Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

School of Engineering and Sciences

Quantum walk-based algorithm on Scale Free Networks

A thesis presented by

German Alamilla Peralta

Submitted to the
School of Engineering and Sciences
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

Atizapán, Estado de México, November, 2019

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

School of Engineering and Sciences

The committee members, hereby, certify that have read the thesis presented by German Alamilla Peralta and that it is fully adequate in scope and quality as a partial requirement for the degree of Master of Science in Computer Science.

Dr. Salvador Elías Venegas Andraca
Tecnológico de Monterrey
Principal Advisor

Dr. Marco Lanzagorta
Naval Research Laboratory, US Navy
Committee Member

Dr. Miguel González Mendoza
Tecnológico de Monterrey
Committee Member

Dr. Raúl Monroy Borja
Associate Dean of Graduate Studies
School of Engineering and Sciences

Declaration of Authorship

I, **German Alamilla Peralta**, declare that this thesis titled, "Quantum walk-based algorithm on Scale Free Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

German Alamilla Peralta
Atizapán, Estado de México, November, 2019

©2019 by German Alamilla Peralta
All Rights Reserved

Dedication

This work is dedicated to

My family:

Wilehado Alamilla, Teresa Mendoza†, Manuel Peralta†, Dora Alcocer†, Wiliado Alamilla Mendoza, Gloria Peralta Alcocer, Brothers and Sister.

Acknowledgements

I am forever grateful to Monterrey Institute of Technology and Higher Education (ITESM), for believing in me, for opening the doors to a new area of knowledge, and providing a complete tuition. I am also grateful with the people of México that gave me a scholarship through our National Council for Science and Technology (CONACYT).
Life has been very kind to me, I am more than grateful. Along the path I have met great people. Thanks for inspiring me and believing in me. Specially, my advisor Dr. Salvador Elias Venegas Andraca, your knowledge, patience, guidance and support has been key to achieve this goal. Your kindness and friendliness is admirable. I leave benefited form your scientific knowledge.
To my supervisors, Dr. Marco Lanzagorta and Dr. Miguel González Mendoza: Thank you very much for trusting me. My admiration and respect as scientists.
To those who believed in me. My family.

Quantum walk-based algorithm on Scale Free Networks

by

German Alamilla Peralta

Abstract

The classification of information contained in complex networks such as the WWW is a central problem in network science for its technological impact. A highly successful methodology for this purpose is Google's PageRank algorithm. It allows to measure the importance or relevance of a web page depending on the relationship between pages and its connections, i.e, the hyperlink structure.

Finding ways to apply the advantages of quantum computers to real world computational tasks is an active research area. In this thesis, we implement a quantum gate-based algorithm that replicates the behaviour of the Quantum PageRank algorithm presented in [43], [44], [35]. We present a comprehensive analysis with the Szegedy quantum walk formalism. The implementation procedure is based in the quantum circuit model of quantum computing. Here, we review a Quantum PageRank algorithm and illustrate the main findings. We show the behaviour of the quantum PageRank algorithm using the IBM's Q experience framework for quantum computing. We investigate the behavior of the Quantum PageRank algorithm using quantum walks by giving explicit connections with corresponding classical algorithm. We make a comparative analysis of our findings and discuss the capabilities of a quantum computing platform applied to the Quantum PageRank algorithm. Within the restrictions imposed by hardware, we implement a multiple-controlled gate using ancilla qubits and all other elements required for implementing our quantum algorithm.

