UNIVERSITA' DEGLI STUDI DI PERUGIA
Facoltà di Scienze Matematiche Fisiche e Naturali

CORSO DI LAUREA SPECIALISTICA IN MATEMATICA

Tesi di Laurea

Classical simulation of quantum circuits

Laureando
Tommaso Gagliardoni

Relatore
Ch.mo Prof. Marco Baioletti

Anno Accademico 2009/2010

Questo lavoro è dedicato alla mia famiglia, che mi ha sopportato e aiutato nei momenti difficili. Grazie a tutti voi.

Acknowledgements

It is a pleasure to thank the many people who helped me to achieve this result. I am really grateful to my advisor, Marco Baioletti, for all his support and his patience, and for having accepted to help me in this quest. I want to thank mostly Simone Severini from University College London, who helped me a lot along all the making of the work despite distance. I am also thankful to Jens Eisert, which pointed me to many good ideas during the months of my Erasmus in his research group in Potsdam, a great experience that I will never forget. I have met a lot of friendly people there, but I want to thank in particular Arnau Riera for the many interesting discussions on Fermionic Hamiltonians, Niel de Beaudrap for useful discussions about complexity classes and Andrea Mari for having been a precious source of informations. I owe a debt of gratitude to Rita Vincenti, who encouraged and allowed me to do new experiences, both from a human and an academical perspective. Thanks also to Ivan Gerace, who gave me useful hints on how to extend some of the final results, to Loriano Storchi, his advices and support were really precious, to Stefano Mancini from Università di Camerino, for having introduced me to the topic of quantum circuit simulation, and to Beatrice Vinci, who solved many bureaucratic difficulties I had. Last but not least, I want to thank my family for having helped me in every moment of my life, but especially in the moment of greatest need.

Contents

Chapter 1

Introduction

Quantum Computing is a branch of Computer Science; its main object of study is the quantum computer, an abstract model of computation which differs from a classical computer in being described by the laws of Quantum Mechanics (QM).

QM is a physical theory born at the end of the 19th century to deal with unexpected experimental results which classical physics routinely failed to explain (see, *e.g.*, [22]). This gave rise to a mathematically elegant theory, but with counterintuitive consequencies still not completely understood. It was not before many years and the discovery of surprising phenomena correctly predicted by this theory that QM was accepted as something more than a mere mathematical formalism [26]. To date, QM is probably one of the most successful physical theory ever devised, despite its foundations being still not completely understood, and its consequences remarkable.

The idea that quantum processes are computationally hard to simulate already appeared in the work of Richard Feynman during the early 80s. Such a vision described what today is a very large and active area of multidisciplinary research with fringes between Physics, Computer Science, Mathematics, and Engineering: Quantum Information Processing (QIP). The goal of QIP is to understand how information can be stored and processed with the use of quantum mechanical systems. The basic conceptual difference with respect to classical information processing (where 'classical' customarily means 'non-quantum') is that the quantum evolution of a system completely iso-

lated from the environment is reversible. In other words, reversibility implies that no information can be deleted during a computation and that a previous stage can be recovered by running the computation backwards. This fact is interpreted as an incarnation of the Landauer's Principle, stating that deleting information has a thermodynamical cost. Mathematically, reversibility is expressed by the fact that the time evolution of a quantum system is induced by unitary operators. While unitarity implies strong constraints on the dynamics of the system (for example, the impossibility of copying quantum information), it also gives rise to important phenomena, like interference, and non-classical correlations between subsystems. These phenomena, in turn, lead to interesting applications which make QIP an intriguing research area. Examples of these applications include teleportation of information, superdense coding [5], quantum cryptography and the possibility of breaking the security of the most used cryptosystems for Internet communications, financial transactions and military purposes [24]. Moreover, a deeper understanding of a quantum computational model could lead to an insightful comprehension of the physical foundations of QM itself.

Despite a great interest into this field, the practical realization of a model of quantum computer has proven to be very challenging; this is because of physical constraints like the difficulty of obtaining noise-free systems, or the characteristic of quantum states of being quickly degradated by their interactions with the environment (the so-called phenomenon of decoherence). For this reason, a different approach which has been taken into account is the possibility of simulating the behaviour of a quantum system [15,19] through a classical computational model – that is, a classical computer. In the general case this can only be done very inefficiently, *i.e.*, with an unaffordable cost in terms of computational space and time [27,3], but for certain classes of quantum systems it is possible nevertheless. In such instances we talk of *efficient classical simulation of quantum systems* [25].

A very used theoretical model for quantum computing is the quantum circuit model [6]. This is a formalism describing a circuit in terms of wires, gates and registers, in analogy to the classical circuit model describing a classical computer. The difference is that in the quantum model we deal

with quantum objects and quantum information; *e.g.*, we do not have bits but qubits ('qu-antum bits'), and quantum gates which are rather different from classical logical gates.

In 2008, a work by Richard Jozsa and Akimasa Miyake [17] showed that there is a particular class of quantum circuits, called *matchgates*, that can be classically efficiently simulated under a certain topological condition of the circuit itself. The condition is about locality of interactions, namely, matchgates acting on two geometrically nearest-neighbouring qubits can be efficiently simulated, while other matchgates in general can not. Previous works had shown [10] that any quantum circuit can be obtained by the composition of just nearest-neighbour matchgates and next-nearest-neighbour matchgates (that is, acting on two quantum wires distant at most one wire apart from each other). This gives a stunning result: classical and quantum computation power are bridged by a seemingly innocuous property of the circuit, that is, the possibility of these gates to act on near or distant lines. Matchgates acting on distant lines are in general not efficiently simulatable [16]. They appear in any circuit model performing quantum computation believed to be stronger than its classical counterpart. So, it seems that next-nearest-neighbours matchgates define exactly the computational superiority of the quantum model.

In this work we give the following improvements to this approach:

- a sufficient criterion to test whether a next-nearest-neighbour matchgate is efficiently simulatable;
- an algorithm to perform an exhaustive search to build simulatable matchgates;
- an explicit example of a simulatable next-nearest-neighbour matchgate found through an implementation of this algorithm in C language;
- a new framework to efficiently simulate matchgates acting on any arbitrary pair of lines provided their structure respects a particular form.

The central idea behind this last result is based on a work by Frank Verstraete and Juan Ignacio Cirac [7] on the study of Hamiltonians for multidimensional Fermionic systems.

The remainder of this work will be structured as follows.

In Chapter 2 we give the necessary preliminary notions about linear algebra and QM used in the rest of this work.

In Chapter 3 we describe both classical and quantum models of computation, marking differences and analogies.

In Chapter 4 we give an introduction to computational complexity theory, both for the classical and the quantum computational models; we discuss the concept of resource cost of a computation and the concept of classical simulability of a quantum computer.

In Chapter 5 we discuss the Fermionic representation for a quantum system and the Jordan-Wigner Transform, a useful tool for the description of quantum Hamiltonians.

In Chapter 6 we introduce matchgates, and we show that every quantum circuit can be represented using only nearest-neighbour and next-nearest-neighbour matchgates.

In Chapter 7 we introduce a Clifford algebra representation for matchgates, and we prove that through this formalism it is possible to efficiently simulate nearest-neighbour matchgates by classical means up to arbitrary accuracy.

In Chapter 8 we give the forementioned improvements to these results.

Finally, we discuss some possible future directions of our work.

Chapter 2

Preliminary notions

In this chapter we will fix the notations and the fundamental concepts used in the rest of the work. In Section 1 we will define Hilbert spaces and tensor products and will address the basic properties of these objects; we will also define particular classes of operators and some of their properties. In Section 2 we will introduce the theory of QM and will explain the concept of qubit and the process of measurement of an observable.

Let $\mathbb{N}$ be the set of natural numbers not including zero. Let $\mathbb{N}_0 = \mathbb{N} \cup \{ 0 \}$. Let $\mathbb{Z}$ be the set of integers, $\mathbb{R}$ be the set of the reals, and $\mathbb{C}$ the set of complex numbers. The imaginary unit is $i = \sqrt{-1}$.

If α is a complex number then $\overline{\alpha}$ will denote its complex conjugate. Similarly, if A is a complex matrix or an operator, $\overline{A}$ will denote its complex conjugate.

If A is a matrix, then A^T will denote its transpose, while $A^\dagger$ will denote its conjugate transpose, *i.e.*, $A^\dagger = \overline{A^T}$. If A is an operator, then $A^\dagger$ will be called its adjoint operator or Hermitian conjugate. We denote by O and I the zero and identity matrix respectively.

If $\mathcal{E}$ is a set, then we will denote by $\#\mathcal{E}$ its cardinality. Given a subset $\mathcal{S} \subset \mathcal{E}$, we will denote by $\mathcal{E} \setminus \mathcal{S}$ the corresponding complementary set.

Given two sets $\mathcal{E}$ and $\mathcal{F}$, we will denote by $\mathcal{E} \times \mathcal{F}$ their Cartesian product.

The *Kronecker delta* is $\delta_{jk} = \begin{cases} 1 & \text{if } j = k \\ 0, & \text{otherwise} \end{cases}$, for $j, k \in \mathbb{N}$.

If A and B are operators and the image of B is contained in the domain of A, then $A \circ B$ is the composition of A after B.

The symbol $\doteq$ is used in definitions.

2.1 Linear Algebra

We will restrict the following definitions to the case of the complex field $\mathbb{C}$, even though most of the following theory holds in the case of generic fields. Our choice is mainly motivated by physical reasons, one of the postulates of QM being the need of a complex space to model physical systems.

2.1.1 Hilbert spaces

We will denote by $\mathbf{0}$ the zero vector. We will also restrict our discussion to finite-dimensional spaces. This choice is made for the sake of simplicity: in QM it is not uncommon to deal with infinite-dimensional Hilbert spaces when representing even low-complexity systems (*i.e.*, the position of a particle, the time evolution of a wave function, *etc.*); in the Quantum Information Theory considered in this work, we will only take into account finite-dimensional spaces. A good introduction to Hilbert spaces (comprising proofs of most of the following results) can be found in [4].

The definition of Hilbert space is as follows:

Definition 2.1.1 (Hilbert space)**.** *A* complex Hilbert space $\mathcal{H}$ *is a complex vector space with an inner product operation* $(.,.) : \mathcal{H} \times \mathcal{H} \to \mathbb{C}$ *such that:*

1. $(\mathbf{u}, \mathbf{v}) = \overline{(\mathbf{v}, \mathbf{u})}$, *for all* $\mathbf{u}, \mathbf{v} \in \mathcal{H}$;

2. $(\alpha\mathbf{u} + \beta\mathbf{v}, \mathbf{w}) = \alpha(\mathbf{u}, \mathbf{w}) + \beta(\mathbf{v}, \mathbf{w})$, *for all* $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathcal{H}$, *for all* $\alpha, \beta \in \mathbb{C}$;

3. $(\mathbf{u}, \mathbf{u}) \geq 0$, *for all* $\mathbf{u} \in \mathcal{H}$, *with equality holding if and only if* $\mathbf{u} = \mathbf{0}$.

In the general case we also require the following condition:

4. *the space* $\mathcal{H}$ *is complete as a metric space defined by the norm induced by* $(.,.)$.

In the finite-dimensional case this last condition can be proven to be redundant.

We will denote by $\langle \mathbf{v_1}, \ldots, \mathbf{v_k} \rangle$ the linear subspace spanned by the elements $\mathbf{v_1}, \ldots, \mathbf{v_k} \in \mathcal{H}$. Obviously, if $\mathcal{H}$ is a Hilbert space, then also $\langle \mathbf{v_1}, \ldots, \mathbf{v_k} \rangle$ is a Hilbert space (contained in $\mathcal{H}$) with the same inner product.

We give a definition of basis for a generic Hilbert space as follows:

Definition 2.1.2 (Hilbert basis)**.** *a* Hilbert basis *for* $\mathcal{H}$ *is a set* $\{ \mathbf{e_1}, \ldots, \mathbf{e_n} \} \subset \mathcal{H}$ *such that:*

1. $\|\mathbf{e_j}\| = 1$, *for all* $j = 1, \ldots, n$

2. $(\mathbf{e_j}, \mathbf{e_k}) = \delta_{j,k}$, *for all* $j, k = 1, \ldots, n$

3. $\langle \mathbf{e_1}, \ldots, \mathbf{e_n} \rangle$, *as a vector subspace, is* dense *in* $\mathcal{H}$.

In the finite-dimensional case, the last property can be rephrased by the request that $\{ \mathbf{e_1}, \ldots, \mathbf{e_n} \}$ spans the whole space.

As for vector spaces it can be shown that, for a given Hilbert space, every Hilbert basis has the same cardinality (in this case n). We define that cardinality to be *the dimension* of the Hilbert space. It is easy to see that a Hilbert basis for $\mathcal{H}$ (as a Hilbert space) is also a basis for $\mathcal{H}$ seen as a vector space. The converse is generally false for infinite-dimensional spaces.

In general, a set of vectors $\{ \mathbf{e_1}, \ldots, \mathbf{e_k} \}$ will be called an *orthonormal set* if it satisfies only conditions 1 and 2 from Definition 2.1.2. Notice that, given a set $\{ \mathbf{v_1}, \ldots, \mathbf{v_k} \}$ of linearly independent vectors, it is always possible to find an orthonormal set $\{ \mathbf{e_1}, \ldots, \mathbf{e_k} \}$, where each $\mathbf{e_j}$ is a linear combination of $\mathbf{v_1}, \ldots, \mathbf{v_k}$, using the Graham-Schmidt orthonormalization process.

Given an arbitrary vector $\mathbf{v} \in \mathcal{H}$ and a Hilbert basis $\{ \mathbf{e_1}, \ldots, \mathbf{e_n} \}$, it is always possible to decompose $\mathbf{v}$ as a linear combination of basis elements:

$$\mathbf{v} = \sum_{j=1}^{n} \alpha_j \mathbf{e_j},$$

where $\alpha_j = (\mathbf{v}, \mathbf{e_j}) \in \mathbb{C}$, for all $j = 1, \ldots, n$. The element $(\mathbf{v}, \mathbf{e_j})\, \mathbf{e_j}$ is called the *projection of* $\mathbf{v}$ *onto the direction* $\mathbf{e_j}$.

2.1.2 Tensor products

The intuitive meaning of tensor product between two spaces – representing the space of the possible states of two separate physical systems – is to consider a single, 'larger' space, representing the space of the possible states of the two physical systems taken together. Similarly, the tensor product of two vectors in two different spaces (each one defining the state of a different physical system) is a vector of a larger space defining the state of the joint system; the tensor product of two operators (each acting on a different space) is an operator acting on the tensor products of the respective domain and target spaces.

The theory of tensor products is quite complex and even giving the necessary definitions can be tricky. We will restrict ourselves only to the simplest cases. A general introduction to tensor products in A-modules can be found in [2].

Tensor products in vector spaces

We start by giving the definition of bilinear mapping between vector spaces:

Definition 2.1.3 (Bilinear mapping). *Let $\mathcal{U}, \mathcal{V}, \mathcal{W}$ be complex vector spaces. An application $f : \mathcal{U} \times \mathcal{V} \to \mathcal{W}$ is said to be* bilinear *if and only if:*

1. *the application $\mathbf{v} \mapsto f(\mathbf{u}, \mathbf{v})$ is linear for every $\mathbf{u} \in \mathcal{U}$;*
2. *the application $\mathbf{u} \mapsto f(\mathbf{u}, \mathbf{v})$ is linear for every $\mathbf{v} \in \mathcal{V}$.*

The following proposition is fundamental:

Proposition 2.1.4. *Let $\mathcal{U}, \mathcal{V}$ be two complex vector spaces. Then there exists a pair $(\mathcal{T}, f)$, where $\mathcal{T}$ is a vector space and $f : \mathcal{U} \times \mathcal{V} \to \mathcal{T}$ is bilinear, with the following properties:*

1. *if $\mathcal{W}$ is another vector space and $g : \mathcal{U} \times \mathcal{V} \to \mathcal{W}$ is another bilinear application, then there exists a unique linear application $\phi_{\mathcal{W}} : \mathcal{T} \to \mathcal{W}$ such that $g = \phi_{\mathcal{W}} \circ f$;*
2. *if $(\mathcal{T}, f)$ and $(\mathcal{T}', f')$ are two pairs with the previous property, then there esists an unique isomorphism $\psi : \mathcal{T} \to \mathcal{T}'$ such that $f' = \psi \circ f$.*

This leads to the following definition:

Definition 2.1.5 (Tensor product of two vector spaces). *With respect to Proposition 2.1.4, given a pair $(\mathcal{T}, f)$, the* tensor product between $\mathcal{U}$ and $\mathcal{V}$ *is defined as the vector space $\mathcal{T}$ and is denoted as $\mathcal{U} \otimes \mathcal{V}$. We say that $\mathcal{T}$ is a* representation *of $\mathcal{U} \otimes \mathcal{V}$ via f.*

Notice that the meaning of $\mathcal{U} \otimes \mathcal{V}$ is ambiguous if f is not chosen. However, in such a case the possible resulting spaces are all isomorphic. The role of f is to select a choice of basis for $\mathcal{U}$ and $\mathcal{V}$ when constructing $\mathcal{T}$.

Definition 2.1.6 (Tensor product of two vectors). *With respect to Proposition 2.1.4, given a pair $(\mathcal{T}, f)$, the* tensor product between $\mathbf{u}$ and $\mathbf{v}$ *is defined as the vector $\mathbf{u} \otimes \mathbf{v} \doteq f(\mathbf{u}, \mathbf{v})$.*

Unless otherwise stated, we will consider understood the choice of f. This will be usually done by considering the standard basis of the spaces involved, with their respective standard ordering.

Proposition 2.1.7. *If* $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ *is a basis for* $\mathcal{U}$ *and* $\{\mathbf{f}_1, \ldots, \mathbf{f}_m\}$ *is a basis for* $\mathcal{V}$*, then the set* $\{\mathbf{e}_j \otimes \mathbf{f}_k \mid j = 1, \ldots, n; k = 1, \ldots, m\}$ *is a basis for* $\mathcal{U} \otimes \mathcal{V}$*.*

This means that every vector $\mathbf{w} \in \mathcal{U} \otimes \mathcal{V}$ can be decomposed as:

$$\mathbf{w} = \sum_{j=1}^{n} \sum_{k=1}^{m} \alpha_{jk} \mathbf{e}_j \otimes \mathbf{f}_k,$$

where $\alpha_{jk} \in \mathbb{C}$ for all j, k.

Definition 2.1.8 (Tensor product of two operators)**.** *Let* $\mathcal{U}, \mathcal{V}, \mathcal{R}, \mathcal{S}$ *be vector spaces and* $A : \mathcal{U} \to \mathcal{R}, B : \mathcal{V} \to \mathcal{S}$ *two linear maps. We define the action of* $A \otimes B : \mathcal{U} \otimes \mathcal{V} \to \mathcal{R} \otimes \mathcal{S}$ *as:*

$$(A \otimes B)(\mathbf{w}) \doteq \sum_{j,k} \alpha_{jk} A(\mathbf{e_j}) \otimes B(\mathbf{f_k}), \text{ for all } \mathbf{w} \in \mathcal{U} \otimes \mathcal{V},$$

where $\mathbf{w} = \sum_{j,k} \alpha_{jk} \mathbf{e_j} \otimes \mathbf{f_k}$ *is any decomposition of* $\mathbf{w}$ *with respect to Hilbert bases* $\{\mathbf{e_j} \mid j = 1 \ldots, n\}$ *and* $\{\mathbf{f_k} \mid k = 1 \ldots, m\}$ *of* $\mathcal{U}$ *and* $\mathcal{V}$ *respectively.*

Notice that this definition is well posed because A and B are linear operators, hence the choice of particular bases is ininfluent. The resulting product operator is, of course, still linear.

A tensor product can also be defined between a vector and a linear functional, where by 'functional' we mean an application mapping vectors to scalars. The result is an operator:

Definition 2.1.9 (Tensor product of a vector and a functional)**.** *Let* $\mathcal{U}, \mathcal{V}$ *be vector spaces,* $\mathbf{u} \in \mathcal{U}$ *and* $a : \mathcal{V} \to \mathbb{C}$ *a linear functional. Then we define* $\mathbf{u} \otimes a : \mathcal{V} \to \mathcal{U}$ *as*

$$(\mathbf{u} \otimes a)(\mathbf{v}) \doteq a(\mathbf{v})\, \mathbf{u}, \text{ for all } \mathbf{v} \in \mathcal{V}.$$

Tensor products can be generalized to much more complex objects (*e.g*, arbitrary tensors), but we will not cover these topics here.

Tensor products in Hilbert spaces

All the theory still works when we deal with Hilbert spaces instead of just vector spaces:

Proposition 2.1.10. *If $\mathcal{U}$ and $\mathcal{V}$ are Hilbert spaces with inner products $(.,.)_{\mathcal{U}}$ and $(.,.)_{\mathcal{V}}$ respectively, then $\mathcal{U} \otimes \mathcal{V}$ is a Hilbert space with inner product:*

$$(\mathbf{u}_1 \otimes \mathbf{v}_1, \mathbf{u}_2 \otimes \mathbf{v}_2)_{\mathcal{U}\otimes\mathcal{V}} \doteq (\mathbf{u}_1, \mathbf{u}_2)_{\mathcal{U}} (\mathbf{v}_1, \mathbf{v}_2)_{\mathcal{V}},$$

for every $\mathbf{u}_1, \mathbf{u}_2 \in \mathcal{U}$, and for every $\mathbf{v}_1, \mathbf{v}_2 \in \mathcal{V}$. Moreover, if $\{ \mathbf{e}_1, \ldots, \mathbf{e}_n \}$ is a Hilbert basis for $\mathcal{U}$ and $\{ \mathbf{f}_1, \ldots, \mathbf{f}_m \}$ is a Hilbert basis for $\mathcal{V}$, then the set $\{ \mathbf{e}_j \otimes \mathbf{f}_k \mid j = 1, \ldots, n; k = 1, \ldots, m \}$ is a Hilbert basis for $\mathcal{U} \otimes \mathcal{V}$.

In particular, if $\mathcal{U}$ is an n-dimensional Hilbert space and $\mathcal{V}$ is an m-dimensional Hilbert space, then $\mathcal{U} \otimes \mathcal{V}$ is nm-dimensional.

The tensor product between a vector and a linear functional assumes a particular significance in Hilbert spaces, because vectors are in a one-to-one *isometric* correspondence with linear functionals. One direction of this correspondence is trivial:

Proposition 2.1.11. *If $\mathbf{u} \in \mathcal{H}$, then the application:*

$$(\mathbf{u}, .) : \mathbf{v} \mapsto (\mathbf{u}, \mathbf{v}),$$

is a linear functional for every $\mathbf{v} \in \mathcal{H}$. Moreover, the application $: \mathbf{u} \mapsto (\mathbf{u}, .)$ is an isometry, in the sense that $\sup_{\mathbf{v}\in\mathcal{H}\setminus\{\mathbf{0}\}} \frac{|(\mathbf{u},\mathbf{v})|}{\|\mathbf{v}\|} = \|\mathbf{u}\|$.

But the opposite also holds:

Theorem 2.1.12 (Riesz Representation Theorem)**.** *For every linear functional $a : \mathcal{H} \to \mathbb{C}$, there exists a (unique) $\mathbf{u_a}$ such that:*

$$a(\mathbf{v}) = (\mathbf{u_a}, \mathbf{v}),$$

for every $\mathbf{v} \in \mathcal{H}$. Moreover, the application $: a \mapsto \mathbf{u_a}$ is an isometry.

We will call $(\mathbf{u}, .)$ the *dual* of $\mathbf{u}$ and it will be sometimes denoted as $\mathbf{u}^\dagger$. Note that this notation is consistent, for if we represent $\mathbf{u}$ as a column vector, then its dual is just the complex conjugate of $\mathbf{u}$ written as a row vector. In this way:

$$\mathbf{u}^\dagger(\mathbf{u}) = (\mathbf{u}, \mathbf{u}) = \mathbf{u}^\dagger \mathbf{u} \in \mathbb{C},$$

while:

Definition 2.1.13 (External product). *If $\mathbf{u}, \mathbf{v} \in \mathcal{H}$, we define the* external product *between $\mathbf{u}$ and $\mathbf{v}^\dagger$:*

$$\mathbf{u} \otimes \mathbf{v}^\dagger : \mathbf{w} \mapsto (\mathbf{v}, \mathbf{w})\, \mathbf{u} \in \mathcal{H},$$

for every $\mathbf{w} \in \mathcal{H}$. Note that the external product of a vector and the dual of another vector is an operator, analogously to the tensor product between a vector and a linear functional in a generic vector space.

Kronecker products

The *Kronecker product* of two matrices A and B is an operation resulting in the matrix of $A \otimes B$, when A and B are seen as operators with respect to the canonical choice of basis.

