Programa de Doctorado "Matemáticas"

PhD Dissertation

Quantum Algorithms for the Combinatorial Invariants of Numerical Semigroups

Author
Joaquín
Ossorio Castillo

Supervisor
Prof. Dr. *José María*
Tornero Sánchez

March 10, 2019

A mi abuela María

Agradecimientos

Hay varias personas a las que me gustaría agradecer su contribución, directa o indirecta, a esta tesis doctoral.

En primer lugar, a José María Tornero, por aceptarme como discípulo allá por 2013 y por abrirme las puertas no solo de la investigación sino también de su casa (aunque solo fuese una vez, pero bueno algo es algo). Por todas las conversaciones y discusiones interesantes que hemos tenido a lo largo de estos años (algunas de ellas incluso sobre matemáticas), y por adaptarse siempre a mis caóticos ritmos de trabajo y a quedar para tomar algo los domingos a las 9 de la mañana habiendo avisado a las 8:55 tras tres meses sin dar señales de vida (es posible que esté siendo un poco exagerado, aunque por ahí debe andar la cosa). Soy consciente de que esta tesis doctoral nunca habría sido posible con otro director. O sí, quién sabe, ya casi nada me sorprende, aunque lo veo bastante improbable. Ah, por cierto, he encontrado una excusa perfecta para que sigamos trabajando juntos después de esto, no te vas a librar de mí tan fácilmente. Pero primero vamos a lo que vamos... *Ready to save the world again?*

A Julio González y a Fran Pena, de la Universidad de Santiago de Compostela, por acogerme en su grupo de investigación durante mis primeros años en Galicia, y por todo lo que he aprendido junto a ellos de optimización matemática y de computación cuántica adiabática. Al principio de esta aventura ambas ramas me eran completamente desconocidas, y al final terminaron siendo una parte fundamental de esta tesis doctoral. A mis compañeros del ITMATI, y en especial a Diego y a Gabriel, por su infinita paciencia a la hora de enseñarme a desarrollar de verdad (vamos, a programar en `linux` y a usar `git`). También me gustaría agradecer de nuevo a Fran y al resto de miembros del tribunal, Jorge Ramírez Alfonsín, Jesús Soto, Fernando Sancho e Ignacio Ojeda, por acceder a evaluar esta tesis y sobre todo por darme la oportunidad de discutir personalmente con ellos los resultados de ésta.

Echando la vista atrás, las matemáticas no siempre me gustaron, e incluso llegué a aborrecerlas en algún momento de mi vida. Me gustaría agradecer a dos grandes (y eternos) amigos, Rafa y Juan Pablo, las horas que dedicaron durante el instituto a ayudarme a sacar esta materia adelante. Gracias a ellos descubrí durante aquellos años tan importantes que las matemáticas no solo no son difíciles, sino que son hermosas. Una lástima que acabaran como ingenieros...

A lo largo de mi época universitaria y laboral me perdí tres veces, y tres veces me volví a encontrar. Gracias Rosa, debido a tu testimonio y a tus consejos fui capaz de adentrarme a estudiar matemáticas cuando ya llevaba cuatro años en la universidad, una decisión que en su día fue difícil pero de la que jamás me he arrepentido y de la que jamás me arrepentiré. Gracias Maca, por compartir tu experiencia conmigo y enseñarme lo bonita que es la investigación, y también por hacer de *celestina* entre el Sr. Tornero y yo (aunque supongo que él tendrá una opinión diferente sobre si agradecerte esto o no...). Y gracias Manu, por mostrarme el camino para llegar a Santiago, y por acogerme como a un hermano durante aquellos primeros meses de adaptación a la lluvia y a los churros fríos.

Al resto de amigos matemáticos que tengo repartidos por el mundo, pero en especial a los que siguen siendo una parte importante de mi vida y que espero sigan ahí siempre. Ale, Fran, Jara, Jesús, José Luis, Manolo, Pedro (y Maca y Manu, que ya los mencioné antes pero que también merecen estar aquí), gracias por enseñarme durante aquellos maravillosos años el verdadero significado de la universidad.

A mis padres, Joaquín y Ana, y a mi hermano, José María, porque han estado ahí en mis buenas y malas decisiones y me han enseñado a trabajar duro y a sacrificarme por mis objetivos. Equivocarse siempre ha sido mucho más fácil sabiendo que ellos estarían detrás para levantarme. A mi querida mujer, Leti, a quien conocí al poco de empezar esta tesis doctoral y que la ha vivido de cerca durante casi su totalidad (pobrecita... seguro que estaréis pensando). Sin tu apoyo todo esto habría sido mucho más difícil; gracias por estar a mi lado, y por querer compartir los buenos y malos momentos conmigo. Soy el que soy gracias a vosotros; siempre habéis creído en mí y me habéis apoyado, incluso cuando yo no lo hacía. Esta tesis doctoral es tan vuestra como mía. Un agradecimiento especial para mi perro, Bebo, por calentarme los pies mientras estaba trabajando en esto y por fingir tener que bajar a mear cada vez que veía que llevaba demasiadas horas sentado en la silla.

Gracias a todos. :)

Santiago de Compostela, 10 de marzo de 2019

Contents

1

Introduction

"You have nothing to do but mention the quantum theory, and people will take your voice for the voice of science, and believe anything."
– George Bernard Shaw, *Geneva, a Fancied Page of History in Three Acts*

It was back in spring 2014 when the author of this doctoral dissertation was finishing its master thesis, whose main objective was the understanding of Peter W. Shor's most praised result, a quantum algorithm capable of factoring integers in polynomial time. During the development of this master thesis, me and my yet-to-be doctoral advisor studied the main aspects of quantum computing from a purely algebraic perspective. This research eventually evolved into a sufficiently thorough canvas capable of explaining the main aspects and features of the mentioned algorithm from within an undergraduate context.

Just after its conclusion, we seated down and elaborated a research plan for a future Ph.D. thesis, which would expectantly involve quantum computing but also a branch of algebra whose apparently innocent definitions hide some really hard problems from a computational perspective: the theory of numerical semigroups. As will be seen later, the definition of numerical semigroup does not involve sophisticated knowledge from any somewhat obscure and distant branch of the tree of mathematics. Nonetheless, a number of combinatorial problems associated with these numerical semigroups are extremely hard to solve, even when the size of the input is relatively small. Some examples of these problems are the calculations of the Frobenius number, the Apéry set, and the Sylvester denumerant, all of them bearing the name of legendary mathematicians.

This thesis is the result of our multiple attempts to tackle those combinatorial problems with the help of a hypothetical quantum computer. First, Chapter 2 is devoted to numerical semigroups and computational complexity theory, and is divided into three sections. In Section 2.1, we give the formal definition of a numerical semigroup, along with a description of the main problems involved with them. In Section 2.2, we sketch the fundamental concepts of complexity theory, in order to understand the true significance within the inherent *hardness* concealed in the resolution of those problems. Finally, in Section 2.3 we prove the computational complexity of the problems we aim to solve.

Chapter 3 is the result of our outline of the theory of quantum computing. We give the basic definitions and concepts needed for understanding the particular place that quantum computers occupy in the world of Turing machines, and also the main elements that compose this particular model of computation: quantum bits and quantum entanglement. We also explain the two most common models of quantum computation, namely quantum circuits and adiabatic quantum computers. For all of them we give mathematical definitions, but always having in mind the physical experiments from which they stemmed.

Chapter 4 is also about quantum computing, but from an algorithmical perspective. We present the most important quantum algorithms to date in a standardized way, explaining their context, their impact and consequences, while giving a mathematical proof of their correctness and worked-out examples. We begin with the early algorithms of Deutsch, Deutsch-Jozsa, and Simon, and then proceed to explain their importance in the dawn of quantum computation. Then, we describe the major landmarks: Shor's factoring, Grover's search, and quantum counting.

Chapter 5 is the culmination of all previously explained concepts, as it includes the description of various quantum algorithms capable of solving the main problems inside the branch of numerical semigrops. We present quantum circuit algorithms for the Sylvester denumerant and the numerical semigroup membership, and adiabatic quantum algorithms for the Apéry Set and the Frobenius problem. We also describe a `C++` library called `numsem`, specially developed within the context of this doctoral thesis and which helps us to study the computational hardness of all previously explained problems from a classical perspective.

This thesis is intended to be autoconclusive at least in the main branches of mathematics in which it is *supported*; that is to say numerical semigroups, computational complexity theory, and quantum computation. Nevertheless, for the majority of concepts explained here we give references for the interested reader that wants to delve more into them.

2

Numerical Semigroups and the Frobenius problem

"One Ring to rule them all, One Ring to find them,
One Ring to bring them all and in the darkness bind them
In the Land of Mordor where the Shadows lie."
– J. R. R. Tolkien, *The Lord of the Rings*

2.1 Numerical Semigroups

The study of numerical semigroups has its origins at the end of the 19th Century, when English mathematician James Joseph Sylvester (1814 – 1897) and German mathematician Ferdinand Georg Frobenius (1849 – 1917) were both interested in what is now known as the Frobenius problem, which we proceed to enunciate.

Definition 2.1.1. *Let $a_1, a_2, \dots, a_n \in \mathbb{Z}_{\geq 0}$ with $\gcd(a_1, a_2, \dots, a_n) = 1$, the Frobenius problem, or* FP, *is the problem of finding the largest positive integer that cannot be expressed as an integer conical combination of these numbers, i.e., as a sum*

$$\sum_{j=1}^{n} \lambda_i a_i \text{ with } \lambda_i \in \mathbb{Z}_{\geq 0}.$$

This problem, so easy to state, can be extremely complicated to solve in the majority of cases, as will be seen in Section 2.3. It can be found in a wide variety of contexts, being the most famous the problem of finding the largest amount of money which cannot be obtained with a certain set of coins: if, for example, we have an unlimited amount of coins of 2 and 5 units, we can represent any quantity except 1 and 3.

In order to understand the relationship between this problem and numerical semigroups, we shall first define the latter.

Definition 2.1.2. *A semigroup is a pair $(S, +)$, where S is a set and $+$ is a binary operation $+ : S \times S \to S$ that is associative.*

Definition 2.1.3. *A numerical semigroup S is a subset of the non-negative integers $\mathbb{Z}_{\geq 0}$ which is closed under addition, contains the identity element 0, and has a finite complement in $\mathbb{Z}_{\geq 0}$.*

From now on, we shall denote numerical semigroups as S, taking for granted that they are commutative and that their associated operation is the addition. As it can be easily noted, a numerical semigroup is trivially a semigroup. In other words, a numerical semigroup is a semigroup that, additionally, is a monoid (i.e., it also has an identity element) and has finite complement in $\mathbb{Z}_{\geq 0}$. In order to work with numerical semigroups, it will be necessary to characterize them somehow. For that, let us set forth the following lemma.

Lemma 2.1.4. *Let $A = \{a_1, ..., a_n\}$ be a nonempty subset of $\mathbb{Z}_{\geq 0}$. Then,*

$$S = \langle A \rangle = \langle a_1, ..., a_n \rangle = \{\lambda_1 a_1 + ... + \lambda_n a_n \mid \lambda_i \in \mathbb{Z}_{\geq 0}\}$$

is a numerical semigroup if and only if $\gcd(a_1, ..., a_n) = 1$.

Proof. Assume S to be a numerical semigroup with $\gcd(a_1, \ldots, a_n) = d$ and let $x = \max(\mathbb{Z}_{\geq 0} \setminus S)$ —as $\mathbb{Z}_{\geq 0} \setminus S$ is finite, it has a maximum—. If $s \in S$, then $d|s$. As $x+1 \in S$ and $x+2 \in S$, then $d \mid (x+1)$ and $d \mid (x+2)$, which implies that $d = 1$.

Let $S = \langle A \rangle = \langle a_1, ..., a_n \rangle$ with $\gcd(a_1, \ldots, a_n) = 1$, then by Bézout's identity [23] there exist integers $x_1, x_2, \ldots, x_n \in \mathbb{Z}$ such that $a_1x_1 + a_2x_2 + \cdots + a_nx_n = 1$. If $a_1x_1 + \cdots + a_nx_n = p + q$, where p contains the positive terms and q the negative ones, then $p = 1 - q$. We can define $r = -q$ and, as $p \in S$, $r \in S$ and $r + 1 \in S$. Let $s \geq (r-1)r + (r-1)$, and let t and u such that $s = tr + u$ with $0 \leq u < r$. It follows that $u \leq r - 1 \leq t$ and finally we have that $s = u(r+1) + (t-u)r \in S$. Thus, S has finite complement in $\mathbb{Z}_{\geq 0}$ and consequently is a numerical semigroup. □

The previous lemma tells us that, drawing from a set $A \subseteq \mathbb{Z}_{\geq 0}$, it is possible to generate a semigroup $S = \langle A \rangle$ as long as the elements of A satisfy a certain condition. In this context, any set A such that $S = \langle A \rangle$ for a certain numerical semigroup S is called a system of generators of S. Even more, it can be proved that any numerical semigroup can be expressed that way, as will be shown next.

Theorem 2.1.5. *Every numerical semigroup S admits a unique minimal system of generators, which can be calculated as $S^* \setminus (S^* + S^*)$ with $S^* = S \setminus \{0\}$.*

Proof. Let $S^* = S \setminus \{0\}$ and let $s \in S^*$. If $s \notin S^* \setminus (S^* + S^*)$, then $s \in S^* + S^*$ and there exist $t, u \in S^*$ such that $s = t + u$. As $t, u < s$, we can repeat this procedure a finite number of steps until we find $s_1, \ldots, s_n \in S^* \setminus (S^* + S^*)$ with $s = s_1 + \cdots + s_n$. Thus, we can conclude that $S^* \setminus (S^* + S^*)$ is a system of generators of S.

Besides, let A be a system of generators of S and let $s \in S^* \setminus (S^* + S^*)$, then there exist $t \in \mathbb{Z}_{>0}$, $\lambda_1, \ldots, \lambda_n \in \mathbb{Z}_{\geq 0}$ and $a_1, \ldots, a_n \in A$ such that $s = \lambda_1 a_1 + \cdots + \lambda_n a_n$. As $s \notin S^* + S^*$, we have that $s = a_i$ for a certain $i \in \{1, \ldots, n\}$ and thus $S^* \setminus (S^* + S^*)$ is contained in any system of generators of S. □

Corollary 2.1.6. *Let S be a numerical semigroup generated by $A = \{a_1, \ldots, a_n\}$ with $0 \neq a_1 < a_2 < ... < a_n$. Then, A is a minimal system of generators of S if and only if $a_{i+1} \notin \langle a_1, a_2, \ldots, a_i \rangle$, for all $i \in \{1, \ldots, n-1\}$.*

It will be proven later in this section that the minimal system of generators of a numerical semigroup is in fact finite. Hereinafter, if we say that $S = \langle A \rangle$ with

$A = \{a_1, a_2, \ldots, a_n\}$ is a numerical semigroup, then we shall assume without loss of generality that $a_1 < a_2 < \cdots < a_n$, $\gcd(a_1, a_2, \ldots, a_n) = 1$, and that A is the minimal system of generators of S. Some examples of numerical semigroups, which will be used for the rest of the section, are:

$$\begin{aligned}
S_a &= \langle 3, 7\rangle = \{0, 3, 6, 7, 9, 10, 12, \rightarrow\} \\
S_b &= \langle 4, 9\rangle = \{0, 4, 8, 9, 12, 13, 16, 17, 18, 20, 21, 22, 24, \rightarrow\} \\
S_c &= \langle 5, 8, 11\rangle = \{0, 5, 8, 10, 11, 13, 15, 16, 18, \rightarrow\} \\
S_d &= \langle 5, 7, 9\rangle = \{0, 5, 7, 9, 10, 12, 14, \rightarrow\}
\end{aligned}$$

Where $\rightarrow$ means that all integers thenceforth are also included in the numerical semigroup. Having thus defined and characterized what numerical semigroups are, we proceed to describe some of their combinatorial invariants.

Definition 2.1.7. *Let $S = \langle a_1, a_2, \ldots, a_n\rangle$ be a numerical semigroup, then*

$$m(S) = a_1 \text{ and } e(S) = n$$

are called respectively the multiplicity of S and the embedding dimension of S.

Lemma 2.1.8. *Let S be a numerical semigroup, then*

$$m(S) = \min S^*$$

Definition 2.1.9. *The set of gaps of a numerical semigroup S is defined as*

$$G(S) = \mathbb{Z}_{\geq 0} \backslash S.$$

Its cardinal,

$$g(S) = |G(S)|,$$

is called the genus of S; and its maximum,

$$f(S) = \max G(S),$$

is called the Frobenius number of S.

In other words, as by definition any numerical semigroup S has a finite complement in $\mathbb{Z}_{\geq 0}$, we can define the maximum of such complement as $f(S)$, known as the Frobenius number. In fact, the Frobenius problem described at the beginning of this chapter in Definition 2.1.1 can be enunciated as the problem of finding $f(S)$ for a certain numerical semigroup S. We shall expand the concepts related to the difficulties that surround the calculation of the Frobenius number in Section 2.3. Table 2.1 shows the combinatorial invariants associated for the previously given examples of semigroups.

Greek-French mathematician Roger Apéry (1916 – 1994), better known for proving in 1979 the irrationality of $\zeta(3)$ [11], also laid the background in the context of the resolution of singularities of curves [10] for an important set associated to a numerical semigroup S and one of its elements.

S	A	$m(S)$	$e(S)$	$G(S)$	$g(S)$	$f(S)$
S_a	$\langle 3,7\rangle$	3	2	$\{1,2,4,5,8,11\}$	6	11
S_b	$\langle 4,9\rangle$	4	2	$\{1,2,3,5,6,7,10,11,14,15,19,23\}$	12	23
S_c	$\langle 5,8,11\rangle$	5	3	$\{1,2,3,4,6,7,9,12,14,17\}$	10	17
S_d	$\langle 5,7,9\rangle$	5	3	$\{1,2,3,4,6,8,11,13\}$	8	13

Table 2.1: Combinatorial invariants for some examples of semigroups

Definition 2.1.10. *The Apéry set of a numerical semigroup S with respect to a certain $s \in S^*$ is defined as*

$$Ap(S,s) = \{x \in S \mid x - s \notin S\}.$$

A possible characterization of the elements of the Apéry set one by one, which will come to special use in Chapter 5, is given by the following lemma.

Lemma 2.1.11. *Let S be a numerical semigroup and let $s \in S^*$. Then, $Ap(S,s) = \{\omega_0, \omega_1, ..., \omega_{s-1}\}$ where ω_i is the least element of S congruent with i modulo s, for all $i \in \{0, ..., s-1\}$. Consequently, $|Ap(S,s)| = s$.*

Proof. Let $x \in Ap(S,s)$. Then, by definition, $x \in S$ and $x - s \notin S$. As $x \equiv i \mod s$ for a certain $i \in \{0, \dots, s-1\}$, it follows that $x = \omega_i$. As a result, $Ap(S,s) \subseteq \{\omega_0, \dots, \omega_{s-1}\}$.

Also, let $\omega_i = \min\{x \in S \mid x \equiv i \mod s\}$, it is clear that $\omega_i \in S$ and $\omega_i - s \notin S$. Thus, $\{\omega_0, \dots, \omega_{s-1}\} \subseteq Ap(S,s)$. □

By means of an example on how to calculate the Apéry set of a semigroup with respect to a certain element, let

$$S_c = \langle 5,8,11\rangle = \{0,5,8,10,11,13,15,16,18,\rightarrow\}.$$

Then $Ap(S_c,5) = \{\omega_0, \dots, \omega_4\}$, where

$$\begin{aligned}
\omega_0 &= \min\{x \in S \mid x \equiv 0 \mod 5\} = 0\\
\omega_1 &= \min\{x \in S \mid x \equiv 1 \mod 5\} = 11\\
\omega_2 &= \min\{x \in S \mid x \equiv 2 \mod 5\} = 22\\
\omega_3 &= \min\{x \in S \mid x \equiv 3 \mod 5\} = 8\\
\omega_4 &= \min\{x \in S \mid x \equiv 4 \mod 5\} = 19
\end{aligned}$$

We proceed to prove some results via the properties of the Apéry set.

Proposition 2.1.12. *The minimal system of generators of a numerical semigroup S is finite.*

Proof. Let $s \in S^*$. Then, it is easy to see that for every $t \in S$ there exists a unique pair $(u,v) \in \mathbb{Z}_{\geq 0} \times Ap(S,s)$ such that $t = us + v$. Thus, S is generated by $A = Ap(S,s) \cup \{s\}$ and, as A is finite, the unique minimal system of generators must be finite. □

Lemma 2.1.13. *Let S be a numerical semigroup, then*

$$e(S) \leq m(S).$$

Proof. Let $a = m(S)$ and let $A = Ap(S,a) \cup \{a\}$. Thus, as S can be generated by $A \setminus \{0\}$ and $|A \setminus \{0\}| = a$, we can conclude that $e(S) \leq m(S)$. □

The Apéry set is noteworthy in the context of the Frobenius problem as there exists a relationship between its members (regardless of the element $s \in S^*$ we choose) and the Frobenius number, which we proceed to enunciate and prove.

Theorem 2.1.14. (A. Brauer – J. E. Shockley, 1962) [31] *Let S be a numerical semigroup and let $s \in S^*$. Then*

$$f(S) = \max Ap(S,s) - s$$

$$g(S) = \frac{1}{s}\left(\sum_{\omega \in Ap(S,s)} \omega\right) - \frac{s-1}{2}$$

Proof. As $Ap(S,s) = \{x \in S \mid x - s \notin S\}$, we have that $\max\{Ap(S,s)\} - s \notin S$. Let $y > \max\{Ap(S,s)\} - s$, then $y + s > \max\{Ap(S,s)\}$. Let $z \in Ap(S,s)$ such that $z \equiv y + s \mod s$, then as $z < y + s$ we have that $y = z + ks$ for some $k \geq 0$, hence $y - s = z + (k-1)s \in S$. This proves the first identity.

As for the genus of S, for every $w \in Ap(S,s)$, $w \equiv i \mod s$ with $i \in \{0, \ldots, s-1\}$ implies that there exists $k_i \in \mathbb{Z}_{\geq 0}$ such that $w = k_i s + i$. This way, we can write the elements of the Apéry Set as $\omega_i = k_i s + i$ for $i \in \{0, \ldots, s-1\}$.

Thus, as $x \equiv \omega_i \mod s$ implies that $x \in S$ if and only if $\omega_i \leq x$, then we can conclude that

$$g(S) = \sum_{i=1}^{s-1} k_i = \frac{1}{s}\sum_{i=1}^{s-1}(k_i s + i) - \frac{s-1}{2} = \frac{1}{s}\sum_{i=1}^{s-1}\omega_i - \frac{s-1}{2}.$$

□

Now we exemplify the relationship between numerical semigroups and combinatorial optimization, as one of the most important problems in the latter branch of mathematics, known as the knapsack problem or rucksack problem, and more concretely one of its variants [93] (p. 374), can be seen as the problem of deciding if a given integer t belongs to a certain numerical semigroup S.

Definition 2.1.15. *The numerical semigroup membership problem, or* NSMP*, is the problem of determining if, given a certain integer $t \in \mathbb{Z}_{\geq 0}$ and a numerical semigroup $S = \langle a_1, ..., a_n \rangle$, the integer t is contained in S. That is to say, if there exist non-negative integers $\lambda_1, \ldots, \lambda_n \in \mathbb{Z}_{\geq 0}$ such that*

$$\sum_{i=1}^{n} \lambda_i a_i = t.$$

Finally, we show another important problem in numerical semigroups related to the previous one. In the mid-19th century, J. J. Sylvester studied the number of partitions of an integer with respect to a certain subset of non-negative integers [109, 110]. In the context of this chapter, this problem can be seen as an extension of the NSMP, but rather than answering whether or not an integer is contained in a numerical semigroup, we go farther and want to calculate the number of distinct representations of that integer with respect to the minimal system of generators of the semigroup.

Definition 2.1.16. *The Sylvester denumerant of a non-negative integer $t \in \mathbb{Z}_{\geq 0}$ with respect to a numerical semigroup $S = \langle a_1, ..., a_n \rangle$, denoted by $d(t, S)$ or by $d(t; a_1, \dots, a_n)$, is defined as the number of solutions of the Diophantine equation*

$$\sum_{i=1}^{n} \lambda_i a_i = t,$$

where $\lambda_1, \dots, \lambda_n \in \mathbb{Z}_{\geq 0}$.

By means of an example, we recall the previously defined semigroup:

$$S_d = \langle 5, 7, 9 \rangle = \{0, 5, 7, 9, 10, 12, 14, \rightarrow\}$$

The Frobenius number of S_d is equal to $f(S_d) = 13$, which means that any integer greater than 13 is contained in the semigroup. However, the number of possible representations may differ between them. Number 15, for example, has a unique representation with respect to 5, 7 and 9:

$$15 = 3 \times (5) + 0 \times (7) + 0 \times (9),$$

while on the other hand, number 14 has two possible representations:

$$14 = 1 \times (5) + 0 \times (7) + 1 \times (9)$$

and

$$14 = 0 \times (5) + 2 \times (7) + 0 \times (9).$$

This tells us that $d(15; 5, 7, 9) = 1$ and $d(14; 5, 7, 9) = 2$.

The study of the Sylvester denumerant is of great importance in many branches of mathematics and, as can be easily deduced, is at least as hard to solve as the numerical semigroup membership problem. This result will be formally explained in Section 2.3.

A thorough study of numerical semigroups can be found in the book by J. C. Rosales and P. A. García-Sánchez [101]. The majority of problems and definitions surrounding the Frobenius number and the Sylvester denumerant can be found in the book by J. L. Ramírez-Alfonsín [98].

2.2 Computational Complexity Theory

The aim of this section is to provide a background on computational complexity theory, a branch of mathematics and computer science which focuses on the definition of classes of computational problems in accordance with their inherent difficulty, and on the relationships between those classes.

This framework will become useful for two of the main objectives of this Ph.D. thesis: first, for placing the Frobenius problem within the context of those problem classes —i.e., what does exactly mean that the Frobenius problem is hard to solve—; and second, for contextualizing the quantum model of computation inside the universal Turing machines used for standardizing the aforementioned classes of problems. In order to achieve that, we begin by defining two basic ingredients needed for this problem classification.

Definition 2.2.1. [62, 77] *A decision problem or recognition problem* Π *is a function with a one-bit output:* `YES` *or* `NO` *—i.e., a problem that can be posed as a yes-no question of the input—. To specify a decision problem, one must define the set* A *of possible inputs and the subset* $B \subseteq A$ *of* `YES` *instances.*

In the context of automata and abstract computers, it is necessary to define a new concept that generalizes the notion of a machine capable of solving a certain decision problem. We give from now on some essential notions of the so-called Turing machines, named after English computer scientist and mathematician Alan Turing (1912 – 1954) and its groundbreaking and dazzling work [111]. These machines may appear to be simple or limited, but they are capable of simulating any computer algorithm and of doing everything that a real computer can do [108]. More information on the subject can be found in [77].

Definition 2.2.2. *A deterministic Turing machine* $\mathcal{M}$*, or* DTM*, is defined by a triple* $\mathcal{M} = (S, \Sigma, \delta)$*, where* S *is a finite set of states (with an initial state* $s_0 \in S$ *and a set of final states* $F \subseteq S$*),* Σ *is an alphabet containing at least the blank symbol* $\#$*, and* $\delta : S \setminus F \times \Sigma \to S \times \Sigma \times \{L, R\}$ *is a partial function called the transition function.*

The definition of Turing machine is quite abstract, but it can be seen as a device that manipulates symbols on a strip of tape according to a table of rules. This tape is divided into cells, each of them containing a symbol from the alphabet Σ, and the machine has a head that reads or writes on the tape and moves the tape left or right. The machine follows a single and deterministic computation path starting in the state s_0 and ending in one of the final states. A representation of this path is shown in Figure 2.1.

Now that we have defined what a deterministic Turing machine is, we can start a classification of decision problems with respect to them.

Definition 2.2.3. *The class* P*, or* PTIME*, contains all decision problems solvable by a deterministic Turing machine using a polynomial amount of computation time with respect to the size of the input.*

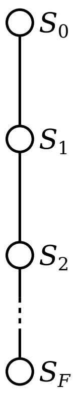


Figure 2.1: Example of computation of a deterministic Turing machine

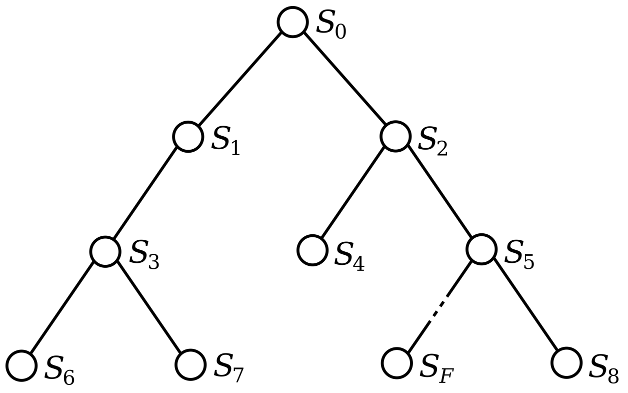


Figure 2.2: Example of computation of a non-deterministic Turing machine

The class P is known to contain many elemental problems in mathematics, like addition and multiplication, and other not so elemental such as the problem of deciding whether an integer is prime [7], or the resolution of a certain type of linear optimization problem [12, 73].

Another concept that generalizes the notion of a Turing machine one step further is the following:

Definition 2.2.4. *A non-deterministic Turing machine $\mathcal{M}$, or* NTM*, is defined by a triple $\mathcal{M} = (S, \Sigma, \delta)$, where S is a finite set of states (with an initial state $s_0 \in S$ and a set of final states $F \subseteq S$), Σ is an alphabet containing at least the blank symbol $\#$ and $\delta \subseteq (Q \setminus A \times \Sigma) \times (Q \times \Sigma \times \{L, R\})$ is a relation on states and symbols called the transition relation.*

As can be seen, the only difference between a deterministic Turing machine and a non-deterministic Turing machine is the substitution of the transition function for a transition relation. This distinction can be interpreted as a change in the computation path followed by the machine. While in the deterministic Turing machine the transition between a state and the next is deterministic and unique, in the non-deterministic Turing machine there can be more than one state after a certain step in the computation. The computation path may branch after a state, making an indeterminate number of copies of the machine which will continue working in parallel. Thus, instead of a computation path, the behavior of a non-deterministic

Turing machine is better seen as a computation tree, represented in Figure 2.2.

Although the concept of a non-deterministic Turing machine is more philosophical than realistic and has not been implemented in practice, there have been recent designs of a model for non-deterministic Turing machine that exploit the ability of DNA to replicate itself [40].

In the same way as with deterministic Turing machines, we can also classify all decision problems with respect to the non-deterministic Turing machines.

Definition 2.2.5. *The class* NP *contains all decision problems that can be solved by a non-deterministic Turing machine that runs in polynomial time with respect to the size of the input. If the answer is* `YES`*, then at least one computation path accepts. On the other hand, if the answer is* `NO`*, all computation paths must reject.*

Another interpretation of the set of decision problems that are contained in NP states that a problem is solvable by a non-deterministic Turing machine in polynomial time if and only if the problem of veryfing if a certain polynomial-size proof of this fact is correct is in P. In other words, the statements *verifiable in polynomial time by a deterministic Turing machine* and *solvable in polynomial time by a non-deterministic Turing machine* are totally equivalent, and a proof of this fact can be found in [108]. For example, the problem of finding a prime factor of a certain integer N, also called prime factorization, is not known to be in P. Nevertheless, as the problem of determining if a certain prime p is a factor of N can be checked in polynomial time, we know that prime factoring is in NP.

As a deterministic Turing machine is also trivially a non-deterministic Turing machine, it is easy to conclude that $\mathrm{P} \subseteq \mathrm{NP}$. However, the question of whether there exists a problem in NP that is not in P is one of the most important open problems in theoretical computer science —and in mathematics in general—, and is known as the P versus NP problem, or $\mathrm{P} \overset{?}{=} \mathrm{NP}$. In other words, does the non-determinism of a Turing machine make a difference? Intuitively, one might say it should, but that remains to be proven. Interesting surveys about this subject can be found in [107], [38] and [4].

Apart from all problems in P, a myriad of compelling problems in mathematics are also known to be in NP, such as the previously mentioned prime factorization, the Boolean satisfiability problem —also known as SAT— and many others. In fact, the SAT problem was the first to be proven to be complete for the class NP [37], a concept which we proceed to explain.

Definition 2.2.6. *Let* C *be a complexity class and let* Π *be a decision problem. Then, problem* Π *is said to be hard for the complexity class* C *under a given type of reduction if there exists a reduction —of the given type— from any problem in* C *to* Π*. The complexity class of all problems that are hard for* C *is called* C-hard.

Definition 2.2.7. *Let* C *be a complexity class and let* Π *be a decision problem. Then, problem* Π *is said to be complete for the complexity class* C *under a given type of reduction if* Π *is contained in* C *and* Π *is hard for* C *under that type of* *reduction. The complexity class of all problems that are complete for* C *is called* C-complete*.*

Both concepts and an example of what a reduction is will be thoroughly explained in the next section of the present chapter. Note that a polynomial-time deterministic algorithm for a NP-complete problem would violate the standard P $\subsetneq$ NP conjecture. Moreover, as all NP-complete problems can be seen as equally hard to solve, and they are the hardest problems inside NP, such a result would give us a polynomial-time deterministic algorithm for every problem in NP.

Before proceeding to the next section, let us explain another variation of the notion of Turing machine, which will be specially useful in Chapter 3 for explaining the concept of universal quantum computer, with its corresponding complexity class.

Definition 2.2.8. *A probabilistic Turing machine $\mathcal{M}$, or* PTM*, is defined by a triple $\mathcal{M} = (S, \Sigma, \delta)$, where S is a finite set of states (with an initial state $s_0 \in S$ and a set of final states $F \subseteq S$), where Σ is an alphabet containing at least the blank symbol # and where δ is the transition function. In this case, however, the transition function does not define deterministic transitions as in the case of a deterministic Turing machine, but gives a probability distribution of possible transitions according to $\delta : S \times \Sigma \times S \times \Sigma \times \{L, R\} \to [0, 1]$.*

The behavior of a probabilistic Turing machine can be seen as that of a non-deterministic Turing machine where, instead of multiplying itself when more than one path is possible from a certain state, it decides which path to follow according to a probability distribution. Thus, we have a computation path that can be seen as superimposed over the computation tree of a non-deterministic Turing machine, as highlighted in red in Figure 2.3.

Definition 2.2.9. *The class* BPP*, which stands for bounded-error probabilistic polynomial-time, contains all decision problems that can be solved in polynomial time by probabilistic Turing machines with error probability bounded by 1/3 for all inputs.*

The class BPP can also be seen as the class of decision problems solvable by a non-deterministic Turing machine such that, if the answer is `YES`, then at least 2/3 of the computation paths accept the input, and if the answer is `NO`, then at most 1/3 of the computation paths accept. Nonetheless, although it is easy to prove that P $\subseteq$ BPP, there is no known relation between BPP and NP. Note also that the choice of 1/3 is arbitrary, as a swap between 1/3 and any $x \in \mathbb{R}$ such that $0 < x < 1/2$ in the definition will keep unchanged the contents of the class BPP. For more details on BPP, we refer to [57], where it was first defined.

Finally, we define two complexity classes that will help us in the understanding of the bigger picture where all of this is contained.