List of Figures

2.1 Visual representation of graph G. 7
2.2 A multigraph 7
2.3 A directed graph or digraph. 8
2.4 Symmetric digraph 8
2.5 Generation of the double cover bipartite graph from an irregular graph of $N = 4$ vertices. 11
2.6 Examples of complete graphs 11
2.7 Graph for C_7 12
2.8 Power law distribution 12

4.1 An unrestricted walk on a line. 26

5.1 Quantum circuit for the $|011\rangle$ state. 41
5.2 Histogram 42
5.3 Circuit representation for the $C^n(U)$ operation, where U is a unitary operator on k qubits for $n = 4$ and $k = 3$ 49
5.4 Network implementing the $C^n(U)$ operation for $n = 5$ 49
5.5 Quantum circuit implementing a quantum walk along a 16-length cycle. . 50
5.6 Increment on n qubits 51
5.7 Decrement 51
5.8 Quantum circuit implementation of the swap operator $\mathcal{S}$. 52
5.9 General Quantum circuit implementation of $\hat{U}_{walk}$ 52
5.10 Yorktown quantum processor. 54

6.1 Representation of a web with four pages. 56
6.2 A directed graph 62
6.3 Complete circuit for U_{walk} 63
6.4 K_{b_1} 63
6.5 K_{b_2} 64
6.6 K_{b_3} 64

6.7 Controlled gate corresponding to $T_{1,y}$ 64
6.8 Qiskit implementation of controlled gate corresponding to $T_{1,y}$ 65
6.9 Circuit for K_{b_2} 65

7.1 A 4-node graph. The starting vertex is labeled as 00. 67
7.2 Quantum walk circuit implementation on a graph with four nodes 67
7.3 One-step quantum walk on 4 nodes. 69
7.4 Two-step quantum walk on 4 nodes. 69
7.5 Three-step quantum walk on 4 nodes. 70
7.6 Circuit 70
7.7 Initial superposition 71
7.8 Measurement results of an equal superposition of 3 qubits. 73
7.9 One-place cyclic permutation to the right. 75
7.10 Two-place cyclic permutation to the right. 75
7.11 One-place cyclic permutation to the left results. 76
7.12 Two-place cyclic permutation to the left. 76
7.13 3-controlled gate. 77
7.14 3-controlled gate implemented in qiskit. 78
7.15 Controlled cyclic permutation to the left. 80
7.16 $R_yR_y^\dagger = I$ 82
7.17 Circuit for Pauli-z controlled operation, D. 83
7.18 Histogram results of the Szegedy quantum walk. 86
7.19 Szegedy walk results. 86
7.20 Average Quantum PageRank with Szegedy walk. 87
7.21 A general Szegedy quantum walk circuit. 87
7.22 Szegedy circuit using ancilla qubits. 88

8.1 Linking every dangling node to every node in the web ensures ensures stochasticity. 89
8.2 IBM Q 5 Tenerife. 92
8.3 Reflection about state $|0\rangle$. 93
8.4 Reflection about state $|1\rangle$. 93

A.1 Circuit corresponding to the $T_{1,y}$ transformation. 104
A.2 Circuit corresponding to $T_{2,y}$. 104
A.3 $K_{b_1}^\dagger$ circuit. 105
A.4 Circuit implementing $K_{b_2}^\dagger$. 106
A.5 Circuit implementing K_{b_3}. 107
A.6 Circuit implementation for Pauli-z operator. 108

A.7 K_{b_3} circuit. 109
A.8 K_{b_2} circuit. 110
A.9 K_{b_1} circuit- 111
A.10 $T^{\dagger}_{2y}$ circuit. 112
A.11 $T^{\dagger}_{1y}$ circuit. 112
A.12 Swap implementation. 113
A.13 https://bit.ly/2FFKqpX 120

List of Tables

6.1 First iterations using Eq. 6.1 on Fig. 6.1 58

Contents

Abstract v

List of Figures viii

List of Tables ix

1 Introduction 1
- 1.1 Problem definition 3
- 1.2 Objectives 4
- 1.3 Overview 4

2 Graph Theory for Quantum Walks 6
- 2.1 Basic definitions 6
- 2.2 Scale-free networks 11

3 A concise introduction to Quantum Mechanics 14
- 3.1 Postulates of Quantum Mechanics 19

4 Quantum walks 24
- 4.1 Discrete-time Quantum walks 25
  - 4.1.1 Coined based quantum walk 25
  - 4.1.2 Szegedy Quantum Walk 30
- 4.2 Equivalence of discrete time quantum walk and Szegedy's quantum walk 33