Definition 2.1.14 (Kronecker product of two matrices). *If A is an $n_A \times m_A$ matrix and B is an $n_B \times m_B$ matrix, then we define $A \otimes B$ as the $(n_A n_B) \times (m_A m_B)$ matrix of the corresponding product operator.*

A Kronecker product of two matrices A and B can be constructed by blocks as follows: if $A = [a_{j,k}]_{j,k}$, then:

$$A \otimes B = \left(\begin{array}{c|c|c} a_{1,1}B & \dots & a_{1,m_A}B \\ \hline \vdots & \ddots & \vdots \\ \hline a_{n_A,1}B & \dots & a_{n_A,m_A}B \end{array}\right).$$

A vector can then be seen as a column matrix, so that:

$$\mathbf{u} \otimes \mathbf{v} = \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} \otimes \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix} = \begin{pmatrix} \underline{u_1\mathbf{v}} \\ \vdots \\ \underline{u_n\mathbf{v}} \end{pmatrix} = \begin{pmatrix} u_1v_1 \\ \vdots \\ u_1v_m \\ \vdots \\ \vdots \\ u_nv_1 \\ \vdots \\ u_nv_m \end{pmatrix}.$$

That is, the Kronecker product between an n-dimensional column vector and an m-dimensional column vector is an nm-dimensional column vector, while:

$$\mathbf{u} \otimes \mathbf{v}^\dagger = \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} \otimes \begin{pmatrix} \overline{v_1} & \dots & \overline{v_m} \end{pmatrix} = \begin{pmatrix} u_1\overline{v_1} & \dots & u_1\overline{v_m} \\ \vdots & \ddots & \vdots \\ u_n\overline{v_1} & \dots & u_n\overline{v_m} \end{pmatrix}.$$

That is, the Kronecker product between an n-dimensional column vector and an m-dimensional row vector is an $n \times m$ matrix representing the external product between the two vectors with respect to the standard basis.

Definition 2.1.15 (Kronecker sum)**.** *We define the* Kronecker sum *of two square matrices A $(n \times n)$ and B $(m \times m)$ as:*

$$A \oplus B \doteq (A \otimes I_m) + (I_n \otimes B),$$

where I_k is the k-dimensional identity matrix.

Notice that the Kronecker sum is generally not equivalent to the vectorial direct sum.

Definition 2.1.16 (Matrix exponentiation)**.** *If A is a square matrix, we define:*

$$e^A \doteq \sum_{k=0}^{\infty} \frac{A^k}{k!}$$

It can be proven that this series converges for every $n \times n$ matrix A, and the limit is another $n \times n$ matrix.

Proposition 2.1.17 (Properties of the matrix exponential). *If A and B are square matrices, then:*

1. $e^O = I$;
2. $e^{A+B} = e^A e^B$ *if and only if* $AB = BA$;
3. $e^{A^T} = \left(e^A\right)^T$;
4. $e^{A^\dagger} = \left(e^A\right)^\dagger$;
5. *if B is invertible, then* $e^{BAB^{-1}} = Be^A B^{-1}$;
6. $det\left(e^A\right) = e^{tr(A)}$,

where $tr(A)$ is the trace *of A,* i.e., *the sum of its diagonal elements.*

We are now ready to state the properties of the Kronecker product for matrices A, B, C, D (omitting their dimensions) and scalar λ:

Proposition 2.1.18 (Properties of the Kronecker product).

1. $A \otimes B \otimes C = A \otimes (B \otimes C) = (A \otimes B) \otimes C$;
2. $A \otimes (B + C) = A \otimes B + A \otimes C$;
3. $(A + B) \otimes C = A \otimes C + B \otimes C$;
4. $\lambda (A \otimes B) = (\lambda A) \otimes B = A \otimes (\lambda B)$, *for every* $\lambda \in \mathbb{C}$;
5. $(A \otimes B)(C \otimes D) = AC \otimes BD$;
6. $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$;
7. $(A \otimes B)^T = A^T \otimes B^T$;
8. $(A \otimes B)^\dagger = A^\dagger \otimes B^\dagger$;
9. $e^{A \oplus B} = e^A \otimes e^B$.

We will just say 'tensor product' instead of 'Kronecker product' when the context makes clear that the choice of the basis is the standard one (which will always happen, unless otherwise stated).

2.1.3 Bra-Ket notation

A widely used notation in QM is the *bra-ket notation*, which is as follows.

If $\mathbf{u}$ is a vector of a Hilbert space $\mathcal{H}$, then it will be written as $|u\rangle$, while $\langle u|$ will be its dual, that is, the linear application $\mathbf{u}^\dagger$. In this way the inner product between $|u\rangle$ and $|v\rangle$ can be written as $\langle u|v\rangle$, so that $\|u\|^2 = \langle u|u\rangle$, while the properties of the inner product can be rewritten as:

$$\langle u|v\rangle = \overline{\langle v|u\rangle}, \text{ for every } \mathbf{u}, \mathbf{v} \in \mathcal{H};$$

$$\langle u|u\rangle \geq 0, \text{ for every } \mathbf{u} \in \mathcal{H};$$

$$\langle u|u\rangle = 0 \text{ if and only if } \mathbf{u} = \mathbf{0}.$$

Here we are using u and v just like labels to denote different bras or kets: we will usually denote these labels by lowercase Greek letters ($\phi, \psi, \ldots$). It is then possible to *sum kets* and the result is another ket:

$$|\phi\rangle + |\psi\rangle = |\xi\rangle\,;$$

or to *multiply kets by scalars*, either by left or right, yielding another ket:

$$\lambda\,|\psi\rangle = |\psi\rangle\,\lambda.$$

The same can be done with *bras*, with the usual sum and product by scalars of linear applications.

If $A : \mathcal{H} \to \mathcal{I}$ is an operator between Hilbert spaces $\mathcal{H}$ and $\mathcal{I}$, it acts from left on a ket and yields another ket:

$$A\,(|\psi\rangle) = A\,|\psi\rangle = |\phi\rangle\,;$$

but it can also act from the right on a bra, yielding another bra:

$$(\langle\psi|)\,A = \langle\psi|\,A = \langle\phi|\,.$$

An operator can also be composed *between* a ket and a bra:

$$\langle\psi|A|\phi\rangle = (\langle\psi| \circ A)(|\phi\rangle) = \langle\psi|(A|\phi\rangle).$$

There is always the dual correspondence:

$$(\langle\psi| A)^\dagger = A^\dagger |\psi\rangle,$$

that is,

$$\langle\psi|A|\phi\rangle = \overline{\langle\phi|A^\dagger|\psi\rangle},$$

for every $|\psi\rangle, |\phi\rangle \in \mathcal{H}$, while the notations $A\langle\psi|$ or $|\psi\rangle A$ are meaningless and we will regard them as forbidden. In general, the *associativity axiom* for bra and ket multiplication allows us to always apply the associative property, as long as the operation makes sense, and to ignore the operation symbol whenever there is no ambiguity. In this way, we can write the external product between a bra and a ket as:

$$|\psi\rangle \otimes |\phi\rangle^\dagger = |\psi\rangle \otimes \langle\phi| = |\psi\rangle\langle\phi| = (|\phi\rangle\langle\psi|)^\dagger,$$

which is an operator. We can even merge the labels in a single bra or ket when performing tensor products, since this adds no ambiguity:

$$|\psi\rangle \otimes |\phi\rangle = |\psi\rangle|\phi\rangle = |\psi\phi\rangle.$$

We will find the following theorem useful in many circumstances:

Theorem 2.1.19 (Resolution of the identity). *Let $|\gamma_1\rangle, \ldots, |\gamma_n\rangle$ be a Hilbert basis for $\mathcal{H}$. Then:*

$$\sum_{k=1}^{n} |\gamma_k\rangle\langle\gamma_k| = I.$$

Proof. Let $|\psi\rangle \in \mathcal{H}$. It can be decomposed as:

$$|\psi\rangle = \sum_{k=1}^{n} \langle\gamma_k|\psi\rangle |\gamma_k\rangle = \sum_{k=1}^{n} |\gamma_k\rangle\langle\gamma_k|\psi\rangle.$$

We can then apply the associative property to the last member:

$$|\psi\rangle = \left(\sum_{k=1}^{n} |\gamma_k\rangle\langle\gamma_k|\right)|\psi\rangle \Rightarrow \sum_{k=1}^{n} |\gamma_k\rangle\langle\gamma_k| = I.$$

□

2.1.4 Hermitian and Unitary operators

We give now definitions and basic properties for some particular classes of operators between Hilbert spaces, and their associated matrices.

Hermitian matrices

Definition 2.1.20 (Hermitian operator). *An operator $A : \mathcal{H} \to \mathcal{H}$ is said to be* Hermitian *or* self-adjoint *if and only if $A = A^\dagger$.*

We say that a matrix is *Hermitian* if it represents a Hermitian operator.

Notice that:

- the elements on the diagonal of a Hermitian matrix must be real numbers;
- the sum of two Hermitian matrices is a Hermitian matrix;
- the inverse of an invertible Hermitian matrix is Hermitian;
- the product of two Hermitian matrices is Hermitian if and only if they commute;
- the Hermitian $n \times n$ matrices form a vector space over the real numbers, but not over the complexes (because multiplication by i does not yield a Hermitian matrix).

We can use Hermitian operators to generate Hilbert bases:

Theorem 2.1.21 (Spectral decomposition for Hermitian operators). *Let $A = A^\dagger : \mathcal{H} \to \mathcal{H}$ be a Hermitian operator. Then:*

1. *A's eigenvalues are reals;*
2. *A's eigenvectors form an orthogonal set;*
3. *A's eigenvectors span $\mathcal{H}$.*

As a corollary we obtain a Hilbert basis for $\mathcal{H}$ by normalizing the eigenvectors of a Hermitian operator:

Corollary 2.1.22. *If A is Hermitian, then there exists a Hilbert basis for $\mathcal{H}$ composed of eigenvectors of A, with real associated eigenvalues.*

If $\mathbf{v}$ is an eigenvector for A with associated eigenvalue α, *i.e.*:

$$A\mathbf{v} = \alpha\mathbf{v},$$

then we will often write $\mathbf{v}$ as a ket $|\alpha\rangle$ labeled by the same eigenvalue:

$$A\,|\alpha\rangle = \alpha\,|\alpha\rangle\,.$$

Notice that this notation is only coherent if α is *not degenerate*, that is, there is only one eigenvalue associated to α, but in this work we will have to deal mostly with non-degenerate operators, *i.e.*, without degenerate eigenvalues).

Unitary matrices

Definition 2.1.23 (Unitary operator)**.** *An operator $U : \mathcal{H} \to \mathcal{H}$ is said to be* unitary *if and only if it is invertible and $U^{-1} = U^\dagger$.*

We say that a matrix is *unitary* if it represents a unitary operator.

The $n \times n$ unitary matrices form a (non-Abelian) group with respect to matrix multiplication. This group is called the *unitary group* and is often denoted as $U(n)$. It is a subgroup in the *general linear group* of invertible $n \times n$ matrices, $GL(n)$. Notice also that:

Proposition 2.1.24. *Let U and V be unitary matrices. Then $U \otimes V$ is also unitary.*

We can also define the following:

Definition 2.1.25 (Special unitary matrix)**.** *A matrix U is said to be* special unitary *if and only if it is unitary and $det(U) = 1$.*

Special unitary matrices form a subgroup of $U(n)$ called the *special unitary group*, denoted by $SU(n)$.

The real vector space of Hermitian matrices and $U(n)$ are bound by the theory of *Lie groups*. This is a very vast topic (we recommend [13] for a survey), but we are only interested in the following results:

Theorem 2.1.26 (Generation of $U(n)$)**.** *For every $U \in U(n)$, there exists a (unique) $n \times n$ Hermitian matrix H such that $U = e^{iH}$.*

This leads to the following, by recalling 2.1.17:

Corollary 2.1.27 (Generation of $SU(n)$)**.** *For every $U \in SU(n)$ there exists a (unique) $n \times n$ Hermitian matrix H with $tr(H) = 0$ such that $U = e^{iH}$.*

We say that Hermitian matrices *generate* $U(n)$, while *trace-free* Hermitian matrices *generate* $SU(n)$.

Pauli matrices

There are three very special operators widely used in QM:

Definition 2.1.28 (Pauli matrices)**.** *If $\mathcal{H}$ is a 2-dimensional complex Hilbert space, we define the* Pauli matrices *with respect to the standard basis:*

$$\sigma_x \doteq \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_y \doteq \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma_z \doteq \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

While the above notation is the standard one in most of the literature, for the sake of simplicity we will denote these matrices as X, Y, Z respectively, while, if I is the 2×2 identity matrix, we denote the *Pauli operator X for the j-th space in a n-fold product space* by:

$$X_k \doteq \overbrace{I \otimes \ldots \otimes I \otimes \underbrace{X}_{k^{th}\ position} \otimes I \otimes \ldots \otimes I}^{n},$$

and analogously for Y_k and Z_k.

Notice that Pauli matrices are Hermitian and trace-free. Moreover:

Theorem 2.1.29 (Decomposition with Pauli matrices). *Every trace-free Hermitian matrix A can be written as a linear combination:*

$$A = \alpha X + \beta Y + \gamma Z,$$

with $\alpha, \beta, \gamma \in \mathbb{R}$.

Proposition 2.1.30. *Each of the Pauli matrices has eigenvalues $+1$ and -1. The corresponding eigenvectors will be denoted as $|+_\sigma\rangle$ and $|-_\sigma\rangle$ respectively, with $\sigma \in \{ X, Y, Z \}$.*

So, *e.g.*, $X |+_X\rangle = |+_X\rangle$, while $Z |-_Z\rangle = - |-_Z\rangle$.

The following will be also useful in the next chapters:

Definition 2.1.31 (Levi-Civita symbol for Pauli matrices).

$$\epsilon_{ijk} = \begin{cases} 1, \text{ if } (i,j,k) = (X,Y,Z), (Y,Z,X) \text{ or } (Z,X,Y) \\ -1, \text{ if } (i,j,k) = (Z,Y,X), (Y,X,Z) \text{ or } (X,Z,Y) \\ 0, \text{ otherwise.} \end{cases}$$

In other words, ϵ_{ijk} is 1 if and olny if (i,j,k) is an even permutation of (X,Y,Z), -1 if and only if it is an odd permutation and 0 if any of the subscripts is repeated.

Two very useful operators when dealing with linear operators are:

Definition 2.1.32 (Commutator and anticommutator). *If A and B are two $n \times n$ matrices, we define:*

- *the* commutator of A and B: $[A, B] \doteq AB - BA$;
- *the* anticommutator of A and B: $\{A, B\} \doteq AB + BA$.

Notice that $[A, B] = O$ if and only if A and B commute, while $\{A, B\} = O$ if and only if A and B anticommute, that is, $AB = -BA$.

Proposition 2.1.33 (Properties of Pauli matrices). *Let (A, B, C) be a permutation of (X, Y, Z). Then:*

1. $[A, B] = 2i\epsilon_{ABC}C$

2. $\{A, B\} = 2i\delta_{AB}I$

moreover:

3 $X^2 = Y^2 = Z^2 = I$.

Notice in particular that X, Y, Z are unitary.

2.2 Quantum Mechanics

Quantum Mechanics (QM) is a branch of physics born at the very end of the 19th century to deal with unexpected experimental results discovered in the previous years. Such experiments involved the study of light or, more in general, electromagnetic radiation, or other microscopic systems. Classical physics routinely failed to explain the outcomes of these experiments, two problems usually arising in the investigation of such phenomena:

- physical items supposed to behave like particles show instead some behaviour typical of waves, a classic experiment being the interference between beams of electrons;
- some physical observables (that is, physical properties of a system which can be measured through an experiment), despite being tied to continuous-valued quantities like energy or momentum, seem to be quantized for many physical processes, typical experiments being the measurement of the black-body radiation or the photoelectric effect.

QM tries to address these issues by introducing the concept of wave function: to every observable of a physical system is associated a function satisfying the Schroedinger wave equation and representing the probability amplitude of obtaining a certain value for that observable through an observation.

We are not interested in the general physical theory though, but only in its mathematical formulation.

2.2.1 Postulates of Quantum Mechanics

QM is a probabilistic physical theory based on an axiomatic mathematical formulation. Many different formulations can be given, all leading to the same resulting theory. Among these formulations, one of the most used in in quantum computing is the Von Neumann axiomatization, which is given by four postulates [26].

Postulate 1

Every physical system is associated to a topologically separable complex Hilbert space. Different states of the system are in a one-to-one correspondence with equivalence classes of normalized vectors.

Separability is a mathematically convenient hypothesis, with the physical interpretation that countably many observations are enough to uniquely determine the state, but in the finite-dimensional case we can ignore this request. The dimension of the space depends on the physical system we are considering: it can be as small as 2 for very simple systems, or it can be much larger, or even infinite.

System states can be identified by equivalence classes of vectors of length 1, where two normalized vectors represent the same state if and only if they differ only by a *phase factor* (complex number of norm 1). In other words, system states are normalized representatives of points in the projective space defined by the Hilbert space.

Postulate 2

The Hilbert space associated to a composite system is the tensor product of the Hilbert spaces associated to the separate subsystems.

So, if $|\psi\rangle$ is the state of the system represented by the space $\mathcal{H}$, and $|\phi\rangle$ is the state of the system represented by the space $\mathcal{I}$, the state of the composite system will be $|\psi\phi\rangle$.

Postulate 3

Every observable of the system is represented by a Hermitian operator. A measurement on a state 'collapses' it onto an eigenstate of that operator (with amplitude of probability equal to the inner product between the state and the eigenstate), and produces the relative eigenvalue as an outcome.

An *eigenstate* of an observable is a normalized eigenvector. This postulate states that the process of measurement of an observable is probabilistic, meaning that, after a measurement, an arbitrary state becomes one of the eigenstates of the observable taken into account (yielding the correspondent eigenvalue as an outcome of the measurement); the probability of collapsing into a particular eigenstate or another depends on how much 'related' the initial state is to that eigenstate. That is, the outcome itself of the measurement is in general probabilistic. This is very counterintuitive, but it can be proven that the expected value of the measurement for macroscopical objects is very close to the classical expectations.

Postulate 4

Every simmetry of a physical system is represented by a unitary operator.

With respect to our needs, a *simmetry* of a physical system is to be intended as a *reversible evolution* of the system, where by 'evolution' we mean a change of state. So, for example, the translation of a system that is in a certain state is a unitary operator applied on that state, yielding another state.

2.2.2 Qubits

A *qubit* is the fundamental measure unit of quantum information. It is a possible state of a 2-dimensional complex Hilbert space (notice, in fact, that in a 1-dimensional Hilbert space there is only one physical state because of Postulate 1, hence no information can be embedded in such a state).

Just like classical bits, qubits can assume 0 or 1 values with respect to a computational basis of the space and, just like classical bits, it is possible to build logic gates to perform operations. But unlike classical bits, which can be either just 0 or just 1, a qubit can be in a superposition of the two basis states. This has interesting computational consequences.

Definition 2.2.1 (Computational basis). *Let A be an observable. We call* computational basis with respect to A *a basis of $\mathcal{H}$ made of eigenstates of A.*

Notice that this definition is well-posed for Corollary 2.1.22.

In the 2-dimensional case, recall that the Pauli operators are Hermitian, and are hence observables. Physically, they represent the measurement along one of the three coordinate axis of the *spin* (magnetic angular momentum) of a *spin-$\frac{1}{2}$ system* (the simplest quantum system, like an electron, where you can obtain only two possible different outcomes from a measurement).

Definition 2.2.2 (Canonical computational basis). *Let $\mathcal{H}$ be a 2-dimensional Hilbert space. We call* canonical computational basis *the set of the two eigenstates of the observable Z, that is, $\{\ |+_Z\rangle,\ |-_Z\rangle\ \}$.*

This definition generalizes to n-fold products of 2-dimensional Hilbert spaces via tensor products of the computational basis elements in the underlying spaces.

From now on, we will denote the two basis elements $|+_Z\rangle$ and $|-_Z\rangle$ by $|0\rangle$ and $|1\rangle$ respectively. So, *e.g.*, in a 4-dimensional product Hilbert space we have the 2-qubits canonical computational basis:

$$|00\rangle \qquad |01\rangle \qquad |10\rangle \qquad |11\rangle\,,$$

while in a 8-dimensional product space we have the 3-qubits basis:

$$|000\rangle \quad |001\rangle \quad |010\rangle \quad |011\rangle \quad |100\rangle \quad |101\rangle \quad |110\rangle \quad |111\rangle\,.$$

In this way, for example, $Z_1\,|01\rangle = |01\rangle$, while $Z_2\,|01\rangle = -\,|01\rangle$. We will always intend a computational basis to be the canonical one, unless otherwise

stated.

Basis elements have a natural representation in terms of column vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

so that, for example:

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix};$$

$$|101\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix};$$

$$|1\rangle \langle 0| = |1\rangle \otimes \langle 0| = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}.$$

A general state $|\alpha\rangle$ is then a complex linear combination of basis elements. For example, in a 8-dimensional, 3-fold product space we have:

$$|\alpha\rangle = \sum_{h,j,k \in \{0,1\}} \alpha_{hjk} |hjk\rangle,$$

where the α_{hjk}s are complex numbers called *the components of* $|\alpha\rangle$ *with respect to the computational basis*. In this way, if $S : \mathcal{H} \to \mathcal{H}$ is a generic linear operator, we have:

$$S|\alpha\rangle = \sum_{h,j,k=0}^{1} \alpha_{hjk} S |hjk\rangle.$$

We can then represent S as a matrix in terms of basis elements. For

example, the operator:

$$S \doteq \begin{array}{c|c} & \begin{array}{cccc} |00\rangle & |01\rangle & |10\rangle & |11\rangle \end{array} \\ \hline \begin{array}{c} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{array} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

acts as a *qubit swap operator* in a 2^2-dimensional space.

Finally, let us underline that from a theoretical point of view we are not limited to 2^n-dimensional product spaces. We may have, for instance, a computational basis in a 3-dimensional Hilbert space (with respect to a certain Hermitian operator A), and denote basis elements by $|0\rangle, |1\rangle$ and $|2\rangle$. The resulting linear combinations are called *qudits* (*'qu-antum d-ig-its'*). The whole theory works in a similar way, but the computational consequencies of this approach are rather different and will be not investigated here.

2.2.3 Observables and measurements

Let $\mathcal{H}$ be an n-dimensional Hilbert space, and $A = A^\dagger : \mathcal{H} \to \mathcal{H}$ be an observable with real non-degenerate eigenvalues $a_1, \ldots, a_n$ and eigenstates $|a_1\rangle, \ldots, |a_n\rangle$.

For Corollary 2.1.22, the eigenstates of A form a Hilbert basis for $\mathcal{H}$, so that an arbitrary state $|\alpha\rangle$ can be decomposed as:

$$|\alpha\rangle = \sum_{j=1}^{n} \langle a_j|\alpha\rangle \, |a_j\rangle \, .$$

Notice that this decomposition can also be obtained by Theorem 2.1.19.

Postulate 3 of QM tells us that a measurement of A on the state $|\alpha\rangle$ is a probabilistic process after which $|\alpha\rangle$ is collapsed into a certain $|a_j\rangle$ with amplitude of probability $\langle a_j|\alpha\rangle$, that is, with probability $|\langle a_j|\alpha\rangle|^2$, yielding a_j as the outcome of the measurement. The request that the observables must be Hermitian (and have therefore real eigenvalues) reflects the fact that we

expect a real valued outcome – hence with a physical significance – and not a complex one.

By the conservation of total probability, we must also have:

$$\sum_{j=1}^{n} |\langle a_j|\alpha\rangle|^2 = 1,$$

so that, for any state $|\alpha\rangle$:

$$|\langle\alpha|\alpha\rangle|^2 = 1.$$

This is the reason why we take normalized states as representative of physical states.

The process of measurement being probabilistic, we might wonder what is the mean value we can expect as an outcome from a certain experiment.

Proposition 2.2.3 (Expected value of an observable)**.** *The* expected value *of an observable A over a state $|\alpha\rangle$ is $\langle\alpha|A|\alpha\rangle$.*

Proof. By applying Theorem 2.1.19 twice, we have:

$$\langle\alpha|A|\alpha\rangle = \langle\alpha|IAI|\alpha\rangle = \sum_{j=1}^{n}\sum_{k=1}^{n} \langle\alpha|a_j\rangle \langle a_j|A|a_k\rangle \langle a_k|\alpha\rangle =$$

$$= \sum_{j=1}^{n} a_j |\langle a_j|\alpha\rangle|^2 = \sum_{j=1}^{n} p_j a_j,$$

where p_j is the probability of observing value a_j, as from Postulate 3, hence satisfying the classic definition of expected value in probability. □

We will denote just by $\langle A\rangle$ the expected value of A when the state $|\alpha\rangle$ is understood.

Obviously, if $|\alpha\rangle$ is already an eigenstate $|a_k\rangle$, for the orthonormality of the basis we have:

$$|\langle a_j|a_k\rangle|^2 = \delta_{jk},$$

and in particular:

$$\langle a_k|A|a_k\rangle = a_k.$$

This means that the measurement of an observable over an eigenstate of that observable will always produce the correspondent eigenvalue with probability 1, without changing the state, so that consecutive measurements of the same observable will always (quite intuitively) produce the same outcome.

Of course, this is no longer true when if we change the observables: for instance, if we first measure an observable A over a state $|\alpha\rangle$ (obtaining the value a_j and an output state $|a_j\rangle$), and then perform a measurement of another observable B on the obtained state $|a_j\rangle$ (yielding b_h as an outcome and $|b_h\rangle$ as a result state), then, if we perform again a measurement of A over $|b_h\rangle$, we may obtain an outcome $a_k \neq a_j$. This simple algebraic argument explains in very minimal terms one of the key consequences of QM: the *uncertainty principle* which, roughly stated, tells us that 'measurements can disturb the state of a system', so that we generally cannot know the exact values of A and B at the same time for a certain system.

There is a more general form of quantum measurement which is given by the concept of a positive operator valued measurement (for short, POVM). Such a measurement consists of a set of Hermitian operators that form a resolution of the identity. When the operators are in a certain form, we obtain the measurement described here. We will not take into account POVMs in this work however, a good survey can be found in [22].