Definition 2.2.10. *The class* PP*, which stands for probabilistic polynomial-time, contains all decision problems solvable by a non-deterministic Turing machine in polynomial time such that, if the answer is* `YES`*, then at least 1/2 of computation paths accept and, if the answer is* `NO`*, then less than 1/2 of computation paths accept.*

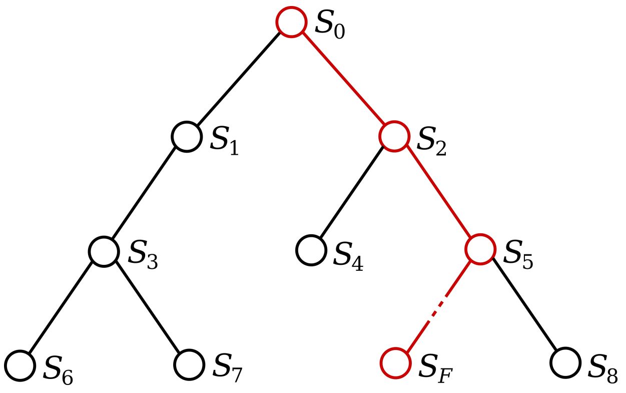


Figure 2.3: Example of computation of probabilistic Turing machine

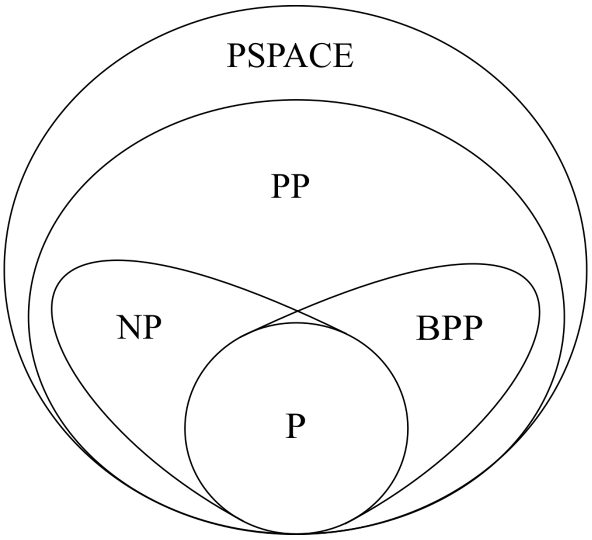


Figure 2.4: Diagram of complexity classes

Definition 2.2.11. *The class* PSPACE*, which stands for polynomial space, contains all decision problems solvable by a deterministic Turing machine using a polynomial amount of space, regardless of the total time needed.*

The complexity classes we have defined here form a hierarchical structure [4], which can be seen in Figure 2.4. As can be noted, PSPACE contains all of them, a quite surprising result. In fact, to date nobody has proven that P $\subsetneq$ PSPACE, meaning that the full hierarchy may collapse in the future if P equals PSPACE. In Figure 2.5 we also show the relationship between P, NP and the complexity classes that surround them, depending on the answer to the P versus NP problem. In the left diagram, NP is represented by the circle, not by the space between NP-complete and P —that space corresponds to NP-intermediate, a class that will be explained in Section 4.5—.

Now that we have formed the basis for the understanding of the main computational complexity classes, let us show where does some of the main problems introduced in the section dedicated to numerical semigroups fit into all this. For an almost complete and updated list of complexity classes, we refer to the vast database found in [5], result of the efforts of American theoretical computer scientist Scott

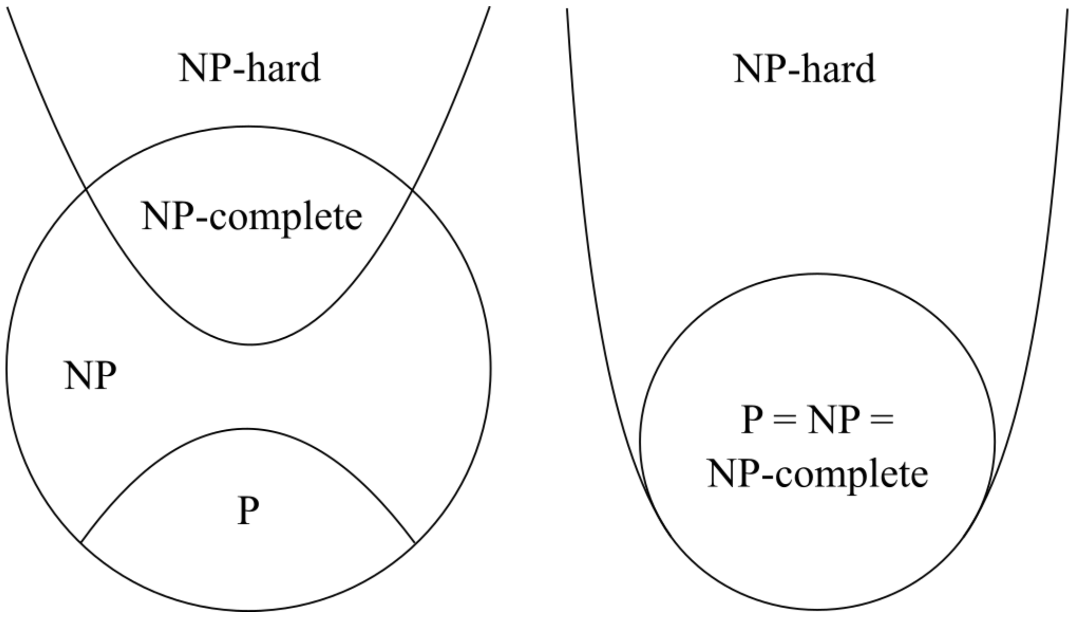


Figure 2.5: Diagram of P and NP-related complexity classes

Aaronson (b. 1981).

2.3 Computational Complexity of the Frobenius Problem and the NSMP

In this section we are going to apply the background studied up to this point for proving that the Frobenius problem is in NP-hard. For that, first we are going to prove that the Boolean satisfiability problem is complete for the complexity class NP —i.e., that it is in NP, and is also hard for NP—. Then, we show that the numerical semigroup membership problem is also in NP-complete. Finally, we prove that the Frobenius problem is hard for the class NP under Turing reductions to the NSMP.

Definition 2.3.1. *The Boolean satisfiability problem, or* SAT*, is the problem of determining if there exists an interpretation for the binary variables $x_1, x_2, \ldots, x_n$ that satisfies a given Boolean formula. In other words, given m clauses $C_1, C_2, \ldots, C_m$, is the propositional logic formula $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ satisfiable?*

Definition 2.3.2. *Let Π and Π' be two decision problems, a polynomial-time many-one reduction from Π to Π' is a polynomial-time algorithm for transforming inputs to problem Π into inputs to problem Π', such that the transformed problem has the same output as the original problem.*

Theorem 2.3.3. (S. Cook – L. Levin) [37, 83] *The Boolean satisfiability problem is in* NP-complete*.*

Proof. In order to prove that the SAT problem is in NP-complete, we must demostrate two statements: first, that the SAT problem is in NP; second, that it is in NP-hard.

It is easy to see that the first affirmation holds. Let $C_1, C_2, \ldots, C_m$ be a set of logical clauses involving the binary variables $x_1, x_2, \ldots, x_n$, a certificate for the veracity of all the clauses would be a vector in $\{0,1\}^n$ representing a truth assignement. Naturally, a deterministic Turing machine that runs in polynomial time exists for checking that our certificate makes every one of the clauses true.

The second part is customarily accomplished by showing that a well known NP-complete problem can be polynomially transformed to our problem; in this case, SAT. However, SAT was the first problem to be proven to be in NP-complete, how was this accomplished? This result comes originally from the separated works of American-Canadian computer scientist and mathematician Stephen Cook (b. 1939) and Soviet-American computer scientist Leonid Levin (b. 1948), as both of them proved that every problem in NP can be reduced in polynomial time to SAT [37, 83]. The proof we present here of Cook-Levin theorem is based on the ones found in [55] and [93].

We should relate somehow all problems in NP by a common characteristic. As defined previously, a decision problem is in NP if there exists a non-deterministic Turing machine $\mathcal{M}$ that accepts or rejects an instance of the problem of size n in time $p(n)$, where p is a polynomial function. This NTM is represented as $\mathcal{M} = (S, \Sigma, \delta)$, where S is the set of states, $s_0 \in S$ is the initial state, $F \subseteq S$ is the set of final states, and δ is the transition relation.

The idea behind the proof is to transform any NTM that solves a problem in NP in polynomial time to an instance of the SAT problem. First, let us define the following Boolean variables:

1. $T_{i,j,k} \sim$ True if tape cell i of our NTM contains symbol j at step k of the computation.
2. $H_{i,k} \sim$ True if the read/write head of our NTM is at tape cell i at the step k of the computation.
3. $S_{s,k} \sim$ True if our NTM is in the state $s \in S$ at the step k of the computation.

Note that there are $\mathcal{O}(p(n))$ variables of type $S_{s,k}$ and $\mathcal{O}(p(n)^2)$ variables of type $T_{i,j,k}$ and $H_{i,k}$. Now we can define a set of Boolean clauses which will represent all the conditions required for the NTM to be able to solve the problem in time less or equal than $p(n)$. We can assume then that we have $-p(n) \leq i \leq p(n)$ tape cells, a fixed number $|\Sigma|$ of symbols, and that the computation ends in at most $p(n)$ steps; i.e., $0 \leq k \leq p(n)$.

1. $T_{i,j,0} \sim$ True if tape cell i contains symbol j at step $k = 0$; i.e., at the initial state of the computation.
2. $\neg T_{i,j,k} \vee \neg T_{i,j',k}, \forall j \neq j' \sim$ True if there is at most one symbol per tape cell at any step k.

3. $\bigvee_{j\in\Sigma} T_{i,j,k} \sim$ True if there is at least one symbol per tape cell at any step k.

4. $S_{s_0,0} \sim$ True if the initial state $s_0 \in S$ is the state of the computation at $k = 0$.

5. $\neg S_{s,k} \vee \neg S_{s',k}, \forall s \neq s' \sim$ True if our NTM is at only one state at any step k.

6. $H_{0,0} \sim$ True if the initial position of the read/write head of our NTM is in tape cell $i = 0$.

7. $\neg H_{i,k} \vee \neg H_{i',k}, \forall i \neq i' \sim$ True if the read/write head of our NTM is at only one tape cell at any step k.

8. $T_{i,j,k} \wedge T_{i,j',k+1} \to H_{i,j}, \forall j \neq j' \sim$ True if tape cell i remains unchanged unless it has been written.

9. $(H_{i,k} \wedge S_{s,k} \wedge T_{i,\sigma,k}) \to \bigvee_{((s,\sigma),(s',\sigma',d))\in\delta} (H_{i+d,k+1} \wedge S_{s',k+1} \wedge T_{i,\sigma',k+1}), \forall k < p(n) \sim$ True for all possible transitions at a computation step $k \leq p(n)$ when head is at position i.

10. $\bigvee_{0\leq k\leq p(n)} \bigvee_{f\in F} S_{f,k} \sim$ True if the machine ends in one of the final states $f \in F$ in a step $k \leq p(n)$.

It is easy to see that there are $\mathcal{O}(1)$ clauses of type 4, 6 and 10; $\mathcal{O}(p(n))$ clauses of type 1 and 5; $\mathcal{O}(p(n)^2)$ clauses of type 2, 3, 8 and 9; and $\mathcal{O}(p(n)^3)$ clauses of type 7. This way, we can deduce that if there is an accepting computation for our NTM on a certain input, then the conjunction of all previously defined clauses is satistifiable just by asigning the variables their corresponding value during the computation. Furthermore, if we solve the instance of SAT related to the conjunction of those clauses, we can find a computation for $\mathcal{M}$ that follows the solution for the Boolean variables.

As there are $\mathcal{O}(p(n))$ variables and $\mathcal{O}(p(n)^3)$ clauses, the size of our SAT problem is $\mathcal{O}(\log(p(n))p(n)^3)$. Consequently, we have found a polynomial-time many-one reduction from any problem in NP to SAT, thus proving that SAT is in fact complete for the class NP.

□

Note that the previous theorem does not only show the existence of problems in NP-complete, but also states that if we find a polynomial time classical algorithm for the SAT problem, we can also find a polynomial time algorithm for every problem in NP. However, this will unlikely happen, as it would imply that P = NP.

Now it is time to prove that the numerical semigroup membership problem is in NP-complete. A possible way of achieving this is seeing that, as said in Section 2.2, NP-complete = NP ∩ NP-hard. In order to prove that a problem is in NP, it suffices to show that a candidate solution is verifiable in polynomial time, which in this case is trivial. For proving the NP-hardness of the NSMP, we must give first the following definitions, which can also be found in [59] and [93].

Definition 2.3.4. *Let Π and Π' be two decision problems, a polynomial time Turing reduction from Π to Π' is an algorithm Λ_Π which solves Π by using a hypothetical subroutine $\Lambda_{\Pi'}$ for solving Π', such that, if $\Lambda_{\Pi'}$ were a polynomial time algorithm for Π' then Λ_Π would be a polynomial time algorithm for Π. In this context, we say that the decision problem Π can be Turing reduced to Π'.*

Proposition 2.3.5. *Let Π be a decision problem, then Π is* NP-hard *if there exists an* NP-complete *decision problem Π' such that Π' can be Turing reduced to Π.*

Definition 2.3.6. *Let U, V and W be sets such that $|U| = |V| = |W|$, and $T \subseteq U \times V \times W$. Then, the 3-dimensonal matching problem, or* 3DM, *is the problem of determining if there exists $M \subseteq T$ with $|M| = |U|$ such that*

$$(u, v, w) \neq (u', v', w') \iff u \neq u', v \neq v', w \neq w'$$

for all $(u, v, w), (u', v', w') \in M$.

Definition 2.3.7. *Let $F = \{S_1, \dots, S_n\}$, where $S_i \subseteq S = \{u_1, \dots, u_{3m}\}$ and $|S_i| = 3$ for all $i \in \{1, \dots, n\}$. Then, the 3-exact cover problem, or* 3EC, *is the problem of determining if there is a subfamily of m subsets that covers S.*

Definition 2.3.8. *Let $c_1, \dots, c_n \in \mathbb{Z}_{\geq 0}$ and $k \in \mathbb{Z}_{\geq 0}$. Then, the 0-1 knapsack problem, or* 0-1KP, *is the problem of determining if there exists $S \subseteq \{1, \dots, n\}$ such that*

$$\sum_{j \in S} c_j = k.$$

Theorem 2.3.9. *The numerical semigroup membership problem is in* NP-complete.

Proof. [93] It is trivial to see that 3DM, 3EC and 0-1KP are in NP. What remains to be seen is that all of them are in NP-complete, and also that the NSMP transforms to one of them. First, we are going to prove that 3DM is in NP-complete, giving a polynomial transformation of an instance of SAT to an instance of 3DM. Let $x_1, \dots, x_n$ be a set of binary variables and $C_1, \dots, C_m$ a set of Boolean clauses. We shall define an instance (U, V, W, T) of 3DM such that M exists if and only if $F = C_1 \wedge \dots \wedge C_m$ is satisfiable. Let

$$U = \{x_i^j, \overline{x}_i^j : i = 1, \dots, n; j = 1, \dots, m\},$$

$$\begin{aligned} V = &\{a_i^j : i = 1, \dots, n; j = 1, \dots, m\} \\ &\cup \{v_j : j = 1, \dots, m\} \\ &\cup \{c_i^j : j = 1, \dots, m, i = 1, \dots, n-1\}, \end{aligned}$$

$$\begin{aligned} W = &\{b_i^j : i = 1, \dots, n; j = 1, \dots, m\} \\ &\cup \{w_j : j = 1, \dots, m\} \\ &\cup \{d_i^j : j = 1, \dots, m, i = 1, \dots, n-1\}, \end{aligned}$$

and

$$\begin{aligned} T = &\{(a_i^j, b_i^j, x_i^j) : i = 1, \ldots, n; j = 1, \ldots, m\} \\ &\cup \{(a_i^{j+1}, b_i^j, \overline{x}_i^j) : i = 1, \ldots, n; j = 1, \ldots, m\} \\ &\cup \{(v_j, w_j, \lambda^j) : j = 1, \ldots, m; \lambda \text{ a literal (i.e. an atomic formula) of } C_j\} \\ &\cup \{(c_i^j, d_i^j, \lambda^k) : i = 1, \ldots, n-1; j = 1, \ldots, m; k = 1, \ldots, m; \lambda \text{ a literal}\}, \end{aligned}$$

In order words, U contains a copy of each literal x_i for each clause C_j; and V and W contain three kinds of nodes each. It follows that F is satisfiable if and only if there exists a perfect matching M for (U, V, W, T) (see [93] Theorem 15.7 for more details).

Second, we have to polynomially transform 3DM to 3EC, which happens if we define $S = U \cup V \cup W$ and $F = \{\{a, b, c\} : (a, b, c) \in T\}$. In fact, 3-dimensional matching is a special case of 3-exact cover.

Third, we have to polynomially transform 3EC to 0-1KP. Let us define

$$c_j = \sum_{u_i \in S_j} (n+1)^{i+1}$$

and

$$K = \sum_{j=0}^{3m-1} (n+1)^j.$$

Then, it is easy to see that a subfamily F that covers $\{u_1, \ldots, u_{3m}\}$ exists if and only if the instance $\{c_1, \ldots, c_n; K\}$ of 0-1KP thus defined has a solution.

Last, we have to polynomially transform 0-1KP to NSMP. Let $M = 2n(n+1)K$, if we define an instance $\{d_1, \ldots, d_{2n}; L\}$ of the NSMP, where

$$d_j = \begin{cases} M^{n+1} + M^j + c_j & \text{if } j \leq n \\ M^{n+1} + M^{j-n} & \text{otherwise} \end{cases}$$

and

$$L = n \cdot M^{n+1} + \sum_{j=1}^{n} M^j + K,$$

then there exist integers $y_1, \ldots, y_{2n} \geq 0$ such that

$$\sum_{j=1}^{2n} d_j y_j = L$$

if and only if there exist integers $c_1, \ldots, c_n \geq 0$ such that

$$\sum_{j=1}^{n} c_j x_j = K.$$

Although we have skipped some details of the validity of all previous transformations for the sake of simplicity, a more exhaustive explanation can be found at [93] (Chapter 15). Thus, we have transformed an instance of the SAT problem into an instance of the NSMP, proving that the NSMP is complete for the class NP. $\square$

Now that we have proved that the numerical semigroup membership problem is in NP-complete, it is time to finally prove that the Frobenius problem is in NP-hard. For that, we are going to define a polynomial algorithm Λ_{NSMP} for solving the NSMP which uses as a subroutine an unknown algorithm Λ_{FP} that solves the Frobenius problem. Thus, we prove that the NSMP can be Turing reduced to the FP. As the NSMP is in NP-complete, we shall conclude that the FP is in NP-hard. The proof we present here is adapted from the original result published by Jorge Ramírez-Alfonsín in 1996 [97].

Let $t \in \mathbb{Z}_{\geq 0}$ and $\{a_1, \dots, a_n\}$ with $\gcd(a_1, \dots, a_n) = 1$. Then, the numerical semigroup membership problem is defined as the problem of determining if there exists a combination of non-negative integers $\lambda_1, \dots, \lambda_n \in \mathbb{Z}_{\geq 0}$ such that

$$\sum_{i=1}^{n} \lambda_i a_i = t.$$

The algorithm Λ_{NSMP} for the numerical semigroup membership problem will answer `YES` if the aforementioned combination $\lambda_1, \dots, \lambda_n$ exists for a certain input $\{t, a_1, \dots, a_n\}$ —in other words, if $t \in S_0 = \langle a_1, \dots, a_n \rangle$—, and `NO` otherwise.

In the first place, Λ_{NSMP} makes use of Λ_{FP} and calculates $f_0 = f(S_0)$. Then, three possible situations may appear: $t > f_0$, $t = f_0$ and $t < f_0$. If $t > f_0$, it is clear that the answer is `YES`; if $t = f_0$, then the answer is `NO`; if $t < f_0$, then the answer is `YES` if and only if $f_2 < f_1$, with f_1 and f_2 defined as follows.

Let $b_i = 2a_i$ for $i = 1, \dots, n$ and let $b_{n+1} = 2f_0 + 1$, then we define

$$S_1 = \langle b_1, \dots b_n, b_{n+1} \rangle$$

and

$$f_1 = f(S_1).$$

Let $b_{n+2} = f_1 - 2t$, we define

$$S_2 = \langle b_1, \dots b_n, b_{n+1}, b_{n+2} \rangle$$

and

$$f_2 = f(S_2).$$

Note that Λ_{NSMP} makes use again of Λ_{FP} for calculating f_1 and f_2. It remains to be proven that the algorithm Λ_{NSMP} answers correctly the NSMP. In order to achieve that, we need the following proposition.

Proposition 2.3.10. *Let f_0 and f_1 be defined as in algorithm Λ_{NSMP}, then $f_1 = 4f_0 + 1$.*

Proof. As $f_1 = f(S_1)$, the proof needs only to show two statements: first, that $p > 4f_0 + 1$ implies $p \in S_1$; second, that $4f_0 + 1 \notin S_1$.

Let $p \in \mathbb{Z}_{\geq 0}$ such that $p > 4f_0 + 1$, then we define q as follows:

$$q = \begin{cases} p & \text{if } p \equiv 0 \bmod 2 \\ p - b_{n+1} & \text{if } p \equiv 1 \bmod 2 \end{cases}$$

It is easy to see that $q > 2f_0$ in any case: in the first case,

$$q > 4f_0 + 1 > 2f_0,$$

and in the second case,

$$q = p - b_{n+1} > 4f_0 + 1 - (2f_0 + 1) = 2f_0.$$

As $b_{n+1} = 2f_0 + 1$, then $b_{n+1} \equiv 1 \mod 2$ and so $q \equiv 0 \mod 2$ in both cases, which implies that $q/2 > f_0$.

The last inequality implies that $q/2 \in S_0$, which means that there exist some integers $\alpha_0, \dots, \alpha_n \in \mathbb{Z}_{\geq 0}$ such that

$$\frac{q}{2} = \sum_{i=1}^{n} \alpha_i a_i.$$

Following the definition of $b_i = 2a_i$ we obtain that $q \in S_1$, as

$$q = \sum_{i=1}^{n} \alpha_i b_i.$$

As $S_1 \subset S_2$, then $q \in S_2$ and thus $p \in S_2$, which proves the first statement.

The second statement can be proved by *reductio ad absurdum.* Let us suppose that $4f_0 + 1 \in S_1$. This implies that there exist $\beta_1, \dots, \beta_n, \beta_{n+1} \in \mathbb{Z}_{\geq 0}$ such that

$$4f_0 + 1 = \sum_{i=1}^{n+1} \beta_i b_i.$$

As $4f_0 + 1 \equiv 1 \mod 2$ and $b_i \equiv 0 \mod 2$ for $i = 1, \dots, n$ (by definition of $b_i = 2a_i$), then forcefully $\beta_{n+1} \neq 0$. Let us suppose that $\beta_{n+1} \geq 2$: then,

$$\beta_{n+1} b_{n+1} \geq 2b_{n+1} = 4f_0 + 2 > 4f_0 + 1,$$

which leads to a contradiction. It remains to be seen what happens if $\beta_{n+1} = 1$: we obtain

$$4f_0 + 1 = \sum_{i=1}^{n} \beta_i b_i + b_{n+1},$$

so

$$2f_0 = \sum_{i=1}^{n} \beta_i b_i,$$

and finally

$$f_0 = \sum_{i=1}^{n} \beta_i a_i,$$

which cannot be possible as $f_0 \notin S_0$. □

Theorem 2.3.11. (J. L. Ramírez-Alfonsín) [97] *The Frobenius problem is in* NP-hard.

Proof. In order to prove that the Frobenius problem is NP-hard, suffices to show that the aforedescribed algorithm Λ_{NSMP} for the numerical semigroup membership problem is correct. The only statement that is not trivial is the following. Supposing $t < f_0$, then:

$$\text{There exist } \lambda_1, \dots, \lambda_n \in \mathbb{Z}_{\geq 0} \text{ such that } t = \sum_{i=1}^{n} \lambda_i a_i \iff f_2 < f_1.$$

The first implication can be easily proven. If

$$t = \sum_{i=1}^{n} \lambda_i a_i,$$

then

$$2t = \sum_{i=1}^{n} \lambda_i b_i$$

and, as $b_{n+2} = f_1 - 2t$ by definition, we obtain that

$$f_1 = \sum_{i=1}^{n} \lambda_i b_i + b_{n+2},$$

which implies that $f_1 \in S_2$. We already know that $f_2 \leq f_1$ (as $S_1 \subseteq S_2$), so we can conclude that $f_2 < f_1$ (as $f_2 = f_1$ would imply that $f_1 \notin S_2$, which we have already proved not being the case).

The second implication makes use of Proposition 2.3.10. If $f_2 < f_1$ and $f_1 = 4f_0 + 1$, then $4f_0 + 1 \in S_2$, which implies that there exists $\lambda_1, \dots, \lambda_{n+2}$ such that

$$4f_0 + 1 = \sum_{i=1}^{n+2} \lambda_i b_i.$$

As $f_1 \notin S_1$, forcefully $\lambda_{n+2} \geq 1$. It follows that

$$\lambda_{n+2} b_{n+2} = \lambda_{n+2}(f_1 - 2t) > \lambda_{n+2}\left(4f_0 + 1 - \frac{1}{2}(4f_0 + 1)\right) = \frac{1}{2}\lambda_{n+2}(4f_0 + 1).$$

If $\lambda_{n+2} \geq 2$, then

$$\lambda_{n+2} b_{n+2} > 4f_0 + 1,$$

which leads to a contradiction. If $\lambda_{n+2} = 1$, then

$$4f_0 + 1 = \sum_{i=1}^{n+1} \lambda_i b_i + b_{n+2} = \sum_{i=1}^{n+1} \lambda_i b_i + f_1 - 2t,$$

and so

$$2t = \sum_{i=1}^{n+1} \lambda_i b_i.$$

Even more, as $b_{n+1} = 2f_0 + 1 > 2t$, then

$$2t = \sum_{i=1}^{n} \lambda_i b_i,$$

and finally

$$t = \sum_{i=1}^{n} \lambda_i a_i.$$

□

Please note that we have used an algorithm for solving the Frobenius problem in order to prove that it is in NP-hard. However, the Frobenius problem was not stated as a decision problem, as its answer is an integer rather than a `YES` or `NO` output. These problems are known as function problems [62] and are usually redefined into equivalent decision problems. In this case, the decision version of the Frobenius problem should be stated as follows:

Definition 2.3.12. *Let S be a numerical semigroup and let $k \in \mathbb{Z}_{\geq 0}$, the decision version of the Frobenius problem answers* `YES` *if $f(S) \leq k$ and* `NO` *otherwise.*

As explained in [62], an algorithm that solves a function problem can be polynomially transformed into an algorithm that solves the decision version of that same problem. In fact, function problems have its own complexity classes —for example, FP and FNP, which are the counterparts of P and NP—. However, FP and FNP are seldom used when a decision version for a function problem can be defined, as in this case.

In addition, the numerical semigroup membership problem of a certain $t \in \mathbb{Z}_{\geq 0}$ with respect to a numerical semigroup S can be trivially Turing reduced to the problem of finding the Sylvester denumerant $d(t, S)$, as $d(t, S) \geq 1 \iff t \in S$. Thus, we can obtain the following result:

Corollary 2.3.13. *The Sylvester denumerant problem is in* NP-hard.

3

Quantum Computation

"I make up the rules, I'm allowed to do that."
– Richard P. Feynman, *Simulating Physics with Computers*

"Someday you will be old enough to start reading fairy tales again."
– C. S. Lewis, *The Chronicles of Narnia*

3.1 Introduction

The origins of quantum computation date back to 1980, when American physicist Paul Benioff (b. 1930) described a computing model defined by quantum mechanical Hamiltonians [19]. Later that year, Russian mathematician Yuri Manin gave a first idea on how to simulate a quantum system with a computer governed by quantum mechanics [86]. Both of them laid the groundwork for two of the basic components of quantum computing: quantum Turing machines and quantum computers [91].

Two years later, American theoretical physicist Richard Feynman (1918 – 1988) talked in one of his most seminal papers about the problems of simulating physics with a classical computer [52], and introduced independently a quantum model of computation. He stated that, being the world quantum mechanical, the inherent difficulty within the possibility of exactly replicating the behavior of nature is related to the problem of simulating quantum physics. This way, the most important rule defined by Feynman deals with the computational complexity at the time of efficiently simulating a quantum system. If one doubles the dimensions of the system, it would be ideal that the size of the computational resources needed for this task also double in the worst case, instead of experimenting an exponential growth.

Feynman also stated the underlying limitations that appear when it comes to simulate the probabilities of a physical system. Instead of calculating the probabilities of such a system, which he proved to be impossible, he proposed that the computer itself should have a probabilistic nature. To this new kind of machine he gave the name of quantum computer, and stated that it had a distinct essence than the well-known Turing machines. He also noted that with one of them it should be possible to simulate correctly any quantum system, and the physical world itself. Feynman asked himself if it would be possible to define a universal quantum computer, capable of modeling all possible quantum systems and detached from the possible problems that originate from its physical implementation, in the same way that a classical one is.

3.2 Quantum Turing Machines

Although the credit for introducing the concept of a universal quantum computer goes to Richard Feynman, it was British physicist David Deutsch (b. 1953) the first to properly describe, generalize and formalize it [41]. Supported in the works by Feynman, Manin and Benioff, he also introduced the concept of quantum Turing machine, which we proceed to define:

Definition 3.2.1. *A quantum Turing machine $\mathcal{M}$, or* QTM*, is defined by a triple $\mathcal{M} = (S, \Sigma, \delta)$, where the set of states S is replaced by the pure and mixed states of a Hilbert space, the alphabet Σ is finite, and the transition function δ is substituted for a set of unitary transformations which are automorphisms of S.*

This definition is rather informal and leaves out many important details. In fact, the study of quantum Turing machines is quite intricate, as many of its ingredients —such as the head position— can exist in a superposition of classical states [15]. Fortunately, an equivalent and much more friendly paradigm of computation called the quantum circuit model exists, and will be explained later on in this chapter along with all its details. Nevertheless, the reader interested in the complete and original definition of quantum Turing machines and all of their characteristics, can refer to the seminal papers where they were first outlined and formalized: [41], [42] and [22]. Besides, a quantum Turing machine can also be seen as a probabilistic Turing machine that obeys the rules of quantum probability instead of classical probability [106].

Definition 3.2.2. *The complexity class* BQP*, which stands for bounded-error quantum polynomial-time, contains all decision problems that can be solved in polynomial time by a quantum Turing machine with error probability bounded by 1/3 for all inputs.*

The latter class is usually taken as a reference for representing the power of quantum computers. Thanks to [22] and [41], we already know that BQP $\subseteq$ PSPACE and that BPP $\subseteq$ BQP. However, at the present time there is no known relationship between NP and BQP, except that P is inside their intersection. There is a strong belief that NP $\nsubseteq$ BQP; consequently, a polynomial-time quantum algorithm for an NP-complete problem would be surprising, as it would violate this conjecture. So far, we can update our previous complexity class diagram with this new class, as seen in Figure 3.2. A problem that is not known to be in BPP —or in P— is the factoring problem, but we already know thanks to Peter W. Shor a polynomial-time algorithm for this problem that runs on a quantum computer [105]. This algorithm, among many others, will be thoroughly explained in the next chapter.

It can be deduced from the definition of the complexity class BQP that the inner nature of quantum computation is probabilistic. In order to measure the performance of a quantum algorithm that solves a certain problem, we do not usually

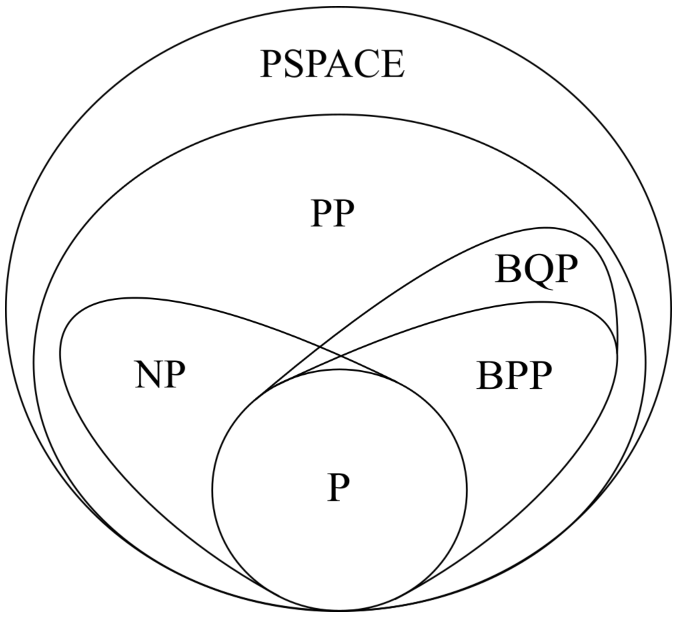


Figure 3.1: Diagram of complexity classes (incl. BQP)

take into account the time needed for obtaining the solution to that problem. What we do is to study the relationship between the probability of obtaining a correct solution and the computation time. In order for a quantum algorithm to be considered efficient, it must return a correct solution in polynomial time with a probability of at least 2/3 —although an additional complexity class, called EQP, includes all decision problems that are solved by quantum Turing machines with probability 1—. For a groundbreaking study on the algorithmic limitations of quantum computing, we refer to [20].

3.3 Quantum Bits and Quantum Entanglement

In classical computation, the basic unit of information is the *bit* —a portmanteau of binary digit—. A bit can only be in one of its two possible states, and may therefore be physically implemented with a two-state device. This pair of values is commonly represented with 0 and 1. On the other hand, we have an analogous concept in quantum computation: the *qubit* —short for quantum bit— which is a mathematical representation of a two-state quantum-mechanical system.

Definition 3.3.1. *The basis states of a quantum bit are $|0\rangle$ and $|1\rangle$, whose vector representations are:*

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \textit{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

However, there is a difference between bits and qubits, a qubit can also be in a state other than $|0\rangle$ and $|1\rangle$. Its generic state is, in fact, a linear combination over the complex numbers of both basis states.

Definition 3.3.2. *A pure qubit state $|\psi\rangle$ is a linear combination of the basis states,*

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

where $\alpha, \beta \in \mathbb{C}$ are called the amplitudes of the state and for which the constraint

$$|\alpha|^2 + |\beta|^2 = 1$$

holds.

Remark: Note that we have introduced a new notation, $|\psi\rangle$, termed *ket*, for describing a quantum state. This notation is part of the bra-ket notation, also named Dirac notation in honor of English theoretical physicist Paul Dirac (1902 – 1984), who first introduced it in 1939 [45]. Alternatively, we will also use $\langle\psi|$, called *bra*, to describe $|\psi\rangle^*$ (the Hermitian adjoint of $|\psi\rangle$).

Thus, we can say that $|0\rangle$ and $|1\rangle$ form an orthonormal $\mathbb{C}$-basis of $\mathbb{C}^2$ and that the state of a single-qubit system $|\psi\rangle$ is a unit vector of $\mathbb{C}^2$ (i.e., $|\psi\rangle \in \mathbb{C}^2$ with $|| \, |\psi\rangle|| = 1$). From now on, the basis formed by $|0\rangle$ and $|1\rangle$ will be called the computational basis of a qubit. Nevertheless, there are other commonly used basis for the states of a quantum bit. Another example that will come in handy later, known as the Hadamard basis in honor of French mathematician Jacques Hadamard (1865 – 1963), is defined by

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

and

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

It is easy to see that $|+\rangle$ and $|-\rangle$ also form an orthonormal $\mathbb{C}$-basis of $\mathbb{C}^2$, and that our generic qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ can be seen as

$$|\psi\rangle = \frac{\alpha + \beta}{\sqrt{2}}\,|+\rangle + \frac{\alpha - \beta}{\sqrt{2}}\,|-\rangle\,.$$

We know that qubits exist in nature thanks to the Stern-Gerlacht experiment, first conducted by German physicists Otto Stern (1888 – 1969) and Walther Gerlach (1889 – 1979) in 1922 [56]. A possible geometrical representation of the states of a single-qubit system, known as the Bloch sphere [24, 91] —see Figure 3.2—, leans on the fact that a generic qubit $\alpha\,|0\rangle + \beta\,|1\rangle$ with $|\alpha|^2 + |\beta|^2 = 1$ can be represented uniquely as a point (θ, ϕ) of the unit 2-sphere, with the north and south poles typically chosen to correspond to the standard basis vectors $|0\rangle$ and $|1\rangle$, and where $\alpha = \cos(\theta/2)$ and $\beta = e^{i\phi}\sin(\theta/2)$. Thus, we can see a qubit system as a unitary vector, and any transformation we would want to apply to it must preserve its norm. We shall explain these transformations in Section 3.4, but for now let us focus on what happens when we try to measure the state of a single-qubit system.