5 A concise introduction to QISKIT 36
- 5.1 Programming quantum circuits using Qiskit 37
- 5.2 Quantum Gates for Quantum circuit design 42
  - 5.2.1 Three-qubit gates 47
- 5.3 Higher order control gates 48

5.4 Circuit implementation of Quantum Walks 50
5.4.1 Szegedy Circuit 51
5.5 Qiskit experience 52

6 Classical and Quantum PageRank 56
6.1 Google Pagerank 56
6.2 Quantum PageRank Algorithm with Szegedy Walks 60
6.2.1 Szegedy circuit for the quantum PageRank algorithm 62

7 Simulations 66
7.1 Quantum PageRank algorithm 85

8 Discussion 89

9 Conclusions 95

A Appendix 98

Bibliography 118

Chapter 1

Introduction

Quantum computation is an interdisciplinary scientific field emerged from a cross-fertilisation between Quantum Mechanics and the Theory of Computation, and it is devoted to building computers and information processing systems that exploit the quantum mechanical properties of nature. Research in quantum computing is centered in building algorithms that make use of the quantum mechanical properties of those systems used to implement them.

Quantum algorithms will be useful in practice if we are able to simulate and efficiently implement them on a quantum computer. It is necessary to exploit the 'quantum advantage' which refers to the goal that a quantum computing will be able to outperform a classical computer, for certain processes at least. Hence, the reason why we study quantum algorithms.

Our goal is to explore several aspects of the realisation of quantum algorithms that take place in graphs, specifically on directed graphs such as scale free networks. We explore the implementation of a quantum mechanical version of PageRank, being its classical version an algorithm developed for ranking nodes according to their relevance. We also present the general framework and relevant details of our implementation of Quantum Page Rank on IBM's Q platform.

To describe the processes that take place in quantum algorithms, we situate ourselves in a frame of reference that integrates the following characteristics. First, to express any given computational process in terms of a set of basic elements, i.e. a universal model. Second, ease-of-implementation on available platforms. The best known models of quantum computation that solve specific problems and for ease-of-implementation on certain physical platforms, include models based on quantum circuits, the quantum adiabatic theorem and quantum walks.

Quantum walks have already been proved useful to provide quadratic [54] and exponential speedup [17]. Furthermore, quantum walks constitute a universal model of quantum computation so any quantum algorithm can be formulated as a quantum walk-based algorithm [16] and vice versa, i.e. it is also possible to reformulate a quantum walk-based algorithm in terms of another universal model of quantum computation, for instance the quantum circuit model [20], [35]. In this thesis, we consider the implementation of quantum walks using the quantum circuit model available in the IBM Q experience open source platform [28].

In the quantum walk model we encounter two types, the discrete time quantum walk and continuous time quantum walk, both taking place on a discrete position space defined by graphs. However, their evolution timing is different. The continuous case is based on evolving a quantum state according to the solution of the Schrödinger equation using Hamiltonian dynamics based on the Laplacian matrix or adjacency matrix of the graph. For the discrete case, the evolution is determined by the definition of shift and coin operators. The coined quantum walk was introduced by Y. Aharanov et al. [2]. The discrete case includes an alternative to the coined model, the Szegedy quantum walk introduced by M. Szegedy [54], which present some advantages over the first two previous models on directed graphs. In this research, we primarily study the discrete time Szegedy walk model which represent an advantage in the study of directed graphs, as proposed in [35]. A substantial amount of work has been done to implement quantum walk algorithms using efficient quantum circuits [35], [50], [20].

Network and graph theory is present in nearly all aspects of quantum information and computation [8]. Network science provides the framework to analyse and understand networks like the web or networks associated with transportation even biological and social networks [6]. As we will see, the quantum walk takes place in a space defined by a graph, thus fitting naturally in the study of network problems.