2.2.4 Hamiltonians

Let us consider now an operator U describing the time evolution of a system. For our concerns, we will only take into account a discrete evolution, that is, $U|\psi\rangle$ is the state we obtain from $|\psi\rangle$ after an elementary discrete amount of time has passed. If we set, *e.g.*, this elementary time to be 1 second, then a state $|\psi\rangle$ after t seconds will be in the state $U^t|\psi\rangle$.

In this work we only take into account reversible processes. That is, it must be always possible, in theory, to know which state $|\phi\rangle$ gave origin to an evoluted state $|\psi\rangle$, provided that we know the exact structure of the evolution operator. This means that we regard time evolution as a simmetry and therefore, by Postulate 4 of QM, U must be represented by a unitary operator. By Theorem 2.1.21, this means that there must exist a Hermitian operator H that generates time evolution: $U = e^{iH}$.

Definition 2.2.4 (Hamiltonian). *Let U be the time evolution operator for a physical system. We call* Hamiltonian *of the system the Hermitian operator H such that $U = e^{iH}$.*

Being a Hermitian operator, a Hamiltonian must also be an observable of some physical quantity of the system. It can be proven by physical arguments that it is indeed the observable of the energy of the system, its eigenvalues being the possible *energy levels* that the system can assume. The minimum of those eigenvalues is called *ground level* of the system, and the associated eigenstates are called *ground states*. The difference between the ground level and the first (lower) energy level different from the ground level is called *gap* of the Hamiltonian.

Chapter 3

Quantum Computing

In this chapter we will present a basic introduction to Quantum Computing.

In Section 1 we will give the elementary notions of classical computation theory, by investigating the Turing Machine model and, in particular, the circuit model; we will also introduce the concept of reversible computing, explaining the reasons for the definition of such a model.

In Section 2 we will define the quantum circuit model and explain how a quantum circuit works.

3.1 Classical computation

In this section we will focus on the basics of classical computation theory by presenting two different – but equivalent – approaches to define a model for the computation of a function: the Turing Machine approach and the circuital approach. We will mostly focus on the latter, being more practical and more easily applicable to the quantum model of computation.

An *algorithm* is a procedure, expressed in a finite list of elementary instructions, to perform a computation. By *performing a computation* we mean computing the value of a function f. We will restrict to the case of *binary Boolean functions*, that is, from the set $\{0,1\}^n$ (where $n \in \mathbb{N}$ to the set $\{0,1\}$. If $b \in \{0,1\}^n$, we call b a *binary string* and n the *length of* b. We will assume that each elementary instrucion takes a finite unspecified amount

of time to be executed. Notice that even if an algorithm is expressed by a finite number of elementary instructions, some of these instructions may be repeated more times. An algorithm is *finite* if it terminates after a finite amount of time for every possible input.

Definition 3.1.1 (Computable function). *A function* $f : \{0,1\}^n \to \{0,1\}$ *is* computable *if and only if there exists a finite algorithm which computes* f.

A *model of computation* is an abstract object capable of performing an algorithm. A function $f : \{0,1\}^n \to \{0,1\}$ is said to be *computable for a certain computational model* $\mathcal{M}$ if it is computable according to Definition 3.1.1 and $\mathcal{M}$ is capable of performing the algorithm computing f.

Many classical computational models have been proposed, but they have been proven to be equivalent, in the sense that they compute the same class of functions. We will only take into account two of the most powerful models known: Turing machines and circuits.

3.1.1 Turing Machines

A *Turing Machine* is a theoretical computational model. In its simplest form it consists of four components:

- a *finite state controller* s, *i.e.*, a single variable that can assume a value in a finite set of elements $\mathcal{S} = \{\sigma_1, \ldots, \sigma_k, \sigma_b, \sigma_h\}$, where σ_b and σ_h are special elements called *beginning state* and *halt state*;
- a *tape* X, *i.e.*, a sequence of symbols $(x_j)_{j \in \mathbb{N}}$ of a finite alphabet $\mathcal{A}$;
- a *program* $\mathcal{P}$, *i.e.*, a finite set $\{r_1, \ldots, r_q\}$ of *rules* of the form (y_1, s_1, y_2, s_2, d), where $y_1, y_2 \in \mathcal{A}, s_1, s_2 \in \mathcal{S}, d \in \{0, +1, -1\}$;
- a *tape-head* t, *i.e.*, a single variable that can assume a value in $\mathbb{N}$.

The tape-head t points to the position of the tape that is currently readable or writable (that is, x_t) and starts in the value 1, while s starts in the value σ_b. The computation proceeds step-by-step, in the following way:

- if $x_t = y_1, s = s_1$ and there is a rule of the form $(y_1, s_1, y_2, s_2, d) \in \mathcal{P}$, then the machine changes the internal state s to s_2, changes tape element value x_t to y_2 and changes tape-head state t to $t + d$ (with the constraint that $t + d$ must always be greater than zero)..
- otherwise, it sets s to σ_h and terminates.

A Turing Machine is an elementary model of computation, but it can compute many binary functions, *e.g.* by encoding the argument as a binary string in the initial configuration of the tape and the output 0 or 1 as the tape symbol pointed by the tape-head, x_t, when the program halts. In this case we will say that the Turing machine *implements* the algorithm computing that function.

A Turing Machine T can also compute a function describing the behaviour of another Turing Machine S by encoding the description of S itself as a binary string Y. Then, if S computes the function $f(X)$, T can compute $f(X)$ too by setting the initial configuration of its tape to the concatenation of X and Y. Such a Turing Machine can hence simulate the behaviour of any other Turing Machine over any possible input state. We will call this machine T an *Universal Turing Machine (UTM)*.

Despite their simplicity, Turing Machines are a very powerful computational model: the *Church-Turing Thesis* states that every computable function (according to Definition 3.1.1) is indeed computable by a Turing Machine, and *viceversa*.

However, not every binary function is computable. The most famous example is the function computing the *halting problem*:

Theorem 3.1.2. *Given a Turing Machine T (described as a binary string) and a binary input X, let the function f be defined as:*

$$f(T, X) = \begin{cases} 1, \textit{if } T \textit{ eventually halts on input } X, \\ 0, \textit{otherwise.} \end{cases}$$

Then f is incomputable.

This means that there is no way to know in advance if an arbitrary Turing machine will halt or not on a given input: the only way to discover it is to run the machine on X and wait for an output.

3.1.2 Circuits

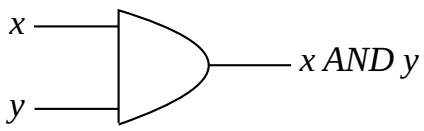
Another model for computation is the circuit model. It can be proven to be equivalent to the Turing Machine model in computational terms, but it is more practical for our investigation. We will restrict to *Boolean circuits*, *i.e.*, acting on *bits* (elements of the set $\{0,1\}$) and computing Boolean functions.

A *circuit* is made of four elements:

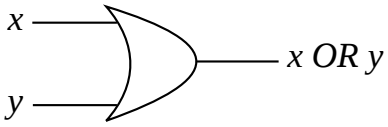
- an *input register*, *i.e.*, an n-dimensional array of bits representing the input binary string;
- *gates*, *i.e.*, functions of the form $g : \{0,1\}^j \to \{0,1\}^k$ from j input bits to k output bits;
- *wires*, *i.e.*, connections between gates;
- an *output register*, *i.e.*, an m-dimensional array of bits representing the output binary string.

Wires form paths from the input register to the output register by connecting gates in an acyclic directed graph. Every wire brings a single bit of information and circuits perform operations by linking gates with wires. We will often assume that $n = m$ and that the single-bit output of the circuit is encoded in a fixed element of the output register.

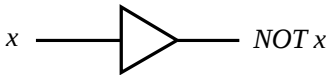
We will represent circuits in a graphical way by considering the direction of the wires from left to right. For example, the following circuit:



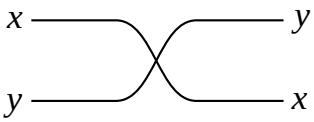
takes a 2-bit register as an input, performs a logical AND operation between bits x and y and produces a single bit as output. We call this gate the AND gate. Other examples are the OR gate:



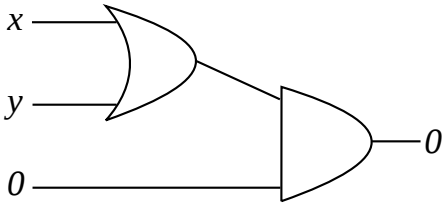
the NOT gate:



and the $SWAP$ gate:



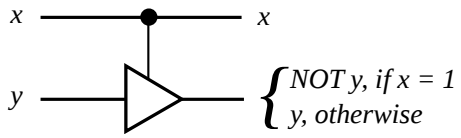
Sometimes we will consider some of the input bits as having fixed values. We will call them *ancilla* bits; they are not to be considered as part of the input, but as constants of the circuit. For example, the following circuit:



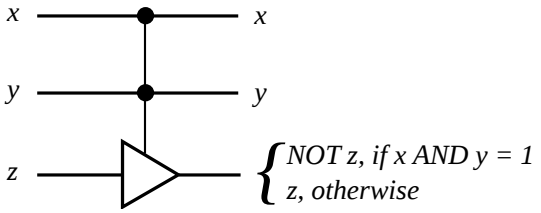
takes a 2-bit value as an input and computes the constant 0 function as a single output bit, because the last AND gate will always yield a 0 output (being the ancilla bit 0 one of its arguments). Ancilla bits can be used as 'extra memory slots', *i.e.*, they can be used to store new information. For example, the following $COPY$ gate:

takes a single bit as an input and an ancilla bit, and outputs two copies of the input bit.

We can also have *controlled gates*, *i.e.*, gates that perform a certain action on *target* bits $x_1, \ldots, x_l$ if and only if the value of another bit, called *control bit*, is 1. For example, the following is a *controlled-NOT (CNOT)* gate:



These gates generalize to *k-multi-controlled gates*, *i.e.*, gates that perform a certain operation if and only if all of the values of k other bits are 1. For example, the following is a *controlled-controlled-NOT (CCNOT)* gate, which is also called a *Toffoli* gate:



We will denote a generic gate G controlled by k bits as C^kG.

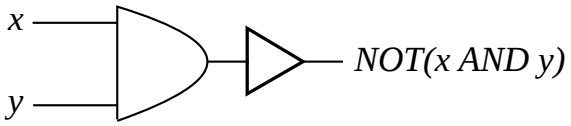
It can be proven that by linking together different gates it is possible to compute every computable function, *i.e.*, circuits are equivalent to Turing Machines.

Universal sets

A finite set of circuit gates $\{ G_1, \ldots, G_k \}$ is said to be *universal* if every other gate can be expressed as a circuit composed only of a finite number of gates in the set. For instance, the following can be proven:

Proposition 3.1.3. *The set consisting only of AND, OR, NOT and $COPY$ gates is universal.*

Let us take into account the following circuit:



We define a $NAND$ gate as the gate computing the same function as the above circuit. This is a very special gate:

Proposition 3.1.4. *The set consisting only of $NAND$ and $COPY$ gates is universal.*

Proof. It is sufficient to note that, given input bits x and y:

- the gate $NOT\ x$ can be implemented as $1\ NAND\ x$, using an ancilla bit;
- the gate $x\ AND\ y$ can then be implemented as $NOT\ (x\ NAND\ y)$;
- the gate $x\ OR\ y$ can then be implemented by using De Morgan's Law: $x\ OR\ y = NOT((NOT\ x)AND(NOT\ y))$;

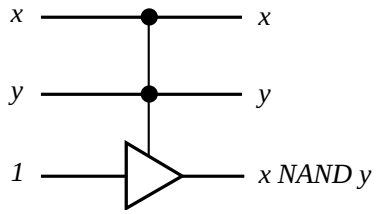
and the thesis follows from Proposition 3.1.3. □

Finally, the Toffoli gate is also universal:

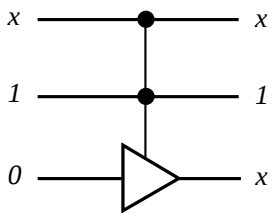
Proposition 3.1.5. *The set consisting only of the $CCNOT$ gate is universal.*

Proof. It is sufficient to note that, given input bits x and y:

- the gate $x\ NAND\ y$ can be implemented by setting x and y as control bits and 1 as an ancilla target bit, in the following way:



- the $COPY$ gate can be implemented by setting x as one of the two control bits, an ancilla bit 1 as the other control bit and an ancilla bit 0 as the target bit, in the following way:

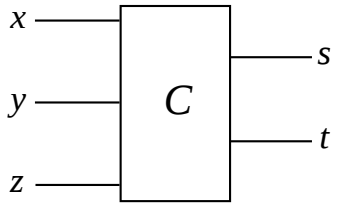


and the thesis follows from Proposition 3.1.5. □

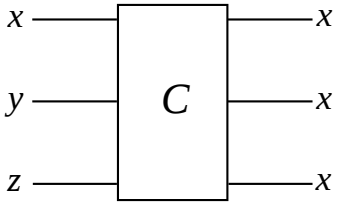
3.1.3 Reversible computing

We say that a computation is *reversible* if it does not erase information. In the case of the circuit model it means that by observing the output register we can always know the state of the input register for any possible (input, output) pair of binary strings. We say that a circuit (or gate) is *reversible* if it performs a reversible computation. For example, the $SWAP$ gate is reversible, while the AND gate is not.

Obviously, a necessary condition for a circuit to be reversible is that the number of output variables must be at least as large as the number of the input variables (not including ancilla bits, that are considered constants of the computation and hence carry no information). The following circuit C, *e.g.*, cannot be reversible if x, y, z are arbitrary bits:



This is not, of course, a sufficient condition. For example, the following circuit is also not reversible:

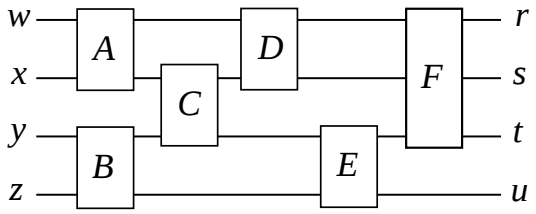


because the values of y and z are lost during the computation.

To achieve reversible computation we restrict our circuit model to the following scenario for a given circuit C:

- the input register of C is composed by both input and ancilla bits used during the computation;
- the output register of C contains a special bit that will assume the single 0 or 1 value of the Boolean function computed by C over the input bits;
- the size of the input and output registers are the same;
- C is composed only of reversible gates.

We will call this model a *reversible circuit model*. We define *width* of a reversible circuit the size of its input and output registers. We define *depth* of a reversible circuit the maximum length (expressed in number of wire connections) between an input register element and an output register element. For example, the following circuit:



has width 4 and depth 5, one of the maximal length path being, *e.g.*, $wACEFt$.

It is easy to see that this model is no less powerful than the general circuit model, *i.e.*, reversible circuits are computationally equivalent to non-reversible circuits and Turing Machines.

Proposition 3.1.6. *Let f be a computable binary Boolean function. Then there exists a reversible circuit U computing that function.*

Proof. By the Church-Turing Thesis, if f is computable then there exists a Turing Machine – or equivalently a circuit C – computing f. By Proposition 3.1.5 we can build a circuit U equivalent to C made only of Toffoli gates. But a Toffoli gate is indeed a reversible gate, so U is reversible. □

The interest for reversible computation comes from thermodynamical considerations that go beyond the scope of this work. The important point here is that, if a computation is reversible according to our definition, then it is in theory possible to perform that computation as a reversible process of a physical system, *i.e.*, without spending energy. More formally, the following holds:

Theorem 3.1.7 (Landauer's principle). *A* computer, i.e., *a physical system performing a computation, consumes an amount of energy of at least* $K_B T \log 2$ *every time it erases a single bit of information, where* K_B *is a positive constant with the dimension of an energy over a temperature and* T *is the absolute temperature of the system.*

Seen from the perspective of Quantum Computing, by recalling Postulate 4 of QM, this means that we could be able to model computational operations as unitary operators acting on a quantum system.

3.2 Quantum circuits

We are now ready to introduce the *quantum circuit model* for computation, which is an extension of the reversible circuit model. It is important to underline that this model is computationally equivalent to the circuit or Turing Machine model in regard to classes of computable functions. What makes it different – and interesting – will be explained in the next chapter.

As in the classical case, a quantum circuit is made of registers, wires and gates. We prepare a quantum register in an input state encoding the instance of the problem, then we perform a computation via quantum gates acting on the state of the register, and finally we retrieve information by reading the final state of the register.

3.2.1 Quantum registers and wires

The first element of a quantum circuit is a *quantum register*. This is an n-dimensional array of elements, analogous to a classical register, but in this case the elements are qubits. So it is possible for the register to assume as a value a state of a 2^n-dimensional Hilbert space $\mathcal{H}$, *i.e.*, a ket.

As in the classical case, a quantum wire carries information; but, unlike the classical wire, it carries quantum information. That is, a single quantum wire carries a single qubit of information and will be hence called a *qubit line*. When drawing an n-qubits circuit, qubit lines will be numbered from 1 to n starting from the uppermost.

A state of a 1-qubit quantum register can be $|0\rangle$ or $|1\rangle$, just like a single-bit register can assume 0 or 1 values, but it can also be in a *superposition* of $|0\rangle$ and $|1\rangle$, *i.e.*:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,,$$

with $\alpha, \beta \in \mathbb{C}$ and$|\alpha|^2 + |\beta|^2 = 1$. In the same way, an n-dimensional quantum register can assume any unitary superimposed state of the 2^n Hilbert basis ket elements.

Entanglement

One of the simplest possible configurations for a quantum state is the following:

Definition 3.2.1 (Product state). . *We call* $|\psi\rangle \in \mathcal{H}$ *a* product state *if it can be decomposed as:*

$$|\psi\rangle = \bigotimes_{j=1}^{n} |\psi_j\rangle\,,$$

where each $|\psi_j\rangle$ *is an element of a* 2*-dimensional Hilbert space.*

In other words, a product state can be seen as just a (tensor) product of single qubits. This is the simplest configuration possible for a quantum

register, but it is not the only one, since a 2^n-dimensional Hilbert space does not contain just product elements. For example, the state:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

is not a product state of a 2^2-dimensional Hilbert space, because it cannot be written as a product of two single-qubit states. We will call these states *entangled*.

The existence of entangled states tells us that, unlike in the classical case, we cannot see a quantum register as just an n-dimensional array of unbound elementary units. We must instead see a quantum register as a larger, whole quantum state, because individual qubits can be 'tied together', *i.e.*, the state of a qubit can be physically bound to the state of another.

Entanglement is a puzzling properties of certain quantum mechanical states. Namely, entanglement enables stronger than classical correlations between quantum subsystems. Such correlations can not be simulated with shared randomness and appear to be genuinely of a quantum mechanical nature. The idea of entanglement is in the impossibility of describing certain quantum states with a tensor product structure. Applying local unitary matrices, acting on the single subspaces corresponding to the tensor product, an entangled state can not be put in a separable form. This is a form in which all the subsystems evolve independently.

The uses of entanglements are many: it is sufficient to point out that every quantum cryptographic protocol devised so far makes use of entanglement; that striking protocols for communication (like teleportation) are based on entanglement; that quantum algorithms for complex problems and procedures for outperforming classical communication complexity need entanglement as a key ingredient. One of the most interesting uses of entanglement is to prove that classical correlations are stronger than classical ones, therefore providing a testing ground at the foundational level for the differences between quantum and classical physics. Indeed, entanglement allows to improve the efficiency of communication complexity protocols with a provable quantum/classical exponential gap (see, *e.g.*, [5]). The generation

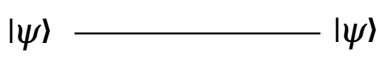
of entangled state in the laboratory is today a challenging task which fuels research in many areas of experimental physics, including cold atoms, linear optics, quantum dots, *etc.*

3.2.2 Single-qubit gates

As in the classical circuit model, a *quantum gate* is an object computing a function. Single-qubit gates are the simplest of these objects: they act on a single input qubit and produce a single output qubit.

Since we restrict ourselves to the reversible computing model, we request these gates to be linear invertible operators. Moreover, since they must also keep unaltered the norm of the ket they act upon, we request them to be unitary. So a *single-qubit gate* is a unitary operator, which we represent as a 2×2 matrix with respect to the computational basis.

The simplest single-qubit gate is the *identity gate* I. It can be represented as the identity matrix I, so that for any arbitrary qubit $|\psi\rangle$ we have $I|\psi\rangle = |\psi\rangle$. That is, it leaves the qubit unchanged, so that we can draw it as just a simple qubit line:



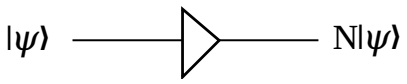
Another simple single-qubit gate is the *NOT gate*. It can be represented as the matrix:

$$N \doteq |0\rangle\langle 1| + |1\rangle\langle 0| = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

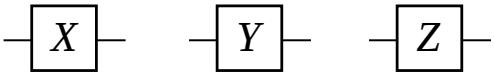
so that:

$$N(\alpha|0\rangle + \beta|1\rangle) = \beta|0\rangle + \alpha|1\rangle,$$

that is, it inverts the components of $|0\rangle$ and $|1\rangle$, analogously to the classical *NOT* gate. We will draw this gate as:



Other three simple gates we have already met are the Pauli X, Y, Z operators:



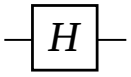
Another useful operator is the *Hadamard operator* H:

$$H \doteq \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

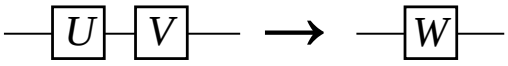
Notice that:

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle;$$
$$H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle;$$

that is, a Hadamard gate evenly 'mixes' the components of a basis element. This will be useful later to produce entangled states. We will draw a Hadamard gate as:

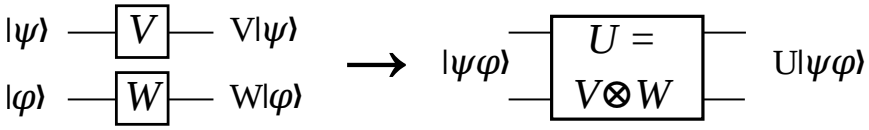


Finally, notice that if we have two adjacent single-qubit gates U and V acting on the same qubit wire, we can replace them with the single gate $W = UV$:



3.2.3 Multi-qubit gates

A *k-qubit gate* is a unitary operator acting on the space spanned by k qubits. The simplest case is when $k = 2$ and we have a 2-qubit operator U which is a tensor product of two 1-qubit operators V and W:



Not all k-qubit gates can be decomposed as tensor products. One such example is the *SWAP operator* between 2 qubits, S:

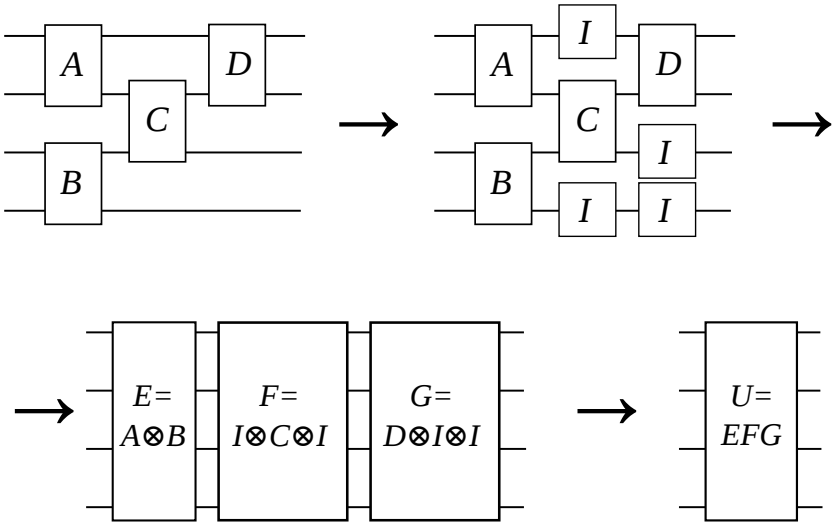
$$S \doteq \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

It is called 'SWAP' gate because:

$$S\left|01\right\rangle = \left|10\right\rangle, \qquad S\left|10\right\rangle = \left|01\right\rangle.$$

If a unitary gate U acts on qubit lines j and $j+2$, we can interpret it as a unitary operator that is a tensor product between two operators: the operator representing U acting on the space spanned by qubits j and $j+2$, and the identity operator I on the space spanned by qubit $j+1$, with the tensor product performed between the relative spaces. The resulting unitary operator acts on lines $j, j+1$ and $j+2$.

In general, recall that we can insert an identity gate I at any place on a qubit line of a quantum circuit without changing the action of the circuit. In this way we can *contract gates*, *i.e.*, we can multiply every unitary gate of an n-qubit circuit to obtain a single n-qubit unitary gate U:



This leads to the following, by recalling Proposition 2.1.24:

Theorem 3.2.2 (Representation of quantum circuits). *Let Q be a n-qubit quantum circuit. Then it can be represented by a unitary operator,* i.e., *there exists a unitary operator $U : \mathcal{H} \to \mathcal{H}$ such that, for every possible state $\left|\psi\right\rangle \in \mathcal{H}$ of the input register, Q computes the output state $U\left|\psi\right\rangle$.*

Controlled gates

Just like in the classical case, it is possible to build *controlled quantum gates* using unitary operators. The first example is a *quantum controlled-NOT* gate $CNOT$:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Another example is the *quantum Toffoli gate* $CCNOT$:

$$CCNOT = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

In general, a k-controlled U gate with control qubit lines $1, \ldots, k$ and target qubit lines $k+1, \ldots, k+l$ can be constructed by blocks:

$$\left(\begin{array}{c|c} I_k & O_{2^k \times 2^l} \\ \hline O_{2^l \times 2^k} & U \end{array}\right),$$

where $O_{n \times m}$ denotes a $n \times m$ matrix with all zero entries.