One of the key features that makes a quantum computer differ dramatically from its classical counterpart is the process of measuring the state of a quantum bit. A measurement, also called observation, of a generic single-qubit state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ yields a result from the orthonormal basis, depending on the values of α and β. However, the measurement process inevitably disturbs $|\psi\rangle$, forcing it to collapse to either $|0\rangle$ or $|1\rangle$ and thus generally making impossible the task of

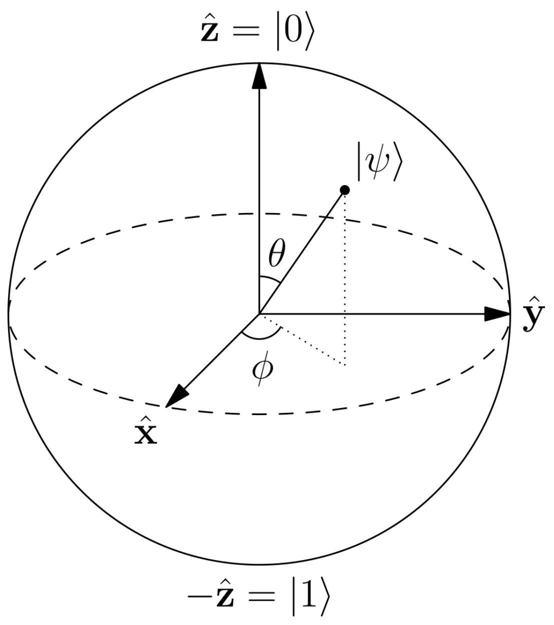


Figure 3.2: Bloch sphere[1]

finding out the actual values of α and β. This collapse to either $|0\rangle$ or $|1\rangle$ is non-deterministic and is ruled by the given probabilities $|\alpha|^2$ and $|\beta|^2$ respectively. It will be shown in Section 3.4 how to change these probabilities without violating the unitarity constraint. But first, let us explain some concepts needed to understand how multiple-qubit systems behave.

Definition 3.3.3. *Let V and W be vector spaces of dimensions n and m respectively. The tensor product of V and W, denoted by $V \otimes W$, is an nm-dimensional vector space whose elements are linear combinations of tensor products $v \otimes w$ satisfying the subsequent properties:*

- $z(v \otimes w) = (zv) \otimes w = v \otimes (zw)$
- $(v_1 + v_2) \otimes w = (v_1 \otimes w) + (v_2 \otimes w)$
- $v \otimes (w_1 + w_2) = (v \otimes w_1) + (v \otimes w_2)$

where $z \in \mathbb{C}$, $v, v_1, v_2 \in V$ and $w, w_1, w_2 \in W$.

A related definition that will come to use in Section 3.4 is the concept of tensor product between linear operators.

Definition 3.3.4. *Let A and B be linear operators defined on V and W respectively, then the linear operator $A \otimes B$ operating on $V \otimes W$ is defined as*

$$(A \otimes B)(v \otimes w) = Av \otimes Bw$$

with $v \in V$ and $w \in W$, and has the following matrix representation with respect to the canonical base:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}B & a_{n2}B & \cdots & a_{nn}B \end{bmatrix}$$

[1]Credit: Glosser.ca (CC BY-SA 3.0)

where A and B are $n \times n$ and $m \times m$ matrices respectively that correspond to the matrix representations of the linear operators A and B with respect to the canonical base. Note that we can conclude that the matrix representation of $A \otimes B$ has dimension $nm \times nm$.

By means of an example on how the matrix representations of the tensor products of linear operators are calculated, let

$$A = \begin{bmatrix} 1 & -1 \\ -2 & 0 \end{bmatrix} \text{ and } B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

be two linear operations defined on $\mathbb{R}^2$ and $\mathbb{R}^3$ respectively. Then, their tensor product is calculated as follows:

$$A \otimes B = \begin{bmatrix} 1 & -1 \\ -2 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 2 & 0 & 0 & -2 & 0 \\ 0 & 0 & 3 & 0 & 0 & -3 \\ -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & 0 & 0 \\ 0 & 0 & -6 & 0 & 0 & 0 \end{bmatrix}$$

where $A \otimes B$ is a linear operator defined on $\mathbb{R}^6$.

The tensor product we have thus defined can also be extended to vectors and non-square matrices, and will be useful at the time of calculating the basis states of a quantum system with more than one qubit. For instance, if $|0\rangle$ and $|1\rangle$ are the basis states of a quantum bit, the tensor product $|1\rangle \otimes |0\rangle$ will be given by:

$$|1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Before continuing with the possible states of a multiple qubit system, let us introduce some notation. In a classical computer, we represent an integer $a \in \mathbb{Z}_{\geq 0}$ such that $a < 2^n$ —i.e., such that it can be described with n bits— with the base-2 numeral system:

$$a = \sum_{l=0}^{n-1} a_l 2^l$$

where $a_l \in \{0, 1\}$ are the binary digits of a. In a quantum computer, we can also represent an integer $a < 2^n$ with n *qubits* as follows:

$$|a\rangle_n = |a_{n-1} \cdots a_1 a_0\rangle = \bigotimes_{l=0}^{n-1} |a_l\rangle$$

Thus, for example, number 29 can be represented with 5 qubits (as $29 < 2^5$) like this:

$$|29\rangle_5 = |11101\rangle = |1\rangle \otimes |1\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle .$$

From now on, the notation $|\psi\rangle_n$ will imply that we are describing a n-qubit system —where $n \geq 2$— instead of a single-qubit one, which will remain to be indicated with the absence of a subindex. We will also make use sometimes of the notation $|uv\rangle$ to describe the tensor product $|u\rangle \otimes |v\rangle$ of two basis states, with $u, v \in \{0, 1\}$. Now that we know what $|\psi\rangle_n$ and $|a\rangle_n$ really mean, we are finally in the position to begin studying the possible states of a multiple qubit system.

Definition 3.3.5. *The basis states of a two-qubit system are the tensor products of the basis states of a single-qubit system:*

$$|0\rangle_2 = |00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$|1\rangle_2 = |01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$|2\rangle_2 = |10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$|3\rangle_2 = |11\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

We shall see what happens when two qubits interact. The generic state of two different single-qubit systems, described independently, can be represented as

$$|\psi_0\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

and

$$|\psi_1\rangle = \gamma\,|0\rangle + \delta\,|1\rangle = \gamma \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \delta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \gamma \\ \delta \end{bmatrix},$$

where $|\alpha|^2 + |\beta|^2 = 1$ and $|\gamma|^2 + |\delta|^2 = 1$. This means that the state of this 2-qubit system should be described as the tensor product of both of them:

$$|\psi_0\rangle \otimes |\psi_1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \otimes \begin{bmatrix} \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \alpha\gamma \\ \alpha\delta \\ \beta\gamma \\ \beta\delta \end{bmatrix}.$$

On the other hand, if we want to describe a generic 2-qubit system $|\psi\rangle_2$ with the basis states defined in 3.3.5, we would have

$$|\psi\rangle_2 = \alpha_0 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \alpha_1 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix},$$

where

$$|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 = 1$$

must hold —remember that any quantum state must be described as a unitary vector—.

Note that a new constraint has arisen. If our generic two-qubit system described by $|\psi\rangle_2$ is to be decomposed in two single-qubit states (i.e., $|\psi\rangle_2 = |\psi_0\rangle \otimes |\psi_1\rangle$), then $\alpha_0 = \alpha\gamma$, $\alpha_1 = \alpha\delta$, $\alpha_2 = \beta\gamma$ and $\alpha_3 = \beta\delta$. It is easy to see that the equality $\alpha_0\alpha_3 = \alpha_1\alpha_2$ is imposed; however, there exists a physical phenomenon called **quantum entanglement** which implies that the quantum state of each one of the particles of a two-qubit system may not be described independently. In other words, it is not mandatory that the constraint $\alpha_0\alpha_3 = \alpha_1\alpha_2$ holds in a generic two-qubit system, which leads us to the subsequent definitions:

Definition 3.3.6. *A two-qubit general state is a linear combination of the basis states of a two-qubit system:*

$$|\psi\rangle = \alpha_0 |0\rangle_2 + \alpha_1 |1\rangle_2 + \alpha_2 |2\rangle_2 + \alpha_3 |3\rangle_2$$

holding the following constraint:

$$|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 = 1$$

Definition 3.3.7. *A two-qubit general state $|\psi\rangle_2$ is called mixed or entangled if there does not exist two one-qubit states $|\psi_0\rangle$ and $|\psi_1\rangle$ such that $|\psi\rangle_2 = |\psi_0\rangle \otimes |\psi_1\rangle$ (i.e., if it cannot be tensor-factorized).*

Quantum entanglement was first observed in nature in 1935, and in early days it was known as the EPR or Einstein–Podolsky–Rosen paradox. It was first studied by German-born theoretical physicist Albert Einstein (1879 – 1955) and his colleagues Boris Podolsky (1896 – 1966) and Nathan Rosen (1909 – 1995) [47], and later by Austrian physicist Erwin Schrödinger (1887 – 1961) [104]. The role and importance of quantum entanglement in quantum algorithms operating on pure states and in quantum computational speed-up was extensively discussed by Richard Jozsa and Noah Linden in [70].

Foreseeably, quantum entanglement occurs not only in two-qubit systems, but also in n-qubit systems, and fundamentally changes the way we see the information stored inside an n-qubit register.

Definition 3.3.8. *The state $|\psi\rangle_n$ of a generic n-qubit system is a superposition of the 2^n states of the computational basis $|0\rangle_n, |1\rangle_n, \ldots, |2^n - 1\rangle_n$. In particular,*

$$|\psi\rangle_n = \sum_{j=0}^{2^n-1} \alpha_j |j\rangle_n ,$$

with amplitudes $\alpha_j \in \mathbb{C}$ constrained to

$$\sum_{j=0}^{2^n-1} |\alpha_j|^2 = 1.$$

This can be seen as an obvious advantage with respect to classical computation. In a conventional computer we can store one but only one integer between 0 and $2^n - 1$ inside a n-bit register, which can be seen as a probability distribution between all possible integers where the integer we have stored has probability 1 and the rest have 0. In a quantum register, the probability can be distributed between all those integers instead of having just one possibility when it comes to read the register. Even more, if we are to simulate this quantum behavior with a classical computer, we would need 2^n registers of n bits, instead of a single n-qubit register as in the quantum case. This is precisely one of the benefits of quantum computing that Richard Feynman foretold in his paper [52].

Analogously to the single-qubit case, observing an n-qubit system unavoidably interferes with $|\psi\rangle_n$ and impels it to collapse in one of the vectors of the computational basis (i.e., in $|j\rangle_n$ with $0 \leq j < 2^n$). This collapse is again non-deterministic and is governed by the probability distribution given by $|\alpha_j|^2$. Thus, all the information that may have been stored in the amplitudes α_j is inevitably lost after the measurement process.

By way of illustration, let us suppose that we have the following 3-qubit quantum system:

$$|\psi\rangle_3 = \frac{1}{2}|1\rangle_3 + \frac{1}{2}|3\rangle_3 + \frac{1}{2}|5\rangle_3 + \frac{1}{2}|7\rangle_3 .$$

Then, if we measure this system, we will obtain with identical probability one of these possible outcomes: 1, 3, 5 or 7. Additionally, it is interesting to see the behavior of a quantum system if, rather than measuring all qubits at once, we measure them one by one. Our previous quantum system can be seen as

$$|\psi\rangle_3 = \frac{1}{2}|001\rangle + \frac{1}{2}|011\rangle + \frac{1}{2}|101\rangle + \frac{1}{2}|111\rangle .$$

But also as

$$|\psi\rangle_3 = \frac{1}{\sqrt{2}}|0\rangle \otimes \Big(\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle\Big) + \frac{1}{\sqrt{2}}|1\rangle \otimes \Big(\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle\Big)$$

or as

$$|\psi\rangle_3 = \Big(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\Big) \otimes \Big(\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle\Big).$$

If we measure the first qubit, we have the same probability of obtaining 0 or 1. However, as the measurement collapses the state of the qubit, the two remaining qubits will be forced to be in a state that is somewhat linked to the one we have obtained for the first qubit —i.e., the part that is tensored with the result we obtain for the first qubit—. Let us suppose that by measuring the first qubit, we have obtained a 1. Then, our 3-qubit system has collapsed to

$$|\psi\rangle_3 = |1\rangle \otimes \Big(\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle\Big),$$

which can also be seen as

$$|\psi\rangle_3 = |1\rangle \otimes \Big(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\Big) \otimes |1\rangle .$$

Note that the third qubit is already in one of the states of the computational basis, which means that, if we measure it right now, we will certainly obtain the value 1. The only remaining qubit that is not in the computational basis is the second one. Looking at the current state of our system, it is easy to see that we have the same probability of obtaining 0 or 1 by measuring it, which means that we will obtain 5 or 7.

It is of interest to see if the result we obtain from one of the qubits will condition the possible values for the remaining qubits. Let

$$|\psi\rangle_2 = \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle) + \frac{1}{\sqrt{2}}(|1\rangle \otimes |1\rangle)$$

be one of the four possible Bell states [18, 91], named after Northern Irish physicist John Stewart Bell (1928 – 1990). This state is composed of two entangled qubits —they cannot be described as two single-qubit states—. If we measure the second qubit, we have the same probability of obtaining 0 or 1. However, if we first measure the first qubit, and obtain 1, then the state of the second qubit will collapse —without having observed it— to 1, as the value 1 for the second qubit is only tensored with the value 1 of the first qubit. Thus, the result we obtain from a qubit or a set of qubits can be conditioned by the order in which we proceed to measure the rest of qubits. As will be seen in Chapters 4 and 5, the order in which we choose to read the members of a quantum register is one of the most important aspects of a quantum algorithm.

Another set of operations between qubits that are of great value are the inner and outer products, which we proceed to define.

Definition 3.3.9. *Let $|\psi_0\rangle_n$ and $|\psi_1\rangle_n$ be two n-qubit systems, the inner product of $|\psi_0\rangle_n$ and $|\psi_1\rangle_n$ is defined as usual by*

$$\langle\psi_0|\psi_1\rangle_n = |\psi_0\rangle_n^* |\psi_1\rangle_n$$

The inner product has the following properties:

- $\langle\psi_0|\psi_1\rangle_n = \langle\psi_1|\psi_0\rangle_n^*$
- $\langle\psi_0|(a\,|\psi_1\rangle + b\,|\psi_2\rangle)\rangle_n = a\,\langle\psi_0|\psi_1\rangle_n + b\,\langle\psi_0|\psi_2\rangle_n$
- $\langle\psi|\psi\rangle_n = ||\,|\psi\rangle_n\,||^2$

Definition 3.3.10. *Let $|\psi_0\rangle_n$ and $|\psi_1\rangle_n$ be two n-qubit systems, the outer product of $|\psi_0\rangle_n$ and $|\psi_1\rangle_n$ is defined by*

$$|\psi_0\rangle\,\langle\psi_1|_n = |\psi_0\rangle_n\,|\psi_1\rangle_n^*$$

For example, let

$$|\psi_0\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

and

$$|\psi_1\rangle = \gamma\,|0\rangle + \delta\,|1\rangle = \begin{bmatrix} \gamma \\ \delta \end{bmatrix}$$

be two generic single-qubit systems, then the matrix representations of the inner and outer product between $|\psi_0\rangle$ and $|\psi_1\rangle$ are calculated as follows:

$$\langle\psi_0|\psi_1\rangle = \begin{bmatrix} \alpha^* & \beta^* \end{bmatrix} \begin{bmatrix} \gamma \\ \delta \end{bmatrix} = \alpha^*\gamma + \beta^*\delta,$$

$$|\psi_0\rangle\,\langle\psi_1| = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \begin{bmatrix} \gamma^* & \delta^* \end{bmatrix} = \begin{bmatrix} \alpha\gamma^* & \alpha\delta^* \\ \beta\gamma^* & \beta\delta^* \end{bmatrix}.$$

Note that $\langle\psi|\psi\rangle = 1$ for any quantum state $|\psi\rangle$.

3.4 Quantum Circuits

The language of quantum circuits is a model of computation which is equivalent to quantum Turing machines and to universal quantum computers [115]. Currently, it is the more extensively used when it comes to describe an algorithm that runs on a quantum machine, and draws upon a sequence of register measurements —as described in Section 3.3— and discrete transformations —which will be explained in this section, as promised—. This is mainly due because all its elements can be treated as classical, with the sole exception of the information that is going through the wires.

First, we shall see what kind of transformations can be applied to the state of an n-qubit system. As a quantum state is always represented by a unit vector, we need the most general operator that preserves this property and the dimension of the vector.

Definition 3.4.1. *A matrix $A \in \mathcal{M}_{\mathbb{C}}(n)$ is unitary if*

$$A^*A = AA^* = I$$

where I is the identity matrix and A^* is the Hermitian adjoint of A.

Definition 3.4.2. *The unitary group of degree n, denoted by $U_{\mathbb{C}}(n)$, is the group of $n \times n$ unitary matrices, with matrix multiplication as the group operation.*

Proposition 3.4.3. *Let $A \in U_{\mathbb{C}}(n)$ be a unitary matrix and let $x \in \mathbb{C}^n$ be a unit vector, then $Ax \in \mathbb{C}^n$ is also a unit vector.*

In this context, a unitary transformation acting on n-qubits is called an n-qubit quantum gate, and can be represented by a unitary matrix. Let us expand this concept and its physical implications.

Definition 3.4.4. *A quantum gate that operates on a space of one qubit is represented by a unitary matrix $A \in U_{\mathbb{C}}(2)$. More generally, a quantum gate acting on an n-qubit system is represented by a unitary matrix $A \in U_{\mathbb{C}}(2^n)$.*

Please note that quantum gates necessarily have the same number of inputs and outputs, as opposed to classical logic gates. From now on, all quantum gates will be represented with a bold symbol, in order to differenciate them from mere matrices. We will show the most frequently used quantum gates as examples, and various results that simplify in a dramatic way the difficulty of implementing physically any quantum gate.

Definition 3.4.5. *The Hadamard gate is a single-qubit gate with the following matrix representation:*

$$\boldsymbol{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

which is unitary.

The circuit representation of the Hadamard gate is

$$|\psi_0\rangle \;-\!\boxed{\boldsymbol{H}}\!-\; |\psi_1\rangle$$

and, applied to each of the basis states, it has the following effect:

$$\boldsymbol{H} : |j\rangle \to \frac{1}{\sqrt{2}}\Big(|0\rangle + (-1)^j |1\rangle \Big).$$

As will be seen in Chapter 4, the Hadamard transformation is one of the most important quantum gates, since it is the basis of the majority of quantum algorithms developed to date. Its importance settles in the role it has at the time of generating all possible basis states, all of them with the same amplitude, inside a quantum register. Let us suppose that we have a qubit whose quantum state is $|\psi_0\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2 + |\beta|^2 = 1$. Then, the state of this single-qubit system after applying the Hadamard gate to it is:

$$\boldsymbol{H}|\psi_0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} \alpha+\beta \\ \alpha-\beta \end{bmatrix} = \frac{\alpha+\beta}{\sqrt{2}}|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}|1\rangle$$

Let us suppose now that, rather than having a generic state, we have the basis state $|\psi_0\rangle = |0\rangle$ in our one-qubit system. In that case, the result of applying the Hadamard gate will be as follows:

$$\boldsymbol{H}|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

As can be appreciated, we have transformed a basis state, $|0\rangle$, into a linear combination of the two basis states, $|0\rangle$ y $|1\rangle$, with identical amplitudes. If we measure the qubit at this moment, we will obtain with equal probability one of the two possible basis states. That said, what will happen if, rather than having a single quantum state, we have a n-qubit quantum system, all of them also in their basis state $|0\rangle$?

$$\boldsymbol{H}^{\otimes n}|0\rangle_n = \frac{1}{\sqrt{2^n}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}^{\otimes n} \begin{bmatrix} 1 \\ 0 \end{bmatrix}^{\otimes n} = \frac{1}{\sqrt{2^n}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle_n$$

What we have obtained is a superposition of all basis states of the system with identical probability. In other words, if we measure our n-qubit register right now, we will obtain a certain integer $j \in \{0, \dots, 2^n - 1\}$ with probability $1/2^n$.

Definition 3.4.6. *The Pauli gates are single-qubit gates with the following matrices:*

$$\boldsymbol{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\boldsymbol{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

$$\boldsymbol{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

which are unitary, but also Hermitian.

The Pauli matrices have the following effect on the basis states:

$$\boldsymbol{X} : |j\rangle \to |1 \oplus j\rangle$$

$$\boldsymbol{Y} : |j\rangle \to (-i)^j \, |1 \oplus j\rangle$$

$$\boldsymbol{Z} : |j\rangle \to (-1)^j \, |j\rangle$$

The three previous quantum gates are named after Austrian-born Swiss and American theoretical physicist Wolfgang Pauli (1900 — 1958). The three of them, along with the identity matrix I, form a basis for the vector space of 2×2 Hermitian matrices multiplied by real coefficients. However, all previously defined quantum gates have their limitations. In fact, quantum gates that are the direct product of single-qubit gates cannot produce entanglement.

Definition 3.4.7. *The $\boldsymbol{C}_{NOT}$ gate, which stands for controlled-not, is a two-qubit quantum gate with the following matrix representation:*

$$\boldsymbol{C}_{NOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Applied to a two-qubit basis state, the $\boldsymbol{C}_{NOT}$ gate has the next effect:

$$\boldsymbol{C}_{NOT} : |i\rangle \otimes |j\rangle \to |i\rangle \otimes |i \oplus j\rangle \, .$$

The $\boldsymbol{C}_{NOT}$ gate is another one of the key quantum gates, as it can be used to entangle and disentangle Bell states. In fact, it is the most simple gate that produces quantum entanglement. For example, let

$$|\psi\rangle_2 = \frac{1}{\sqrt{2}} \left(|0\rangle_2 + |2\rangle_2 \right)$$

be a separable quantum state (as $|\psi\rangle_2 = |+\rangle \otimes |0\rangle$). If we apply the $\boldsymbol{C}_{NOT}$ gate to it, we obtain

$$\begin{aligned} |\psi'\rangle_2 &= \boldsymbol{C}_{NOT}\left(|\psi\rangle_2\right) \\ &= \frac{1}{\sqrt{2}}\Big(\boldsymbol{C}_{NOT}(|0\rangle \otimes |0\rangle) + \boldsymbol{C}_{NOT}(|1\rangle \otimes |0\rangle)\Big) \\ &= \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle\right), \end{aligned}$$

which is one of the entangled Bell states.

Theorem 3.4.8. [46, 15] *Let $\boldsymbol{A} \in \mathcal{M}_{\mathbb{C}}(2^n)$ be a n-qubit gate, then it can be expressed as a finite number of tensor products of single-bit gates $M_i \in \mathcal{M}_{\mathbb{C}}(2)$ and the two-qubit $\boldsymbol{C}_{NOT}$ gate.*

The previous result imply that every unitary transformation on an n-qubit system can be implemented physically using only single-qubit gates and the $\boldsymbol{C}_{NOT}$ gate. In other words, single-qubit gates and the $\boldsymbol{C}_{NOT}$ gates form a set of universal gates. In fact, the most interesting result that links quantum entanglement and quantum computing performance over classical computation is the following:

Theorem 3.4.9. (Gottesman–Knill) *A quantum algorithm that starts in a computational basis state and does not feature quantum entanglement can be simulated in polynomial time by a probabilistic classical computer* [60].

Thus, we have explained the main notions needed for the correct comprehension of the rest of this doctoral thesis: a quantum circuit algorithm will consist in a set of transformations of two different types, observations and unitary transformations, to a n-qubit register. But before proceeding with the explanation of some of the most important quantum algorithms to date, let us describe a different model of quantum computing to that of quantum circuits.

3.5 Adiabatic Quantum Computing

Adiabatic quantum computing, or AQC, is an alternative model of quantum computing which apparently does not have much in common with quantum circuits. Nevertheless, both models are equivalent, as will be mentioned later, and thus adiabatic quantum computers are also tantamount to quantum Turing machines.

As hinted in the previous section —it will be fully explained in Chapter 4— a quantum circuit obtains an answer for a certain problem as a result of a succesion of quantum gates and register observations, in a process that has a discrete nature. On the other hand, the behavior of an adiabatic quantum computer is essentialy continuous —in fact, quantum circuits and adiabatic quantum computers are often respectively labelled as digital and analogic quantum computers— and is governed by the adiabatic theorem, a concept in quantum mechanics developed by German physicist and mathematician Max Born (1882 – 1970) and Soviet physicist Vladimir Fock (1898 – 1974), and first stated in 1928 [27]. Although the original paper is in German, extensive English-written studies on the adiabatic theorem can be found in [72, 88, 99]. Before proceeding with the adiabatic theorem, we need a few definitions.

Definition 3.5.1. *A Hamiltonian, denoted by $\mathcal{H}$, is an operator corresponding to the total energy of a quantum system. Its spectrum —the set of its eigenvalues— is the set of all possible outcomes that one could obtain when measuring the energy of such a system.*

Definition 3.5.2. *The ground state of a quantum system defined by a Hamiltonian $\mathcal{H}$ is the eigenvector of the smallest eigenvalue of $\mathcal{H}$. In the same way, its first excited state is the eigenvector of the second smallest eigenvalue of $\mathcal{H}$.*

We are now in position to understand the Schrödinger equation [103, 104], a mathematical equation result of the research of Erwin Schrödinger, which tells us how a quantum system evolves over time:

$$i\hbar\frac{\mathrm{d}}{\mathrm{d}t}\left|\psi(t)\right\rangle = \mathcal{H}(t)\left|\psi(t)\right\rangle.$$

Where $|\psi(t)\rangle$ is the time-dependent quantum state of the system.

Thus, the adiabatic theorem can be informally stated as:

“A physical system remains in its ground state if a given perturbation is acting on it slowly enough and if there is a gap between the associated eigenvalue and the rest of the Hamiltonian's spectrum.”

Although the original formulation of the adiabatic theorem is due to Born and Fock, the first proper proof of this result [9] was made by Japanese mathematician Tosio Kato (1917 – 1999), and can be found in [72].

The main idea behind adiabatic quantum computation is to define a Hamiltonian with respect to time in such a way that its initial ground state would be easy to calculate, and its final ground state would be the solution to a certain problem. This process, called quantum adiabatic evolution, can be seen as the following equation:

$$\mathcal{H}(t) = \Gamma(t)\mathcal{H}_I + \Lambda(t)\mathcal{H}_F$$

where $\mathcal{H}_I$ —the initial Hamiltonian— has a known ground state and $\mathcal{H}_F$ —the final Hamiltonian— has an unknown one that encodes the solution to our problem; t goes from 0 to T; $\Gamma(t)$ decreases from 1 to 0; and $\Lambda(t)$ increases from 0 to 1 monotonically with time. A typical case is when $\Gamma(t) = s(t)$ and $\Lambda(t) = 1 - s(t)$ with $s(t) = 1 - t/T$, which gives us:

$$\mathcal{H}(t) = s(t)\mathcal{H}_I + (1 - s(t))\mathcal{H}_F$$

Early works on the subject tried to replicate the performance of simulated annealing, a probabilistic technique for approximating the global minimum of a certain objective function —see [74] for more details—, in a quantum manner —i.e., taking advantage of tunneling and quantum fluctuations instead of thermal fluctuations in order to escape from local minima—. In these works, however, where it was called quantum annealing —see, for example, the seminal paper by Tadashi Kadowaki and Hidetoshi Nishimori [71]—, the main idea was to simulate this quantum behavior classically. The first experimental approach to quantum annealing came in 1999 usign a disordered quantum ferromagnet [32, 33], a result that hinted a possible quantum model of computation based on the adiabatic theorem.

The first quantum adiabatic algorithm, which solves certain instances of a combinatorial optimization problem —more concretely, the Boolean satisfiability problem, which we have already proven to be complete for NP—, was given by Edward Farhi, Jeffrey Goldstone, Sam Gutmann and Michael Sipser in 2000 [50, 49]. Afterwards, Wim van Dam, Michele Mosca and Umesh Vazirani studied in [112] the complexity of this algorithm, while also giving an adiabatic version of Grover's database search that also offers a quadratic speedup over the classical search algorithm.

Up to this point, the Hamiltonians involved in this kind of optimization problems share a common factor: they are *stoquastic* —meaning that all its off-diagonal matrix elements in the standard basis are real and non-positive—. However, the generalization of the model of adiabatic quantum computing that depends also on non-stoquastic Hamiltonians was proven to be equivalent to the quantum circuit model [8, 89] and thus to quantum Turing machines. This larger model is capable of solving any Turing-computable problem, instead of just the typically NP-hard combinatorial optimization problems associated with quantum annealing (see [87], which in turn cites [102]). In fact, it can polynomially simulate any quantum algorithm based on the quantum circuit model [8].

A problem associated with adiabatic quantum computing is the set of difficulties that arise at the time of measuring the performance of an adiabatic quantum algorithm. The most important deals with the computational time needed for evolving the system from $\mathcal{H}(0)$ to $\mathcal{H}(T)$ —i.e., from $\mathcal{H}_I$ to $\mathcal{H}_F$— and obtaining a solution, with respect to the probability that the solution is correct. In order to explain that, we give a formal enunciation of the adiabatic theorem.

Theorem 3.5.3. (Adiabatic Theorem) *Let $\mathcal{H}(t)$ be the Hamiltonian of a quantum system that evolves in time from $t = 0$ to $t = T$ and where $\mathcal{H}(t)$ describes an adiabatic quantum algorithm as in*

$$\mathcal{H}(t) = s(t)\mathcal{H}_I + (1 - s(t))\mathcal{H}_F,$$

where $s(t)$ decreases from 1 to 0 as t increases from 0 to T. Let $\tilde{\mathcal{H}}(s)$ be the equivalent Hamiltonian in the time scale s. Thus, by the Schrödinger equation, we have

$$i\frac{\mathrm{d}}{\mathrm{d}s}\left|\psi(s)\right\rangle = \tau(s)\tilde{\mathcal{H}}(s)\left|\psi(s)\right\rangle,$$

where $\tau(s)$ is the rate at which $\tilde{\mathcal{H}}(s)$ changes as a function of s. If:

- *$\tilde{\mathcal{H}}(s)$ has a non-degenerate ground state for all $s \in [0, T]$.*
- *The quantum system is in its ground state at $s = 0$.*
- *The process evolves slowly enough such that*

$$\tau(s) \gg \frac{1}{(\delta_{min})^2}\left|\left|\frac{\mathrm{d}}{\mathrm{d}s}\tilde{\mathcal{H}}(s)\right|\right|$$

where δ_{min} is the minimum spectral gap of the Hamiltonian —i.e. the minimum difference between the energy of the ground state and the first excited state of the system—.

- $\delta_{min} > 0$ *during the whole process.*

Then, the process will finish in the ground state of the final Hamiltonian $\mathcal{H}_F$.

A proof of this result can be found in [99] and [112]. The formulation here presented is based on the one appearing in [87]. Thus, the problem has to evolve slowly enough in order for the ground state and the first excited state do not interfere with each other, and it can be deduced from the adiabatic theorem that the time T needed for obtaining this feature must be such that

$$T \sim \mathcal{O}\left(\frac{1}{\delta_{min}^2}\right).$$

However, δ_{min} turns out to be a very difficult parameter to control and calculate in practice [39].

Naturally, it was stated in the description of the original quantum adiabatic algorithm [50] that we must determine how big has T to be in order to have a fairly good probability of success. Although various instances of NP-complete problems are shown in [50] such that T grows polynomially with respect to the size of the problem, it was later demonstrated in [112] and [99] that we would need a exponential time T in order to solve those NP-complete problems in the worst case.

As previously stated, a form of adiabatic quantum computation [87] is quantum annealing, where a known initial configuration of a quantum system evolves towards the ground state of a Hamiltonian that encodes the solution of an NP-hard optimization problem. The Canadian company D-Wave Systems announced in 2011 the first commercially available quantum annealer, composed of arrays of eight superconducting flux quantum bits with programmable spin–spin couplings, and published their results [68]. Subsequently in 2013, S. Boixo et al. [25] published their experimental results on the 108-qubit D-Wave One device. Their last chip, released in 2017 and called D-Wave 2000Q, has 2,048 qubits in a Chimera graph architecture [2], and can be seen as a computer that solves the Ising spin problem, which we proceed to describe.

The Ising spin model, originally formulated by physicist Wilhelm Lenz and first solved by his student, Ernst Ising [67], consists of a model of ferromagnetism in statistical mechanics in which we have to find the ground state of a system of n interacting spins. If we represent the spins of these particles as binary variables $s_i \in \{-1, 1\}$ with $i \in \{1, \dots, n\}$, then the Ising Spin problem can be expressed as an integer optimization problem whose objective is to find the spin configuration of the system that minimizes the function

$$H(s_1, \dots, s_n) = \sum_{i=1}^{n} h_i s_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} J_{ij} s_i s_j,$$

where $h_i \in \mathbb{R}$ is the energy bias acting on particle i —i.e. the external forces applied to each of the individual particles— and $J_{ij} \in \mathbb{R}$ is the coupling energy between the spins i and j —i.e. the interaction forces between adjacent spins—.

This problem can also be seen as the one of finding the lowest energy configuration —i.e. the ground state— of the following Hamiltonian:

$$\mathcal{H}_{Ising} = \sum_{i \in V} h_i \sigma_i^z - \sum_{(i,j) \in E} J_{ij} \sigma_i^z \sigma_j^z$$

where the notation σ_i^z indicates that the Pauli-Z operator

$$\boldsymbol{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

is applied to qubit i, i.e.

$$\sigma_i^z = \underset{1}{\boldsymbol{I}} \otimes \cdots \otimes \underset{i-1}{\boldsymbol{I}} \otimes \underset{i}{\boldsymbol{Z}} \otimes \underset{i+1}{\boldsymbol{I}} \otimes \cdots \otimes \underset{n}{\boldsymbol{I}},$$

and where (V, E) is the graph corresponding to the arrangement of qubits. Recall that the Pauli-Z operator has the following effect on the computational basis:

$$\boldsymbol{Z} : |k\rangle \to (-1)^k |k\rangle .$$

Thus, if we apply $\mathcal{H}_{Ising}$ to an n-qubit computational basis state, we obtain:

$$\begin{aligned}
\mathcal{H}_{Ising}(|k\rangle_n) &= \left(\sum_{i \in V} h_i \sigma_i^z - \sum_{(i,j) \in E} J_{ij} \sigma_i^z \sigma_j^z \right) (|k\rangle_n) \\
&= \left(\sum_{i \in V} h_i \sigma_i^z - \sum_{(i,j) \in E} J_{ij} \sigma_i^z \sigma_j^z \right) \left(\bigotimes_{l=1}^{n} |k_l\rangle \right) \\
&= \sum_{i \in V} h_i (-1)^{k_i} + \sum_{(i,j) \in E} J_{ij} (-1)^{k_i} (-1)^{k_j} \\
&= \sum_{i=1}^{n} h_i s_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} J_{ij} s_i s_j
\end{aligned}$$

In the last steps, we have assumed that V has n vertices and that $J_{ij} = 0$ if $(i, j) \notin E$. This problem was proved to be in NP-hard by Francisco Barahona [14], and can be effectively solved with the hardware implemented by D-Wave, whose chip permits to program independently the values of h_i and J_{ij} [68, 25]. It will be shown in Section 5.4 how to embed a certain optimization problem inside the D-Wave machine.

4

Quantum Algorithms

"For nothing is truly complete until the day it is finally destroyed."
– Brandon Sanderson, *Mistborn: The Hero of Ages*

" 'The Wheel weaves as the Wheel wills,' Moiraine mumbled. 'No eye can see the Pattern until it is woven.' "
– Robert Jordan, *Wheel of Time: The Eye of the World*

4.1 Introduction

In this section, we present a chronological summary of the first quantum algorithms that were shown to be more efficient than their best known classical counterparts. Our objective is to define them in the context of the previous chapter, while showcasing their main properties and proving their correctness. We also give worked-out examples for some of them, thus paving the way for the correct comprehension of the contributions of this doctoral thesis, described in Chapter 5.

