POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Design of a Quantum Circuit for Quantum Random Walks on Johnson Graphs

Solving the ISD problem for Code-based Post-quantum Cryptosystems

Tesi di Laurea Magistrale in
Computer Engineering - Ingegneria Informatica

Author: **Giacomo Lancellotti, Matteo Lodi**

Student ID: 928088, 928407
Advisor: Prof. Alessandro Barenghi
Co-advisors: Prof. Gerardo Pelosi, Simone Perriello
Academic Year: 2020-21

Abstract

Code-based cryptosystems have been thoroughly studied in literature due to their quantum resistant properties. The most effective attacks to such systems are based on solving the Information Set Decoding problem. Our work is based on Kachigar and Tillich's studies on Quantum Random Walks used to tackle this issue. We detail the construction of a quantum circuit that explores a Johnson graph, which to the best of our knowledge has never been accomplished in literature. This thesis also shows how to apply such circuit to solve a sub-procedure of the Finiasz-Sendrier ISD algorithm. We provide a quantitative analysis of the performance of the circuit in term of the required amount of qubits and total quantum gates.

Keywords: Coding Theory, Information Set Decoding, Quantum Computing, Post-Quantum Cryptography, Quantum Random Walks, Johnson Graph

Abstract in lingua italiana

I crittosistemi basati su codici lineari sono oggetto di approfonditi studi a causa delle loro proprietà di resistenza agli attacchi quantistici. Gli attacchi più efficaci a questi sistemi sono basati sul risolvedere il problema chiamato *Information Set Decoding*. Il nostro lavoro si fonda sugli studi di Kachigar e Tillich sulla risoluzione del problema sfruttando le Camminate Casuali Quantistiche. Mostriamo in dettaglio la costruzione di un circuito quantistico che esplora un grafo di Johnson, circuito che, al meglio delle nostre conoscenze, non è mai stato definito in letteratura. Questa tesi definisce come applicare il suddetto circuito per risolvere una sotto procedura dell'algoritmo di ISD di Finiasz-Sendrier. Forniamo infine un'analisi quantitativa delle prestazioni del circuito in termini di numero di qubit e porte quantistiche usate.

Parole chiave: Teoria dei Codici, Information Set Decoding, Computazione Quantistica, Crittografia Post-Quantistica, Camminate Casuali Quantistiche, Grafi di Johnson

Contents

Introduction

In recent years, Quantum computers have drastically risen in popularity mainly due to big investments from both public and private institutions. What 40 years ago was merely conceptualized as a mathematical construct, today is a reality. The main reason behind this technological break-through is that we are trying to achieve the so called *Quantum Advantage*: the possibility of solving very specific problems in a much more efficient way compared to classical methods of computation.

Cryptography is one of the mathematical fields that are deeply influenced by such advantage, which is the reason why today's literature focuses on a category of algorithms defined as quantum-resistant. Code-based cryptosystems rely on the hardness of finding a minimum-weight codeword in a seemingly randomly structured linear code and, according to literature, they seem to fit this category. The most effective attacks to such systems try to solve what is called the Information Set Decoding problem, as a consequence any Quantum procedure that could speedup its solving is of great significance.

In this work we show a circuital implementation of a Quantum random walk in a Johnson graph which, to the best of our knowledge, has only been studied in literature from a pure mathematical point of view. We also prove that this solution is mathematically correct by comparing simulation results obtained from Qiskit, IBM's framework for quantum simulation, with Thomas Wong's work [39] on Lackadaisical Quantum random walks. In addition, we also analyze the circuit's performance when applied to Finasz-Sendrier's Information Set Decoding algorithm, indicating its gate number as well as depth using the algorithm's parameters.

In Chapter 1 we present the state of the art regarding Coding Theory. Specifically, we focus on Linear Codes, Decoding Strategies and a brief explanation of how McEliece and Niederretier's cryptosystems work. An analysis of the main Information Set Decoding algorithms is also present.

Chapter 2 explores the main properties and physical laws that rule the world of Quantum Computing. A short description of how classical computation works introduces the concept, followed by a description of Quantum Algorithms as well as their speedup and

complexity.

An overview of a key topic is present in Chapter 3: Quantum Random Walks. It starts from the description of the classical counterpart followed by the Quantum version explaining their difference. The chapter continues by analyzing their properties regarding search problems and shows an Hypercube example. The last Sections address such walks on Johnson graphs. After a brief mathematical analysis, we explain in detail our circuit implementation of a quantum random walk on a $J(4, 2)$ graph, showing the results of different simulations as well as gate and depth analysis.

Chapter 4 details the application of our walk on a Information Set Decoding algorithm called Finiasz-Sendrier. After specifying the its encoding on the graph, we proceed to analyze phase by phase how the quantum circuit accomplishes a sub-procedure of the original algorithm. We end the Chapter by analyzing its complexity, showing experimental results as well as evaluating the performance of the walk.

Appendix A lists basic linear algebra elements that are used throughout this work.

1 | Coding Theory

For millennia mankind has been looking for new ways of communication, from the invention of new languages to the modern internet. A turning point was reached with the birth of the first electronic communication devices, as a consequence the need for a new theory of communication with a solid mathematical foundation emerged.

The first developments of this new theory are attributed to Claude Shannon [32] [33] who was the first to realise its potential. Its most general scheme is to send a message through a noisy channel and receive it without it being compromised such as Figure 1.1. The basic idea, as intuitive as it is elegant, is to add redundant information to the message to protect it from a certain number of errors due to external noise.

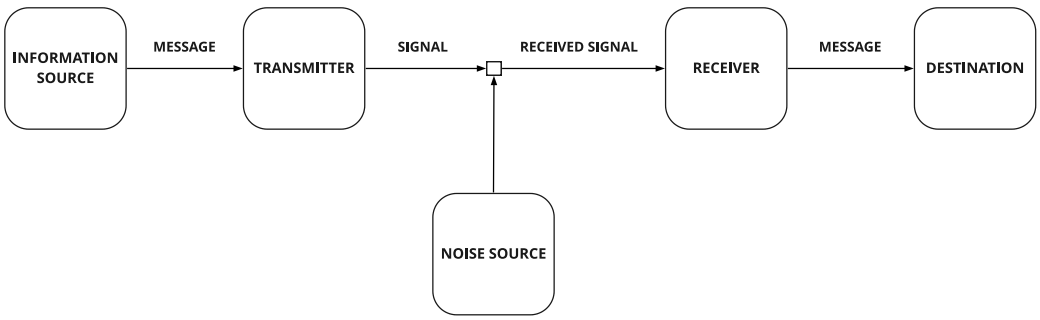


Figure 1.1: High-level description of the communication model proposed by Shannon in [32]

How much redundancy do we need to correct a given number of errors? Intuitively, maximising the number of corrected errors and minimising redundancy are two opposite goals. The objective is to find the optimal trade-off for the specific application. From this point on, we will focus on a sub-class of codes that is more relevant to the purpose of this work, the linear codes. The notation and mathematical definitions that will follow are based on the work of Guruswami [15] while the style of pseudo-algorithm is drawn from the thesis work of Perriello [27], similar to those that can be found in literature on the subject.

1.1. Linear Codes

1.1.1. Notation

The first mathematical notation we need to introduce is $\mathbb{F}_q$, which indicates a finite field of order q. In this manuscript we will mainly work with order $q = 2$, i.e. finite fields with binary coefficients. Over a finite field we can define other mathematical structures such as $\mathbf{v} \in \mathbb{F}_q^n$ which is a column vector of n elements and its transposed $\mathbf{v}^T$. With $M \in \mathbb{F}_q^{x \times y}$ we define all the $(x \times y)$-sized matrices with entries over the field. Same as before, M^T is the transposed matrix. Special cases are $0_{x \times y}$ and I_r, which denote the $x \times y$ all-zero matrix and the $r \times r$ identity matrix, respectively. Given $A \in \mathbb{F}_q^{x \times y}$, to show that the concatenation of the columns of the matrices $A_1 \in \mathbb{F}_q^{x \times z}$ and $A_2 \in \mathbb{F}_q^{x \times (y-z)}$ results in the starting matrix, we use the special notation $A = [A_1|A_2]$. Moreover another important concept is indexing/projection $A_{\mathcal{I}}$. Given a matrix $A \in \mathbb{F}_q^{x \times y}$, $A_{\mathcal{I}}$ consists of the column vectors of A that are indexed by a set $\mathcal{I} \subseteq \{1, ..., n\}$.

1.1.2. Definition

To be complete with our definition of linear codes, we must first define what a linear subspace is.

Definition 1.1 (Linear Subspace). *A non-empty subset $S \subseteq \mathbb{F}_q^n$ is a linear subspace if the linearity property hold:*

1. *$\forall \boldsymbol{x}, \boldsymbol{y} \in S$, $\boldsymbol{x} + \boldsymbol{y} \in S$ where addition is the vector addition over the finite field $\mathbb{F}_q$.*
2. *$\forall$ scalar $\alpha \in \mathbb{F}_q$ and $\boldsymbol{x} \in S$, $\alpha \cdot \boldsymbol{x} \in S$ where the multiplication is done component-wise over the finite field $\mathbb{F}_q$.*

Definition 1.2 (Span). *A set S of vectors from a vector space, denoted span(S), is the smallest linear subspace which contains the set. It can be characterized either as the intersection of all linear subspaces that contain S, or as the set of linear combinations of elements of S.*

Therefore starting from the above definition, a linear code can be represented as the span of a minimal set of codewords or a basis in terms of linear algebra.

Definition 1.3 (Linear Code). *Assume q to be a prime power, namely $q = p^s$ for some prime p and some integer $s \geq 1$. $\mathcal{C} \subseteq \{0, 1, \ldots, q-1\}^n$ is a* linear code *if it is a linear subspace of $\{0, 1, \ldots, q-1\}^n$. If $\mathcal{C}$ has dimension k it will be written as $[n, k]_q$. The*

vectors $(\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_k)_q \in \mathcal{C}$ *are called codewords of* $\mathcal{C}$.

Usually, for clarity of expression, the basis codewords are grouped into a non-unique matrix called generator matrix for the code $\mathcal{C}$.

Definition 1.4 (Generator Matrix)**.** *If* $S \subseteq \mathbb{F}_q^n$ *is a linear subspace there exists at least a set of vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in S$ *called basis element such that every* $\boldsymbol{x} \in S$ *can be expressed as* $\boldsymbol{x} = a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + ... + a_n\boldsymbol{v}_n$ *with* $a_i \in \mathbb{F}_q$ *and* $i \in [1, k]$*. We denote as* G *or generator matrix a full rank* $k \times n$ *matrix with entries from* $\mathbb{F}_q$ *which can express every element of* S*, in other words* $\boldsymbol{x} = (a_1, a_2, ..., a_k) \cdot G$.

$$G = \begin{bmatrix} \leftarrow \mathbf{v}_1 \rightarrow \\ \leftarrow \mathbf{v}_2 \rightarrow \\ \vdots \\ \leftarrow \mathbf{v}_k \rightarrow \end{bmatrix}$$

We can give a more compact description of linear code which we will follow for the rest of the manuscript. It is easy to understand that a codeword $\mathbf{c}$ is just a linear combination of the rows of the generator matrix.

Remark 1.1. *With our previous definitions we can view a linear code as a set of vectors,* $\mathcal{C} := \{\boldsymbol{c} \in \mathbb{F}_q^n | \boldsymbol{c} = G^\top \boldsymbol{m}, G \in \mathbb{F}_q^{k\times n}, \boldsymbol{m} \in \mathbb{F}_q^k\}$.

The generator matrix allows us to encode any word $\mathbf{m}$ of length k into a codeword $\mathbf{c}$ of length n belonging to $\mathcal{C}$. It's worth noticing that part of the information contained in the codeword is not used to encode the message (as it is shorter in length). This leads us to give a better definition of *Information* vs. *Redundancy*. To formalize this notion we need a couple of definitions.

Definition 1.5 (Information Set)**.** *An information set of* $\mathcal{C}$ *is a set of coordinates which correspond to* k *linear independent columns of* G.

By duality the remaining $n - k$ columns of G form the Redundancy set.

We have seen how we can divide the information contained in a linear code, but we have not yet given a definition of how efficient this system is, the efficiency of a code can be measured in terms of the code rate.

Definition 1.6 (Code Rate)**.** *Code rate or information rate is defined as* $R = \frac{k}{n}$.

It is useful now to introduce the concept of distance between codewords in order to better

understand how they work. This is of fundamental importance as it is directly linked to the number of errors that can be corrected.

Definition 1.7 (Hamming Distance). *Given two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^m$ the Hamming distance dist($\boldsymbol{x}$,$\boldsymbol{y}$) is defined as the number of positions in which $\boldsymbol{x}_i \neq \boldsymbol{y}_i$, $i \in [1, m]$.*

In simple terms, the Hamming distance measures the minimum number of substitutions to change one vector into the other, or the minimum number of errors that could have transformed one vector into the other.

Remark 1.2. *If $q = 2$ Definition 1.7 can be seen as the number of set terms in the binary vector obtained as the boolean eXclusive OR between two vectors, i.e. $\boldsymbol{x} \oplus \boldsymbol{y}$.*

Definition 1.8 (Hamming Weight). *Given a vector $\boldsymbol{x} \in \mathbb{F}_q^m$ the Hamming weight $weight(\boldsymbol{x})$ is defined as $weight(\boldsymbol{x}) = dist(\boldsymbol{x}, 0)$.*

The Hamming weight is the number of nonzero components of $\mathbf{x}$.

Definition 1.9 (Code Distance). *The code distance is the distance d, such that $d := min\{dist(\boldsymbol{c}_1, \boldsymbol{c}_2) | \boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathcal{C},\ \boldsymbol{c}_1 \neq \boldsymbol{c}_2\}$ for each codeword pair of $\mathcal{C}$.*

This leads to the widely spread notation of $[n, k, d]$-code. From a theoretical point of view, Definition 1.9 is crucial as it provides a measure of how resistant to errors a code is. With the code $[n, k, d]$ we are able to detect up to $d-1$ errors before a word is indistinguishable from another and correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors deterministically. If an error has a Hamming weight greater than $\lfloor \frac{d-1}{2} \rfloor$, multiple codewords may correspond to the corrupted word, leaving it up to the receiver to decide which is the correct one. In the following chapters we will focus only on the subset of linear code with a maximum error correction capability of $\lfloor \frac{d-1}{2} \rfloor$.

So far nothing has been said about how similar two codes are. Also in this case the distance plays a key role.

Definition 1.10 (Code Equivalence). *Two linear codes $[n, k, d]$, $\mathcal{C}_1$ and $\mathcal{C}_2$ are equivalent if and only if there exists a bijective mapping function $f : \mathbb{F}_q^n \to \mathbb{F}_q^n$ such that $\forall \boldsymbol{c}_{1,1}, \boldsymbol{c}_{1,2} \in \mathcal{C}_1$ there exist mappings $f(\boldsymbol{c}_{1,1}), f(\boldsymbol{c}_{1,2}) \in \mathcal{C}_2$ with $dist(\boldsymbol{c}_{1,1}, \boldsymbol{c}_{1,2}) = dist(f(\boldsymbol{c}_{1,1}), f(\boldsymbol{c}_{1,2}))$.*

Note that the $\mathbf{m} \to \mathbf{c}$ mapping between message space and codeword space is not the same for two equivalent codes, the notion of equivalence is based on preserving the code distance between two codes.

Equivalent codes have some interesting properties:

Lemma 1.1. *Given a linear code $\mathcal{C}_1$ with a generator matrix $G_1 \in \mathbb{F}_q^{k\times n}$ and an equivalent linear code $\mathcal{C}_2$ with $G_2 \in \mathbb{F}_q^{k\times n}$ the following proposition are true:*

- *$\mathcal{C}_1$ is a linear code $[n, k, d] \Leftrightarrow \mathcal{C}_2$ is a linear code $[n, k, d]$.*
- *G_2 can be generated from G_1 and vice-versa using basic matrix operations.*

It is possible to generate a new equivalent code using only basic linear algebra operations: $G_2 = AG_1M$ where $A \in \mathbb{F}_q^{k\times k}$ and $M \in \mathbb{F}_q^{n\times n}$ is a monomial matrix.

Remark 1.3. *A monomial matrix is a generalized case of a permutation matrix. Given the identity matrix I_n the permutation matrix $P \in \mathbb{F}_q^{n\times n}$ is derived by applying a series of columns permutation to I_n. The monomial matrix $M \in \mathbb{F}_q^{n\times n}$ is directly derived from P by multiplying the columns of P by a scalars $a_1, ..., a_n \in \mathbb{F}_q$ with $a_i \neq 0\ \forall i$.*

A relevant application of the equivalence property is to highlight the difference between Information and Redundancy sets, in order to do this some mathematical transformations need to be applied.

Theorem 1.1. *Every linear code $\mathcal{C}_1$ with a generator matrix $G_1 \in \mathbb{F}_q^{k\times n}$ has an equivalent code $\mathcal{C}_2$ with a generator matrix $G_2 = [I_k|Q]$, where $Q \in \mathbb{F}_q^{k\times(n-k)}$. A generator matrix in this form is called systematic.*

It is easy to see how the first k columns of G_2 represent the information set (Definition 1.5) which encodes the original message $\mathbf{m}$ in the code $\mathcal{C}$. The remaining $n - k$ columns of the matrix form the redundancy set. This way, to recover the original message during decoding it is sufficient to read the first k symbols of the received message and use the remaining $n - k$ to correct possible errors.

It is useful to show a different way to define a linear code via a parity check matrix.

Definition 1.11 (Parity Check Matrix)**.** *Given a linear code $\mathcal{C}$ with generator matrix $G \in \mathbb{F}_q^{k\times n}$, any matrix $H \in \mathbb{F}_q^{(n-k)\times n}$ with $HG^\top = 0_{(n-k)\times k}$ and $rank(H) = n - k$ is called parity check matrix of the code $\mathcal{C}$.*

Remark 1.4. *As it's been done in Theorem 1.1, given a generator matrix in the systematic form $G = [I_k|Q]$ with $Q \in \mathbb{F}_q^{k\times(n-k)}$ we can express $H = [-Q^\top|I_{n-k}]$ with $Q^\top \in \mathbb{F}_q^{(n-k)\times k}$ which is said to be a parity check matrix in systematic form.*

The parity check matrix lead us to an alternative formalization of linear code: given that $H\mathbf{c} = HG^\top\mathbf{m} = 0$ we can restate the usual definition as $\mathcal{C} := \{\mathbf{c} \in \mathbb{F}_q^n | H\mathbf{c} = 0, H \in \mathbb{F}_q^{(n-k)\times n}, 0_{n-k}\}$.

Up to this point we have seen how a linear code is defined and some of its properties. However, the aim of this thesis is to show the practical applications of this family of codes. Assume that $\mathbf{y} \in \mathbb{F}_q^n$ is a message afflicted by an error $\mathbf{e}$ to be decoded, i.e. $\mathbf{y} = \mathbf{e} + \mathbf{c}$. To check if $\mathbf{y}$ is error-free we can use the parity check matrix: $H\mathbf{y} = H(\mathbf{e}+\mathbf{c}) = H\mathbf{e}+H\mathbf{c} = 0$. If this holds true, the error component is null.
Generally, however, the resulting vector is $\mathbf{s} = H\mathbf{y}$ which is called syndrome of the vector $\mathbf{y}$.

Definition 1.12 (Syndrome). *Given a parity check matrix $H \in \mathbb{F}_q^{(n-k)\times n}$ of a code $\mathcal{C}$ and a vector $\boldsymbol{y} \in \mathbb{F}_q^n$, we call $\boldsymbol{s} = H\boldsymbol{y}$ with $\boldsymbol{s} \in \mathbb{F}_q^{(n-k)}$ syndrome of the vector $\boldsymbol{y}$.*

In the next section we will show how the syndrome is involved in decoding.

1.2. Decoding Strategies

So far we have discussed about linear codes as a pure mathematical concept. Generally speaking, the purpose of a linear code is to allow information to be sent through a noisy channel. Here is where decoding enters the game: given a received message $\mathbf{y}$ the decoding method tries to find the right codeword $\mathbf{c}$ and then retrieve the initial message $\mathbf{m}$. In this section we will see how linear codes can be applied to cryptography.
In order to understand how the mathematical concepts just presented can take part in the art of information hiding, first of all it is necessary to understand the true complexity behind decoding. The following are the main decoding strategies, from the most common to the most complex and interesting.

1.2.1. Syndrome Decoding

The first approach we show is the so called list based decoding. Suppose that the received message $\mathbf{y}$ is affected by an error $\mathbf{e}$ of a certain weight $weight(\mathbf{e}) = t$. By recalling Definition 1.12 of the syndrome $\mathbf{s} = H\mathbf{y} = H(\mathbf{e} + \mathbf{c}) = H\mathbf{e} + H\mathbf{c} = H\mathbf{e}$, it can be seen that the syndrome of the message $\mathbf{y}$ is associated with the error vector. We can thus use the syndrome directly to correct errors in messages.
Usually this is done employing a lookup table $\mathbf{T}$, which maps all the possible syndrome $\mathbf{s}$ to errors $\mathbf{e}$ with $weight(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$. Note that the table has a size of $2^{(n-k)} - 1$ and can be precomputed.
The goal is to retrieve the correct codeword $\mathbf{c} = \mathbf{y} - \mathbf{T}(\mathbf{s}) = \mathbf{y} - \mathbf{e}$.

Definition 1.13 (Syndrome Decoding Problem (SDP)). *Given a parity check matrix H, a message $\boldsymbol{y} = \boldsymbol{e} + \boldsymbol{c}$ and a syndrome $\boldsymbol{s} = H\boldsymbol{y} = H\boldsymbol{e}$ find the error vector $\boldsymbol{e}$ such that $weight(\boldsymbol{e}) = t$.*

In code-based cryptography there is a problem strictly related to this. Suppose we have a code $\mathcal{C}$, the goal of this problem is to find a codeword $\mathbf{c}$ with the smallest possible positive weight w.

Definition 1.14 (Codeword Finding Problem (CFP)). *Given a parity check matrix H and a weight w, find a codeword $\boldsymbol{c}$ with $weight(\boldsymbol{c}) \leq w$ and $H\boldsymbol{c} = 0$.*

Notice how the codeword finding problem is reduced to finding a good sum of the columns of the parity check matrix since the only values of H which are involved in the sum are those in correspondence to the no-null positions of the vector c i.e. the 1s of the codeword.

The last point to be clarify is how to look for the right sum of the columns. The most naive approach is, as often happens in computer science, the exhaustive search.
A pseudocode of this type of search is reported below.

Algorithm 1 Syndrome decoding - exhaustive search

Input: $\mathbf{s}$: $n-k$ bit long syndrome
H: $(n-k) \times n$ parity check matrix
t: weight of an error vector
Output: $\mathbf{e}$: n-bit error vector s.t. $H\mathbf{e} = \mathbf{s}$ with $weight(\mathbf{e}) = t$

```
e ← [0_n]
for j ← 1 to (n choose t) do
  // J is a set of t distinct integers in {0, ..., n − 1}
  J ← IntegerToCombination(j)
  if Σ_{i∈J} h_i = s then
    for all i ∈ J do
      e ← e + [0_i|1|0_{n−1−i}]
      return e
```

Algorithm 1 complexity is the following:

$$\begin{aligned} C_{ES}(n,k,t) &= \binom{n}{t}(C_{IntToComb} + t(n-k)) \\ C_{IntToComb} &= \mathcal{O}((2n-t)\left(\log\binom{n}{t}^2\right)) \\ S_{ES}(n,k) &= \mathcal{O}(n^2 - nk)) \end{aligned} \tag{1.1}$$

where C_{ES} is the time complexity and S_{ES} is the spatial complexity. The function IntegerToCombination() maps a given integer into a set of t distinct indexes in the interval $\{0,\ldots,n-1\}$. These will be used as selectors for the columns of the H matrix.

Algorithm 1 is the simplest way to tackle the problem, in the next sections we see how it can be optimised.

1.2.2. Information Set Decoding

In this section we will extensively use the concept of Information set (Definition 1.5) explaining how it plays a key role in speeding up the decoding.
By applying the notion of Information set to the two matrices G and H, some interesting observations can be done.

Definition 1.15 (Information Set Generator Matrix). *Given a linear code $[n,k]$ with a generator matrix $G \in \mathbb{F}_q^{k\times n}$, a set $\mathcal{I} \subseteq \{1,...,n\}$ of size k is an information set if and only if $rank(G_{\mathcal{I}}) = |\mathcal{I}| = k$.*

The matrix $G_{\mathcal{I}} \in \mathbb{F}_q^{k\times k}$ is the portion of G that encode any message $\mathbf{m} \in \mathbb{F}_q^k$ in its correspondence codeword or in other word $G_{\mathcal{I}}$ represents any possible non-redundant linear transformation on $\mathbf{m}$. The information set applies to H in the same way.

Definition 1.16 (Information Set Parity Check Matrix). *Given a linear code $[n,k]$ with a parity check matrix $H \in \mathbb{F}_q^{(n-k)\times n}$, a set $\mathcal{I} \subseteq \{1,...,n\}$ of size k is an information set if and only if $rank(H_{\mathcal{I}^*}) = |\mathcal{I}^*| = n-k$, with $\mathcal{I}^* := \{1,...,n\}\backslash\mathcal{I}$.*

Let's see how we can take advantage of an information set. Given a matrix G and a received message $\mathbf{y} = \mathbf{c} + \mathbf{e}$, once an information set $G_{\mathcal{I}}$ is defined we can write

$$\mathbf{y}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} = (\mathbf{c}+\mathbf{e})_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} = \mathbf{c}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} + \mathbf{e}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} \tag{1.2}$$

Assuming $\mathbf{e}_{\mathcal{I}} = 0$ and recalling that $\mathbf{c} = G^\top \mathbf{m}$ we derive the original message $\mathbf{m}$

$$\mathbf{y}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} = \mathbf{c}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} = (\mathbf{m}^\top G)_{\mathcal{I}} G_{\mathcal{I}}^{-1} = \mathbf{m}^\top \tag{1.3}$$

On the other hand if $\mathbf{e}_{\mathcal{I}} \neq 0$, by multiplying every term by G we obtain

$$\begin{aligned} \mathbf{y}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} G &= (\mathbf{c} + \mathbf{e})_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} G = \mathbf{c}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} G + \mathbf{e}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} G \\ &= (\mathbf{m}^\top G)_{\mathcal{I}} G_{\mathcal{I}}^{-1} G + \mathbf{e}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} G \\ &= \mathbf{m}^\top G + \mathbf{e}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} G = \mathbf{c}^\top + \mathbf{c}_e^\top \end{aligned} \tag{1.4}$$

where $\mathbf{c}_{\mathbf{e}}^\top = \mathbf{e}_{\mathcal{I}}^\top G_{\mathcal{I}}^{-1} G$.

The idea behind information set decoding, or ISD, is to guess an information set for which the indexed part of the error is null or for which we know $\mathbf{e}_{\mathcal{I}}$.

Due to its smaller size, we can use the parity matrix H to speed up computation. Before doing this, some transformations need to be applied to H. After selecting uniformly an information set $\mathcal{I}$ we use a permutation matrix $P \in \mathbb{F}_q^{n \times n}$ (Remark 1.3) to shift to the right side the $n-k$ columns of H indexed by $\mathcal{I}^* := \{1, ..., n\} \backslash \mathcal{I}$, obtaining

$$\tilde{H} = HP \tag{1.5}$$

We apply elementary row operation to $\tilde{H}$ to bring it into systematic form (Remark 1.4). All the row transformations are grouped in a matrix R.

$$\hat{H} = R\tilde{H} = RHP = [V | I_{(n-k)}] \tag{1.6}$$

where $V \in \mathbb{F}_q^{(n-k) \times k}$. Notice that $\hat{H} \in \mathbb{F}_q^{(n-k) \times n}$.

We need to also define a new error vector $\hat{\mathbf{e}} = P^{-1}\mathbf{e} = P^\top \mathbf{e}$. The effect of the inverse of the permutation matrix on $\mathbf{e}$ is to group all the $n-k$ elements indexed by $\mathcal{I}^*$ at the bottom of $\hat{\mathbf{e}}$. What we now have is

$$\hat{H}\hat{\mathbf{e}} = RHPP^\top \mathbf{e} = RHPP^{-1}\mathbf{e} = RH\mathbf{e} = \hat{\mathbf{s}} \tag{1.7}$$

Solving $\hat{H}\hat{\mathbf{e}} = \hat{\mathbf{s}}$ is equal to solve $R\mathbf{s} = \hat{\mathbf{s}}$.

To get a clearer picture, Figure 1.2 shows the weights distribution and the division of $\hat{H}, \hat{\mathbf{e}}$ and $\hat{\mathbf{s}}$.

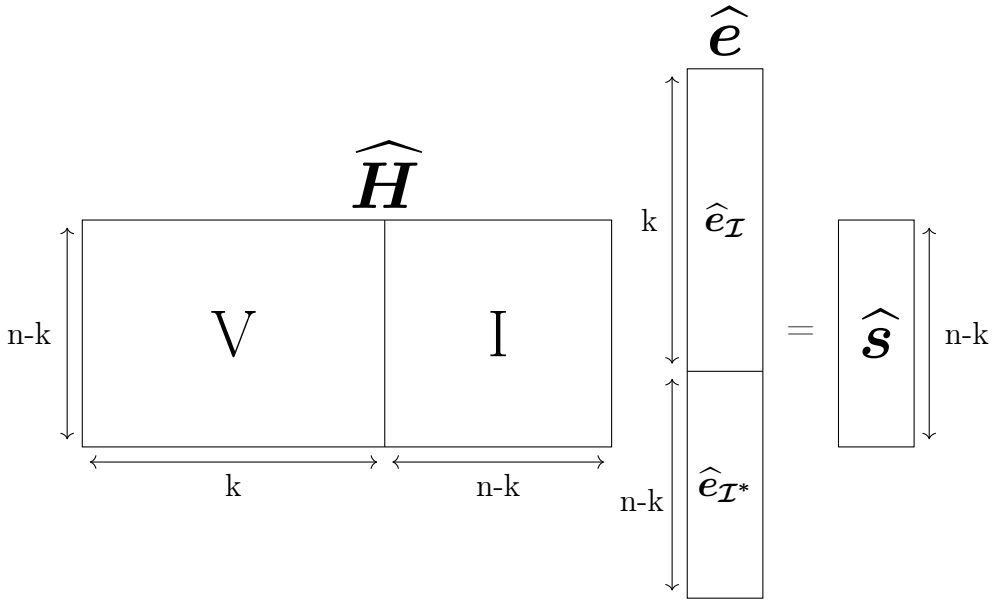


Figure 1.2: Structure of $\widehat{H}\widehat{e} = \widehat{s}$

This is reflected in Equation (1.7)

$$\begin{aligned} \hat{H}\hat{\mathbf{e}} &= [V|I_{(n-k)}][\hat{\mathbf{e}}_{\mathcal{I}}^{\top}|\hat{\mathbf{e}}_{\mathcal{I}^*}^{\top}]^{\top} \\ &= V\hat{\mathbf{e}}_{\mathcal{I}} + I\hat{\mathbf{e}}_{\mathcal{I}^*} \\ &= V\hat{\mathbf{e}}_{\mathcal{I}} + \hat{\mathbf{e}}_{\mathcal{I}^*} = \hat{s} \end{aligned} \tag{1.8}$$

Additionally, we can derive $\hat{\mathbf{e}}_{\mathcal{I}^*}$ as $\hat{\mathbf{e}}_{\mathcal{I}^*} = \hat{\mathbf{s}} - V\hat{\mathbf{e}}_{\mathcal{I}}$.