Hadamard gates and superposition

A special state of an n-qubit register is the following:

Definition 3.2.3 (Uniform superposition)**.** *A ket* $|\psi\rangle$ *represents an* uniform superposition state over n qubits *if:*

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=1}^{2^n} |\gamma_j\rangle,$$

where $\{\ |\gamma_j\rangle \mid j = 1, \ldots, 2^n\ \}$ *is the computational basis.*

Notice that such a state can be written as:

$$|\psi\rangle = \bigotimes_{j=1}^{n} \frac{1}{\sqrt{2}} \left(|0\rangle + |1\rangle\right).$$

Hadamard gates can be used to produce this particular state starting from a $|0 \dots 0\rangle$ state:

Proposition 3.2.4. *Let $H_n = \bigotimes_1^n H$ be the n-fold tensor product of Hadamard operators. Then $H_n |0 \dots 0\rangle$ is an uniform superposition over n qubits.*

The No-cloning Theorem

An important feature of quantum computing is that, unlike in the classical case, it is not possible to build a circuit that copies a qubit:

Theorem 3.2.5 (No-cloning Theorem)**.** *Let $|\psi\rangle$ be an arbitrary qubit. Then there exists no unitary operator U such that $|\psi 0\rangle = |\psi\psi\rangle$.*

Proof. First of all notice that we must use an ancilla qubit to not erase information. Here we use the ancilla qubit $|0\rangle$ but the choice is of course not influential.

Let us assume we have a unitary operator U mapping $|00\rangle$ to $|00\rangle$ and $|10\rangle$ to $|11\rangle$ (which is a minimum requirement if we want it to copy a qubit).

Let $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$. In this way we have:

$$U |\psi 0\rangle = \alpha U |00\rangle + \beta U |10\rangle = \alpha |00\rangle + \beta |11\rangle,$$

because U is linear. On the other hand:

$$|\psi\psi\rangle = (\alpha |0\rangle + \beta |1\rangle) \otimes (\alpha |0\rangle + \beta |1\rangle) = \alpha^2 |00\rangle + \alpha\beta |01\rangle + \beta\alpha |10\rangle + \beta^2 |11\rangle.$$

These two equations can obviously hold together only if $\alpha = 0$ or $\beta = 0$, *i.e.*, the classical case. □

3.2.4 Measurements

We must address a last question to complete the description of the quantum circuit model, that is: how do we extract information from the register to read the output of the computation?

As from the previous description of QM, it is clear that after the computation is performed by a circuit (unitary operator) U on a input state $|\psi\rangle$, information must be extracted from the register – having output state $U|\psi\rangle$ – through a measurement of a particular observable A. Postulate 3 of Quantum Mechanics tells us that this process is probabilistic, *i.e.*, if we perform many times the same computation on the same input state, we will generally obtain different outcomes. The expected value of the measurement will be:

$$\langle\psi|U^\dagger AU|\psi\rangle\,.$$

This means that this model of computation is probabilistic: we must take care in designing quantum algorithms such that the expected value is as close as possible to the correct value for the function we want to compute, for every possible choice of input states.

Since we are concerned with Boolean functions, we can safely assume without loss of generality that the output bit is a measurement of Z_1, that is, the observable Z over the first qubit of the register. Recall that Z has eigenvalues 1 and -1; then a 1 outcome will be interpreted as a 0 output bit and a -1 outcome as a 1 output bit.

3.2.5 Alternative models for Quantum Computing

In the context of computational perspective, the circuit model of quantum computation is somehow the most studied so far because it is defined in close parallel with its classical analogue. However, the mathematical formalism of QM suggests different models which from the theoretical point of view are very rich, and deeply valuable in the perspective of guiding experimental work. It is sufficient to mention measurement based quantum computation

and adiabatic quantum computation. In measurement based quantum computation, we prepare the system in an particular type of initial state and then we drive the computation without applying unitary operators, but by performing appropriate measurements with respect to chosen bases; in adiabatic quantum computation we leave the Hamiltonian of the system act adiabatically and drive the computation towards a particular state encoding the solution of the problem. However, this model has been demonstrated to have issues, because a computation may stop in some undesired configuration.

Chapter 4

Computational Complexity Theory and Efficient Simulation of Quantum Circuits

In this chapter we will investigate the theoretical computational differences between classical and quantum computing. In fact, although the two models are equivalent from the point of view of computability, it is unknown if they are equivalent from the point of view of computational complexity theory.

In Section 1 we give an introduction to computational complexity theory, using Turing Machines as a tool to classify functions into complexity classes. We will explain what an 'efficient computation' is according to this computational model.

In Section 2 we will see that quantum circuits can implement algorithms computing certain functions in an efficient way.

Finally, in Section 3, the possibility of simulating quantum algorithms via classical circuits will be investigated. We will give two possible definitions of 'classical simulation' and we will prove some results about what these two definitions imply.

4.1 Computational complexity theory

The role of computational complexity theory is to classify problems into *complexity classes*. A complexity class is a set of mathematically definible problems which share an inherent degree of difficulty, where 'difficulty' means a cost in terms of necessary resources to find or check a solution for a given instance of a problem.

There is a vast literature on the topic and we cannot afford to go too much deep in details in this work. We will just give basic definitions and results. Most of the omitted proofs can be found in [28] or in [21].

Definition 4.1.1 (Decision problem). *Let $\mathcal{X}, \mathcal{X}_{yes}, \mathcal{X}_{no}$ be sets of binary strings such that:*

1. $\mathcal{X} = \mathcal{X}_{yes} \cup \mathcal{X}_{no}$;
2. $\mathcal{X}_{yes} \cap \mathcal{X}_{no} = \emptyset$;
3. *there exists a computable function $f : \mathcal{X} \to \{0, 1\}$ such that:*

$$f(\xi) = \begin{cases} 1, & \text{if } \xi \in \mathcal{X}_{yes} \\ 0, & \text{otherwise} \end{cases}$$

for every $\xi \in \mathcal{X}$.

We define decision problem *the pair $(\mathcal{X}_{yes}, \mathcal{X}_{no})$. We call an element $\xi \in \mathcal{X}$ an* instance *of the problem, while the length of ξ will be called* difficulty of the instance *and will be denoted as $|\xi|$.*

Notice that we restrict the above definition to decidable problems, but it is also possible to define *undecidable problems*, *i.e.*, by setting f as uncomputable. Notice also that this definition can be rephrased in terms of languages over a binary alphabet: a decision problem is the process of computing if a given string belongs to a language or not.

The process of solving a certain instance of a decision problem, *i.e.*, computing $f(\xi)$, has a *cost* in terms of computational resources. Since we can

model the computational process with Turing Machines, we can define this resources according to the properties of the (most efficient) Turing Machine computing the underlying function f:

Definition 4.1.2 (Computational resources cost for Turing Machines)**.** *The* space cost *(or* memory cost*) of computing $f(\xi)$ is the minimum number of tape slots used among all the Turing Machines computing $f(\xi)$.*

The time cost *of computing $f(\xi)$ is the minimum number of computational steps performed among all the Turing Machines computing $f(\xi)$.*

In most of the cases we do not know in advance which is the best Turing Machine to solve a certain decision problem, so we will just refer to the best *known* Turing Machine. We will mostly focus on time cost in the rest of this work because of practical considerations.

Notice that the computational cost of a particular instance of a decision problem depends on the instance difficulty. We will express this cost in terms of the instance length n taking into account only the *asymptotic behaviour* of the relationship, *i.e.*, if a decision problem has a cost of $3n^2 - n + 5$ for an instance length n, we will just say that it has a cost of $\mathcal{O}(n^2)$.

From now on, we will refer both to the pair $(\mathcal{X}_{yes}, \mathcal{X}_{no})$ or to the function f interchangeably as 'decision problem', since there is no ambiguity.

The definition of computational cost has been given with regard to the properties of the underlying Turing Machines involved. But since we know that other models (*e.g*, circuits) are computationally equivalent to Turing Machines, we might wonder if this equivalence also holds from a computational complexity perspective. The answer is 'polinomially yes':

Proposition 4.1.3. *Let $f : \mathcal{X} \to \{0, 1\}$ be a function computable by a Turing Machine T in space s and time t (where s and t must be intended as functions of the problem instance length). Then there exists a circuit C of width w and depth d, computing f. Moreover, w and d are polynomial functions of s and t respectively.*

The opposite also holds:

Proposition 4.1.4. *Let $f : \mathcal{X} \to \{0, 1\}$ be a function computable by a circuit C of width w and depth d (where w and d must be intended as functions of the problem instance length). Then there exists a Turing Machine T computing f in space s and time t. Moreover, s and t are polynomial functions of w and d respectively.*

In this case we say that T and C are *computationally equivalent*.

By the previous two propositions, it is clear that if we restrict to a definition of resource cost *modulo polinomial functions*, we can give an equivalent definition for circuits:

Definition 4.1.5 (Computational resources cost for circuits). *The* space cost *(or* memory cost*) of a decision problem f is the minimum width among all the circuits computing f (as a function of the difficulty of the instance).*

The time cost *of a decision problem f is the minimum depth among all the circuits computing f (as a function of the difficulty of the instance).*

With this formalism, we are now ready to classify decision problems according to their computational cost.

4.1.1 Classical deterministic complexity classes

As we have just seen, the concept of 'polynomial equivalence' is a basic building block for the definition of computational classes. This leads to the following:

Definition 4.1.6 (complexity class **P**). ***P*** *is the class of all the functions that are computable in polynomial time on a Turing Machine with respect to the size of the input.*

Definition 4.1.7 (complexity class **NP**). ***NP*** *is the class of all the functions for which the correctness of a given result is provable in polynomial time on a Turing Machine with respect to the size of the input.*

We can restrict these definitions to binary Boolean functions (*i.e.*, decision problems) and will say that a decision problem is in **P** or **NP** accordingly.

Roughly speaking, **P** is the class of all the problems 'easy' to solve on a classical computer, while **NP** is the class of all the problems for which it is 'easy' to check, using a traditional computer, whether a given string is actually a solution or not.

Obviously, $\mathbf{P} \subset \mathbf{NP}$. The infamous problem to decide if $\mathbf{P} = \mathbf{NP}$ or not is the most well-known open problem in theoretical computer science and one of the most important problems in mathematics. It is widely believed that these two classes are indeed different, but no proof has been found to date.

NP-completeness and SAT

We can define a partial ordering between decision problems, according to their difficulty:

Definition 4.1.8 (Efficient reduction). *Let* $f : \mathcal{X} \to \{0,1\}, g : \mathcal{Y} \to \{0,1\}$ *be two decision problems. We call a function* $\phi : \mathcal{Y} \to \mathcal{X}$ *an* efficient reduction from f to g *if* $\phi \in \mathbf{P}$ *and* $f(\xi) = g(\phi(\xi))$*, for every* $\xi \in \mathbf{X}$*. In this case we will write* $f \leq g$*.*

The relationship $\leq$ just defined is clearly reflexive and transitive, so it defines a partial ordering. For instance, we can define:

Definition 4.1.9 (**NP**-completeness). *A decision problem* f *is said to be* **NP**-complete *if* $f \in \mathbf{NP}$ *and* $g \leq f$ *for every* $g \in \mathbf{NP}$*.*

Roughly speaking, **NP**-complete problems are the 'hardest' problems in the **NP** category. If we denote as **NPC** this class of problems, we obviously have $\mathbf{NPC} \subset \mathbf{NP}$. The following can be proven (see [18]):

Theorem 4.1.10. *If* $\mathbf{P} \neq \mathbf{NP}$*, then there are infinitely many non-equivalent difficulty classes in* **NP***.*

The typical example of **NPC** problem can be constructed in the following way. We define a Boolean formula ϕ over n Boolean variables to be *satisfiable* if there exists a truth assignment $x \in \{0,1\}^n$ such that $\phi(x) = 1$. A Boolean formula over n variables is always encodable as a binary string of length m, where m is polynomial in n.

Definition 4.1.11 (SAT). *Let $\mathcal{B}$ be the set of all Boolean formulae (encoded as binary strings), and let $f : \mathcal{B} \to \{0,1\}$ such that:*

$$f(\phi) = \begin{cases} 1, & \text{if } \phi \text{ is satisfiable} \\ 0, & \text{otherwise} \end{cases},$$

for every $\phi \in \mathcal{B}$. Then f represents a decision problem, that we will denote as SAT.

SAT is an archetypal example of **NPC** problem (see [8] for a proof of the following theorem):

Theorem 4.1.12 (Cook-Levin). *SAT is in* **NPC**.

In other words, SAT is the problem of deciding if a given Boolean formula can be satisfied or not. This is a very hard problem if n is large, but even for large n it is easy to check if a given a truth assigment satisfies or not a given formula. Since this problem is in **NPC**, every other problem in **NP** can be efficiently (that is, in polynomial time) reduced to an instance of SAT. Hence, finding a algorithm able to solve SAT in polynomial time would imply that $\mathbf{P} = \mathbf{NP} = \mathbf{NPC}$.

4.1.2 Classical non-deterministc complexity classes

A *randomized algorithm* is an algorithm in which the execution of certain instructions occurs only with a certain probability every time the algorithm is run. A *non-deterministic Turing Machine* is an abstract model of computation capable of performing a randomized algorithm by employing *coin tosses*, *i.e.*, aleatory variables that can assume 0 or 1 values with equal probability. To classify decision problems under this model we must take into account not only the cost of the computation, but also its accuracy.

We will denote by $poly(x_1, \ldots, x_k)$ a generic function that grows at most polynomially in $x_1, \ldots, x_k$.

Definition 4.1.13 (complexity class **PP**). *We denote as* **PP** *the class of all the decision problems that can be solved in* $poly(n)$ *time by a non-deterministic Turing Machine for every possible choice of input values of size* n*, in such a way that the output is correct with probability greater than* $\frac{1}{2} + \frac{1}{2^{poly(n)}}$.

Definition 4.1.14 (complexity class **BPP**). *We denote as* **BPP** *the class of all the decision problems that can be solved in* $poly(n)$ *time by a non-deterministic Turing Machine for every possible choice of input values of size* n*, in such a way that the output is correct with probability greater than* $\frac{1}{2} + \epsilon$*, where* ϵ *is a positive, arbitrarily small constant.*

The two definitions above look similar, but there is actually a very deep difference: in **BPP** the success rate of the algorithm must be bound to a constant independent from the input (the **B** in **BPP** stays just for 'Bounded probability'). It is possible to prove that the constant itself does not matter as long as it is greater than zero. In **PP** instead, the constant can depend on the input and become closer and closer to zero as the input size grows (for example it can be $\frac{1}{n}$ for n-bit inputs). So in the latter case it is possible to find a sequence of input values for which the correctness of the algorithm tends to zero: notice in fact that a real 'useless' random algorithm is only the one which gives the correct answer 50% of the times, while an algorithm that gives the wrong answer the 100% of the times would be as useful as one giving always the correct answer, since we are dealing with boolean 'yes-or-no' functions.

Notice also that by the term *correctness of a randomized algorithm* we mean the minimum of the two success probabilities both in the case of a 0 or 1 correct result. In other words: we take into account (as a meaning of 'correctness') both the case where a 0 answer would be expected and the algorithm actually outputs a 0 (*rejection probability*), and the case where a 1 answer would be expected and the algorithm actually outputs a 1 (*acceptance probability*), and stick to the worst case scenario. Consider as an example a hypothetical algorithm to test the primality of integers, running

in polynomial time with respect to the number of digits of the input number. Suppose that this algorithm marks an input number as prime if the number is actually prime with probability 1, while erroneously marks a composite number as prime with probability $\frac{1}{2}$. This algorithm would not even be in **PP** because it would have a correctness of just 50%.

But if the random outcome of this hypothetical algorithm in the case of a composite input integer does not depend on that input (*i.e.*, it is completely non-deterministic), then we could repeat the test twice for a candidate integer. If just one of the two runs of the test is rejected, then we are guaranteed that the number is indeed composite, while if the candidate passes both tests, then we know that it is prime with at least $1 - \frac{1}{2} \cdot \frac{1}{2} = 75\%$ probability. That is, we have a correctness ratio over $75\% = \left(\frac{1}{2} + \frac{1}{4}\right)$, and the algorithm is now in **BPP** (since we have only doubled the necessary running time).

This is indeed exactly what happens in many popular primality test algorithms, such as Fermat's or Rabin-Miller's algorithms (see, *e.g.*, [23]), that is, a low-accuracy *round* is repeated many times to improve the probability of having a correct answer.

Obviously $\mathbf{P} \subset \mathbf{BPP} \subset \mathbf{PP}$. Actually the class **PP** is so huge that it can be proven to contain also **NP** and a many other classes:

Theorem 4.1.15. $\mathbf{NP} \subset \mathbf{PP}$.

Another open question is to understand if adding this property of 'randomness' to our classical algorithms actually improves in a significant manner the power of those algorithms or else it adds nothing new. It is clear that $\mathbf{P} \subset \mathbf{BPP}$ (because a classical Turing Machine can be seen as a non-deterministic Turing Machine without the ability to perform random operations), but it is currently unknown if $\mathbf{BPP} = \mathbf{P}$ or not: this is another very famous open problem in computer science.

Despite the intuitive and empirical evidence of randomized algorithms being capable of solving certain problems much faster than deterministic ones, we must face the following result (see *e.g.* [14]):

Theorem 4.1.16. *Either* $\mathbf{P} \neq \mathbf{NP}$ *or* $\mathbf{P} \neq \mathbf{BPP}$*, but not both.*

So strong is the belief on the first inequality in the academic community that it is indeed widely conjectured that $\mathbf{P} = \mathbf{BPP}$. This means that a suitable computational notion of 'randomness' in algorithm design does probably add nothing important to the efficiency of these, and that is always possible to solve a decision problem in a deterministic way with a time overhead at most polynomial with respect to the time employed by the best non-deterministic algorithm solving that problem.

4.2 Quantum algorithms

A *quantum algorithm* is an algorithm specifically modeled upon a quantum computational model instead than a classical one. If we restrict to the circuit model, for instance, we call an algorithm 'quantum' if it runs on a quantum circuit.

First of all this means that a quantum algorithm is in general probabilistic: since we follow the quantum circuit model, it must operate on a quantum register via quantum gates and produce an output through the probabilistic process of measurement.

Moreover, we are not limited to the traditional computational primitives of the classic models (like AND, OR, NOT, integer addictions, swaps *etc.*), but we can employ every possible operation computed, *e.g.*, by a quantum gate such as Hadamard. Most importantly, we can work over entangled computational states.

4.2.1 Reasons for quantum algorithms

Since we have already claimed (and we will see a detailed proof in the next section) that quantum and classical computing models are equivalent from a computability perspective, we might wonder why to devise algorithms especially built for quantum circuits. The reason is that it is currently unknown if quantum and classical computing are equivalent from a computational complexity perspective.

It is clear that the quantum circuit model is at least as powerful (in terms of computational complexity) as the classical circuit model, because we have already seen that it is possible to build a Toffoli gate as a single quantum gate: therefore the depth of a quantum circuit computing a certain function can be equal to the depth of a classical circuit computing the same function.

It is still not clear if the opposite holds, but in the recent years some problems (which are not traditionally believed to have efficient solutions on a classical computer) have been found to be efficiently solvable on a quantum computer. Given the extreme interest these problems address in real-world situation, a great concern for Quantum Computing has risen.

As it was originally pointed out by Richard Feynman, the simulation of a quantum system with the use of a quantum system seems to be a major application of quantum algorithms. Indeed, the large number of degrees of freedom required in the description of a quantum evolution can only be dealt with in a reasonably efficient way if we approach the simulation problem directly with quantum mechanical resources. This is an important motivation towards the physical implementation of the quantum computer, a machine wich can greatly speed up the simulation of quantum processes. Applications would be fundamental in quantum chemistry, in the study of new materials, and contributing towards the understanding of quantum effects in noisy biological systems.

Let us underline that the problems presently known to be efficiently solvable on a quantum computer, but not believed to be so on a classical one, are scarce. This could either mean that existing problems with this property are actually scarce, that designing efficient quantum algorithms is really difficult (both for their counterintuitive approach and because of the vast amount of research already done on classical algorithms) or that quantum computing is in truth no more powerful than classical computing. Many considerations about this matter can be found in [24]. We will just cite a couple of the most well-known quantum algorithms discovered so far.

Shor's algorithm

The problem of multiplying two large prime numbers is trivial to solve. But the opposite problem, that is, to factorize an integer in its prime factors, is considered to be very hard to solve on a classical computer. A lot of research has been done on the subject, but the best known classical algorithms for integer factorization have a superpolynomial time complexity. Many of today's most secure cryptographic protocols (used in e-commerce, telecommunications, military and so on) rely on the assumption that is not feasible to factorize large integers.

In 1994, Peter Shor found a quantum algorithm able to efficiently factorize integers of n digits in time $\mathcal{O}(n^3)$. Given the extreme importance of factorization, this algorithm is considered a cornerstone in the field of quantum computing, and it started a race to build a physical realization of a quantum computer.

Grover's algorithm

Suppose we have an unsorted array of n integers and we want to know the position of a certain element. It is clear that with a classic algorithm, in the worst case, we have to check at least $n - 1$ positions before being sure of finding the desired element at a certain position.

In 1996, Lov Grover found a quantum algorithm to find a certain element in a quantum register by performing only $\sqrt{n}$ of these queries. Although the speedup is only polynomial, this algorithm can be applied to every **NP** problem, resulting in a practical advantage in many real-world situations.

4.2.2 Quantum complexity classes

We conclude this section by giving definitions of two very important *quantum complexity classes*:

Definition 4.2.1 (complexity class **PQP**). *We define* **PQP** *as the class of all the decision problems that can be solved in poly(n) time by a quantum circuit for every possible choice of input values of size n, in such a way that the output is correct with probability greater than* $\frac{1}{2} + \frac{1}{2^{poly(n)}}$.

Definition 4.2.2 (complexity class **BQP**). *We define* **BQP** *as the class of all the decision problems that can be solved in poly(n) time by a quantum circuit for every possible choice of input values of size n, in such a way that the output is correct with probability greater than $\frac{1}{2} + \epsilon$, where ϵ is a positive, arbitrarily small constant.*

We have already seen that it is possible to implement a classical Toffoli gate as a quantum CCNOT gate; this justifies the following:

Proposition 4.2.3. $\mathbf{BPP} \subset \mathbf{BQP}$, *and* $\mathbf{PP} \subset \mathbf{PQP}$;

but it can also be proven that:

Proposition 4.2.4. $\mathbf{PP} = \mathbf{PQP}$;

(see, *i.e.*, [27] for both proofs), while it is still unknown if $\mathbf{BQP} = \mathbf{BPP}$ or not, but it is considered unlikely.

4.3 Classical simulation of quantum algorithms

The notion of simulatabilty of a quantum circuit has grown in interest in the last years and is looked at as one of the possible ways to better understand the difference between quantum and classical computing. Moreover, it has practical consequences with respect to the simulation of the behaviour of quantum systems, which has direct applications in many fields.

The term 'classical simulation of quantum algorithms' means a classical procedure (that is, executable on a deterministic Turing Machine) that allows us to predict the behaviour of a quantum algorithm without being forced to actually 'build' a quantum computer capable of running that algorithm. This is interesting for two reasons:

- the actual physical realization of a quantum computer is really hard, even for a small number of qubits. To date, the most advanced prototypes of quantum computer work on the order of tenths of qubits;
- a quantum computer is widely believed to be more powerful than a classical one, in the sense that it is believed that $\mathbf{BQP} \neq \mathbf{BPP}$.

Notice that, to date, in all the instances where a speedup between quantum and classical computation is provable, this is at most polynomial (*e.g.*, Grover's algorithm). The only 'hints' we have about the alleged superiority of quantum computation is the existence of problems which are provably solvable in polynomial time on a quantum computer, but are believed to be not on a classical computer. Interestingly enough, such examples are often found among the same few candidates believed to mark the difference between **P** and **NP**, such as factorization and discrete logarithm. But notice that none of these examples has yet been proven to be not in **P**, neither has any **NPC** problem been proven to be in **BQP**. Actually, it is believed that there is no inclusion gerarchy between **NP** and **BQP**.

4.3.1 Strong simulation

There are many different notions of 'classical simulation' for a quantum circuit. We will use the following:

Definition 4.3.1 (Strong simulation). *A quantum circuit C of width n and depth $poly(n)$ is* classically strongly efficiently simulatable *with respect to:*

- *a class of input states Λ,*
- *a class of output observables Ξ,*

if the expected value of the measurement $\langle\lambda| C^\dagger \xi C |\lambda\rangle$ is computable up to accuracy m in $poly(n, m)$ time by a classical deterministic algorithm, for every $\lambda \in \Lambda$ and for every $\xi \in \Xi$.

In this work 'accuracy m' stands for 'up to m digits of accuracy'. Notice that this is a very strong requirement: much less strict definitions could also be given, as 'up to $\log m$ digits of accuracy'. Notice also that we require the classical algorithm to be deterministic. So the above is indeed one of the strongest notions of 'efficient classical simulation' we can achieve. It is so much strong that the following holds:

Theorem 4.3.2. *If it is possible to strongly simulate every quantum circuit, then* $\mathbf{P} = \mathbf{PP}$.