But first, let us explain two concepts that will be common to many of the algorithms here presented. The first one is the hidden subgroup problem, which we proceed to define:

Definition 4.1.1. *Let G be a finitely generated group, let X be a finite set, and let $f : G \to X$ be a function that is constant on the orbits of a certain subgroup $K \subseteq G$ and distinct for each one of the orbits. The hidden subgroup problem, or* HSP*, is the problem of determining a generating set for K, using f as a black box.*

As will be shown, the superior performance of those algorithms relies on the ability of quantum computers to solve the hidden subgroup problem for finite Abelian groups. All those HSP-related algorithms were developed independently by different people, but the first to notice a common factor between them and to find a generalization was Australian mathematician Richard Jozsa (b. 1953) [69].

The other common factor, closely related to the HSP, is the possibility of building a quantum gate that can code a certain function f that is given as a black box —i.e., as a digital circuit—. A proof of this fact that uses the properties of reversible computation can be found in [91], and gives us another important quantum gate.

Definition 4.1.2. *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a function, the oracle gate $\boldsymbol{O}_f$ is the unitary transformation that has the following effect on the basis states of a quantum system:*

$$\boldsymbol{O}_f : |j\rangle_n \otimes |k\rangle_m \to |j\rangle_n \otimes |k \oplus f(j)\rangle_m ,$$

where $\oplus$ is the bitwise exclusive disjunction operation.

The first algorithms that shared those elements eventually evolved into Shor's factoring algorithm, probably the most celebrated of all quantum algorithms and the one that gave birth to another of the greatest achievements in quantum computation: the quantum Fourier transform. All those algorithms are capable of solving their respective problems in polynomial time; however, for some of them the inexistence of polynomial-time classical algorithms for those same problems has yet to be proven.

Another class of algorithms, which will be shown at the end of the chapter and that also uses the oracle gate, are based on Grover's quantum search, whose objective is to speed up the finding of a solution for a problem whose candidate solutions can be verified in polynomial time —i.e., all decision problems in NP—.

Finally, we explain the algorithm of quantum counting, which makes use of both worlds.

4.2 Deutsch's Algorithm

Let $f : \{0,1\} \to \{0,1\}$ be a function, it is clear that either $f(0) = f(1)$ or $f(0) \neq f(1)$. Let us suppose that we are given f as a black box, and that we want to know if f is constant. From a classical perspective, it is completely neccesary to evaluate the function both in $f(0)$ and $f(1)$ if we are to know this property with accuracy. Deutsch's algorithm [41] shows us that, with the help of a quantum computer, it is possible to achieve this with only a single evaluation of f.

We can see the previous question as an instance of the hidden subgroup problem, where $G = (\{0,1\}, \oplus)$, $X = \{0,1\}$, and K is either $\{0\}$ or $\{0,1\}$ depending on the nature of f. Note that in this case the cosets of $\{0\}$ are $\{0\}$ and $\{1\}$ and that the only coset of $\{0,1\}$ is precisely $\{0,1\}$.

$\mathbb{SETUP}$

```
$|\psi_0\rangle_{1,1} \leftarrow |0\rangle \otimes |1\rangle$
```

Deutsch's algorithm needs only two one-qubit registers. The first one is initialized at $|0\rangle$, and the second one at $|1\rangle$. As will be seen, this is due to the properties of the Hadamard gate when applied to the canonical basis states, and it is a frequent way of initializing a quantum algorithm.

STEP **1**

$$\boxed{|\psi_1\rangle_{1,1} \leftarrow \boldsymbol{H}^{\otimes 2}\left(|\psi_0\rangle_{1,1}\right)}$$

On the first step of Deutsch's algorithm we apply the Hadamard gate to both quantum registers, thus transforming the values of the canonical basis into the respective ones of the Hadamard basis.

$$\begin{aligned}
|\psi_1\rangle_{1,1} &= \boldsymbol{H}^{\otimes 2}\left(|\psi_0\rangle_{1,1}\right)\\
&= \boldsymbol{H}^{\otimes 2}\left(|0\rangle \otimes |1\rangle\right)\\
&= \Big(\boldsymbol{H}\,|0\rangle\Big) \otimes \Big(\boldsymbol{H}\,|1\rangle\Big)\\
&= \Big(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\Big) \otimes \Big(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\Big)\\
&= |+\rangle \otimes |-\rangle
\end{aligned}$$

STEP **2**

$$\boxed{|\psi_2\rangle_{1,1} \leftarrow \boldsymbol{O}_f\left(|\psi_1\rangle_{1,1}\right)}$$

The second step needs the oracle gate, defined at the beginning of this chapter for a generic function and for n and m qubits. Note that, in this case, the function f associated to the oracle gate as a black box is the one given for this instance of the hidden subgroup problem: $f : \{0, 1\} \to \{0, 1\}$. In fact, the oracle gate is not a constant transformation as are the Hadamard or the Pauli gates, but it rather depends on the problem. It is thus constructed *ad hoc* subject to the question we want to answer, provided that we have a logic circuit that implements f. The effect the oracle gate has on our quantum register can be seen as follows:

$$\begin{aligned}
|\psi_2\rangle_{1,1} &= \boldsymbol{O}_f\left(|\psi_1\rangle_{1,1}\right)\\
&= \boldsymbol{O}_f\left(|+\rangle \otimes |-\rangle\right)\\
&= \boldsymbol{O}_f\Big[\Big(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\Big) \otimes |-\rangle\Big]\\
&= \frac{\boldsymbol{O}_f(|0\rangle \otimes |-\rangle) + \boldsymbol{O}_f(|1\rangle \otimes |-\rangle)}{\sqrt{2}}\\
&= \frac{(-1)^{f(0)}\,|0\rangle \otimes |-\rangle + (-1)^{f(1)}\,|1\rangle \otimes |-\rangle}{\sqrt{2}}\\
&= \left(\frac{(-1)^{f(0)}\,|0\rangle + (-1)^{f(1)}\,|1\rangle}{\sqrt{2}}\right) \otimes |-\rangle
\end{aligned}$$

Please notice that all operations are just algebraic manipulations which allow us to see more clearly the information we have inside our quantum computer. We are not modifying anything, we are just reshaping the equation in order to have a better picture of what is happening.

STEP 3

$$\boxed{\left|\psi_3\right\rangle_{1,1} \leftarrow (\boldsymbol{H}\otimes\boldsymbol{I})\left(\left|\psi_2\right\rangle_{1,1}\right)}$$

The third and final step before measuring our quantum register involves again the Hadamard gate $\boldsymbol{H}$, but this time it is only applied to the first register. The second register is left alone, which is represented with an identity gate $\boldsymbol{I}$. In fact, the information inside the second register is no longer relevant, as it was only used as the auxiliary register needed for the oracle gate.

$$\begin{aligned}
\left|\psi_3\right\rangle_{1,1} &= (\boldsymbol{H}\otimes\boldsymbol{I})\left(\left|\psi_2\right\rangle_{1,1}\right)\\
&= (\boldsymbol{H}\otimes\boldsymbol{I})\left[\left(\frac{(-1)^{f(0)}\left|0\right\rangle + (-1)^{f(1)}\left|1\right\rangle}{\sqrt{2}}\right)\otimes\left|-\right\rangle\right]\\
&= \left(\frac{(-1)^{f(0)}\boldsymbol{H}\left|0\right\rangle + (-1)^{f(1)}\boldsymbol{H}\left|1\right\rangle}{\sqrt{2}}\right)\otimes\left|-\right\rangle\\
&= \left(\frac{(-1)^{f(0)}\left|+\right\rangle + (-1)^{f(1)}\left|-\right\rangle}{\sqrt{2}}\right)\otimes\left|-\right\rangle\\
&= \left(\frac{(-1)^{f(0)}\left|0\right\rangle + (-1)^{f(0)}\left|1\right\rangle + (-1)^{f(1)}\left|0\right\rangle - (-1)^{f(1)}\left|1\right\rangle}{2}\right)\otimes\left|-\right\rangle\\
&= \left(\frac{[(-1)^{f(0)} + (-1)^{f(1)}]\left|0\right\rangle + [(-1)^{f(0)} - (-1)^{f(1)}]\left|1\right\rangle}{2}\right)\otimes\left|-\right\rangle\\
&= (-1)^{f(0)}\left|f(0)\oplus f(1)\right\rangle\otimes\left|-\right\rangle
\end{aligned}$$

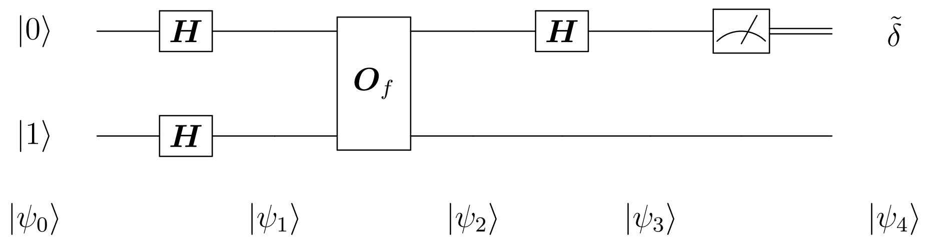
In order to understand the last part of the equation, one must take into account that, if $f(0) = f(1)$, then $(-1)^{f(0)} - (-1)^{f(1)} = 0$ and $f(0)\oplus f(1) = 0$. A similar reasoning goes for $f(0) \neq f(1)$, which leads us to the last expression for $\left|\psi_3\right\rangle_{1,1}$. Note also that $f(0)\oplus f(1) = 0$ if and only if $f(0) = f(1)$, and that $f(0)\oplus f(1) = 1$ if and only if $f(0) \neq f(1)$.

STEP 4

$$\boxed{\tilde{\delta} \leftarrow \text{measure the first register of } \left|\psi_3\right\rangle_{1,1}}$$

As the reader has surely noted, the information we wanted to obtain from the function f is already in the first register. We measure it now, thus destroying all the information related to the amplitudes of the basis states, and obtain a certain $\tilde{\delta} \in \{0,1\}$. If $\tilde{\delta} = 0$, then $K = \{0,1\}$ and $f(0) = f(1)$. If $\tilde{\delta} = 1$, then $K = \{0\}$ and $f(0) \neq f(1)$. A circuit representation of Deutsch's algorithm can be found in Figure

Figure 4.1: Circuit representation of Deutsch's algorithm



2.1.

At this moment, the inherent capabilities of quantum computing begin to surface. A problem which needs two evaluations of a function f in its classical version, can be reduced to just one evaluation of the same function in its quantum counterpart thanks to quantum paralelism. One may wonder if this property could be scaled to a function acting on $\{0,1\}^n$ rather than just $\{0,1\}$. That is the objective of the next algorithm.

4.3 Deutsch–Jozsa Algorithm

The following algorithm is a generalization of the previous one. Its original version appeared in [43], and is due again to David Deutsch and also to Richard Jozsa. Let $f : \{0,1\}^n \to \{0,1\}$ be a function that is either constant for all values in $\{0,1\}^n$, or is else balanced —i.e., equal to 0 for exactly half of all possible values in $\{0,1\}^n$, and to 1 for the other half—. The problem of determining if the function f is constant or balanced, using it as a black box, is called Deutsch's problem. In the classical version, a solution for this problem requires $2^{n-1}+1$ evaluations of f in the worst case. Let us see if we can improve that bound with the help of a quantum computer.

SETUP

$$|\psi_0\rangle_{n,1} \leftarrow |0\rangle_n \otimes |1\rangle$$

We need a quantum computer with $n+1$ qubits, where the first n qubits will be initialized at $|0\rangle$ and the remaining one at $|1\rangle$. Again, the single-qubit register is only used as the auxiliary qubit required for the oracle gate.

STEP **1**

$$|\psi_1\rangle_{n,1} \leftarrow \boldsymbol{H}^{\otimes n+1}\left(|\psi_0\rangle_{n,1}\right)$$

The first transformation we apply to our system is again the Hadamard gate. As explained in 3.4, when applied to the basis state $|0\rangle_n$ the Hadamard transformation gives us a superposition of all basis states with identical probability, thus obtaining the following quantum state:

$$\begin{aligned}
|\psi_1\rangle_{n,1} &= \boldsymbol{H}^{\otimes n+1}\Big(|\psi_0\rangle_{n,1}\Big) \\
&= \boldsymbol{H}^{\otimes n+1}\left(|0\rangle_n \otimes |1\rangle\right) \\
&= \Big(\boldsymbol{H}^{\otimes n}|0\rangle_n\Big) \otimes \Big(\boldsymbol{H}|1\rangle\Big) \\
&= \Big(\boldsymbol{H}|0\rangle\Big)^{\otimes n} \otimes \Big(\boldsymbol{H}|1\rangle\Big) \\
&= \Big(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\Big)^{\otimes n} \otimes \Big(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\Big) \\
&= \Big(\frac{1}{\sqrt{2^n}}\sum_{i=0}^{2^n-1}|i\rangle_n\Big) \otimes |-\rangle
\end{aligned}$$

𝕊𝕋𝔼ℙ **2**

$$\boxed{\quad |\psi_2\rangle_{n,1} \leftarrow \boldsymbol{O}_f\Big(|\psi_1\rangle_{n,1}\Big) \quad}$$

Now, we apply the oracle gate, which in this case is constructed for $n+1$ qubits and for the specific function f that we want to know if it is constant or balanced.

$$\begin{aligned}
|\psi_2\rangle_{n,1} &= \boldsymbol{O}_f\Big(|\psi_1\rangle_{n,1}\Big) \\
&= \boldsymbol{O}_f\Big[\Big(\frac{1}{\sqrt{2^n}}\sum_{i=0}^{2^n-1}|i\rangle_n\Big) \otimes |-\rangle\Big] \\
&= \frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}\boldsymbol{O}_f(|j\rangle_n \otimes |-\rangle) \\
&= \frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}(-1)^{f(j)}|j\rangle_n \otimes |-\rangle
\end{aligned}$$

The last step is better understood if we apply it separately to a generic basis state $|j\rangle_n$ with $j \in \{0, \dots, 2^n-1\}$, tensored with the Hadamard basis state $|-\rangle$.

$$\begin{aligned}
\boldsymbol{O}_f\Big(|j\rangle_n \otimes |-\rangle\Big) &= \boldsymbol{O}_f\Big(|j\rangle_n \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}\Big) \\
&= \frac{\boldsymbol{O}_f(|j\rangle_n \otimes |0\rangle) - \boldsymbol{O}_f(|j\rangle_n \otimes |1\rangle)}{\sqrt{2}} \\
&= \frac{|j\rangle_n \otimes |f(j)\rangle - |j\rangle_n \otimes |1 \oplus f(j)\rangle}{\sqrt{2}} \\
&= (-1)^{f(j)}|j\rangle_n \otimes \Big(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\Big) \\
&= (-1)^{f(j)}|j\rangle_n \otimes |-\rangle
\end{aligned}$$

Thus, we end up again with a superposition of all basis states in the first register, all of them with identical probability. The only difference with the previous state is that the amplitude of the states $|j\rangle_n$ remains identical if $f(j) = 0$, and is negated when $f(j) = 1$. Taking into account that either f is constant or balanced —i.e., all amplitudes are the same now, or half the amplitudes are positive and the other ones negative—, is there any way to obtain this information from our quantum register? Note that, until now, although we have applied f to every possible $j \in \{0, \dots, 2^n - 1\}$, we have used the gate that implements it only once.

$\mathbb{STEP}$ **3**

$$\boxed{|\psi_3\rangle_{n,1} \leftarrow (\boldsymbol{H}^{\otimes n} \otimes \boldsymbol{I}) \left(|\psi_2\rangle_{n,1}\right)}$$

The last step involves again the Hadamard transform. We apply it to the first n qubits of our quantum system, and obtain the following:

$$\begin{aligned}
|\psi_3\rangle_{n,1} &= (\boldsymbol{H}^{\otimes n} \otimes \boldsymbol{I}) \left(|\psi_2\rangle_{n,1}\right) \\
&= (\boldsymbol{H}^{\otimes n} \otimes \boldsymbol{I}) \left(\frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} (-1)^{f(j)} |j\rangle_n \otimes |-\rangle\right) \\
&= \left(\frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} (-1)^{f(j)} \boldsymbol{H}^{\otimes n} |j\rangle_n\right) \otimes |-\rangle \\
&= \left[\frac{1}{2^n} \sum_{k=0}^{2^n-1} \left(\sum_{j=0}^{2^n-1} (-1)^{f(j)+j\cdot k}\right) |k\rangle_n\right] \otimes |-\rangle
\end{aligned}$$

In order to understand the last equation, we must first fathom the effects of the Hadamard gate on a n-qubit basis state. Let $j \in \{0, \dots, 2^n - 1\}$, a closer inspection leads us to the following identity:

$$\boldsymbol{H}^{\otimes n}(|j\rangle_n) = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} (-1)^{j\cdot k} |k\rangle_n ,$$

where $j \cdot k$ is the bitwise inner product of j and k, modulo 2, i.e.,

$$j \cdot k = \sum_{l=1}^{n} j_l k_l \mod 2,$$

with j_l and k_l being the binary digits of j and k respectively. Likewise, the last identity is a generalization of the effect of the one-qubit Hadamard gate, which can be seen as:

$$\begin{aligned}\boldsymbol{H}(|j\rangle) &= \frac{1}{\sqrt{2}}\Big(|0\rangle + (-1)^j |1\rangle\Big)\\ &= \frac{1}{\sqrt{2}}\sum_{k=0}^{1}(-1)^{jk}|k\rangle\end{aligned}$$

$\mathbb{STEP}$ **4**

$\boxed{\tilde{k} \leftarrow \text{measure the first register of } |\psi_3\rangle_{n,1}}$

Finally, we are able to measure the qubits, thus destroying the information inside the register and obtaining a number $\tilde{k} \in \{0, \dots, 2^n - 1\}$ according to the probability distribution given by the amplitudes

$$\alpha_k = \sum_{j=0}^{2^n-1}(-1)^{f(j)+j\cdot k}.$$

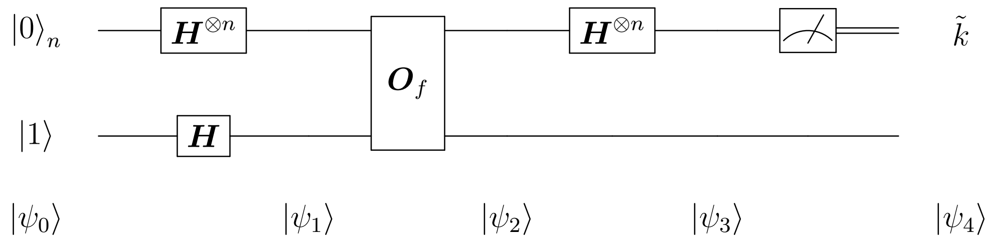
Let us have a closer look to the probability of obtaining $\tilde{k} = 0$. As explained in Chapter 3, in accordance with the laws of quantum mechanics the probability of obtaining a certain basis state $|k\rangle_n$ is equal to the square of the modulus of its amplitude α_k. In this case:

$$|\alpha_0|^2 = \left|\frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}(-1)^{f(j)}\right|^2$$

It is easy to see that $|\alpha_0|^2 = 1$ if and only if the function f is constant for all $j \in \{0, \dots, 2^n-1\}$, and that $|\alpha_0|^2 = 0$ if and only if the function f is balanced (recall that we are promised that f is of one of those two natures). Having said that, if we measure now the first register and obtain some $\tilde{k}$, we can conclude that f is constant if $\tilde{k} = 0$, and that f is balanced otherwise. With just a single evaluation of f we have answered the question, as opposed to the $2^{n-1} + 1$ evaluations needed in the classical version. A circuit representation of Deutsch-Jozsa algorithm is displayed in Figure 4.2.

The previous algorithm has much more profound implications than the posibility of solving Deutsch's problem exponentially faster with the help of a quantum

Figure 4.2: Circuit representation of Deutsch-Jozsa algorithm



computer. It also tells us that, relative to an oracle —i.e., a black box that solves a certain problem or function, namely f— we can establish a difference between the classes P and EQP. Note that this does not imply that P $\neq$ EQP, it just tells us that there exists an oracle separation between P and EQP.

4.4 Simon's Algorithm

4.4.1 Definition

Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function such that, for some $s \in \{0,1\}^n$ with $s \neq (0,0,\dots,0)$, $f(j) = f(k)$ if and only if either $j = k$ or $j \oplus k = s$ for all $j, k \in \{0,1\}^n$ —where $\oplus$ is again the bitwise exclusive disjunction operation, also called bitwise xor—. Simon's problem is defined as: given such an f as a black box, figure out the value of s, which is usually called the xor-mask of f. Both the problem and the quantum algorithm we proceed to explain were both first presented in [106] by Daniel R. Simon, hence their names.

Simon's problem can also be seen as an instance of the hidden subgroup problem, where $G = (\{0,1\}^n, \oplus)$, $X \subseteq \{0,1\}^n$ is any finite set, and $K = \{0, s\}$ for some $s \in \{0,1\}^n$. In the classical version, a solution for this problem requires that we find a pair of values $x, y \in \{0,1\}^n$ such that $f(x) = f(y)$, and then compute $x \oplus y$. This solution requires $\mathcal{O}(2^{n/2})$ evaluations of f in the worst case whereas, as will be proved later, Simon's algorithm only needs $\mathcal{O}(n)$ evaluations of f.

SETUP

$$\boxed{|\psi_0\rangle_{n,n} \leftarrow |0\rangle_n \otimes |0\rangle_n}$$

In this algorithm, we need $2n$ qubits, all of them initialized at $|0\rangle$.

STEP **1**

$$\boxed{|\psi_1\rangle_{n,n} \leftarrow (\boldsymbol{H}^{\otimes n} \otimes \boldsymbol{I}^{\otimes n}) \left(|\psi_0\rangle_{n,n}\right)}$$

We first apply the Hadamard transformation to the first half of our qubit set, thus obtaining the following quantum state.

$$\begin{aligned}
|\psi_1\rangle_{n,n} &= (\boldsymbol{H}^{\otimes n} \otimes \boldsymbol{I}^{\otimes n}) \left(|\psi_0\rangle_{n,n}\right) \\
&= (\boldsymbol{H}^{\otimes n} \otimes \boldsymbol{I}^{\otimes n})(|0\rangle_n \otimes |0\rangle_n) \\
&= (\boldsymbol{H}^{\otimes n} |0\rangle_n) \otimes (|0\rangle_n) \\
&= \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle_n \otimes |0\rangle_n
\end{aligned}$$

STEP **2**

$$|\psi_2\rangle_{n,n} \leftarrow \boldsymbol{O}_f \left(|\psi_1\rangle_{n,n}\right)$$

Next, we use the oracle gate, built particularly for the function f. Note that, thanks to quantum paralelism, we apply here the function f to all possible values in $\{0,1\}^n$ with just a single iteration of $\boldsymbol{O}_f$. Thus, all possible values of f are now present in the second register.

$$\begin{aligned}
|\psi_2\rangle_{n,n} &= \boldsymbol{O}_f \left(|\psi_1\rangle_{n,n}\right) \\
&= \boldsymbol{O}_f \left(\frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle_n \otimes |0\rangle_n\right) \\
&= \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} \boldsymbol{O}_f \left(|j\rangle_n \otimes |0\rangle_n\right) \\
&= \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle_n \otimes |f(j)\rangle_n
\end{aligned}$$

STEP **3**

$\tilde{\delta} \leftarrow$ measure the second register of $|\psi_2\rangle_{n,n}$

$|\psi_3\rangle_n \leftarrow |\psi_2\rangle_{n,n}$ after measuring the second register

In this step we see for the first time the true effects of measuring part of our quantum system before completing the execution of an algorithm. If we measure now the second register, we shall end up with a value $\tilde{\delta} = f(\tilde{j})$ for a certain $\tilde{j} \in \{0,1\}^n$. Thus, only the values in $f^{-1}(\tilde{\delta})$ will remain in the first register —before measuring, they were the only ones tensored with $|\tilde{\delta}\rangle$—. In any case, as $f^{-1}(\tilde{\delta}) = \{\tilde{j}, \tilde{j} \oplus s\}$ —we remark that $j \oplus k = s$ if and only if $j \oplus s = k$—, we end up with the following quantum state:

$$|\psi_3\rangle_n = \frac{1}{\sqrt{2}} \left(|\tilde{j}\rangle_n + |\tilde{j} \oplus s\rangle_n\right)$$

STEP **4**

$$|\psi_4\rangle_n \leftarrow \boldsymbol{H}^{\otimes n} \left(|\psi_3\rangle_n\right)$$

The last transformation we apply to our quantum system is, again, the Hadamard gate.[1] Before that, we could have measured the first register and obtain a certain

[1]Did we mention that it is one of the important ones?

value in $f^{-1}(\tilde{\delta}) = \{\tilde{j}, \tilde{j} \oplus s\}$. However, in that case we would have ended with the same information as if we had just made a single classical evaluation of f. The Hadamard gate, on the other hand, will let us obtain much more information than from a single evaluation of f: it will give us some precious information about s. Following a similar reasoning as with Deutsch-Jozsa algorithm, we end up with:

$$\begin{aligned}
|\psi_4\rangle_n &= \boldsymbol{H}^{\otimes n}\left(|\psi_3\rangle_n\right) \\
&= \boldsymbol{H}^{\otimes n}\left[\frac{1}{\sqrt{2}}\left(|\tilde{j}\rangle_n + |\tilde{j}\oplus s\rangle_n\right)\right] \\
&= \frac{1}{\sqrt{2}}\left(\boldsymbol{H}^{\otimes n}|\tilde{j}\rangle_n + \boldsymbol{H}^{\otimes n}|\tilde{j}\oplus s\rangle_n\right) \\
&= \frac{1}{\sqrt{2}}\left[\frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}(-1)^{\tilde{j}\cdot k}|k\rangle_n + \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}(-1)^{(\tilde{j}\oplus s)\cdot k}|k\rangle_n\right] \\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{k=0}^{2^n-1}(-1)^{\tilde{j}\cdot k+(\tilde{j}\oplus s)\cdot k}|k\rangle_n \\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{k=0}^{2^n-1}(-1)^{\tilde{j}\cdot k}\left[1+(-1)^{s\cdot k}\right]|k\rangle_n
\end{aligned}$$

STEP 5

```
$\tilde{\omega} \leftarrow$ measure $|\psi_4\rangle_n$
```

Let us suppose that we measure our quantum system right now. It is clear that the current amplitudes of the basis states are

$$\alpha_k = \left|\frac{1}{\sqrt{2^{n+1}}}\left[1+(-1)^{s\cdot k}\right]\right|^2$$

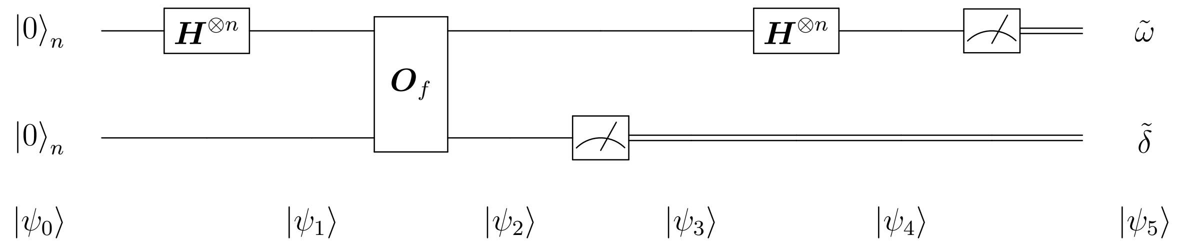
for $k \in \{0,1\}^n$.

However, it can be noted that $\alpha_k \neq 0$ if and only if $s \cdot k \equiv 0 \mod 2$, which happens for half the values of k necessarily. Even more, in those cases the amplitude is equal to $1/2^{n-1}$. Analyzing the outcome, we have ended up with some $\tilde{\omega}$ such that $\tilde{\omega} \cdot s \equiv 0 \mod 2$. If we are able to find $n-1$ linearly independent values of $\tilde{\omega}$, namely $\tilde{\omega}_1, \ldots, \tilde{\omega}_{n-1}$, we will arrive at a system of equations whose solutions are 0 and s. Before proving this, let us explain the performance of Simon's algorithm with a worked-out example.

4.4.2 Example with $n = 4$

Let us suppose that we are given as a black box a function that fulfills the requirements of Simon's problem. This function, namely $f : \{0,1\}^4 \to \{0,1\}^4$, has the following outcome:

Figure 4.3: Circuit representation of Simon's algorithm (one iteration)



$$\begin{array}{ccccc}
f(0) & = & f(5) & = & 0 \\
f(1) & = & f(4) & = & 1 \\
f(2) & = & f(7) & = & 2 \\
f(3) & = & f(6) & = & 3 \\
f(8) & = & f(13) & = & 4 \\
f(9) & = & f(12) & = & 5 \\
f(10) & = & f(15) & = & 6 \\
f(11) & = & f(14) & = & 7
\end{array}$$

Of course, as the function is given as a black box, this information is only available to us if we evaluate $f(j)$ for all $j \in \{0, \ldots, 15\}$. A closer inspection of these values tells us that this function has in fact a xor-mask and that its value is $s = 5$. Our objective is to arrive at this knowledge without evaluating f classically for all values in $\{0,1\}^4$.

Now, let us suppose that we do not know this information yet. As a start, Simon's algorithm would need the quantum state

$$|\psi_0\rangle_{4,4} = |0\rangle_4 \otimes |0\rangle_4 .$$

After applying the Hadamard gate, we would obtain

$$|\psi_1\rangle_{4,4} = \frac{1}{16}\sum_{j=0}^{15} |j\rangle_4 \otimes |0\rangle_4$$

and, after the oracle gate, our system is in the state

$$|\psi_2\rangle_{4,4} = \frac{1}{16}\sum_{j=0}^{15} |j\rangle_4 \otimes |f(j)\rangle_4 .$$

If we make use of the information we know (but we should not!) about f, we could see the previous equation as:

$$\begin{aligned}|\psi_2\rangle_{4,4} = \frac{1}{8}\Big[&\big(|0\rangle_4 + |5\rangle_4\big) \otimes |0\rangle_4 + \big(|1\rangle_4 + |4\rangle_4\big) \otimes |1\rangle_4 + \big(|2\rangle_4 + |7\rangle_4\big) \otimes |2\rangle_4 + \\ &\big(|3\rangle_4 + |6\rangle_4\big) \otimes |3\rangle_4 + \big(|8\rangle_4 + |13\rangle_4\big) \otimes |4\rangle_4 + \big(|9\rangle_4 + |12\rangle_4\big) \otimes |5\rangle_4 + \\ &\big(|10\rangle_4 + |15\rangle_4\big) \otimes |6\rangle_4 + \big(|11\rangle_4 + |14\rangle_4\big) \otimes |7\rangle_4\Big]\end{aligned}$$

Please note again that the previous state is actually happening inside our quantum computer whether or not we know the values for $f(j)$. As we have constructed our oracle gate using f as a black box, it necessarily has the previous effect on the Hadamard state.

Up until now all steps were deterministic. However, the next step, the measurement of the second register, will have a non-deterministic outcome. Let us suppose that we measure it and obtain, for example, the value $\tilde{\delta} = 6$. Necessarily, our quantum system is now in the state

$$|\psi_3\rangle_4 = \frac{1}{\sqrt{2}}(|10\rangle_4 + |15\rangle_4)$$

and, if we apply now the Hadamard transform, we end up with

$$|\psi_4\rangle_4 = \frac{1}{\sqrt{2^5}}\sum_{k=0}^{15}(-1)^{\tilde{j}\cdot k}\left[1+(-1)^{s\cdot k}\right]|k\rangle_n\,,$$

where $\tilde{j} \in f^{-1}(6) = \{10, 15\}$ and s is the (yet unknown!) xor-mask of f.

As can be seen, the values we get of $\tilde{j}$ and $\tilde{\delta}$ are unimportant. What we need is the outcome of the measurement of our quantum system at this moment. As previously said, we will end up with a number $\tilde{\omega}$ such that $s \cdot \tilde{\omega} = 0 \mod 2$. In this case, $\tilde{\omega} \in \{0, 2, 5, 7, 8, 10, 13\}$ for $s = 5$, where all of them have the same probability of coming up.

Let us suppose that we have run three complete iterations of Simon's algorithm, and obtain $\tilde{\omega}_1 = 2$, $\tilde{\omega}_2 = 7$ and $\tilde{\omega}_3 = 10$. Thus, draining from the fact that $\tilde{\omega}_i \cdot s = 0$ mod 2 for all of them, if we define $s = s_3s_2s_1s_0$ as the bitwise representation of s (with $s_i \in \{0, 1\}$), we can consider the system of equations

$$\begin{array}{ccccccccc} & & & & s_1 & & & = & 0 \mod 2 \\ & & s_2 & + & s_1 & + & s_0 & = & 0 \mod 2 \\ s_3 & + & s_2 & & & + & s_0 & = & 0 \mod 2, \end{array}$$

with each one of the equations giving us respectively the following set of solutions:

$$\begin{array}{rcl} \Omega_1 & = & \{0, 1, 4, 5, 8, 9, 12, 13\}, \\ \Omega_2 & = & \{0, 3, 5, 6, 8, 11, 13, 14\}, \\ \Omega_3 & = & \{0, 1, 4, 5, 10, 11, 14, 15\}. \end{array}$$

Clearly, as our xor-mask s must satisfy all previous equations, we can deduce that $s \in \Omega_1 \cap \Omega_2 \cap \Omega_3 = \{0, 5\}$ and, as $s \neq 0$ by definition, we can conclude that our xor-mask is, in fact, $s = 5$. Note that as the process of obtaining the different values for $\tilde{\omega}$ is non-deterministic, the remaining question is this: how many times do we have to execute Simon's algorithm in order to obtain such a system with enough probability?

4.4.3 Proof of correctness

Theorem 4.4.1. *Simon's algorithm finds the correct solution for Simon's problem in $\mathcal{O}(n)$ steps with probability greater than* $1/3$.

Proof. Let us suppose that we have obtained m linearly independent equations, namely with $\tilde{\omega}_1, \ldots, \tilde{\omega}_m$. Then, the probability of obtaining another linearly independent equation in the next interation of Simon's algorithm is

$$\frac{2^n - 2^m}{2^n}.$$

Thus, assuming $n \geq 3$, the probability of obtaining $n-1$ linearly independent equations after $n-1$ iterations of Simon's algorithm is

$$P = \left(1 - \frac{1}{2^n}\right)\left(1 - \frac{2}{2^n}\right)\cdots\left(1 - \frac{2^{n-2}}{2^n}\right) \geq \left(1 - \sum_{k=2}^{n} \frac{1}{2^k}\right) \geq \frac{2^{n-1} - 1}{2^n} > \frac{1}{3}$$

□

Even though Simon's algorithm is of little practical use in precisely the same way as Deutsch's and Deutsch-Jozsa's are, it shows once more that there exist problems such that a quantum computer is capable of solving them efficiently while a classical one is not. In fact, Simon's algorithm shows that there exist problems such that a quantum Turing machine is exponentially faster than a probabilistic Turing machine [91]. The difference between Simon's and Deutsch-Jozsa is that the latter can be solved by a PTM with an arbitrarily small error, while the former would take an exponential time to solve with such a machine. Finally, although Simon's algorithm stablishes an oracle separation between BPP and BQP, we still do not know if BPP $\neq$ BQP, as Simon's problem depends on a black box. An adiabatic version of Simon's algorithm can be found in [66].

4.5 Shor's Factoring Algorithm

4.5.1 Introduction

Let us begin with a problem in number theory that stems from one of the most well known theorems of all time:

Definition 4.5.1. *Let $N \in \mathbb{Z}_{\geq 0}$, the fundamental theorem of arithmetic tells us that there exists a unique factorization of N as a product of prime powers:*

$$N = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k} = \prod_{i=1}^{k} p_i^{\alpha_i}.$$

The prime factorization problem, or PFP, *is the problem of finding such a factorization for a given number $N \in \mathbb{Z}_{\geq 0}$.*

Many mathematicians have worked on algorithms that calculate the prime factorization of an integer. To understand the ideas behind the most recent solutions to this problem, we must go back to the 17th Century, when a French lawyer called Pierre de Fermat (1607 – 1665) invented an elegant factorization method that today bears his name. Fermat's method consists in representing an odd number N as a difference of squares, which is easily proven to exist. Then, as $N = n^2 - m^2 = (n+m)(n-m)$, we have that $\gcd(n+m, N)$ and $\gcd(n-m, N)$ are non-trivial factors of N.