To sum up what we have done so far: we started from the syndrome decoding problem (Definition 1.13) and rephrased it in terms of $\hat{H}$, $\hat{e}$ and $\hat{s}$. The goal is still to find a correct error vector $\mathbf{e}$ of weight t, but now we can directly guess a weight p on the $\hat{\mathbf{e}}_{\mathcal{I}}$ bits, compute $\hat{\mathbf{e}}_{\mathcal{I}^*}$ and check our hypothesis

$$weight(\hat{\mathbf{e}}_{\mathcal{I}^*}) = t - weight(\hat{\mathbf{e}}_{\mathcal{I}}) = t - p \tag{1.9}$$

If our guess is correct we compute $\hat{\mathbf{e}}$ with $weight(\hat{\mathbf{e}}) = t$ through the concatenation of $\hat{\mathbf{e}}_{\mathcal{I}}$ and $\hat{\mathbf{e}}_{\mathcal{I}^*}$. To derive the initial error $\mathbf{e}$ we simply apply the the permutation matrix in the reverse order $\mathbf{e} = P\hat{\mathbf{e}}$. However if our hypothesis is wrong we need to try another weight p or to choose a different information set $\mathcal{I}$.

Almost all ISD algorithms share as a common structure the random choice of the information set or the algebraic transformation on the matrix H, therefore we group these common steps in the sub procedure: ISextract.

Algorithm 2 ISextract

Input: $\mathbf{s}$: $(n-k)$-bit long syndrome
H: $(n-k) \times n$ parity check matrix
Output: $\hat{\mathbf{s}}$: $(n-k)$-bit long new syndrome
P: $n \times n$ permutation matrix
V: $(n-k) \times k$ binary matrix

1: **repeat**
2: $P \leftarrow$ RandomPermutationGen(n) // random choice of $\mathcal{I}$
3: $\tilde{H} \leftarrow HP$
4: $(U, [V|W]) \leftarrow$ RedRowEchelonForm($\tilde{H}$) // reduction of $\tilde{H}$ into systematic form
5: **until** $W \neq I^{n-k}$
6: $\hat{\mathbf{s}} \leftarrow U\mathbf{s}$
7: **return** $(\hat{\mathbf{s}}, P, [V|W])$ =0

Algorithm 2 complexity is the following:

$$\begin{aligned} C_{IS}(n,k) &= \frac{1}{\sum_{i=1}^{n-k}(1-2^{-i})} C_{RREF}(n,k) + (n-k)^2 \\ C_{RREF}(n,k) &= \mathcal{O}\left(\frac{n(n-k)^2}{2} + \frac{n(n-k)}{2} - \frac{(n-k)^3}{6} + (n-k)^2 + \frac{(n-k)}{6} - 1\right) \\ S_{IS}(n,k) &= \mathcal{O}(n(n-k)) \end{aligned} \tag{1.10}$$

where C_{IS} is the time complexity and S_{IS} is the spatial complexity.

Algorithm 2 takes as input the syndrome $\mathbf{s}$ and the parity check matrix H and it returns the new syndrome $\hat{\mathbf{s}}$, the permutation matrix P and the permuted parity check matrix in systematic form $\hat{H}$ i.e. $[V|W]$. The main loop (line 1-5) is composed by three steps: the procedure RandomPermutationGen() (line 2), used to generate a random permutation matrix P, the computation of the permuted parity check matrix $\tilde{H}$ (line 3) and the reduction to systematic form of the latter through the procedure RedRowEchelonForm().

At this point all that is needed to conclude the discussion is how to compute the error vector. Different algorithms are discussed.

1.2.3. Prange algorithm

The first algorithm for ISD was proposed by Prange in 1962 [29], it is considered the starting point for the literature of this field. Prange's algorithm is based on a simple observation: if $\hat{\mathbf{e}}_{\mathcal{I}} = 0$ then from $\hat{\mathbf{s}}$ we can derive all the entries of $\hat{\mathbf{e}}_{\mathcal{I}^*}$, the weight

distribution is shown in Figure 1.3.

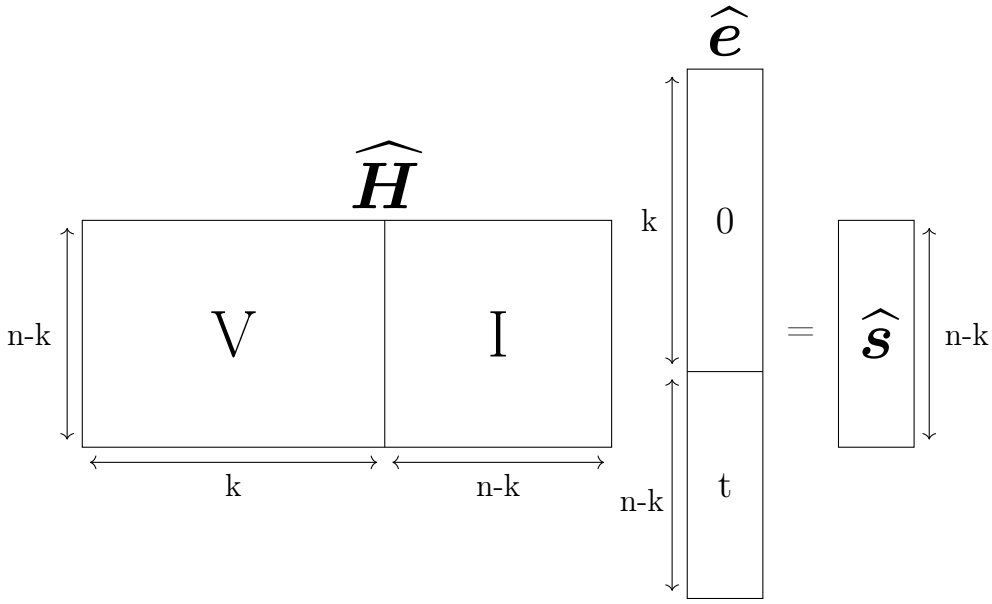


Figure 1.3: Weight distribution for Prange's algorithm

From equation 1.8 we have that $\hat{\mathbf{e}}_{\mathcal{I}^*} = \hat{\mathbf{s}}$ and $weight(\hat{\mathbf{e}}) = weight(\hat{\mathbf{e}}_{\mathcal{I}^*}) = weight(\hat{\mathbf{s}}) = t$. By recovering the initial error $\mathbf{e}$ we can solve the syndrome decoding problem 1.13. The idea is to guess the right IS by randomly selecting a permutation on H and verifying the weight of the resulting syndrome $\hat{\mathbf{s}}$. The pseudo code is described in algorithm 3

Algorithm 3 Syndrome decoding - Prange

Input: $\mathbf{s}$: $n-k$ bit long syndrome
H: $(n-k) \times n$ parity check matrix
t : the weight of the error vector

Output: $\mathbf{e}$: n-bit error vector s.t. $H\mathbf{e} = \mathbf{s}$ with $weight(\mathbf{e}) = t$
$\hat{\mathbf{e}}$: $n-k$ bit new syndrome
P : $n \times n$ permutation matrix
V : $(n-k) \times k$ binary matrix

```
repeat
    (ŝ, P, [V|W]) ← ISextract(s, H)
until weight(ŝ) = t
ê ← [0_k|ŝ]
return Pê
```

Algorithm 3 has the following complexity:

$$C_{PR}(n,k,t) = \frac{\binom{n}{t}}{\binom{n-k}{t}}(C_{IS} + \mathcal{O}(n)) \tag{1.11}$$

where C_{PR} is the time complexity and C_{IS} is defined in Equation 1.10.

1.2.4. Lee-Brickell algorithm

An improvement of Prange's idea is done by Lee and Brickell in [18]. It is based on the fact that it is unlikely that the whole weight of $\hat{\mathbf{e}}$ is concentrated in the $n-k$ positions indexed by $\mathcal{I}^*$. The authors allow exactly p non zero entries in the first k positions of the permuted error. Notice how Prange is a special case of Lee-Brickell with $p = 0$, the weight distribution is reported in Figure 1.4.

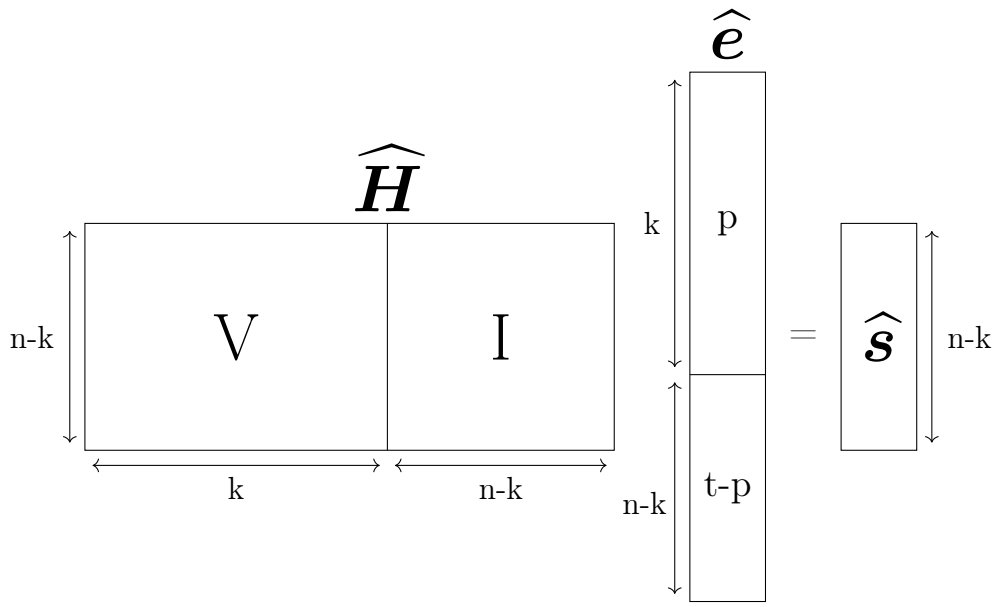


Figure 1.4: Weight distribution for Lee-Brickell's algorithm

From the equation 1.8 we derive $\hat{\mathbf{e}}_{\mathcal{I}^*} = \hat{\mathbf{s}} - V\hat{\mathbf{e}}_{\mathcal{I}}$ and $weight(\hat{\mathbf{e}}_{\mathcal{I}^*}) = weight(\hat{\mathbf{s}} - V\hat{\mathbf{e}}_{\mathcal{I}}) = t-p$. The structure of this algorithm is similar to that of Prange's, with an additional procedure for the choice of how to distribute the weight p over k indexes namely $\binom{k}{p}$. The pseudo code is described in algorithm 4. Specifically, we need to try all the possible combinations of the weights (line 4) and at any iteration we map this choice with a set of columns of V (line 5).

Algorithm 4 Syndrome decoding - Lee-Brickell

Input: $\mathbf{s}$: $n-k$ bit long syndrome
H: $(n-k) \times n$ parity check matrix
t : the weight of the error vector
p : the weight of the first k bit of $\hat{\mathbf{e}}$ with $0 \leq p \leq t$

Output: $\mathbf{e}$: n-bit error vector s.t. $H\mathbf{e} = \mathbf{s}$ with $weight(\mathbf{e}) = t$
$\hat{\mathbf{s}}$: $n-k$ bit new syndrome
P : $n \times n$ permutation matrix
V : $(n-k) \times k$ binary matrix

```
repeat
    (ŝ, P, [V|W]) ← ISextract(s, H)
    for j ← 1 to (k choose p) do
        J ← IntegerToCombination(j)
        ê_I* ← ŝ − Σ_{i∈J} v_i
        if weight(ê_I*) = t − p then
            ê ← [0|ê_I*]
            for i ∈ J do
                ê ← ê + [0_i|1|0_{n−1−i}]
            end for
            return Pê
```

Algorithm 4 has the following complexity:

$$
\begin{aligned}
C_{LB}(n,k,t,p) &= \frac{\binom{n}{t}}{\binom{k}{p}\binom{n-k}{t-p}} \left(C_{IS} + \binom{k}{p}(C_{IntToComb} + p(n-k)) \right) \\
C_{IntToComb} &= \mathcal{O}((2k-p)(\log \binom{k}{p})^2) \\
S_{LB}(n,k,t,p) &= \mathcal{O}(n(n-k) + (n-k)^2 + (p-1)(n-k) + n - k)
\end{aligned}
\tag{1.12}
$$

where C_{LB} is the time complexity and S_{LB} is the spatial complexity.

1.2.5. Leon algorithm

A further improvement to Leon-Brickell's idea is made by Leon in [19]. It is now assumed that the contribution of the first l bit of the syndrome $\hat{\mathbf{s}}$ comes from the columns of V. This hypothesis implies that in the error $\hat{\mathbf{e}}$ is present a slide of l zeros in the first bits indexed by $\mathcal{I}^*$ that cancel the contribution of the first l columns of the matrix I.
To give a clearer view, the matrices division is shown in Figure 1.5.

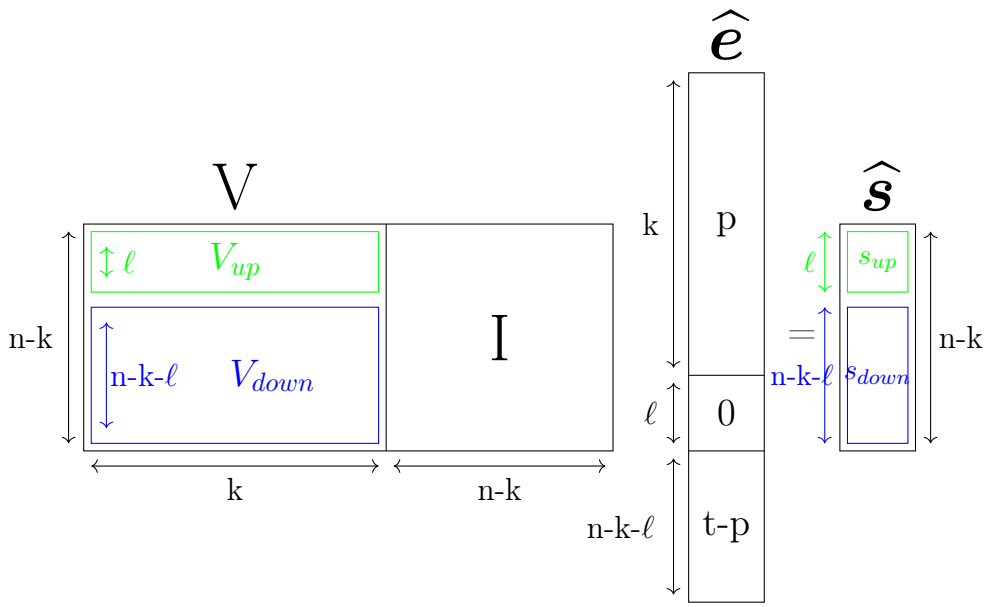


Figure 1.5: Weight distribution for Leon's algorithm

This type of assumption reduces the probability of success because we are excluding all possible solutions that do not have a slide of l zeros in the permuted error. On the other hand we have the possibility to precompute the sum of the first l rows of the matrix composed by the p columns of V instead of the sum of the entire columns.

1.2.6. Stern algorithm

Developing the idea of precomputing some sub-part of the problem we present Stern' algorithm [36]. Starting from Leon's idea it employs a meet-in-the-middle strategy to find which sub-part of the matrix V forms the first l bits of the syndrome.
The idea is to split $\hat{\mathbf{e}}_{\mathcal{I}}$ in two intervals of $\frac{k}{2}$ bit, each with a weight of $\frac{p}{2}$. In Figure 1.6 is reported the weight distribution.

Solving the equation:

$$V_{up}\hat{\mathbf{e}}_{\mathcal{I}} = \hat{\mathbf{s}}_{up} \Leftrightarrow \hat{\mathbf{s}}_{up} = V_{up1}\hat{\mathbf{e}}_{\mathcal{I},up1} + V_{up2}\hat{\mathbf{e}}_{\mathcal{I},up2} \tag{1.13}$$

we derive $V_{up2}\hat{\mathbf{e}}_{\mathcal{I},up2} = \hat{\mathbf{s}}_{up} + V_{up1}\hat{\mathbf{e}}_{\mathcal{I},up1}$. Using the result we have just found we can precompute $\hat{\mathbf{s}}_{up} + V_{up1}\hat{\mathbf{e}}_{\mathcal{I},up1}$ for all possible $\binom{k/2}{p/2}$ combinations and store the values in a list θ, computing the remaining part $V_{up2}\hat{\mathbf{e}}_{\mathcal{I},up2}$ and check if the result is present in θ. If a match is found we have a candidate for $\hat{\mathbf{e}}_{\mathcal{I}}$ and we can proceed to the next checks on $\hat{\mathbf{e}}_{\mathcal{I}^*}$.

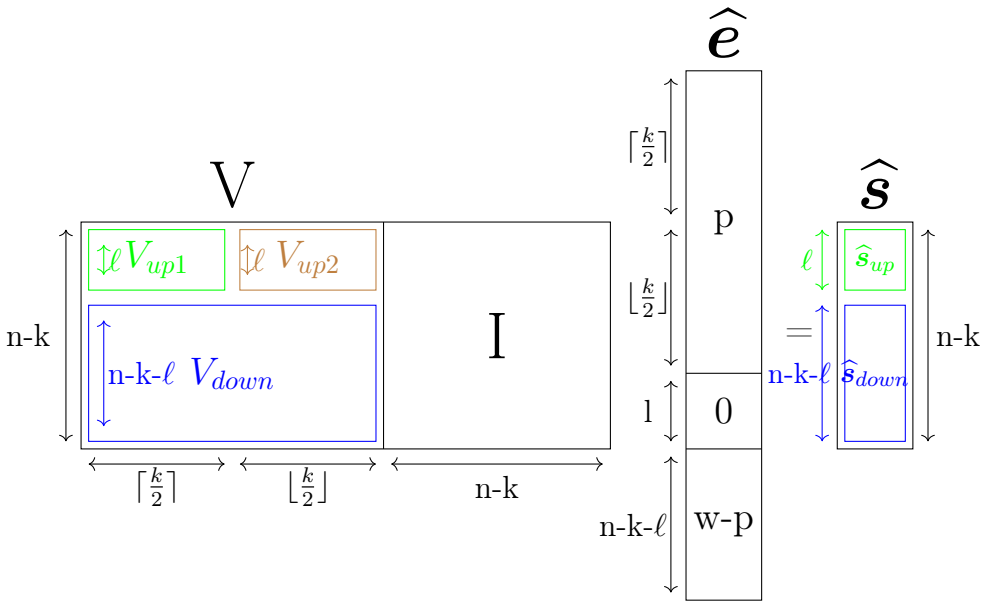


Figure 1.6: Weight distribution for Stern's algorithm

1.2.7. Finiasz-Sendrier algorithm

Finiasz and Sendrier in [13], differently from Leon and Stern, dismiss the presence of a l long strip of zeros in the permuted error $\hat{\mathbf{e}}_{\mathcal{I}^*,up} = 0^l$. They allow for the distribution of the weight p also in the region $\hat{\mathbf{e}}_{\mathcal{I}^*,up}$. The idea behind it is now to split $\hat{\mathbf{e}}_{\mathcal{I}}$ in two interval of $\frac{k+l}{2}$ bit of weight $\frac{p}{2}$. The weight distribution for the new matrix division is reported in Figure 1.7.

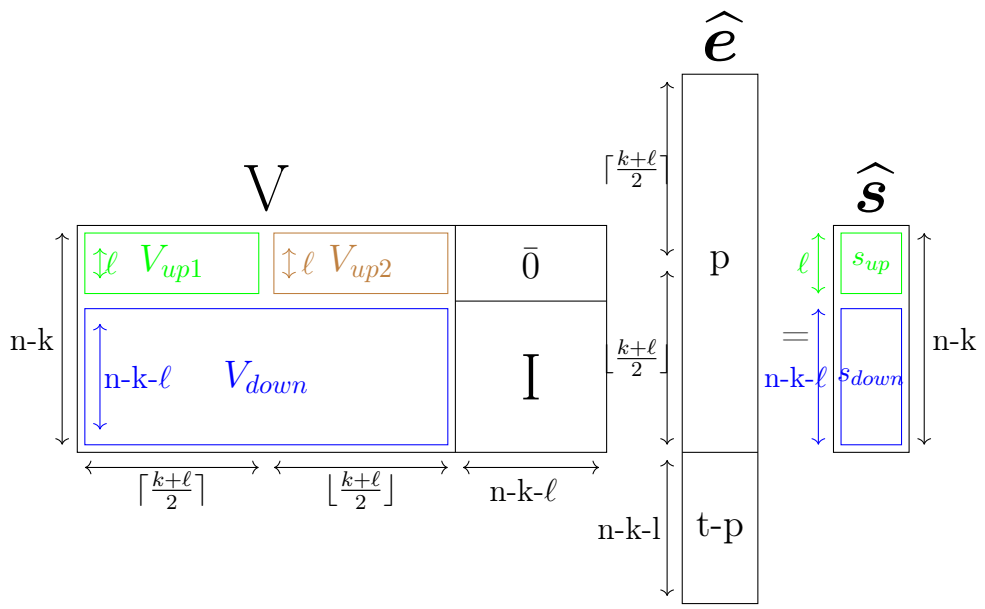


Figure 1.7: Weight distribution for Finiasz-Sendrier's algorithm

We follow the same principle of the Stern's Equation 1.13 to create the list θ and checking for a collision between all $\binom{(k+l)/2}{p/2}$ values of $V_{up2}\hat{\mathbf{e}}_{\mathcal{I},up2}$ and $\hat{\mathbf{s}}_{up} + V_{up1}\hat{\mathbf{e}}_{\mathcal{I},up1}$. However with this approach we have to slightly modify the ISextract procedure. As a consequence of having the division of Figure 1.7 we admit the possibility of selecting $k + l$ columns of V hence the need to have a modified version of the aforementioned procedure that we will call ISextract$_{FS}$. The only difference between the two is that in the modified one we need to stop earlier the RedRowEchelon sub-procedure, obtaining $(n - k - l) \times (n - k - l)$ identity matrix in the lower rightmost portion of $\hat{H}$. Algorithm 5 shows the pseudo-code of Finiasz-Sendrier ISD.

Algorithm 5 Syndrome decoding - Finiasz-Sendrier

Input: s: $n-k$ bit long syndrome
H: $(n-k) \times n$ parity check matrix
t : the weight of the error vector
p : the weight of the first k bit of $\hat{e}$ with $0 \leq p \leq t$
l : input parameter $0 \leq l \leq n-k-t+p$

Output: e : n-bit error vector s.t. $He = s$ with $weight(e) = t$
$\hat{s}$: $n-k$ bit new syndrome
P : $n \times n$ permutation matrix
V : $(n-k) \times k$ binary matrix
θ : list containing $\hat{s}_{up} - V_{up1}\hat{e}_{\mathcal{I},up1}$ result, initially is a empty list

```
repeat
    (ŝ, P, [V|W]) ← ISextract_FS(s, H)
    for l_1 ← 1 to C((k+l)/2, p/2) do
        J_1 ← IntegerToCombination(l_1))
        θ ← ŝ_up − Σ_{l_1∈J_1} v_{up1_{l_1}}
    end for
    for l_2 ← 1 to C((k+l)/2, p/2) do
        J_2 ← IntegerToCombination(l_2))
        if Collision(Σ_{l_2∈J_2} v_{up2_{l_2}}, θ) then
            ê_{I*} ← ŝ − Σ_{l_1∈J_1} v_{up1_{l_1}} − Σ_{l_2∈J_2} v_{up2_{l_2}}
            if weight(ê_{I*}) = t − p then
                ê ← [0_{k+l} | ê_{I*}]
                for l_1 ∈ J_1 do
                    ê ← ê + [0_{l_1} | 1 | 0_{n−1−l_1}]
                end for
                for l_2 ∈ J_2 do
                    ê ← ê + [0_{l_2} | 1 | 0_{n−1−l_2}]
                end for
                return Pê
```

Algorithm 5 has the following complexity:

$$
\begin{aligned}
C_{FS}(n,k,t,p,l) &= \frac{\binom{n}{t}}{\binom{n-k-l}{t-p}\binom{(k+l)/2}{p/2}^2}\Big(C_{IS-FS}(n,k,l) + \\
&\binom{(k+l)/2}{p/2}\left(2C_{IntToComb} + C_{Collision} + \frac{\binom{(k+l)/2}{p/2}}{2^l}p(n-k-l)\right)\Big) + p \\
S_{FS}(n,k,t,p,l) &= \mathcal{O}(n(n-k) + \left(\binom{(k+l)/2}{p/2}(\frac{p}{2}\log_2(\frac{k+l}{2}) + l)\right))
\end{aligned}
\tag{1.14}
$$

where C_{FS} is the time complexity, S_{FS} is the spatial complexity, C_{IS-FS} is the complexity of the ISextract$_{FS}$ operation on line 1 which has the same complexity as the original ISextract (1.10) but with a new $C_{Par-RREF}$ equal to

$$\begin{aligned} C_{Par-RREF} = \mathcal{O}\Big(& -\frac{l^2 n}{4} - \frac{l^2(n-k)}{4} - \frac{ln(n-k)}{2} - \frac{ln}{4} \\ & - \frac{l(n-k)^2}{2} - \frac{l(n-k)}{4} - \frac{3n(n-k)^2}{4} + \frac{n(n-k)}{4} \\ & - \frac{n}{2} + \frac{3(n-k)^3}{4} - \frac{3(n-k)^2}{4} - \frac{(n-k)}{2} \Big) \end{aligned} \tag{1.15}$$

The relevant part is the complexity of the Collision operation in line 9, $C_{Collision}$, which is the part of the procedure we will try to optimize via our novel quantum solution explained in Chapter 4. Its classical complexity depends on the method of computation used: either we perform a Binary-Range search or we employ a lookup table. The complexities are

$$\begin{aligned} C_{Collision,BR} &= \mathcal{O}\left(2\log_2 \binom{(k+l)/2}{p/2}\right) \\ C_{Collision,LUT} &= \mathcal{O}\left(\frac{\binom{(k+l)/2}{p/2}}{2^l}\right) \end{aligned} \tag{1.16}$$

Finiasz and Sendrier's algorithm closes the section dedicated to information set decoding. Again, this will be the starting point for the our attack based on a novel quantum algorithm.

1.3. Code based cryptosystems

Up to present day, the increase of available computational power has led to a constant improvement and evolution of cryptographic systems. The computational security of modern asymmetric cryptography is rooted in complex mathematical problems such as factoring the product of two primes or solving the discrete logarithm problem. Some of the most widely used public key algorithms such as RSA and Diffie-Hellman are based on these mathematical problems [9, 31]. The arrival of quantum computing in the early '90 and the consequent exponential growth in computing power has been a turning point for many sectors especially in cryptography. In 1994 Peter Shor proved the superiority of the new paradigm by providing in [35] a quantum algorithm able to solve the factorization problem in polynomial time, achieving an almost exponential improvement over any known classic algorithm. Hence the need to continue the research in this field to make cryptography quantum resistant, linear code based cryptosystems seem to fulfil this task. The computational security of these encryption algorithms rely on the difficulty of finding

a minimum weight codeword in a linear code with a random structure (Definition 1.14) that is proven to be a NP-Hard problem [6]. In this section we present two of the most famous ciphers based on linear code the McEliece and Niederretier cryptosystem showing their general structures and the encryption/decryption procedures.

1.3.1. McEliece

Robert McEliece in [22] was one of the first to present a cipher based on linear codes. Although decoding a linear code have been demonstrated to be NP-Complete in [6] this cryptosystem has never really been adopted, this is due to the length of the keys and a low code rate compared to other algorithms such as RSA. The recent increase in quantum computing capacity has rekindled the interest in this cryptosystem thanks to its apparent quantum resistant property. The first proof of quantum resistance is given by [10] where it is shown how the cipher is immune to Quantum Fourier Sampling, a common procedure underlying many famous quantum algorithms such as Shor and Simon's algorithms. However, estimating the real security of the parameters employed is not an easy task, which is why the search for increasingly effective attacks must be carried on.

To complete the description of the cryptosystem, we first show how the public and private keys are generated in Algorithm 6

Algorithm 6 McEliece Keys generation

Input: n, k, d, q, t: integer value

Output: pk: public key

sk: secrete key

P: $n \times n$ permutation matrix over $\mathbb{F}_q$

A: $k \times k$ matrix of rank k over $\mathbb{F}_q$

G: $k \times n$ generator matrix over $\mathbb{F}_q$

1: $\mathcal{C} \leftarrow$ ChooseLinearCode(n, k, d)
2: $G \leftarrow$ GenMatrix$(\mathcal{C})$
3: $P \leftarrow$ RandPerm()
4: $A \leftarrow$ RandMatrix()
5: $G' = AGP$
6: $pk = (G', n, k, t, q)$
7: $sk = (G, P, A)$
8: **return** (pk, sk) =0

The procedure ChooseLinearCode() at (line 1) is used to choose randomly a linear code $\mathcal{C}$-

$[n, k, d]$ over the finite field $\mathbb{F}_q$ such that an efficient decoding algorithm $\text{Decode}_G()$ exists and it corrects up to $\lfloor \frac{d-1}{2} \rfloor$ errors. The main idea is hide the linear code $\mathcal{C}$ generated by G with an equivalent linear code $\mathcal{C}'$ with generator matrix G'. Thanks to Definition 1.1 using only simple linear algebra operation we derive $G' = AGP$, even if the two code $\mathcal{C}$ and $\mathcal{C}'$ are equivalent the mapping between the message space and the codeword space $\mathbf{m} \to G^\top \mathbf{m}$ is completely different.