Proof. For each problem in **PQP** there exists a quantum circuit Q computing solutions to that problem, with depth n and accuracy at least $\frac{1}{2} + \frac{1}{2^{poly(n)}}$, for every instance of size n. If Q is strongly simulatable by a classical circuit C, then we can compute the expected value of Q's output over any problem instance. Since the notion of strong simulation involves a precision up to m digits, we can set $m = poly(n)$, and hence be able to always discriminate between a 0 instance and a 1 instance.

We would therefore have found a classical deterministic Turing Machine solving our problem in polynomial time, hence having $\mathbf{P} = \mathbf{PQP}$, and the Theorem follows by Proposition 4.2.4. □

Then we also have from Theorem 4.1.15:

Corollary 4.3.3. *If it is possible to strongly simulate every quantum circuit, then* $\mathbf{P} = \mathbf{NP}$.

This means that this notion of simulation is so much strong that it is reasonable only for particular classes of quantum circuits.

4.3.2 Weak simulation

A weaker meaning of 'simulation' is often taken into account:

Definition 4.3.4 (Weak simulation). *A quantum circuit C of width n and depth $poly(n)$ is* classically weakly efficiently simulatable *with respect to:*

- *a class of input states Λ,*
- *a class of output observables Ξ,*

if there exists a randomized algorithm in **BPP** *(running in $poly(n)$ time) for which the output probability distribution is the same as the probability distribution of the measurement outcomes of ξ on $C|\lambda\rangle$ (up to accuracy m), for every $\xi \in \Xi$ and for every $\lambda \in \Lambda$.*

Here we just ask to be able to *sample* the output of the quantum circuit, up to a certain accuracy. Indeed, if we could weakly efficiently simulate every quantum computation in this way (for every possile input state and observable), this would be enough to prove that $\mathbf{BPP} = \mathbf{BQP}$:

Theorem 4.3.5. *If every quantum circuit of width n and depth $poly(n)$ is efficiently weakly classically simulatable with respect to every possible input state and observable, then* $\mathbf{BQP} = \mathbf{BPP}$.

Proof. It is trivial: if we have a quantum algorithm with bounded success rate and an equivalent non-deterministic algorithm with the same output probability distribution, then the success rate of the deterministic algorithm is also bounded. □

In this work we will deal only with the notion of strong simulation, hence in the following we will just write 'efficient simulation' for 'strong efficient simulation (up to exponential accuracy)'.

4.3.3 Some remarks on simulation

It is important to notice the following fact. If we have a quantum circuit C of width n and depth $poly(n)$, we can represent it as a single unitary matrix (Theorem 3.2.2), while we can represent any possible input state $|\phi\rangle$ as a vector, and any observable A as a Hermitian matrix. So, in theory, we could algebraically compute the expected value of the computation as:

$$\langle\phi|C^\dagger AC|\phi\rangle\,,$$

that is, we could strongly simulate C by performing simple matrix multiplications.

However, this is not feasible because of the size of the matrices involved: for an n-qubit system (circuit width n) the dimension of the Hilbert space is 2^n, hence matrix multiplication is also exponentially hard in n.

In the rest of this work we will focus on a very particular simulation technique, but it is important to clarify that the most part of the research in efficient simulation is done by trying to 'improve' the naive matrix multiplication method mentioned above. There are, in fact, certain states and operators that make possible an efficient direct simulation by performing matrix operations in a certain order: this is called *tensor network contraction*

and it is a vast topic of investigation.

The simplest example is when we have a product state (PS) as an input:

$$|\phi\rangle = |\alpha_1 \dots \alpha_n\rangle \,,$$

a product operator as a circuit:

$$U = U_1 \otimes \dots \otimes U_n,$$

and a product observable as a measurement:

$$A = A_1 \otimes \dots \otimes A_n.$$

We can then apply the naive matrix product as:

$$\langle\phi|C^\dagger AC|\phi\rangle = \prod_{j=1}^{n} \langle\alpha_j|A_j^\dagger U_j A_j|\alpha_j\rangle \,,$$

where each $\langle\alpha_j|A_j^\dagger U_j A_j|\alpha_j\rangle$ is computable in polynomial time since it involves just a 1-qubit subspace (and hence 2×2 matrices).

These are of course very particular states, but they can be generalized to states that have a similarly efficient contraction.

Chapter 5

Fermionic Representation and the Jordan-Wigner Transform

In this chapter we will introduce a different representation of a system of n qubits using Fermionic operators. These are particular operators that define properties of a certain class of quantum physical systems, which are called Fermions. The resulting representation will be useful in the next chapters.

In particle physics, Fermions are objects that follow the Fermi-Dirac statistics. The behaviour of these particles is dictated by the *Pauli exclusion principle* which, roughly stated, tells us that two different particles cannot be in the same state. So if one particle is present in a certain physical state (position, energy, *etc.*), then no other particle occupies the same state. Fermions are looked at with interest in quantum information theory, for they represent a suitable model for quantum computation.

However we are not concerned with the physical interpretation of the Fermionic model itself. We will just focus our attention on the abstract Fermionic operators, keeping in mind that these operators arise from the necessity of describing a mathematical model for a Fermionic quantum system.

In Section 1 we will give a description of Fermions by postulating the existence of a set of operators on $\mathcal{H}$ satisfying certain properties, then we will infer the existence of a computational basis with respect to certain observables built by those operators.

In Section 2 we will introduce Fermionic Hamiltonians and will both give

a complete mathematical characterization and a physical interpretation of these objects.

In Section 3 we will finally present the Jordan-Wigner Transform (JWT), which allows to switch between a Fermionic description of a system and a description based upon Pauli matrices.

5.1 Fermionic representation

We start by defining the following:

Definition 5.1.1 (Fermionic operators). *Let $\mathcal{H}$ be a Hilbert space and $a_1, \dots, a_n : \mathcal{H} \to \mathcal{H}$ a set of non-Hermitian operators. We call the a_js* Fermionic operators *if they satisfy the* canonical anticommutation relations (CARs)*:*

- $\left\{a_j, a_k^\dagger\right\} = \delta_{j,k} I$,
- $\{a_j, a_k\} = \left\{a_j^\dagger, a_k^\dagger\right\} = O$,

for every $j, k = 1, \dots, n$.

This implies in particular that $a_j^2 = \left(a_j^\dagger\right)^2 = O$, for every j. The mere existence of these operators tells us many things on the structure of $\mathcal{H}$, namely:

Proposition 5.1.2. *Let $\mathcal{H}$ be a Hilbert space and $a_1, \dots, a_n$ Fermionic operators on $\mathcal{H}$. Then the following hold:*

1. *the operators $H_j \doteq a_j^\dagger a_j$ are observables (Hermitian operators) with associated eigenvalues 0 and 1 for every $j = 1, \dots, n$;*
2. *the operator a_j acts as a* lowering operator *for H_j, in the sense that if $|\psi\rangle$ is a normalized eigenvector of H_j with associated eigenvalue 1, then $a_j |\psi\rangle$ is a normalized eigenvector of H_j with associated eigenvalue 0;*
3. *similarly, the operator $a_j^\dagger$ acts as a* raising operator *for H_j, in the sense that if $|\psi\rangle$ is a normalized eigenvector of H_j with associated eigenvalue 0, then $a_j^\dagger |\psi\rangle$ is a normalized eigenvector of H_j with associated eigenvalue 1;*

4. *the operators H_j are mutually commuting;*

5. *there exists a state $|vac\rangle \in \mathcal{H}$ which is a simultaneous eigenvector of all H_js, with associated eigenvalue 0 for every j.*

By applying combinations of the raising operators $a_j^\dagger$ on $|vac\rangle$, we obtain a total of 2^n simultaneous eigenstates for all the H_js. We are going to label these eigenstates with $\{|\xi\rangle\}_{\xi\in\{0,1\}^n}$; namely: if $\xi = (x_1, \ldots, x_n)$ with $x_j \in \{0, 1\}$ for every j, then $|\xi\rangle = \left(a_1^\dagger\right)^{x_1} \ldots \left(a_n^\dagger\right)^{x_n} |vac\rangle$. In particular, we denote $|vac\rangle$ as $|0 \ldots 0\rangle$.

This notation may look ambiguous with respect to the similar notation for a computational basis, but the following holds:

Proposition 5.1.3. *Let $\mathcal{V} \subset \mathcal{H}$ be the space spanned by the 2^n simultaneous eigenstates of the H_js. Then $\mathcal{H}$ can be decomposed as $\mathcal{V} \otimes \mathcal{D}$, where $\mathcal{D}$ is a complex Hilbert space with finite dimension d, and $dim(\mathcal{H}) = dim(\mathcal{V})\, dim(\mathcal{D})$.*

This means that the action of the a_js is nontrivial only on $\mathcal{V}$, while $\mathcal{D}$ is left unmodified. We can hence safely ignore $\mathcal{D}$ for our purposes and just set $\mathcal{H} = \mathcal{V}$. This is called the *fundamental representation for the Fermionic CARs*, and the simultaneous eigenstates $|\xi\rangle_{\xi\in\{0,1\}}$ are indeed a computational basis for $\mathcal{H}$ (since the H_js are Hermitian). Proofs of all the above can be found in [20].

5.1.1 Fermions

We can identify qubits with Fermions by interpreting the action of the a_js in the following way:

- if the j-th qubit is set to 0, then $a_j^\dagger$ sets it to 1, while a_j maps it in the **0** state
- if the j-th qubit is set to 1, then a_j sets it to 0, while $a_j^\dagger$ maps it in the **0** state

- the Hermitian operator $H_j = a_j^\dagger a_j$ is an observable with two possible outcomes (0 or 1) on the j-th qubit. This is called *counting operator* (because it 'counts' the presence or not of a Fermion in position j).

Notice that in order to have this interpretation for a n-qubit system, it is necessary to fix an *ordering* among Fermions. Notice also that in this interpretation it is clear that $a_j^\dagger a_j + a_j a_j^\dagger = I$ for every j, as we should expect.

This representation is very useful because Fermionic Hamiltonians usually have a direct physical interpretation. The complete description of a quantum system in terms of its energy levels (Hamiltonian eigenvalues) can lead to a straightforward efficient simulation of the system.

5.1.2 Majorana spinors

Another useful representation for Fermions is the following:

Definition 5.1.4 (Majorana spinors). *Let $a_1, \ldots, a_n$ be Fermionic operators. Then the following $2n$ operators:*

$$c_{2k-1} \doteq a_k + a_k^\dagger \qquad c_{2k} \doteq -i\left(a_k - a_k^\dagger\right), \qquad k = 1, \ldots, n,$$

satisfy the anticommutation relation:

$$\{c_j, c_h\} = 2\delta_{j,h} I,$$

for every $j, h = 1, \ldots, 2n$. We will call $c_1, \ldots, c_{2n}$ Majorana spinors.

These operators are related to the spin (Pauli) operators by mean of a particular transformation that we will see later in this chapter. Spinors will be used in the following chapters to build a representation of a Clifford algebra.

5.2 Fermionic Hamiltonians

A Fermionic Hamiltonian is a linear combination of Fermionic terms, each term being a product of an even number of Fermionic operators (this is a

general rule which holds for every observable, because of conservation rules and the fact that the Hermitianity of the operator must be preserved).

We will try to give a physical interpretation to these Hamiltonians by recalling that they are observables of the energy of the system. We are interested in finding the ground state of Fermionic Hamiltonians in which interactions are *local*, *i.e.*, in our case, only related to nearest-neighbour sites (Fermions or qubit lines). We will take into account just *two-body interactions*, that is, we are assuming that the overall interaction of the qubits in the system can be decomposed as interactions of just pairs of qubits (this is not always necessarily true, but the models for many-body systems are far more complicated and not usually manageable). In this way we are going to have only three kinds of Fermionic terms:

5.2.1 'Counting' terms

These are quadratic terms of the form $a_j^\dagger a_j$, hence they are just counting operators H_j for the j-th Fermion. Their effect, in the Hamiltonian, is to 'select' single qubit values in the ground state. Let us see, *e.g.*, the effect of the Hamiltonian:

$$H = a_1^\dagger a_1$$

in a 2-qubit system. It is trivial to see that the eigenstates of H are of the form:

$$|0?\rangle\,, \qquad |1?\rangle\,,$$

where we do not care about the value of ?. But the difference is that the state $|0?\rangle$ is mapped in the $\mathbf{0}$ state (and hence it has associated eigenvalue 0), while $|1?\rangle$ is mapped to itself (and so it has associated eignevalue 1). Because we are looking for the smallest eigenvalue, it follows that our ground state will be of the form $|0?\rangle$. From this example it is evident that the effect of a counting term $a_j^\dagger a_j$ in the Hamiltonian is 'to select' the value 0 for the j-th qubit in the ground state. Analogously, the adjoint term $a_j a_j^\dagger = -a_j^\dagger a_j$ sets the value 1 for the j-th qubit in the ground state.

5.2.2 'Hopping' terms

These are quadratic terms of the form $a_k^\dagger a_j$, with $j \neq k$. They are called 'hopping' because of the interpretation previously given to the a_js: a_j 'destroys' a Fermion on site j, while $a_k^\dagger$ 'creates' a Fermion on site k. So if this term is applied to a state which represents a Fermion in the site j, it makes that Fermion 'migrate' to site k.

But notice that we will never find the term $a_k^\dagger a_j$ alone in a Hamiltonian, because it is not Hermitian. We will always find the adjoint term $a_j^\dagger a_k$ too, which has the opposite effect: it makes a Fermion 'migrate' from site k to site j. So, keeping in mind the superposition principle, if we have a 2-qubit system with a Hamiltonian like:

$$H = a_1^\dagger a_2 + a_2^\dagger a_1,$$

the overall effect is that the eigenstates will be in a superposition, more precisely they will be of the form:

$$|10\rangle \pm |01\rangle .$$

It is easy to see that in this case the minus sign leads to the minimum eigenvalue, while if we had the Hamiltonian:

$$H = -a_1^\dagger a_2 - a_2^\dagger a_1$$

we would have had the ground state:

$$|10\rangle + |01\rangle .$$

To summarize: the effect of the hopping terms in the ground state is 'to mix' the values of the qubits involved.

5.2.3 'Interaction' terms

These are quartic terms of the form $a_k^\dagger a_k a_j^\dagger a_j$. They are called 'interaction terms' because of the interpretation given to the *counting operator* $a_k^\dagger a_k$: it

'counts' the number of Fermions in the site k, then these quantities are multiplied (representing an 'interaction' between the Fermions involved).

Let us take as an example, for a 2-qubit system, the Hamiltonian:

$$H = a_1^\dagger a_1 a_2^\dagger a_2$$

It has 4 (classes of) eigenstates:

$$|00\rangle \quad , \quad |01\rangle \quad , \quad |10\rangle \quad , \quad |11\rangle \, ;$$

but the first three have associated eigenvalue 0, while the last one has 1. So the ground state will be a linear combination of the form:

$$\alpha_1 \, |00\rangle + \alpha_2 \, |01\rangle + \alpha_3 \, |10\rangle$$

That is: a term of the form $a_k^\dagger a_k a_j^\dagger a_j$ 'excludes' a $(1,1)$ combination in ground state positions k and j. Analogously:

- $a_j^\dagger a_j a_k a_k^\dagger$ excludes a $(1,0)$ combination;
- $a_j a_j^\dagger a_k^\dagger a_k$ excludes a $(0,1)$ combination;
- $a_j a_j^\dagger a_k a_k^\dagger$ excludes a $(0,0)$ combination.

5.2.4 General case

We can have Hamiltonians that include two or more of the above types of terms, which may have opposite actions on the definition of a ground state. In these cases, the overall action will be a superposition of the actions of the singular separate terms.

Let us consider, *e.g.*, the following Hamiltonian for a 2-qubit system:

$$H = c_1 a_1^\dagger a_1 + c_2 a_2^\dagger a_2 + c_3 a_1 a_1^\dagger a_2 a_2^\dagger .$$

According to what has been said before, the first term will 'force' a ground state of the form $|0?\rangle$, while the second will force one of the form $|?0\rangle$. But the last interaction term will exclude a term of the form $|00\rangle$, so we are facing a *minimization problem* where a balance must be found between opposing 'forces'. In these cases the solution will depend analitically on the values of the c_js.

5.2.5 Fermi quadratic Hamiltonians

A *Fermi quadratic Hamiltonian* H on $\mathcal{H}$ is a Fermionic Hamiltonian which is quadratic with respect to the a_js, *i.e.*, it has the form:

$$H = \sum_{j,k} \left(\alpha_{j,k} a_j^\dagger a_k - \bar{\alpha}_{j,k} a_j a_k^\dagger + \beta_{j,k} a_j a_k - \bar{\beta}_{j,k} a_j^\dagger a_k^\dagger \right),$$

where the matrix $A \doteq [\alpha_{j,k}]_{j,k}$ is Hermitian, while $B \doteq [\beta_{j,k}]_{j,k}$ is antisymmetric (this is to ensure that H is indeed Hermitian).

It turns out that these Hamiltonians are very easy to diagonalize (see, *e.g.*, [20]). In order to accomplish this is crucial that the Hamiltonian contains only quadratic terms. These kinds of Hamiltonians are very often found in systems that can be described in terms of interactions between just pairs of elements (mutually interacting Fermions). The interesting case for our purposes is a system composed of a lattice of hopping Fermions interacting only locally. Whatever the dimension of the lattice, we will get Hamiltonians that are going to be composed just by terms of the form $a_j^\dagger a_k$, where j and k label two geometrically adjacent Fermions. Hence, these *local Hamiltonians* are special cases of Fermi quadratic Hamiltonians.

5.3 The Jordan-Wigner Transform

Suppose we have an n-qubit system in the *spin representation*, that is, with Pauli observables $X_j, Y_j, Z_j, j = 1, \ldots, n$, and the related canonical computational basis. The *Jordan-Wigner Transform (JWT)* allows us to switch from this representation to a Fermionic representation in terms of operators a_js. That is: it is possible to find Fermionic operators $a_1, \ldots, a_n$ such that the computational bases in the two representations coincide. This can be done in the following way:

$$a_j \doteq -S_1^{j-1} \sigma_j,$$

where:

$$\sigma_j \doteq \bigotimes_1^{j-1} I \otimes (|0\rangle \langle 1|) \otimes \bigotimes_{j+1}^{n} I,$$

while:

$$S_l^m \doteq \left(\prod_{k=l}^{m} Z_k \right),$$

for $l \leq m$, is the *Z-string operator from site l to site m*. Notice that here we must pay attention to the *ordering* of the Fermions we are choosing. This is called the *Jordan-Wigner Transform (JWT)*.

The JWT can be inverted too, holding:

$$\begin{aligned} Z_j &= a_j a_j^\dagger - a_j^\dagger a_j; \\ X_j &= -S_1^{j-1} \left(a_j + a_j^\dagger \right); \\ Y_j &= i S_1^{j-1} \left(a_j^\dagger - a_j \right). \end{aligned}$$

It is important to note the following fact: it is not generally convenient to express the X_j, Y_j in such a way, because of the large number of a_js involved. But it turns out that, for certain simple products of Pauli operators, the expressions simplify to:

$$\begin{aligned} X_j X_{j+1} &= \left(a_j^\dagger - a_j \right) \left(a_{j+1} + a_{j+1}^\dagger \right); \\ Y_j Y_{j+1} &= -\left(a_j^\dagger + a_j \right) \left(a_{j+1}^\dagger - a_{j+1} \right); \\ X_j Y_{j+1} &= i \left(a_j^\dagger - a_j \right) \left(a_{j+1}^\dagger - a_{j+1} \right); \\ Y_j X_{j+1} &= i \left(a_j^\dagger + a_j \right) \left(a_{j+1}^\dagger + a_{j+1} \right); \end{aligned}$$

and so quadratic terms of the above form map to quadratic terms of Fermionic operators. This follows from the relations:

$$\begin{aligned} \sigma &= \tfrac{1}{2}(X - \imath Y); & \sigma^\dagger &= \tfrac{1}{2}(X + \imath Y); \\ \sigma Z &= -\tfrac{1}{2}\imath(Y + X); & \sigma^\dagger Z &= -\tfrac{1}{2}\imath(Y - X); \\ Z\sigma &= \tfrac{1}{2}\imath(\imath Y - X); & Z\sigma^\dagger &= \tfrac{1}{2}(\imath Y + X); \end{aligned}$$

and by observing that, due to the fact that all Pauli operators square to the identity, string operators S_1^j associated to consecutives indexes j and $j+1$ cancel each other. This also holds for quadratic local Fermionic terms, i.e.:

$$
\begin{aligned}
a_j^\dagger a_{j+1} &= \left(-Z\otimes\ldots\otimes Z\otimes\sigma^\dagger\otimes I\otimes I\ldots\right)\left(-Z\otimes\ldots\otimes Z\otimes Z\otimes\sigma\otimes I\ldots\right)\\
&= \left(I\otimes\ldots\otimes I\otimes\sigma^\dagger Z\otimes\sigma\otimes I\ldots\right)\\
&= \frac{1}{4}\left(iX_jX_{j+1}+X_jY_{j+1}-iY_jX_{j+1}-Y_jY_{j+1}\right).
\end{aligned}
$$

We are then facing the following, crucial observation: if $n=1$ (*e.g.*, monodimensional case), the JWT maps Fermionic terms into Pauli terms and viceversa *keeping locality*.

The problem here is that the meaning of 'locality' itself strictly depends on the geometrical dimension of the problem. There is a large conceptual gap between monodimensional and multidimensional circuit simulation.

5.3.1 Monodimensional case

In this case we have, *e.g.*, a finite *string* of n qubits. An interesting feature here is that geometrically neighbouring qubits are indexed with consecutive js. So if we have a Hamiltonian expressed in terms of Pauli operators, which is *quadratic* (that is, every product term is at most quadratic in the number of operators involved) and *local* (that is, it involves only products of operators related to adjacent qubits), we can translate it to a *quadratic Fermionic Hamiltonian* which is still local with respect to the a_j' involved by using the JWT, and hence it can be efficiently diagonalized.

We will use this approach in the following chapters as the key ingredient to show how it is possible to efficiently simulate quantum circuits composed only by a certain class of locally-acting gates.

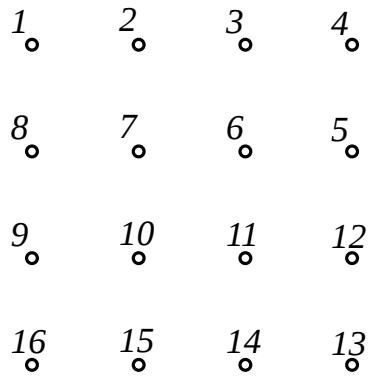
5.3.2 Dimension greater than 1

The difference here is that, despite every possible ordering of the qubits, not every pair of geometrically adjacent qubits will be indexed by consecutive js, so there is usually no chance of ending up with a quadratic Hamiltonian

wich is also local.

A trick to switch from a local quadratic Fermionic Hamiltonian to a local *quartic* spin Hamiltonian in multiple dimensions has been proposed in [7], using 'additional degrees of freedom' in the form of *auxiliary Fermions*, which are coupled to the original ones. The idea is to build a system which can be decomposed as a tensor product of two distinct systems: the original (physical) one and the *auxiliary* one. Then the JWT is applied to the global system in such a way that non-locality terms (string operators) are canceled each other, allowing an efficient description of the system. Finally this (solved) global system is decomposed back into the original subsytems, providing an efficient description for the physical system we were initially interested in.

Let us take as an example the bidimensional case, considering an $n \times n$ square grid, *e.g.*, $n = 4$. Every site of the grid corresponds to a qubit, and we are going to number the positions in the following way:



So, *e.g.*, X_4 would be the Pauli X-operator acting on the top right corner qubit, while Y_2Y_7 is a vertical interaction between nearest-neighbour (n.n.) qubits 2 and 7. In this case, *'local'* can refer either to horizontal pairs of n.n. sites (which are indexed by consecutive js) or to vertical ones (which are usually not indexed in this way, except for pairs $(4, 5)$, $(8, 9)$ and $(12, 13)$.

We consider Hamiltonians with quadratic and quartic local Fermionic

terms, of the form:

$$a_k^\dagger a_j \tag{5.1}$$

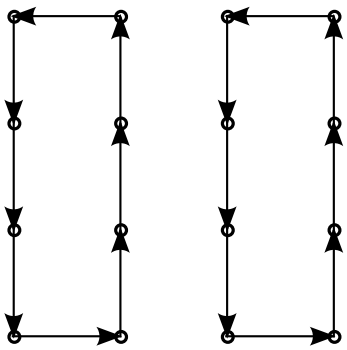
$$a_k^\dagger a_k a_j^\dagger a_j, \tag{5.2}$$

where either $k = j$ or k and j index two geometrically adjacent sites, as usual. The JWT will map operators of the form 5.2 into local quartic spin operators (because string operators from a_k and $a_k^\dagger$ cancel each other), but will map operators of the form 5.1 into quadratic local spin operators only when $j = k \pm 1$, that is all the horizontal hopping terms and very few vertical ones. So there is a problem only for the vertical hopping terms.