But it was not until the beginning of the 20th Century that some improvements were made, as mathematicians like Maurice Kraitchik in 1922 [78], Derrick Henry Lehmer and Ralph Ernest Powers in 1931 [81], Michael A. Morrison and John Brillhart in 1975 [90] and Richard Schroeppel at the end of the 1970s (unpublished, but described in [94]) developed factorization methods whose ideas were around the original Fermat's method. These upgrades eventually arrived at its maximum expression with the Quadratic Sieve developed by Carl Pomerance in 1984 [95] and the General Number Field Sieve due to John Pollard in 1989 [82, 21]. For more information about the story behind the evolution of Fermat's idea, see [96].

The two previous methods are currently the most efficient classical algorithms for factoring an integer. However, they still have a problem: their computational complexity is super-polynomial in the number $\log N$ of digits in N. In fact, the GNFS, which has proven to be the most efficient known classical algorithm for factoring integers larger than 10^{100}, has the following computational order:

$$\mathcal{O}\left(e^{(\log N)^{\frac{1}{3}}(\log\log N)^{\frac{2}{3}}}\right)$$

Unfortunately, Pollard's method has the constraints of any super-polynomial algorithm and, at the present time, no known ponynomial-time classical algorithm exists for the factoring problem —i.e., PFP is not known to be in P—. Nevertheless, verifying that a candidate solution for this problem is in fact the actual solution is computationally easy; thus, PFP is in NP.

There is a very well known result in computational complexity theory, due to American computer scientist Richard E. Ladner (b. 1943) [79], that tells us the following: if P $\neq$ NP, then there exists a non-empty class, called NP-intermediate, that contains all problems in NP which are neither in the class P nor in NP-complete. It is widely believed that the prime factorization problem is inside this class.

What we surely know, thanks to American mathematician Peter W. Shor (b. 1959) and its acclaimed polynomial-time quantum algorithm for prime factorization [105], is that the PFP is in BQP. The objective of this subsection is to describe such result. For that, we shall first define Shor's algorithm as a classical one that relies on a black box that finds the multiplicative order of a modulo n. Next, we will provide a quantum algorithm that substitutes that black box. Finally, we will describe its performance via a worked-out example.

4.5.2 Definition: Classical part

Let $N \in \mathbb{Z}_{\geq 0}$, we proceed to define Shor's algorithm for factoring N.

$\mathbb{STEP}$ **1**

```
$x \leftarrow$ random integer such that $1 < x < N$
$d \leftarrow \gcd(x, N)$
```

It is clear that, if $d > 1$, we have already found a factor of N. However, the probability of such an unlikely event is small, and in this case we proceed to next step. Note that the computational complexity of this step —i.e., of calculating the greatest common divisor of x and N— has order $\mathcal{O}(\log^2 N)$ [76].

$\mathbb{STEP}$ **2**

```
$r \leftarrow O_N(x)$
```

This is the step that we shall resolve with the aid of a quantum computer, as will be explained later. For now, let us recall that the multiplicative order of x modulo N, provided that $\gcd(x, N) = 1$, is defined as

$$O_N(x) = \min\{r \in \mathbb{Z}_{>0} : x^r \equiv 1 \mod N\}.$$

Calculating the multiplicative order is a hard problem in the general case, and the best known classical algorithm that solves it has a super-polynomial computational complexity [36].

Right now, we have obtained a certain r such that $r = O_N(x)$. However, not any value of r serves our purposes. At the end of this step, we shall check if r is an even number and, if that holds, we have to also check if $x^{r/2} + 1 \not\equiv 0 \mod N$. If any of those two conditions fail, we shall go back to the beginning of the algorithm, and repeat it again with a different random value for x. The unavoidable question is: what is the probability of not making it?

Theorem 4.5.2. *Let $N \in \mathbb{Z}_{\geq 0}$ such that $2 \nmid N$ and whose prime factorization is*

$$N = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}.$$

Suppose x is chosen at random, with $1 < x < N$ and $\gcd(x, N) = 1$, and let $r = O_N(x)$. Then:

$$\text{Prob}\left[(2 \mid r) \wedge (x^{r/2} + 1 \not\equiv 0 \text{ mod } N)\right] \geq 1 - \frac{1}{2^{k-1}}$$

Proof. [48] Appendix B. □

In other words, the probability of obtaining a number x that fulfills all the conditions of the algorithm is greater than 1/2 in the worst case (i.e., when N has only two different prime factors).

STEP **5**

$$d_1 \leftarrow \gcd(x^{r/2}+1, N)$$
$$d_2 \leftarrow \gcd(x^{r/2}-1, N)$$

As $2 \mid r$ and $x^{r/2}+1 \not\equiv 0 \mod N$, it is easy to see that

$$x^r - 1 \equiv (x^{r/2}-1)(x^{r/2}+1) \equiv 0 \text{ mod } N.$$

We can conclude that d_1 and d_2 are non-trivial factors of N, thus accomplishing the main purpose of the algorithm. As promised, what remains to be seen is the calculus of the multiplicative order of x modulo N with the help of a quantum computer. We proceed to describe this process.

4.5.3 Definition: Quantum part (order finding)

SETUP

$$|\psi_0\rangle_{t,n} \leftarrow |0\rangle_t \otimes |0\rangle_n$$

First, we need a quantum computer with two registers of sizes t and n respectively, where $n = \lceil \log_2 N \rceil$ and $t = 2n$ —the reason behind this will be clear later—. All qubits are initialized at 0.

STEP **2.1**

$$|\psi_1\rangle_{t,n} \leftarrow (\boldsymbol{H}^{\otimes t} \otimes \boldsymbol{I}^{\otimes n}) \left(|\psi_0\rangle_{t,n}\right)$$

This transformation is now a common factor of our quantum algorithms and there is no need of explaining it furthermore. The crucial point is that after its application the first register is in a superposition of all states of the computational basis with equal amplitudes given by $1/\sqrt{2^t}$. More precisely:

$$|\psi_1\rangle_{t,n} = \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle_t \otimes |0\rangle_n$$

STEP **2.2**

$$|\psi_2\rangle_{t,n} \leftarrow \boldsymbol{M}_{x,N}\left(|\psi_1\rangle_{t,n}\right)$$

Let n, t, x and N defined as in the context of this algorithm, the modular exponentiation gate is the unitary operator that has the following effect on the basis states of a quantum system:

$$\boldsymbol{M}_{x,N} : |j\rangle_t \otimes |k\rangle_n \rightarrow |j\rangle \otimes |k + x^j \text{ mod } N\rangle .$$

This transformation is unitary, and its construction takes $\mathcal{O}(\log^3 N)$ steps [105]. Thus, as will be clear at the end of this subsection, it represents the bottleneck of Shor's algorithm.

$$\begin{aligned} |\psi_2\rangle_{t,n} &= \boldsymbol{M}_{x,N}\left(|\psi_1\rangle_{t,n}\right) \\ &= \frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1} \boldsymbol{M}_{x,N}\left(|j\rangle_t \otimes |0\rangle_n\right) \\ &= \frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1} |j\rangle_t \otimes \left|x^j \bmod N\right\rangle_n \end{aligned}$$

Thanks again to quantum parallelism, we have now generated all powers of x modulo N simultaneously. From now on, in order to make the tracking of the algorithm cleaner, we shall suppose that r is a power of 2. In this case, our current quantum state can be expressed as follows:

$$|\psi_2\rangle_{t,n} = \frac{1}{\sqrt{2^t}}\sum_{b=0}^{r-1}\left[\left(\sum_{a=0}^{\frac{2^t}{r}-1} |ar+b\rangle_t\right) \otimes |x^b \bmod N\rangle_n\right]$$

The general case where r may not be a power of 2 is more difficult to express, and is better to explain it in the part devoted to the example.

STEP **2.3**

```
$\tilde{\delta} \leftarrow$ measure the second register

$|\psi_3\rangle_t \leftarrow |\psi_2\rangle_{t,n}$ after measuring the second register
```

Let us suppose that, for a certain $b_0 \in \{0, \dots, r-1\}$, we obtain the value $\tilde{\delta} = x^{b_0} \bmod N$. Thus, the computer is now in the following quantum state:

$$|\psi_3\rangle_t = \sqrt{\frac{r}{2^t}}\sum_{a=0}^{\frac{2^t}{r}-1} |ar+b_0\rangle_t \, .$$

Note that now we only have $2^t/r$ terms in the sum, instead of the previous 2^t ones, and that the value we are looking for (i.e., r) is beginning to surface inside our quantum system in the form of a period. The next step will provide us with a tool capable of extracting this period from a quantum state.

STEP **2.4**

```
$|\psi_4\rangle_t \leftarrow \boldsymbol{F}_n\left(|\psi_3\rangle_t\right)$
```

This step requires that we define a new quantum gate: the quantum discrete Fourier transform, or QFT, whose effect on the quantum basis states is:

$$\boldsymbol{F}_n : |j\rangle_n \to \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{-2\pi ijk/2^n} |k\rangle_n$$

After applying the QFT to the first register, we can express the obtained state as follows:

$$\begin{aligned}
|\psi_4\rangle_t &= \boldsymbol{F}_t\left(|\psi_3\rangle_t\right) \\
&= \sqrt{\frac{r}{2^t}} \sum_{a=0}^{\frac{2^t}{r}-1} \boldsymbol{F}_t(|ar+b_0\rangle_t) \\
&= \sqrt{\frac{r}{2^t}} \sum_{a=0}^{\frac{2^t}{r}-1} \left(\frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{-2\pi ij(ar+b_0)/2^t} \,|\, j\rangle_t \right) \\
&= \frac{1}{\sqrt{r}} \left(\sum_{j=0}^{2^t-1} \left[\frac{r}{2^t} \sum_{a=0}^{\frac{2^t}{r}-1} e^{\frac{-2\pi ija}{2^t/r}} \right] e^{-2\pi ijb_0/2^t} \,|\, j\rangle_t \right) \\
&= \frac{1}{\sqrt{r}} \left(\sum_{k=0}^{r-1} e^{-2\pi i\frac{k}{r}b_0} \left| \frac{k2^t}{r} \right\rangle_t \right)
\end{aligned}$$

Please note that, in order to arrive at the last version of the state, we have just rearranged the summation order while also using the following property of the exponential sums:

$$\frac{1}{N} \sum_{j=0}^{N-1} e^{2\pi ijk/N} = \begin{cases} 1 \text{ if } N \mid k \\ 0 \text{ otherwise,} \end{cases}$$

then we can rewrite the state as:

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \left(\sum_{k=0}^{r-1} e^{-2\pi i\frac{k}{r}b_0} \left| \frac{k2^t}{r} \right\rangle_t \right) |x^{b_0}\rangle_n$$

Now, at last, it is time to measure the first register.

STEP 2.5

$$\boxed{\tilde{\omega} \leftarrow \text{measure } |\psi_4\rangle_t}$$

We obtain, for some unknown $k_0 \in \{0, \ldots, r-1\}$ (the probability of obtaining each of them is the same, but this changes in the general case where r may not be a power of 2):

$$\tilde{\omega} = \frac{k_0 2^t}{r}$$

If $\tilde{\omega} = 0$, we obtain no information and we must come back to the beginning. If, otherwise, $\tilde{\omega} \neq 0$, we can obtain some information about r (or the actual value of r indeed), just by putting $\tilde{\omega}/2^t$ as a fraction in lowest terms and taking the denominator. If we call c that denominator and $x^c \not\equiv 1 \mod N$, then c is a factor of r. If, on the other hand $x^c \equiv 1 \mod N$, then $r = c$ and we have obtained the multiplicative order of x modulo N. In the former case, we should rerun the quantum part of Shor's algorithm with x^c instead of with x, and by repeating this process a finite number of times we shall end up with the correct value of r. Now let us show with an example how Shor's Algorithm works for a particular input.

4.5.4 Example: Factoring 217

We want to find the prime factors of 217 using Shor's Factoring Algorithm. First, the algorithm chooses a random integer x such that $1 < x < 217$. Let us suppose that we obtain $x = 5$, which fulfills the first condition: $\gcd(5, 217) = 1$. Now is the time of finding the multiplicative order of 5 modulo 217, which is achieved with the help of a quantum computer.

We calculate t and n and initialize the quantum system. In this case, $n = \lceil \log_2 217 \rceil = 8$ and consequently $t = 2n = 16$. Thus, our quantum registers have the following initial states:

$$\boxed{|\psi_0\rangle_{16,8} \leftarrow |0\rangle_{16} \otimes |0\rangle_8}$$

Next, we apply the Hadamard transformation, hence obtaining a superposition of all basis states in the first register, all with identical amplitudes. This way, all integers between 0 and $2^{16} - 1$ are now *somewhere* in the first register.

$$\boxed{|\psi_1\rangle_{16,8} \leftarrow (\boldsymbol{H}^{\otimes 16} \otimes \boldsymbol{I}^{\otimes 8}) \left(|\psi_0\rangle_{16,8}\right)}$$

$$|\psi_1\rangle_{16,8} = \frac{1}{\sqrt{2^{16}}} \sum_{j=0}^{2^{16}-1} |j\rangle_{16} \otimes |0\rangle_8$$

Afterwards, we apply the quantum gate that calculates the powers of 5 modulo 217, which has the following effect.

$$\boxed{|\psi_2\rangle_{16,8} \leftarrow \boldsymbol{M}_{5,217} \left(|\psi_1\rangle_{16,8}\right)}$$

$$\begin{aligned}
|\psi_2\rangle_{16,8} &= \boldsymbol{M}_{5,217} \left(|\psi_1\rangle_{16,8}\right) \\
&= \frac{1}{\sqrt{2^{16}}} \sum_{j=0}^{2^{16}-1} \boldsymbol{M}_{5,217} \left(|j\rangle_{16} \otimes |0\rangle_8\right) \\
&= \frac{1}{\sqrt{2^{16}}} \sum_{j=0}^{2^{16}-1} |j\rangle_{16} \otimes \left|5^j \bmod 217\right\rangle_8
\end{aligned}$$

If we expand the sum, we can alternatively express the result as follows (please note that, for the sake of simplicity, we have omitted the subindices and the tensor product operators):

$$\begin{aligned}|\psi_2\rangle = \frac{1}{\sqrt{2^{16}}}(&|0\rangle|1\rangle + |1\rangle|5\rangle + |2\rangle|25\rangle + |3\rangle|125\rangle + |4\rangle|191\rangle + |5\rangle|87\rangle +\\ &|6\rangle|1\rangle + |7\rangle|5\rangle + |8\rangle|25\rangle + |9\rangle|125\rangle + |10\rangle|191\rangle + |11\rangle|87\rangle +\\ &|12\rangle|1\rangle + |13\rangle|5\rangle + |14\rangle|25\rangle + |15\rangle|125\rangle + |16\rangle|191\rangle + |17\rangle|87\rangle +\\ &|18\rangle|1\rangle + |19\rangle|5\rangle + |20\rangle|25\rangle + |21\rangle|125\rangle + |22\rangle|191\rangle + |23\rangle|87\rangle +\\ &|24\rangle|1\rangle + |25\rangle|5\rangle + |26\rangle|25\rangle + |27\rangle|125\rangle + |28\rangle|191\rangle + |29\rangle|87\rangle +\\ &|30\rangle|1\rangle + ...)\end{aligned}$$

After a close inspection, we can observe that the values on the second register are periodic. If we make common factor, we end up with the state:

$$\begin{aligned}|\psi_2\rangle = \frac{1}{\sqrt{2^{16}}}\big(&(|0\rangle + |6\rangle + |12\rangle + |18\rangle + ... + |65526\rangle + |65532\rangle)|1\rangle +\\ &(|1\rangle + |7\rangle + |13\rangle + |19\rangle + ... + |65527\rangle + |65533\rangle)|5\rangle +\\ &(|2\rangle + |8\rangle + |14\rangle + |20\rangle + ... + |65528\rangle + |65534\rangle)|25\rangle +\\ &(|3\rangle + |9\rangle + |15\rangle + |21\rangle + ... + |65529\rangle + |65535\rangle)|125\rangle +\\ &(|4\rangle + |10\rangle + |16\rangle + |22\rangle + ... + |65530\rangle)|191\rangle +\\ &(|5\rangle + |11\rangle + |17\rangle + |23\rangle + ... + |65531\rangle)|87\rangle\big)\end{aligned}$$

Thanks to this representation, it is easier to understand what will happen if we measure the second register.

```
$\tilde{\delta} \leftarrow$ measure the second register

$|\psi_3\rangle_{16} \leftarrow |\psi_2\rangle_{16,8}$ after measuring the second register
```

It is clear that we shall obtain a value $\tilde{\delta}$ such that $\tilde{\delta} = 5^{\tilde{j}} \bmod 217$ for a certain $\tilde{j} \in \{0, \ldots, 2^{16} - 1\}$. Thus, $\tilde{\delta} \in \{1, 5, 25, 125, 191, 87\}$ (which are the powers of 5 modulo 217). Let us suppose that we get $\tilde{\delta} = 25$. The register will collapse into $|25\rangle$ and all other possible values will be destroyed and gone forever, the information about the rest of possible powers of 5 modulo 217 lost. However, the first register will also collapse into the values that were tensored with $|25\rangle$, thus discarding the remaining ones. What interests us is that all the basis states tensored with $|25\rangle$ correspond with the exponents $\tilde{j}$ such that $\tilde{\delta} = 5^{\tilde{j}} \bmod 217$. More specifically:

$$\begin{aligned}|\psi_3\rangle_{16} &= \frac{1}{\sqrt{10923}}(|2\rangle_{16} + |8\rangle_{16} + |14\rangle_{16} + |20\rangle_{16} + ... + |65528\rangle_{16} + |65534\rangle_{16})\\ &= \frac{1}{\sqrt{10923}}\left(\sum_{a=0}^{10922} |6a+2\rangle_{16}\right)\end{aligned}$$

A pattern has arisen, as the basis states on the first register display some periodic behavior. This period naturally corresponds with the multiplicative order of 5 modulo 217, which happens to be equal to 6. Of course, this information is yet hidden to us, we are just using some knowledge of the problem for providing a mathematical explanation of the performance of the algorithm in this particular case. On the other hand, the constant 10923 is just a normalization of the amplitudes after the collapse of the register, corresponding to the total number of exponents that return 25 modulo 217 between 0 and $2^{16}-1$.

As explained previously, if we want to obtain the period from within the bowels of our quantum system, we should use the quantum Fourier transform:

$$\boxed{|\psi_4\rangle_{16} \leftarrow \boldsymbol{F}_{16}\left(|\psi_3\rangle_{16}\right)}$$

$$\begin{aligned}
|\psi_4\rangle_{16} &= \boldsymbol{F}_{16}\left(|\psi_3\rangle_{16}\right)\\
&= \boldsymbol{F}_{16}\left(\frac{1}{\sqrt{10923}}\sum_{a=0}^{10922}|6a+2\rangle_{16}\right)\\
&= \frac{1}{\sqrt{10923}}\sum_{a=0}^{10922}\boldsymbol{F}_{16}\big(\,|6a+2\rangle_{16}\,\big)\\
&= \frac{1}{\sqrt{10923}}\sum_{a=0}^{10922}\left(\frac{1}{\sqrt{2^{16}}}\sum_{k=0}^{2^{16}-1}e^{-2\pi i\frac{(6a+2)k}{2^{16}}}\,|k\rangle_{16}\right)\\
&= \frac{1}{\sqrt{2^{16}}}\sum_{k=0}^{2^{16}-1}\left(\left[\frac{1}{\sqrt{10923}}\sum_{a=0}^{10922}e^{-2\pi i\frac{6ak}{2^{16}}}\right]e^{-2\pi i\frac{2k}{2^{16}}}\,|k\rangle_{16}\right)
\end{aligned}$$

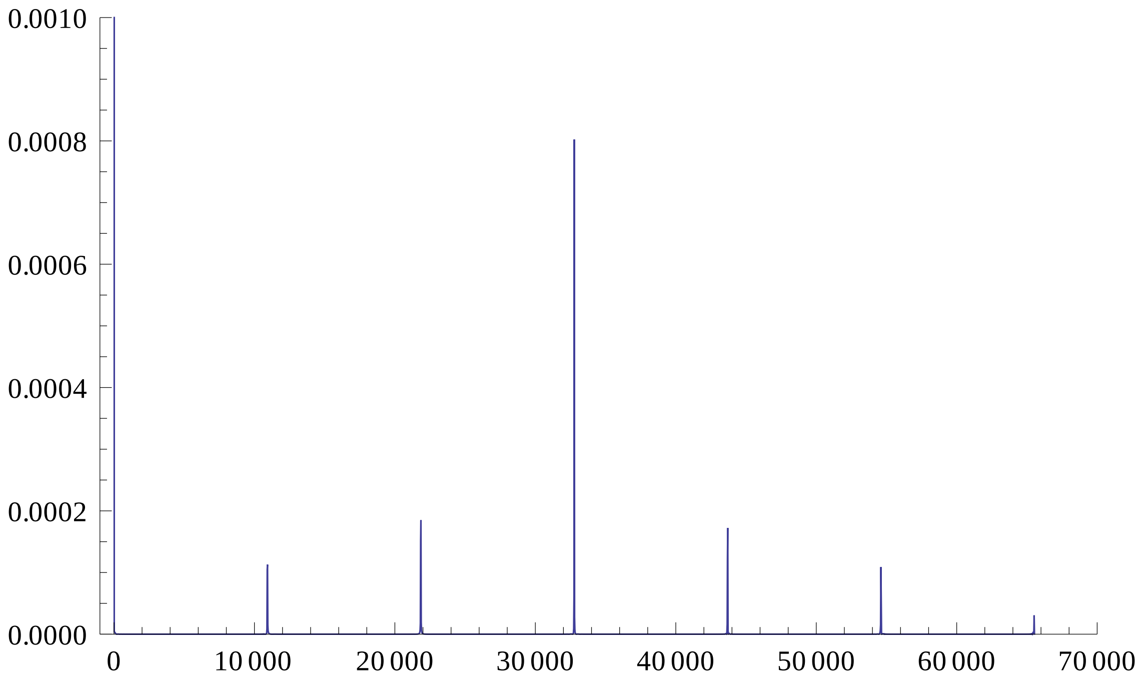
Thus, we end up again with a distribution of the amplitudes defined by

$$\alpha_k = \frac{1}{\sqrt{2^{16}\cdot 10923}}\left(\sum_{a=0}^{10922}e^{-2\pi i\frac{6ak}{2^{16}}}\right)e^{-2\pi i\frac{2k}{2^{16}}},$$

which gives us the following probability distribution:

$$|\alpha_k|^2 = \frac{1}{2^{16}\cdot 10923}\left|\sum_{a=0}^{10922}e^{-2\pi i\frac{6ja}{2^{16}}}\right|^2$$

with $k=0,\ldots,2^{16}-1$. If we represent this distribution in a graph, it is easier to understand the final steps of the algorithm.



$$\boxed{\tilde{\omega} \leftarrow \text{ measure } |\psi_4\rangle_{16})}$$

We then measure the first register, and obtain non-deterministically a value from one of the seven peaks. The first peak yields a 0, thus forcing us to start again the quantum part of the algorithm. If, otherwise, we obtain a value from the other six peaks, we can retrieve (using continued fractions) the order r or a factor of r from it.

Let us explain this last step more formally (we took details for this from [84]). We recall that a continued fraction is represented as

$$[a_0; a_1, \ldots, a_K] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\cdots + \cfrac{1}{a_K}}}}}$$

where $a_0 \in \mathbb{Z}_{\geq 0}$ and $a_1, \ldots, a_K \in \mathbb{Z}_{>0}$. Let $[a_0; a_1, \ldots, a_K]$, then there exists a unique $q \in \mathbb{Q}_{>0}$ such that $q = [a_0; a_1, \ldots, a_K]$. We define the k-th convergent of $[a_0; a_1, \ldots, a_K]$, where $0 \leq k < K$, as

$$q_k = [a_0; a_1 \ldots, a_k].$$

Provided a certain $q \in \mathbb{Q}_{>0}$, we can compute the members of its corresponding continued fraction as follows:

$$\begin{aligned} a_0 &\leftarrow \lfloor q \rfloor \\ q_0 &\leftarrow q - a_0 \\ a_{k+1} &\leftarrow \lfloor 1/q_k \rfloor \\ q_{k+1} &\leftarrow 1/q_k - a_{k+1} \end{aligned}$$

Even more, each of the convergents q_k can be expressed as $q_k = b_k/c_k$, where $\gcd(b_k, c_k) = 1$ and

$$\begin{aligned} b_0 &\leftarrow a_0 \\ c_0 &\leftarrow 1 \\ b_1 &\leftarrow a_0 a_1 + 1 \\ c_1 &\leftarrow a_1 \\ b_{k+2} &\leftarrow a_{k+2} b_{k+1} + b_k \\ c_{k+2} &\leftarrow a_{k+2} c_{k+1} + c_k \end{aligned}$$

It can be proven [84] that we can obtain the period r (or a non trivial factor of it) from $\tilde{w}$ with the following algorithm: starting at $k = 1$, we compute b_k and c_k as previously explained from the continued fraction of $q = \tilde{w}/2^n$. Then, we check if

$$x^{c_k} \equiv 1 \mod N.$$

If the answer is affirmative, we have obtained the order r or a factor of it; if not, we do it again for $k + 1$.

For example, a possible value from the second peak is 10915, and if we divide it by 2^{16} we obtain the continued fraction

$$\frac{10915}{65536} = [0; 6, 237, 3, 1, 1, 6],$$

whose first convergent is $\frac{1}{6}$. Thus, as $5^6 \equiv 1 \mod 217$ we have that the order (or a factor of it) is 6.

Another possible example is 32763, a value from the middle peak. Doing the same operation:

$$\frac{32763}{65536} = [0; 2, 3276, 3, 3]$$

from where we obtain a first convergent of $\frac{1}{2}$. In both cases, we have to verify if 6 and 2 are, respectively, the order of 5. The first case returns a positive answer, but the second forces us to restart the algorithm to get the order of 5^2. In any case, after a finite number of iterations we finally obtain $r = 6$.

Remains to be seen if $x = 5$ and $r = 6$ fulfill the final conditions of the algorithm. As 6 is an even number, and $5^{6/2} + 1 = 126 \neq 0 \bmod 217$, we can finally proceed to the last step. As explained, we can calculate now two non-trivial factors of 217:

$$d_1 = \gcd(126, 217) = 7$$

and

$$d_2 = \gcd(124, 217) = 31,$$

which in fact are the only prime factors of 217. Thus ends Shor's algorithm for this particular case.

$$\boxed{217 = 7 \times 31}$$

The first experimental demonstration of Shor's factoring algorithm came in 2001, [113] when a group at IBM factored 15 into 3 and 5 using a nuclear magnetic resonance (NMR) quantum computer with seven spin-1/2 nuclei in a molecule as qubits. In 2007, Shor's algorithm was implemented with photonic qubits by two different groups [80, 85], with both of them observing quantum entanglement in the process. In 2012, number 143 was factored with the help of an adiabatic quantum computer [114].

4.6 Grover's Search Algorithm

4.6.1 Definition

Grover's search algorithm was first described by Indian-American computer scientist Lov K. Grover (b. 1961) in [63, 64, 65], hence its name. The original aim of Grover's algorithm is the following: we have an unstructured and disorganized database with 2^n elements, identified from now on with the indices $0, \dots, 2^n - 1$, and we want to find one that satisfies a certain property. The algorithm leans on two hypotheses: first, that such an element exists inside the database; and second, that this element is unique.

If we are to search for this mentioned element with a classical computer, in the worst case we will have to check all members of the database, which tells us that this problem has a classical computational complexity of $\mathcal{O}(2^n)$. As will be shown, Grover's quantum search algorithm will make only $\mathcal{O}(\sqrt{2^n})$ queries to the database, thus strictly improving the performance of its classical counterpart.

$\mathbb{SETUP}$

$$\boxed{|\psi_0\rangle_{n,1} \leftarrow |0\rangle_n \otimes |1\rangle}$$

We need a quantum computer with $n+1$ qubits, where the first n qubits will be initialized at $|0\rangle$ and the remaining one at $|1\rangle$.

$\mathbb{STEP}$ **1**

$$\boxed{|\psi_1\rangle_{n,1} \leftarrow \boldsymbol{H}^{\otimes n+1}\left(|\psi_0\rangle_{n,1}\right)}$$

In the first proper step of the algorithm, we apply the Hadamard quantum gate to all the qubits in our system. This way, the obtained result is a combination of all basis states inside our first n qubits, and the $|-\rangle$ state in the remaining one.

$$\begin{aligned}
|\psi_1\rangle_{n,1} &= \boldsymbol{H}^{\otimes n+1}\left(|\psi_0\rangle_{n,1}\right)\\
&= \boldsymbol{H}^{\otimes n+1}\left(|0\rangle_n \otimes |1\rangle\right)\\
&= \left(\boldsymbol{H}^{\otimes n}|0\rangle_n\right) \otimes \left(\boldsymbol{H}|1\rangle\right)\\
&= \left(\boldsymbol{H}|0\rangle\right)^{\otimes n} \otimes \left(\boldsymbol{H}|1\rangle\right)\\
&= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)^{\otimes n} \otimes \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)\\
&= \left(\frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}|j\rangle_n\right) \otimes |-\rangle
\end{aligned}$$

For the sake of simplicity, we reintroduce a useful quantum state that will appear many times throughout the rest of the algorithm. It will also be needed in order to define one of the key quantum gates of Grover's Search method, as will be shown in the next step.

$$|\gamma\rangle_n = \frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}|j\rangle_n$$

This definition helps us to express our quantum state as follows:

$$|\psi_1\rangle_{n,1} = |\gamma\rangle_n \otimes |-\rangle$$

$\mathbb{STEP}$ **2.1**

$$\boxed{|\psi_2\rangle_{n,1} \leftarrow \boldsymbol{O}_f\left(|\psi_1\rangle_{n,1}\right)}$$

The next step of the algorithm leans on the following assumption: we can build a quantum gate, called $\boldsymbol{O}_f$, that makes use of a function capable of recognizing the element of the database we are searching for. This function f can be defined as follows (note that the desired element is identified by the unknown index $j_0 \in \{0, \dots, 2^n - 1\}$).

$$f(j) = \begin{cases} 1 & \text{if } j = j_0 \\ 0 & \text{otherwise} \end{cases}$$

The $\boldsymbol{O}_f$ quantum gate is in fact the oracle gate described in previous algorithms, which has the following effect on the basis states of a $n+1$-qubit system

$$\boldsymbol{O}_f : |j\rangle_n \otimes |k\rangle \rightarrow |j\rangle_n \otimes |k \oplus f(j)\rangle\,.$$

We recall from Deutsch-Jozsa algorithm that the oracle gate has the following effect on the state $|j\rangle_n \otimes |-\rangle$:

$$\boldsymbol{O}_f\Big(\,|j\rangle_n \otimes |-\rangle\,\Big) = (-1)^{f(j)}\,|j\rangle_n \otimes |-\rangle$$

It can be seen that, in those cases, $\boldsymbol{O}_f$ inverts the sign of the amplitude corresponding to the basis state that codifies the searched element in the first n qubits, while keeping intact the rest of the amplitudes.

In order to make the explanation and understanding of the algorithm easier and simpler, we introduce another n-qubit quantum state,

$$|\rho\rangle_n = \frac{1}{\sqrt{2^n-1}} \sum_{\substack{j=0\\ j\neq j_0}}^{2^n-1} |j\rangle_n\,,$$

whose relationship with the aforedescribed state $|\gamma\rangle_n$ is

$$|\gamma\rangle_n = \frac{\sqrt{2^n-1}}{\sqrt{2^n}}\,|\rho\rangle_n + \frac{1}{\sqrt{2^n}}\,|j_0\rangle_n\,.$$

Please note that $|\rho\rangle$ depends on the value of j_0, but we shall write $|\rho\rangle$ instead of $|\rho\,(j_0)\rangle$ for the sake of simplicity. It can be observed that the introduction of $|\rho\rangle_n$ helps in the separation of the searched elemenet $|j_0\rangle$ from the rest of the quantum basis states.

Thus, after applying $\boldsymbol{O}_f$ to our quantum system it will look like this:

$$\begin{aligned}
|\psi_2\rangle_{n,1} &= \boldsymbol{O}_f\Big(\,|\psi_1\rangle_{n,1}\Big)\\
&= \boldsymbol{O}_f\Big(\,|\gamma\rangle_n \otimes |-\rangle\Big)\\
&= \boldsymbol{O}_f\left(\left(\frac{\sqrt{2^n-1}}{\sqrt{2^n}}\,|\rho\rangle_n + \frac{1}{\sqrt{2^n}}\,|j_0\rangle_n\right) \otimes |-\rangle\right)\\
&= \left(\frac{\sqrt{2^n-1}}{\sqrt{2^n}}\,|\rho\rangle_n - \frac{1}{\sqrt{2^n}}\,|j_0\rangle_n\right) \otimes |-\rangle\\
&= \left(\,|\gamma\rangle_n - \frac{2}{\sqrt{2^n}}\,|j_0\rangle_n\right) \otimes |-\rangle
\end{aligned}$$

The motivation behind the last interpretation of $|\psi_2\rangle_{n,1}$ will become clear in the next step.

STEP **2.2**

$$\boxed{\left|\psi_3\right\rangle_{n,1} \leftarrow (\boldsymbol{\Gamma}_n \otimes \boldsymbol{I}) \left(\left|\psi_2\right\rangle_{n,1}\right)}$$

For this step, we must construct a new quantum gate, denoted by $\boldsymbol{\Gamma}_n$, that will affect only the first n qubits. The remaining qubit will remain intact —this is represented with the single-qubit identity gate $\boldsymbol{I}$—. The definition of $\boldsymbol{\Gamma}_n$ is:

$$\boldsymbol{\Gamma}_n = 2\left|\gamma\right\rangle_n \left\langle\gamma\right|_n - \boldsymbol{I}^{\otimes n}$$

Let us see what happens when we apply this new quantum gate to the first n qubits of our quantum state $\left|\psi_2\right\rangle_{n,1}$ defined in the previous step.

$$\begin{aligned}
\boldsymbol{\Gamma}_n\left(\left|\psi_2\right\rangle_n\right) &= \left(2\left|\gamma\right\rangle_n \left\langle\gamma\right|_n - \boldsymbol{I}^{\otimes n}\right)\left(\left|\gamma\right\rangle_n - \frac{2}{\sqrt{2^n}}\left|j_0\right\rangle_n\right) \\
&= 2\left|\gamma\right\rangle_n \left\langle\gamma|\gamma\right\rangle_n - \frac{4}{\sqrt{2^n}}\left|\gamma\right\rangle_n\left\langle\gamma|j_0\right\rangle_n - \left|\gamma\right\rangle_n + \frac{2}{\sqrt{2^n}}\left|j_0\right\rangle_n \\
&= 2\left|\gamma\right\rangle_n - \frac{4}{2^n}\left|\gamma\right\rangle_n - \left|\gamma\right\rangle_n + \frac{2}{\sqrt{2^n}}\left|j_0\right\rangle_n \\
&= \frac{2^{n-2}-1}{2^{n-2}}\left|\gamma\right\rangle_n + \frac{2}{\sqrt{2^n}}\left|j_0\right\rangle_n
\end{aligned}$$

For this, we have used the definition of the inner product seen in Section 3.3 to calculate

$$\left\langle\gamma|\gamma\right\rangle_n = 1$$

and

$$\left\langle\gamma|i_0\right\rangle_n = \frac{1}{\sqrt{2^n}}.$$

The $\boldsymbol{\Gamma}_n$ gate is also called Grover diffusion operator, and can also be seen as

$$\boldsymbol{\Gamma}_n = (\boldsymbol{H}^{\otimes n})(2\left|0\right\rangle_n\left\langle 0\right|_n - \boldsymbol{I}^{\otimes n})(\boldsymbol{H}^{\otimes n}),$$

which is a much more painless way of implementing it in practice. Steps 2.1 and 2.2 are usually treated as a single step, represented by the Grover gate

$$\boldsymbol{G} = (\boldsymbol{\Gamma}_n \otimes \boldsymbol{I})(\boldsymbol{O}_f).$$

STEP **3**

In order to achieve the objective of Grover's algorithm, one must apply the $\boldsymbol{G}$ gate repeatedly until the probability of obtaining the index j_0 is maximal —a general overview of the algorithm can be seen in Figure 4.4—. After the desired probability has been reached, we measure the first n-qubit register, and obtain the index j_0. The optimal number of times $\boldsymbol{G}$ is applied has order $\mathcal{O}(\sqrt{2^n})$, which will be proved later. But first, let us explain the behavior of the algorithm with an example.