The encryption procedure illustrated in Algorithm 7 is the encoding of the plaintext $\mathbf{c} = \mathbf{m}G'$ with the addition of a random error $\mathbf{e}$ of $weight(\mathbf{e}) = t$, to improve the security it is a good practice to choose the highest possible weight for $\mathbf{e}$.

Algorithm 7 McEliece encryption

Input: $\mathbf{m}$: plain text over $\mathbb{F}_q^k$
pk : public key
Output: $\mathbf{y}$: cipher text over $\mathbb{F}_q^n$
$\mathbf{e}$: a random error over $\mathbb{F}_q^n$

1: $\mathbf{c} = \mathbf{m}G'$
2: $\mathbf{e} \leftarrow$ random error with $weight(\mathbf{e}) = t$
3: $\mathbf{y} = \mathbf{c} + \mathbf{e}$
4: **return** $\mathbf{y}$

The decryption procedure shown in Algorithm 8 requires a little more attention, all the matrices operation are explained in detail:

$$\begin{aligned} \mathbf{d} = P\mathbf{y}^\top = P(\mathbf{m}G')^\top + P\mathbf{e}^\top = P(\mathbf{m}AGP)^\top + P\mathbf{e}^\top = \\ PP^\top G^\top A^\top \mathbf{m} + P\mathbf{e}^\top = (AG)^\top \mathbf{m} + P\mathbf{e}^\top \end{aligned} \tag{1.17}$$

Thanks to the remark 1.3 $P^{-1} = P^\top$ and applying a permutation to the error vector does not change the weight.

Algorithm 8 McEliece decryption

Input: $\mathbf{y}$: cipher text over $\mathbb{F}_q^n$
sk : secret key
Output: $\mathbf{m}$: plain text over $\mathbb{F}_q^k$
$\mathbf{e}$: a random error over $\mathbb{F}_q^n$

1: $\mathbf{d} = P\mathbf{y}^\top$
2: $\mathbf{a} = \text{Decode}_G(\mathbf{d}) = A^\top \mathbf{m}$
3: $\mathbf{m} = A^{-1}\mathbf{a}$
4: **return m**

There are two main family of attacks which can be applied to McEliece cryptosystem in order to test its security level, the Structural and the Unstructural/Decoding attacks.

Structural attack. The attacker can attempt to recover the structure of $\mathcal{C}$ and thus find an efficient decoding algorithm Decode_{atk} to break the encryption. This type of attack highly depends on the family of code from which $\mathcal{C}$ is chosen, the original proposal of using Goppa Codes is still immune to this attack.

Unstructural or Decoding attack. The attacker can retrive the plaintext $\mathbf{m}$ from the message $\mathbf{y}$ ignoring the structure of the code just using an algorithm for decoding a linear code like Syndrome decoding or ISD. This attack which is often referred as brute force relies only on the computational power available for the attacker because as shown in this chapter every general decoding algorithm as exponential running time.

To give practical idea on the parameters employed McEliece in his original paper suggest using binary Goppa code, extensively explained in [5], with security parameters sizes of $n = 1024$, $k = 524$ and $t = 50$. Concerning post quantum security the proposed parameter are in the order of $n = 6960$, $k = 5413$ and $t = 119$ suggested in [2]. The search for McEliece cryptosystem parameters that offer the best trade-off between performance and post-quantum security is still open for discussion.

1.3.2. Niederretier

The second cryptosystem based on linear code was introduced by Harald Niederreiter in [24], it follows the structure of the McEliece cryptosystem and it offers the same level of security of the latter. Its main idea consists of using the parity check matrix H in the encryption procedure and the syndrome as chipertext. The procedure of key generation, encryption and decryption are described below.

Algorithm 9 Niederretier Keys generation

Input: n, k, d, q, t: integer value

Output: pk : public key

sk : secrete key

P : $n \times n$ permutation matrix over $\mathbb{F}_q$

A : $k \times k$ matrix of rank k over $\mathbb{F}_q$

H : $(n-k) \times n$ generator matrix over $\mathbb{F}_q$

1: $\mathcal{C} \leftarrow$ ChooseLinearCode(n,k,d)
2: $H \leftarrow$ ParityMatrix($\mathcal{C}$)
3: $P \leftarrow$ RandPerm()
4: $A \leftarrow$ RandMatrix()
5: $H' = AHP$
6: $pk = (H', n, k, t)$
7: $sk = (H, P, A)$
8: **return** (pk, sk)

Algorithm 10 Niederretier encryption

Input: $\mathbf{m}$: plain text over $\mathbb{F}_q^k$

pk : public key

Output: $\mathbf{y}$: cipher text over $\mathbb{F}_q^n$

$\mathbf{e}$: a random error over $\mathbb{F}_q^n$

1: $\mathbf{c} = H'\mathbf{m}^\top$
2: $\mathbf{e} \leftarrow$ random error with $weight(\mathbf{e}) = t$
3: $\mathbf{y} = \mathbf{c} + \mathbf{e}$
4: **return** $\mathbf{y}$

Algorithm 11 Niederretier decryption

Input: $\mathbf{y}$: cipher text over $\mathbb{F}_q^n$

sk : secret key

Output: $\mathbf{m}$: plain text over $\mathbb{F}_q^k$

$\mathbf{e}$: a random error over $\mathbb{F}_q^n$

1: $\mathbf{d} = A^{-1}\mathbf{y}$
2: $\mathbf{a}$=SyndromeDecode$_H(\mathbf{d}) = P\mathbf{m}^\top$
3: $\mathbf{m}^\top = P^{-1}\mathbf{a}$
4: **return** $\mathbf{m}$

2 | Quantum Computing

The word Computer today has a very specific meaning: a machine, a construct, whose only purpose is to execute a given command. Its shape and form can be wildly different, but our relationship with it is always the same: we give it an input, an output comes out. In our society, it's a given.
The mathematical logic behind its reasoning has been thoroughly studied in the '30s and '40s by Alan Turing and John von Neumann, shortly after that, advancements in the field of electronics led to the realization as well as miniaturization of the execution of those logical axioms. Our phones and laptops are based on chips that follow the rules of voltages, currents, capacities. What we are going to examine in this work act in accordance of that same mathematical logic by following a radically different set of rules whose properties have historically been a challenge to comprehend: Quantum Mechanics.

The content of the following chapter is based on the work of Nielsen and Chuang [25].

2.1. History

At the beginning of the twentieth century, the foundations of physics were brought into question. Up until that point, the laws of physics managed to model, to give meaning to our reality from the point of view of our everyday world. Observation and experimentation were limited to what a scientist could experience. These laws worked so long as the world they were applied on (or more like taken from) was big: we can experience quite easily what is gravity, what is an electrical current. Problems arose when these rules where confronted with a very different world, a very small one. Atoms, light particles and electrons seemed not to obey particularly well to those models. These laws where not enough, a completely new paradigm had to be found.

2.2. Quantum Mechanics

Quantum Mechanics is, like the Classical laws we have just described, a mathematical framework or set of rules for the construction of physical theories. These rules have

always been proven hard to grasp, to the point that Einstein himself couldn't reconcile his own work with them. Yet, Quantum Mechanics today offers the most accurate and complete description of the laws that rule the physical world we live in.

2.2.1. Properties

Before analyzing in detail the fundamental postulates this new paradigm is based upon, we need to provide an intuition about a few key properties that characterize Quantum Mechanics and directly Quantum Computing: the concept of **Superposition**, how **Measurements** work and **Entanglement**.

The most basic element of classic computation is called *bit*, and it can famously only be in two states: either 0 or 1. In Quantum Computing, the corresponding computation element is the *quantum bit* or shortly *qubit*. Just like its classic counterpart, the qubit has as possible states $|0\rangle$ and $|1\rangle$, but the key difference is that its state can also be any linear combination between 0 and 1:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

This property is called **Superposition**. Here α and β are complex numbers called *amplitudes*, that is why the state of a qubit is a vector in a two-dimensional complex vector space. The special states $|0\rangle$ and $|1\rangle$ are known as *computational basis states*, and form an orthonormal basis for this vector space.
There is a catch: Quantum Mechanics tells us that we can only acquire much more restricted information about the quantum state. This is the introduction to the **Measurement** problem: when we measure a qubit we cannot determine its quantum state. A state amplitude, if squared, represent the probabilities of measuring that specific base state of the state $|\psi\rangle$, and when a measurement is applied we only obtain that base state. It naturally follows that for any qubit $|\alpha|^2 + |\beta|^2 = 1$. Geometrically, we can interpret this as the condition that the qubit's state vector be normalized to length 1, that is why generally a qubit is a unit vector in a two-dimensional complex vector space.

The phenomenon of **Entanglement** is the last key element of these new laws that is going to be a resource of great utility for quantum computation.

As we will understand better later by looking at the definition in postulate 4, an entangled state is a composite system that can't be written as a product of states of its component systems, meaning there is no single qubit states $|a\rangle$ and $|b\rangle$ such that $|\psi\rangle = |a\rangle\,|b\rangle$.

Consider the following two qubit state called EPR pair:

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.2}$$

This is the main counter-intuitive property that characterizes Quantum Mechanics whose effects are still studied today: measuring the state $|\psi\rangle$ of (2.2) gives rise to a behaviour that puzzled physicists for quite some time. By following its definition, if we apply a **Measurement** to the two qubits we will have the same probability of observing either $|00\rangle$ or $|11\rangle$. However, if we decide to measure only one of those qubits observing, say, a $|0\rangle$, and observe the other, then the whole state collapses to $|00\rangle$. This means that observing the first qubit immediately fixes the second to a specific value: with no direct observation we already know its state just by interacting with the first qubit. The surprising part is that these qubits do not have to be close to each other once they have been entangled. This behaviour is one of the reasons why scientist where initially skeptical about the laws of Quantum Mechanics since it breaks a fundamental notion of physics: information cannot travel faster than light itself. It turns out, on the other hand, that we are not conveying anything, since we are simply aware from our qubit's observation what is happening on the other end of the entangled state.

There are a couple of noteworthy properties before continuing, these are especially important in case we want to achieve some kind of computation with this new paradigm. The first is called **No-Cloning Theorem** from [40]:

Theorem 2.1. (No-Cloning Theorem) *It is not possible to make an identical copy of an arbitrary unknown quantum state.*

This means that a device that, given a quantum state, outputs one or multiple copies of that same state does not exists and never will. This is one of those truths imposed by reality, a fundamental characteristic of the quantum world. If we want to perform computation in this field this may sound quite troublesome, but new rules mean new issues as well as new opportunities.

Quantum Teleportation will sound like a direct counter-example of the No-Cloning theorem, yet a small detail will make a big difference. It turns out that it is possible to transfer a quantum state from a qubit to another if they are entagled. This operation comes at the cost of the measurement of the sender qubit, thus the quantum state is lost from one side of the channel. Investigated for the first time in [4], the process involves, other the a shared EPR pair, just 2 bits of classic information that are transmitted in a classical channel. Once one qubit of the pair has been measured, information about the

measured state is then sent to the other qubit recipient, who will interact with its system accordingly. The result is that the sender is left with a measured system, meaning the original quantum state is lost, whereas the receiver qubit's state is exactly equal to the original sender's state.

These basic properties have been useful in giving us a quick high level representation of this new model. We will now delve into the actual physical axioms that form the foundation of Quantum Mechanics.

2.2.2. Postulates

We will briefly describe the fundamental postulates that Paul Dirac [11] and John Von Neumann [38] introduced in the early '30s. For a better understanding of the Dirac notation and linear algebra, we suggest reading Appendix A.

Postulate 1. *Any isolated physical system is associated with a complex Hilbert space called* state space *of the system. The system is completely described by a unitary vector in this state space, called* state vector*.*

As we have already described, the simplest isolated physical system is a two-dimensional state space called *qubit*. Its Hilbert space is $\mathbb{C}^2$. If we consider $|0\rangle$ and $|1\rangle$ as orthonormal basis of this state space (the quantum version of the classical 0 and 1), then an arbitrary state vector can be written as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.3}$$

where $\alpha, \beta \in \mathbb{C}$. The unitarity condition expressed in the postulate is modeled via the formula we have already illustrated $|\alpha|^2 + |\beta|^2 = 1$.

The following postulate gives a description of how a system changes with time.

Postulate 2. *The evolution of a* closed *quantum system is described by a* unitary transformation*. Meaning, the state $|\psi\rangle$ of the system at time t_1 is related to the state $|\psi'\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2*

$$|\psi'\rangle = U\,|\psi\rangle \tag{2.4}$$

A closed system does not interact in any way with any other system. This is to be considered an approximation since no physical system in nature is completely isolated. We can refine the second postulate by describing the evolution of a closed physical system in continuous time.

Postulate 3. *The time evolution of the state of a closed physical system is defined by the* Schrödinger equation

$$i\hbar \frac{d\,|\psi\rangle}{dt} = H\,|\psi\rangle \tag{2.5}$$

Here $\hbar$ is Planck's constant*, whose value has been experimentally determined. H is a fixed Hermitian operator known as the Hamiltonian of the closed system.*

In principle, knowing a system's Hamiltonian means that we understand its dynamics completely, yet figuring it out can be extremely complicated. What is worth pointing out is that there is a one-to-one correspondence between the discrete-time description of dynamics using unitary operators and the continuous time descriptions using Hamiltonians. In order to observe what is going on in a physical system we are forced to interact with it and rendering it not closed, meaning not necessarily subject to unitary evolution. We need to describe what happens when the system is subject to measurement.

Postulate 4. *Any* Observable *M (any property of a physical system that can be measured) is represented by an Hermitian operator on the system's state space that is observed. M is equal to*

$$M = \sum_m m P_m \tag{2.6}$$

where P_m is the projection onto the subspace formed by M's eigenvectors that correspond to the eigenvalue m. The possible results of an observable M's measurement are the eigenvalues m of M. After a measurement of M in the state $|\psi\rangle$, the probability that result m occurs is

$$p(m) = \langle\psi|\,P_m\,|\psi\rangle = ||P_m\,|\psi\rangle\,||^2 \tag{2.7}$$

The state of the system immediately after a measurement with result m is

$$\frac{P_m\,|\psi\rangle}{\sqrt{p(m)}} \tag{2.8}$$

P_m operators satisfy the completeness equation

$$\sum_m P_m = I \tag{2.9}$$

The equation above denotes the fact that the sum of the probabilities of a measurement is 1. Meaning

$$1 = \sum_m p(m) = \sum_m \langle\psi|\,P_m\,|\psi\rangle = \langle\psi|\psi\rangle \tag{2.10}$$

In reality, the type of measurement we have just described is called *projective measurement* which can be considered a special case of measurement, but so long as we perform only unitary transformations this specialization is equivalent to the general case.

Generally speaking, the result of the measurement of an observable M in a state $|\psi\rangle$ is random. By measuring multiple copies of a given system it is possible to study the probabilistic distribution of the possible results. We can analyze it by introducing the definition of *average* $\mathbb{E}$ of a random variable (the observable M) which corresponds to the weighted average over all the possible outcomes.

$$\begin{aligned}\mathbb{E}_{|\psi\rangle}(M) &= \sum_m m P_m \\ &= \sum_m m \langle\psi| P_m |\psi\rangle \\ &= \langle\psi| \left(\sum_m P_m\right) |\psi\rangle \\ &= \langle\psi| M |\psi\rangle \equiv \langle M\rangle\end{aligned} \tag{2.11}$$

We can extend even further via probability theory notation and define the *standard deviation* associated to an observable M as

$$\begin{aligned}[\Delta(M)]^2 &= \langle (M - \langle M\rangle)^2\rangle \\ &= \langle M^2\rangle - \langle M\rangle^2\end{aligned} \tag{2.12}$$

which is a measure of the typical spread of the observed values upon measurement of M. The *average* and *standard deviation* formulations in terms of observables are especially useful in that they simplify a lot of calculations and come particularly in handy during the definition of the *Heisenberg uncertainty principle*.

Lastly, the following postulate describes how the state space of a composite system is built up from the state spaces of the component systems.

Postulate 5. *The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. If we have systems numbered 1 through n, and system number i is prepared in the state $|\psi_i\rangle$, then the joint state of the total system is $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$*

The reason why a composite system is defined as the tensor product of the component systems can be easily understood when looking at how the concept of superposition is mathematically represented: instead of applying the principle to an isolated system $|A\rangle$,

if applied to a composite one $|A\rangle |B\rangle$ we come to the tensor product in the postulate.

To wrap everything up, all the fundamental postulates can be summed up as the following:

- Postulate 1 defines how the state of an isolated quantum system is to be described.
- Postulate 2 and 3 tell us that the dynamics of closed quantum systems are described by the Schrödinger, thus by unitary evolution.
- Postulate 4 gives us a way to extract information from a quantum system by describing the concept of measurement.
- Postulate 5 gives the definition of how to combine different quantum systems thus describing the composite system.

The main difference between the quantum world and classic physics is the fact that we can't directly observe the state vector in the latter. A fundamental property of an object is no longer directly observable, yet we can indirectly interact with it. Measuring in this environment is like an imposition, a forced standpoint that can no longer be taken lightly since by Postulate 3 this action changes the state of the system. On the other hand, this weird and counter-intuitive behaviour is exactly the strong point that these new set of rules gives us: we can exploit the hidden nature of the state vector to do information processing tasks beyond what is possible in the classical world.

2.3. Information Modeling

Before examining how we can actually perform any kind of computation in this quantum environment, we need to briefly introduce how computation itself has been historically modeled. The first problem scientist has to agree on was the formulation of the concept of algorithm. After all, we compute by following a series of instructions, thus by following an algorithm. The fundamental notions of modern algorithm theory, i.e. computation theory, were introduced in the 1930s by Alan Turing and Alonzo Church who were trying to understand whether an algorithm that could solve all mathematical problems existed or not. The answer turned out to be negative, nonetheless from their work the model of modern computation was formalized.

The **Turing Machine** is a class of machines that captures the notion of an algorithm that performs a computational task. It consists of:

- A program, a list of instructions the machine has to follow.
- A finite state control, it represents the internal state of the machine during the

computation. It always includes a starting state and one (or more) halting states.

- A tape, it essentially acts like a memory.
- A read-write tape-head, it points to the position on the tape which is currently readable or writable.

To summarize its functioning: the machine starts its operation with the finite state control set to the starting state and with the read-write head at the leftmost tape element. The computation proceeds in a step by step manner according to the program's instructions. An instruction is executed only if the current element under the tape and the current internal state are correct, if so the instruction may specify a change in the current state as well as a write on the tape and the movement of the head. The execution terminates whenever a halting state is reached and what can be read on the tape is considered the output.

The computational capabilities of such a machine are considerable. So much so, Turing conjectured that it is a model for computation that completely captures the notion of computing a function using an algorithm. Church achieved the same result by studying lambda calculus.

Thesis 1. (Church-Turing) *The class of functions computable by a Turing machine corresponds exactly to the class of functions which we would naturally regard as being computable by an algorithm.*

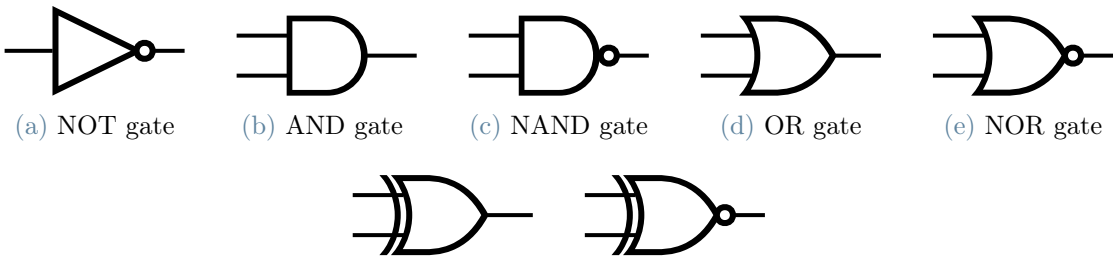
So far, no evidence that suggests this thesis to not be true has been found. The Church-Turing thesis suggests that a Turing machine is an ideal starting point for the creation of a computing construct. The ideal part is a key problem here, a real device is finite in size from any point of view.

We now analyze a model called **circuit model** that is computationally equivalent to Turing machines but much more useful when confronted with the problem of applications on a real device. A *circuit* is made up of:

- Wires, that connect the different components of the circuit and carry the information content.
- Gates, the structures that actually perform the computation based on the content of the input wires.

Gate types may vary, yet they all can be considered placeholders for the execution of a general boolean function $f : \{0,1\}^k \to \{0,1\}^l$ which takes as input k bits and outputs

l bits, therefore k wires will be needed as inputs of the gate and l wires as outputs. A comprehensive list of possible classical gates as well as their truth table is shown below.



(a) NOT gate (b) AND gate (c) NAND gate (d) OR gate (e) NOR gate

(f) XOR gate (g) XNOR gate

Figure 2.1: A summary of the most used logic gates

a	b	$\neg a$	$a \wedge b$	$\neg(a \wedge b)$	$a \vee b$	$\neg(a \vee b)$	$a \oplus b$	$\neg(a \oplus b)$
0	0	1	0	1	0	1	0	1
0	1	1	0	1	1	1	1	0
1	0	0	0	1	1	0	1	0
1	1	0	1	0	1	0	0	1

Table 2.1: Truth table of the gates shown above: $\neg$ NOT, $\wedge$ AND, $\vee$ OR, $\oplus$ XOR

Like a puzzle, we can put together may of these gates which can be considered the building blocks of modern computation to perform a huge amount of different computations. It can be shown, in fact, that these elements can be used to compute *any* boolean function whatsoever.

This model of computation can be proven to be computationally equivalent to a Turing machine. Specifically, it is possible to implement any Turing Machine with a synchronous circuit with Boolean gates and single-bit synchronous memory elements. The proof is quite lengthy, the key point is that it is possible for an algorithm running on a Turing machine to give a description of a circuit. The circuit model therefore inherits its capability concerning the notion of computing explained in (1).

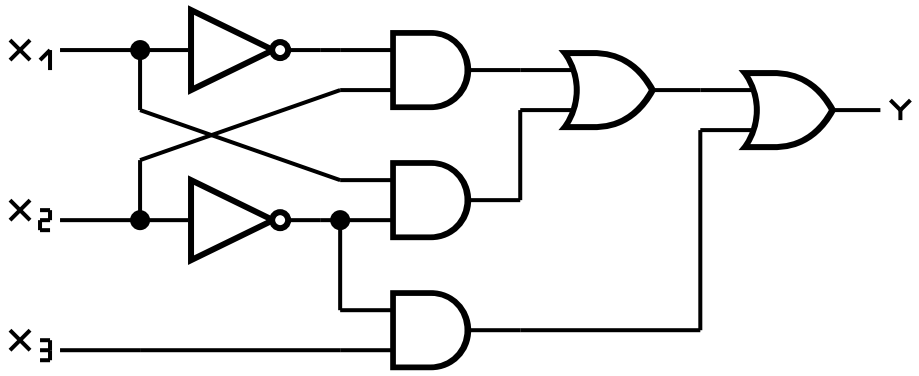


Figure 2.2: An example of a classic combinatorial circuit

What we have just outlined is the theory at the foundation of computation that we need in order to be able to give an accurate description of what a quantum circuit is. We can thus proceed to analyze its components.

2.4. Quantum Circuit

Having described the basics from classical theory, we can now delve into the quantum model. The quantum circuit model can be considered today the framework we can base quantum computation upon, and, just like its classical counterpart, gates are the sub-structures that perform the actual computation followed and/or preceded by wires that connect multiple instances.

Again, we suggest reading Appendix A for an explanation of the Dirac notation as well as some basic linear algebra concepts.

2.4.1. Single Qubit gates

The most basic gate type is the single qubit gate. As we have already mentioned, a single qubit is a vector $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, parametrized by two complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$. Since operations on a qubit must always preserve this norm, a single qubit transformation is described by a 2×2 unitary matrix. This means that, while classically there is only one non-trivial operation on a bit (the **NOT** operation), in the quantum model we can have multiple when concerned with a single qubit. The corresponding quantum operation of the **NOT** gate is called **X** and is defined as:

$$\mathbf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.13}$$

Indeed, on a $\alpha|0\rangle + \beta|1\rangle$ qubit an **X** gate applies the following transformation:

$$\mathbf{X}\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\beta\\ \alpha\end{bmatrix}$$

where it is clear the connection with the **NOT** operation: it swaps the amplitudes of the base states $|0\rangle$ and $|1\rangle$.
Among the single qubit operations, there are three more that are worth noting. The first two are the **Z** gate and the **Y** gate.

$$\mathbf{Z} = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix} \tag{2.14}$$

We can read how a transformation alters a quantum state by looking at the two column vectors of the 2×2 matrix: the first one shows how the base state $|0\rangle$ is transformed whereas the second concerns the base state $|1\rangle$. For example, the **Z** gate leaves unaltered the $|0\rangle$ base state while flipping the sign of the $|1\rangle$ component. Together, the **X**, **Y** and **Z** gates are called *Pauli matrices* and represent respectively the x, y and z components of an electron's spin: they each apply a π radians rotation to the quantum system around the corresponding axis. When exponentiated, the Pauli matrices produce the *rotation operators*:

$$\mathbf{R}_x(\theta) \equiv e^{-i\theta\frac{X}{2}} = \cos\left(\frac{\theta}{2}\right) I - i\sin\left(\frac{\theta}{2}\right) X = \begin{bmatrix}\cos\frac{\theta}{2} & -i\sin\frac{\theta}{2}\\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{bmatrix} \tag{2.15}$$

$$\mathbf{R}_y(\theta) \equiv e^{-i\theta\frac{Y}{2}} = \cos\left(\frac{\theta}{2}\right) I - i\sin\left(\frac{\theta}{2}\right) Y = \begin{bmatrix}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2}\\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{bmatrix} \tag{2.16}$$

$$\mathbf{R}_z(\theta) \equiv e^{-i\theta\frac{Z}{2}} = \cos\left(\frac{\theta}{2}\right) I - i\sin\left(\frac{\theta}{2}\right) Z = \begin{bmatrix}e^{-i\frac{\theta}{2}} & 0\\ 0 & e^{i\frac{\theta}{2}}\end{bmatrix} \tag{2.17}$$

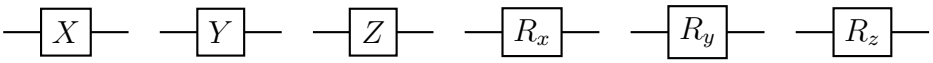
Lastly, one of the most useful single qubit gates is the *Hadamard* gate:

$$\mathbf{H} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \tag{2.18}$$

Its effect is to transform a basis state into a superposition such that a measurement would output one of the two basis states with the same probability. As we will learn, this transformation is the centerpiece of any search-related algorithm since a parallelized application of n of these gates on n qubits can quickly create the superposition of all the

possible combinations of those qubits basis states.

On a circuit, the gates we have analyzed so far are simply represented by their respective letters in a box acting on a single wire.



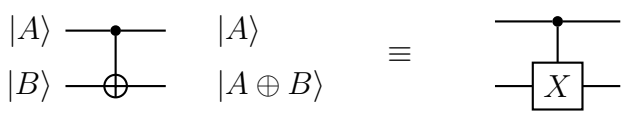
As an additional note, the *measurement* operation can be considered a single qubit operation as well, although not technically a gate. Its purpose is to represent the measurement action on the wire which will output, as described in 2.2.1, one of the two possible basis states depending on the current state's amplitudes. Since only one of the basis states can be measured, post-measurement the wire can be effectively considered a classical wire, therefore the operation is usually applied at the end of the circuit.

2.4.2. Multi Qubit gates

The ability to perform controlled operations is a fundamental part of any form of computation model. The same principle can be applied when dealing with quantum transformations. A *controlled gate* is a gate with one or more *control qubit(s)* and a *target qubit*. The state of the target qubit after the transformation depends on the state of the control qubit before it.

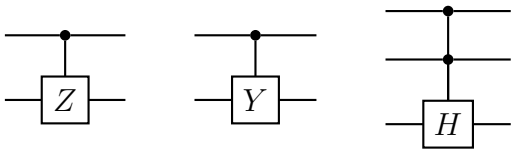
The most classic controlled operation is the **CNOT** wherein the control qubit state determines whether the target qubit state is gonna be flipped or not. More specifically, an **X** gate is applied to the target qubit if the control qubit's state equals to the basis state $|1\rangle$. It acts like a quantum variant of the classic **XOR** operator, indeed in bra-ket notation its action can be expressed as $|A\rangle\,|B\rangle \rightarrow |A\rangle\,|A \oplus B\rangle$ where A is the control qubit and B is the target qubit.

$$\textbf{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.19}$$



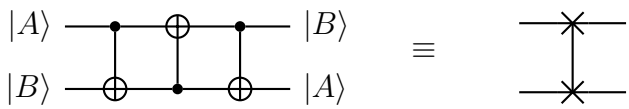
It's worth noting that, like any unitary transformation, the **CNOT** operation is reversible: if we were to reapply the same gate on the same wires the quantum system's state would go

back to the original state. This does not generally hold true for any classical operation. As an example, from the output of the boolean **OR** operation it is not possible to determine what input configuration caused the bit to have that final value.
Any single qubit gate can undergo the same process: the gate's transformation is applied to the target qubit if the control qubit's state is equivalent to $|1\rangle$.