The idea is to introduce an additional 'layer of Fermions', in the form of additional Fermionic operators b_ks for each operator a_k, and then changing the Hamiltonian of the system in such a way that string operators from every b_j will cancel out those from the a_js. We impose that the state composed only by these new Fermions is in the ground state of the auxiliary Hamiltonian:

$$H_a \doteq -\sum_{(j,k)\in\Gamma} P_{jk},$$

where $P_{jk} \doteq \left(b_j - b_j^\dagger\right)\left(b_k + b_k^\dagger\right)$, and Γ is the set of directed edges in the following picture:



The following is trivial:

Proposition 5.3.1. *The P_{jk}s commute and $P_{jk}^2 = I$, for every (j,k).*

This implies that the P_{jk}s have eigenvalues ± 1 and they share a common basis of eigenstates, so the gap of H_a is 2 and hence it has a unique ground state, $|\chi\rangle_a$, with the property:

$$P_{jk}\,|\chi\rangle_a = |\chi\rangle_a\,,$$

for every $(j,k) \in \Gamma$.

Next, we define the operators $c_j \doteq \left(b_j + b_j^\dagger\right)$ and $d_j \doteq -i\left(b_j - b_j^\dagger\right)$, so that $P_{jk} = ic_jd_k$, and we consider the system composed of the original Fermions $\{\, a_j \mid j = 1, \ldots, n \,\}$ and the auxiliary ones $\{\, b_k \mid k = 1', \ldots, n' \,\}$ together. Geometrically we assume that each site in the grid is occupied by two Fermions, one repesented with the a_js and one represented with the $b_{k'}$s.

Then we modify the original physical Hamiltonian by altering the vertical hopping terms which appear there in the following way:

$$a_j^\dagger a_k \longmapsto a_j^\dagger a_k P_{j',k'}.$$

We will call H_p this new, modified Hamiltonian, while the total Hamiltonian of the whole system will be $H_t \doteq H_p + H_a$.

Proposition 5.3.2. *The ground state $|\chi\rangle$ of the whole system can be decomposed as a tensor product of the ground states $|\chi\rangle_p$ and $|\chi\rangle_a$ of the two separate subsystems, provided that we multiply H_a for a positive real constant.*

Proof. First note that $H_p = H_{p'} + H_{vert}$, where $H_{p'}$ is composed only by interaction terms and horizontal hopping terms (hence acting only on the physical subsystem), while H_{vert} is made only of terms of the form $a_j^\dagger a_k P_{j',k'}$. So $H_{p'}$ commutes trivially with H_a, while H_{vert} commutes with H_a if and only if every term of the form $a_j^\dagger a_k P_{j',k'}$ commutes with every term in H_a, which is also trivial because the $P_{j',k'}$s commute each other. So:

$$[H_a, H_p] = 0,$$

and this completes the proof since the gap in H_a is greater than zero (being such, it is sufficient to multiply H_a for a large enough factor to achieve a gap larger than the gap of H_p). □

The advantage of this formalism is obtained by reordering the Fermions as $1, 1', 2, 2', \ldots$ when doing the JWT on the whole system. Let us see what happens in such case.

Interaction terms of the physical subsytem

In this case the JWT will map $a_j^\dagger a_j a_k^\dagger a_k$ in (omitting minus signs and assuming $j \geq k$):

$$\left(\overbrace{Z \otimes \ldots \otimes Z}^{2(j-1)} \otimes \sigma^\dagger \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-j)+1}\right)\left(\overbrace{Z \otimes \ldots \otimes Z}^{2(j-1)} \otimes \sigma \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-j)+1}\right)$$
$$\left(\overbrace{Z \otimes \ldots \otimes Z}^{2(k-1)} \otimes \sigma^\dagger \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-k)+1}\right)\left(\overbrace{Z \otimes \ldots \otimes Z}^{2(k-1)} \otimes \sigma \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-k)+1}\right) =$$
$$= \overbrace{I \otimes \ldots \otimes I}^{2(j-1)} \otimes \underbrace{\sigma^\dagger \sigma}_{site\ j} \otimes \overbrace{I \otimes \ldots \otimes I}^{2(j-k)} \otimes \underbrace{\sigma^\dagger \sigma}_{site\ k} \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-k)+1},$$

which is still expressible as a local quadratic term in Pauli operators, as we should expect (because j and k index two adjacent sites).

Horizontal hopping terms of the physical subsystem

In this case the JWT will map $a_j^\dagger a_k$ in (omitting minus signs and assuming $k = j + 1$):

$$\left(\overbrace{Z \otimes \ldots \otimes Z}^{2(j-1)} \otimes \sigma^\dagger \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-j)+1}\right)\left(\overbrace{Z \otimes \ldots \otimes Z}^{2j} \otimes \sigma \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-j)-1}\right) =$$
$$= \overbrace{I \otimes \ldots \otimes I}^{2(j-1)} \otimes \sigma^\dagger \otimes Z \otimes \sigma \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-j)+1},$$

which is, despite the 'cubic' appearance, still expressible as a tensor product (done in respect to the physical and auxiliary system separately) of quadratic local Pauli operators, namely:

$$\left(\overbrace{I \otimes \ldots \otimes I}^{j-1} \otimes \underbrace{\sigma^\dagger \otimes \sigma}_{sites\ j\ and\ j+1} \otimes \overbrace{I \otimes \ldots \otimes I}^{n-j-1}\right) \bigotimes$$

$$\bigotimes \left(\overbrace{I \otimes \ldots \otimes I}^{j-1} \otimes \underbrace{Z}_{site\ j'} \otimes \overbrace{I \otimes \ldots \otimes I}^{n-j} \right).$$

Note that this also holds for half of the vertical hopping terms at the edges of the grid ((4, 5), (8, 9) and (12, 13)).

Vertical hopping terms of the physical subsystem

In this case the JWT will map $a_j^\dagger a_k P_{j',k'}$ (assuming $k > j$ in:

$$\left(\overbrace{Z \otimes \ldots \otimes Z}^{2(j-1)} \otimes \underbrace{\sigma^\dagger}_{site\ j} \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-j)+1} \right) \left(\overbrace{Z \otimes \ldots \otimes Z}^{2(k-1)} \otimes \underbrace{\sigma}_{site\ k} \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-k)+1} \right)$$

$$\left(\overbrace{Z \otimes \ldots \otimes Z}^{2j-1} \otimes \underbrace{\sigma^\dagger + \sigma}_{site\ j'} \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-j)} \right)$$

$$\left(\overbrace{Z \otimes \ldots \otimes Z}^{2k-1} \otimes \underbrace{\sigma^\dagger - \sigma}_{site\ k'} \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-k)} \right) =$$

$$= \left(\overbrace{I \otimes \ldots \otimes I}^{2(j-1)} \otimes \underbrace{\sigma^\dagger Z \otimes \sigma^\dagger + \sigma}_{sites\ j\ and\ j'} \otimes \overbrace{I \otimes \ldots \otimes I}^{2(k-j-1)} \otimes \underbrace{\sigma Z \otimes \sigma^\dagger - \sigma}_{sites\ k\ and\ k'} \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-k)} \right)$$

$$= \left(\overbrace{I \otimes \ldots \otimes I}^{j-1} \otimes \underbrace{\sigma^\dagger Z}_{site\ j} \otimes \overbrace{I \otimes \ldots \otimes I}^{k-j-1} \otimes \underbrace{\sigma Z}_{site\ k} \otimes \overbrace{I \otimes \ldots \otimes I}^{n-k} \right) \bigotimes$$

$$\bigotimes \left(\overbrace{I \otimes \ldots \otimes I}^{j-1} \otimes \underbrace{\sigma^\dagger + \sigma}_{site\ j'} \otimes \overbrace{I \otimes \ldots \otimes I}^{k-j-1} \otimes \underbrace{\sigma^\dagger - \sigma}_{site\ k'} \otimes \overbrace{I \otimes \ldots \otimes I}^{n-k} \right),$$

so these terms are expressible as tensor products of local quadratic Pauli operators too (in their respective subspaces).

Terms of the auxiliary system

In this case the JWT will map terms $P_{j',k'}$ in:

$$\left(\overbrace{I \otimes \ldots \otimes I}^{n} \right) \bigotimes$$

$$\bigotimes \left(\overbrace{Z \otimes \ldots \otimes Z}^{2j-1} \otimes \underbrace{\sigma^\dagger + \sigma}_{site\ j'} \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-j)} \right)$$

$$\left(\overbrace{Z \otimes \ldots \otimes Z}^{2k-1} \otimes \underbrace{\sigma^\dagger - \sigma}_{site\ k'} \otimes \overbrace{I \otimes \ldots \otimes I}^{2(n-k)} \right) =$$

$$= \left(\bigotimes_1^n I \right) \bigotimes \left(\overbrace{I \otimes \ldots \otimes I}^{j-1} \otimes \underbrace{\sigma^\dagger + \sigma}_{site\ j'} \otimes \overbrace{Z \otimes \ldots \otimes Z}^{k-j-1} \otimes \underbrace{\sigma^\dagger - \sigma}_{site\ k'} \otimes \overbrace{I \otimes \ldots \otimes I}^{n-k} \right),$$

hence in this case there is an issue: a string operator appears, which makes the resulting Pauli operator non-local.

To address this problem, we have to modify H_a by applying the following substitution to its terms:

$$P_{j',k'} \longmapsto \begin{cases} P_{j',k'}, & \text{if } j = mn, \text{ for } m \in \mathbb{N}; \\ P_{j',k'} P_{(j+1)',(k-1)'}, & \text{if } (j \bmod n) \leq \lfloor \frac{n}{2} \rfloor; \\ P_{j',k'} P_{(j-1)',(k+1)'}, & \text{if } (j \bmod n) > \lfloor \frac{n}{2} \rfloor. \end{cases}$$

Due to the fact that all the $P_{j',k'}$ commute, this substitution does not affect the ground state and the gap of H_a, but the JWT will map these new terms (*e.g.*, if $j < \lfloor \frac{n}{2} \rfloor$) into:

$$\left(\overbrace{I \otimes \ldots \otimes I}^{n} \right) \bigotimes$$

$$\bigotimes \left(\overbrace{I \otimes \ldots \otimes I}^{j-1} \otimes \underbrace{\sigma^\dagger + \sigma}_{site\ j'} \otimes \overbrace{Z \otimes \ldots \otimes Z}^{k-j-1} \otimes \underbrace{\sigma^\dagger - \sigma}_{site\ k'} \otimes \overbrace{I \otimes \ldots \otimes I}^{n-k} \right)$$

$$\left(\overbrace{I \otimes \ldots \otimes I}^{j} \otimes \underbrace{\sigma^\dagger + \sigma}_{site\ (j+1)'} \otimes \overbrace{Z \otimes \ldots \otimes Z}^{k-j-3} \otimes \underbrace{\sigma^\dagger - \sigma}_{site\ (k-1)'} \otimes \overbrace{I \otimes \ldots \otimes I}^{n-k+1} \right) =$$

$$= \left(\bigotimes_1^n I \right) \bigotimes \left(\overbrace{I \otimes \ldots \otimes I}^{j-1} \otimes \underbrace{\sigma^\dagger + \sigma}_{site\ j'} \otimes \overbrace{Z \left(\sigma^\dagger + \sigma \right)}^{site\ (j+1)'} \otimes \right.$$

$$\otimes \underbrace{I \otimes \ldots \otimes I}_{k-j-3} \otimes \overbrace{Z\left(\sigma^{\dagger} - \sigma\right)}^{site\ (k-1)'} \otimes \underbrace{\sigma^{\dagger} - \sigma}_{site\ k'} \otimes \overbrace{I \otimes \ldots \otimes I}^{n-k} \Bigg),$$

so we obtain only 4-local *plaquette* terms in the auxiliary system, hence *almost* preserving locality.

Chapter 6

Matchgates as universal sets for quantum circuits

The physical realization of a quantum circuit is in general very difficult. Unlike classical digital circuits, quantum gates form a *continuum* in the space of possible operators, so it is impossible to generate any possible quantum circuit by assembling building blocks chosen among a finite set of primitives. To face this issue, it is very important to study the possibility of representing an arbitrary circuit with simpler gates.

In Section 1 we will show that any quantum circuit can be simulated using more elementary gates. In particular, we are interested in representing an arbitrary quantum circuit using only 1-qubit and CZ gates.

In Section 2 we will introduce matchgates: a particular class of 2-qubit gates which can efficiently simulate both 1-qubit and CZ gates.

6.1 Reduction of a quantum circuit to elementary gates

In this section we introduce the concept of *universal set*:

Definition 6.1.1 (Strict universality). *A set $\mathcal{S}$ of quantum gates is said to be* strictly universal *if there exists a constant n_0 such that, for any $n \geq n_0$,*

the subgroup generated by $\mathcal{S}$ is dense in $SU(2^n)$.

We will show that 1-qubit and CZ gates are a strictly universal set for quantum computation. The approach we use is the same as in [6], and all the proofs can be found there.

6.1.1 Decomposition into two-levels unitary gates

Let U be a generic $d \times d$ unitary matrix.

Definition 6.1.2 (Two-level unitary gates)**.** *We say that U is* two-level unitary *if it acts non-trivially only on a 2-dimensional space spanned by two basis elements.*

In other words, a two-level unitary matrix only acts on two vector components. It can be written as $I_{d-2} \otimes V$, where V is a unitary 2×2 matrix and the tensor product is to be intended related to the subspace where U acts upon. For example, if $d = 6$ and U acts only on the subspace spanned by basis elements 3 and 6, we have:

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & a & 0 & 0 & b \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & c & 0 & 0 & d \end{pmatrix},$$

where $V = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

The following holds:

Proposition 6.1.3. *Any $d \times d$ unitary matrix can be written as a product of at most $\mathcal{O}(4^d)$ $d \times d$ two-level unitary matrices.*

It is clear that this decomposition is very inefficient. It is possible to find unitary gates admitting a more efficient decomposition, but in general this bound is optimal: there exist unitary gates which cannot be simulated using less than $\mathcal{O}(4^d)$ two-level matrices (see [6]).

6.1.2 Decomposition by multi-controlled gates

It is clear that a 2-level unitary matrix, acting only on a 2-dimensional subspace, is equivalent to a single-qubit operator up to a reordering of the subspaces involved:

Proposition 6.1.4. *Any two-level unitary matrix U over n qubit can be rewritten as a product of a multi-controlled 1-qubit gate V (which we will denote as C^kV), $\mathcal{O}(n)$ multi-controlled C^kNOTs and $\mathcal{O}(n)$ 1-qubit X gates.*

The trick is to perform a rearrangement of the basis elements using C^kNOTs with control lines activated either by $|0\rangle$ or $|1\rangle$ states following a pattern built upon a *Grey code*, in such a way that the two basis elements spanning the subspace the matrix acts upon are in consecutive positions. Then the C^kV gate is applied and finally the subspaces are reordered by C^kNOTs. The X gates are used to build *reverse controlled gates*, that is, controlled gates that are activated when the (quantum) control bits are set to 0 instead of 1.

Proposition 6.1.5. *Any multi-controlled 1-qubit gate C^dU with one target qubit and d control qubits can be written as a product of $\mathcal{O}(d)$ $SWAPs$, Toffoli $CCNOT$ gates and a single CU gate, with a circuit width overhead of $\mathcal{O}(d)$ ancilla qubits.*

The above Proposition does include C^kNOT gates too, so up to now we can simulate any quantum circuit using only:

- X gates,
- single-controlled 1-qubit CU gates,
- Toffoli $CCNOT$ gates,
- $SWAP$ gates.

6.1.3 Decomposition with more elementary gates

We can further reduce the complexity of the set we have found:

Proposition 6.1.6. *A single Toffoli gate can be constructed using only a finite number of SWAP, CNOT and 1-qubit gates.*

Proposition 6.1.7. *A single-controlled 1-qubit CU gate can be constructed using four 1-qubit unitary gates and two CNOT gates.*

We can now simulate any quantum circuit using only:

- 1-qubit gates,
- *CNOT* gates,
- *SWAP* gates.

6.1.4 Final decomposition

Finally we can further reduce our simulating set thanks to the following observations:

Proposition 6.1.8. *Is it possible to build a SWAP gate by using three n.n. CNOT gates, and a CNOT gate by using one n.n. CZ gate and two 1-qubit Hadamard gates.*

We can finally enunciate:

Proposition 6.1.9. *Every quantum circuit can be implemented using only n.n. CZ and 1-qubit gates, so that 1-qubit gates and n.n. CZ form a strictly universal set for quantum computing.*

6.1.5 Discrete universal sets

It is worthy to mention that in many real-world situation we cannot afford to implement a quantum circuit by using an infinite set of gates such as the one just found. The problem is that unitary gates over n qubits form a continuous space of operators, so it is impossible to perfectly simulate arbitrary gates with a finite number of elementary gates. In these cases we require a less strict definition of universality:

Definition 6.1.10 (Computational universality). *A set $\mathcal{S}$ of quantum gates is said to be* computationally universal *if it can be used to simulate any quantum circuit of width n and depth d up to accuracy ϵ with an overhead of at most $log(poly(n, d, \frac{1}{\epsilon}))$.*

Here *accuracy* ϵ stands for the distance (in matrix norm) between the target unitary circuit and the simulated circuit, which corresponds to an error on the measurements performed over the simulated system. If this error is small, the original and the simulated systems are virtually undistinguishable. The *Solovay-Kitaev Theorem* (see, *e.g.*, [9]) gives a very fast rate of convergence in norm for computationally universal sets. One of such sets is the following (see [1]):

Theorem 6.1.11. *The set composed by the Hadamard and Toffoli gates is computationally universal.*

6.2 Matchgates

In this section we will introduce *matchgates*, a particular class of 2-qubit operators. We will show that matchgates are a universal set for quantum computation.

Definition 6.2.1 (Matchgate). *Let A, B be two unitary 2×2 matrices, namely:*

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \qquad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix},$$

with $A, B \in SU(2)$ or $det(A) = det(B) = -1$. We define matchgate generated by A and B *the 4×4 unitary matrix (2-qubit operator):*

$$G(A, B) \doteq \begin{pmatrix} a & 0 & 0 & b \\ 0 & e & f & 0 \\ 0 & g & h & 0 \\ c & 0 & 0 & d \end{pmatrix}.$$

In other words, $G(A, B)$ is a unitary operator which applies A to the 2-dimensional even qubit parity subspace $\langle |00\rangle, |11\rangle \rangle$, and B to the 2-dimensional

odd qubit parity subspace $\langle |01\rangle, |10\rangle\rangle$. If $det(A) \neq det(B)$, we denote by $\tilde{G}(A,B)$ the same matrix as $G(A,B)$ and we will call it *quasi-matchgate*.

If we have a circuit with $n > 2$ qubit lines, we say that a matchgate M is a *nearest-neighbour (n.n.) matchgate* if it acts on qubit lines j and $j+1$, for $j \in \{1, \ldots, n-1\}$. We say that a matchgate M is a *next-nearest-neighbour (n.n.n.) matchgate* if it acts on qubit lines j and $j+2$, for $j \in \{1, \ldots, n-2\}$. Notice that we can write a $SWAP$ gate between two qubits as:

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \tilde{G}(I, X).$$

In fact, $det(X) = -1 \neq 1 = det(I)$, so that a $SWAP$ is not a matchgate, but just a quasi-matchgate.

6.2.1 Matchgates as a universal set

The following result has been proven in [17]:

Theorem 6.2.2 (Universality for n.n. and n.n.n. matchgates)**.** *Any quantum circuit C of depth d can be rewritten as a circuit of depth poly(d) containing only n.n. and n.n.n. matchgates.*

Proof. We start by introducing the following 4-qubit encoding for the original circuit:

$$|0\rangle \mapsto |0000\rangle, \qquad |1\rangle \mapsto |1001\rangle,$$

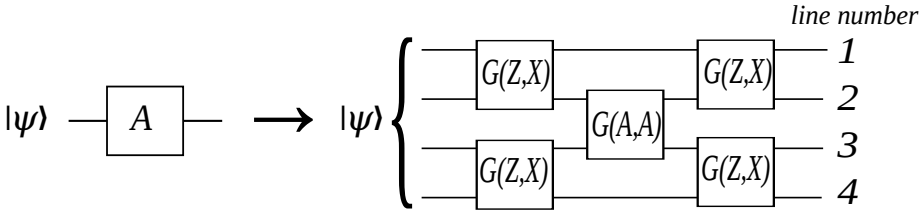
so that a generic 1-qubit state $|\psi\rangle$ will be encoded as the product state $|\psi 00 \psi\rangle$. By doing so we have only a polynomial overhead in the width of the circuit; the aim is to keep the whole computation within the tensor product space spanned by the elements $|0000\rangle$ and $|1001\rangle$.

We have seen in the previous section that 1-qubit gates and n.n. CZ gates are universal for quantum computation, so we will assume that C is made only of these gates. The proof will proceed in three steps:

1. We show that it is possible to simulate an arbitrary 1-qubit gate on this encoding by using a polynomial number of n.n. matchgates. Namely, we can perform the action of a 1-qubit gate A in the following way:

$$A \mapsto G(Z,X)_{12} \cdot G(Z,X)_{34} \cdot G(A,A)_{23} \cdot G(Z,X)_{12} \cdot G(Z,X)_{12},$$

where subscripts denote line numbers the gate acts upon, *e.g.*, $G(A,A)_{23}$ is the matchgate $G(A,A)$ acting on qubit lines 2 and 3. The following picture explains the circuit:

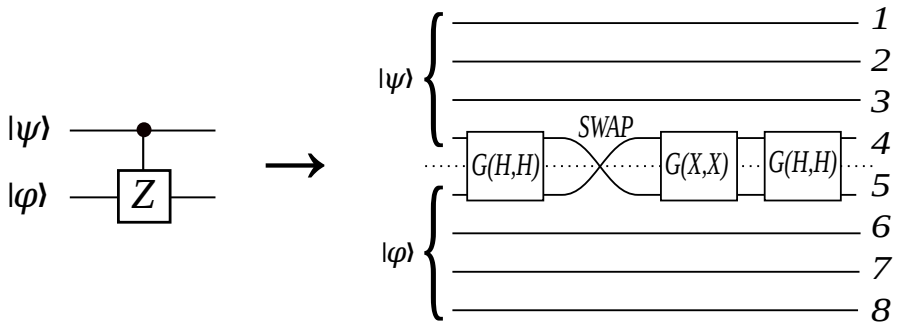


To see how this works just observe that, because of our state encoding, the gate:

$$G(Z,X) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

behaves like a $SWAP$ between line pairs $(1,2)$ and $(3,4)$.

2. We show that it is possible to simulate a CZ over two adjacent qubits (encoded as qubit quadruples $(1,2,3,4)$ and $(5,6,7,8)$) by using a polynomial number of n.n. matchgates and $SWAP$ gates. To do this, we just apply a CZ on the bordering 4 and 5 lines using three n.n. matchgates and a $SWAP$, in the following way:



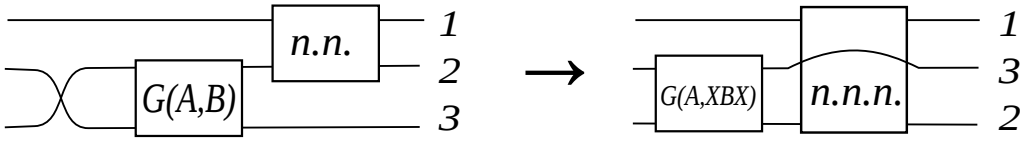
3. We show that it is possible to eliminate the $SWAP$ gates from the obtained representation, by replacing some of the n.n. matchgates with n.n.n. matchgates. Namely, we can eliminate $SWAP$s by commuting them at the end of the circuit and simply re-numbering output lines; by doing so, the $G(X, X)$ and $G(H, H)$ gates are altered, but for any matchgate $G(A, B)$ we have:

$$G(A, B) \cdot SWAP = SWAP \cdot SWAP \cdot G(A, B) \cdot SWAP$$

$$= SWAP \cdot G(A, XBX),$$

that is, $SWAP$s can be moved towards the end of the circuit, and intermediate matchgates are changed but still remain matchgates.

Finally, notice that since $SWAP$s are used only between bordering lines $(4, 5), (8, 9)$, *etc.*, of the quadruples $(1, 2, 3, 4), (5, 6, 7, 8)$, *etc.*, no line can be moved more than one position distant from its original location. This means that, after commuting all the $SWAP$s at the end of the circuit, we are left with only n.n. and n.n.n. matchgates:



Because of the 4-qubit encoding we use, it will be then possible to perform a measurement on the output lines with the same behaviour of the original circuit C (possibly after a re-numbering of the lines where the actions of the $SWAP$s do not cancel each other). This completes the proof. □

This means that using just n.n. and n.n.n. matchgates, we can efficiently (that is, with only a polynomial depth and width overhead) represent any quantum circuit. In the following chapter we will see that there is a big difference between n.n. and n.n.n. matchgates in terms of classical simulation.

Chapter 7

Efficient classical simulation of n.n. matchgates quantum circuits

In this chapter we present some recent results on efficient classical simulation of quantum circuits composed by n.n. matchgates. We follow the approach presented in [17], which extends previous works on similar topics (see i.e. [12] and [25]).

The result obtained is that n.n. matchgates can be classically (strongly) efficiently simulated, while n.n.n. matchgates cannot. Hence, given the results on matchgate universality in the previous chapter, we are left with a stunning conclusion: the gap between classical and quantum computation (if any) is bridged only by a seemingly 'innocuous' topological property of the graph associated to the circuit, *i.e.*, the use of swaps. Hence, the alleged superior power of quantum computing is tied to a conceptually very weak resource. Indeed it has been shown [16] that the class of functions computable by a poly-depth circuit of n.n. matchgate coincides with the class of functions computable in poly-time by quantum circuit using only logarithmic memory space, which is a subset of **P**.

