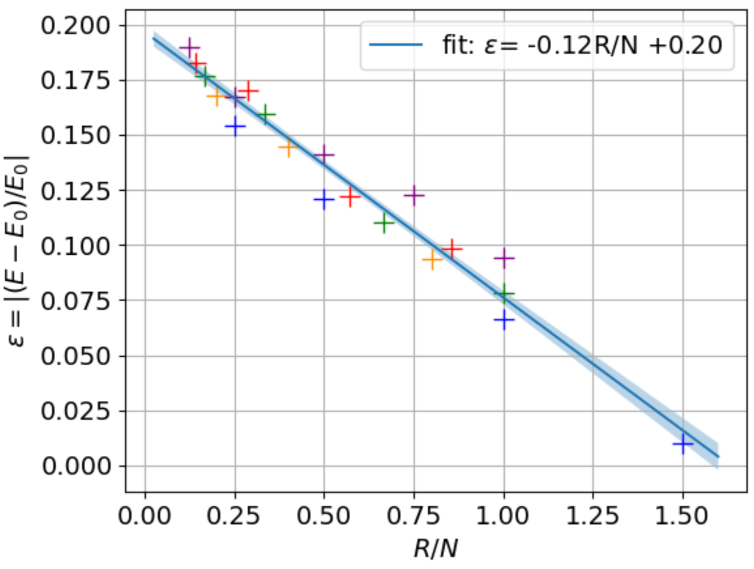


Figure 20: Relative error on the energy against the ratio R/N at $J/h = 1$; The markers are coloured according to the number of qubits using the same colours as figure 17

.

4 Conclusion

In this work, a ground state optimization algorithm based stabilizer decomposition of arbitrary states was presented. This algorithm allows us to generate stabilizer decompositions that approximate TIM ground states which are, in general, not stabilizer states themselves. The implementation makes use of the CH-form representation, which is an efficient way to store and update stabilizer states with polynomial ressources and time. At first the optimiser relied on the Metropolis-Hastings algorithm to work but due to low convergence speed other options where tried. The parallel tempering method has been chosen as the best fit due to issues implementing the quantum natural gradient. The implementation of the algorithm has been first done in Python, however its insufficient execution speed has motivated the shift towards a Julia implementation which significantly reduced execution time. Using the latter framework, stabilizer state were shown to approximate arbitrary states slightly better than computational basis states at fixed rank. Stabilizer state decomposition type offer a promising way of simulating quantum states with good accuracy and improved performance compared to the usual vector state reprensatation. In fact, we have found that the full stabilizer rank of ground states grow as $\mathcal{O}(2^{\alpha N})$ with $\alpha \approx 0.15$. Furthermore, we empirically devised a relation between the stabilizer rank of a decomposition and its relative error on the energy for a given system size. However this relation is only valid for small R/N.

Acknowledgements

I would like to express my most sincere thanks to Prof. Vincenzo Savona his guidance and for giving me the opportunity to study at LPTN throughout my Master's degree. I also wish to thank Dr. Fabrizio Minganti and Dr. Riccardo Rota for their help during all my time at LPTN. I am very grateful to Dr. Sofia Vallecorsa for allowing me to do my thesis at CERN Openlab, and Dr. Michele Grossi for his help and for coordinating the groups at EPFL and CERN. Finally, I want to give my heartfelt thanks to my friends and family for their love and support throughout my studies, without which they would have been incredibly harder.

References

[1] *IBM Quantum breaks the 100-qubit processor barrier*. en. Feb. 2021. URL: https://research.ibm.com/blog/127-qubit-quantum-processor-eagle (visited on 08/16/2022).

[2] *Simulators overview*. en. URL: https://quantum-computing.ibm.com/services/resources/docs/runtime/manage/simulator/ (visited on 08/16/2022).