2.4.3. Example and Properties

Gathering all pieces of information described so far, a quantum circuit follows the circuit model analyzed in 2.3: an arbitrary number of gates acting on different wires implement a boolean function whose output is then measured. Seeing that the measurement is an intrinsically stochastic operation, it may be necessary to repeat the whole process multiple times in order to obtain an accurate representation of the quantum state at the end of the computation.
As an example, we show a circuit that performs a swap of the states of two given qubits:



We can have a look at how the quantum system changes after each single gate application.

$$
\begin{aligned}
|A, B\rangle &\rightarrow |A, A \oplus B\rangle \\
&\rightarrow |A \oplus (A \oplus B), A \oplus B\rangle = |B, A \oplus B\rangle \\
&\rightarrow |B, (A \oplus B) \oplus B\rangle = |B, A\rangle
\end{aligned}
$$

A few key differences between the quantum circuit model and the classical one are worth noting. In a quantum circuit *loops* are not allowed, circuits have to be *acyclic*. Secondly, a classical circuit allows two wires to be joined together performing what is called a **FANIN** operation which is a bit-wise **XOR** of the two inputs. Again, in a quantum setting this is not possible since this is not a reversible operation. Lastly, the inverse operation **FANOUT** is not viable anymore since it is basically a copy operation which violates the No-Cloning theorem described in (2.1).
To put it shortly, the number of outputs (wires) coming out of any quantum transformation has to be equal to the number of inputs, otherwise the transformation has no possibility of being reversible.

2.5. Quantum Algorithms

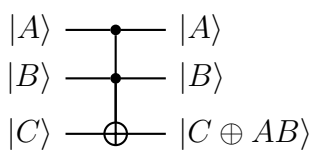
Having described the model of computation, we try to analyze what so far has been accomplished with it. The similarities with the classical model, and especially the differences, can be better understood if the perspective is now the steps of an actual algorithm instead of a simple component description. Lastly, we show a possible classification of these algorithms based on the different paradigms they make use of.

2.5.1. Classic Computation & Quantum Parallelism

Being it based on quantum mechanics as well, it is expected that a classic logic circuit can be simulated using a quantum circuit. It turns out that it is indeed possible, but a few changes may be needed. The most fundamental difference between the two models is the fact that unitary quantum logic gates are always invertible thus reversible, whereas some classic logic gates are inherently irreversible. In other words, the function a quantum circuit computes must always be bijective, a condition that is not necessary when considering a classical circuit.

Any classical circuit can be replaced by an equivalent circuit containing only reversible elements by using a gate known as *Toffoli gate*. It has two control qubit and a target qubit controlled via an **X** gate.

$$\textbf{TOFFOLI} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.20}$$



A proper sequence of applications of this gate can achieve the simulation of the **NAND** gate as well as the **FANOUT** operations. Given the universality of these operations, it becomes possible to simulate all other transformations, meaning an arbitrary classical

circuit can be simulated by an equivalent reversible circuit. Once the necessary changes are done, the equivalent classical circuit built via the usage of Toffoli gate can be simulated by a quantum circuit, noting that two subsequent application of the Toffoli gate return the original quantum state $|A, B, C\rangle \rightarrow |A, B, C \oplus AB\rangle \rightarrow |A, B, C \oplus AB \oplus AB\rangle \equiv |A, B, C\rangle$ demonstrating its reversibility. It is thus possible, given a general classic function f, to build a quantum circuit that performs the following transformation:

$$|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle \tag{2.21}$$

where x is the input and y is are the allocated qubits for the output.

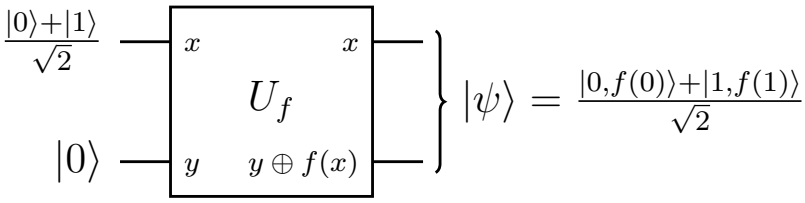
We can further expand upon these considerations regarding simulation by taking into consideration probabilistic classic computers. This class of machines bases the computation on the generation of random bits. Being randomness a fundamental component of the quantum environment, a quantum machine can easily simulate such constructs. A very basic circuit that could represent a simple coin toss is achievable via the use of a single Hadamard gate.

$$|0\rangle \;—\boxed{H}—\; \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

Measuring the output state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ will result in the state $|0\rangle$ or $|1\rangle$ with 0.5/0.5 probability, which is the same probabilistic behaviour of a coin toss. Any random process is inherently efficient to simulate thanks to gates like Hadamard, in view of the fact that any superposition is naturally a stochastic process.

This property is the reason why it is possible to define a key feature of many quantum algorithms: **Quantum Parallelism**. The ability to efficiently simulate probabilistic processes allows quantum computers to evaluate a function $f(x) : \{0, 1\} \rightarrow \{0, 1\}$ for *different* values of x simultaneously.
First we give the mapping (2.21) a name U_f (for our purpose we can consider it a black box). Secondly, we note that:

- If we put $|0\rangle$ in the second register $|y\rangle$ the final state of the second register will be exactly $|f(x)\rangle$.
- If we put a superposition of $|0\rangle$ and $|1\rangle$ in the first register we can evaluate the function for both inputs at the same time.



The final state is a superposition of two states that contains both the inputs of the function together with their evaluations. Unlike classical parallelism, where multiple circuits can evaluate different inputs at the same time, quantum parallelism exploits the superposition properties and manages with a *single* circuit to evaluate many different inputs.

There is the not-so-small detail that it is not possible to obtain all the possible outputs with a single measurement. As we have observed in 2.2.1, here measurements give *either* $|0, f(0)\rangle$ *or* $|1, f(1)\rangle$. In order to extract information about more than one value of $f(x)$ from superposition states like these the most naive approach would be to repeat the whole process obtaining another possible output state. With more and more measurements of the same circuit one can obtain a piece of information about the probabilistic behaviour of the final quantum state approximated to a sufficient extent. Differently, particular algorithms may obtain this kind of information from a generic quantum state using more specific methods.

2.5.2. Main Algorithm Classes

Broadly speaking, the quantum algorithms that so far have been proven to provide an advantage over the corresponding classic algorithm can be grouped under three different classes.

A first possible class is characterized by algorithms that are based upon the quantum version of the **Fourier transform**. Algorithms that belong to this class are the Deutsch-Josza [8] and Shor [35] algorithms. The Discrete Fourier transform (DFT) is described as transforming a set $x_0, \ldots, x_{N-1}$ of N complex numbers into a set of N complex numbers $y_0, \ldots, y_{N-1}$ defined as

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2\pi ijk}{N}} x_j$$

Generally speaking the Fourier transform has always had a huge number of applications in any branch of science. On a quantum computer the Walsh-Hadamard transformation (a size-2 DFT) can be accomplished with an exponential speedup compared to the classical equivalent, but suffers again from accessing the information hidden in the amplitudes of

the quantum state.

Secondly, algorithms may fall under the **quantum search** category, whose main representative is Grover's algorithm [14]. All algorithms that belong to this class solve the following problem: given a N-sized search space containing information with no structure, we want to find an element inside it, more specifically an element that satisfies a given property. This problem is classically solved in $O(N)$ operations because in the worst case scenario we would need to look at all the elements in the database. A quantum search algorithm on the other hand can solve it in $O(\sqrt{N})$ operations. The achievable speedup is only quadratic compared to the previous class, but the applications regarding this problem are even wider.

The last class of quantum algorithms is **Quantum simulation**, whereby a quantum computer simulates a quantum system. The evolution of natural quantum mechanical systems are extremely difficult to study using a classical machine. By definition, the classic computation necessary to model a quantum system grows *exponentially* with its size rather than linearly. Storing information regarding a quantum system with n distinct components takes c^n bits of memory with c being a constant that depends on how a qubit is defined. A quantum computer can represent that same system with kn qubits, again with k being a different constant defined on the qubit's definition. This means that a quantum computer is, unsurprisingly, much more efficient in performing a simulation of a naturally quantum system than a classical computer. One important field this class of algorithms could bring a significant contribute to is quantum chemistry as the behaviour of molecules is increasingly difficult to simulate the larger their size.

2.6. Quantum Speedup & Complexity

In order to better understand how and why such a machine can have an advantage or not, we briefly discuss the basics of **computational complexity theory**.
The main focus of this theory is to classify computational problems based on the quantity of resources, typically time and/or space related, required to solve them on a particular machine. A complexity *class* can be thought of as a collection of computational problems that share a common feature regarding the usage of resources to solve them.

Classically speaking, the two most important classes are **P** and **NP**: **P** is the class of computational problems that can be *solved* in polynomial time in the size of the input on a classical computer whereas **NP** contains problems that have solutions that can be *checked* in polynomial time in the size of the input on a classical computer. Although it is easy to understand why $\mathbf{P} \subseteq \mathbf{NP}$, their relationship is not trivial, as to this day no one

knows if these classes are really different or not. In other words:

$$\mathbf{P} \overset{?}{\neq} \mathbf{NP}$$

Despite being able to solve some problems that are believed to be in **NP** but not in **P**, it is not yet known whether quantum computers can be used to quickly solve all the problems in **NP**. The search-based advantage that derives from quantum parallelism does not always yield an efficient solution to a problem in **NP**, only some examples have been found so far.

In the quantum setting the class of computational problems which can be solved efficiently on a quantum computer is called **BQP**. In this case, seeing that these are probabilistic procedures by nature, a bounded probability error is allowed. Specifically, the final measurement of an encoded problem in such a machine yields the correct answer with probability $Pr > \frac{1}{2} + k$ with $\frac{1}{2} + \frac{1}{poly(n)} < k < 1 - \frac{1}{2^n}$ where n is the input length. Following from the fact that the Toffoli gate is an universal gate for reversible classic computations, it can be stated that $\mathbf{P} \subseteq \mathbf{BQP}$. Yet, nobody can confirm whether $\mathbf{NP} \subseteq \mathbf{BQP}$, unless **P**=**NP**.

Hence no proven superiority does not rule out the fact that further experimentation is not worthwhile. As of the writing of this theses the so called *Quantum Supremacy* has not been achieved, if ever possible. Still, in the past few years the quantum computing field has witnessed a tremendous growth both in terms of grants for research as well as the number of new startups that bring new tools based on quantum technologies to the market. This suggests that interest is rapidly increasing and that industries from different fields can benefit from this new model of computation, whether their business domain is financial, chemical or logistics.

3 | Quantum Random Walks

In July of 1905 the English mathematician Karl Pearson posed an interesting question in an issue on *Nature* [26]. He wondered if someone knew the solution to the problem he called *the Random Walk*. He defined it as finding an integrated solution of the distance from an origin that an agent covers when traveling in a random fashion. Specifically, starting from the origin O the agent chooses randomly an angle and walks l yards in a straight line. Once this initial distance has been travelled, the process is then repeated n times.

Random walks will later find many applications in different fields and will be considered the base model for numerous stochastic processes. Being the Quantum world stochastic by nature, at the beginning of the 1990s interest around possible applications of this topic in this new model grew. In 1993 Aharonov, Davidovich and Zagury coined the term **Quantum Random Walks** [1].

An introductory overview on the topic from J. Kempe can be found in [17] and an in-depth book about this topic is [28].

3.1. Overview

3.1.1. Classical Random Walk

In [1], before introducing the quantum equivalent, the authors give a slightly different definition of random walk compared to the original paper written 90 years before. The walk is performed on a line: the random choice of an angle is reduced to a simple left or right decision.

The **Classical One-dimensional Random Walk** is a mathematical representation of an agent that can move along a line. This means that it is defined in terms of the probabilities of a particle to make a step of a given length to the left or to the right. It is a simplification, but it can be considered an initial example without any loss of generality.

Such a walk can be represented as a probability distribution on a discrete line of the

position of a walker that starts the exploration in its middle. Given a time-step t and a position n we can compute what is the probability p of the particle being in that position at that time step as:

$$p(t,n) = \frac{1}{2^t}\binom{t}{\frac{t+n}{2}} \tag{3.1}$$

where $\binom{a}{b} = \frac{a!}{(a-b)!b!}$

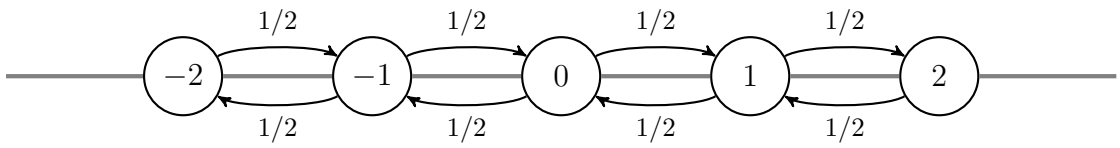


Figure 3.1: Classical random walk on the line

If the agent starts on an initial position 0 and does not walk, the probability of remaining in the origin is unsurprisingly 100%, meaning $p(t=0, n=0) = 1$. At the next time step there will be a 50% probability of being in position -1 and 50% probability of being in position 1, $p(t=1, n=1) = 1/2$, $p(t=1, n=-1) = 1/2$. At each time step, we repeat the same process: randomly choose left or right, then move.

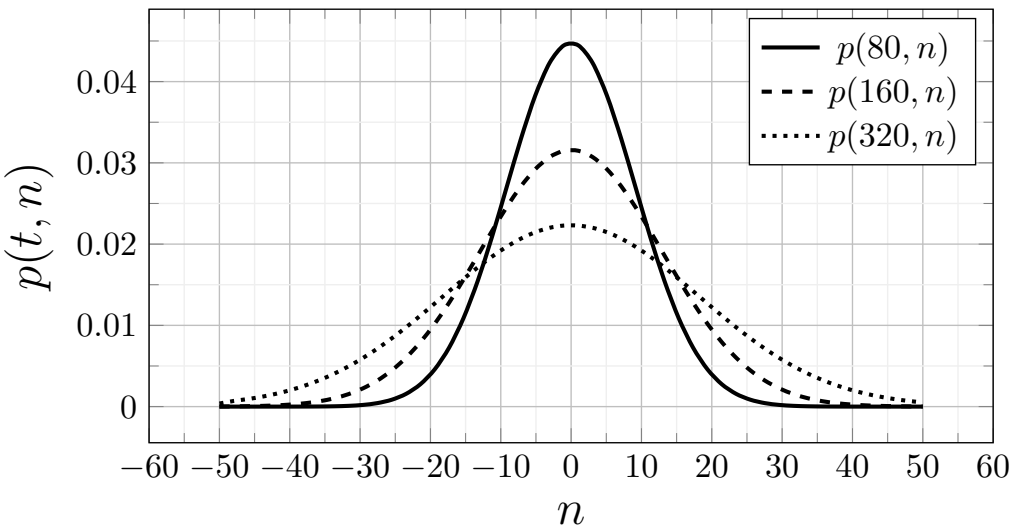


Figure 3.2: Binomial distribution of the classic random walk

If we were to keep iterating the process and plot the result on a graph (3.2), we can see that for a fixed value of t, $p(t,n)$ is a *binomial distribution*. It is interesting to analyze some properties of such curve.

The *average position* on the curve is $\mathbb{E}_n = \sum_{n=-\infty}^{\infty} n\, p(t,n) = 0$ which can be noticed by

the symmetric behaviour of the curve around the origin. The initial position will always be the most probable to be in after any t time steps. As t increases, the height of the midpoint of the curve decreases, whereas the width increases.
The *standard deviation* captures the idea of how far away from the origin we can expect to find the agent as time goes on and is defined as $\sigma(t) = \sqrt{\sum_{n=-\infty}^{\infty} n^2\, p(t,n)} = \sqrt{t}$.
We will better understand later why this value has a great importance for the topic. For now, we simply state that in a classical random walk the *expected distance*, given t time step, is $\sqrt{t}$.

3.1.2. Classical Discrete Markov Chains

A proper mathematical framework that better represents such behaviour is needed. Average and standard deviation alone are not sufficient to thoroughly study this process.

Definition 3.1. (Classic Markov chain) *is a stochastic process that assumes values in a discrete set of states that are randomly selected after each time step. States can be seen as vertices in a directed graph where edges indicate what the possible next states are. They obey the* markovian *property: the choice of the next state is not influenced by the past.*

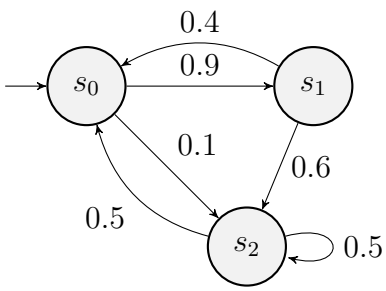


Figure 3.3: Markov chain with three states

Classical *discrete* Markov chains are chains with discrete time variables, meaning that at each time step there is an associated probability distribution (the set of probabilities for an agent to begin in each state/vertex). A finite vector can describe such a distribution. Let $G(X, E)$ be a graph with a set of vertices $X = \{x_1, \ldots, x_n\}$ ($|X| = n$) and a set of

edges E. The probability distribution takes the following form:

$$\boldsymbol{p}(t) = \begin{bmatrix} p_1(t) \\ \vdots \\ p_n(t) \end{bmatrix} \tag{3.2}$$

where $p_i(t)$ is the probability of the agent being in vertex x_i at time t.

We cannot determine precisely the evolution of the walk's probability distribution at each time step without considering the structure of the graph. In order to encode this in a matrix form, we use the *transition matrix* M.

Definition 3.2. (Transition (or Stochastic) Matrix) *contains the information regarding the probability of moving from a vertex i to j for every pair (i, j) of vertices in the graph. Each column vector of the matrix represents the probabilities of the outward edges of a specific vertex. A generic definition is:*

$$M = \begin{bmatrix} M_{1,1} & M_{2,1} & \dots & M_{i,1} & \dots & M_{n,1} \\ M_{1,2} & M_{2,2} & \dots & M_{i,2} & \dots & M_{n,2} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ M_{1,j} & M_{2,j} & \dots & M_{i,j} & \dots & M_{n,j} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ M_{1,n} & M_{2,n} & \dots & M_{i,n} & \dots & M_{n,n} \end{bmatrix}$$

The total state transition probability from a generic state i to all its adjacent states (including itself) must be one, meaning $\sum_{j=1}^{n} M_{i,j} = 1$.

If no edge is present from i to j then $M_{i,j} = 0$ whereas if the graph is undirected $M_{i,j} = M_{j,i}$. A complete graph having the property that every edge of a generic vertex has the same probability of being chosen is represented by $M_{i,j} = 1/d_j$ where d_j is the *degree* if vertex j.

Putting everything together, if the probability distribution is known at time t, its evolution at time $t + 1$ is defined as:

$$p_i(t+1) = \sum_{j=1}^{n} M_{i,j}\, p_j(t) \qquad \text{or} \qquad \boldsymbol{p}(t+1) = M\, \boldsymbol{p}(t) \tag{3.3}$$

An important property of this equation is that it is recursive, as such it can written as

$$\boldsymbol{p}(t) = M^t \, \boldsymbol{p}(0) \tag{3.4}$$

where $\boldsymbol{p}(0)$ is the initial condition. It is thus possible to multiply the initial probability distribution by the transition matrix to the t-th power to compute any successive evolution. This is useful because in a stochastic environment we must consider all possible evolutions of a system. The opportunity of describing all its possible transformations in a matrix form will come extremely useful in the quantum setting where, instead of strictly positive real numbers, the elements of M can be negative and complex (they are probability amplitudes).

3.1.3. Quantum Random Walk

Taking directly from the postulates (Section 2.2.2), we can extend the same principles to a quantum system since the concept of measurement is based on a probability distribution. We will follow the same example of a walk on a line shown in Figure 3.1.

In the *discrete model* of the quantum random walk the walker's position n should be a vector in Hilbert space $\mathcal{H}_P$ whose size depends on the degrees of freedom of the physical system, but we can initially consider it of infinite dimension. Its computational basis can be represented as $\{|n\rangle \, : \, n \in \mathbb{Z}\}$. The random left or right choice of the walk depends on a quantum *coin*: if *heads* is flipped and the walker is in position $|n\rangle$ then in the next step its position will be described by $|n+1\rangle$, if *tails* is flipped then the position will be $|n-1\rangle$. The Hilbert space of this system is thus described as $\mathcal{H} = \mathcal{H}_C \otimes \mathcal{H}_P$. $\mathcal{H}_P$ describes the position whereas $\mathcal{H}_C$ represents this coin choice, which is a two-dimensional Hilbert space with computational basis $\{|0\rangle, |1\rangle\}$.

These two transformations must be described by unitary operators that act on a vector of the corresponding Hilbert spaces. We will call S the shift operator and C the coin operator. The coin operator acts as the decision maker regarding the next step direction, whereas the shift operator executes the coin choice by modifying the position of the walker. The transformations are

$$\begin{aligned} C\,|0\rangle\,|n\rangle &= (a\,|0\rangle + c\,|1\rangle)\,|n\rangle \\ C\,|1\rangle\,|n\rangle &= (b\,|0\rangle + d\,|1\rangle)\,|n\rangle \end{aligned} \tag{3.5}$$

$$\begin{aligned} S\,|0\rangle\,|n\rangle &= |0\rangle\,|n+1\rangle \\ S\,|1\rangle\,|n\rangle &= |1\rangle\,|n-1\rangle \end{aligned} \tag{3.6}$$

where C is a generic coin operator defined as $C = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$.

One step of a quantum random walk is a succession of these two transformation which act on $\mathcal{H}_C$ first (C) then $\mathcal{H}_P$ later (S). We call *evolution operator*(U) the transformation that acts on the whole Hilbert space $\mathcal{H} = \mathcal{H}_C \otimes \mathcal{H}_P$ defined as:

$$U = S(C \otimes I) \tag{3.7}$$

where I is the identity matrix.

A quantum walk begins by tossing a coin, essentially choosing the future walk direction. This is performed by applying the coin operator to the initial state. The rotation of the coin state may result in a superposition of states. This means that via this superposition we can take both directions at the same time. We start the walk at the central position $n = 0$ at time $t = 0$ with an initial direction of $|0\rangle$

$$|\psi(0)\rangle = |0\rangle\,|n = 0\rangle \tag{3.8}$$

$|\psi(t)\rangle$ denotes the state of the quantum walk at time t.
We choose as the coin the Hadamard gate. As we have explained in Section 2.4.1, its effect is to transform a basis state in a superposition that outputs one of the two basis states with the same probability, which seems like a good property if we would like to traverse the line in a symmetrical fashion. This implies that, given any of the two directions in the coin space, this gate will randomly and evenly output 'right' or 'left'.
So, we compute one step of the walk, we toss the coin first then follow it up with a shift:

$$\begin{aligned} |\psi(0)\rangle = |0\rangle \otimes |n = 0\rangle &\xrightarrow{H\otimes I} \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |n = 0\rangle \\ &\xrightarrow{S} \frac{1}{\sqrt{2}}(|0\rangle \otimes |n = 1\rangle + |1\rangle \otimes |n = -1\rangle) \end{aligned} \tag{3.9}$$

As we can see we are in a superposition of position $|n = 1\rangle$ with direction $|0\rangle$ and position $|n = -1\rangle$ with direction $|1\rangle$.

Applying a measurement after one step will yield $|n = 1\rangle$ with probability 50% and $|n = -1\rangle$ with probability 50%, so the behaviour compared to the classical random walk is the same. If we don't measure the system and keep applying the $U = S(C \otimes I)$ transformation, each superposition will itself give rise to two more exploring elements, with either constructive or destructive interference with regard to other quantum components that are generated by other superpositions. Similarly to the classical walk evolution (3.4),

this process can be described recursively

$$|\psi(t)\rangle = U^t \, |\psi(0)\rangle \tag{3.10}$$

We could iteratively apply this definition to obtain $|\psi(t)\rangle$ for any t, but in order to compute $|\psi(100)\rangle$ we would have to repeat the process 100 times. We can achieve the same thing via a recursive formula: the generic walk state at time t is a linear combination of the computational basis as

$$|\psi(t)\rangle = \sum_{n=-\infty}^{\infty} \left(A_n(t)\,|0\rangle + B_n(t)\,|1\rangle\right)|n\rangle \tag{3.11}$$

that satisfies the constraint

$$\sum_{n=-\infty}^{\infty} |A_n(t)|^2 + |B_n(t)|^2 = 1 \tag{3.12}$$

ensuring the norm of $|\psi(t)\rangle$.
If we use the Hadamard gate as the coin, the recursive formulas of A_n and B_n are

$$\begin{aligned} A_n(t+1) &= \frac{A_{n-1}(t) + B_{n-1}(t)}{\sqrt{2}} \\ B_n(t+1) &= \frac{A_{n+1}(t) - B_{n+1}(t)}{\sqrt{2}} \end{aligned} \tag{3.13}$$

At this point we can appropriately choose the initial conditions $A_n(0)$ and $B_n(0)$ for $n \neq 0$, corresponding to the center of the line at time $t = 0$, and calculate the probability distribution using the formula

$$p(t, n) = |A_n(t)|^2 + |B_n(t)|^2 \tag{3.14}$$

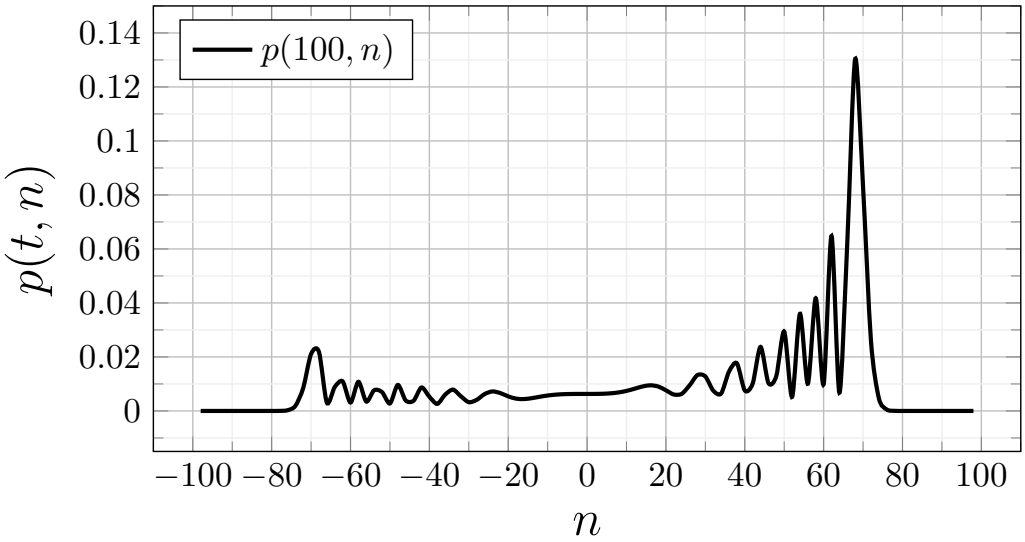


Figure 3.4: Asymmetric Quantum Random Walk example

Plotting the probability distribution of this quantum walk yields a very different behaviour compared to its classic equivalent. First off, the distribution is not concentrated in the central points. The curve is relatively flat in that area while spiking toward the edges. As it turns out, for any value of t the probability distribution reaches its maximum at around $t/\sqrt{2}$.

Looking at the distribution relative to our example (Figure 3.4), the curve is asymmetric, specifically it has a tendency to go right since there are higher probabilities around that area. The Hadamard coin is non-biased when applied to $|0\rangle$, but there is a sign difference on the generated quantum components when applied to $|1\rangle$. The result is that the rightward components (relative to direction $|0\rangle$) never meet destructive interference whereas the leftward ones are propagated half of the times with a negative sign, leading to cancellations.

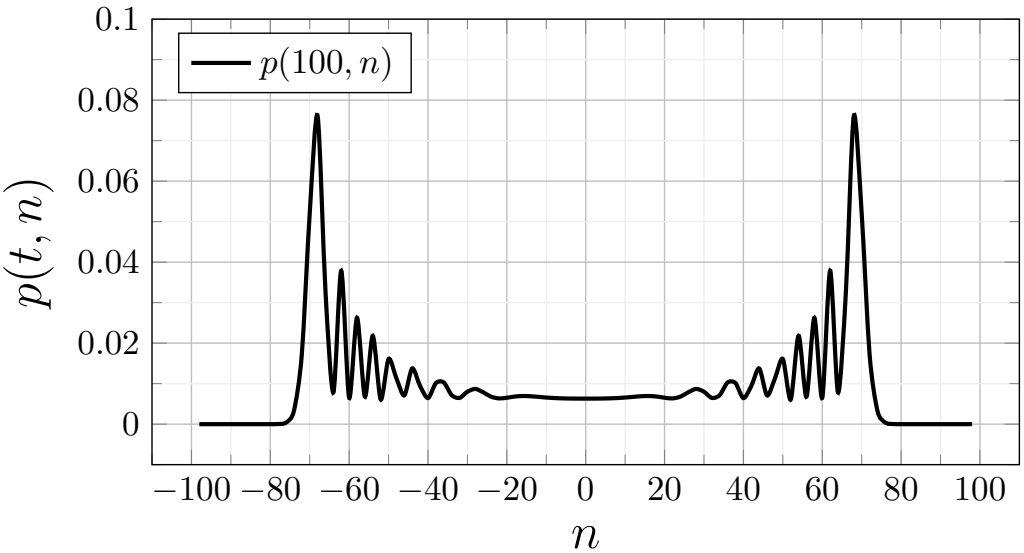


Figure 3.5: Symmetric Quantum Random Walk example

Two solutions are possible to obtain a *symmetric* walk (Figure 3.5): either we keep the coin and change the initial position/direction (for example the initial state $|\psi(0)\rangle = \frac{|0\rangle - i|1\rangle}{\sqrt{2}}|n=0\rangle$ will yield a symmetric distribution when using a normal Hadamard coin since it solves the sign difference) or we use a symmetric coin. A symmetric walk will have the same *expected position* as the classic equivalent $\mathbb{E}_n = \sum_{n=-\infty}^{\infty} n\, p(t,n) = 0$ but a very different *standard deviation*: $\sigma(t) = \sqrt{\sum_{n=-\infty}^{\infty} n^2\, p(t,n)} = 0.54t$.

This is exactly the advantage that characterizes quantum machines. These weird phenomena that rule this world turn out to behave in extremely convenient ways in the right settings. Search problems are one of them. This linear dependence of the position standard deviation against time is an asymptotical improvement compared to the classic random walk. Via a single procedure, a quantum random walk not only explores both directions on the line, but on average at a quadratically faster speed ($\sigma(t) = \sqrt{t}$ vs. $\sigma(t) = 0.54t$) (Figure 3.6). This means that given the same number of time steps, the quantum exploration on average will move quadratically farther away from the initial position.