Recently, quantum computers have become available as a cloud-based service [32]. While currently available quantum computers have less than 100 qubits, quantum computer hardware is expected to grow in terms of total number of available qubits, quality, and connectivity. A collection of the principal quantum algorithms implemented on quantum software is presented in [18], [29].

In this thesis,

1. We translate the Quantum Page Rank algorithm presented in [35], originally stated in terms of Szegedy's quantum walks, into the quantum circuit model. This job is crucial for the advancement of quantum algorithms because expeditious recasting of quantum walk-based algorithms as quantum circuits is a new area of scientific research.

2. We shall implement and simulate the quantum circuits produced in the previous step on IBM Q Experience platform (both WWW-based service and QISKIT local services). In this step, we shall present some techniques for running quantum algorithms, originally stated in terms of quantum walks, on IBM's Q platform. Our results will contribute towards identifying the capacities and limitations of current IBM's Q platform.

1.1 Problem definition

The exploration and classification of massive amounts of data stored in complex networks such as the World Wide Web (WWW) is a key activity in contemporary science. A central problem of information retrieval is to classify such information by its relevance, i.e. data ranking. A successful procedure for this purpose is the famous PageRank algorithm proposed by L. Page and S. Brin [42].

Building quantum networks, potentially resembling the WWW in terms of size and complexity, is a crucial goal of quantum information science and technology [56, 58, 27]. In the hypothetical situation where large-scale quantum networks became a reality, classifying the quantum information stored in those systems would be a priority. In this thesis, we analyse the quantum equivalent of PageRank algorithm [43, 44] using the model of quantum walks and translated into the quantum circuit model.

As previosly stated in this chapter, quantum walks constitute a universal model of quantum computation, meaning that any problem that can be solved by an algorithm running on a general-purpose quantum computer can also be solved by a quantum walk-based algorithm [16, 57]. Due to the computational universality of quantum walks and the quantum circuit model, it is always possible to reformulate a quantum walk-based algorithm in terms of quantum gates [33].

The study of quantum walks on directed graphs is a real challenge that has not been extensively explored. Among the works on quantum walks on directed graphs, we encounter [37], in which Montanaro uses discrete time quantum walk, and gives a necessary and sufficient condition for running a quantum walk on a graph, named reversibility.

In this thesis, we implement a quantum gate-based algorithm that replicates the behaviour of the Quantum PageRank algorithm presented in [43], [44], [35]. Additionally, we argue in section 4.1.2 that the Szegedy model of quantum walk presented in [54] is particularly suited for working on problems that are defined on directed graphs because it, the Szegedy model, represents a robust abstraction of the coined model (Sec. 4.2).

The complexity of a quantum gate-based algorithm is measured in terms of the number of gates used as a function of number of bits of input. The quantum circuit implementation of quantum walks has been investigated on highly symmetric graphs [20] and the continuous-time quantum walk on the circulant class of graphs and other types [50].

1.2 Objectives

The main objective of this thesis is:

- To recast Szegedy's quantum walk model in terms of the quantum circuit model.
- To introduce a general mathematical framework to efficiently rewrite a Szegedy's quantum walk as a quantum circuit.
- To simulate the quantum PageRank algorithm as a quantum circuit, on IBM's quantum platform [28], [35].
- To discuss design principles for the development of efficient quantum circuit-based implementations of quantum walks.

Additionally, our particular objectives are:

- To calculate the average Quantum PageRank.
- To discuss implementation drawbacks on IBM's platform for Szegedy's quantum circuit.

1.3 Overview

In general, we explore how currently available tools and techniques within the realm of quantum computation could actually help computer practitioners to develop quantum algorithms. The discussion provided in this thesis can be used to obtain a succinct guide to some of the concepts of quantum mechanics needed to be initiated in the fields of quantum computation and quantum walks.