[3] Xiu-Zhe Luo et al. "Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design". en-GB. In: *Quantum* 4 (Oct. 2020). Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften, p. 341. DOI: 10.22331/q-2020-10-11-341. URL: https://quantum-journal.org/papers/q-2020-10-11-341/ (visited on 08/02/2022).

[4] Sergey Bravyi et al. "Simulation of quantum circuits by low-rank stabilizer decompositions". en. In: *Quantum* 3 (Sept. 2019). arXiv: 1808.00128, p. 181. ISSN: 2521-327X. DOI: 10.22331/q-2019-09-02-181. URL: http://arxiv.org/abs/1808.00128 (visited on 05/16/2021).

[5] Elliott Lieb, Theodore Schultz, and D. Mattis. "Two Soluble Models of an Antiferromagnetic Chain". In: *Annals of Physics* 16 (Dec. 1961), pp. 407–466. ISSN: 978-3-642-06093-9. DOI: 10.1016/0003-4916(61)90115-4.

[6] Pierre Pfeuty. "The one-dimensional Ising model with a transverse field". en. In: *Annals of Physics* 57.1 (Mar. 1970), pp. 79–90. ISSN: 0003-4916. DOI: 10.1016/0003-4916(70)90270-8. URL: https://www.sciencedirect.com/science/article/pii/0003491670902708 (visited on 07/18/2022).

[7] Markus Heyl, Anatoli Polkovnikov, and Stefan Kehrein. "Dynamical Quantum Phase Transitions in the Transverse Field Ising Model". en. In: (June 2012). DOI: 10.1103/PhysRevLett.110.135704. URL: https://arxiv.org/abs/1206.2505v2 (visited on 07/18/2022).

[8] Christopher M. Dawson and Michael A. Nielsen. *The Solovay-Kitaev algorithm*. arXiv:quant-ph/0505030. Aug. 2005. DOI: 10.48550/arXiv.quant-ph/0505030. URL: http://arxiv.org/abs/quant-ph/0505030 (visited on 08/18/2022).

[9] Daniel Gottesman. "The Heisenberg Representation of Quantum Computers". In: *arXiv:quant-ph/9807006* (July 1998). arXiv: quant-ph/9807006. URL: http://arxiv.org/abs/quant-ph/9807006 (visited on 05/16/2021).

[10] Scott Aaronson and Daniel Gottesman. "Improved Simulation of Stabilizer Circuits". en. In: *Phys. Rev. A* 70.5 (Nov. 2004). arXiv: quant-ph/0406196, p. 052328. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.70.052328. URL: http://arxiv.org/abs/quant-ph/0406196 (visited on 05/16/2021).

[11] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. en. ISBN: 9780511976667 Publisher: Cambridge University Press. Dec. 2010. DOI: 10.1017/CBO9780511976667. URL: https://www.cambridge.org/highereducation/books/quantum-computation-and-quantum-information/01E10196D0A682A6AEFFEA52D53BE9AE (visited on 08/18/2022).

[12] Héctor J. García, Igor L. Markov, and Andrew W. Cross. "On the Geometry of Stabilizer States". In: *arXiv:1711.07848 [quant-ph]* (Nov. 2017). arXiv: 1711.07848. URL: http://arxiv.org/abs/1711.07848 (visited on 12/16/2021).

[13] M. Van den Nest. “Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond”. In: *arXiv:0811.0898 [quant-ph]* (Oct. 2009). arXiv: 0811.0898. URL: http://arxiv.org/abs/0811.0898 (visited on 05/16/2021).

[14] Francesco Tacchino et al. “Quantum computers as universal quantum simulators: state-of-art and perspectives”. In: *Adv Quantum Tech* 3.3 (Mar. 2020). arXiv: 1907.03505, p. 1900052. ISSN: 2511-9044, 2511-9044. DOI: 10.1002/qute.201900052. URL: http://arxiv.org/abs/1907.03505 (visited on 12/16/2021).