The frame of the proof is the following: first a Clifford algebra formalism is introduced for a certain class of operators. Then it is shown that Gaussian operations built on quadratic Hamiltonians in this new algebra can be efficiently classically simulated (for a restricted class of input states and ob-

servables), but it is also shown that n.n. matchgates are a particular type of such gates. Finally, Clifford operations are used to extend the simulation to arbitrary classes of input states and observables.

7.1 Observables in a Clifford algebra and quadratic Hamiltonians

The key ingredient for the efficient classical simulation of matchgates is a particular algebrical structure generalizing the concept of polynomial ring; we aim to express matchgates as elements of this structure in such a way that some computations become simpler. This structure is a *Clifford algebra*:

Definition 7.1.1 (Clifford algebra). *Let $\mathcal{H}$ be a 2^n-dimensional Hilbert space. The* generator set of a Clifford algebra *is a finite set of n Hermitian operators $\mathcal{G} = \{ g_1, \ldots, g_n \mid g_k = g_k^\dagger : \mathcal{H} \to \mathcal{H}, \text{ for every } k = 1, \ldots, n \}$ such that its elements satisfy the anticommutation relations:*

$$\{g_j, g_k\} = 2\delta_{j,k} I, \text{ for every } j, k = 1, \ldots, n. \tag{7.1}$$

A (complex) Clifford algebra $\mathcal{C}_n$ *on $\mathcal{G}$ is the (formal) polynomial ring* $\mathbb{C}[g_1, \ldots, g_n]$.

Notice that, since every generator squares to the identity, the elements of $\mathcal{C}_n$ are polynomial operators of (maximum (formal) degree n. Therefore every element $x \in \mathcal{C}_n$ can be written as a linear combination of non-ordered monomials of generators:

$$x = \sum_{\{ j_1, \ldots, j_k \} \subset \{ 1, \ldots, n \}} \alpha_{j_1, \ldots, j_k} g_{j_1} \cdots g_{j_k},$$

with $\alpha_{j_1,\ldots,j_k} \in \mathbb{C}$ for every $\{ j_1, \ldots, j_k \}$ (where the subsets $\{j_1, \ldots, j_k\}$ can also be empty), and so $\mathcal{C}_n$, as a vector space, has dimension 2^n.

For an n-qubit system we want to find $2n$ such generators, $\{ c_1, \ldots, c_{2n} \}$, for the Clifford algebra $\mathcal{C}_{2n}$, because in this case the vector space will have dimension $2^{2n} = 2^n \times 2^n$, and thus the (Hermitian) matrix representation of

the c_js will involve matrices of size $2^n \times 2^n$.

Operators satisfying equation 7.1 can be found in the formalism of Fermionic physics: they are the Majorana spinors $c_1, \ldots, c_{2n}$. With this formalism we can define a quadratic Fermionic Hamiltonian as an element in $\mathcal{C}_{2n}$ of the form:

$$H = i \sum_{j,k=1}^{2n} h_{j,k} c_j c_k.$$

Moreover we can assume that $[h_{j,k}]_{j,k}$ is a $2n \times 2n$ real antisymmetric matrix (because $c_j c_k = -c_k c_j$ and $H = H^\dagger$).

7.2 Gaussian operations and efficient simulation

The quantum operations we will consider in the following section will be of a particular form:

Definition 7.2.1 (Gaussian operation). *Let H be a quadratic Hamiltonian in $\mathcal{C}_{2n}$. A* Gaussian operation *is the operator:*

$$U = e^{iH}.$$

These operators are obviously unitary, hence they represent the action of some kind of quantum gates on a $2n$-qubit circuit. These gates are interesting because they can be (strongly) efficiently simulated by classical means, as the following shows:

Lemma 7.2.2. *Let H be a quadratic Hamiltonian in $\mathcal{C}_{2n}$ defined by a real antisymmetric coefficient matrix h, and $U = e^{iH}$ the corresponding Gaussian operation. Then:*

$$U^\dagger c_j U = \sum_{k=1}^{2n} R_{j,k} c_k,$$

for every $j = 1, \ldots, 2n$, where $R \doteq e^{4h}$.

Proof. Write c_j as $c_j(0)$, $U(t) \doteq e^{iHt}$ and $c_j(t) \doteq U(t)c_j(0)U(t)^\dagger$. Then:

$$\begin{aligned}
\frac{dc_j(t)}{dt} &= \frac{d}{dt}\left(e^{iHt}c_j(0)e^{-iHt}\right) \\
&= \left(\frac{d}{dt}e^{iHt}\right)c_j(0)e^{-iHt} + e^{iHt}c_j(0)\frac{d}{dt}e^{-iHt} \\
&= iHe^{iHt}c_j(0)e^{-iHt} - e^{iHt}c_j(0)iHe^{-iHt} \\
&= i\left(Hc_j(t) - c_j(t)H\right) \\
&= i\left[H, c_j(t)\right] \\
&= i\left[i\sum_{k_1,k_2=1}^{2n} h_{k_1k_2}c_{k_1}c_{k_2}, c_j(t)\right] \\
&= -\sum_{k_1,k_2=1}^{2n} h_{k_1,k_2}\left[c_{k_1}c_{k_2}, c_j(t)\right] \\
&= -\sum_{k_1,k_2=1}^{2n} h_{k_1,k_2}U(t)\left[c_{k_1}c_{k_2}, c_j\right]U(t)^\dagger.
\end{aligned}$$

Now we can distinguish between two cases, recalling eq. 7.1:

1. if $j \neq k_1$ and $j \neq k_2$ then: $[c_{k_1}c_{k_2}, c_j] = c_{k_1}c_{k_2}c_j - c_jc_{k_1}c_{k_2} = 0$;
2. if $j = k_1$ or $j = k_2$ then (assuming $j = k_1$): $[c_{k_1}c_{k_2}, c_{k_1}] = c_{k_1}c_{k_2}c_{k_1} - c_{k_1}c_{k_1}c_{k_2} = -c_{k_1}c_{k_1}c_{k_2} - c_{k_1}c_{k_1}c_{k_2} = -2c_{k_2}$.

In this way we have:

$$\frac{dc_j(t)}{dt} = \sum_{k_1=1}^{2n} 2h_{k_1,j}U(t)c_{k_1}U(t)^\dagger + \sum_{k_2=1}^{2n} 2h_{j,k_2}U(t)c_{k_2}U(t)^\dagger = \sum_{k=1}^{2n} 4h_{j,k}c_k(t)$$

(because h is antisymmetric). This holds for every $j = 1, \ldots, 2n$, so we can write it in compact vectorial form:

$$\frac{d}{dt}\begin{pmatrix} c_1(t) \\ \vdots \\ c_{2n}(t) \end{pmatrix} = 4h\begin{pmatrix} c_1(t) \\ \vdots \\ c_{2n}(t) \end{pmatrix},$$

which is a linear system of first order ODEs, whose solution is trivially computable by direct exponentiation:

$$\begin{pmatrix} c_1(t) \\ \vdots \\ c_{2n}(t) \end{pmatrix} = e^{4ht} \begin{pmatrix} c_1(0) \\ \vdots \\ c_{2n}(0) \end{pmatrix},$$

and namely, for every j:

$$c_j(t) = \sum_{k=1}^{2n} R_{j,k}(t) c_k(0),$$

where $R(t) = e^{4ht}$. The lemma follows by just setting $t = 1$. □

The meaning of this lemma is the following: a Gaussian operation involves an exponentiation of a quadratic Hamiltonian H, which is a sum generally involving all the products of all generators, so the operator $U^\dagger c_j U$ could potentially finish up everywhere in the exponentially large (2^{2n}-dimensional) Clifford algebra $\mathcal{C}_{2n}$. However, this is not the case: it always happens to stay within the polynomially small ($2n$-dimensional) subspace spanned by just the generators themselves.

This observation leads to a first step in the strong simulation of those gates:

Theorem 7.2.3 (Efficient simulation of Gaussian gates)**.** *If the expected value of c_j on a certain input state $|\psi_{in}\rangle$ is poly-time computable, then the action of a Gaussian gate U on $|\psi_{in}\rangle$ (relative to c_j) is strongly classically simulatable up to precision $\mathcal{O}(exp(m))$ (*i.e.*, up to m digits of precision), in $\mathcal{O}(poly(n, m))$ time, for every $j = 1, \ldots, 2n$.*

Proof. The Gaussian gate U acts on the input state $|\psi_{in}\rangle$ producing $|\psi_{out}\rangle = U|\psi_{in}\rangle$, and then a measurement of c_j is performed on this output state. We can compute the expected value of this measurement using Lemma 7.2.2:

$$\langle c_j \rangle_{out} = \langle \psi_{out}| c_j |\psi_{out}\rangle = \langle \psi_{in}| U^\dagger c_j U |\psi_{in}\rangle = \sum_{k=1}^{2n} R_{j,k} \langle \psi_{in}| c_k |\psi_{in}\rangle .$$

This sum only involves $2n$ terms. The matrix $R = e^{4h}$ is computable up to $poly(m)$ digits of precision (that is, up to $exp(m)$ accuracy), where m is the order of the partial sum used to approximate the exponentiation in the Clifford algebra series expansion, being hence computable in $poly(n, m)$ steps. The expected value on the input state $\langle c_j \rangle_{in}$ is also poly-time computable by hypothesis. Thus, $\langle c_j \rangle_{out}$ is poly-time computable. □

This method only works for observables in the Clifford algebra, and only for input states whose expected value on those operators can be known in polynomial time. We must thus find a way to extend this result to a broader range of instances.

Consider now input product states, that is, of the form: $|\psi\rangle_{in} = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \ldots \otimes |\psi_n\rangle$, and suppose that the c_js can be written as product operators: $c_j = P_{1,j} \otimes P_{2,j}, \ldots, P_{n,j}$ (where the $P_{k,j}$s are one-qubit Hermitian operators). Then we would have:

$$\langle c_j \rangle_{in} = \langle \psi_{in}| c_j |\psi_{in}\rangle = \prod_{k=1}^{n} \langle \psi_k| P_{k,j} |\psi_k\rangle ,$$

which is also poly-time computable, involving a polynomial number of computations of the expectation value of single-qubit observables. Hence, product states are suitable for the classical simulation of Gaussian circuits with respect to product operators in the Clifford algebra.

Finally, recall that $\mathcal{C}_{2n}$, as a vector space, has dimension $2^n \times 2^n$, so that any n-qubit observable can be expressed as a linear combination of elements in the Clifford algebra. This means that any observable ξ must be expressible as a polynomial in the c_js. Let d be the degree of this polynomial, and assume that ξ is a monomial, namely $\xi = \prod_{j=1}^{d} c_j$. Let $|\gamma_1\rangle, \ldots, |\gamma_{2^n}\rangle$ be our

computational basis, and recall Lemma 7.2.2. Then:

$$\begin{aligned}
\langle \xi \rangle_{out} &= \langle \psi_{in} | \, U^\dagger c_1 \dots c_d U \, | \psi_{in} \rangle \\
&= \langle \psi_{in} | \left(U^\dagger c_1 U \right) \left(U^\dagger c_2 U \right) \dots \left(U^\dagger c_d U \right) | \psi_{in} \rangle \\
&= \langle \psi_{in} | \prod_{j=1}^{d} U^\dagger c_j U \, | \psi_{in} \rangle \\
&= \langle \psi_{in} | \prod_{j=1}^{d} \left(\sum_{k_j=1}^{2n} R_{j,k_j} c_{k_j} \right) | \psi_{in} \rangle \\
&= \sum_{k_1 \neq \dots \neq k_d = 1}^{2n} R_{1,k_1} \dots R_{d,k_d} \langle \psi_{in} | \, c_{k_1} \dots c_{k_d} \, | \psi_{in} \rangle \, ,
\end{aligned}$$

where the last sum involves $\mathcal{O}(n^d)$ terms, each of them being poly time computable (notice that, if every c_j is a product operator, then also the $c_{k_1} \dots c_{k_d}$ are product operators and thus their expected values on a product state can be computed in polynomial time). The whole expression can then be computed in $\mathcal{O}(poly(n^d, m))$ time, which translates to $\mathcal{O}(poly(n, m))$ if d is bounded to a constant. It is clear that this result can be generalized to observables ξ which are not monomials, provided that they are bounded-degree polynomials in $\mathcal{C}_{2n}$.

So we have just shown that Gaussian operators can be (strongly) efficiently simulated if we:

- are able to find a representation of the c_js as product operators;
- restrict to observables which can be written as bounded-degree polynomials in $\mathcal{C}_{2n}$;
- restrict to product input states.

We will address the first and (part of) the second point in the following section.

7.3 The Jordan-Wigner representation

The *Jordan-Wigner representation* is strictly tied to the Jordan-Wigner Transform and gives an explicit representation of the generators c_js in terms of tensor products of Pauli matrices. Namely, for an n-qubit system, we will define the following:

$$\begin{array}{rclrcl}
c_1 &=& X_1 I_2 \dots I_n, & c_2 &=& Y_1 I_2 \dots I_n, \\
c_3 &=& Z_1 X_2 I_3 \dots I_n, & c_4 &=& Z_1 Y_2 I_3 \dots I_n, \\
&\vdots& & &\vdots& \\
c_{2k-1} &=& Z_1 \dots Z_{k-1} X_k I_{k+1} \dots I_n, & c_{2k} &=& Z_1 \dots Z_{k-1} Y_k I_{k+1} \dots I_n, \\
&\vdots& & &\vdots& \\
c_{2n-1} &=& Z_1 \dots Z_{n-1} X_n, & c_{2n} &=& Z_1 \dots Z_{n-1} Y_n.
\end{array}$$

Notice that this is a representation of the c_js as product operators, so the first point of the previous section is satisfied. Now let us see how the corresponding Gaussian gates do look like when we use this representation. We must also find a suitable class of observables to actually perform quantum computation which must be expressible as constant-degree polynomials of these operators.

7.3.1 N.n. matchgates

The surprising result here is that n.n. matchgates are indeed a particular class of Gaussian gates when the Jordan-Wigner representation is used:

Theorem 7.3.1 (N.n. matchgates representation in $\mathcal{C}_{2n}$). *For every n.n. matchgate M acting only on lines j and $j+1$, there exists a (unique) quadratic Hamiltonian, H_M, comprising only terms in $\mathcal{C}_{2n}$ relative to n.n. qubit lines j and $j+1$ (namely, only $c_{2j-1}, c_{2j}, c_{2j+1}$ and c_{2j+2}), such that $e^{iH_M} = M$. That is, n.n. matchgates are Gaussian operations in the Clifford algebra $\mathcal{C}_{2n}$.*

Proof. Recall that $M = G(A, B)$, where A and B are unitary 2×2 matrices with the same determinant. So there must exist two Hermitian 2×2 matrices, A' and B', such that $A = e^{iA'}$ and $B = e^{iB'}$. Hence, to express a matchgate M as a Gaussian operation, it is sufficient to be able to represent – in the even and odd parity subspaces separately – every Hermitian matrix by means

of linear combinations of (at most) quadratic terms in $\mathcal{C}_{2n}$, and then we could generate M by direct exponentiation.

To do this, it is sufficient to represent all the elements of a basis for $SU(2)$ - *i.e.*, Pauli matrices X, Y, Z - separately in the two parity subspaces as quadratic polynomials. This is indeed possible, in the following way:

$$
\begin{aligned}
X^{odd}_{j,j+1} &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \frac{1}{2}\left(X_j X_{j+1} + Y_j Y_{j+1}\right) = \frac{1}{2}\left(c_{2j-1}c_{2j+1} + c_{2j}c_{2j+2}\right); \\
X^{even}_{j,j+1} &= \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \frac{1}{2}\left(X_j X_{j+1} - Y_j Y_{j+1}\right) = \frac{1}{2}\left(c_{2j-1}c_{2j+1} - c_{2j}c_{2j+2}\right); \\
Y^{odd}_{j,j+1} &= i\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \frac{1}{2}\left(Y_j X_{j+1} - X_j Y_{j+1}\right) = \frac{1}{2}\left(c_{2j}c_{2j+1} - c_{2j-1}c_{2j+2}\right); \\
Y^{even}_{j,j+1} &= i\begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \frac{1}{2}\left(Y_j X_{j+1} + X_j Y_{j+1}\right) = \frac{1}{2}\left(c_{2j}c_{2j+1} + c_{2j-1}c_{2j+2}\right); \\
Z^{odd}_{j,j+1} &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \frac{1}{2}\left(Z_j I_{j+1} - I_j Z_{j+1}\right) = -\frac{i}{2}\left(c_{2j-1}c_{2j} - c_{2j+1}c_{2j+2}\right); \\
Z^{even}_{j,j+1} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} = \frac{1}{2}\left(Z_j I_{j+1} + I_j Z_{j+1}\right) = -\frac{i}{2}\left(c_{2j-1}c_{2j} + c_{2j+1}c_{2j+2}\right).
\end{aligned}
$$

Notice that it is not possible to express the identity matrix I separately in the two parity subspaces in this way, but we can represent the matrix I (in the overall space) as a square of any operator, so it is possible to

introduce an arbitrary phase factor in $M = e^{iH}$; that is, we are not only restricted to n.n. matchgates $G(A, B)$ where $A, B \in SU(2)$, but also to the case $det(A) = det(B) = \phi$, where ϕ is an arbitrary phase – *i.e.*, any n.n. matchgate. □

So this proves that n.n. matchgates are classically efficiently simulatable, and hence every circuit that can be decomposed in n.n. matchgates is also efficiently simulatable.

7.3.2 Other Gaussian gates

N.n. matchgates are only a very particular class of Gaussian gates in this algebra. We already know that every Gaussian gate is classically simulatable, so we might wonder if it could be possible to find some other interesting classes of gates among them – not computationally equivalents to n.n. matchgates. It can be proven that this is not possible (see [17]):

Theorem 7.3.2 (Decomposition of other Gaussian gates). *Every Gaussian gate in $\mathcal{C}_{2n}$ can be written as a circuit of at most $\mathcal{O}(n^3)$ n.n. matchgates.*

So every Gaussian gate is computationally equivalent, and hence it adds nothing important to our theory. Can we hope to find a Gaussian representation also for n.n.n. matchgates, being hence able to classically efficiently simulate universal quantum computation?

7.3.3 N.n.n. matchgates

As we should expect, the answer is no:

Theorem 7.3.3 (Non-Gaussianity of n.n.n. matchgates). *N.n.n. matchgates cannot be in general represented as Gaussian operators in $\mathcal{C}_{2n}$.*

Proof. Let it M be a n.n.n. matchgate acting, *e.g.*, on n.n.n. lines j and $j+2$. Then, when we perform quadratic products of operators associated to

those lines, we could get something like:

$$\begin{aligned} c_{2j}c_{2j+4} &= (Z_1 \ldots Z_{j-1}Y_jI_{j+1}\ldots I_n)\,(Z_1 \ldots Z_{j+1}Y_{j+2}I_{j+3}\ldots I_n) \\ &= I \otimes \ldots \otimes I \otimes iX \otimes \underbrace{Z}_{\text{line } j+1} \otimes Y \otimes I \otimes \ldots \otimes I. \end{aligned}$$

In this case a Z-string operator appears, acting on the middle line. So, when we perform exponentiation of this quadratic term to yield a Gaussian operator, we obtain a gate acting non-trivially on all the three lines $j, j+1$ and $j+2$, hence not a matchgate. □

7.3.4 Bounded-degree observables for computation

Recall that we must still find observables suitable for quantum computation that can be represented as constant-degree polynomials in $\mathcal{C}_{2n}$. This is indeed possible for, among the others, a large class of single-line observables, *e.g.*:

$$Z_k = -ic_{2k-1}c_{2k}.$$

This is enough for our model of quantum circuit, since a quantum algorithm that ends with an observation of Z on a certain line can give 'yes' or 'no' answers. Any other question could then be answered with $\mathcal{O}(\log n)$ 'yes'-or-'no' questions (*e.g.*: 'is the result greater than x? Is it lesser than y?' *etc.*).

So now the only issue we are left with is that input states (for this simulation framework) must be product states – that is, no entanglement is allowed. This would not be a very interesting result, since it is already known [6] that entanglement-free quantum computation is no stronger than classical computation.

7.4 Extending input states and observables with Clifford operations

There is indeed a method to introduce entanglement in the input without disrupting the simulation scheme.

Definition 7.4.1 (Pauli group). *The* Pauli group *on n qubits, $\mathcal{P}_n$, is defined as the set:*

$$\mathcal{P}_n \doteq \{\, \lambda \bigotimes_{k=1}^{n} P_k \mid \lambda \in \{\, +1, -1, +i, -i \,\}, P_k \in \{\, I, X, Y, Z \,\} \text{ for every } k \,\},$$

together with the product operation between $2^n \times 2^n$ matrices.

It is trivial to see that this is a subgroup of $GL(n)$.

Definition 7.4.2 (Clifford operation). *An n-qubit operation T is a* Clifford operation *if $T^\dagger A T \in \mathcal{P}_n$, for every $A \in \mathcal{P}_n$.*

Notice that conjugation by a Clifford operation preserves the Pauli group structure. The following trivially holds:

Proposition 7.4.3. *If $\{\, c_j \,\}_j$ is a family of operators such that $\{c_j, c_k\} = 2\delta_{j,k} I$, for every j, k, then for every Clifford operation V the family $\{\, V^\dagger c_j V \,\}_j$ also verifies the anticommutation property.*

With this in mind recall that Lemma 7.2.2 only relies on the anticommutation relations of the generators of $\mathcal{C}_{2n}$, hence it also holds if we substitute every c_j with $V^\dagger c_j V$, where V is an arbitrary Clifford operation. Moreover, the Clifford operation V can always be taken outside the quadratic Hamiltonian and the corresponding Gaussian operation U':

$$\begin{aligned}
U' &= e^{\sum_{j,k} h_{j,k} \left(V^\dagger c_j V\right)\left(V^\dagger c_k V\right)} \\
&= e^{V^\dagger \left(\sum_{c_j, c_k} h_{j,k} c_j c_k\right) V} \\
&= e^{V^\dagger H V} \\
&= \sum_{m=0}^{\infty} \frac{\left(V^\dagger H V\right)^m}{m!} \\
&= V^\dagger \sum_{m=0}^{\infty} \frac{H^m}{m!} V \\
&= V^\dagger e^H V \\
&= V^\dagger U V.
\end{aligned}$$

This means that, if we have a circuit composed of these kind of gates, we can just consider the Clifford operation at the beginning and at the end

of the circuit (because middle terms cancel each other), without disrupting the Pauli product structure of the observable, hence allowing us to maintain efficient classical simulability:

$$\langle V^\dagger c_j V\rangle_{out} = \langle\psi_{in}|\left(V^\dagger U^\dagger V\right)\left(V^\dagger c_j V\right)\left(V^\dagger U V\right)|\psi_{in}\rangle = \langle\psi_{in}|\, V^\dagger\, U^\dagger c_j U\, V\,|\psi_{in}\rangle\,,$$

so we can just think the new circuit as the old one, but this time we have an observable $V^\dagger c_j V$ and an input state $V\,|\psi_{in}\rangle$. This is enough to introduce entanglement in the input state and to have a generic multi-line observable (see [17]). More precisely, the following is known:

Theorem 7.4.4 (Representation of Clifford operations). *Every Clifford operation can be expressed as a circuit of CNOT, Hadamard and diag*$(1, i)$ *gates.*

It is important to remark that with the use of Clifford operations only we are able to construct valuable families of quantum states, which, *a priori*, contain all the properties that make the states amenable to be useful in quantum information processing. Namely, Clifford operations can give rise to multipartite entanglement which is a necessary ingredient for quantum speed-up. Notice that in the course of our treatment, we have chosen to not make reference to entanglement, since we are mainly interested in the aspects of classical simulation without considering this type of constraints.

Chapter 8

Extending the simulation to n.n.n. matchgates

In this chapter we propose new ideas about the possibility of extending the class of efficiently simulatable matchgates beyond the n.n. ones. As far as we know, the results presented herein address some open question about the simulability of certain n.n.n. matchgates and open new possible directions for further investigation.

In Section 1 we investigate n.n.n. matchgate with the Clifford algebra formalism presented in the previous chapter. We give a sufficient condition for one of such matchgates to be efficiently simulatable despite Theorem 7.3.3; *i.e.*, given a certain n.n.n. matchgate in the above formalism, we have a criterion for its efficient simulatability. We also give a simple algorithm to perform an exhaustive search for Gaussian operators that translates to n.n.n. matchgates, and we give an explicit example of one of those matchgates found through a C implementation of the algorithm.

In Section 2 we propose a different approach for the simulation of matchgates based on the JWT technique to map bidimensional local Hamiltonians of Fermions into local Hamiltonian of spins (as seen in Chapter 5). Our result is obtained by representing operators in a sub-algebra of a larger Clifford algebra: by adding an additional layer of qubits we will be able to remove Z-string operators from half of the system. If our quadratic Hamiltonians

have a particular form, then the resulting Gaussian gates can be expressed as a product of two gates, one of the two acting only on a single, arbitrary pair of circuit lines (not only n.n. and n.n.n. lines). This leads to the encoding of a matchgate in the physical system, as seen in the previous chapter.

8.1 Simulatable n.n.n. matchgates

In this section we try to characterize those n.n.n. matchgates that are still Gaussian in the Clifford algebra representation introduced in the previous chapter. In fact, Theorem 7.3.3 does not say that every n.n.n. is not Gaussiane, but only that there exist non-Gaussian n.n.n. matchgates. So we can hope to find a class of simulatable matchgates strictly larger than the n.n. matchgates class.