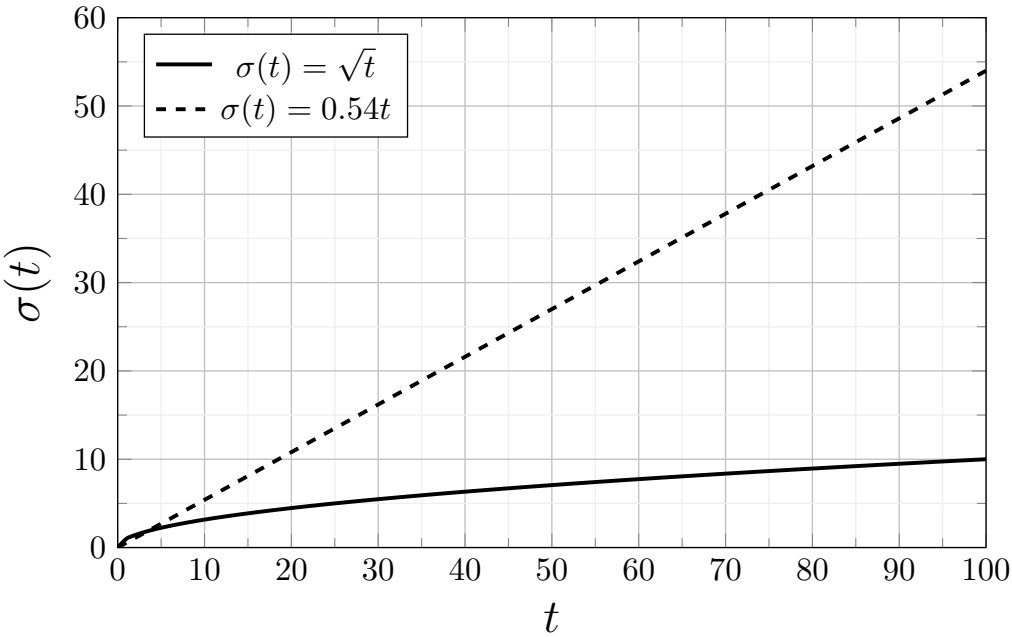


Figure 3.6: Difference between Classical and Quantum Walk standard deviations

3.2. Search Problems

A search problem, from its most general definition to the most structured one, has a fundamental role in computer science. In this section we briefly describe what a search problem is using an approach based on quantum algorithms and showing how quantum walk search can achieve a significant advantage. Subsequently, we will show an example of quantum walks on a more structured search problem where information about the search space can be used to improve the exploration.

3.2.1. Definitions

Generally speaking a search problem can be defined as finding a subset M for which each element x^* satisfies a given boolean predicate $P(x)$ in some set $x \in X$. In the generic case where no information is given in advance on the search space, the naive classical strategy is to check the condition $P(x)$ over the entire set of elements X expecting to find the right element in $\mathcal{O}(Z/N)$ queries where $|X| = N$ and $|M| = Z$. In a classical setting this task can be implemented in various ways, for example by differentiating the entries of a table or nodes in a tree, but the key problem is how to effectively differentiate the desired elements in the initial set.

Suppose f is a function with domain $\{0, \ldots, N-1\}$ where $N = 2^n$ with n some positive

integer.

$$f(x) = \begin{cases} 1 & \text{if } x = x^* \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

In a classical circuit the function f is implemented via a set of logic gates as stated in the previous chapter, however in a quantum setting we need some special unitary operator O that is able to compute f. Following the model of Equation (2.21) where the domain and the co-domain of f are mapped onto two Hilbert spaces $|x, y\rangle \mapsto |x, y \oplus f(x)\rangle$ we can initialize the y register in superposition using an X gate followed by a Hadamard gate:

$$|x\rangle |0\rangle \xrightarrow{X} |x\rangle |1\rangle \xrightarrow{H} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |x\rangle |-\rangle \tag{3.16}$$

Applying the operator O to the system we derive:

$$O |x\rangle |-\rangle = |x\rangle |- \oplus f(x)\rangle \tag{3.17}$$

If x is not the solution of the function nothing changes, otherwise we get a qubit flip on the second register. From a different point of view the new operator applies a phase shift of π on the global state that depends on the result of f.

$$|x\rangle |- \oplus f(x)\rangle = (-1)^{f(x)} |x\rangle |-\rangle \tag{3.18}$$

Notice how the second register is not affected by the result of f so it can be simplified in the expression obtaining a compact representation

$$(-1)^{f(x)} |x\rangle \tag{3.19}$$

In the literature, the operator O (Figure 3.7) is called **oracle** and it is the key ingredient for any search algorithm. Therefore the oracle operator can be used to mark the state that corresponds to the solution or a set of them of the original problem.

Grover's algorithm [14] is a search algorithm originally designed, given an unsorted collection of elements and a function, to look for the elements whose image of the given function is 1. It has been shown that this algorithm is optimal (up to a multiplicative constant), meaning no other algorithm will ever achieve better performance when solving this exact problem. With N elements in the search space, it takes $O(\sqrt{N})$ steps to find the marked element with high probability using $O(\log(N))$ storage space. More specifically,

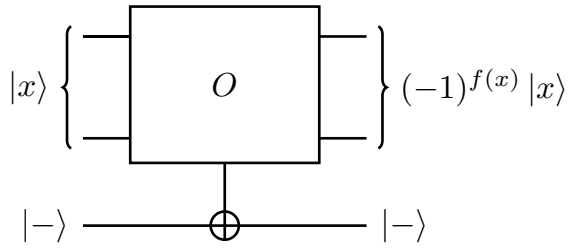


Figure 3.7: High level description of black-box oracle

Grover's algorithm solves the problem by querying the oracle f $\lfloor \frac{\pi}{4}\sqrt{N} \rfloor$ times.

The goal here is to find x^* with as little queries to the oracle as possible, thus reducing as much as possible the steps of the algorithm. Our f now is realized via the use of two registers in the computation: the first stores the domain elements whereas the second their images. We call this operator O

$$O|x\rangle|i\rangle = |x\rangle|i \oplus f(x)\rangle \tag{3.20}$$

where $\oplus$ is a bitwise xor operation.
We can easily understand how this operator behaves

$$O|x\rangle|0\rangle = \begin{cases} |x^*\rangle|1\rangle & \text{if } x = x^* \\ |x\rangle|0\rangle & \text{otherwise} \end{cases} \tag{3.21}$$

Since we have no prior knowledge of where the marked element is within the database, we need to create a superposition of all possible states of the first register. That is why we define the uniform superposition $|D_N\rangle$ as our search space.

$$|D_N\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \tag{3.22}$$

The second unitary operator accomplishes the following:

$$G = (2|D_N\rangle\langle D_N| - I_N) \otimes I_2 \tag{3.23}$$

where I_2 is the identity matrix of dimension 2.

The operator G usually, defined as Grover Diffuser, carries out a procedure called **Amplitude Amplification**. This is the centerpiece of the quantum advantage regarding

search problems. Once the oracle has inverted the sign of the marked element, this procedure amplifies its amplitude while at the same time shrinking all the other amplitudes (the norm has to be preserved, increasing an amplitude forces the others to decrease in modulus). An example with three qubits is shown in Figure 3.8

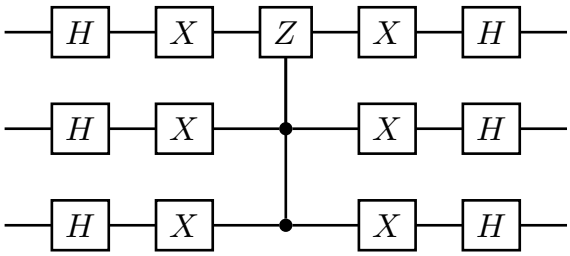
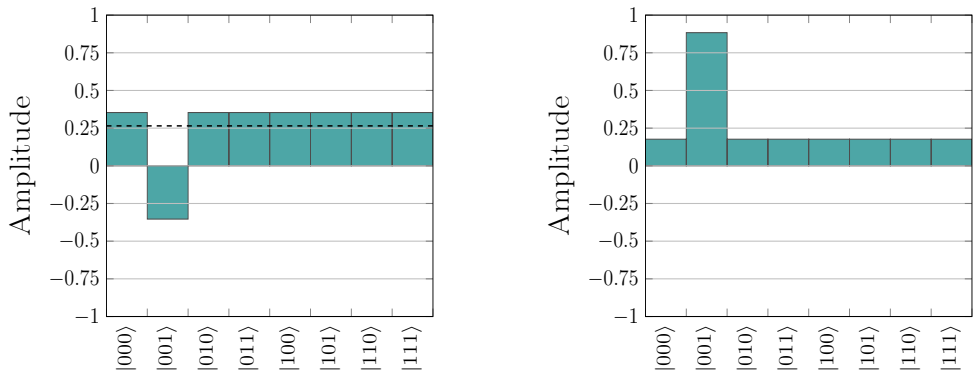


Figure 3.8: Grover Diffuser example (G)

After the oracle's application, the marked element has negative amplitude, which means that the average is slightly lower than the amplitude of a non-marked element. Mathematically, the Diffuser performs a reflection about average over the whole state. Consequently, non-marked elements will be subjected to a shrunk of their amplitudes, whereas the marked element amplitude will be amplified. The following graphs show this in action.



(a) After the Oracle application (b) After the Diffuser application

Figure 3.9: Amplitude Amplification
The dashed line shows the average of the amplitudes after the Oracle application

Now one step of the algorithm is first a query to the oracle O, then the application of the G operator. This can be considered the *evolution operator* as defined in (3.7), thus we will give it the same symbol

$$U = G\,O \tag{3.24}$$

Before the iterative application of U, the initial condition is

$$|\psi_0\rangle = |D_N\rangle\,|-\rangle \tag{3.25}$$

Finally, measuring the first register will yield x^* with probability greater than $1 - \frac{1}{N}$.

Algorithm 12 Grover's algorithm.

Input: N and f defined in (3.15), a 2-register quantum computer with $n+1$ qubits
Output: x^*, the marked element with probability greater than or equal to $1 - \frac{1}{N}$

1: Prepare the initial state $|D_N\rangle\,|-\rangle$
2: Apply U^t where $t = \lfloor \frac{\pi}{4}\sqrt{N} \rfloor$ and U given by (3.24)
3: Measure the first register in the computational basis

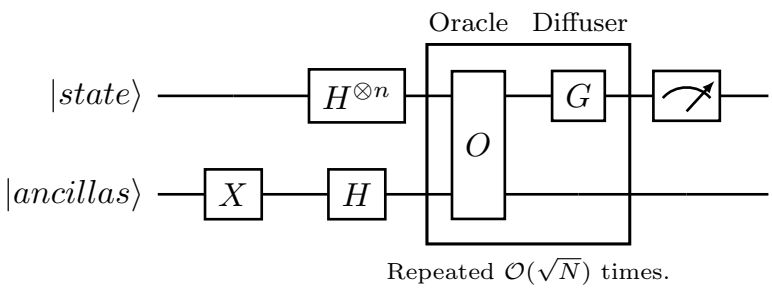


Figure 3.10: A generic Grover's algorithm circuit

Geometrically speaking, O and G can be considered *reflection operators*.
O is a reflection around the vector space orthogonal to the vector space spanned by $|x^*\rangle\,|-\rangle$. Taking from equation (3.21), this means that

$$\begin{aligned} O\,|x^*\rangle\,|-\rangle &= \frac{O\,|x^*\rangle\,|0\rangle - O\,|x^*\rangle\,|1\rangle}{\sqrt{2}} \\ &= \frac{|x^*\rangle\,|0\rangle - |x^*\rangle\,|1\rangle}{\sqrt{2}} \\ &= -\,|x^*\rangle\,|-\rangle \end{aligned} \tag{3.26}$$

If we apply the same definition on the on a non-marked element we obtain $O\,|x\rangle\,|-\rangle = |x\rangle\,|-\rangle$ for $x \neq x^*$. An application of O on a linear combination with real coefficients of $|x^*\rangle\,|-\rangle$ with a vector orthogonal to $|x^*\rangle\,|-\rangle$ inverts the sign of component $|x^*\rangle\,|-\rangle$ while preserving the sign of the component orthogonal to $|x^*\rangle\,|-\rangle$.
In a similar fashion, the second operator, G, is also a reflection but around the vector space spanned by $|D_N\rangle$, implying $G\,|D_N\rangle\,|-\rangle = |D_N\rangle\,|-\rangle$ from equation (3.23). Again,

if G is applied to a linear combination with real coefficients of $|D_N\rangle\,|-\rangle$ with a vector orthogonal to $|D_N\rangle\,|-\rangle$, it inverts the sign of the latter while preserving the former.

Summing everything up, starting from the initial condition $|D_N\rangle$ one application of U rotates $|D_N\rangle$, the uniform superposition, approximately by $\frac{2}{\sqrt{N}}$ radians towards $|x^*\rangle$. Since $|D_N\rangle$ and $|x^*\rangle$ are almost orthogonal when N is large, the strategy of the algorithm is to rotate the initial condition by $\frac{\pi}{2}$ radians before applying the measurement. This transformation is repeated enough times to achieve the closest position as possible to $|x^*\rangle$, which turns out to be $t_f = \lfloor \frac{\pi}{4}\sqrt{N} \rfloor$. Excessive repetitions of U may not only lead to the overshoot of the marked element state, but to a sinusoidal behaviour of the circuit. In the end, the smaller the angle between the final state and the marked state, the higher the probability of measuring x^* is.

This same methodology of checking a certain condition with the oracle and then modifying the amplitudes of the state space via the Diffuser can be applied in a similar fashion on the Quantum walks paradigm. The evolution of both algorithms is tied to first an application of some sort of marking operator followed by an update of the state space controlled by the marking. The next Section will explain this relationship in detail.

3.2.2. Quantum Random Walk Search

In this section we show how a Quantum random walk can be applied to a search problem by presenting the model proposed by Shenvi, Kempe and Whaley in [34], formalising the general structure and highlighting its main properties. This procedure is called SKW algorithm and follows the model known in literature as coined Quantum walk search.

In order to create a search algorithm using Quantum random walks we need a useful representation of the search problem. We have already explained in Section 3.1.2 how Markov Chains work. To give a quick recap, $G(X, E)$ is a graph with a set of vertices $X = \{x_1, \ldots, x_n\}$ ($|X| = n$) and a set of edges E. Even though our description was built upon the definition of a classical one, this framework can efficiently describe Quantum random walks as well. Specifically for search problems, a graph can be considered a representation of a structured database that contains one or more marked elements. Our goal is to modify the probability distribution of the Markov chain associated to the graph such that the final measurement will yield, with reasonable probability, the marked element(s) of the graph.
In essence, the principles we are going to explain can be considered an extension of Subsection 3.1.3, with a slight difference in the functioning of the coin.

Similarly to Grover's algorithm, one step of this procedure begins with the application of

an oracle operator O whose task is to understand if the current state $|x\rangle$ satisfies a given property. The result of such interrogation is stored on additional qubit(s) which are then used to control the coin application. In other words, the search problem that we want to solve is encoded via a differentiation of the choice of the coin that depends on the oracle: on marked vertices, those that satisfy the oracle condition, one kind of coin is applied whereas a different one is applied to non-marked vertices. Figure 3.11 shows the choice of which coin to apply based on the output of the checked condition: the coin C_1 is applied if $|x\rangle$ respects the condition, otherwise C_0 is applied.

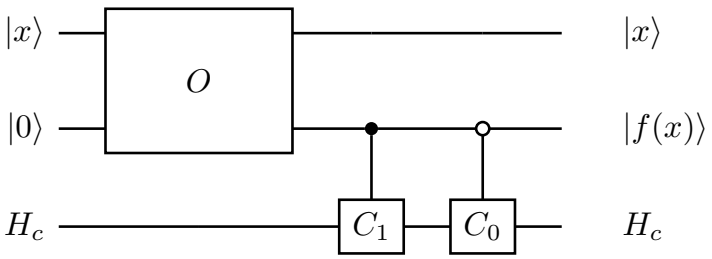


Figure 3.11: Depending on the oracle result, apply C_1 or C_0. (H_C is the coin register)

To proceed with the algorithm formalization we need to chose a proper coin operator for C_0 and C_1. Usually the choice for C_0 is the Grover diffuser operator (Equation (3.23)) as it distributes the same amplitude on all the n directions, meaning the probability of choosing a given direction is the same for all possible directions. It is defined as:

$$C_0 = G = (2\,|D_C\rangle\,\langle D_C| - I) \tag{3.27}$$

where $|D_C\rangle$ follows the same concept of 3.22 but is the equal superposition over all directions.

For the choice of C_1, we consider the case $C_1 = -I$ which is also called SKW coin. Therefore the effect of the two operator C_0 and C_1 combined into the new global coin C' is:

$$\begin{aligned} C' &= C_0 \otimes I + (C_1 - C_0) \otimes |x^*\rangle\,\langle x^*| \\ &= G \otimes I + (-I - G) \otimes |x^*\rangle\,\langle x^*| \end{aligned} \tag{3.28}$$

Following from the definition in Equation (3.7), we derive the new evolution operator U'

$$\begin{aligned} U' &= SC' = S(G \otimes I - (I + G) \otimes |x^*\rangle \langle x^*|) \\ &= S((2|D_C\rangle \langle D_C| - I) \otimes I - (I + (2|D_C\rangle \langle D_C| - I)) \otimes |x^*\rangle \langle x^*|) \\ &= U \otimes I - 2S(|D_C\rangle \langle D_C|) \otimes |x^*\rangle \langle x^*| \end{aligned} \tag{3.29}$$

The pseudo code of the SKW algorithm is proposed in Algorithm 13.

Algorithm 13 SKW's algorithm.

Input: N and f defined in (3.15), a 2-register quantum computer with $n + d$ qubits

Output: x^*, the marked element with probability greater than or equal to $\frac{1}{2} - \frac{1}{N}$

1: Prepare the initial state $|D_N\rangle |D_C\rangle$

2: Apply $(U')^t$ where $t = \lfloor \frac{\pi}{2}\sqrt{N} \rfloor$ and U' given by (3.29)

3: Measure the first register in the computational basis

We show what the most generic circuit for a Quantum random walk search looks like in Figure 3.12.

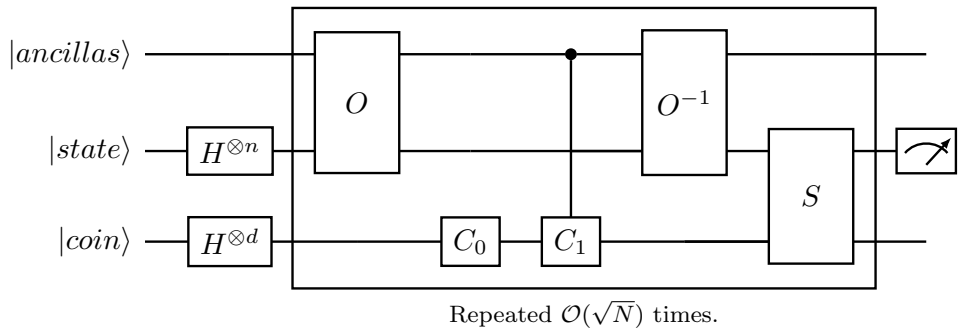


Figure 3.12: Generic Quantum random walk search circuit

The oracle is applied again, but reversed, after the coin control for uncomputation purposes. This is done in order to have the ancilla qubits ready for the next step, whose oracle needs to have clean (i.e. at base state $|0\rangle$) qubits to properly control the coin again.

Notice that it is also possible, as shown in Figure 3.12, to apply the two coins differently than explained before: either they are both controlled by the oracle's output(s) qubit(s) (Figure 3.11) or only one is. The coin that is specifically applied when the marked vertex is found has to be always controlled. The second coin instead can either be controlled, meaning applied when the vertex is not marked, or non-controlled. In other words, this

second solution first applies a coin to every vertex of the graph then immediately applies a second one for the marked vertex. With an appropriate selection of coins, these two methods are mathematically equivalent.

In addition to the SKW coin defined in Equation (3.28), another solution is also possible: reapplying again the Grover diffuser operator as C_1. The new Equation for C' is now:

$$C' = G \otimes I + G \otimes |x^*\rangle \langle x^*| \tag{3.30}$$

The two options are shown in Figure 3.15 and yield a different behaviour regarding the probability of measuring the marked vertex at the end of the circuit.

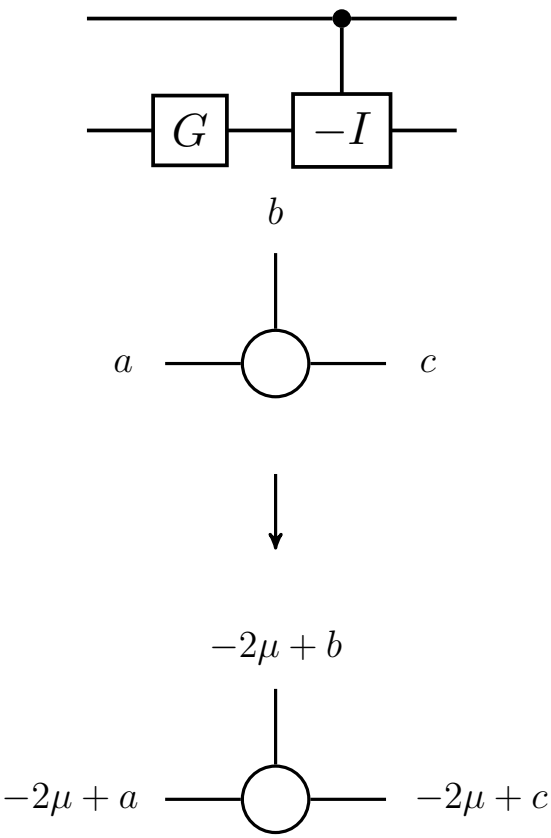


Figure 3.13: G + SKW marking coin

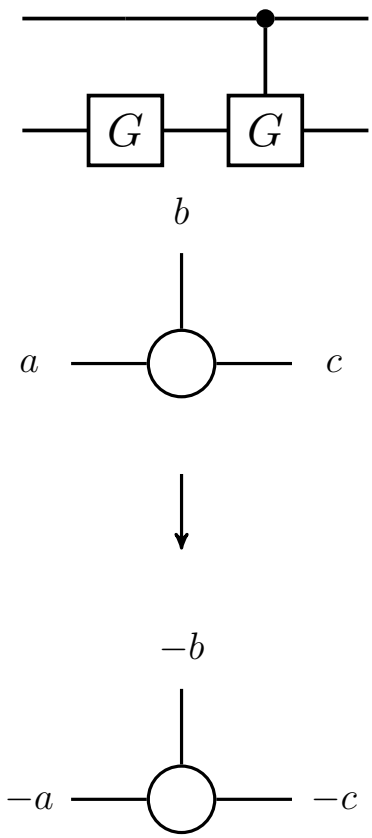


Figure 3.14: G + Grover marking coin

Figure 3.15: Effects of the two different marking coins on a generic vertex with $degree = 3$ (where $\mu = \frac{a+b+c}{3}$)

Later in our work, we will show this difference by looking at the simulated probabilities regarding the exploration of the Johnson graph (Figure 3.29).

3.2.3. Costs

In order to analyze the performance of Quantum Random walks algorithms in solving search problems, Magniez et al. [20] define the following cost system:

- T_s is the cost of SETUP, it is the cost of sampling the first vector and initializing the data structure.
- T_c is the cost of CHECK, it represents the oracle cost.
- T_u is the cost of UPDATE, it is the cost of sampling a neighbour and updating the data structure.

This is the most generic cost definition possible, as is it considers the data structure that is chosen to navigate the graph as generic.

The formula for the total cost of a Quantum walk search is

$$T_s + \frac{1}{\sqrt{\varepsilon}}\left(T_c + \frac{1}{\sqrt{\delta}}T_u\right) \tag{3.31}$$

We need to define parameters ε and δ. In Subsection 3.2.1 we specified as $x^* \in M$ the subset of elements in some set $x \in X$ that respect a certain condition $P(x)$. If $\|X\| = N$ and $\|M\| = Z$, we define as $\varepsilon = Z/N$ the ratio of the number of these elements w.r.t the total number of elements in the search space. This is related both to the oracle costs well as the update cost: the higher this ratio the more oracle calls as well as data structure updates the search will have to perform.
The second parameter we need is the spectral gap of the transition matrix M defined as $\delta := 1 - \max_{i>1} |\lambda_i|$ where $(\lambda_i)_{i>1}$ are the eigenvalues of M not equal to 1. It represents how many times the update of the data structure has to take place.

3.2.4. Hypercube example

In order to clarify the concept described so far, we show an implementation example. As search space we use a four dimensional hypercube i.e. a regular graph with $|V| = 2^n$ nodes an degree equal to $d = n$ where $n = 4$.
Each node $v_i \in V$ can be labeled with a n-bit binary string and two nodes v_i and v_j are connected by an edge if their Hamming distance is 1 i.e. $dist(v_i, v_j) = 1$. To set up the SKW algorithm we need a $\mathcal{H}^n$ coin space to encode all the directions and a $\mathcal{H}^{2^n}$ node space for the vertices.

The computational basis is

$$\{|d, v\rangle, d \in D = \{00, 01, 10, 11\}, v \in V = \{0000, 0001, \cdots, 1111\}\} \tag{3.32}$$

The value of the coin computational basis d determines the direction to follow and every edge of a node is associated with a value of d. In this example if $d = 00$ the walker will move to the node where the first binary value differs from the current node, if $d = 01$ it will move to the node where the second binary value differs from the current node and so on.

For the shift S operator we can exploit the symmetric structure of the hypercube, it maps $|d, v\rangle$ onto $|d, v \oplus e_d\rangle$ where e_d is the d-basis of the hypercube.

$$S = \sum_{d=1}^{n} \sum_{v=1}^{2^n} |d, v \oplus e_d\rangle \langle d, v| \tag{3.33}$$

We assume that the marked node is the one with the label $v = 1011$ and using the evolution operator U' defined in Equation 3.29 we are able to perform the SKW algorithm. The quantum circuit implementation is proposed in Figure 3.16

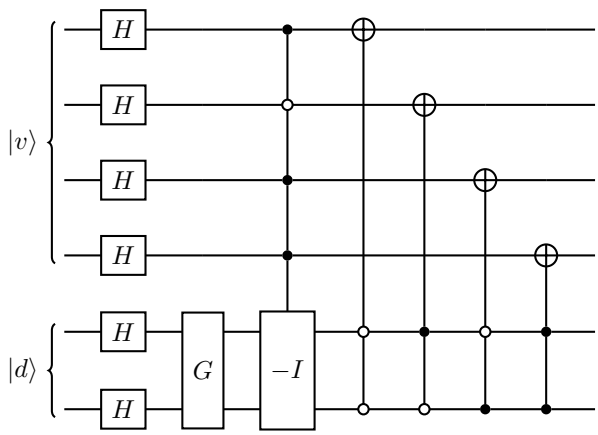
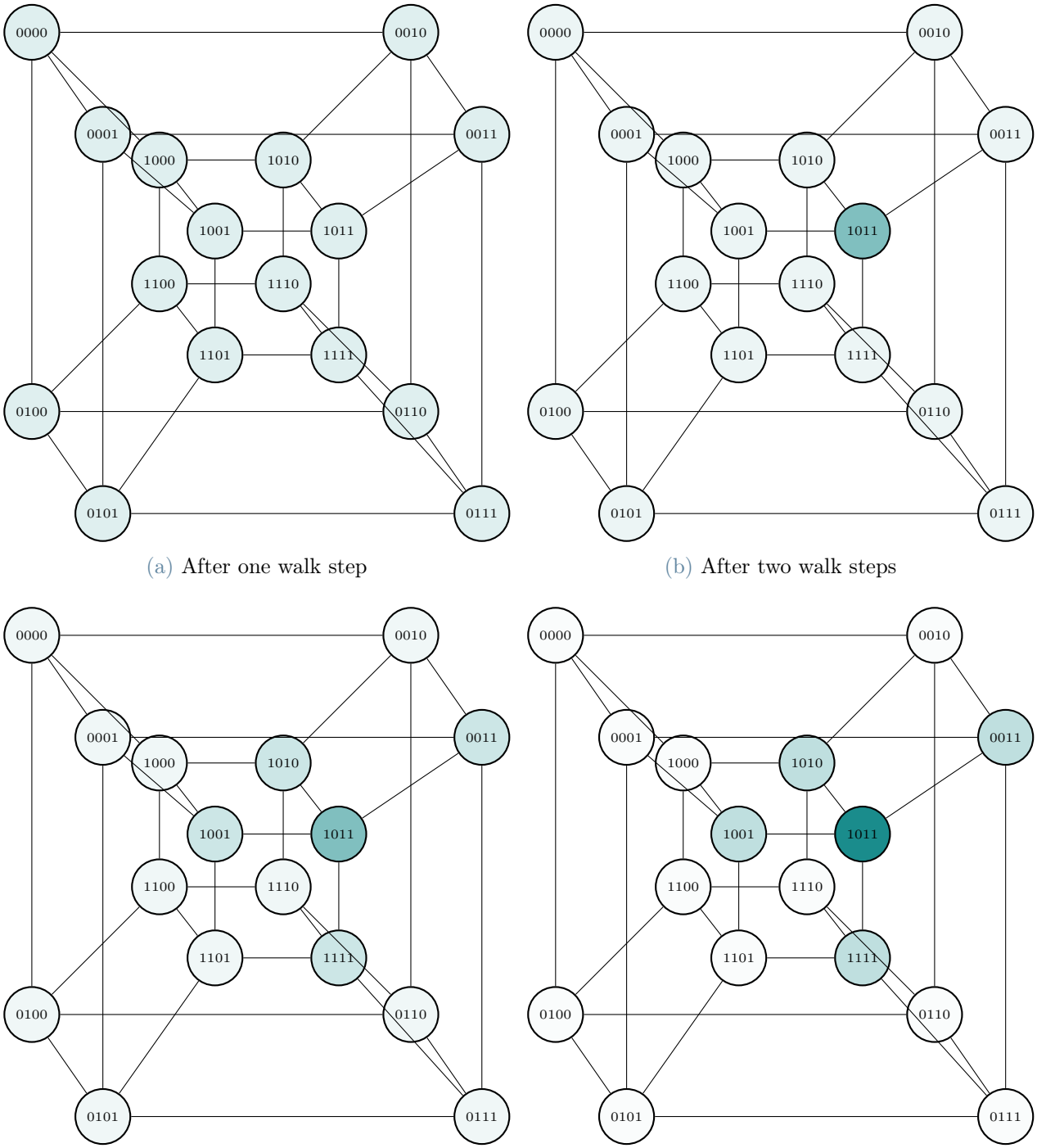


Figure 3.16: Implementation of one step of SWK algorithm on 4-dimension hypercube and marked node $v = 1011$



(a) After one walk step

(b) After two walk steps

(c) After three walk steps

(d) After four walk steps

Figure 3.17: Hypercube quantum random walk example with marked vertex 1011. From the initial superposition we reach the marked element. Darker colors indicate higher probabilites

3.3. Johnson Graphs

So far, the definitions we have given hold true for any generic graph, but historically speaking some graphs have proven to be more useful than others. We first explain why Johnson graphs are particularly interesting when explored by a Quantum random walk, followed by our example of implementation. A great write-up about Quantum random walks on this kind of graphs can be found in [37].