[15] IBM. *Quantum Experience*. en. 2021. URL: https://quantum-computing.ibm.com/ (visited on 05/27/2021).

[16] David J. Earl and Michael W. Deem. “Parallel tempering: Theory, applications, and new perspectives”. en. In: *Phys. Chem. Chem. Phys.* 7.23 (2005), p. 3910. ISSN: 1463-9076, 1463-9084. DOI: 10.1039/b509983h. URL: http://xlink.rsc.org/?DOI=b509983h (visited on 08/18/2022).

[17] James Stokes et al. “Quantum Natural Gradient”. In: *Quantum* 4 (May 2020). arXiv:1909.02108 [quant-ph, stat], p. 269. ISSN: 2521-327X. DOI: 10.22331/q-2020-05-25-269. URL: http://arxiv.org/abs/1909.02108 (visited on 07/25/2022).

[18] Naoki Yamamoto. *On the natural gradient for variational quantum eigensolver*. arXiv:1909.05074 [quant-ph]. Sept. 2019. DOI: 10.48550/arXiv.1909.05074. URL: http://arxiv.org/abs/1909.05074 (visited on 07/25/2022).

[19] D. Gross. “Hudson’s Theorem for finite-dimensional quantum systems”. In: *Journal of Mathematical Physics* 47.12 (Dec. 2006). arXiv:quant-ph/0602001, p. 122107. ISSN: 0022-2488, 1089-7658. DOI: 10.1063/1.2393152. URL: http://arxiv.org/abs/quant-ph/0602001 (visited on 08/11/2022).

A Scalar product of CH-form states

A.1 Scalar product of CH-form and computational basis states

We will explain here in more detail the method to compute scalar product of CH-form stabilizer states and computational basis states (section 2.2.4). The aim is to compute the scalar product $\langle x|\phi\rangle = \langle 0^N| X(x)\omega U_C U_H |s\rangle$. Recall that the idea is to construct an operator $Q = i^\mu Z(t)X(u) = i^\mu \prod_{j=1}^N Z_j^{t_j} X_j^{u_j}$ such that $\langle x|\phi\rangle = \omega \langle 0^N| Q U_H |s\rangle$. In practice, this means computing the N-bit strings u and t and the integer (modulo 4) μ from the bit string x and the CH-form. This requires using the definition of the stabilizer tableau of U_C (equation (8)). By using these definitions and $U_C |0^N\rangle = |0^N\rangle$, we get

$$\omega \langle 0^N| X(x)U_C U_H |s\rangle = \omega \langle 0^N| \prod_{p=1}^N (U_C^{-1} X_p U_C)^{x_p} U_H |s\rangle = \omega \langle 0^N| \prod_{p=1}^n (i^{\gamma_p} \prod_{j=1}^N X_j^{F_{p,j}} Z_j^{M_{p,j}})^{x_p} U_H |s\rangle, \tag{78}$$

where $x_p \in 0,1$ is the p^{th} element of the bit string x. Now, we use $X_p^x Z_p^z = (-1)^{xz} Z_p^z X_p^x$ and $X_p^x Z_j^z = Z_j^z X_p^x$ for $j \neq p$ to reorder the products of Pauli operators closer to the desired operator Q,

$$\omega \langle 0^N| X(x)U_C U_H |s\rangle = \omega \langle 0^N| \prod_{j=1}^N \prod_{p=1}^N (i^{\gamma_p x_p N^{-1}} (-1)^{x_p F_{p,j} x_p M_{p,j}} Z_j^{x_p M_{p,j}} X_j^{x_p F_{p,j}}) U_H |s\rangle. \tag{79}$$