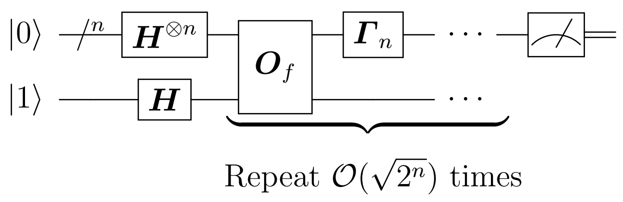


Figure 4.4: Circuit representation of Grover's Search Algorithm

4.6.2 Example with $n = 4$ qubits

Let us illustrate how Grover's Search works with this exemplifying case: suppose we have an unstructured database with its elements listed as $0, 1, \ldots, 15$ indexed via 4 qubits, and that we want to find a certain item that is indexed with the number 7 —note that we do not know this yet, but it can be supposed in order to explain the algorithm straightforwardly—.

As previously explained, we need a quantum computer with $n+1$ qubits, where n is the number of bits needed for codifying the indices of the members of the database —in this case, $n = 4$—. In order to correctly setup our quantum system, the first 4 qubits must be in the state $|0\rangle$ and the remaining one must be in the state $|1\rangle$. Thus, our quantum system begins as follows:

$$|\psi_0\rangle_{4,1} = |0\rangle_4 \otimes |1\rangle$$

The first transformation we apply to the quantum system is the Hadamard gate, or $\boldsymbol{H}$, which converts it to the following state:

$$\begin{aligned} |\psi_1\rangle_{4,1} &= (\boldsymbol{H}^{\otimes 4} |0\rangle_4) \otimes (\boldsymbol{H} |1\rangle) \\ &= \Big(\frac{1}{4}\sum_{j=0}^{15} |j\rangle_4\Big) \otimes |-\rangle \end{aligned}$$

We remind the reader that from now on we will make use of the states $|\gamma\rangle_4$ and $|\rho\rangle_4$ in the interest of simplifying the writing of the whole process.

$$|\gamma\rangle_4 = \frac{1}{4}\sum_{j=0}^{15} |j\rangle_4$$

$$|\rho\rangle_4 = \frac{1}{\sqrt{15}}\sum_{\substack{j=0 \\ j\neq 7}}^{15} |j\rangle_4$$

$$|\gamma\rangle_4 = \frac{\sqrt{15}}{4} |\rho\rangle_4 + \frac{1}{4} |7\rangle_4$$

The introduction of these states allows us to write $|\psi_1\rangle_{4,1}$ as:

$$|\psi_1\rangle_{4,1} = \left(\frac{\sqrt{15}}{4}|\rho\rangle_4 + \frac{1}{4}|7\rangle_4\right) \otimes |-\rangle$$

On $\mathbb{STEP}$ 2.1 we make use of the transformation $\boldsymbol{O}_f$, which applies to all possible states inside the first 4 qubits an oracle f that identifies 7 as the correct index for the element we are looking for.

$$\begin{aligned}
|\psi_2\rangle_{4,1} &= \boldsymbol{O}_f(|\psi_1\rangle_{4,1}) \\
&= \boldsymbol{O}_f\left[\left(\frac{\sqrt{15}}{4}|\rho\rangle_4 + \frac{1}{4}|7\rangle_4\right) \otimes |-\rangle\right] \\
&= \left(\frac{\sqrt{15}}{4}|\rho\rangle_4 - \frac{1}{4}|7\rangle_4\right) \otimes |-\rangle \\
&= \left(|\gamma\rangle_4 - \frac{1}{2}|7\rangle_4\right) \otimes |-\rangle
\end{aligned}$$

On $\mathbb{STEP}$ 2.2, we apply the quantum gate $\boldsymbol{\Gamma}_4$ to the first 4 qubits of the computer. We use $|\rho\rangle_4$ to make more clear which part of the state has the index 7 in it and which one does not have it.

$$\begin{aligned}
\boldsymbol{\Gamma}_4\Big(|\gamma\rangle_4 - \frac{1}{2}|7\rangle_4\Big) &= (2|\gamma\rangle_4\langle\gamma|_4 - I)\left(|\gamma\rangle_4 - \frac{1}{2}|7\rangle_4\right) \\
&= 2|\gamma\rangle_4\langle\gamma|\gamma\rangle_4 - |\gamma\rangle_4 - |\gamma\rangle_4\langle\gamma|7\rangle_4 + \frac{1}{2}|7\rangle_4 \\
&= 2|\gamma\rangle_4 - |\gamma\rangle_4 - \frac{1}{4}|\gamma\rangle_4 + \frac{1}{2}|7\rangle_4 \\
&= \frac{3}{4}|\gamma\rangle_4 + \frac{1}{2}|7\rangle_4 \\
&= \frac{3\sqrt{15}}{16}|\rho\rangle_4 + \frac{11}{16}|7\rangle_4
\end{aligned}$$

$$|\psi_3\rangle_{4,1} = \Big(\frac{3\sqrt{15}}{16}|\rho\rangle_4 + \frac{11}{16}|7\rangle_4\Big) \otimes |-\rangle$$

Thus, we have completed the first iteration of $\boldsymbol{G}$, and if we are to measure our quantum state now, we have a probability

$$p = \Big(\frac{11}{16}\Big)^2 = \frac{121}{256} \approx 0.4726$$

of obtaining the index 7, and a probability

$$\bar{p} = \Big(\frac{3\sqrt{15}}{16}\Big)^2 = \frac{135}{256} \approx 0.5274$$

of obtaining any other index. As can be seen, the probability of finding the searched element is greater than of finding any other element, but is still not big enough, which tells us that at least another round of the algorithm is needed.

We repeat again the application of $\boldsymbol{G}$. First, we apply $\boldsymbol{O}_f$ and end up with the following quantum state, defined again as a combination of $|\gamma\rangle_4$ and $|7\rangle_4$.

$$\begin{aligned}|\psi_4\rangle_{4,1} &= \boldsymbol{O}_{f,4}(|\psi_3\rangle_{4,1})\\ &= \left(\frac{3\sqrt{15}}{16}|\rho\rangle_4 - \frac{11}{16}|7\rangle_4\right)\otimes|-\rangle\\ &= \left(\frac{3}{4}|\gamma\rangle_4 - \frac{14}{16}|7\rangle_4\right)\otimes|-\rangle\end{aligned}$$

And then we apply $\boldsymbol{\Gamma}_4$.

$$\begin{aligned}|\psi_5\rangle_4 &= (2|\gamma\rangle_4\langle\gamma|_4 - I)\left(\frac{3}{4}|\gamma\rangle_4 - \frac{14}{16}|7\rangle_4\right)\\ &= \frac{3}{2}|\gamma\rangle_4\langle\gamma|\gamma\rangle_4 - \frac{7}{4}|\gamma\rangle_4\langle\gamma|7\rangle_4 - \frac{3}{4}|\gamma\rangle_4 + \frac{7}{8}|7\rangle_4\\ &= \frac{5}{16}|\gamma\rangle_4 + \frac{7}{8}|7\rangle_4\\ &= \frac{5\sqrt{15}}{64}|\rho\rangle_4 + \frac{61}{64}|7\rangle_4\end{aligned}$$

Thus, after two iterations of $\boldsymbol{G}$ we end up with a state whose possibility of returning the index i_0 is

$$p = \left(\frac{61}{64}\right)^2 = \frac{3721}{4096} \approx 0.9084,$$

which gives us a fairly enough chance of completing the execution of Grover's Search Algorithm successfully. However, how can we be sure that we have obtained the maximum probability of retrieving the desired element from the database if we cannot measure the amplitudes inside our quantum register? Even more, whence comes the required order of $\mathcal{O}(\sqrt{2^n})$ needed in the number of iterations of $\boldsymbol{G}$?

Both questions are pivotal in the correct performance of the algorithm. Let us show their significance with a simple example: what happens if we continue applying the $\boldsymbol{G}$ gate to our quantum system?

If we apply $\boldsymbol{G}$ once more, we will have the following two states, the first after $\boldsymbol{O}_f$ and the second after $\boldsymbol{\Gamma}_4$.

$$|\psi_6\rangle = \frac{5}{16}|\gamma\rangle_4 - \frac{33}{32}|7\rangle_4$$

$$|\psi_7\rangle = -\frac{13\sqrt{15}}{256}\,|\rho\rangle_4 - \frac{251}{256}\,|7\rangle_4$$

As can be seen, the probability of obtaining index 7 now is

$$p = \left(\frac{251}{256}\right)^2 \approx 0.9613.$$

Yet, if we are still not satisfied with a 96% chance of success, we can run over $\boldsymbol{G}$ once again, and obtain first

$$|\psi_8\rangle = -\frac{13}{64}\,|\gamma\rangle_4 - \frac{238}{256}\,|7\rangle_4$$

and finally

$$|\psi_9\rangle = -\frac{342\sqrt{15}}{2048}\,|\rho\rangle_4 + \frac{1562}{2048}\,|7\rangle_4\,,$$

which gives us a probability

$$p = \left(\frac{1562}{2048}\right)^2 \approx 0.5817$$

of obtaining $|7\rangle_4$. Yes, it seems that to unabatedly perform numberless iterations of $\boldsymbol{G}$ does not guarantee a continuous increment in the probability of success.[2] Even more, it can waste all previously done work. We shall see why this has happened in the next section.

4.6.3 Proof of correctness

In this section we sketch a proof of the correctness of Grover's algorithm. The main ideas behind this proof are taken from [28], and will come in handy in the next subsection, where different and more general versions of the database search algorithm are discussed.

Theorem 4.6.1. *Grover's Search Algorithm for a unique solution needs $m \sim \mathcal{O}(\sqrt{2^n})$ iterations of $\boldsymbol{G}$ for maximizing the probability of obtaining the desired element with unknown index j_0.*

Proof. First, we are interested in redefining all the possible states that occur during the execution of the algorithm as a function of the different amplitudes involved. As was shown earlier, the only amplitude that will differ from the rest after every Grover's iteration is the one associated with the basis state that identifies the searched element. Thus, suffices to define the generic state in the following way:

$$|\psi(\alpha,\beta)\rangle_n = \alpha\,|j_0\rangle_n + \beta \sum_{\substack{j=0\\ j\neq j_0}}^{2^n-1} |j\rangle_n\,.$$

[2] *"There is thy gold, worse poison to men's souls."* (Romeo and Juliet, Act 5, Scene 1)

With amplitudes α and β constrained to

$$|\alpha|^2 + (2^n - 1)|\beta|^2 = 1.$$

Note that we are only having into account the first register, the one with n qubits. The remaining qubit will behave as explained before, starting at $|1\rangle$ and remaining as $|-\rangle$ throughout the rest of the algorithm, but for the sake of simplicity will be omitted during the proof.

Let us see what happens when we apply all quantum gates that make up the Grover transform $\boldsymbol{G}_n$ to a generic state $|\psi(\alpha,\beta)\rangle_n$. First, we employ $\boldsymbol{O}_f$, which recognizes the searched element and flips its amplitude. Thus, after $\boldsymbol{O}_f$ we obtain

$$|\psi'(\alpha,\beta)\rangle_n = -\alpha\,|j_0\rangle_n + \beta \sum_{\substack{j=0\\ j\neq j_0}}^{2^n-1} |j\rangle_n$$

In order to apply the $\boldsymbol{\Gamma}_n$ quantum gate, it is interesting to previously see our current state as a function of $|\gamma\rangle_n$, as was done before:

$$|\psi'(\alpha,\beta)\rangle_n = -(\alpha+\beta)\,|j_0\rangle_n + \sqrt{2^n}\beta\,|\gamma\rangle_n\,.$$

We are finally in condition of applying $\boldsymbol{\Gamma}_n$:

$$\begin{aligned}
|\psi''(\alpha,\beta)\rangle_n &= \boldsymbol{\Gamma}_n\Big(\,|\psi'(\alpha,\beta)\rangle_n\Big)\\
&= \boldsymbol{\Gamma}_n\Big(-(\alpha+\beta)\,|j_0\rangle_n + \sqrt{2^n}\beta\,|\gamma\rangle_n\Big)\\
&= \Big(2\,|\gamma\rangle_n\langle\gamma|_n - \boldsymbol{I}^{\otimes n}\Big)\Big(-(\alpha+\beta)\,|j_0\rangle_n + \sqrt{2^n}\beta\,|\gamma\rangle_n\Big)\\
&= (\alpha+\beta)\,|j_0\rangle_n + \Big(2\sqrt{2^n}\beta - \frac{2}{\sqrt{2^n}}(\alpha+\beta) - \sqrt{2^n}\beta\Big)\,|\gamma\rangle_n\\
&= \Big(\frac{2^{n-1}-1}{2^{n-1}}\alpha + \frac{2^n-1}{2^{n-1}}\beta\Big)\,|j_0\rangle_n + \Big(-\frac{1}{2^{n-1}}\alpha + \frac{2^{n-1}-1}{2^{n-1}}\beta\Big)\,|\gamma\rangle_n
\end{aligned}$$

Now that we know the effect $\boldsymbol{G}_n$ makes to a generic state, we are in condition of predicting in which iteration of the algorithm we have more possibilities of obtaining the desired element i_0. If we define

$$|\psi_{k+1}(\alpha_{k+1},\beta_{k+1})\rangle_n = \boldsymbol{G}_n(|\psi_k(\alpha_k,\beta_k)\rangle_n)$$

where

$$\alpha_1 = \beta_1 = \frac{1}{\sqrt{2^n}}$$

and

$$\begin{aligned}
\alpha_{k+1} &= \frac{2^{n-1}-1}{2^{n-1}}\alpha_k + \frac{2^n-1}{2^{n-1}}\beta_k\\
\beta_{k+1} &= -\frac{1}{2^{n-1}}\alpha_k + \frac{2^{n-1}-1}{2^{n-1}}\beta_k
\end{aligned}$$

for $j \geq 1$, then we can try to find a more tractable closed-form formula for the amplitude of i_0. Note that, for $j = 0$, it is not possible to define $|\psi_0\rangle$ as a function of α_0 and β_0, thus only the cases where $j \geq 1$ will be defined as such.

If we designate θ such that
$$\sin^2\theta = \frac{1}{2^n},$$
we can easily prove by induction that
$$\alpha_j = \sin((2j-1)\theta)$$
and that
$$\beta_j = \frac{1}{\sqrt{2^n-1}}\cos((2j-1)\theta).$$

Let us suppose now that, for an unknown step $j = m+1$ (note that m is equivalent to the number of times we have applied $\boldsymbol{G}_n$), we want to assure that $\alpha_m = 1$. This occurs when
$$(2m+1)\theta = \frac{\pi}{2},$$
and expressly, when
$$m = \frac{\pi - 2\theta}{4\theta}.$$

Obviously, we can not perform a non-integer number of iterations of $\boldsymbol{G}_n$. If we take
$$m = \left\lfloor \frac{\pi}{4\theta} \right\rfloor,$$
we can conclude that the number of iterations of $\boldsymbol{G}_n$ needed for achieving the maximum probability of success is close $\frac{\pi}{4}\sqrt{2^n}$ (note that
$$\theta \approx \sin\theta = \frac{1}{\sqrt{2^n}}$$
when 2^n is large enough). Thus, we can conclude that the number m of iterations of $\boldsymbol{G}_n$ has order
$$m \sim \mathcal{O}(\sqrt{2^n}).$$

□

4.6.4 Mutiple solutions

The main limitation of Grover's Search Algorithm deals with the number of solutions: it assumes that there is one and only one element in the database that matches our search. As will be seen in Chapter 5, there are many cases in which Grover's Search may prove useful, but in which the number of solutions is unknown, or maybe we do not even know if there is indeed a solution. In this subsection we proceed to explain an alternate version of Grover's Algorithm that originally appeared in [28] which takes care of this drawback.

Let us suppose that we have an unstructured database, indexed by $0, 1, \ldots, 2^n - 1$. We are interested in finding an element inside the database that fulfills a certain property, but we do not know if such an element exists, or if there are more than one. We name $A \subseteq \{0, 1, \ldots, 2^n - 1\}$ the set of possible solutions, with $|A| = t$ and $t \in \{0, 1, \ldots, 2^n - 1\}$. At first sight, one can only hope to just obtain the same performance results as in the original Grover's Algorithm just by applying $\boldsymbol{G}$ the same number of steps. However, it is easy to find a counterexample showing that the probability of success after $\frac{\pi}{4}\sqrt{2^n}$ iterations changes dramatically when $t \neq 1$.

In order to show how this variation of the algorithm works, we define $B = \overline{A}$, with $|B| = 2^n - t$. Following a similar approach, we can assume that every quantum state of our system after first applying globally the Hadamard transform can be expressed as

$$|\psi(\alpha, \beta)\rangle_n = \alpha \sum_{i \in A} |i\rangle_n + \beta \sum_{i \in B} |i\rangle_n ,$$

where

$$t\alpha^2 + (2^n - t)\beta^2 = 1.$$

We only take into account the first n qubits, as previously done. We would like to clarify that the algorithm is essentially the same, so there is no need for explaining it again. The only substantial change is the expected number of iterations of $\boldsymbol{G}$ needed for maximizing the probability of success. Thus, if we apply $\boldsymbol{O}_f$ to a certain state, we end up with the following configuration:

$$|\psi'(\alpha, \beta)\rangle_n = -\alpha \sum_{i \in A} |i\rangle_n + \beta \sum_{i \in B} |i\rangle_n$$

Just like before, after a complete iteration of G the quantum system is in the state:

$$\boldsymbol{\Gamma}_n\Big(|\psi'(\alpha, \beta)\rangle_n \Big) = \Big(\frac{2^{n-1} - t}{2^{n-1}}\alpha + \frac{2^n - t}{2^{n-1}}\beta\Big) \sum_{i \in A} |i\rangle_n + \Big(-\frac{t}{2^{n-1}}\alpha_j + \frac{2^{n-1} - t}{2^{n-1}}\beta_j\Big) \sum_{i \in B} |i\rangle_n$$

As done previously, if we define

$$\boldsymbol{G}_n(|\psi_j(\alpha_j, \beta_j)\rangle_n) = |\psi_{j+1}(\alpha_{j+1}, \beta_{j+1})\rangle_n ,$$

we can induce a recursive formula for the general state of the system, where

$$\alpha_1 = \beta_1 = \frac{1}{\sqrt{2^n}}$$

and

$$\begin{aligned} \alpha_{j+1} &= \frac{2^{n-1} - t}{2^{n-1}}\alpha_j + \frac{2^n - t}{2^{n-1}}\beta_j \\ \beta_{j+1} &= -\frac{t}{2^{n-1}}\alpha_j + \frac{2^{n-1} - t}{2^{n-1}}\beta_j \end{aligned}$$

If we define θ such that

$$\sin^2 \theta = \frac{t}{2^n},$$

then we can arrive at the following closed-up formula, just by using induction and some trigonometric identities:

$$\alpha_j = \frac{1}{\sqrt{t}} \sin((2j-1)\theta)$$

$$\beta_j = \frac{1}{\sqrt{2^n - t}} \cos((2j-1)\theta)$$

.

Theorem 4.6.2. *Let t be the unknown number of solutions, and θ be as previously defined. Let m be an arbitrary positive integer and j be an integer chosen at random following the discrete uniform distribution $\mathcal{U}\{1, m\}$. Then, if $j-1$ corresponds to the number of times we have applied $\boldsymbol{G}$ to the state $|\gamma\rangle_n$ and we observe the state, the probability of obtaining one of the t solutions in A is*

$$P_m = \frac{1}{2} - \frac{\sin(4m\theta)}{4m\sin(2\theta)}.$$

Proof. After $j-1$ iterations of $\boldsymbol{G}$, the probability of obtaining one of the possible solutions is

$$t\alpha_j^2 = \sin^2((2j-1)\theta).$$

If $1 \leq j \leq m$ is chosen randomly, then the average probability of success is given by

$$\begin{aligned} P_m &= \sum_{j=1}^{m} \frac{1}{m} \sin^2((2j-1)\alpha) \\ &= \frac{1}{2m} \sum_{j=1}^{m} \Big(1 - \cos((2j-1)\theta)\Big) \\ &= \frac{1}{2} - \frac{\sin(4m\theta)}{4m\sin(2\theta)}. \end{aligned}$$

In the last step, we have used the following trigonometric identity:

$$\sum_{j=1}^{m} \cos((2j-1)\alpha) = \frac{\sin(2m\alpha)}{2\sin\alpha}$$

□

Thus, Grover's Search Algorithm for multiple solutions follows the next scheme:

```
$m \leftarrow 1$
$\lambda \leftarrow \lambda \in (1, \frac{4}{3})$
$j \leftarrow \mathcal{U}\{1, m\}$
$|\psi\rangle \leftarrow (\boldsymbol{G}^{j-1})(|\gamma\rangle_n)$
Apply the $\boldsymbol{G}$ quantum gate $j-1$ times to $|\gamma\rangle_n$
$i \leftarrow$ Value of the first register
**if** $i \in A$ **then**
    return $i$
**else**
    $m \leftarrow \min(\lambda m, \sqrt{2^n})$
    go to 3
**end if**
```

The average number of iterations of Grover's algorithm is given via the following result:

Theorem 4.6.3. *Let $t \leq 3 \times 2^{n-2}$, the expected time for finding a solution with the previous algorithm has order $\mathcal{O}(\sqrt{2^n/t})$.*

Proof. [28] Let

$$m_0 = \frac{1}{\sin(2\theta)} = \frac{2^{n-1}}{\sqrt{(2^n - t)t}} < \sqrt{\frac{2^n}{t}},$$

then the expected total number of Grover iterations that we need to reach the critical point is at most

$$\frac{1}{2}\sum_{s=1}^{\lceil \log_\lambda m_0 \rceil} \lambda^{s-1} < \frac{1}{2}\frac{\lambda}{\lambda - 1} m_0 = 3m_0$$

and the expected number of Grover iterations needed to succeed once the critical point has been reached is

$$\frac{1}{2}\sum_{u=0}^{\infty} \frac{3^u}{4^{u+1}} \lambda^{u+\lceil \log_\lambda m_0 \rceil} < \frac{\lambda}{8 - 6\lambda} m_0 = \frac{3}{2} m_0.$$

Thus, the expected number of Grover iterations is upper bounded by

$$\frac{9}{2} m_0 \sim \mathcal{O}\left(\sqrt{\frac{2^n}{t}}\right)$$

□

Grover's algorithm, which is also known as quantum amplitude amplification, is specially useful in problems where the best known classical solution is to iterate over all possible candidates in the worst case. Although the acceleration is not exponential, but quadratic, we do know that it is a strict improvement with respect to the classical approach, provided that we are able to identify a solution in polynomial time. The superiority of Shor's algorithm, on the other hand, relies on the unproven conjecture that factoring is not in P. An adiabatic version of Grover's algorithm can be found in [50] and [100].

4.7 Quantum Counting

4.7.1 Definition

Product of the work of Gilles Brassard, Peter Høyer and Alain Tapp, the Quantum Counting algorithm is a variation of Grover's search in which we use the quantum Fourier transform for counting the solutions to the database search problem, in case that this number is unknown to us. It was first sketched in [28] and thoroughly described in [29]. Additionaly, some details of the proof of its correctness that we present here are taken from [44].

The counting problem can be seen as the following: let us suppose that we have an unstructured database, whose elements are indexed by $\{0, 1, \ldots, 2^n - 1\}$, and that we want to know how many elements fulfill a certain property. Note that we are interested in the number of solutions, not in returning any of them. If $A \subseteq \{0, \ldots, 2^n - 1\}$ is the set of indices that fulfill our query, the quantum counting problem can be seen as the problem of calculating $t = |A|$. We shall also define $B = \{0, \ldots, 2^n - 1\} \setminus A$ as the set of indices that do not fulfill the property. Therefore, $|B| = 2^n - t$.

$\mathbb{SETUP}$

$$\boxed{\left|\psi_0\right\rangle_{p,n} \leftarrow \left|0\right\rangle_p \otimes \left|0\right\rangle_n}$$

The quantum counting algorithm starts with $p + n$ qubits initialized at 0, where p depends on n as will be shown later.

$\mathbb{STEP}$ **1**

$$\boxed{\left|\psi_1\right\rangle_{p,n} \leftarrow \boldsymbol{H}^{\otimes p+n}\left(\left|\psi_0\right\rangle_{p,n}\right)}$$

As usual, the first step of the algorithm involves the Hadamard gate, which gives us the state

$$\begin{aligned}\left|\psi_1\right\rangle_{p,n} &= \boldsymbol{H}^{\otimes p+n}\left(\left|\psi_0\right\rangle_{p,n}\right)\\ &= \left|\gamma\right\rangle_p \otimes \left|\gamma\right\rangle_n,\end{aligned}$$

where we recall that

$$\left|\gamma\right\rangle_n = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \left|k\right\rangle_n.$$

If we use the notations

$$\left|a\right\rangle_n = \frac{1}{\sqrt{t}} \sum_{k \in A} \left|k\right\rangle_n,$$

$$|b\rangle_n = \frac{1}{\sqrt{2^n - t}} \sum_{k \in B} |k\rangle_n \,,$$

$$|\mu^+\rangle_n = \frac{1}{\sqrt{2}} \left(|b\rangle_n - i\,|a\rangle_n\right)$$

and

$$|\mu^-\rangle_n = \frac{1}{\sqrt{2}} \left(|b\rangle_n + i\,|a\rangle_n\right)$$

and define ω such that $\sin^2(\pi\omega) = t/2^n$, we can express $|\gamma\rangle_n$ as:

$$\begin{aligned}
|\gamma\rangle_n &= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle_n \\
&= \frac{1}{\sqrt{2^n}} \left(\sum_{k \in A} |k\rangle_n + \sum_{k \in B} |k\rangle_n \right) \\
&= \sin(\pi\omega) \left(\frac{1}{\sqrt{t}} \sum_{k \in A} |k\rangle_n \right) + \cos(\pi\omega) \left(\frac{1}{\sqrt{2^n - t}} \sum_{k \in B} |k\rangle_n \right) \\
&= \sin(\pi\omega)\,|a\rangle_n + \cos(\pi\omega)\,|b\rangle_n \\
&= \frac{i\sin(\pi\omega)}{\sqrt{2}} \left(|\mu^+\rangle_n - |\mu^-\rangle_n\right) + \frac{\cos(\pi\omega)}{\sqrt{2}} \left(|\mu^+\rangle_n + |\mu^-\rangle_n\right) \\
&= \frac{e^{i\pi\omega}}{\sqrt{2}} |\mu^+\rangle_n + \frac{e^{-i\pi\omega}}{\sqrt{2}} |\mu^-\rangle_n
\end{aligned}$$

STEP 2

$$\boxed{|\psi_2\rangle_{p,n} \leftarrow \boldsymbol{C}_{p,n}(|\psi_1\rangle_{p,n})}$$

For the next step, we need a new unitary transformation called the counting gate, which is represented by $\boldsymbol{C}_{p,n}$. This quantum gate has the following effect on a quantum state of the form $|m\rangle_p \otimes |\psi\rangle_n$, where $|m\rangle_p$ is a basis state on p qubits, $|\psi\rangle_n$ is any quantum state on n qubits and $\boldsymbol{G}_n$ is the Grover gate described in the previous algorithm.

$$\boldsymbol{C}_{p,n} : |m\rangle_p \otimes |\psi\rangle_n \to |m\rangle_p \otimes (\boldsymbol{G}_n)^m\,|\psi\rangle_n$$

Let us see what happens when we apply this new quantum gate to our previous quantum system.

$$\begin{aligned}
|\psi_2\rangle_{p,n} &= \boldsymbol{C}_{p,n}(|\psi_1\rangle_{p,n}) \\
&= \boldsymbol{C}_{p,n}\Big(|\gamma\rangle_p \otimes |\gamma\rangle_n\Big) \\
&= \frac{1}{\sqrt{2^p}}\sum_{j=0}^{2^p-1}\Big[|j\rangle_p \otimes \boldsymbol{G}_n^j\left(|\gamma\rangle_n\right)\Big] \\
&= \frac{1}{\sqrt{2^p}}\sum_{j=0}^{2^p-1}\left[|j\rangle_p \otimes \boldsymbol{G}_n^j\left(\frac{e^{i\pi\omega}}{\sqrt{2}}|\mu^+\rangle_n + \frac{e^{-i\pi\omega}}{\sqrt{2}}|\mu^-\rangle_n\right)\right] \\
&= \frac{1}{\sqrt{2^{p+1}}}\sum_{j=0}^{2^p-1}\left[|j\rangle_p \otimes \Big(e^{i\pi\omega}\boldsymbol{G}_n^j\left(|\mu^+\rangle_n\right) + e^{-i\pi\omega}\boldsymbol{G}_n^j\left(|\mu^-\rangle_n\right)\Big)\right] \\
&= \frac{1}{\sqrt{2^{p+1}}}\sum_{j=0}^{2^p-1}\left[|j\rangle_p \otimes \Big(e^{i\pi\omega(2j+1)}|\mu^+\rangle_n + e^{-i\pi\omega(2j+1)}|\mu^-\rangle_n\Big)\right] \\
&= \frac{e^{i\pi\omega}}{\sqrt{2^{p+1}}}\sum_{j=0}^{2^p-1} e^{2\pi i\omega j}|j\rangle_p \otimes |\mu^+\rangle_n + \frac{e^{-i\pi\omega}}{\sqrt{2^{p+1}}}\sum_{j=0}^{2^p-1} e^{2\pi i(1-\omega)j}|j\rangle_p \otimes |\mu^-\rangle_n
\end{aligned}$$

In order to understand what happened in the last identities of the equation, we should see the effect that $\boldsymbol{G}_n^j$ has on the states $|\mu^+\rangle_n$ and $|\mu^-\rangle_n$. First:

$$\begin{aligned}
\boldsymbol{G}_n\left(|a\rangle_n\right) &= (2|\gamma\rangle\langle\gamma|_n) - \boldsymbol{I})\big(-|a\rangle_n\big) \\
&= -2|\gamma\rangle\langle\gamma|a\rangle_n + |a\rangle_n \\
&= -2\sin(\pi\omega)|\gamma\rangle_n + |a\rangle_n \\
&= -2\sin(\pi\omega)\big(\cos(\pi\omega)|b\rangle_n + \sin(\pi\omega)|a\rangle_n\big) + |a\rangle_n \\
&= -2\sin(\pi\omega)\cos(\pi\omega)|b\rangle_n + (1-2\sin^2(\pi\omega))|a\rangle_n \\
&= -\sin(2\pi\omega)|b\rangle_n + \cos(2\pi\omega)|a\rangle_n
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{G}_n\left(|b\rangle_n\right) &= (2|\gamma\rangle\langle\gamma|_n) - \boldsymbol{I})\big(|b\rangle_n\big) \\
&= 2|\gamma\rangle\langle\gamma|b\rangle_n - |b\rangle_n \\
&= 2\cos(\pi\omega)|\gamma\rangle_n - |b\rangle_n \\
&= 2\cos(\pi\omega)\big(\cos(\pi\omega)|b\rangle_n + \sin(\pi\omega)|a\rangle_n\big) - |b\rangle_n \\
&= (2\cos^2(\pi\omega)-1)|b\rangle_n + 2\sin(\pi\omega)\cos(\pi\omega)|a\rangle_n \\
&= \cos(2\pi\omega)|b\rangle_n + \sin(2\pi\omega)|a\rangle_n
\end{aligned}$$

Which leads us to

$$\begin{aligned}
\boldsymbol{G}_n\left(|\mu^+\rangle_n\right) &= \frac{1}{\sqrt{2}}\Big(\boldsymbol{G}_n(|b\rangle_n) - i\boldsymbol{G}_n(|a\rangle_n)\Big)\\
&= \frac{1}{\sqrt{2}}\Big(\cos(2\pi\omega)\,|b\rangle_n + \sin(2\pi\omega)\,|a\rangle_n + i\sin(2\pi\omega)\,|b\rangle_n - i\cos(2\pi\omega)\,|a\rangle_n\Big)\\
&= \frac{e^{2i\pi\omega}}{\sqrt{2}}(|b\rangle_n - i\,|a\rangle_n)\\
&= e^{2i\pi\omega}\,|\mu^+\rangle_n
\end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{G}_n\left(|\mu^-\rangle_n\right) &= \frac{1}{\sqrt{2}}\Big(\boldsymbol{G}_n(|b\rangle_n) + i\boldsymbol{G}_n(|a\rangle_n)\Big)\\
&= \frac{1}{\sqrt{2}}\Big(\cos(2\pi\omega)\,|b\rangle_n + \sin(2\pi\omega)\,|a\rangle_n - i\sin(2\pi\omega)\,|b\rangle_n + i\cos(2\pi\omega)\,|a\rangle_n\Big)\\
&= \frac{e^{2i\pi\omega}}{\sqrt{2}}(|b\rangle_n + i\,|a\rangle_n)\\
&= e^{-2i\pi\omega}\,|\mu^-\rangle_n
\end{aligned}$$

STEP **3**

$$\boxed{|\psi_3\rangle_{p,n} \leftarrow (\boldsymbol{F}_p^{-1} \otimes \boldsymbol{I}^{\otimes n})(|\psi_2\rangle_{p,n})}$$

The next step involves the inverse quantum Fourier transform $\boldsymbol{F}_n^{-1}$ already explained in Section 4.5, which has the following effect on an n-qubit basis state.

$$\boldsymbol{F}_n^{-1} : |j\rangle_n \to \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{-2\pi i\frac{j}{2^n}k}\,|k\rangle_n$$

We recall that, applied to a certain type of quantum state, the inverse quantum Fourier transform can give us certain information about the distribution of its amplitudes.

$$\boldsymbol{F}_n^{-1}\left(\frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1} e^{2\pi i\frac{k}{2^n}j}\,|j\rangle_n\right) = \boldsymbol{F}_n^{-1}\Big(\boldsymbol{F}_n\,|k\rangle_n\Big) = |k\rangle_n$$

Let us see what happens when we apply it to our current quantum state:

$$\begin{aligned}
|\psi_3\rangle_{p,n} &= (\boldsymbol{F}_p^{-1} \otimes \boldsymbol{I}^{\otimes n})(|\psi_3\rangle_{p,n}) \\
&= (\boldsymbol{F}_p^{-1} \otimes \boldsymbol{I}^{\otimes n}) \\
&= \frac{e^{i\pi\omega}}{\sqrt{2^{p+1}}} \sum_{j=0}^{2^p-1} e^{2\pi i\omega j} \boldsymbol{F}_p^{-1} |j\rangle_p \otimes |\mu^+\rangle_n + \frac{e^{-i\pi\omega}}{\sqrt{2^{p+1}}} \sum_{j=0}^{2^p-1} e^{2\pi i(1-\omega)j} \boldsymbol{F}_p^{-1} |j\rangle_p \otimes |\mu^+\rangle_n \\
&= \frac{e^{i\pi\omega}}{\sqrt{2^{p+1}}} \sum_{l=0}^{2^p-1} \sum_{j=0}^{2^p-1} e^{2\pi i(\omega - \frac{l}{2^p})j} |l\rangle_p \otimes |\mu^+\rangle_n + \\
&\quad \frac{-e^{i\pi\omega}}{\sqrt{2^{p+1}}} \sum_{l=0}^{2^p-1} \sum_{j=0}^{2^p-1} e^{2\pi i((1-\omega) - \frac{l}{2^p})j} |l\rangle_p \otimes |\mu^+\rangle_n
\end{aligned}$$

STEP **4**

```
$\tilde{l} \leftarrow$ measure the first register of $|\psi_3\rangle_{p,n}$
**if** $\tilde{l} > 2^{p-1}$ **then**
    $\tilde{l} \leftarrow 2^p - \tilde{l}$
**end if**
$\tilde{t} \leftarrow 2^n \sin^2\left(\frac{\pi\tilde{l}}{2^p}\right)$
return $\tilde{t}$
```

As can be concluded from the previous step, we could have obtained the value $2^p\omega$ or $2^p(1-\omega)$ if ω were an integer. However, as that is not the case, we end up with $\tilde{l} = 2^p\tilde{\omega}$ or $\tilde{l} = 2^p(1-\tilde{\omega})$, where $\tilde{\omega}$ is an estimator of ω, and from which we can get an estimator $\tilde{t}$ for t. Thus ends the quantum counting algorithm, remains to be proven how good is that estimator, and the probability of obtaining it.