Now we provide an outline of our thesis.

Chapter 2 Graph theory. Graph theory is crucial to understand the dynamics of quantum walks. Hence, in this chapter we present several key concepts from graph theory, such as the different types of graphs in which quantum walks take place.

Chapter 3 Quantum mechanics. This chapter is a concise introduction to the postulates of quantum mechanics needed to set the mathematical basis employed in the study of quantum computation and algorithm development. This chapter has been written with two purposes in mind 1) to provide the necessary background for our work related to quantum walks 2) to serve as a concise guide for scientists from different areas whose desire is to initiate in the field of quantum computation.

Chapter 4 Discrete time quantum walks. In this chapter, we offer a concrete introduction to the main ideas behind quantum walks. We include a mathematical description of the parts that integrate a quantum walk and we define the operators involved in the definition of a quantum walk. Also, we provide our reader with resources that could be a good starting point for learning and doing research in the extensive area of quantum walks.

Chapter 5 A concise introduction to Qiskit. In this chapter we introduce Qiskit, IBM's accessible platform for quantum computing, which is also the platform in which we implemented and simulated our algorithm. The purpose of this chapter is to serve as a guide not just for computer scientists but anyone who wants to employ a quantum computer to implement solutions in their corresponding area.

Chapter 6 Classical and Quantum Pagerank. In this chapter, we explain the classical and quantum PageRank algorithms. We provide explicit relationships between these two algorithms and explain how the classical PageRank is connected with the Quantum PageRank through Markov chains.

Chapter 7 Simulations. We present the main results of our simulations on IBM platform providing step-by-step simulations for the correct implementation of our circuit.

Chapter 8 Discussion. In this chapter, we discuss the key ideas presented in the thesis. We include an analysis of the IBM's platform and the main issues faced. Also, we present in more detail the key concepts in the Markov chain,

Chapter 9 Conclusions. We present and summarise the main ideas developed in this thesis.

Chapter 2

Graph Theory for Quantum Walks

In this section, we introduce some notions of Graph Theory required to understand the main concepts in the study of quantum walks. For an extensive study in graph theory the readers may find references [7, 5, 24, 40, 9] relevant to deepening into the mathematical, physical and algorithmic properties of graphs and networks.

2.1 Basic definitions

Quantum walks (section 4) take place on graphs, which are mathematical structures defined by edges and vertices. In a classical random walk, we consider the vertices as the sites upon which the walker can move and the edges tell the possible directions the walker can move across vertices. In the context of quantum walks, vertices represent the states a quantum walker is allowed to take and edges represent quantum state transitions.

A *graph* consists of two sets, V and E. Each element of V is called a vertex. The elements of E, called edges, are unordered pairs of vertices. An *undirected graph*, or simply a graph, consists of two finite sets, V and E. Is denoted as $G(V, E)$ where $V = \{v_1, \ldots, v_N\}$ is a set of vertices and $E = \{(v_i, v_j), \ldots, (v_k, v_l)\}$ is the set of edges (links or lines). An undirected graph has the property $(v_i, v_j) \in E \iff (v_j, v_i) \in E$. When $(v_i, v_j) \in E$, we say that v_i is connected to v_j, and express it by $v_i \rightarrow v_j$. A graph is said to be *strongly connected* if every vertex is reachable from any other vertex. The degree (or valency) of vertex v is the number of edges incident to the vertex and is denoted by $\deg(v)$.

For instance, the set V might be $\{a, b, c, d\}$ and E might be $\{\{a, b\}, \{b, c\}, \{b, d\}, \{c, d\}\}$. Together V and E are a graph and a visual representation of G is presented in Fig. 2.1. Using the same graph G we illustrate the node degrees as follows. Graph G has $deg(a) = 1$, $deg(b) = 3$, $deg(c) = 2$, $deg(d) = 2$.