Let $G(V, E)$ be the Johnson graph $J(n, k)$ where V is the set of vertices and E is the set of edges. V is the set of k-subsets of $[n]$=$\{1,2,\ldots,n\}$, and two vertices $v, v' \in V$ are adjacent if and only if $|v \cap v'| = k - 1$. The number of vertices is $\binom{n}{k}$. A $J(n, k)$ Johnson graph is d-regular, where d is the degree equal to $d = k(n - k)$. The following Figure shows an example with $n = 4$ and $k = 2$.

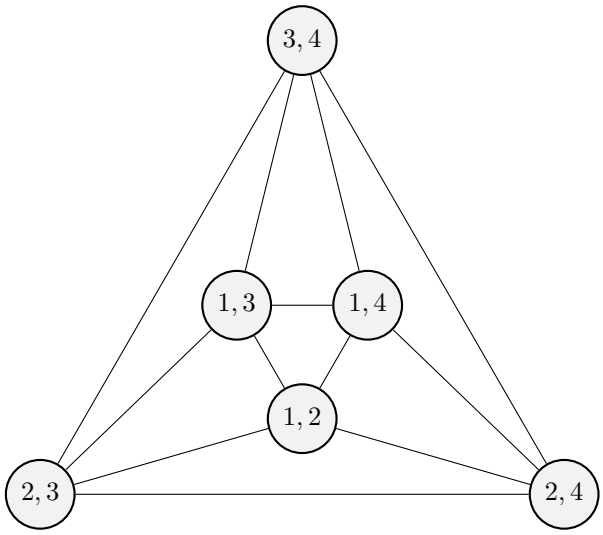


Figure 3.18: Johnson graph $J(4, 2)$ with $[n] = \{1, 2, 3, 4\}$

There are a few properties that favor Johnson graphs in a search problem. First of all, *vertex-transitivity*.

Definition 3.3 (Vertex-transitive Graph). *A graph G is vertex-transitive if, given any two vertices v_1 and v_2 of G, there is some automorphism*

$$f : G \to G \qquad s.t. \qquad f(v_1) = v_2$$

The computational complexity of the search problem in vertex-transitive graph is simpler because independent of the location of the marked vertex. Johnson graphs are not only vertex-transitive but also *distance-transitive*.

Definition 3.4 (Distance-transitive Graph). *A distance-transitive graph is a graph such that, given any two vertices v_1 and v_2 at any distance d and any other two vertices v_3 and v_4 at the same distance d, there is some automorphism that carries v_1 to v_3 and v_2 to v_4.*

This property can be used to find an invariant subspace of the Hilbert space that helps in the calculation of the computational complexity, meaning the vertices can be partitioned into subsets of vertices that have the same distance to the marked vertex. Tanaka et al. in [37] give an analytical explanation of the importance of these properties. Also, an extensive analysis of Johnson graphs specifically tied to their distance-transitivity can be found in [41].

The position-coin notation we have explained in Section 3.1.3 holds true for this kind of graph as well: the Hilbert space is $\mathcal{H}_V \otimes \mathcal{H}_C$ where $\mathcal{H}_V = \{|v\rangle : v \in V\}$ and $\mathcal{H}_C = \{|c\rangle : 1 \leq c \leq d\}$. To achieve the walk evolution, we apply the double coin definition as explained in Equation (3.28), aware of the fact that the coin space is equal to the degree. A first coin C_1 either applies the negation of the d-dimensional identity operator $(-I_d)$ or a d-dimensional Grover coin G on the marked vertex whereas the d-dimensional Grover coin G is applied to unmarked ones, as shown in Figure 3.15.

The final state after t applications of the appropriately modified evolution operator $U = S\,C$ on the initial state $|\psi(0)\rangle = \frac{1}{\sqrt{dV}} \sum_{c=0}^{d-1} \sum_{v=0}^{V-1} |c, v\rangle$ is $|\psi(t)\rangle = U^{t_{run}} |\psi(0)\rangle$ with

$$t_{run} = \left\lfloor \frac{\pi n^{k/2}}{2\sqrt{2k!}} \right\rfloor \approx \frac{\pi\sqrt{N}}{2\sqrt{2}} \tag{3.34}$$

from [37]. We present a circuital implementation of a walk in this kind of graphs.

3.4. Circuit Implementation

Here we show our approach that performs a coined quantum walk search on a Johnson graph. In this example a $J(4, 2)$ is used, however the implementation can be extended to any value of the parameters n and k.

3.4.1. Encoding

In order to be able to explore the graph we need to have a working shifting phase. The first step is to use a suitable encoding to represent the graph. Each node can be labelled with a binary string whose values express whether an element of n is present in the subset k represented by the node. All vertices' labels are n-sized binary strings with exactly k

bits set to 1.
For instance the encoding 1010 means that the node represents the set k with elements in positions p_0 and p_2 of n i.e. the values 0 and 2. The binary encoding graph is presented in Figure 3.19.

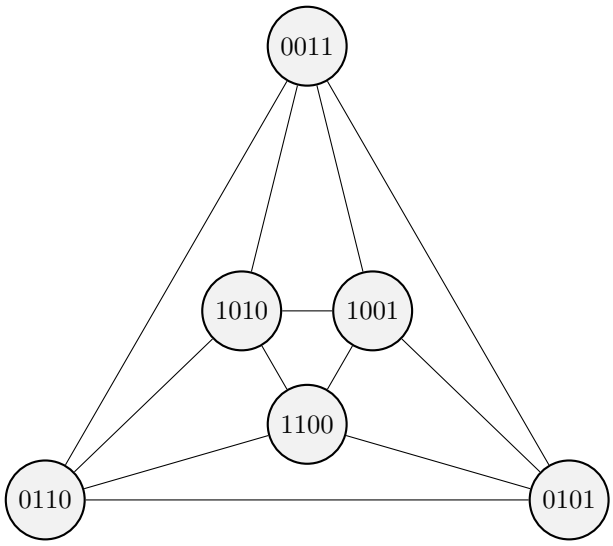


Figure 3.19: Johnson graph $J(4,2)$ with binary labels

In this example we give an asymptotic analysis of the depth of the circuit and the number of qubits employed as a function of the parameters n and k of the Johnson graph, assuming unitary cost for each gate.

3.4.2. Initialization

In order to initialize a Quantum walk on this graph we need to set up a proper superposition that represents all possible vertices in a generic $J(n,k)$ Johnson graph. This is known in literature as Dicke state $|D_k^n\rangle$: it is an equal-weight superposition of all n-qubit states with Hamming weight k (i.e. all strings of length n with exactly k ones over a binary alphabet). It is defined as:

$$|D_k^n\rangle = \binom{n}{k}^{-\frac{1}{2}} \sum_{wt(x)=k} |x\rangle \qquad x \in \{0,1\}^n \tag{3.35}$$

where $wt()$ is the Hamming weight (Definition 1.8).
We implemented in our solution the work done by Bärtschi and Eidenbenz in [3], further improved by Mukherjee et al. in [23]. Because of the fact that in our example we chose to use the $J(4,2)$ graph, we show the circuit that creates the $|D_2^4\rangle$ superposition.

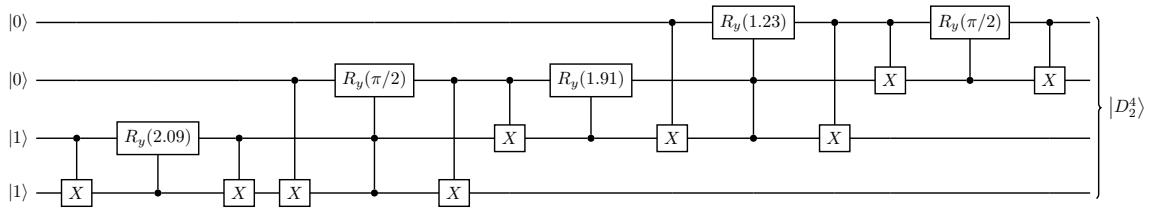


Figure 3.20: $|D_2^4\rangle$ Dicke state circuit

The algorithm is based on the observation that Dicke states can be built inductively according to the formula

$$|D_k^n\rangle = \sqrt{\frac{k}{n}}\,|D_{k-1}^{n-1}\rangle\,|1\rangle + \sqrt{\frac{n-k}{n}}\,|D_k^{n-1}\rangle\,|0\rangle \tag{3.36}$$

This means that the algorithm achieves a specific superposition $|D_k^n\rangle$ by expanding upon the gate sequence that defines a Dicke state with smaller values of n and/or k.

The final state of our example equals to

$$|D_2^4\rangle = \frac{1}{\sqrt{6}}(|1100\rangle + |0011\rangle + |1010\rangle + |0101\rangle + |0110\rangle + |1001\rangle) \tag{3.37}$$

This solution is quite efficient, in that it achieves the generic Dicke state $|D_k^n\rangle$ in depth $O(n)$ with gate number of $O(n+k^2)$ for both **X** and **Ry** gates. More specifically we have

$$\mathbf{CNOT}_{Dicke} = 5nk - 5k^2 - 2n \tag{3.38}$$

$$\mathbf{Ry}_{Dicke} = 4nk - 4k^2 - 2n + 1 \tag{3.39}$$

in our example $|D_2^4\rangle$ this means that we have 40 **CNOT** gates and 9 **Ry** gates.

To initialize the walk to the desired superposition $|\psi(0)\rangle = \frac{1}{\sqrt{dV}}\sum_{c=0}^{d-1}\sum_{v=0}^{V-1}|c,v\rangle$ we apply the circuit in Figure 3.20 on register $|v\rangle$. This register will, from now on, represent the current position of the walker on the Johnson graph and will be called state register. Secondly, we apply an Hadamard gate on each qubit of a second register $|c\rangle$ to obtain an uniform superposition of all the possible base states. We call $|c\rangle$ the coin register, it will perform the coin toss needed to explore the graph.

We show the initialization part of the circuit in Figure 3.21.

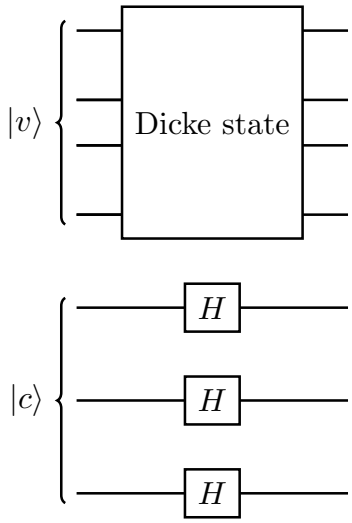


Figure 3.21: Initialization of $J(4, 2)$

3.4.3. Coin

The application of the coin follows the same structure of Figure 3.15. Like we did in the Hypercube example, the coin stage also embeds the functioning of the oracle that directly controls the marking coin C_1. We assume that the marked vertex has a specific label, in this case 1100. Figure 3.22 shows the part of the circuit that implements the coin stage.

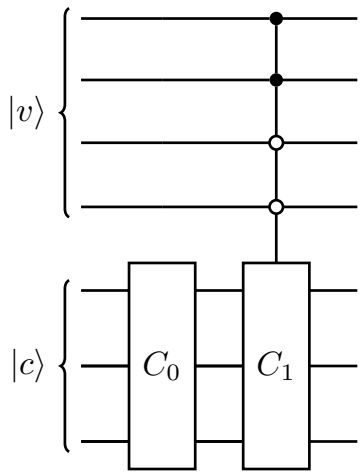


Figure 3.22: Coin application of $J(4, 2)$ with marked vertex 1100

3.4.4. Shift

In the setting of coined quantum walks, finding the most suited way of performing a shift in a graph is of great interest as it is the sub-procedure that most conditions the performance of the walk. Because of the encoding of the Johnson graph, the key element in achieving this operation is the swap gate.

The shift gate, the circuit representation of the shift operator, is made by using a series of swap gates placed in a proper order. The swap gate in Figure 3.23 performs a swap of the states between two qubit.

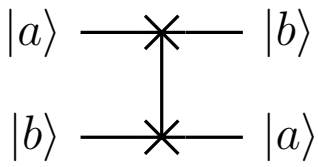
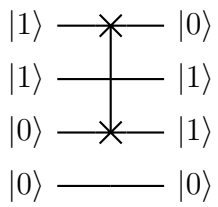
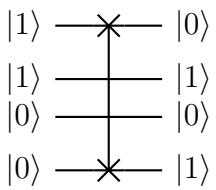


Figure 3.23: The swap gate between two qubit

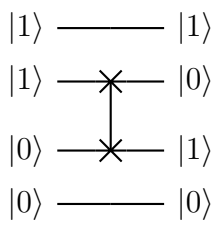
For instance if the walker is in vertex with the label 1100, by taking a step in one direction it will reach one of the possible neighbours among $\{1010, 1001, 0101, 0110\}$. A step or shift is realised by swapping two qubit of the state register with opposite values i.e. a qubit in state 0 with a qubit in state 1 and viceversa, so that the Hamming distance (Definition 1.7) between one vertex and the next is preserved. An example of all possible shifts of vertex 1100 can be seen in Figure 3.24.



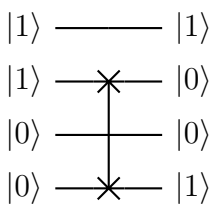
(a) First direction.



(b) Second direction.



(c) Third direction.



(d) Fourth direction.

Figure 3.24: Shifting from 1100 towards its neighbors

In a Johnson graph using binary label of n qubits there are $\binom{n}{2}$ possible applicable swaps but we have to discard those that do not meet the condition on the Hamming distance. Given that two nodes are adjacent if the sets they represent differ by only one element, the Hamming distance between the binary labels of two adjacent nodes is always fixed at 2. Indeed, only $k(n-k)$ swaps are valid. We can rephrase the shifting problem as finding the $k(n-k)$ permutations that preserve a Hamming distance of 2 from a given starting label.

For the sake of clarity all possible combination of swap for the $J(4,2)$ are reported in the Table 3.1 where a swap is expressed with the position (x_i, x_j) of the qubit to swap.

Qubit register base state	Valid permutations
1100 or 0011	(0,2),(0,3)(1,2),(1,3)
1010 or 0101	(0,1),(0,3),(1,2),(2,3)
1001 or 0110	(0,1),(0,2),(1,3),(2,3)

Table 3.1: $J(4,2)$ swap assignment

A first naive approach would be to encode such a shifting system by first analyzing where the walker is during the walk (i.e. the current node) followed by the $k(n-k)$ correct shift operation on the node. This is a solution we tried to implement but were not successful in. Specifically, we haven't been able to design a circuit capable of applying the sequence of correct swap operations controlled by the current node expressed in the state space without resorting to adding an exponential number of additional qubits. A procedure that relies on such amount of resources is not interesting with respect to current literature on quantum search algorithms.

However, this controlled procedure is not strictly necessary. We explain why it is possible to implement the same behaviour in a different way.

Let's assume we are in node 1100, the correct swaps are shown in Table 3.1. Now we also apply the swaps $(0,1)$ and $(2,3)$. The walker will not effectively perform any shift, because the new swaps produce labels with zero hamming distance from the label of the starting node, meaning the new label is identical to the starting one. In practice we introduce two self-loops on the node. If we repeat the same procedure for every node of the graph what we obtain is a modified version of the Johnson graph, shown in Figure 3.25.

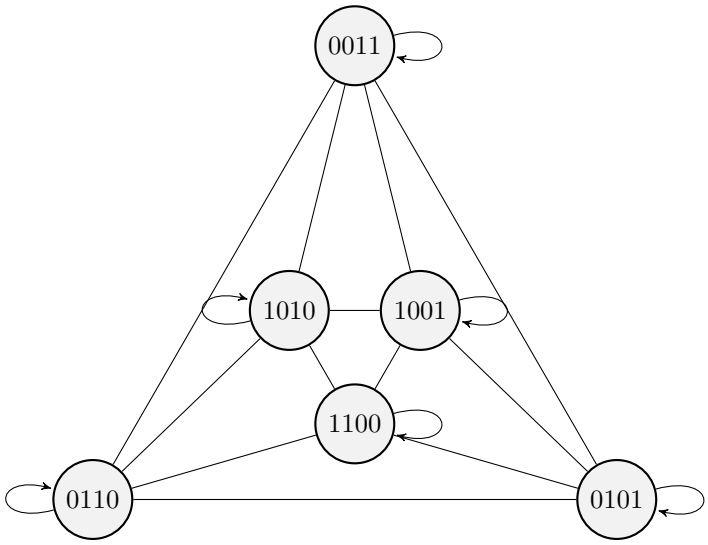


Figure 3.25: Johnson graph $J(4, 2)$ with binary labels and self loops

The single self-loop of each vertex shown in Figure 3.25 merges multiple directions. Now, the likelihood of performing a non-loop shift, meaning an actual changing of node labels, depends on n and k. Specifically, given any node, we have that:

- $k(n-k)$ directions are correct, meaning they encode a shift that when performed changes the state register to one of the neighbours of the current vertex.
- $\binom{n}{2} - k(n-k)$ directions are encoded in the self-loop, meaning this is the number of shifts that will not effectively change the current label. The shifting is performed but no actual label modification is done.

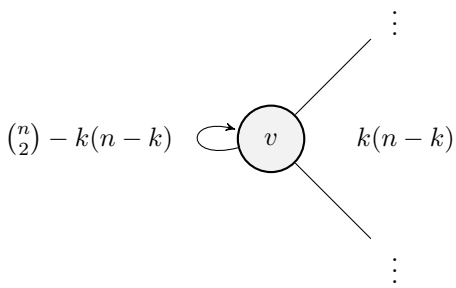


Figure 3.26: Single vertex analysis of the encoded directions.

The circuit representation of the shift operator with self-loops is shown in Figure 3.27.

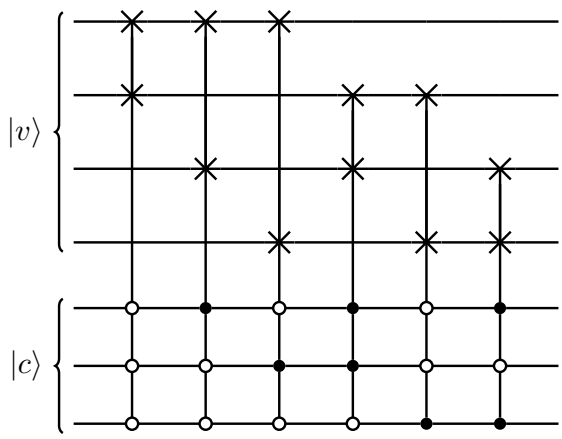


Figure 3.27: Shift circuit on a $J(4,2)$ with self loop

All the possible $\binom{n}{2}$ swaps modify the state register and are controlled by the coin register $|c\rangle$.

Regarding qubit number and depth, this shift operator has the following performance:

$$\text{qubit number}_{shift} = n + \lceil \log_2 \binom{n}{2} \rceil \tag{3.40}$$

$$\text{depth}_{shift} = \binom{n}{2} \tag{3.41}$$

3.4.5. Final Circuit

Considering a Johnson $J(n,k)$, our circuit requires the following two register:

- $|v\rangle$: state register is used to encode the vertices of the graph, every qubit corresponds to a position in the set n thus it has dimension $dim(|v\rangle) = n$
- $|c\rangle$: coin register is used to express all the possible directions of the coin. It has dimension $dim(|c\rangle) = \lceil \log_2 \binom{n}{2} \rceil$

The following Figure shows a complete exploration of a Johnson graph using our implementation (only showing 1 step of the walk).

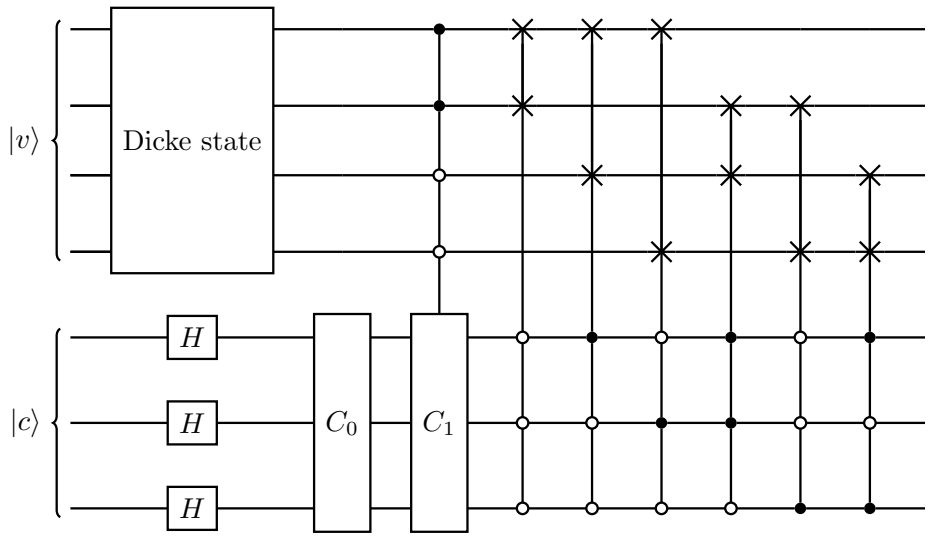


Figure 3.28: Complete Johnson graph walk search with marked vertex 1100 (1 step)

3.4.6. Lackadaisical Quantum Random Walks

Quantum random walks with self-loops have been extensively studied by Thomas Wong who coined the term Lackadaisical quantum random walks. To give a quick definition taken from [39], they are random walks that include an additional coin degree of freedom that encodes the self-loop. The generic Johnson graph's Hilbert space equals to $\mathcal{H}_V \otimes \mathcal{H}_C$ with $\mathcal{H}_V = \{|v\rangle : v \in V\}$ and $\mathcal{H}_C = \{1 \leq c \leq d\}$ where d is the degree of the Johnson graph plus 1.
The evolution operator U follows the equation:

$$U = S(I_V \otimes C_0)(-C_1 \otimes I_d) \tag{3.42}$$

where S is the shift operator and Q is the Grover oracle defined as:

$$C_1 = 2\,|x^*\rangle\,\langle x^*| - I_V \tag{3.43}$$

where $|x^*\rangle$ is the marked vertex. C_0 is the Grover diffusion coin for a weighted graph defined as:

$$C_0 = 2\,|s_c\rangle\,\langle s_c| - I_d \tag{3.44}$$

with

$$|s_c\rangle = \frac{1}{\sqrt{d+l}}(\sum_{c=1}^{d} |c\rangle + \sqrt{l}\,|\circlearrowleft\rangle) \tag{3.45}$$

where l is the weight associated to the self-loop. Following the reasoning expressed in Figure 3.15, U is equivalent to applying C_0 to unmarked vertices and $-C_1$ to the marked vertex, followed by the shift operation. A more detailed analysis regarding the search problem in vertex-transitive graphs is analyzed in [30], which details the following performance analysis parameters:

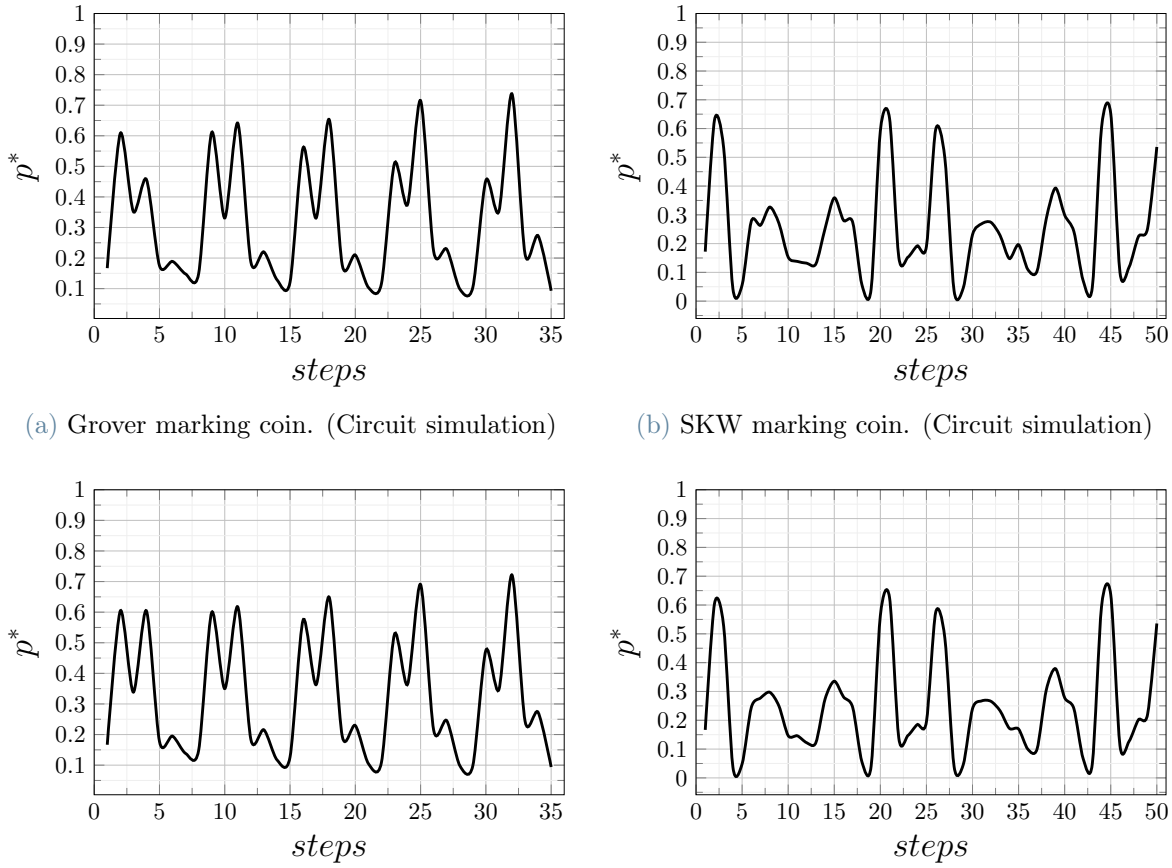
$$\begin{aligned} t^* &= \frac{\pi}{\sqrt{2(l+1)}}\sqrt{V} \\ p^* &= \frac{4l}{(l+1)^2} \end{aligned} \tag{3.46}$$

where t^* represents the number of times we need to apply the evolution operator U (i.e. the number of steps of the walk) to reach the first probability peak of the marked vertex whereas p^* is the value of that probability peak.

Our intuition suggests that our coin application together with the shift phase behave in the same way defined above. We cannot give a formal proof that our solution is mathematically equivalent to a Lackadaisical Quantum random walk because our coin phase is not a direct implementation of the formula expressed in Equation (3.45) since we follow the double coin procedure shown in Figure 3.15. This link between our implementation and Lackadaisical walks would help in better understanding the performance of our solution. In the following Section we express the empirical evidence we have gathered to support this connection.

3.4.7. Simulation & Performance

We simulated the final circuit represented in Figure 3.28 with different steps and two marking coins. We compared the results obtained via the circuit simulation using *qiskit* [12], an open-source SDK for simulating quantum circuits, on IBM's online Quantum Computing lab using Aer simulator (automatic setting) with a mathematical simulation of a Lackadaisical Quantum walk (addendum of [39]) that we adapted for the Johnson graph $J(4, 2)$. We show a comparison of the obtained simulations.



(a) Grover marking coin. (Circuit simulation)

(b) SKW marking coin. (Circuit simulation)

(c) Grover marking coin. (Modified script of [39])

(d) SKW marking coin. (Modified script of [39])

Figure 3.29: Simulated probabilities of measuring the marked vertex with different steps and marking coins on $J(4, 2)$ with $l = 4$

These results support the fact that our implementation complies with the mathematical structure of the Lackadaisical walks. Further studies are needed in order to confirm this design as an instance of this kind of walks.

According to [30] the optimal value for l for a Johnson graph, vertex-transitive graphs in general, is:

$$l_{opt} = \frac{d}{V} = \frac{k(n-k)}{\binom{n}{k}} \tag{3.47}$$

This self-loop weight maximizes p^*. Consequently, by using this weight value it is possible to measure with certainty the marked element(s) of the graph after t^* search steps.

Our solution does not permit us to freely modify the weight l, whose value is constrained by the structure of the circuit. The reason behind this is that parameters n and k determine the ratio between the number of correct directions and the number of self-loops per vertex.

Because of the fact that we need at least $\lceil \log_2(\binom{n}{2}) \rceil$ qubits to apply all $\binom{n}{2}$ swaps, the actual weight of the self-loops is not simply what we have shown in Figure 3.26, but it depends on the number of qubits in the coin register. This is explained by the fact that the Grover diffuser performs the inversion about average of all $2^{qubits\text{-}number}$ combinations, even though we do not control all of them in the successive shift phase. This means that the weight l of the self-loops in our implementation is equal to:

$$l = 2^{\lceil \log_2 \binom{n}{2} \rceil} - k(n-k) \tag{3.48}$$

This formula deviates from the optimum more and more the bigger n becomes. Specifically, the bigger the n the higher the value of l and the more likely it is to remain in the same node while exploring. Plotting l with different values of n and k shows the following behaviour:

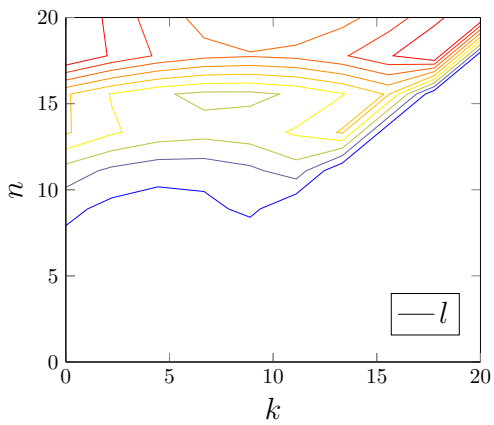


Figure 3.30: Self-loops weight value of our implementation for $0 \leq n, k \leq 20$. (The redder the line, the higher the value)

Using our parameter l, the following plots suggest that parameter t^* increases whereas the parameter p^* lowers as the Johnson graph grows in dimension.