4.7.2 Proof of correctness

Theorem 4.7.1. *Let $f : \{0, 1, \ldots, 2^n - 1\} \to \{0, 1\}$ be the indicator function of a certain set $A \subseteq \{0, 1, \ldots, 2^n - 1\}$ with $t = |A| = |f^{-1}(1)|$, let $p \geq 2$ and let $\tilde{t}$ be the return value of the previously described algorithm. Then, with probability of at least $8/\pi^2$, it follows that*

$$|t - \tilde{t}| \leq \frac{2\pi}{2^p}\sqrt{t(2^n - t)} + \frac{\pi^2}{2^{2p}}|2^n - 2t|.$$

Proof. Let us suppose that we have a quantum state of the form

$$\frac{1}{2^p} \sum_{l=0}^{2^p-1} \sum_{k=0}^{2^p-1} e^{2\pi i(\omega - \frac{l}{2^p})k} |l\rangle_p .$$

As the amplitudes of such a state can be expressed as

$$\alpha_l = \frac{1}{2^p}\sum_{k=0}^{2^p-1} e^{2\pi i(\omega - \frac{l}{2^p})k} = \frac{1 - e^{2\pi i 2^p(\omega-\frac{l}{2^p})}}{2^p(1-e^{2\pi i(\omega-\frac{l}{2^p})})},$$

its probability distribution can be seen as

$$|\alpha_l|^2 = \frac{\sin^2(2^p\pi(\omega - \frac{l}{2^p}))}{2^{2p}\sin^2(\pi(\omega-\frac{l}{2^p}))}.$$

As $l_1 = \lfloor 2^p\omega \rfloor$ and $l_2 = \lceil 2^p\omega \rceil$ are the closest values of l to $2^p\omega$, we can deduce that

$$\begin{aligned}
P\left(\left|\frac{\tilde{l}}{2^p} - \omega\right| \le \frac{1}{2^p}\right) &= P(|\tilde{l} - 2^p\omega| \le 1) \\
&= P(\tilde{l} = l_1) + P(\tilde{l} = l_2) \\
&= |\alpha_{l_1}|^2 + |\alpha_{l_2}|^2 \\
&= \frac{\sin^2(2^p\pi(\omega - \frac{l_1}{2^p}))}{2^{2p}\sin^2(\pi(\omega - \frac{l_1}{2^p}))} + \frac{\sin^2(2^p\pi(\omega - \frac{l_2}{2^p}))}{2^{2p}\sin^2(\pi(\omega - \frac{l_2}{2^p}))} \\
&= \frac{\sin^2(2^p\pi\Delta)}{2^{2p}\sin^2(\pi\Delta)} + \frac{\sin^2(2^p\pi(\frac{1}{2^p} - \Delta))}{2^{2p}\sin^2(\pi(\frac{1}{2^p} - \Delta))} \\
&\ge \frac{1}{2^{2p}}\left(\frac{1}{\sin^2(\frac{\pi}{2^{p+1}})} + \frac{1}{\sin^2(\frac{\pi}{2^{p+1}})}\right) \\
&= \frac{2}{2^{2p}\sin^2(\frac{\pi}{2^{p+1}})} \\
&> \frac{2}{2^{2p}(\frac{\pi}{2^{p+1}})^2} \\
&= \frac{8}{\pi^2}
\end{aligned}$$

Where $\Delta = (2^p\omega - l_1)/2^p$ and whose minimum is attained at $\Delta = 1/2^{p+1}$. Thus, we have at least a probability of $8/\pi^2$ of obtaining l_1 or l_2 with an error less than $1/2^p$.

It follows that, with $\Lambda = |\tilde{l}/2^p - \omega| \le 1/2^p$ and with probability of at least $8/\pi^2$,

$$\begin{aligned}
|t - \tilde{t}| &= 2^n|\sin^2(\pi\omega) - \sin^2(\pi\frac{\tilde{l}}{2^p})| \\
&= 2^n|\sin^2(\pi\omega) - \sin^2(\pi\omega \pm \pi\Lambda)| \\
&= 2^n|\sin^2(\pi\omega) - (\sin(\pi\omega)\cos(\pi\Lambda) \pm \sin(\pi\Lambda)\cos(\pi\omega))^2| \\
&\le 2^n|\sin(2\pi\Lambda)\sin(\pi\omega)\cos(\pi\omega)| + 2^n\sin^2(\pi\Lambda)|1 - \sin^2(\pi\omega)| \\
&\le 2^{n+1}\pi\Lambda\sqrt{\frac{t}{2^n}(1 - \frac{t}{2^n})} + 2^n(\pi\Lambda)^2|1 - \frac{t}{2^{n-1}}| \\
&\le \frac{\pi}{2^{p-1}}\left(\sqrt{t(2^n - t)} + \frac{\pi}{2^p}|2^{n-1} - t|\right)
\end{aligned}$$

□

5

Quantum Algorithms for the Combinatorial Invariants of Numerical Semigroups

"Do you guys just put the word quantum *in front of everything?"*
– *Ant-Man and the Wasp*

"There were stone cups along the rim of the pool. Arya filled one and brought it to him, so he could drink. The young man stared at her for a long moment when she offered it to him. 'Valar morghulis,' he said. 'Valar dohaeris,' she replied."
– George R. R. Martin, *A Song of Ice and Fire: A Feast for Crows*

5.1 Introduction

What we present here in this chapter are the principal contributions of this doctoral thesis. As explained in the Introduction, our main scope was to study a certain group of computationally hard problems, and more concretely those connected with the theory of numerical semigroups, from the perspective of quantum computation. The algorithms we have thus developed are capable of obtaining the solution for the Sylvester denumerant, the numerical semigroup membership, the Apéry set and the Frobenius number of any numerical semigroup with the help of a hypothetical (or not so hypothetical, as will be seen later) quantum computer.

The first two algorithms lean on the quantum circuit model and are based respectively on Grover's database search and on quantum counting, which were previously explained in detail in Chapter 4. Both of them rely on a generation of all elements of the numerical semigroup up to a certain bound inside the quantum computer thanks to quantum parallelism. This way, we can ask a certain type of questions via an oracle to the distribution of the elements of the numerical semigroup, and then infer the solution to these combinatorial problems.

The last two algorithms follow a completely different approach, as both are based in the quantum adiabatic model, and are specifically adapted so that they would run in the quantum computing hardware developed by the Canadian company D-Wave Systems. What we propose is to solve these two problems from the point of view of mathematical optimization, while at the same time deconstructing the structure of the problem in order to embed it inside the currently available hardware.

Along with a description of the algorithms, we present some tables of numerical results and hypothetical performance over the classical version, simulated with a `C++` library specially developed for this Ph.D. thesis. This library is called `numsem` and can be found at a public GitHub repository [92] along with all the documentation needed for the correct replication of the results here presented.

5.2 The `numsem` library

In this section we give a brief description of the `numsem` library [92], specifically implemented for numerical semigroups and which helps us to classically compute the combinatorial invariants described in this doctoral thesis and also replicate the numerical analyses presented in this chapter. The `numsem` library has been programmed in `C++` and has been tested in Ubuntu 14.04.2. In order to install it, we have just to clone the repository by using the `git` command (provided that we have previously installed `git` inside our computer):

```
git clone https://github.com/jqnoc/numsem.git
```

After we have downloaded the contents from the repository, we have to type the following installation commands:

```
make
cd bin
numsem
```

Thus, we end up in the directory where the actual executable file is. Let us suppose that we want to test the numerical semigroup $S = \langle 7, 11, 17, 23 \rangle$ or obtain its combinatorial invariants. We have first to create a file with the generators (we can also just put random integers inside, and let `numsem` decide whether they generate a numerical semigroup or not). In this case, the contents of the file should be like this (the order does not matter):

```
7
11
17
23
```

Then, if the file with the generators is called `generators.txt`, we just have to write the following command:

```
numsem generators.txt
```

and obtain the output:

```
===================================================================
 NumSem v0.1.0
 Copyright (C) Joaquin Ossorio-Castillo. All rights reserved.
 This software is free for non-commercial purposes.
 Full license information can be found in the LICENSE.txt file.
 https://github.com/jqnoc/numsem/
===================================================================
Numerical semigroup: S = <7, 11, 17, 23>
Total number of generators: 4
```

The correct use of the library is triggered with the option `--help`, and also when the options are not written correctly. To date, the available options are:

```
Use:
   numsem *file* [options]
   *file*: the file with the generators of the numerical semigroup S

Available options:
   --help: print the help
   -f: calculate Frobenius number of S
   -ga: calculate set of gaps of S
   -gn: calculate genus of S
   -ap *s*: calculate the Apery set of s with respect to S (only
      with AMPL)
   -d *t*: calculate Sylvester denumerant for t and S
   -ds *t*: calculate Sylvester denumerant for t and S and print
      all solutions
   -dg *b*: calculate Sylvester function from 0 to b
   -m *t*: calculate if t is in S
   -ampl: use AMPL and Gurobi for Frobenius, genus or Apery set
   -json: write results to a .json file
```

We now proceed to explain some of them, the rest will be described throughout the rest of the chapter. The calculation of the Frobenius number, for example, has been implemented in two ways. The default one is quite straightforward and hardly efficient, as it just simply finds the maximum of the set of gaps. The second one is only applied if the option `-ampl` is also activated, and will use one of the algorithms proposed in Section 5.4. This second option is more efficient, as will be explained later. Same goes for the Apéry set and the genus of S, they have a default method and an alternative method with the option `-ampl`.

Depending on the options used, the program returns a `.json` file with all the computed information. This file has the following format:

```
{
  "generators": [7, 11, 17, 23],
  "frobenius": 27,
  "genus": 17,
  "multiplicity": 7,
  "embedding_dimension": 4,
  "gaps": [1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 13, 15, 16, 19, 20, 26,
     27]
}
```

5.3 Sylvester Denumerant and Numerical Semigroup Membership

Let $S = \langle a_1, ..., a_n \rangle$ be a numerical semigroup. As explained in Chapter 2, the Sylvester denumerant problem resides in determining if an integer $t \in \mathbb{Z}_{\geq 0}$ is in S and, if the answer is positive, calculate the number of solutions of the linear Diophantine equation

$$\sum_{i=1}^{n} \lambda_i a_i = t,$$

where $\lambda_i \in \mathbb{Z}_{\geq 0}$. As previously stated, we will assume that $i < j$ implies $a_i < a_j$ and that $t \geq a_n$ without loss of generality. We recall that this problem is in NP-hard, and that we do not know a polynomial time algorithm for deciding if a candidate solution is in fact a solution. Therefore, the Sylvester denumerant problem is not known to be in NP, and thus finding a polynomial time algorithm (quantum or not) that solves it would be surprising.

In this subsection we present an algorithm for the calculation of the Sylvester denumerant in the general case. It is a variation of the well-known brute-force approach, with the difference that it gets help from a hypothetical quantum computer. The main idea behind the algorithm is to generate all possible elements of the numerical semigroup up to a certain set of bounds, with the bounds depending on each generator respectively. This generation, which would require an exponential number of calculations in the classical version of the algorithm, can be done in one step inside a quantum computer thanks to the Hadamard gate and quantum paralellism.

First, we shall give an upper bound to all possible solutions for the λ_i variables. In order to achieve that, we define

$$b_i = 1 + \left\lfloor \log_2 \left(\frac{t}{a_i} \right) \right\rfloor,$$

which tells us the number of binary digits needed for representing all possible values of λ_i respectively. Thus, we can calculate

$$b = \sum_{i=1}^{n} b_i$$

as the total number of qubits needed. As

$$\begin{aligned} b &= \sum_{i=1}^{n} b_i \\ &= \sum_{i=1}^{n} \left(1 + \left\lfloor \log_2 \left(\frac{t}{a_i}\right)\right\rfloor\right) \\ &= n + \sum_{i=1}^{n} \left(\left\lfloor \log_2 \left(\frac{t}{a_i}\right)\right\rfloor\right) \\ &\leq n + \sum_{i=1}^{n} \Big(\log_2(t) - \log_2(a_i)\Big) \\ &= n\big(1 + \log_2(t)\big) - \sum_{i=1}^{n} \log_2(a_i), \end{aligned}$$

we can deduce that, in this algorithm, the size of our quantum register grows exponentially with respect to the size of the number of generators of the numerical semigroup, and polynomially with respect of the size of t. Thus, fixing the numerical semigroup and just increasing the value of t is not an issue in terms of the computational memory needed, what is really challenging in terms of qubit requirements is adding generators to a numerical semigroup and trying to find the Sylvester denumerant for a fixed t. This is not surprising, as the simple task of storing a solution for the aforementioned equation will always take an exponential memory in terms of n (an algorithm that calculates the Sylvester denumerant without finding the actual solutions may exist, though, but it is not this case).

$\mathbb{SETUP}$

$$\boxed{\left|\psi_0\right\rangle_{p,b} \leftarrow \left|0\right\rangle_p \otimes \left|0\right\rangle_b}$$

Our system will have the same setup as in the quantum counting algorithm, where b is the size previously calculated and where p depends on b in such a way that the desired probability of obtaining the correct result is big enough.

$\mathbb{STEP}$ **1**

$$\boxed{\left|\psi_1\right\rangle_{p,b} \leftarrow \boldsymbol{H}^{\otimes p+n}(\left|\psi_0\right\rangle_{p,b})}$$

$$\left|\psi_1\right\rangle_{p,b} = \left|\gamma\right\rangle_p \otimes \left|\gamma\right\rangle_b$$

As has been already said, thanks to the Hadamard transformation we can obtain all computational basis states of our quantum system inside our register. As the dimension b is calculated with respect to the generators of the numerical semigroup and the integer t, these basis states correspond each one uniquely to a combination for all possible values of the variables λ_i. A representation of this feature is shown in Figure 5.1, where $q_1, q_2, \dots, q_{b_1}$ are the b_1 binary digits of λ_i, and so on.

STEP **2**

$$\boxed{\left|\psi_2\right\rangle_{p,b} \leftarrow \boldsymbol{C}_{p,b}(\left|\psi_1\right\rangle_{p,b})}$$

For the next step, we need the counting gate $\boldsymbol{C}_{p,n}$, previously defined as

$$\boldsymbol{C}_{p,n} : \left|m\right\rangle_p \otimes \left|\psi\right\rangle_n \to \left|m\right\rangle_p \otimes (\boldsymbol{G}_n)^m \left|\psi\right\rangle_n ,$$

where $\boldsymbol{G}_n$ is again the Grover gate. This Grover gate depends in turn on the Oracle gate $\boldsymbol{O}_f$ used in the algorithms described in Chapter 4, and yields the following result when applied to a basis state:

$$\boldsymbol{O}_f : \left|j\right\rangle_n \otimes \left|k\right\rangle_m \to \left|j\right\rangle_n \otimes \left|k \oplus f(j)\right\rangle_m .$$

We recall that the oracle gate is problem specific, and depends on a certain function f capable of recognising a solution for our problem. What we are going to do is to define f in such a way that it identifies if a certain combination of the variables λ_i encoded into our quantum register are a solution for the numerical semigroup membership problem. In other words:

$$f(j) = \begin{cases} 1 & \text{if } \displaystyle\sum_{i=1}^{n} \lambda_i a_i = t \\ 0 & \text{otherwise} \end{cases}$$

The rest of the algorithm continues as previously explained, but what interests us now is the size of the first register, p, which depends on b but also on the probability of obtaining the correct solution for the Sylvester denumerant. Thanks to [28] and [29], we know that in order to have a reasonable probability of succeeding, $p \sim \sqrt{2^b}$. Thus, the computational order of the algorithm remains the same in both memory size and time.

Concerning the performance of the algorithm, let us compare the iterations required for the brute force approach to calculate the Sylvester denumerant for a fixed semigroup and an increasing value of t, against the hypothetical performance of the quantum variant. Let $S = \langle 376, 381, 393, 399\rangle$ be a numerical semigroup and let $t = 10000$, we can calculate $d(10000; 376, 381, 393, 399)$ with the

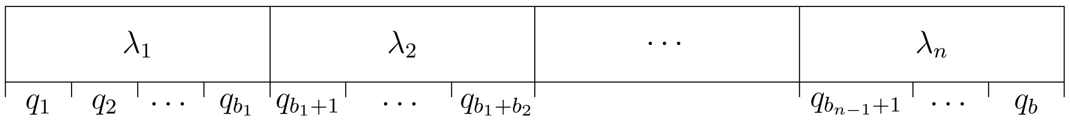


Figure 5.1: Representation of the values of λ_i with respect to the b qubits

```
d(10000; 376, 381, 393, 399) = 9

10*(376) + 4*(381) + 12*(393) + 0*(399)
4*(376) + 13*(381) + 8*(393) + 1*(399)
10*(376) + 5*(381) + 9*(393) + 2*(399)
4*(376) + 14*(381) + 5*(393) + 3*(399)
10*(376) + 6*(381) + 6*(393) + 4*(399)
4*(376) + 15*(381) + 2*(393) + 5*(399)
10*(376) + 7*(381) + 3*(393) + 6*(399)
10*(376) + 8*(381) + 0*(393) + 8*(399)
16*(376) + 0*(381) + 1*(393) + 9*(399)
```

Figure 5.2: `numsem` output for $d(10000; 376, 381, 393, 399)$

`numsem` library, which will give us the output seen in Figure 5.2. Thus, we obtain $d(10000; 376, 381, 393, 399) = 9$ and the values of λ_i for all nine solutions.

What we are going to do know is to calculate $d(t; 376, 381, 393, 399)$ for all values of t up to a certain bound, in order to test the time performance of the brute-force approach and compare it to a hypothetical performance of the quantum approach. First, we can see in Figures 5.3, 5.4 and 5.5 the Sylvester denumerant function for the numerical semigroup $S = \langle 376, 381, 393, 399\rangle$ in the interval (1,20000), computed with `numsem` and plotted using Jupyter Notebooks [75]. This function has been proven to be a quasi-polynomial function in the variable t of degree $n - 1$ in the general case; however, computing all the coefficients of the polynomial is a computationally hard problem (see [98, 16, 17, 13]).

Additionally, in Figure 5.6 we show the operations needed (i.e., the number of possible solutions we have to check) for the computation of $d(t; 376, 381, 393, 399)$ with `numsem`, along with the theoretical iterations needed for the quantum version which has been calculated taking into account the asymptotical behavior of the quantum counting algorithm. Please note that both are exponential with respect to t, but in the quantum algorithm the exponent would be divided by 2 (as was proven in Chapter 4). For example, for $t = 10000$ we would need more than 492,000 classical iterations, and just 702 quantum iterations. However, in the classical case we would obtain the actual solutions for the linear Diophantine equation of the numerical semigroup, while in the quantum case we would *just* obtain the value of the Sylvester denumerant (or one of the possible solutions, if there are any, as will be seen now).

Regarding the membership problem of a semigroup S, we already know that if $t > f(S)$, then $t \in S$, but the computation of $f(S)$ is extremely hard. Moreover, if $t < f(S)$, the membership problem remains unsolved even if we already know the value of $f(S)$ (although, as proven by Jorge Ramírez-Alfonsín [97], it is possible to solve the NSMP in polynomial time if we already have an oracle for the Frobenius problem).

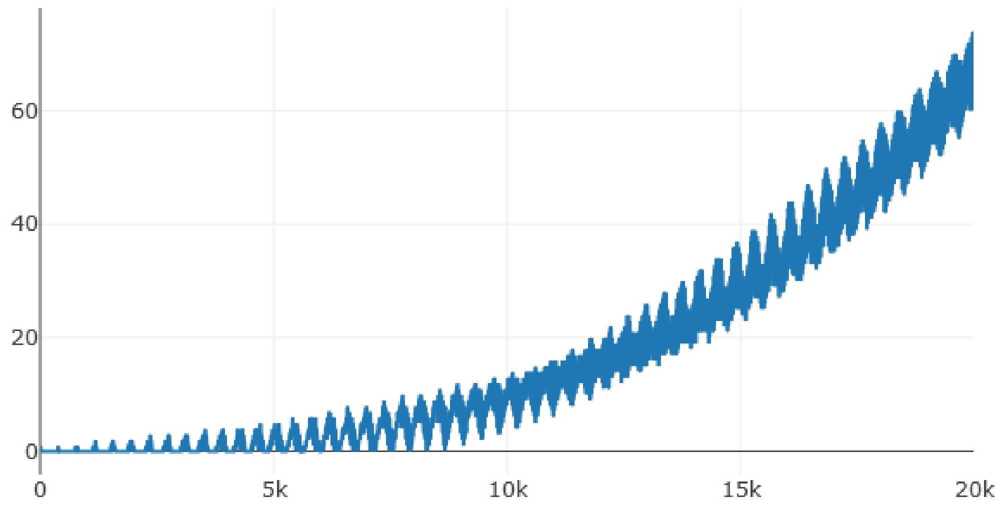


Figure 5.3: Sylvester quasi-polynomial

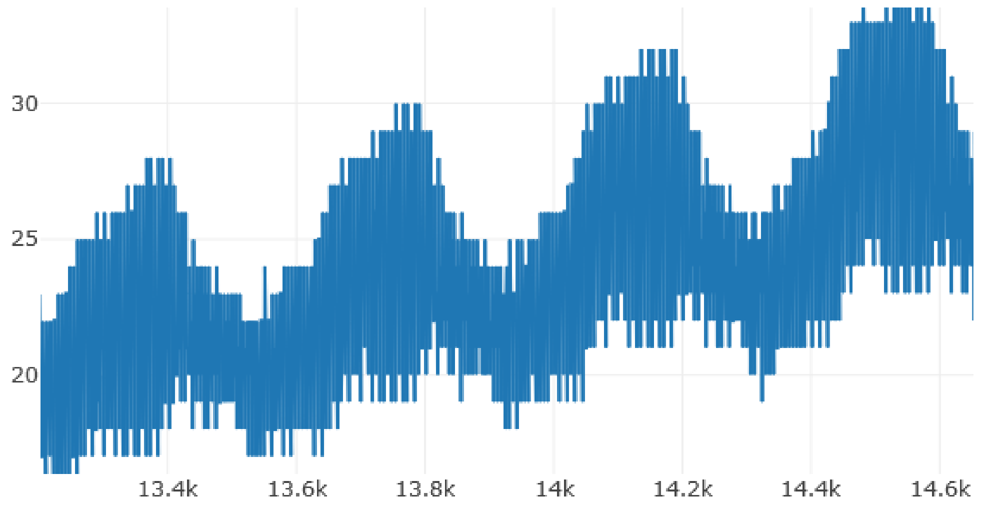


Figure 5.4: Sylvester quasi-polynomial (detail)

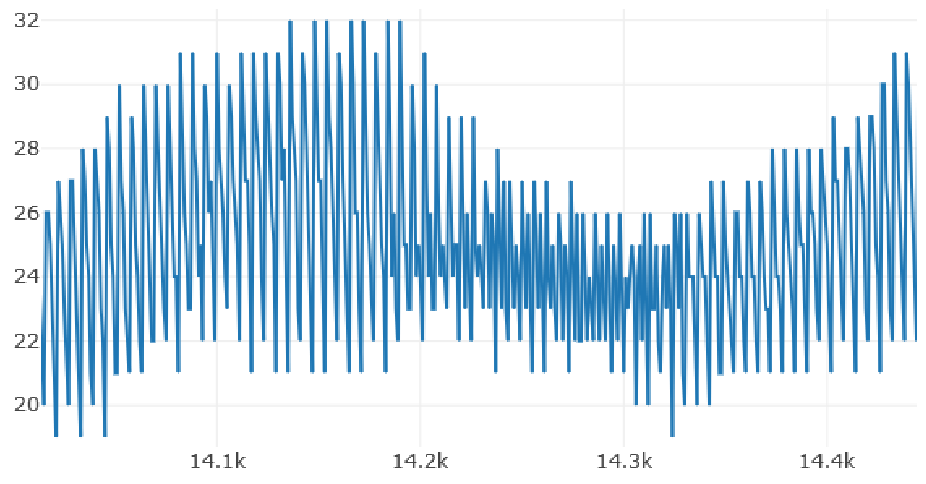


Figure 5.5: Sylvester quasi-polynomial (detail)

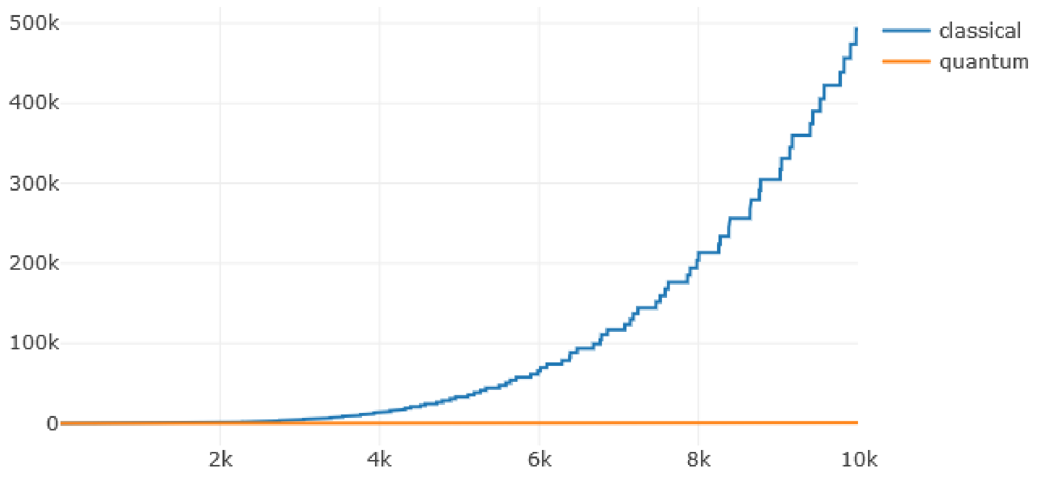


Figure 5.6: Sylvester denumerant iteration comparison

The previous quantum algorithm is capable of solving the numerical semigroup membership problem for a certain $t \in \mathbb{Z}_{\leq 0}$. We have to just calculate the Sylvester Denumerant for t and, if $d(S;t) > 0$ return `YES` and otherwise return `NO`. This is quite straightforward but, how about solving the NSMP constructively, not only answering `YES` or `NO` to the question of whether there exist those $\lambda_i, \ldots, \lambda_n$, but also giving one of its possible combinations if the answer is `YES`. For that purpose, we will make use of Grover's Quantum Search algorithm, already explained in Section 4.6.

Using Grover's Algorithm, we can provide a solution for the NSMP linear Diophantine equation in the same $\sqrt{2^b}$ steps as for the Sylvester denumerant. The actual order of both algorithms with respect to t, the number of generators n of the numerical semigroup, and the generators $a_1, \ldots, a_n$ is:

$$\sqrt{2^{n\left(1+\log_2(t)\right)-\sum_{i=1}^{n}\log_2(a_i)}}$$

We can also go further and obtain all solutions to the Diophantine equation. First, we run the quantum algorithm for the Sylvester denumerant. Then, we apply the quantum algorithm for the NSMP iteratively until we have obtained as much different solutions as the value of the denumerant. This is a well-known problem in probability theory [51] known as the coupon collector's problem; if the denumerant is d, then the expected number of trials $k \geq d$ for obtaining all d solutions grows as

$$\Theta(d\log(d)).$$

For example, if $d = 10$, the expected number of runs of the quantum algorithm for the NSMP before obtaining all different solutions is 33.

5.4 Apéry Set and Frobenius Problem

The next algorithm shows the possibilities of calculating the Apéry set and the Frobenius number of a numerical semigroup with an actual adiabatic quantum computer. As described in Section 3.5, current D-Wave quantum annealers solve a certain kind of mathematical optimization problems known as Ising spin problems, namely

$$H(s_1, \ldots, s_n) = \sum_{i=1}^{n} h_i s_i + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} J_{ij} s_i s_j$$

with $s_i \in \{-1, 1\}$, where the objective is to find the minimum of $H(s_1, \ldots, s_n)$.

On the other hand, we recall that the Apéry set with respect to $s \in S \setminus \{0\}$, where $S = \langle a_1, \ldots, a_n \rangle$ is a numerical semigroup, is defined as

$$Ap(S, s) = \{x \in S \mid x - s \notin S\}.$$

The question is, how could we be able to translate this problem to the Ising spin model solved by the D-Wave machine? We have already proved in Lemma 2.1.11 that $Ap(S, s) = \{\omega_0, \ldots, \omega_{n-1}\}$, where

$$\omega_i = \min\{x \in S : x \equiv i \mod s\}.$$

How about transforming the definition of ω_i into the answer of a mathematical optimization problem?

Definition 5.4.1. [93] *An integer linear program, or* ILP, *is defined in its canonical form as the optimization problem:*

$$\begin{array}{rl} \min & c^T x \\ \textit{subject to:} & Ax = b \end{array}$$

where $x \in \mathbb{Z}^n_{\geq 0}$, $A \in \mathbb{Z}^n \times \mathbb{Z}^m$, $b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^n$.

Thus, it is straightforward to see that the calculation of each of the ω_i can be redefined as the following ILP:

$$\begin{array}{rl} \min & \displaystyle\sum_{j=1}^{n} a_j x_j \\ \textit{subject to:} & \displaystyle\sum_{j=1}^{n} a_j x_j = i + sk, \end{array}$$

with $x_j \in \mathbb{Z}_{\geq 0}$ for all $j = 1, ..., n$ and $k \in \mathbb{Z}_{\geq 0}$.

This mathematical optimization problem represents a way of calculating the Apéry set in its own right and, although integer linear programming is in NP-hard and its recognition version (deciding whether $Ax = b$ has a feasible solution or not regardless of its optimality) is in NP-complete [93], this computational complexity arises in the general case. In our context, it may prove to be an easier problem; however, details on this remain to be worked out. This approach for the calculation of the Apéry set first appeared in [61], although Greenberg's work dealt with the direct calculation of the Frobenius number by means of $Ap(S, a_1)$, as will be discussed later.

We have tested this algorithm using state-of-the-art optimization software for integer optimization. In our case, the optimization problem for ω_i was modeled using AMPL [53, 54], an algebraic modeling language for solving mathematical optimization problems. AMPL has a clear advantage, as our problem is entirely parameterized and AMPL allows us to describe the generic problem in a `.mod` file while defining the actual values for the parameters in a separate `.dat` file. This way, we have just to change the parameters inside the `.dat` file in order to solve a new instance of the Apéry set. The `.mod` file we propose is shown in Figure 5.7 (the names of the parameters and variables of the problem are maintained so that no further explanation is required).

Let us suppose that we are interested in the numerical semigroup $S = \langle 11, 19, 23 \rangle$, and that we want to calculate the Apéry set of $s = 30$ (i.e., $Ap(S, 30)$). The calculation of, for example, $\omega_5 \in Ap(S, 30)$, will have the associated `.dat` file depicted in Figure 5.8. In order to compute ω_5, we also need a `.run` file that will load the model and the data of the problem, and which will also call the solver for solving the problem. In our case, the solver we have chosen is Gurobi [3] which, among other things, solves integer linear problems up to a global optimum. The `.run` file we have used is shown in Figure 5.9, and the corresponding output we obtain can be seen in Figure 5.10. This output tells us that $\omega_5 = 65$ and also that a representation of 65 with respect to the generators of the semigroup is

$$65 = 0 \times (11) + 1 \times (19) + 2 \times (23).$$

We can also write an alternative `.run` file that will directly calculate and display

```
param n;
set N := 1..n;
param a {N};
param s;
param i;
var X {N} integer, >= 0;
var K integer;

minimize T:
   sum {j in N} a[j]*X[j];

subject to C:
   sum {j in N} a[j]*X[j] = i + s*K;
```

Figure 5.7: File `apery_set_member.mod`

```
param n := 3;
param a :=
    1   11
    2   19
    3   23;
```

Figure 5.8: File `numerical_semigroup.dat`

```
model apery_set_member.mod;
data numerical_semigroup.dat;
let s := 30;
let i := 5;
option solver gurobi;
solve;
display X;
display K;
```

Figure 5.9: File `apery_set_member.run`

the whole content of the Apéry set for a certain $s \in S$. First, we have to drop the `param i := 5;` line from the `.run` file, as it will be changed in each iteration of the main loop, and modify the `.run` so that it will look like the one in Figure 5.11. Thus, we obtain $Ap(S, 30) = \{0, 11, 19, 22, 23, 33, 34, 38, 42, 44, 45, 46, 55, 56, 57, 61, 65, 66, 67, 69, 77, 78, 80, 84, 88, 89, 92, 100, 103, 111\}$ (see Figure 5.12 for the actual output).

This algorithm also provides a way for calculating the Frobenius number of a

```
Gurobi 8.0.0: optimal solution; objective 65
3 simplex iterations
1 branch-and-cut nodes
X [*] :=
1  0
2  1
3  2
;

K = 2
```

Figure 5.10: Output for `apery_set_member.run`

```
model apery_set_member.mod;
data numerical_semigroup.dat;
let s := 30;
option solver gurobi;
param apery_set {0..s-1};
for {l in 0..s-1} {
    let i := l;
    solve;
    let apery_set[l] := T;
}
display apery_set;
```

Figure 5.11: File `apery_set.run`

```
apery_set [*] :=
 0   0    4  34    8  38   12  42   16  46   20  80   24  84   28  88
 1  61    5  65    9  69   13 103   17  77   21 111   25  55   29  89
 2  92    6  66   10 100   14  44   18  78   22  22   26  56
 3  33    7  67   11  11   15  45   19  19   23  23   27  57
;
```

Figure 5.12: Output for `apery_set.run`

numerical semigroup. We recall that, for any numerical semigroup S and any integer $s \in S \setminus \{0\}$, then

$$f(S) = \max\{Ap(S, s)\} - s.$$

Thus, as the difficulty of obtaining $f(S)$ this way increments with respect to the number s we choose (we have to solve s ILPs), the smartest way of proceeding is by solving it in the case where $s = \min(S \setminus \{0\})$ (i.e., $s = a_1$, as done by [61]). The AMPL file that represents this approach is shown in Figure 5.13. In our case, the output (Figure 5.14) tells us that $f(S) = 81$. All these files can be found in the public GitHub repository [92]; however, a license for both AMPL and Gurobi is needed in order to run them and obtain the same results (or any result at all).

What we have shown is a classical algorithm for obtaining the Apéry set and the Frobenius number in a general manner that depends on a black box that solves an ILP with global optimality. From now on we will explain the steps we have followed in order to solve this ILP with an adiabatic quantum computer and, specifically, with a D-Wave 2X machine (and also the obstacles we have encountered in that path). In order to transform this ILP problem into the Ising model solved by the D-Wave hardware, one step further involves changing its integer variables into a new set of binary variables with at most a polynomial cost.

Definition 5.4.2. [93] *A binary linear program, or* 0-1LP, *is defined in its canonical form as:*

$$\begin{array}{rl} \min & c^T x \\ \textit{subject to:} & Ax = b \end{array}$$

```
model apery_set_member.mod;
data numerical_semigroup.dat;
let s := a[1];
option solver gurobi;
param m default 0;
for {l in 0..s-1} {
    let i := l;
    solve;
    if T > m then let m := T;
}
param f := m - s;
display f;
```

Figure 5.13: File `frobenius_number.run`

```
f = 81
```

Figure 5.14: Output for `frobenius_number.run`

where $x \in \{0,1\}^n$, $A \in \mathbb{Z}^n \times \mathbb{Z}^m$, $b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^n$.