Once again, using the same commutation relation between X and Z we can compute the result of the product on p and find an expression for Q,

$$\prod_{j=1}^N \prod_{p=1}^N i^{\gamma_p x_p N^{-1}} (-1)^{x_p F_{p,j} x_p M_{p,j}} Z_j^{x_p M_{p,j}} X_j^{x_p F_{p,j}} = \tag{80}$$

$$= (\prod_{p=1}^N i^{\gamma_p x_p N^{-1}} (-1)^{x_p F_{p,1} x_p M_{p,1}} Z_1^{x_p M_{p,1}} X_1^{x_p F_{p,1}})(\prod_{p=1}^N i^{\gamma_p x_p N^{-1}} (-1)^{x_p F_{p,2} x_p M_{p,2}} Z_2^{x_p M_{p,2}} X_2^{x_p F_{p,2}}) \dots \tag{81}$$

$$= (\prod_{p=1}^N i^{2\gamma_p x_p N^{-1} + 2(x_p F_{p,1} x_p M_{p,1} + x_p F_{p,2} x_p M_{p,2})} Z_1^{x_p M_{p,1}} X_1^{x_p F_{p,1}} Z_2^{x_p M_{p,2}} X_2^{x_p F_{p,2}}) \dots \tag{82}$$

$$= (\prod_{p=1}^N i^{2\gamma_p x_p N^{-1} + 2(x_p F_{p,1} x_p M_{p,1} + x_p F_{p,2} x_p M_{p,2} + x_p F_{p,1} x_p M_{p,2})} Z_1^{x_p M_{p,1}} Z_2^{x_p M_{p,2}} X_1^{x_p F_{p,1}} X_2^{x_p F_{p,2}}) \dots \tag{83}$$

$$\vdots$$

$$= i^{\gamma\cdot x + 2b} \prod_{j=1}^N (\prod_{p=1}^N Z_j^{x_p M_{p,j}} \prod_{p=1}^N X_j^{x_p F_{p,j}}) = i^{\gamma\cdot x + 2b} \prod_{j=1}^N Z_j^{t_j} X_j^{u_j}, \tag{84}$$

where we can define or identify $\gamma \cdot x := \sum_{p=1}^N \gamma_p x_p$, $t_j := \sum_{p=1}^N x_p M_{p,j}$, $u_j := \sum_{p=1}^N x_p F_{p,j}$ and

$$b := \sum_{j=1}^N \sum_{l=1}^N x_l F_{l,j} \left(\sum_{k=l}^N x_k M_{k,j} \right). \tag{85}$$

And finally we get equation (34):

$$\omega \left\langle 0^N \right| X(x)U_C U_H \left|s\right\rangle = \omega \left\langle 0^N \right| i^{\gamma\cdot x+2b} \prod_{j=1}^{N} Z_j^{t_j} X_j^{u_j} U_H \left|s\right\rangle .$$

A.2 Product of U_C operators

First, we will concentrate on the first part of equation (40). Using equation (8) we find,

$$(U_C^2 U_C^1)^{-1} Z_q U_C^2 U_C^1 = (U_C^1)^{-1} \prod_{j=1}^{N} Z_j^{G^2_{q,j}} U_C^1 = \prod_{j=1}^{N} ((U_C^1)^{-1} Z_j U_C^1)^{G^2_{q,j}} \tag{86}$$

$$= \prod_{j=1}^{N} \left(\prod_{p=1}^{N} Z_p^{G^1_{j,p}} \right)^{G^2_{q,j}} = \prod_{p} Z_p^{(G^2\cdot G^1)_{q,p}}, \tag{87}$$

which clearly indicate that $G' = G^2 \cdot G^1$. Now for the second equation,

$$(U_C^2 U_C^1)^{-1} X_q U_C^2 U_C^1 = (U_C^1)^{-1} (i^{\gamma_q^2} \prod_{j=1}^{N} X_j^{F^2_{q,j}} Z_j^{M^2_{q,j}}) U_C^1 \tag{88}$$

$$= i^{\gamma_q^2} \prod_{j=1}^{N} (U_C^1)^{-1} X_j^{F^2_{q,j}} U_C^1 (U_C^1)^{-1} Z_j^{M^2_{q,j}} U_C^1 \tag{89}$$