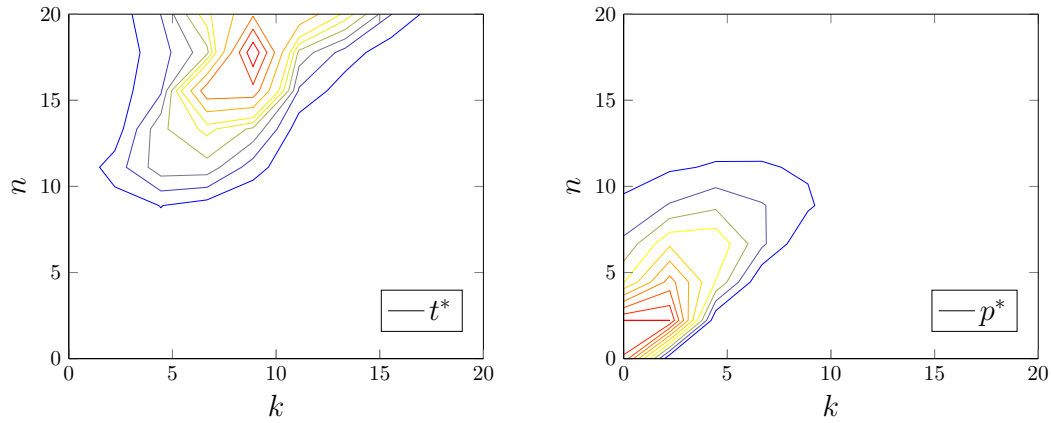


Figure 3.31: t^* and p^* values for $0 \leq n, k \leq 20$. (The redder the line, the higher the value)

This shows that for bigger and bigger values of n our solution performs poorly. The value of l becomes so high that the process struggles in exploring the graph.

Regarding performance, the qubit number starts as $n + \lceil \log_2 \binom{n}{2} \rceil$ according to Equation (3.40), but ancillas may be necessary depending on the oracle. In the example shown no ancillas are required but in the next chapter we will show that this won't hold true. Concerning depth and gate number we can analyze the circuit in phases:

- **Initialization phase**: the Dicke state circuit is applied to the state register, the gate number cost is the sum of Equations (3.39) and (3.38). On the other hand, an Hadamard gate is applied to each qubit of the coin register. Clearly the depth cost is dominated by the Dicke state circuit.
- **Coin phase**: it can be seen in Figure 3.8 that the depth of the Diffuser operator is constant. For both coin applications the depth is the same and is equal to 5: 2 sets of **H** and **X** gates on each qubit plus a controlled-**Z** gate in between.
- **Shift phase**: as explained in Equation (3.41), the cost is $\binom{n}{2}$ controlled-**SWAP** gates.

The costs of the Coin and the Shift phase are repeated for each walk step.

	Initialization	**Coin phase**	**Shift phase**
Gates	$\mathcal{O}(n + k^2)$	$\mathcal{O}(\log \binom{n}{2})$	$\mathcal{O}(\binom{n}{2} \log \binom{n}{2})$
Depth	$\mathcal{O}(n)$	10	$\binom{n}{2}$

Table 3.2: Single Johnson performance

	Initialization	Coin phase	Shift phase
CNOT	$5nk - 5k^2 - 2n$	0	0
RY	$4nk - 4k^2 - 2n + 1$	0	0
H	$\lceil \log_2 \binom{n}{2} \rceil$	$4\lceil \log_2 \binom{n}{2} \rceil$	0
X	0	$4\lceil \log_2 \binom{n}{2} \rceil$	0
CZ	0	2	0
CSWAP	0	0	$\lceil \log_2 \binom{n}{2} \rceil \binom{n}{2}$

Table 3.3: Single Johnson gate numbers

To conclude the discussion on the exploration of the Johnson graph, this is the only circuit that, to the best of our knowledge, accomplishes a Johnson graph exploration.

In the next chapter we will see how to encode a search problem on a product of Johnson graphs to tackle ISD.

4 | Quantum Information Set Decoding Algorithm

In this chapter we are going to give a description of our novel Information Set Decoding algorithm that exploits Quantum Random Walks on a product of Johnson graphs. The first to realise the connection between ISD and Quantum Computing was Bernstein [7], who describes a quantum version of Prange's algorithm (Section 1.2.3) using Grover's search. Kachigar and Tillich [16] are the first to tackle the ISD problem via quantum random walk.

The chapter is organised as follows: in the first section we explain how the Finiasz-Sendrier ISD can be translated in a graph representation useful for exploration, in the second section we present the formulation of the two-sum problem, in the third we explain our Quantum Collision algorithm based on quantum walk on the product of Johnson graphs and in the last sections we discuss the experimental results and the performance achieved.

4.1. Graph formulation of ISD

Before diving deep into the discussion, we give a brief recap of the fundamental aspects of the Finiasz-Sendrier ISD formulation. As we have seen in Section 1.2.7, the goal is to find the correct weight distribution in the two halves of the permuted error $\hat{\mathbf{e}}_{\mathcal{I}}$. In order to solve this FS-ISD with a Quantum walk search algorithm, we recast the problem into a graph formulation. From now on we will focus on the two superior subparts of the matrix V: V_{up1} and V_{up2}.

Suppose we want to find a useful way to encode the selection of the columns of the $V_{up,1}$ submatrix. Selecting a group of columns of V_{up1} means guessing a $\frac{p}{2}$ weight into the first $\frac{k+l}{2}$ half of $\hat{\mathbf{e}}_{\mathcal{I}}$ or, in other terms, choosing the columns of V_{up1} in correspondence to the position of the ones in $\hat{\mathbf{e}}_{\mathcal{I},up1}$.
The columns selection can be encoded employing a Johnson graph as presented in Chap-

ter 3. For instance if we choose V_{up1} with $l = 4$ and $k = 4$, a possible matrix is:

$$V_{up1} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{4.1}$$

By using $\frac{k+l}{2}$ bit long binary labels we can express the selection of the columns with the position in the binary string. For example, suppose that the error $\frac{p}{2}$ equals to 2, the label 1100 will correspond to selecting the columns in position 0 and 1. For our purpose we have to find all the possible way to combine a number of columns equal to $\frac{p}{2}$, i.e. all the ways to distribute the weight in $\hat{\mathbf{e}}_{\mathcal{I},up1}$. As stated in Section 1.2.7 we have $\binom{(k+l)/2}{p/2}$ possible combinations for this column selection. A Johnson graph $J(x, y)$ with $x = \frac{k+l}{2}$ and $y = \frac{p}{2}$ expresses these combinations by using the binary labeling of the nodes explained in Section 3.3.

$$V_{up1} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

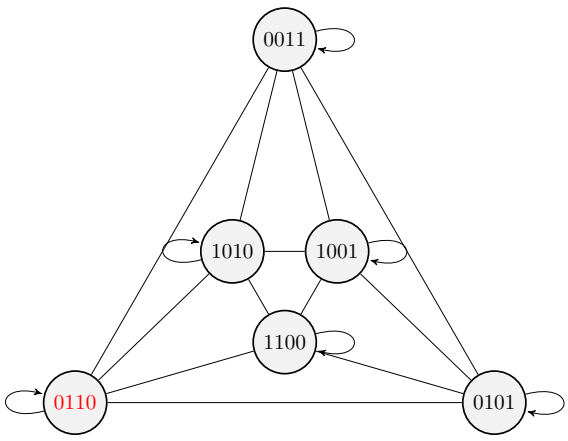


Figure 4.1: V_{up1} encoding of the column selection via a Johnson graph. Vertex 0110 corresponds to the selection of the columns highlighted in red

Now the issue is to find a way of encoding the column selection of the matrix in a quantum circuit.

4.1.1. Quantum circuit for matrix encoding

To translate our l row matrix into the circuit we need l qubit to encode the sum of the selected entries. We call this register *sum*. Every qubit will store the sum of bits alongside a row, this can be obtained by applying an **X** gate for each 1 in a row as shown

in Figure 4.2.

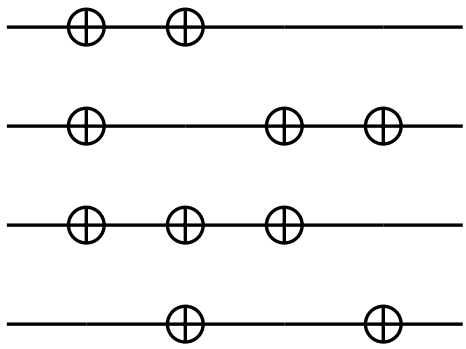


Figure 4.2: The input matrix V_{up1} transposed in a quantum circuit

The next step is to use a $\frac{k+l}{2}$ qubit register $|sel\rangle$ that acts as column selector. The selectors are implemented as controlled **NOT** on the qubit that stores the matrix value as described in Figure 4.3.

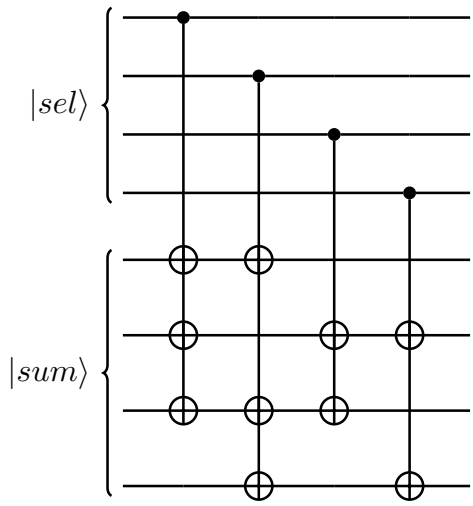


Figure 4.3: The column selectors circuit for $V_{up,1}$

In practice we choose a column out of $\frac{k+l}{2}$ available by activating the corresponding controlled **NOT**. The main idea is: based on the state of the register $|sel\rangle$ we **XOR** together a subset of the columns of the matrix. At the end of the circuit in Figure 4.3, each qubit of the register $|sum\rangle$ will contain the sum module 2 of every ones of its correspondence row. Going back to what we discussed in the previous Subsection, this has the same effect as performing the usual row-column product between the matrix V_{up1} and the column vector $\hat{\mathbf{e}}_{\mathcal{I},up1}$.

In term of performance we have a number of qubit and a circuit depth that scale with the parameters of the matrix

$$\text{qubit number} = \frac{k+l}{2} + l \tag{4.2}$$

$$\text{depth} = \frac{k+l}{2} \tag{4.3}$$

From this point on we will refer to the selectors as the vertices of the Johnson graph according to the scheme explained at the start of this section. Once we have translated the input of the ISD problem into a quantum circuit, in the next Sections we will see how to search for a solution with a Quantum walk algorithm.

4.2. FS-ISD Collision as 2-sum problem

In this Section we show how to solve FS-ISD using a Quantum walk search. Before doing so we need to remind some mathematical concepts. As we have seen in Section 1.2.7, solving the FS-ISD collision procedure means finding a solution to the following Equation

$$V_{up}\hat{\mathbf{e}}_{\mathcal{I}} = \hat{\mathbf{s}}_{up} \Leftrightarrow \hat{\mathbf{s}}_{up} = V_{up1}\hat{\mathbf{e}}_{\mathcal{I},up1} + V_{up2}\hat{\mathbf{e}}_{\mathcal{I},up2} \tag{4.4}$$

Solving this equation can be seen as an instance of finding a solution to the 2-SUM problem. Kachigar and Tillich [16] have already formalized this idea. If we take into consideration the Finiasz-Sendrier formulation, this is exactly what we are trying to achieve when comparing the two halves of $\hat{\mathbf{e}}_{\mathcal{I}}$. The problem we address can now be formulated as follows:

$$\begin{aligned} V_{up1}\mathbf{e}_0 + V_{up2}\mathbf{e}_1 = \hat{\mathbf{s}}_{up} \quad g(\mathbf{e}_0, \mathbf{e}_1) = 0 \\ E_0 = \{(\mathbf{e}_0, 0_{(k+l)/2}) \in \mathbb{F}_2^{l+k}, \mathbf{e}_0 \in \mathbb{F}_2^{(l+k)/2}, wt(\mathbf{e}_0) = p/2\} \\ E_1 = \{(0_{(k+l)/2}, \mathbf{e}_1) \in \mathbb{F}_2^{l+k}, \mathbf{e}_1 \in \mathbb{F}_2^{(l+k)/2}, wt(\mathbf{e}_1) = p/2\} \end{aligned} \tag{4.5}$$

where g is defined as $g(\mathbf{e}_0, \mathbf{e}_1) = 0$ if and only if $\hat{\mathbf{e}}_{\mathcal{I}} = (\hat{\mathbf{e}}_{\mathcal{I},up1}|\hat{\mathbf{e}}_{\mathcal{I},up2})$ is of weight p. In other words if we have multisets of values $V_{up1}\mathbf{e}_0$ and $V_{up2}\mathbf{e}_1$ and a target sum $\hat{\mathbf{s}}_{up}$, the question is to decide whether any subset of those values sum to precisely $\hat{\mathbf{s}}_{up}$. We proceed with the discussion with a numerical example. Assuming that V_{up1}, V_{up2} and $\hat{\mathbf{s}}_{up}$ have the

following values:

$$V_{up1} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \qquad V_{up2} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \qquad \hat{\mathbf{s}}_{up} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{4.6}$$

We want to find a combination of columns of V_{up1} and V_{up2} such that their sum is equal to $\hat{\mathbf{s}}_{up}$. More formally given two subsets $\mathcal{L}_1 \subseteq \{0, \ldots, (k+l)/2-1\}$ and $\mathcal{L}_2 \subseteq \{0, \ldots, (k+l)/2-1\}$ with $|\mathcal{L}_1| = |\mathcal{L}_2| = (k+l)/2$ we have

$$\hat{\mathbf{s}}_{up} = \sum_{l_1 \in \mathcal{L}_1} \mathbf{v}_{up1_{l_1}} + \sum_{l_2 \in \mathcal{L}_2} \mathbf{v}_{up2_{l_2}} \tag{4.7}$$

where $\mathbf{v}_{up1_{l_1}}$ are the columns in position l_1 of V_{up1} and $\mathbf{v}_{up2_{l_2}}$ are those in position l_2 of V_{up2}. Using binary matrices, summing modulo 2 is a XOR between the columns.

$$\hat{\mathbf{s}}_{up} = \sum_{l_1 \in \mathcal{L}_1} \mathbf{v}_{up1_{l_1}} + \sum_{l_2 \in \mathcal{L}_2} \mathbf{v}_{up2_{l_2}} \Leftrightarrow \hat{\mathbf{s}}_{up} = \bigoplus_{l_1 \in \mathcal{L}_1} \mathbf{v}_{up1_{l_1}} \bigoplus_{l_2 \in \mathcal{L}_2} \mathbf{v}_{up2_{l_2}} \tag{4.8}$$

In the given example, the exact combination comes by the columns in position 0 and 2 of V_{up1} and by those in position 1 and 3 of V_{up2}.

$$\overset{V_{up1_0}}{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}} \oplus \overset{V_{up1_2}}{\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}} \oplus \overset{V_{up2_1}}{\begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}} \oplus \overset{V_{up2_3}}{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \hat{s}_{up} \tag{4.9}$$

Using the same graph representation of Section 4.1 this means that we are looking for node 1010 on $J_1(\frac{k+l}{2}, \frac{p}{2})$ and node 0101 on $J_2(\frac{k+l}{2}, \frac{p}{2})$, where $J_1(\frac{k+l}{2}, \frac{p}{2})$ encodes the column selection of V_{up1} and $J_2(\frac{k+l}{2}, \frac{p}{2})$ encodes the column selection of V_{up2}.

We solve our version of the 2-SUM by means of a Quantum walk search on the Cartesian product of two Johnson graphs.

Definition 4.1 (Cartesian Product between Graphs). *Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ their Cartesian product $G_1 \times G_2$ is the graph $G = (V, E)$ where:*

- $V = V_1 \times V_2$ *i.e.* $V = \{(v_1, v_2) | v_1 \in V_1, v_2 \in V_2\}$

- $E = \{(u_1u_2, v_1v_2) | (u_1 = v_1 \wedge (u_2, v_2) \in E_2) \vee ((u_1, v_1) \in E_1 \wedge u_2 = v_2)\}$

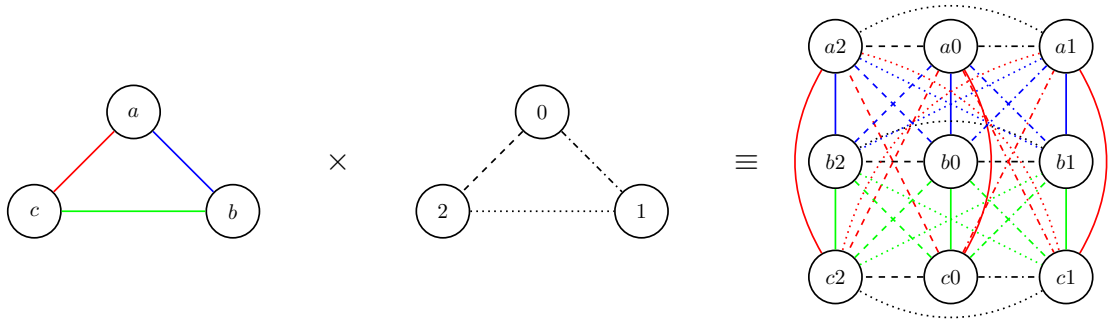


Figure 4.4: Example of a Cartesian product between graphs

For our purpose we need the product between two Johnson graphs $J^2(\frac{k+l}{2}, \frac{p}{2}) = J_1(\frac{k+l}{2}, \frac{p}{2}) \times J_2(\frac{k+l}{2}, \frac{p}{2})$ which has $\binom{(k+l)/2}{p/2}^2$ vertices and is $2(\frac{p}{2})(\frac{k+l}{2} - \frac{p}{2})$-regular ([16]).

In order to take a step on the product graph $J^2(\frac{k+l}{2}, \frac{p}{2})$ we can perform a step separately on the two graphs $J_1(\frac{k+l}{2}, \frac{p}{2})$ and $J_2(\frac{k+l}{2}, \frac{p}{2})$ and check the corresponding vertex of their product by observing the nodes of the two single graphs. The graphic representation of the complete search space is reported in Figure 4.5.

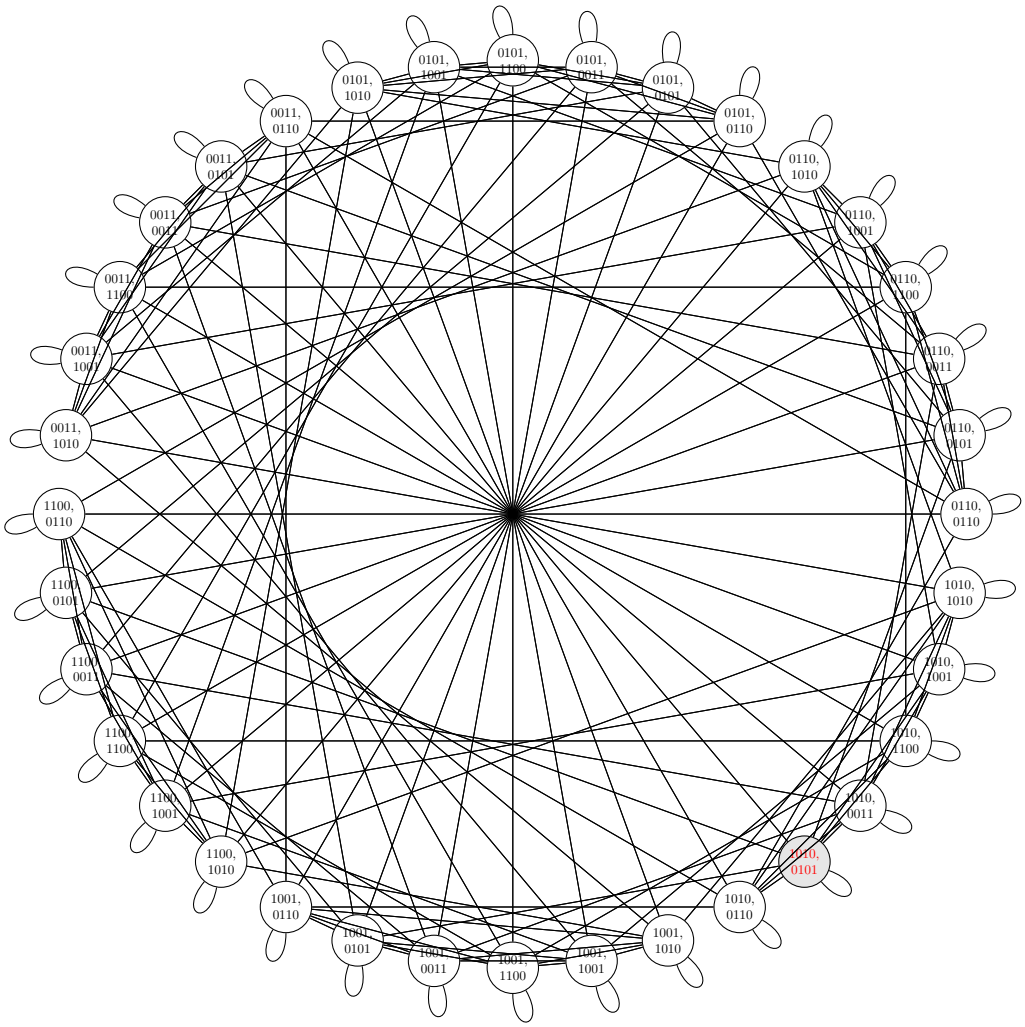


Figure 4.5: $J^2(4,2) = J_1(4,2) \times J_2(4,2)$ with marked vertex $1010, 0101$

4.3. Quantum Collision Algorithm

Now we have all the building blocks of our algorithm. Figure 4.6 shows an high level overview of our implementation which reflects the generic Quantum random walk search circuit structure that we have shown in Figure 3.12. The complete circuit is shown in Figure 4.12.

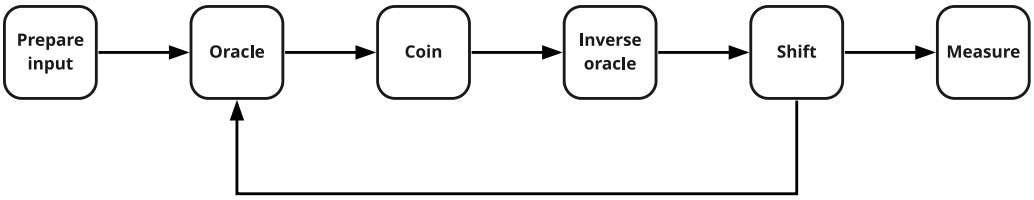


Figure 4.6: High level overview of quantum collision algorithm

Our algorithm requires four register:

- $|v_1\rangle$ is the state register for $J_1(\frac{k+l}{2}, \frac{p}{2})$, every qubit corresponds to a position in the set $\frac{k+l}{2}$ thus it has a dimension $dim(|v_1\rangle) = \frac{k+l}{2}$
- $|v_2\rangle$ is the state register for $J_2(\frac{k+l}{2}, \frac{p}{2})$, every qubit corresponds to a position in the set $\frac{k+l}{2}$ thus it has a dimension $dim(|v_1\rangle) = \frac{k+l}{2}$
- $|sum\rangle$ is the ancilla register: it encodes the matrix values and its dimension depend on the number of rows of V_{up} so $dim(|sum\rangle) = l$
- $|c\rangle$ is the coin register: it encodes all the possible direction of the coin $dim(|c\rangle) = \lceil \log_2 \binom{(k+l)/2}{2} \rceil$

Input preparation. The first stage of the circuit is the registers initialisation. To generate the superposition of the nodes of the two Johnson graphs $J_1(\frac{k+l}{2}, \frac{p}{2})$ and $J_2(\frac{k+l}{2}, \frac{p}{2})$ we apply the Dicke operator (Figure 3.20) on registers $|v_1\rangle$ and $|v_2\rangle$. On the other hand, to obtain an equal superposition of all the direction we apply the Hadamard gate (Eq. 2.18) on the coin register $|c\rangle$.

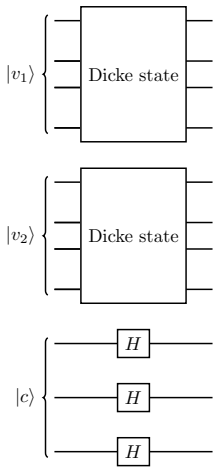


Figure 4.7: Final circuit: Input preparation

Oracle. The oracle phase is used to encode the values of the two matrix V_{up1} and V_{up2} on the register $|sum\rangle$ following the same approach described in Figure 4.3. The selectors for the first matrix are activated by the qubit in register $|v_1\rangle$ and for the second matrix by those in register $|v_2\rangle$. With this structure, at the end of the oracle phase we obtain on register $|sum\rangle$ the syndrome value corresponding to the sum of the selected columns.

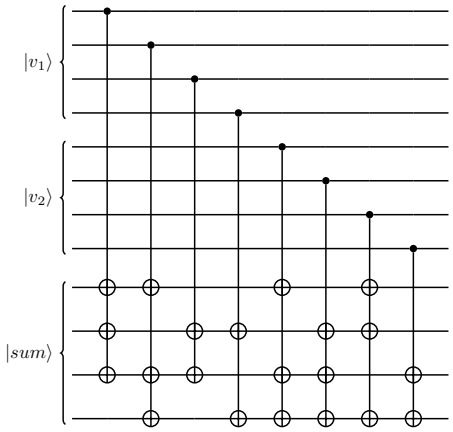


Figure 4.8: Final circuit: Oracle

Coin. The coin phase can be divided into two sub-phases:

- First sub-phase: the Grover diffuser operator (Figure 3.8) is applied on register $|c\rangle$ with $C_0 = G$ regardless of the state value.
- Second sub-phase: in order to mark the right vertex the application of the second coin $C_1 = G$ is conditioned by the desired value of the syndrome computed on $|sum\rangle$. In our example case, $\hat{\mathbf{s}}_{up} = [1100]$.

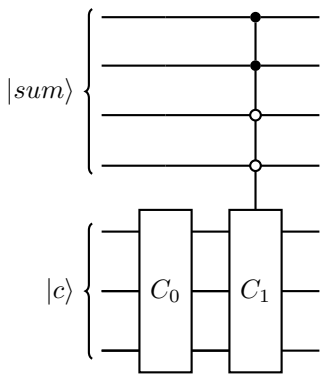


Figure 4.9: Final circuit: Coin

Inverse oracle. This phase is used to uncompute the register $|sum\rangle$ in order to reset

the register for the next walk iteration. This is performed by applying in reversed order the same sequence of operations of the oracle phase.

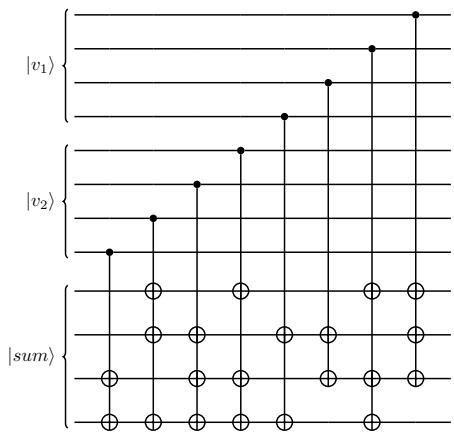


Figure 4.10: Final circuit: Inverse Oracle

Shift. The shift phase has the same structure of Figure 3.27 but is applied twice on the two state registers. This is done in order to move simultaneously in both graphs during the same shift application. Both shift applications are controlled by the register $|c\rangle$, the first one targets $|v_1\rangle$ while the second targets $|v_2\rangle$.

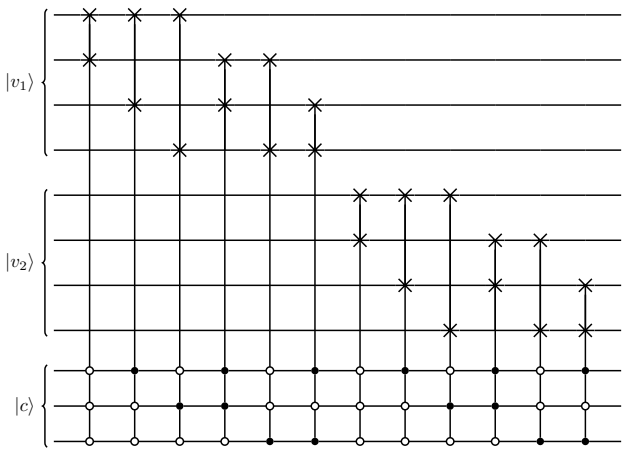


Figure 4.11: Final circuit: Shift

Measurement. It is the last stage of the algorithm. After repeating the phases in the loop body a certain number of times we need to measure the state of two registers $|v_1\rangle$ and $|v_2\rangle$. It will return a $(k+l)$-bit long string where the first half corresponds to the measurement of the state of register $|v_1\rangle$ and the second half to that of register $|v_2\rangle$. In our case we expect 1010 as measurement for $|v_1\rangle$ and 0101 for $|v_2\rangle$.