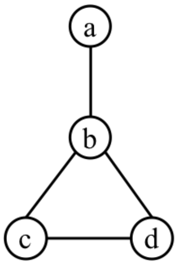


Figure 2.1: Visual representation of graph G.

A generalization of the concept of a simple graph that allows repeated elements in the set of edges between two vertices, is the notion of a *multigraph* Γ, shown in Fig. 2.2. Graphs with loops can be added to this class.

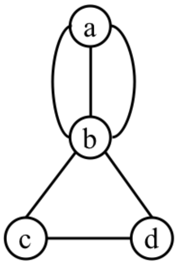


Figure 2.2: A multigraph

In directed graphs (see Fig. 2.3), we distinguish between incoming degree, $indeg(v)$, representing the number of links that point to node v and outgoing degree, $outdeg(v)$, representing the number of links that point from node v to adjacent nodes. A *directed graph or digraph*, $G = (V, A)$, is defined by a vertex set $V(G)$, an arc set $A(G)$ and a function assigning each arc an ordered pair of vertices, (i, j), where i is the tail and j is the head, and (i, j) is called directed edge or simply arc.

Let us use the digraph G in Fig 2.3 to illustrate indegree and outdegree. G has $indeg(a) = 0$, $indeg(b) = 2$, $indeg(c) = 1$, $indeg(d) = 2$; $outdeg(a) = 1$, $outdeg(b) = 1$, $outdeg(c) = 2$, $outdeg(d) = 1$.

Now, let $\vec{G}(V, A)$ be a directed graph such that (i, j) and (j, i) are in $A(\vec{G})$ if and only if $\{i, j\} \in G$. $\vec{G}$ is a symmetric digraph, whose underlying graph is G, as in Fig. 2.4.

In addition to visual representations, we can write down graphs via matrix representations. One such representation is the adjacency matrix. For any graph $G(V, E)$, the adjacency matrix A is an $N \times N$ matrix defined according to Eq. (2.1)

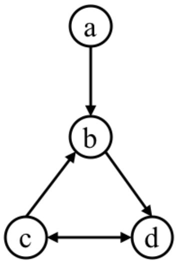


Figure 2.3: A directed graph or digraph.

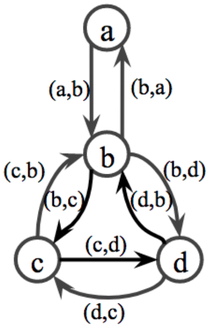


Figure 2.4: Symmetric digraph

$$A_{i,j} = \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Note that the adjacency matrix A of an undirected graph G must be symmetric, i.e. $A = A^T$, equal to its transpose. This is because $(v_i, v_j) \in E \Rightarrow (v_j, v_i) \in E$. For the undirected graph in Fig. 2.1 with four vertices, we see that the entries of its adjacency matrix A are symmetric with respect to the main diagonal.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Directed graphs do not have a symmetric adjacency matrix $(A \neq A^T)$. We can observe this fact from its adjacency matrix below

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

From the adjacency matrix we define the corresponding transition matrix P

$$P_{i,j} = \frac{A_{i,j}}{\sum_{i=1}^{n} A_{i,j}} \tag{2.2}$$

where $\sum_{i=1}^{n} A_{i,j}$ is the in-degree of vertex j and $P_{i,j}$ is the transitions probability of $v_j \to v_i$ with row or column normalization $\sum_{i=1}^{N} P_{i,j} = 1$ and $p_{i,j} \geq 0$. For Fig. 2.1 the transition matrix is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$$

The transition matrix P is used to describe the transitions of a *Markov chain*. A Markov chain is a stochastic process whose immediate future state depends only on its present state. A classical discrete-time *stochastic process* is a sequence of random variables $X_{0,}, X_1, X_2, ...$ denoted by $\{X_t : t \in \mathbb{N}\}$. X_t is the state of the stochastic process at time t and X_0 is the initial state. A random walk can be described by a Markov chain. In Sec. 6.1, we provide a precise relationship between the PageRank algorithm and Markov chains.