Both problems are polynomially equivalent, as shown in [93] (Theorem 13.6), where an upper bound for the number of binary variables representing each integer variable from the original ILP problem is given. However, we can tight this number of binary variables by using our knowledge of the problem and one of the lower bounds of the Frobenius number given in [98] (ideally, we could use $f(S)$). In 1935, Russian mathematician Issai Schur proved in a lecture in Berlin [30, 98] the following result:

Theorem 5.4.3. (I. Schur, 1935) *Let $S = \langle a_1, ..., a_n \rangle$ be a numerical semigroup. Then,*

$$f(S) \leq (a_1 - 1)(a_n - 1) - 1.$$

Thus, if we define

$$t_j = 1 + \left\lfloor \log_2 \left(\frac{(a_1-1)(a_n-1)+s-1}{a_j} \right) \right\rfloor$$

for every $j \in \{1, \ldots, n\}$, and

$$u = 1 + \left\lfloor \log_2 \left(\frac{(a_1-1)(a_n-1)+s-i-1}{s} \right) \right\rfloor$$

we have given an upper bound for the number of bits needed to describe every variable x_j in the worst case (we can also run every known bound for the Frobenius number, and stick to the minimum of them, but for now let us just use Schur's result). It follows that the transformation of our problem from ILP form to 0-1LP delivers the minimization problem shown below:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} a_j \sum_{l=0}^{t_j} 2^l x_{jl} \\ \textit{subject to:} \quad & \sum_{j=1}^{n} a_j \sum_{l=0}^{t_j} 2^l x_{jl} = i + s \sum_{m=0}^{u} 2^m k_m \end{aligned}$$

with $x_{jl} \in \{0,1\}$ and $k_m \in \{0,1\}$ for all j, l and m contained in the summation indices.

Let us come back to the definition of the Ising model. In it, we have binary variables which can have the values -1 or 1. On the other hand, in our 0-1LP for calculating the elements of the Apéry set the variables can be in 0 or 1. It is usually more advantageous to redefine the Ising spin problem as a quadratic unconstrained binary optimizacion problem, or QUBO, which consists of the minimization of the objective function

$$Q(x_1, \ldots, x_n) = c_0 + \sum_{i=1}^{n} c_i x_i + \sum_{1 \leq i < j \leq n} q_{ij} x_i x_j,$$

with $x_i \in \{0,1\}$, thus changing the possible states of its variables from 1 and -1 to 1 and 0. The transformation follows from $s_i = 1 - 2x_i$, and can be easily checked. In fact, the documentation from D-Wave allows to program our problem directly into a QUBO formulation [1], besides the Ising model formulation. One detail remains to be figured out: QUBO problems have no constraints, but ours have one. How to proceed?

The most common way to transform a constrained optimization problem into an unconstrained one is by penalizing the constraints, putting them in the objective function. In this manner, we can force the fulfillment of the constraints of the problem by punishing the error committed in them. In our example, as we have a single equality constraint, namely

$$\sum_{j=1}^{n} a_j x_j = i + sk,$$

we can force it to the objective function this way:

$$\min \quad \sum_{j=1}^{n} a_j x_j + \lambda^{(\nu)} \left(\sum_{j=1}^{n} a_j x_j - i - sk \right)^2$$

where $\lambda^{(\nu)}$ is updated after every iteration in the following way, until no change in λ is obtained:

$$\lambda^{(\nu+1)} = \lambda^{(\nu)} + \alpha \cdot \left| \sum_{j=1}^{n} a_j x_j - i - sk \right|$$

with $\alpha > 0$.

In 0-1LP form, we obtain the following QUBO problem:

$$\min \quad \sum_{j=1}^{n} a_j \sum_{l=0}^{t_j} 2^l x_{jl} + \lambda^{(\nu)} \left(\sum_{j=1}^{n} a_j \sum_{l=0}^{t_j} 2^l x_{jl} - i - s \sum_{m=0}^{u} 2^m k_m \right)^2,$$

with its corresponding penalty updating formula:

$$\lambda^{(\nu+1)} = \lambda^{(\nu)} + \alpha \cdot \left| \sum_{j=1}^{n} a_j \sum_{l=0}^{t_j} 2^l x_{jl} - i - s \sum_{m=0}^{u} 2^m k_m \right|$$

with $\alpha > 0$.

Now, we are going to test the performance of this new procedure for obtaining the Apéry set (and the Frobenius number) via a classical solver. For that, we have just to modify the original `apery_set_member.mod` file into a new one with an unconstrained ILP, with the constraint penalized in the objective function as previously exaplained. This new AMPL file can be found in Figure 5.15, along with its corresponding `.run` file in Figure 5.16. Additionaly, we are going to keep track of the number of iterations of the algorithm (i.e., ν), and on the values of $\lambda^{(\nu)}$ needed for the algorithm to stop. This way, if we start with $\lambda^{(0)} = 0$, the output for this algorithm is found in Figure 5.17. On the other hand, if we start with $\lambda^{(0)} = 100$,

```
param n;
set N := 1..n;
param a {N};
param s; #
param i; #
param lambda;
param rest;
var X {N} integer, >= 0;
var K integer;

minimize T:
   sum {j in N} a[j]*X[j]
   + lambda*((sum {j in N} a[j]*X[j] - i - s*K)^2);
```

Figure 5.15: File `apery_set_member_lagr.mod`

the output is shown in Figure 5.18.

Foreseeably, the Apéry set we obtain for S and $s = 30$ is correct, but that should not be a surprise as both optimization problems are equivalent to the original one. What is of interest here is that a low starting point for λ implies a relatively large number of iterations, which translates in more calls to our integer linear programming solver than in the original constrained version. However, if we start with a large enough value of λ, we can obtain the same results without additional calls to the solver and just one iterarion for each of the elements of the Apéry set. On the other hand, if we exceed in the value for λ, we would also need just one iteration per ω_i, but the solver would need more time in order to find the global optimum. The latter behavior can be observed when taking into account the number of simplex iterarions and branch-and-cut nodes that Gurobi needs for each of the problems.

Nevertheless, this unconstrained reformulation of the problem is not meant to be solved by Gurobi or any other integer linear programming solver, but rather by an ideal adiabatic quantum computer, and more concretely by one of the few D-Wave machines that currently exist worldwide. Our unconstrained problem, in QUBO form, is shown below (please note that, if $x \in \{0, 1\}$, then $x = x^2$):

$$c_0 + \sum_{m=0}^{u} c_m k_m + \sum_{j=1}^{n} \left(\sum_{l=0}^{t_j} c_{jl} x_{jl} \right) + \sum_{1 \le j < j' \le n} \left(\sum_{0 \le l < l' \le \min(t_j, t_{j'})} i q_{jl,j'l'} x_{jl} x_{j'l'} \right)$$

$$+ \sum_{j=1}^{n} \left[\sum_{l=0}^{t_j} \left(\sum_{m=0}^{u} q_{jl,m} x_{jl} k_m \right) \right] + \sum_{0 \le m < m' \le u} q_{m,m'},$$

where

$$\begin{aligned}
c_0 &= \lambda^{(\nu)} i^2 \\
c_m &= \lambda^{(\nu)} \left(2^{2m} s^2 + 2^{m+1} i s\right) \\
c_{jl} &= 2^l a_j + \lambda^{(\nu)} \left(2^{2l} a_j^2 - 2^{l+1} a_j i\right) \\
q_{m,m'} &= 2^{m+m'+1} \lambda^{(\nu)} s^2 \\
q_{jl,m} &= 2^{l+m+1} \lambda^{(\nu)} a_j s \\
q_{jl,j'l'} &= 2^{l+l'+1} \lambda^{(\nu)} a_j a_{j'}.
\end{aligned}$$

Right now, an ideal quantum annealer should be able to solve our QUBO problem for finding the members of the Apéry set, provided that such a computer has a sufficiently large enough amount of qubits, and also that the graph connecting the qubits is complete. However, to date there are no quantum annealers that fulfill those requirements (they may be available in the future, though). The latest quantum annealers commercially available are the D-Wave 2X, with 1152 and 3360 couplers (connections between adjacent qubits), and the D-Wave 2000Q, which has 2048 qubits and 6016 couplers. Their graphs are far from being complete, as every

```
model apery_set_member_lagr.mod;
data numerical_semigroup.dat;
let s := 30;
param apery_set {0..s-1};
param lambdas {0..s-1};
param iterations {0..s-1};
param it default 0;
option solver gurobi;

for {l in 0..s-1} {
    let i := l;
    let lambda := 0;
    let it := 0;
    let rest := 1;
    repeat {
        solve;
        let rest := abs(sum {j in N} a[j]*X[j] - i - s*K);
        let lambda := lambda + rest;
       let it := it + 1;
   } while rest != 0;
   let apery_set[l] := T;
   let lambdas[l] := lambda;
   let iterations[l] := it;
}

display apery_set;
display iterations;
display lambdas;
```

Figure 5.16: File `apery_set_lagr.run`

qubit in their grids are at most connected with six other qubits, as shown in Figure

```
   apery_set [*] :=
   0   0    4  34    8  38   12  42   16  46   20  80   24  84   28  88
   1  61    5  65    9  69   13 103   17  77   21 111   25  55   29  89
   2  92    6  66   10 100   14  44   18  78   22  22   26  56
   3  33    7  67   11  11   15  45   19  19   23  23   27  57
   ;

   iterations [*] :=
0  1    3  2    6  2    9  2   12 22   15  2   18  2   21 11   24 40   27  2
1 63    4  2    7  2   10 22   13 21   16  2   19  2   22  2   25  2   28  2
2  2    5  2    8  2   11  2   14  2   17  2   20 44   23  2   26  2   29  3
;

lambdas [*] :=
   0   0    4  34    8  68   12  32   16  46   20  62   24  62   28  58
   1  62    5  35    9  39   13  62   17  47   21  90   25  85   29  90
   2  62    6  66   10  90   14 104   18  78   22 112   26  56
   3  93    7  67   11 101   15  45   19  79   23  23   27  57
   ;
```

Figure 5.17: Output for `apery_set_lagr.run` and $\lambda^{(0)} = 0$

```
apery_set [*] :=
 0   0    4  34    8  38   12  42   16  46   20  80   24  84   28  88
 1  61    5  65    9  69   13 103   17  77   21 111   25  55   29  89
 2  92    6  66   10 100   14  44   18  78   22  22   26  56
 3  33    7  67   11  11   15  45   19  19   23  23   27  57
;

iterations [*] :=
 0 1    3 1    6 1    9 1   12 1   15 1   18 1   21 1   24 1   27 1
 1 1    4 1    7 1   10 1   13 1   16 1   19 1   22 1   25 1   28 1
 2 1    5 1    8 1   11 1   14 1   17 1   20 1   23 1   26 1   29 1
;

lambdas [*] :=
 0 100    4 100    8 100   12 100   16 100   20 100   24 100   28 100
 1 100    5 100    9 100   13 100   17 100   21 100   25 100   29 100
 2 100    6 100   10 100   14 100   18 100   22 100   26 100
 3 100    7 100   11 100   15 100   19 100   23 100   27 100
;
```

Figure 5.18: Output for `apery_set_lagr.run` and $\lambda^{(0)} = 100$

5.21 (it may be less than that, as some qubits may be off after the last recalibration of the machine).

The importance of the completeness of the graph is problem dependant. Our QUBO instance, for example, has a complete connectivity graph between its variables. With an ideal quantum annealer we would have no problem, but with the D-Wave machine it is mandatory to transform our problem graph into an alternative graph that could be embedded into the Chimera graph. In other words, we have to solve an instance of the subgraph isomorphism problem, which happens to be in NP-complete. Even more, our problem may not be embeddable inside the D-Wave (for example, for the 1152 qubit Chimera grid of the D-Wave 2X, the largest complete graph that can be embedded into it is believed to be the K_{33}). Research on the subject of embedding a problem graph into D-Wave's Chimera graph can be found in [34] and [35], where the concepts of *embedding* and *parameter setting* are explained.

There is a way to skip these limitations, by solving subinstances of our graph instead of the complete graph. For that, D-Wave released a graph partitioning open source library called `qbsolv` [26]. Its corresponding executable needs a certain kind of file format (called `.qubo`), to work. For example, if our QUBO instance is defined by

$$2.6x_0 + 4.5x_1 - 1.8x_2 + 3.5x_0x_1 + 2x_1x_2,$$

where $x_0, x_1, x_2 \in \{0, 1\}$, its corresponding `.qubo` file is the one shown in Figure 5.19. The format is quite: the first line always starts with `p qubo 0`, followed by the number of variables, the number of nonzero diagonals, and the number of nonzero couplings. The output obtained by `qbsolv` for this file can be seen in Figure 5.20.

```
p qubo 0 3 3 2
0 0 2.6
1 1 4.5
2 2 -1.8
0 1 3.5
1 2 2.0
```

Figure 5.19: Example of `.qubo` file

```
3 bits, find Min, SubMatrix= 47, -a o, timeout=2592000.0 sec
001
-1.80000 Energy of solution
0 Number of Partitioned calls, 1 output sample
0.03002 seconds of classic cpu time
```

Figure 5.20: Output for the example `.qubo` file

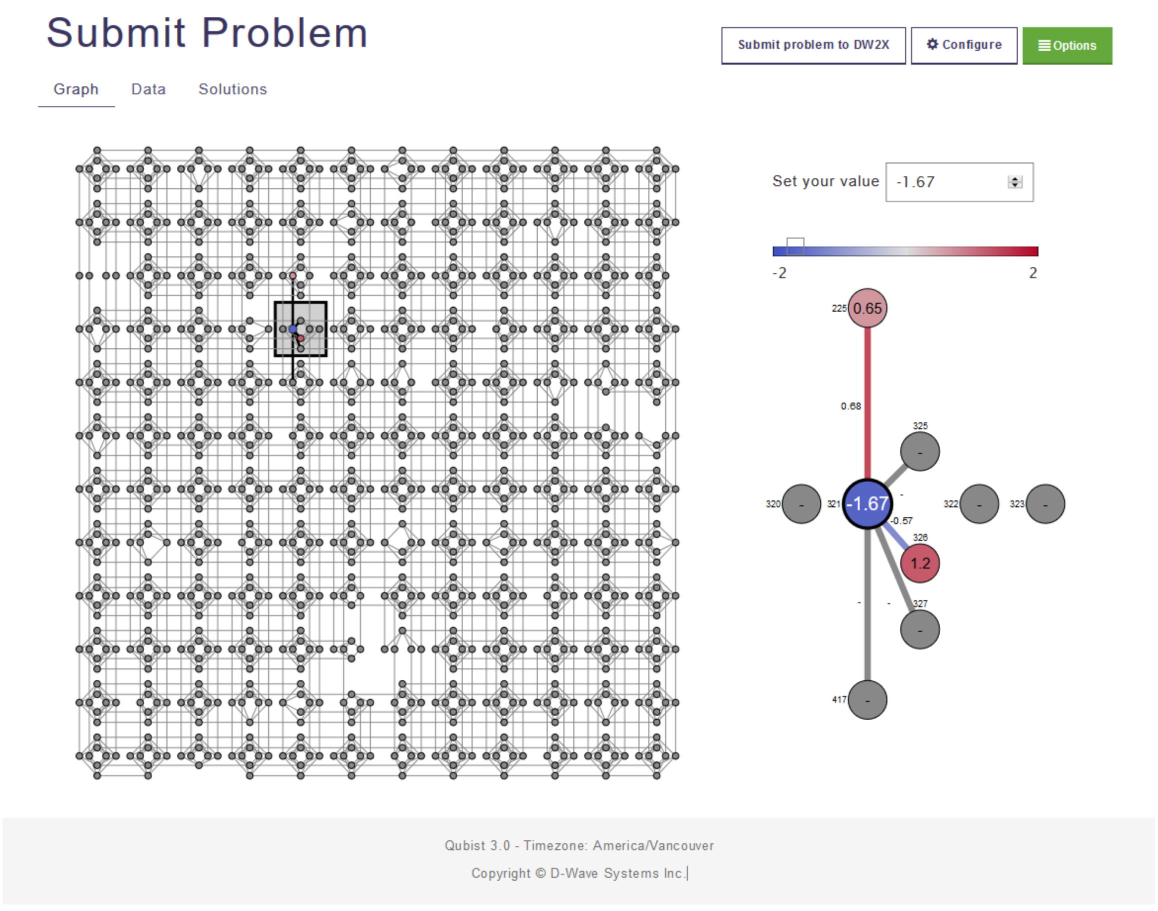


Figure 5.21: Introducing a problem inside the D-Wave 2X

Please note that, in the previous paragraph, we have shown the solution that `qbsolv` obtains for a certain instance of the problem. This is because `qbsolv` has an auxiliary internal optimization solver based on the tabu search [58], which gives a solution for the subproblems and then unifies all the solutions into the complete one for all the variables. However, this classical method does not guarantee a global solution, and is of no help to us in the general case. We can decide if `qbsolv` tries to solve the problem with its tabu search, of if we prefer to connect to a D-Wave machine.

Finally, our QUBO subproblem is solved with the D-Wave 2X via one of the possible inputs allowed. The first one is shown in Figure 5.21, while the second one is again a `.qubo` file obtained internally with `qbsolv`. As part of the project Joint Research Unit Repsol-ITMATI (code file: IN853A 2014/03), we had the opportunity to try the D-Wave 2X machine based on the University of Southern California. However, due to the amount of subproblems needed to solve a proper instance of the Frobenius problem or the Apéry set, it was impractical to do so with the scarce amount of time given and the current size of the graph of the D-Wave 2X machine. In the last chapter we discuss the conclusions at which we arrived during the process.

6

Conclusions and Future Work

What we have presented in this doctoral dissertation is a preliminary study of the capabilities of quantum computation when it comes to solve numerical semigroup-related problems. In order to do that, we have first defined what numerical semigroups are, and then, drawing upon the principles of computational complexity theory, we prove that the problems we want to answer with hypothetical quantum computers are in fact hard to solve. Thus, this work is a first attempt at solving one of the most computationally demanding problems in basic algebra with this emerging and somewhat promising computing paradigm. As have been explained, calculating the Sylvester denumerant and the Frobenius number of a numerical semigroup are not even known to be in the computational class NP (i.e., just deciding if a certain integer is the Frobenius number or the Sylvester denumerant is a challeging problem).

With that in mind, it is important to clarify that we did not expect in any moment to find polynomial-time quantum algorithms for those problems. As with the Sankara stones, doing so would have triggered a series of actions that would have eventually led us to never-ending fortune and glory. It is also important to take into account that quantum computing is not a panacea for computationally hard problems. In fact, there are not many problems that have quantum algorithms faster than their classical counterparts (without the involvement of oracles). Even more, in those few special cases it remains to be proven yet that a classical algorithm comparable to the quantum one in terms of efficiency does not exist. The best example is Shor's algorithm: right now, it is the only known polynomial time algorithm for the factoring problem; however, even though the general number field sieve (i.e., the most efficient classical factoring algorithm) is super-polynomial, it has not been proven that there are no classical polynomial factoring algorithms. Proving so will answer the question of whether there are NP problems not in P.

During these years we have also followed a lead (unrelated to quantum computing) that, if finally proven true, would strengthen the hard nature of the Frobenius problem. There exist the possibility that the Frobenius number is not just one of the problems in the class NP-hard, but also a problem in the class PSPACE-hard (please note that NP $\subseteq$ PSPACE and that the equality, although not properly discarded yet, would be startling). Details of this remain, however, to be worked out yet; due to time limitations we did not arrive to any definite —and, of course, mathematically proven— conclusion during the span of this doctoral dissertation.

As with quantum computing, to date the only algorithm that is proven to be strictly more efficient that its classical counterpart is Grover's Search. However, in order to use this algorithm, one must have a way of identifying in a reasonable time (i.e., polynomial with respect to the size of the input) whether a possible solution is in fact a solution. This, of course, limits its usage to problems in the class NP, and is the reason why we have solved the Sylvester denumerant problem with the help of an oracle for the NSMP. There exists, however, the possibility that powerful quantum gate algorithms exist in their own right for the Sylvester denumerant (without having to find the actual solutions for the NSMP) and also for the Frobenius problem, but the search must continue from this point. The lack of results since Shor and Grover gave birth to their clever ideas is, nonetheless, discouraging, and may be an indicative that the only advantage of quantum computers with respect to classical ones would and ever would be a quadratic speedup in database searches. Furthermore, there are other problems that arise when trying to construct a quantum computer in practice; we have not covered them here, but we refere to [6] for a brief and clever introduction.

Regarding the algorithms for the Apéry set and the Frobenius number, there are two aspects that need to be improved prior to completing a study of its feasibility and performance. First, the current graph (i.e., Chimera graph) architecture of the available adiabatic quantum computers (i.e., the D-Wave machines) extremely obstruct the resolution of problems that have an almost full connectivity index between its variables, as in this case (the use of `qbsolv` is just a temporary workaround, or it should be as so). And second, current adiabatic quantum computers do not guarantee global optimality, as opposed to the theoretical result deducted from the adiabatic theorem; in reality, the D-Wave just make a few runs of the process (instead of just one, as would be in the theoretical case) and, following a certain probability distribution, try to guarantee that the best of the solutions obtained by those runs is in fact the global solution to the problem. This solution, however, cannot be proven to be the global optimum (as global optimality is not known to be in the class NP), which makes useless our attempt to find those two combinatorial invariants via current adiabatic quantum computers. In the future, with more reliable quantum annealers, however, this solution may prove to be effective and faster, but for now it is just a theoretical method.

7

Bibliography

[1] *D-Wave: Programming with QUBOs. Release 1.5.1-beta4, 09-1002A-B.*, tech. rep., D-Wave Systems, Inc., 2016.

[2] *D-Wave: Programming with QUBOs. Release 2.3, 09-1002A-B.*, tech. rep., D-Wave Systems, Inc., 2016.

[3] *Gurobi Optimizer Reference Manual.* http://www.gurobi.com, 2018. Accessed: 2019-02-23.

[4] S. Aaronson, $P \overset{?}{=} NP$, in Open Problems in Mathematics, Springer, 2016, pp. 1–122.

[5] S. Aaronson, G. Kuperberg, and C. Granade, *The Complexity Zoo.* https://complexityzoo.uwaterloo.ca/Complexity_Zoo, 2005. Accessed: 2017-04-25.

[6] S. Aaronson and Z. Weinersmith, *The Talk.* http://www.smbc-comics.com/comic/the-talk-3, 2016. Accessed: 2019-03-03.

[7] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Annals of Mathematics, 160 (2004), pp. 781–793.

[8] D. Aharonov, W. Van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev, *Adiabatic quantum computation is equivalent to standard quantum computation*, SIAM Review, 50 (2008), pp. 755–787.

[9] T. Albash and D. A. Lidar, *Adiabatic quantum computation*, Reviews of Modern Physics, 90 (2018), p. 015002.

[10] R. Apéry, *Géométrie algébrique - sur les branches superlineaires des courbes algebriques*, Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences, 222 (1946), pp. 1198–1200.

[11] ——, *Irrationalité de ζ (2) et ζ (3)*, Astérisque, 61 (1979), p. 1.

[12] B. Aspvall and R. E. Stone, *Khachiyan's linear programming algorithm*, Journal of Algorithms, 1 (1980), pp. 1–13.

[13] V. Baldoni, N. Berline, J. De Loera, B. Dutra, M. Koeppe, and M. Vergne, *Coefficients of Sylvester's denumerant*, arXiv preprint arXiv:1312.7147, (2013).

[14] F. Barahona, *On the computational complexity of Ising spin glass models*, Journal of Physics A: Mathematical and General, 15 (1982), p. 3241.

[15] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, *Elementary gates for quantum computation*, Physical Review A, 52 (1995), p. 3457.

[16] M. Beck, I. M. Gessel, and T. Komatsu, *The polynomial part of a restricted partition function related to the Frobenius problem*, The Electronic Journal of Combinatorics, 8 (2001), p. 7.

[17] E. Bell, *Interpolated denumerants and Lambert series*, American Journal of Mathematics, 65 (1943), pp. 382–386.

[18] J. S. Bell, *On the Einstein Podolsky Rosen paradox*, Physics, 1 (1964), pp. 195–200.

[19] P. Benioff, *The computer as a physical system: a microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines*, Journal of Statistical Physics, 22 (1980), pp. 563–591.

[20] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and weaknesses of quantum computing*, SIAM Journal on Computing, 26 (1997), pp. 1510–1523.

[21] D. J. Bernstein and A. K. Lenstra, *A general number field sieve implementation*, in The Development of the Number Field Sieve, Springer, 1993, pp. 103–126.

[22] E. Bernstein and U. Vazirani, *Quantum complexity theory*, SIAM Journal on Computing, 26 (1997), pp. 1411–1473.

[23] E. Bézout, *Théorie générale des équations algébriques; par m. Bézout...*, de l'imprimerie de Ph.-D. Pierres, rue S. Jacques, 1779.

[24] F. Bloch, *Nuclear induction*, Physical Review, 70 (1946), p. 460.

[25] S. Boixo, T. F. Rønnow, S. V. Isakov, Z. Wang, D. Wecker, D. A. Lidar, J. M. Martinis, and M. Troyer, *Quantum annealing with more than one hundred qubits*, arXiv preprint arXiv:1304.4595, (2013).

[26] M. Booth, S. Reinhardt, and A. Roy, *Partitioning optimization problems for hybrid classical/quantum execution*, tech. rep., D-Wave Technical Report, 2017.

[27] M. Born and V. Fock, *Beweis des Adiabatensatzes*, Zeitschrift für Physik, 51 (1928), pp. 165–180.

[28] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, *Tight bounds on quantum searching*, arXiv preprint quant-ph/9605034, (1996).

[29] G. Brassard, P. Høyer, and A. Tapp, *Quantum counting*, in Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP), Springer, 1998, pp. 820–831.

[30] A. Brauer, *On a problem of partitions*, American Journal of Mathematics, 64 (1942), pp. 299–312.

[31] A. Brauer and J. E. Shockley, *On a problem of Frobenius*, Journal für die Reine und Angewandte Mathematik, 211 (1962), pp. 215–220.

[32] J. Brooke, D. Bitko, G. Aeppli, et al., *Quantum annealing of a disordered magnet*, Science, 284 (1999), pp. 779–781.

[33] J. Brooke, T. Rosenbaum, and G. Aeppli, *Tunable quantum tunnelling of magnetic domain walls*, Nature, 413 (2001), p. 610.

[34] V. Choi, *Minor-embedding in adiabatic quantum computation: I. The parameter setting problem*, Quantum Information Processing, 7 (2008), pp. 193–209.

[35] ——, *Minor-embedding in adiabatic quantum computation: II. Minor-universal graph design*, Quantum Information Processing, 10 (2011), pp. 343–353.

[36] H. Cohen, *A course in computational algebraic number theory*, vol. 138, Springer Science & Business Media, 2013.

[37] S. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, ACM, 1971, pp. 151–158.

[38] ——, *The P versus NP problem*, in The Millennium Prize Problems, American Mathematical Society, 2006, pp. 87–104.

[39] T. S. Cubitt, D. Perez-Garcia, and M. M. Wolf, *Undecidability of the spectral gap*, Nature, 528 (2015), p. 207.

[40] A. Currin, K. Korovin, M. Ababi, K. Roper, D. B. Kell, P. J. Day, and R. D. King, *Computing exponentially faster: implementing a non-deterministic universal Turing machine using DNA*, Journal of the Royal Society Interface, 14 (2017), p. 20160990.

[41] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 400 (1985), pp. 97–117.

[42] ——, *Quantum computational networks*, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 425 (1989), pp. 73–90.

[43] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 439 (1992), pp. 553–558.

[44] Z. Diao, C. Huang, and K. Wang, *Quantum counting: algorithm and error distribution*, Acta Applicandae Mathematicae, 118 (2012), pp. 147–159.

[45] P. A. M. Dirac, *A new notation for quantum mechanics*, Mathematical Proceedings of the Cambridge Philosophical Society, 35 (1939), pp. 416–418.

[46] D. P. DiVincenzo, *Two-bit gates are universal for quantum computation*, Physical Review A, 51 (1995), p. 1015.

[47] A. Einstein, B. Podolsky, and N. Rosen, *Can quantum-mechanical description of physical reality be considered complete?*, Physical Review, 47 (1935), p. 777.

[48] A. Ekert and R. Jozsa, *Quantum computation and Shor's factoring algorithm*, Reviews of Modern Physics, 68 (1996), p. 733.

[49] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda, *A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem*, Science, 292 (2001), pp. 472–475.

[50] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, *Quantum computation by adiabatic evolution*, arXiv preprint quant-ph/0001106, (2000).

[51] W. Feller, *An Introduction to Probability Theory and its Applications*, 1957.

[52] R. P. Feynman, *Simulating physics with computers*, International Journal of Theoretical Physics, 21 (1982), pp. 467–488.

[53] R. Fourer, D. M. Gay, and B. W. Kernighan, *A modeling language for mathematical programming*, Management Science, 36 (1990), pp. 519–554.

[54] ——, *AMPL: A Modeling Language for Mathematical Programming*, Duxbury Press, 2002.

[55] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[56] W. Gerlach and O. Stern, *Der experimentelle Nachweis der Richtungsquantelung im Magnetfeld*, Zeitschrift für Physik, 9 (1922), pp. 349–352.

[57] J. Gill, *Computational complexity of probabilistic Turing machines*, SIAM Journal on Computing, 6 (1977), pp. 675–695.

[58] F. Glover, *Future paths for integer programming and links to artificial intelligence*, Computers & Operations Research, 13 (1986), pp. 533–549.

[59] O. Goldreich, *Computational complexity: a conceptual perspective*, ACM SIGACT News, 39 (2008), pp. 35–39.

[60] D. Gottesman, *The Heisenberg representation of quantum computers*, arXiv preprint quant-ph/9807006, (1998).

[61] H. Greenberg, *An algorithm for a linear Diophantine equation and a problem of frobenius*, Numerische Mathematik, 34 (1980), pp. 349–352.

[62] R. Greenlaw and H. J. Hoover, *Fundamentals of the theory of computation: principles and practice*, Morgan Kaufmann, 1998.

[63] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, ACM, 1996, pp. 212–219.

[64] ——, *Quantum mechanics helps in searching for a needle in a haystack*, Physical Review Letters, 79 (1997), p. 325.

[65] ——, *From Schrödinger's equation to the quantum search algorithm*, American Journal of Physics, 69 (2001), pp. 769–777.

[66] I. Hen, *Period finding with adiabatic quantum computation*, EPL (Europhysics Letters), 105 (2014), p. 50005.

[67] E. Ising, *Report on the theory of ferromagnetism*, Zeitschrift für Physik, 31 (1925), pp. 253–258.

[68] M. W. Johnson, M. H. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, et al., *Quantum annealing with manufactured spins*, Nature, 473 (2011), p. 194.

[69] R. Jozsa, *Quantum factoring, discrete logarithms, and the hidden subgroup problem*, Computing in Science & Engineering, 3 (2001), pp. 34–43.

[70] R. Jozsa and N. Linden, *On the role of entanglement in quantum-computational speed-up*, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 459 (2003), pp. 2011–2032.

[71] T. Kadowaki and H. Nishimori, *Quantum annealing in the transverse Ising model*, Physical Review E, 58 (1998), p. 5355.

[72] T. Kato, *On the adiabatic theorem of quantum mechanics*, Journal of the Physical Society of Japan, 5 (1950), pp. 435–439.

[73] L. G. Khachian, *A polynomial algorithm in linear programming*, Doklady Akademii Nauks SSR, 244 (1979), pp. 1093–1096.

[74] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, Science, 220 (1983), pp. 671–680.

[75] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, et al., *Jupyter Notebooks: a publishing format for reproducible computational workflows*, in 20th International Conference on Electronic Publishing, 2016, pp. 87–90.

[76] D. E. Knuth, *The Art of Computer Programming, vol. 1: Fundamental Algorithms*, Addison-Wesley, 1968.

[77] D. C. Kozen, *Automata and Computability*, Springer Science & Business Media, 2012.

[78] M. Kraitchik, *Recherches sur la théorie des nombres*, vol. 1, Gauthier-Villars, 1924.

[79] R. E. Ladner, *On the structure of polynomial time reducibility*, Journal of the ACM, 22 (1975), pp. 155–171.

[80] B. Lanyon, T. Weinhold, N. K. Langford, M. Barbieri, D. James, A. Gilchrist, and A. White, *Experimental demonstration of a compiled version of Shor's algorithm with quantum entanglement*, Physical Review Letters, 99 (2007), p. 250505.

[81] D. H. Lehmer and R. E. Powers, *On factoring large numbers*, Bulletin of the American Mathematical Society, 37 (1931), pp. 770–776.

[82] A. K. Lenstra, H. W. Lenstra, M. S. Manasse, and J. M. Pollard, *The number field sieve*, in The Development of the Number Field Sieve, Springer, 1993, pp. 11–42.

[83] L. A. Levin, *Universal sequential search problems*, Problemy Peredachi Informatsii, 9 (1973), pp. 115–116.

[84] S. Lomonaco, *Shor's quantum factoring algorithm*, Proceedings of Symposia in Applied Mathematics, 58 (2002), pp. 161–180.

[85] C.-Y. Lu, D. E. Browne, T. Yang, and J.-W. Pan, *Demonstration of a compiled version of Shor's quantum factoring algorithm using photonic qubits*, Physical Review Letters, 99 (2007), p. 250504.

[86] Y. Manin, *The computable and not computable (in Russian)*, 1980.

[87] C. C. McGeoch, *Adiabatic quantum computation and quantum annealing: Theory and practice*, Synthesis Lectures on Quantum Computing, 5 (2014), pp. 1–93.

[88] A. Messiah, *Quantum Mechanics, vol. 1*, John Wiley & Sons, New York, 1958.

[89] A. Mizel, D. A. Lidar, and M. Mitchell, *Simple proof of equivalence between adiabatic quantum computation and the circuit model*, Physical Review Letters, 99 (2007), p. 070502.

[90] M. A. Morrison and J. Brillhart, *A method of factoring and the factorization of F7*, Mathematics of Computation, 29 (1975), pp. 183–205.

[91] M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*, Cambridge University Press, 2002.

[92] J. Ossorio-Castillo, *jqnoc/numsem: numsem console*. https://doi.org/10.5281/zenodo.1257967, 2018.

[93] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Courier Corporation, 1998.

[94] C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, Computational Methods in Number Theory, Part I, (1982), pp. 89–139.

[95] ——, *The quadratic sieve factoring algorithm*, in Workshop on the Theory and Application of of Cryptographic Techniques, Springer, 1984, pp. 169–182.

[96] ——, *A tale of two sieves*, Notices of the American Mathematical Society, 43 (1996), pp. 1473–1485.

[97] J. L. Ramírez-Alfonsín, *Complexity of the Frobenius problem*, Combinatorica, 16 (1996), pp. 143–147.

[98] ——, *The Diophantine Frobenius problem*, vol. 30, Oxford University Press on Demand, 2005.

[99] B. W. Reichardt, *The quantum adiabatic optimization algorithm and local minima*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, 2004, pp. 502–510.

[100] J. Roland and N. J. Cerf, *Quantum search by local adiabatic evolution*, Physical Review A, 65 (2002), p. 042308.

[101] J. C. Rosales and P. A. García-Sánchez, *Numerical semigroups*, vol. 20, Springer Science & Business Media, 2009.

[102] G. Rose and W. Macready, *An introduction to quantum annealing*, D-Wave Systems, (2007).

[103] E. Schrödinger, *An undulatory theory of the mechanics of atoms and molecules*, Physical Review, 28 (1926), p. 1049.

[104] ——, *Discussion of probability relations between separated systems*, Mathematical Proceedings of the Cambridge Philosophical Society, 31 (1935), pp. 555–563.

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

[106] D. R. Simon, *On the power of quantum computation*, SIAM Journal on Computing, 26 (1997), pp. 1474–1483.

[107] M. Sipser, *The history and status of the P versus NP question*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, ACM, 1992, pp. 603–618.

[108] ——, *Introduction to the theory of computation*, vol. 2, Thomson Course Technology Boston, 2006.

[109] J. J. Sylvester, *On the partition of numbers*, Quarterly Journal of Pure and Applied Mathematics, 1 (1857), pp. 141–152.

[110] ——, *Outlines of seven lectures on the partitions of numbers*, Proceedings of the London Mathematical Society, 1 (1896), pp. 33–96.

[111] A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, 2 (1937), pp. 230–265.

[112] W. Van Dam, M. Mosca, and U. Vazirani, *How powerful is adiabatic quantum computation?*, in Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on, IEEE, 2001, pp. 279–287.

[113] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*, Nature, 414 (2001), p. 883.

[114] N. Xu, J. Zhu, D. Lu, X. Zhou, X. Peng, and J. Du, *Quantum factorization of 143 on a dipolar-coupling nuclear magnetic resonance system*, Physical Review Letters, 108 (2012), p. 130501.

[115] A. C.-C. Yao, *Quantum circuit complexity*, in Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science, IEEE, 1993, pp. 352–361.