First of all notice that since there is a 1-to-1 correspondence between a unitary gate $U = e^{iH}$ and its generator H, then we can focus our attention on the properties of the generator itself. It is always possible, given U, to find H:

Proposition 8.1.1. *Let $U \in U(d)$ be a unitary operator. Then it is possible to find in an efficient way the unique H such that $e^{iH} = U$.*

The matrix iH is called *matrix logarithm* of U and can be found as a power series with arbitrary accuracy. It is not generally possible to find a matrix logarithm for an arbitrary complex matrix (being the matrix logarithm a multi-valued application, like the ordinary complex logarithm), but for a unitary matrix it is possible.

So we will restrict our attention to the study of the generator Hamiltonian of a Gaussian gate; we will also restrict to the simplest case $n = 3$ for the study of n.n.n. matchgates, but the results can be easily generalized to circuits of arbitrary width.

8.1.1 A criterion for n.n.n. matchgates simulability

Recall that, since we restrict to $n = 3$, the generators of our $\mathcal{C}_6$ Clifford algebra are:

$$\begin{array}{lll} \text{for line 1:} & c_1 = X_1 I_2 I_3, & c_2 = Y_1 I_2 I_3; \\ \text{for line 2:} & c_3 = Z_1 X_2 I_3, & c_4 = Z_1 Y_2 I_3; \\ \text{for line 3:} & c_5 = Z_1 Z_2 X_3, & c_6 = Z_1 Z_2 Y_3. \end{array}$$

The number of independent quadratic products made from the above generators is $6(6-1)/2+1 = 16$ (because generators anticommute, yielding the same quadratic term save for a minus sign, plus each generator squares to the identity). We will denote these quadratic terms by $q_1, \ldots, q_{16}$.

Given a n.n.n. matchgate U, it is efficiently classically simulatable if it is Gaussian, *i.e.*, its generator Hamiltonian H must be a complex linear combination of $q_1, \ldots, q_{16}$. To see if this holds, we introduce a *matrix inner product*, *i.e.*, an application $((.,.)) : \mathbb{C}^{n\times n} \times \mathbb{C}^{n\times n} \to \mathbb{C}$ which satisfies all the usual properties of an inner product, *e.g.*:

$$((A, B)) = \sum_{j,k=1}^{n} a_{j,k}\overline{b_{j,k}},$$

where $A \doteq [a_{j,k}]_{j,k}$ and $B \doteq [b_{j,k}]_{j,k}$ are $n \times n$ complex matrices. With this product, $\mathbb{C}^{n\times n}$ can be seen as a 2^{n^2}-dimensional complex Hilbert space, and we can use the Graham-Schmidt orthonormalization process on $q_1, \ldots, q_{16}$ to obtain an orthonormal set of elements $g_1, \ldots, g_{16}$.

At this point we decompose U in the subspace spanned by these orthonormal elements:

$$U' \doteq \sum_{j=1}^{16} g_j \left((g_j, U)\right),$$

and so we obtain the following:

Theorem 8.1.2 (Sufficient criterion for simulability). *If $U = U'$, then U is classically efficiently simulatable as a Gaussian gate generated by a quadratic Hamiltonian over a Clifford algebra.*

Finally, notice that this theorem holds for every Gaussian gate, not only for matchgates.

8.1.2 Building simulatable n.n.n. matchgates

In this section we discuss a general method to look for Hamiltonians that are both Gaussian and representing a n.n.n. matchgate. We start by investigating the property of these Hamiltonians and we give an algorithm to perform an exhaustive search of one of these Hamiltonians as a linear combination of $g_1, \ldots, g_{16}$ over a pre-fixed set of complex coefficients. We implemented this algorithm in C language, and the program was able to find an example of simulatable n.n.n. matchgate.

Structure of the Hamiltonian

Let $A, B \in U(2)$ with $det(A) = det(B)$. Recall that a matchgate $M = G(A, B)$ is the two-qubit 4×4 unitary operator applying A to the even parity subspace of the 4-dimensional Hilbert space representing the two qubit lines, and B to the odd parity subspace. In our case we set $n = 3$ and we want a n.n.n. matchgate (that is, acting on lines 1 and 3). We ask ourselves what is the structure of the Hamiltonian H such that $U = e^{iH}$.

Let us denote by H_A and H_B the Hamiltonians generating A and B respectively, namely:

$$H_A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \qquad H_B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}.$$

The Hamiltonian generating A in the even parity subspace related to lines 1 and 3 without altering line 2 is the following:

$$H_A^{even} = \begin{array}{c|c} & \begin{array}{cccccccc} |000\rangle & |001\rangle & |010\rangle & |011\rangle & |100\rangle & |101\rangle & |110\rangle & |111\rangle \end{array} \\ \hline \begin{array}{c} |000\rangle \\ |001\rangle \\ |010\rangle \\ |011\rangle \\ |100\rangle \\ |101\rangle \\ |110\rangle \\ |111\rangle \end{array} & \begin{pmatrix} a & 0 & 0 & 0 & 0 & b & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a & 0 & 0 & 0 & 0 & b \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & c & 0 & 0 & 0 & 0 & d \end{pmatrix} \end{array}.$$

The Hamiltonian generating B in the odd parity subspace related to lines 1 and 3 without altering line 2 is the following:

$$H_B^{odd} = \begin{array}{c|c} & \begin{array}{cccccccc} |000\rangle & |001\rangle & |010\rangle & |011\rangle & |100\rangle & |101\rangle & |110\rangle & |111\rangle \end{array} \\ \hline \begin{array}{c} |000\rangle \\ |001\rangle \\ |010\rangle \\ |011\rangle \\ |100\rangle \\ |101\rangle \\ |110\rangle \\ |111\rangle \end{array} & \left(\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e & 0 & 0 & f & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e & 0 & 0 & f & 0 \\ 0 & g & 0 & 0 & h & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & g & 0 & 0 & h & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \end{array} .$$

So the Hamiltonian H associated to the matchgate $G(A, B)$ is:

$$H = H_A^{even} + H_B^{odd} = \begin{pmatrix} a & 0 & 0 & 0 & 0 & b & 0 & 0 \\ 0 & e & 0 & 0 & f & 0 & 0 & 0 \\ 0 & 0 & a & 0 & 0 & 0 & 0 & b \\ 0 & 0 & 0 & e & 0 & 0 & f & 0 \\ 0 & g & 0 & 0 & h & 0 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & d & 0 & 0 \\ 0 & 0 & 0 & g & 0 & 0 & h & 0 \\ 0 & 0 & c & 0 & 0 & 0 & 0 & d \end{pmatrix} . \tag{8.1}$$

Notice that we do not require that A and B be in $SU(2)$, but to yield a special unitary gate we should require that $tr(H) = 0$, *i.e.*, $tr(H_A) + tr(H_B) = 0$.

Algorithm for the exhaustive search of simulatable n.n.n. matchgates

The following proof-of-concept algorithm tests for linear combinations of quadratic elements in the Clifford algebra until it finds a matchgate with the structure of eq. 8.1:

1. input: a set $\{ 0, \lambda_1, \ldots, \lambda_r \}$ of complex coefficients;
2. begin:
3. generate quadratic terms $q_1, \ldots, q_{16}$;
4. set $T = true$;

5. repeat until $T = false$:

6. set M as a linear combination of $q_1, \ldots, q_{16}$ by elements of $\{ 0, \lambda_1, \ldots, \lambda_r \}$;

7. if M has the form of eq. 8.1 then return M and terminate;

8. if every combination has been tested, then set $T = false$;

9. return error (the set $\{ 0, \lambda_1, \ldots, \lambda_r \}$ is not suitable to generate a n.n.n. simulatable matchgate).

Of course this algorithm is not optimal and many performance improvements can be done. Particular care must be taken in the choice of the input coefficients set, for a too large input set would rapidly slow down the execution time of the algorithm.

An explicit example of simulatable n.n.n. matchgate

By setting $\{ 0, 1, -1, i, -i \}$ as an input coefficient set, we were able to test the algorithm, and our implementation found the following example:

$$H = ic_6c_5 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}.$$

Notice that $tr(H) = 0$. Up to a renormalization, the action of the Gaussian gate generated by this matrix is to apply a different phase factor, depending on the value of qubit line number 3. This is a very elementary example of n.n.n. efficiently simulatable matchgate. Its interest lies in the observation that every Clifford operation can be expressed as a circuit of $CNOT$, Hadamard and $diag(1, i)$ gates (see [17]).

8.2 An alternative representation for matchgates

In this section we will present a method to extend the efficient simulation to quantum circuits made of a particular kind of matchgates acting on any arbitrary pair of lines. The main idea was suggested by the work in [7] because, basically, the fundamental problem arising when we try to represent non-n.n. matchgates in the Clifford algebra $\mathcal{C}_{2n}$ is the presence of string operators appearing on the middle lines.

Our idea is to add an additional 'layer' of qubits (the *'auxiliary'*, or *'secondary'* subsystem) to the *'physical'* (or *'primary'*) subsystem, in such a way that the global system can be represented as a tensor product of the two, yet allowing us to define a particular class of Gaussian gates which acts as matchgates on the primary subsystem; the primary subsystem will be just a copy of the original circuit we wanted to simulate, while the auxiliary subsystem will be used as a 'dump' for the 'non-simulatability' of the new matchgates. Namely, we will show a different representation of the Clifford algebra $\mathcal{C}_{2n}$ as a *subalgebra* of a larger Clifford algebra $\mathcal{C}_{4n}$ on the global system, which allows us to 'unload' in the auxiliary subsystem the string operators arising when considering products of elements related to distant lines, while keeping observables in the physical subsytem to a constant degree. In this way we will yield a class of multi-qubit Gaussian gates that can be decomposed as a tensor product of two operators: a gate on the physical subsystem acting only on two lines j and k and a gate acting generally on every line between j and k in the auxiliary subsystem. This means that, under the assumption that the quadratic Hamiltonian has a certain structure, we can isolate the behaviour of the gate on the physical subsystem to build a matchgate. By keeping the two subsystems separated during all the computation we will thus be able to classically efficiently simulate a circuit encoded in the physical half of the system.

8.2.1 System setup

Suppose we want to simulate a quantum circuit of width n. Then we have an n-qubit quantum register, with qubits sites indexed as $1, 2, \ldots, n$. This will be the primary system. Then we add an additional layer of qubits, indexed as $1', 2', \ldots, n'$. We can choose any ordering of the sites (with respect to Fermionic representation), *e.g.* $1, 1', 2, 2', \ldots, n, n'$. Our technique is independent from the choice of the ordering, but we must keep it in mind once chosen. To represent this fact we introduce the following notation:

Definition 8.2.1. *Let $|\psi_1 \ldots \psi_n\rangle$ be a product state of the primary subsystem and $|\psi_{1'} \ldots \psi_{n'}\rangle$ a product state of the secondary subsystem. Let $(m_1, \ldots, m_{2n})$ be a permutation of the set $\{1, \ldots, n, 1', \ldots, n'\}$ representing the Fermionic subspace ordering we choose. Then we denote as $\tilde{\otimes}$ the tensor product such that:*

$$|\psi_1 \ldots \psi_n\rangle \,\tilde{\otimes}\, |\psi_{1'} \ldots \psi_{n'}\rangle = |\psi_{m_1}, \ldots, \psi_{m_{2n}}\rangle\,;$$

in other words, $\tilde{\otimes}$ is the tensor product keeping in account the Fermion site ordering we choose when representing the joint system. We define $\tilde{\oplus}$ accordingly.

So if, *e.g.*, $n = 2$ and we choose for the global system the ordering $(1, 2', 2, 1')$ (being $(1, 2)$ the ordering on the physical system and $(1', 2')$ the ordering on the auxiliary one), then we have:

$$|\alpha\beta\rangle \,\tilde{\otimes}\, |\gamma\delta\rangle = |\alpha\delta\beta\gamma\rangle\,.$$

This will help readability when we want to indicate a separation between the two systems.

8.2.2 Defining the new operators

So we have now a $2n$-qubit system, hence we could follow the approach seen in the previous chapters to build a $\mathcal{C}_{4n}$ Clifford algebra made of Majorana spinors. We would obtain the same previous results, though.

Instead, we are going to define a $2n$-dimensional *Clifford subalgebra* of $\mathcal{C}_{4n}$, *i.e*, a Clifford algebra contained in $\mathcal{C}_{4n}$ as a subset. We will indicate this structure by $\mathcal{C}'_{2n}$, and corresponding generators $f_1, \ldots, f_{2n}$.

It is easy to see that the following elements form a generator set for a Clifford subalgebra of $\mathcal{C}_{4n}$:

$$\begin{aligned} f_{2k-1} &\doteq (I_1 \ldots I_{k-1} X_k I_{k+1} \ldots I_n) \,\tilde{\otimes}\, (Z_1 \ldots Z_{k-1} X_k I_{k+1} \ldots I_n)\,, \\ f_{2k} &\doteq (I_1 \ldots I_{k-1} Y_k I_{k+1} \ldots I_n) \,\tilde{\otimes}\, (Z_1 \ldots Z_{k-1} X_k I_{k+1} \ldots I_n)\,, \end{aligned}$$

for every $k = 1, \ldots, n$. Now we can follow again step-by-step the matchgate simulation approach as previously seen, and notice the differences.

8.2.3 Algebra structure and Pauli operators

The Clifford algebra structure of the f_js is preserved (because f_js anticommute and square to identity), so Lemma 7.2.2 still holds.

Then, notice that every f_k is still a product operator. We are interested, as a first step, in representing the Z-Pauli observable on any qubit line as a constant degree polynomial in this new algebra. However this time $\mathcal{C}'_{2n}$ as a vector space has only dimension $2^{2n} = 2^n \times 2^n$, so it cannot span all the $2n$-qubit matrices. Hence most of the observables cannot be expressed as polynomials in $\mathcal{C}'_{2n}$.

In our case, though, we do not need such a strong requirement: recall that we are just interested in the simulation of the physical subsystem. Hence it suffices for us to represent only Z_k on the physical subsystem as a constant-degree polynomial. And this is indeed possible, namely: $Z_k = -if_{2k}f_{2k-1}$, which has bounded degree 2. Recall that for our model of quantum circuit computation, a single Z measurement over a single output line is sufficient to obtain yes-or-no answers – and so, to compute Boolean functions.

Thus, we can efficiently classically calculate $\langle Z_k \rangle_{out}$ for every line k in the physical half of a circuit made only of Gaussian gates generated by quadratic Hamiltonians in $\mathcal{C}'_{2n}$.

8.2.4 The new Gaussian gates

Let H be a quadratic Hamiltonian in ${\mathcal{C}'}_{2n}$:

$$H = i \sum_{j,k=1}^{n} h_{j,k} f_j f_k,$$

and a corresponding Gaussian operation:

$$U = e^{iH}.$$

In general, Gaussian operators of this form acts non-trivially across the physical and the auxiliary subsystems. But if the Hamiltonian H has a particular form, then we can separate the action of the Gaussian gate:

Theorem 8.2.2 (Separable gates in ${\mathcal{C}'}_{2N}$)**.** *Let H be a quadratic Hamiltonian in ${\mathcal{C}'}_{2n}$ comprising only terms related to qubit lines j and k (namely, only $f_{2j-1}, f_{2j}, f_{2k-1}$ and f_{2k}). Moreover, let H be of the following form:*

$$H = \sum_{r \in \mathcal{M}} \left(T_r^{phys} \tilde{\oplus} T_r^{aux} \right), \tag{8.2}$$

where $\mathcal{M} = \{1, \ldots, m\} \subset \mathbb{N}$ and T_r^{phys}, T_r^{aux} are the physical and auxiliary half, respectively, of some (formal) quadratic homogeneous polynomial in the variables $f_{2j-1}, f_{2j}, f_{2k-1}$ and f_{2k}.

If $U = e^{iH}$ is H's associated Gaussian gate, then $U = V \tilde{\otimes} W$, where V acts only on lines j and k in the physical subsystem, while W acts only on the lines from j' to k' in the auxiliary subsystem.

Moreover, it is possible to find a Hamiltonian H_M with the same structure as in eq. 8.2 such that $e^{iH_M} = M \tilde{\otimes} W$, where M is a matchgate acting on lines j and k in the physical subsystem, while W is a gate acting only on the lines from j' to k' in the auxiliary subsystem.

Proof. Recall that H is of the form:

$$H = \sum_{r \in \mathcal{M}} \left(T_r^{phys} \tilde{\oplus} T_r^{aux} \right),$$

with:

$$\begin{aligned} T_r^{phys} &= I \otimes \ldots \otimes I \otimes \overbrace{A_r}^{\text{site } j} \otimes I \otimes \ldots \otimes I \otimes \overbrace{B_r}^{\text{site } k} \otimes I \otimes \ldots \otimes I, \\ T_r^{aux} &= I \otimes \ldots \otimes I \otimes \overbrace{C_r \otimes Z \otimes \ldots \otimes Z \otimes D_r}^{\text{sites from } j' \text{ to } k'} \otimes I \otimes \ldots \otimes I, \end{aligned}$$

where A_r, B_r, C_r, D_r are Pauli operators save for a constant. Hence:

$$\begin{aligned} U &= e^{i \sum_r \left(T_r^{phys} \tilde{\oplus} T_r^{aux} \right)} \\ &= \prod_r e^{i \left(T_r^{phys} \tilde{\oplus} T_r^{aux} \right)} \\ &= \prod_r \left(e^{i T_r^{phys}} \tilde{\otimes} e^{i T_r^{aux}} \right) \\ &= \left(\prod_{s \in \mathcal{M}} e^{i T_s^{phys}} \right) \tilde{\otimes} \left(\prod_{t \in \mathcal{M}} e^{i T_t^{aux}} \right) \\ &= \left(e^{i \sum_s T_s^{phys}} \right) \tilde{\otimes} \left(e^{i \sum_t T_t^{aux}} \right) \doteq V \tilde{\otimes} W, \end{aligned}$$

the operators at the exponent being Hermitians (since they are sum and products of Pauli operators), thus U is expressible as tensor product of two distinct unitary gates.

Finally, note that because of the particular structure of the T_r^{aux}s and T_r^{phys}s, the resulting operator in the auxiliary subsystem W will act nontrivially only on lines form j' to k', while the operator in the physical subsytem V will act only on lines j and k. So it is possible to repeat the procedure seen in Theorem 7.3.1 to build a matchgate M acting on lines j and k and such that $V = M$. □

With this theorem we gain the ability to express a certain kind of non-n.n. matchgates in the physical half of the system (those for which the corresponding Hamiltonian can be written in the form of eq. 8.2) as tensor components of a Gaussian operator in the Clifford subalgebra $\mathcal{C}'_{2n}$, hence the ability to efficiently (strongly) classically simulate them.

8.2.5 Allowable input states and observables

Notice, though, that the auxiliary part of the circuit will too, in general, change the input state of the circuit. If we want to predict the behaviour of the physical half of the system, we must choose a class of input states such that the evolution in the auxiliary half does not affect the physical half, *i.e.*, it does not introduce entanglement between the two subsystems. That is, we must restrict to input states of the form $|\phi\rangle \,\tilde{\otimes}\, |\psi\rangle$. In this way the expected value of any observable on the physical half of the output state will not be influenced by the auxiliary half.

Moreover, the method so far only works for product input states, and only for the Z observable on any physical line. But this can be extended to arbitrary (physical) states and arbitrary (physical) generic multi-line observables using Clifford operations as in the previous chapter.

Chapter 9

Conclusions

In this work we have shown that there are reasons to further investigate the matchgate formalism, and that it is possible to extend the set of efficiently simulatable matchgates beyond the nearest-neighbour family. We have found the following results as improvements of previous works on the topic [17]:

- a sufficient criterion to test whether a next-nearest-neighbour matchgate is efficiently simulatable;
- an algorithm to perform an exhaustive search to build simulatable matchgates;
- an explicit example of a simulatable next-nearest-neighbour matchgate found through an implementation of this algorithm in C language;
- a new framework to efficiently simulate matchgates acting on any arbitrary pair of lines provided their structure respects a particular form.

The computational gap between n.n. and n.n.n. matchgates is a stunning result, but it is not generally considered something easy to generalize to further investigation. In this work we have left many questions open.

Consider, as an example, Shor's algorithm. The key ingredient for this cornerstone of quantum computing is a unitary operator called *Quantum Fast Fourier Transform (QFFT)*. This operator has amazing properties which exemplify the power of quantum computing; it is indeed provable, as we should

expect, that representing this operator through matchgates would always involve n.n.n. matchgates. What if we could represent these matchgates as Gaussian operators? A possible future direction of research could be to apply the sufficient criterion from Chapter 8 to check these and other interesting quantum gates and see if they are classically efficiently simulatable.

Another open problem is to understand if the specially structured Hamiltonians in Section 8.2 do actually exist as possible Fermionic Hamiltonians for a quantum system and what do the associated unitary gates look like. If we find 'interesting' unitary operators whose Hamiltonians are of that form, then we could classically simulate them by just doubling the number of qubits (that is, the memory requirement of the simulation). As far as we know, this approach has never been tried.

Finally, using modified versions of our algorithm, it could be interesting to actively look for new simulatable n.n.n. matchgates, and to see if these matchgates do perform interesting computations.

Bibliography

[1] Aharonov D., A Simple Proof that Toffoli and Hadamard are Quantum Universal, 9 Jan 2003.
arXiv:quant-ph/0301040v1 [quant-ph]

[2] Atiyah M. F., McDonald I. G., *Introduction to Commutative Algebra*, Westview Press, 1994.

[3] Bernstein E., Vazirani U., Quantum complexity theory, *Special issue on Quantum Computation of the SIAM Journal of Computing*, Oct. 1997.

[4] Brezis H., *Analisi Funzionale*, Liguori Editore, 1986.

[5] Buhrman H., Quantum Computing and Communication Complexity, *Bulletin of the European Association for Theoretical Computer Science (EATCS)* **70**, pp. 131–141, February 2000.

[6] Chuang I. L., Nielsen M. A., *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.

[7] Cirac J. I., Verstraete F., Mapping local Hamiltonians of fermions to local Hamiltonians of spins, *Journal of Statistical Mechanics: Theory and Experiment* **9**, 0509:P09012, 2005.
arXiv:cond-mat/0508353v3 [cond-mat.str-el]

[8] Cook S. A., The Complexity of Theorem Proving Procedures, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, ACM press, pp. 151–158, New York 1971.

[9] Dawson C. M., Nielsen M. A., The Solovay-Kitaev algorithm, *Quantum Information & Computation* **6**(1), pp. 81–95, 2006.
arXiv:quant-ph/0505030v2 [quant-ph].

[10] Di Vincenzo D. P., Smolin J., Results on two-bit gate design for quantum computers, *Proceedings of the Workshop on Physics and Computation*,

PhysComp '94, IEEE Comput. Soc. Press, p. 14, Los Alamitos, CA, 1994.

[11] Di Vincenzo D. P., Terhal B. M., Adaptive Quantum Computation, Constant Depth Quantum Circuits and Arthur-Merlin Games, *Quantum Information & Computation* **4**(2), p. 134, 2004.
arXiv:0205133v6 [quant-ph]

[12] Di Vincenzo D. P., Terhal B. M., Classical simulation of noninteracting-fermion quantum circuits, *Physical Review A* **65**(3), 032325/1-10, 2002.

[13] Hall B. C., *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*, Springer, 2003.

[14] Impagliazzo R., Wigderson A., P = BPP unless E has Subexponential Circuits: Derandomizing the XOR Lemma, *Proceedings of the 29th annual ACM Symposium on Theory of Computing*, ACM Press, pp. 220-229, New York 1997.

[15] Jozsa R., On the simulation of quantum circuits, 19 Mar 2006.
arXiv:0603163v1 [quant-ph]

[16] Jozsa R., Kraus B., Miyake A., Watrous J., Matchgate and space-bounded quantum computations are equivalent, *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences* **466**, pp. 809–830, 2010.
arXiv:0908.1467v2 [quant-ph]

[17] Jozsa R, Miyake A., Matchgates and classical simulation of quantum circuits, *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences* **464**, pp. 3089–3106, 2008.
arXiv:0804.4050v2 [quant-ph]

[18] Ladner R., On the Structure of Polynomial Time Reducibility, *Journal of the ACM* **22**(1), ACM press, New York 1975.

[19] Markov I. L., Shi Y., Simulating quantum computation by contracting tensor networks, *SIAM Journal on Computing* **38**(3), pp. 963–981, 2008.
arXiv:0511069v7 [quant-ph]

[20] Nielsen M. A., The Fermionic canonical commutation relations and the Jordan-Wigner transform, 29 July 2005.
www.michaelnielsen.org

[21] Papadimitriou C., *Computational Complexity*, Addison Wesley, 1994.

[22] Sakurai J. J., *Meccanica quantistica moderna*, Zanichelli, 1996.

[23] Schoof R., Four primality testing algorithms, *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography*, Cambridge University Press.

[24] Shor P. W., Progress in Quantum Algorithms, *Quantum Information & Processing* **3**, pp. 1–5, October 2004.

[25] Valiant L., Quantum circuits that can be simulated classically in polynomial time, *SIAM Journal on Computing* **31**(4), pp. 1229–1254, 2002.

[26] Von Neumann J., *Mathematical Foundations of Quantum Mechanics*, Princeton University Press, 1996.

[27] Watrous J., Quantum Computational Complexity, *Encyclopedia of Complexity and System Science*, Springer, 2009.
arXiv:0804.3401v1 [quant-ph]

[28] Widgerson A., P, NP and mathematics - a computational complexity perspective, *Proceedings of the ICM Madrid '06* vol. 1, EMS Publishing House, pp. 665–712, Zurich 2007.