$$= i^{\gamma_q^2} \prod_{j=1}^{N} \left(i^{\gamma_j^1} \prod_{p=1}^{N} X_p^{F^1_{j,p}} Z_p^{M^1_{j,p}} \right)^{F^2_{q,j}} \left(\prod_{p=1}^{N} Z_p^{G^1_{j,p}} \right)^{M^2_{q,j}} \tag{90}$$

$$= i^{\gamma_q^2 + (F^2\cdot\gamma^1)_q} \prod_{j=1}^{N} \prod_{p=1}^{N} X_p^{F^2_{q,j} F^1_{j,p}} Z_p^{F^2_{q,j} M^1_{j,p} + M^2_{q,j} G^1_{j,p}}. \tag{91}$$

To simplify the notation, let us define $a^j_{q,p} := F^2_{q,j} F^1_{j,p}$ and $b^j_{q,p} := F^2_{q,j} M^1_{j,p} + M^2_{q,j} G^1_{j,p}$. Using $X_p^x Z_p^z = (-1)^{xz} Z_p^z X_p^x$ and $X_p^x Z_j^z = Z_j^z X_p^x$ for $j \neq p$, we get

$$\prod_{j=1}^{N} \prod_{p=1}^{N} X_p^{F^2_{q,j} F^1_{j,p}} Z_p^{F^2_{q,j} M^1_{j,p} + M^2_{q,j} G^1_{j,p}} = \prod_{j=1}^{N} \prod_{p=1}^{N} X_p^{a^j_{q,p}} Z_p^{b^j_{q,p}} \tag{92}$$

$$= (\prod_{p=1}^{N} X_p^{a^1_{q,p}} Z_p^{b^1_{q,p}})(\prod_{p=1}^{N} X_p^{a^2_{q,p}} Z_p^{b^2_{q,p}})(\prod_{p=1}^{N} X_p^{a^3_{q,p}} Z_p^{b^3_{q,p}}) \dots \tag{93}$$

$$= (\prod_{p=1}^{N} (-1)^{a^2_{q,p} b^1_{q,p}} X_p^{a^1_{q,p} + a^2_{q,p}} Z_p^{b^1_{q,p} + b^2_{q,p}})(\prod_{p=1}^{N} X_p^{a^3_{q,p}} Z_p^{b^3_{q,p}}) \dots \tag{94}$$

$$= (\prod_{p=1}^{N} (-1)^{a^2_{q,p} b^1_{q,p} + a^3_{q,p}(b^1_{q,p} + b^2_{q,p})} X_p^{a^1_{q,p} + a^2_{q,p} + a^3_{q,p}} Z_p^{b^1_{q,p} + b^2_{q,p} + b^3_{q,p}}) \dots \tag{95}$$

$$\vdots$$

$$= \prod_{p=1}^{N} (-1)^{\sum_{j=2}^{N} a^j_{q,p} \left(\sum_{k=1}^{j-1} b^k_{q,p}\right)} X_p^{\sum_{j=1}^{N} a^j_{q,p}} Z_p^{\sum_{j=1}^{N} b^j_{q,p}}. \tag{96}$$

Hence, combining this result back in equation (91) and using the definitions of $a^j_{q,p}$ and $b^j_{q,p}$, we find

$$(U_C^2 U_C^1)^{-1} X_q U_C^2 U_C^1 = i^{\gamma_q^2 + (F^2 \cdot \gamma^1)_q + 2\sum_{p=1}^N \sum_{j=2}^N a^j_{q,p} \left(\sum_{k=1}^{j-1} b^k_{q,p}\right)} \prod_{p=1}^N X_p^{(F^2 \cdot F^1)_{q,p}} Z_p^{(F^2 \cdot M^1 + M^2 \cdot G^1)_{q,p}}. \quad (97)$$

From these equations we identify the relations given in equation (41).