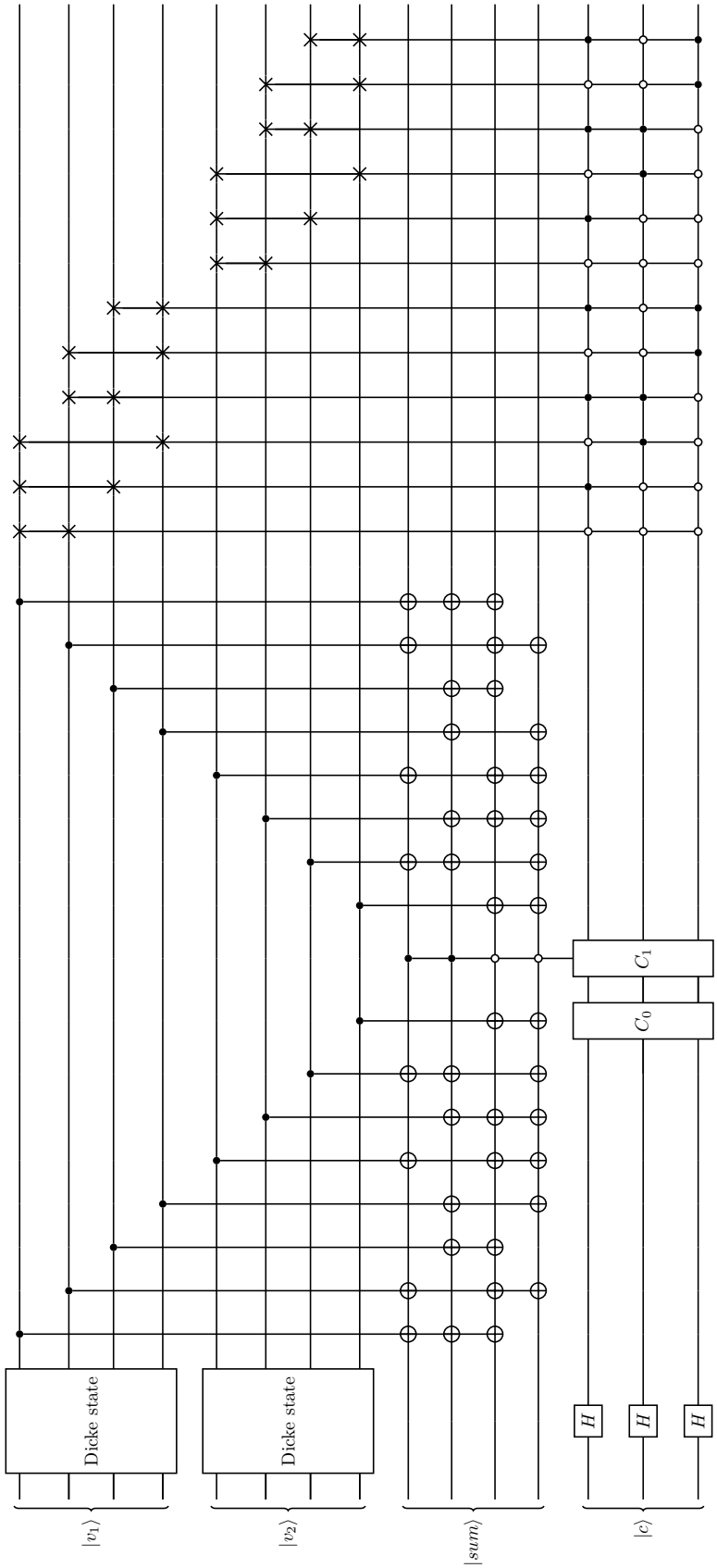


Figure 4.12: Example of one step of a Quantum random walk search on a product of $J(4,2) \times J(4,2)$ with V_{up1}, V_{up2} and $\hat{\mathbf{s}}_{up}$ values described in Equation 4.6

The quantum circuit we have just described behaves according to Algorithm 14. The new parameters γ and t^* of the Johnson graph that we have described in the previous Chapter are now

$$\begin{aligned} \gamma &= 2^{\lceil \log_2 \binom{(k+l)/2}{2} \rceil} - \frac{p}{2}\left(\frac{k+l}{2} - \frac{p}{2}\right) \\ t^* &= \frac{\pi}{\sqrt{2(\gamma+1)}}\sqrt{\binom{(k+l)/2}{p/2}} \end{aligned} \tag{4.10}$$

Algorithm 14 QuantumCollision

Input: $(k+l) + l + \lceil \log_2 \binom{(k+l)/2}{2} \rceil$ qubits
s: $n-k$ bit long syndrome
p : the weight of the first k bit of $\hat{\mathbf{e}}$ with $0 \leq p \leq t$
V_{up1}, V_{up2}: the two $(k+l)/2 \times l$ V_{up} submatrices

Output: $\mathcal{L}_1, \mathcal{L}_2$ two sets of indexes of the valid columns of V_{up1}, V_{up2}

1: Prepare the initial state $|v_1\rangle = |v_2\rangle = |DickeState\rangle$, $|coin\rangle = |H^{\lceil \log_2 \binom{(k+l)/2}{2} \rceil}\rangle$, $|sum\rangle = |0^l\rangle$
2: Apply U^{t^*} where $U = SO^{-1}CO$
3: Measure registers $|v_1\rangle$ $|v_2\rangle$ in the computational basis, yielding $\mathcal{L}_1, \mathcal{L}_2$ sets of indexes

where S, O, C, O^{-1} are the operations we descibed as Shift, Oracle, Coin and Inverse oracle.

The pseudocode of our hybrid classical-quantum FS-ISD algorithm can be derived from the original Finiasz-Sendrier algorithm 5 where we have substituted the classic Collision sub-procedure with our QuantumCollision procedure.

Algorithm 15 Syndrome decoding - Quantum Finiasz-Sendrier

Input: $\mathbf{s}$: $n-k$ bit long syndrome
H: $(n-k) \times n$ parity check matrix
t : the weight of the error vector
p : the weight of the first k bit of $\hat{\mathbf{e}}$ with $0 \leq p \leq t$
l : input parameter $0 \leq l \leq n-k-t+p$

Output: $\mathbf{e}$: n-bit error vector s.t. $H\mathbf{e} = \mathbf{s}$ with $wt(\mathbf{e}) = t$
$\hat{\mathbf{s}}$: $n-k$ bit new syndrome
P : $n \times n$ permutation matrix
V : $(n-k) \times k$ binary matrix

1: **repeat**
2: $(\hat{\mathbf{s}}, P, [V|W]) \leftarrow ISextract_{FS}(\mathbf{s}, H)$
3: **if** QuantumCollision(V_{up1}, V_{up2}) **then**
// $\mathcal{L}_1$ and $\mathcal{L}_2$ set of indexes returned by QuantumCollision
4: $\hat{\mathbf{e}}_{\mathcal{I}^*} \leftarrow \hat{\mathbf{s}} - \sum_{l_1 \in \mathcal{L}_1} \mathbf{v}_{up1_{l_1}} - \sum_{l_2 \in \mathcal{L}_2} \mathbf{v}_{up2_{l_2}}$
5: **if** $wt(\hat{\mathbf{e}}_{\mathcal{I}^*}) = t - p$ **then**
6: $\hat{\mathbf{e}} \leftarrow [0_{k+l} | \hat{\mathbf{e}}_{\mathcal{I}^*}]$
7: **for** l_1 **do**
8: $\hat{\mathbf{e}} \leftarrow \hat{\mathbf{e}} + [0_{l_1} | 1 | 0_{n-1-l_1}]$
9: **end for**
10: **for** l_2 **do**
11: $\hat{\mathbf{e}} \leftarrow \hat{\mathbf{e}} + [0_{l_2} | 1 | 0_{n-1-l_2}]$
12: **end for**
13: **return** $P\hat{\mathbf{e}}$

According to our calculations, the complexity of Algorithm 15 is equal to

$$C_{Q-FS}(n,k,t,p,l) = \frac{\binom{n}{t}}{\binom{n-k-l}{t-p}\binom{(k+l)/2}{p/2}^2} (C_{IS-FS}(n,k,l) + \binom{(k+l)/2}{p/2}\left(C_{QuantumCollision} + \frac{\binom{(k+l)/2}{p/2}}{2^l} p(n-k-l)\right)) + p \tag{4.11}$$

where $C_{QuantumCollision}$ is the complexity of the novel quantum circuit. Giving a value to such complexity is in literature, as of today, not an agreed topic. Today's quantum machines complexity heavily depends on the error correction of the underlying technology which vastly overshadows the complexity of the circuit encoding. Our implementation is transparent to these issues as this thesis does not concern these problems. It is possible

to express the time complexity as:

$$C_{QuantumCollision} = \mathcal{O}(gates * t^*) \tag{4.12}$$

This is justified by the hypothesis that the cost of a single quantum gate is comparable with the cost of a classical gate. The *gates* value is show in the next Section.

On the other hand, the spatial complexity of QuantumCollision() is:

$$S_{QuantumCollision} = \frac{k+l}{2} + l + \lceil \log_2 \binom{(k+l)/2}{2} \rceil \tag{4.13}$$

The circuit returns the marked element with a particular maximum probability that we explain in Section 4.5.

In Section 4.4 we analyze depth and gate numbers of Algorithm 15, which are today a good measurement for the complexity of quantum circuits.

4.4. Performance evaluation

The final circuit shows the following performance:

	Input Preparation	Oracle	Coin	Inverse Oracle	Shift
Gates	$\mathcal{O}(\frac{k+l}{2} + p^2)$	$\mathcal{O}(\frac{k+l}{2} l)$	$\mathcal{O}(\log \binom{(k+l)/2}{2})$	$\mathcal{O}(\frac{k+l}{2} l)$	$\mathcal{O}(\binom{(k+l)/2}{2} \log \binom{(k+l)/2}{2})$
Depth	$\mathcal{O}(\frac{k+l}{2})$	$\mathcal{O}(\frac{k+l}{2})$	10	$\mathcal{O}(\frac{k+l}{2})$	$\binom{(k+l)/2}{2}$

Table 4.1: $J^2(\frac{k+l}{2}, \frac{p}{2})$ performance

	Input Preparation	Oracle	Coin	Inverse Oracle	Shift
CNOT	$5(k+l)p/4 - 5p^2/4 - (k+l)$	$(k+l)l$	0	$(k+l)l$	0
RY	$(k+l)p - p^2 - 2(k+l) + 1$	0	0	0	0
H	$\lceil \log_2 \binom{(k+l)/2}{2} \rceil$	0	$4\lceil \log_2 \binom{(k+l)/2}{2} \rceil$	0	0
X	0	0	$4\lceil \log_2 \binom{(k+l)/2}{2} \rceil$	0	0
CZ	0	0	2	0	0
CSWAP	0	0	0	0	$2\binom{(k+l)/2}{2} \lceil \log_2 \binom{(k+l)/2}{2} \rceil$

Table 4.2: $J^2(\frac{k+l}{2}, \frac{p}{2})$ gate numbers

These number are achieved using a total number of qubits equal to $(k+l)+l+\lceil \log_2 \binom{(k+l)/2}{2} \rceil$.

Table 4.1 gives the performance of the $J^2(\frac{k+l}{2}, \frac{p}{2})$ graph regarding both the depth and number of gates. From a computational point of view, the Shift operation is the most significant part of the circuit because it resulted in having to applying all the possible swaps $\binom{(k+l)/2}{2}$ which has an exponential cost compared to the other phases which are at most polynomial or as for the coin constant. Our solution however does not need ancillas, beside the ones needed for the Oracle phase, meaning it takes much longer to perform the computation but needs very few qubits.

Table 4.2 shows the actual gate numbers at each phase of the Algorithm divided for each gate type. It is clear that the biggest contribute to the overall cost is carried by the **CSWAP** gates of the Shift phase.

4.5. Experimental results

We simulated the final circuit shown in Figure 4.12 on IBM's online Quantum Computing lab using qiskit's Aer simulator (automatic setting) and obtained the following probability curve

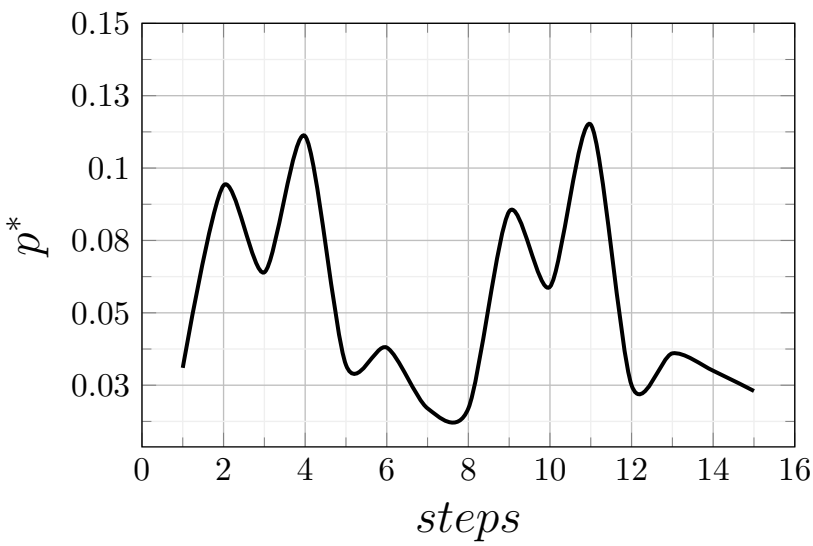


Figure 4.13: Simulated probabilities of measuring the marked vertex with different steps on a product of $J(4,2) \times J(4,2)$ with the values of V_{up1}, V_{up2} and $\mathbf{s}_{up}$ described in Equation 4.6.

We expect the value of the first probability peak of measuring the marked vertex, as

explained in Chapter 3 Section 3.4.6, to be indicated by the parameter p^*, which is now:

$$p^* = \frac{4\gamma}{(\gamma+1)^2} \tag{4.14}$$

with

$$\gamma = 2^{\lceil \log_2 \binom{(k+l)/2}{2} \rceil} - \frac{p}{2}\left(\frac{k+l}{2} - \frac{p}{2}\right) \tag{4.15}$$

We noticed that the simulation in Figure 4.13 is the same shape as the probability curve in Figure 3.29 of the single Johnson graph search but flattened. As the experimental simulation shows, the new probability curve is flattened by a factor of 6 compared to the single Johnson graph search in Figure 3.29a.

The total number of combinations of two single Johnson graphs state spaces is equal to $V = V_1 \times V_2$ as we have explained in Definition 4.1. Our hypothesis is that since the complete state space explores the product of two graphs, the probability of measuring a vertex in the product is divided by the ratio between the dimension of the product and the dimension of one the original graphs, in this case either V_1 or V_2.

In our example, the product has a total of 36 vertices whereas the single graphs have 6. As a consequence, the probability of measuring the marked vertex follows the same shape of the single graph exploration but is divided by a factor of 6.

This shows that the definition of p^* as it is is not suited for a product between graphs.

As a consequence, our circuit's probability of measuring the correct vertex not only lowers as the values of n gets bigger, but is worsened further if employed for the encoding of a product of Johnson graphs. Further experimentation should be carried out in order to confirm these observations.

5 | Conclusions and future developments

The general goal of this thesis has been to explore the usefulness of Quantum Random Walks in ISD algorithms. Current literature on the topic suggests studying Johnson graphs because of their regularity. This led us to find a circuital implementation of such walks on a Johnson graph, which to the best of our knowledge has never been accomplished.

Taking inspiration from the work of Kachigar and Tillich [16], which conceptualized the use of Quantum random walks to solve ISD problems, we set up to develop a quantum algorithm that finds collisions on a sub-procedure of the Finiasz-Sendrier ISD algorithm.

We succeded in obtaining a quantum circuit that performs this operation. We simulated the circuit using Qiskit, IBM's quantum circuit simulation library, and compared it with Thomas Wong's mathematical formalization of Lackadaisical Quantum Random Walks obtaining strong evidence that our solution complies with this type of walk.

Our solution could be employed in any search-based problem involving a structure which can be encoded as a Johnson graph or a product of them.

The most challenging part was figuring out an efficient way of performing the shift operation in a Johnson graph which turned out to be quite complicated due to the combinatorial nature of its encoding. The initial approach of having a loop-free structure has been discarded in favour of a significant reduction in computational resources. The final solution has linear spatial complexity but the maximum probability of measuring the correct element is not close to 1, as Thomas Wong's work explains, and requires a relatively high number of iterations because of the exponential depth of the Shift phase.

Regarding future development, different paths are possible. Firstly, research should focus on understanding if different ISD algorithms can benefit from this quantum solution. Secondly, finding a more efficient solution for the shift phase in terms of both depth and gate number could drastically improve the circuit performance. Our intuition suggests that a

first step in this direction could involve finding an indexing algorithm for combinatorial encodings as shown in [21]. It would be interesting to understand whether the same problem can be solved using other graph topologies on which an efficient shift can be implemented. This work focuses on the coined quantum walk model, nevertheless we do not exclude the possibility of obtaining benefits from using other search paradigms such as the Szegedy Quantum walk model or the Continuous time Quantum Walk.

Bibliography

[1] Y. Aharonov, L. Davidovich, and N. Zagury. Quantum random walks. *Phys. Rev. A*, 48:1687–1690, Aug 1993. doi: 10.1103/PhysRevA.48.1687. URL `https://link.aps.org/doi/10.1103/PhysRevA.48.1687`.

[2] D. Augot. Initial recommendations of long-term secure post-quantum systems. 09 2015. URL `https://pqcrypto.eu.org/docs/initial-recommendations.pdf`.

[3] A. Bärtschi and S. J. Eidenbenz. Deterministic preparation of dicke states. In L. A. Gasieniec, J. Jansson, and C. Levcopoulos, editors, *Fundamentals of Computation Theory - 22nd International Symposium, FCT 2019, Copenhagen, Denmark, August 12-14, 2019, Proceedings*, volume 11651 of *Lecture Notes in Computer Science*, pages 126–139. Springer, 2019. doi: 10.1007/978-3-030-25027-0_9. URL `https://doi.org/10.1007/978-3-030-25027-0_9`.

[4] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Physical Review Letters*, 70:1895–1899, 1993.

[5] E. R. Berlekamp. Goppa codes. *IEEE Trans. Inf. Theory*, 19(5):590–592, 1973. doi: 10.1109/TIT.1973.1055088. URL `https://doi.org/10.1109/TIT.1973.1055088`.

[6] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Trans. Inf. Theory*, 24(3): 384–386, 1978. doi: 10.1109/TIT.1978.1055873. URL `https://doi.org/10.1109/TIT.1978.1055873`.

[7] D. J. Bernstein. Grover vs. mceliece. In N. Sendrier, editor, *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings*, volume 6061 of *Lecture Notes in Computer Science*, pages 73–80. Springer, 2010. doi: 10.1007/978-3-642-12929-2_6. URL `https://doi.org/10.1007/978-3-642-12929-2_6`.

[8] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. Technical report, GBR, 1992.

[9] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976. doi: 10.1109/TIT.1976.1055638. URL https://doi.org/10.1109/TIT.1976.1055638.

[10] H. Dinh, C. Moore, and A. Russell. Mceliece and niederreiter cryptosystems that resist quantum fourier sampling attacks. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 761–779. Springer, 2011. doi: 10.1007/978-3-642-22792-9_43. URL https://doi.org/10.1007/978-3-642-22792-9_43.

[11] P. Dirac. *The Principles of Quantum Mechanics*. Oxford University Press, 1930.

[12] M. S. A. et al. Qiskit: An open-source framework for quantum computing, 2021.

[13] M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In M. Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009. doi: 10.1007/978-3-642-10366-7_6. URL https://doi.org/10.1007/978-3-642-10366-7_6.

[14] L. K. Grover. A fast quantum mechanical algorithm for database search. In G. L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 212–219. ACM, 1996. doi: 10.1145/237814.237866. URL https://doi.org/10.1145/237814.237866.

[15] V. Guruswami, A. Rudra, and M. Sudan. *Essential Coding Theory*. Department of Computer Science and Engineering, University at Buffalo, SUNY. Work supported by NSF CAREER grant CCF-0844796, 2019.

[16] G. Kachigar and J. Tillich. Quantum information set decoding algorithms. In T. Lange and T. Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 69–89. Springer, 2017. doi: 10.1007/978-3-319-59879-6_5. URL https://doi.org/10.1007/978-3-319-59879-6_5.

[17] J. Kempe. Quantum random walks: An introductory overview. *Contemporary Physics*, 44(4):307–327, 2003. doi: 10.1080/00107151031000110776. URL https://doi.org/10.1080/00107151031000110776.

[18] P. J. Lee and E. F. Brickell. An observation on the security of mceliece's public-key cryptosystem. In C. G. Günther, editor, *Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 1988. doi: 10.1007/3-540-45961-8_25. URL `https://doi.org/10.1007/3-540-45961-8_25`.

[19] J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Trans. Inf. Theory*, 34(5):1354–1359, 1988. doi: 10.1109/18.21270. URL `https://doi.org/10.1109/18.21270`.

[20] F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM J. Comput.*, 40(1):142–164, 2011. doi: 10.1137/090745854. URL `https://doi.org/10.1137/090745854`.

[21] S. Marsh and J. B. Wang. Combinatorial optimization via highly efficient quantum walks. *Physical Review Research*, 2(2), jun 2020. doi: 10.1103/physrevresearch.2.023302. URL `https://doi.org/10.1103%2Fphysrevresearch.2.023302`.

[22] R. J. McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244: 114–116, 1978.

[23] C. S. Mukherjee, S. Maitra, V. Gaurav, and D. Roy. Preparing dicke states on a quantum computer. *IEEE Transactions on Quantum Engineering*, 1:1–17, 2020. doi: 10.1109/TQE.2020.3041479.

[24] H. Niederreiter. A public-key cryptosystem based on shift register sequences. In F. Pichler, editor, *Advances in Cryptology - EUROCRYPT '85, Workshop on the Theory and Application of of Cryptographic Techniques, Linz, Austria, April 1985, Proceedings*, volume 219 of *Lecture Notes in Computer Science*, pages 35–39. Springer, 1985. doi: 10.1007/3-540-39805-8_4. URL `https://doi.org/10.1007/3-540-39805-8_4`.

[25] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016. ISBN 978-1-10-700217-3. URL `https://www.cambridge.org/de/academic/subjects/physics/quantum-physics-quantum-information-and-quantum-computation/quantum-computation-and-quantum-information-10th-anniversary-edition?format=HB`.

[26] K. Pearson. The problem of the random walk. *Nature*, 72(1865):294–294, Jul

1905. ISSN 1476-4687. doi: 10.1038/072294b0. URL https://doi.org/10.1038/072294b0.

[27] S. Perriello. Design and development of a quantum circuit to solve the information set decoding problem. Master's thesis, Scuola di Ingegneria Industriale e dell'Informazione, Dipartimento di Elettronica, Informazione e Bioingegneria Politecnico di Milano, 2017-2018.

[28] R. Portugal. *Quantum walks and search algorithms*. New York:Springer, 2013.

[29] E. Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory*, 8(5):5–9, 1962. doi: 10.1109/TIT.1962.1057777. URL https://doi.org/10.1109/TIT.1962.1057777.

[30] M. L. Rhodes and T. G. Wong. Search on vertex-transitive graphs by lackadaisical quantum walk. *Quantum Inf. Process.*, 19(9):334, 2020. doi: 10.1007/s11128-020-02841-z. URL https://doi.org/10.1007/s11128-020-02841-z.

[31] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. doi: 10.1145/359340.359342. URL http://doi.acm.org/10.1145/359340.359342.

[32] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3): 379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. URL https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

[33] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(4): 623–656, 1948. doi: 10.1002/j.1538-7305.1948.tb00917.x. URL https://doi.org/10.1002/j.1538-7305.1948.tb00917.x.

[34] N. Shenvi, J. Kempe, and K. Whaley. A quantum random walk search algorithm. *Physical Review A*, 67, 10 2002. doi: 10.1103/PhysRevA.67.052307.

[35] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.*, 41(2):303–332, 1999. doi: 10.1137/S0036144598347011. URL https://doi.org/10.1137/S0036144598347011.

[36] J. Stern. A method for finding codewords of small weight. In G. D. Cohen and J. Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988. doi: 10.1007/BFb0019850. URL https://doi.org/10.1007/BFb0019850.

[37] H. Tanaka, M. Sabri, and R. Portugal. Spatial search on johnson graphs by discrete-time quantum walk. *CoRR*, abs/2112.03744, 2021. URL `https://arxiv.org/abs/2112.03744`.

[38] J. Von Neumann. *Mathematical Foundations of Quantum Mechanics*. Springer Verlag, 1932.

[39] T. G. Wong. Faster search by lackadaisical quantum walk. *Quantum Inf. Process.*, 17(3):68, 2018. doi: 10.1007/s11128-018-1840-y. URL `https://doi.org/10.1007/s11128-018-1840-y`.

[40] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299: 802–803, 1982.

[41] X.-l. Xue, Y. Ruan, and Z.-h. Liu. Discrete-time quantum walk search on johnson graphs. *Quantum Information Processing*, 18(2):50, Jan 2019. ISSN 1573-1332. doi: 10.1007/s11128-018-2158-5. URL `https://doi.org/10.1007/s11128-018-2158-5`.

A | Linear Algebra

Linear algebra is to quantum computing as boolean algebra is to classical computing. We have gathered here what we consider the mathematical foundations that are needed in order to properly understand the quantum phenomena we have shown.

The following appendix's sources are [25, 27]

A.1. Dirac Notation

Paul Dirac in [11] introduced the Dirac notation to better study Quantum Mechanics. Since our work is built upon the same rules, it makes sense to understand how it works.

The $|\cdot\rangle$ is used to indicate that the object is a vector. Inside, any label for the vector can be used. Traditionally, to indicate a generic vector the ψ symbol is used:

$$|\psi\rangle$$

The position of the two symbols matter. The above vector is called *ket* and represents a column vector. If switched, the symbols indicate a row vector called *bra*.

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad \langle\psi| = \begin{bmatrix} \alpha & \beta \end{bmatrix}$$

Formally, the *bra* vector is the conjugate transpose of a *ket*, meaning $\langle\psi| = (|\psi\rangle^*)^T := |\psi\rangle^\dagger$. For the moment we can consider this just another way of defining a vector. We will later understand why this notation can be very useful when dealing with complicated formulae, especially compared to the standard definition of a vector.

A.2. Vector Spaces

Wherever possible, we will express the same concepts with the Dirac notation.

The two-dimensional vector space $\mathbb{R}^2$ is the set of column vectors having the following

shape:

$$v = \begin{pmatrix} a \\ b \end{pmatrix} \quad \equiv \quad |v\rangle = \begin{bmatrix} a \\ b \end{bmatrix}$$

with $a, b \in \mathbb{R}$. It goes without saying that the same principles that we are going to explain hold true for any vector space dimension.
This is a list of a few key definitions related to vector spaces.

Definition A.1. (Transpose) *The transpose vector of a column vector v is the row vector $v^T = (a, b)$*

Definition A.2. (Vector Norm) *The norm of a vector v is defined as $\|v\| = \sqrt{a^2 + b^2}$ or $\|v\| = \sqrt{\langle v \rangle}$*

Definition A.3. (Normalized (Unitary) vector) *A normalized, or unitary, vector is a vector whose norm is equal to 1, meaning $\|v\| = 1$*

Definition A.4. (Scalar (or Inner) Product) *The scalar product between two vectors $v_1 = \left(\begin{smallmatrix} a_1 \\ b_1 \end{smallmatrix}\right)$, $v_2 = \left(\begin{smallmatrix} a_2 \\ b_2 \end{smallmatrix}\right)$ is defined as:*

$$v_1 \cdot v_2 := v_1^T v_2 = (a_1, b_1)\left(\begin{smallmatrix} a_2 \\ b_2 \end{smallmatrix}\right) = a_1 a_2 + b_1 b_2 = \|v_1\|\|v_2\| \; cos\theta$$

where θ is the angle formed between v_1 and v_2. It can also be represented as $\langle v_1 | v_2 \rangle$.

Definition A.5. (Tensor Product) *Given two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$ defined as*

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & \dots & a_{1q} \\ \vdots & & \vdots \\ b_{p1} & \dots & a_{pq} \end{bmatrix}$$

the tensor product $A \otimes B$ is the matrix $D \in \mathbb{R}^{mp \times nq}$ defined as

$$D = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}$$

In Dirac notation, given two vectors $|v_1\rangle$ $|v_2\rangle$, their tensor product is $|v_1\rangle \otimes |v_2\rangle$ also written as $|v_1 v_2\rangle$.

Definition A.6. (Orthogonal vectors) *Two vector are orthogonal if their scalar product equals to zero, meaning: $v_1 \cdot v_2 = 0$ or $\langle v_1 | v_2 \rangle = 0$.*

Definition A.7. (Linearly Independent Vectors) *A set of vectors* $\{v_i \in \mathbb{R}^2 \mid i = 1,2,\ldots,k\}$ *is a set of linearly independent vectors if*

$$a_1 v_1 + a_2 v_2 + \cdots + a_k v_k = 0, \;\; a_i \in \mathbb{R}$$

otherwise they are said to be linearly dependent.

Definition A.8. (Basis of a vector space) *A basis for a vector space is a set of linearly independent vectors belonging to it. Each other vector in the vector space can be defined as a linear combination of a basis vector.*

Definition A.9. (Orthonormal basis) *A basis for a vector space is orthonormal if all vectors are normalized and orthogonal to each other. The vectors*

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

form an orthonormal basis for $\mathbb{R}^2$ *called Standard basis.*

Definition A.10. (Hilbert Space) *A Hilbert space is a vector space with an inner product and is complete with regards to the metric induced by the norm* $\|\cdot\|$

The possible states of a quantum mechanical system are represented as unit vectors in the complex Hilbert space. A Hilbert space can be considered a special case of a vector space, but since in quantum computing vector spaces always have finite dimensions saying 'Hilbert space' or 'vector space with inner product' is equivalent.

A.3. Qubit representations

It is possible to give a geometric interpretation to a qubit. The Bloch sphere associates a qubit state to a point on the surface of a unitary radius sphere.

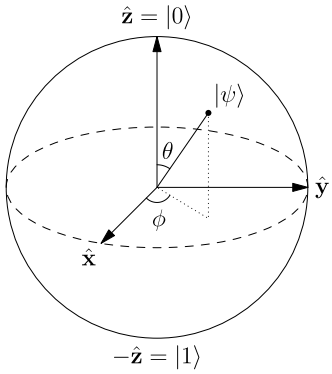


Figure A.1: Bloch Sphere

The south pole corresponds to $|1\rangle$ whereas the north pole $|0\rangle$, any other point on the surface is a superposition of these two base states.
There is a bijective relationship between a generic qubit state and a point on the unitary sphere in $\mathbb{R}^3$ represented by

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = cos(\theta/2)\,|0\rangle + e^{\phi} sen(\theta/2)\,|1\rangle$$

where θ and ϕ are real numbers.

By this definition it seems like it would be possible to encode an infinite amount of information on a qubit, since any slight angular transformation represents a different point on the surface. The reality is that in order to extract information from such a system a measurement is necessary, meaning this sphere will either output the north or the south pole. From this perspective, the probability of measuring one of the two base states is given by the 'latitude' of the system's state.

List of Figures

List of Tables

Acknowledgements

We want to thank prof. Barenghi and prof. Pelosi for introducing us to this innovative topic as well as continuous guidance and support and Simone Perriello for the laughs as well as key tips and tricks he gave us during the last 12 month. We also thank ATOS for providing us access to their quantum simulators.

Finally, we thank our families, friends and cats for supporting us during this adventure.