A graph G is *d-regular* if all its vertices have degree d, that is, each and every vertex in G has exactly d neighbors. A non regular graph (or irregular graph) is a graph in which, for every vertex, neighbors of that vertex have distinct degrees. We can obtain a *degree matrix* D whose diagonal elements contain information about the degree of each vertex. It is used together with the adjacency matrix to construct the Laplacian matrix of a graph.

The *Laplacian matrix* L of a simple graph is defined as $L = D - A$, where D is the degree matrix and A is the adjacency matrix of the graph.

$$L_{i,j} = \begin{cases} \deg(v), & \text{if } v_i = v_j \\ -1, & \text{if } \{v_i, v_j\} \in E \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

Considering the graph in Fig. 2.1 its degree matrix is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Together with the adjacency matrix defined before, we obtain the Laplacian matrix of graph in Fig. 2.1

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}$$

Bipartite graph. A graph G is bipartite if its vertex set can be partitioned into two sets X and Y in such a way that every edge of G has one end vertex in X and the other in Y. In this case, X and Y are called the partite sets. A vertex in X is connected to a vertex in Y if and only if they are connected in the original graph, as shown in Fig. 2.5.

Note that the number of edges in its bipartite double cover is $2|E|$.

Consider a connected bipartite graph $\Gamma(X, Y, E)$, where X, Y are disjoint sets of vertices, and E is the set of non-directed edges. The adjacency matrix of $\Gamma(X, Y, E)$ (considered as biadjacency matrix) is

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \tag{2.4}$$

Szegedy's quantum walk (as we will see in section 4.1.2) occurs on the edges of the bipartite double cover of the original graph.

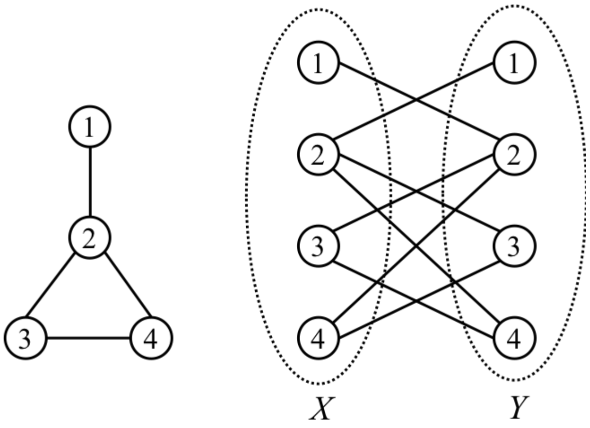


Figure 2.5: Generation of the double cover bipartite graph from an irregular graph of $N = 4$ vertices.

Complete Graph. A graph is said to be complete if every vertex is adjacent to every other vertex. A complete graph of order n is denoted as K_n. Figure 2.6 shows show some examples for different K_n

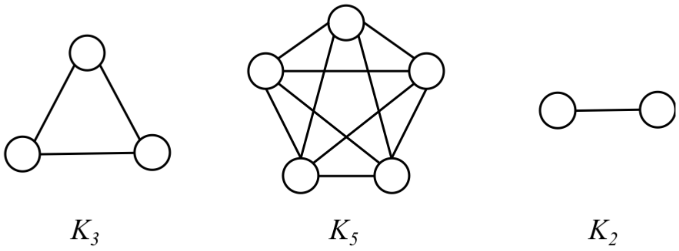


Figure 2.6: Examples of complete graphs

Cycles. A cycle is a circuit with a vertex sequence in which the first and the last vertex are repeated, i.e, $(v_1, v_2, \ldots, v_n, v_1)$. We denote the graph C_n as a cycle on n vertices. For example, a C_7 cycle is shown in Fig. 2.7

2.2 Scale-free networks

A scale-free network is a network whose degree distribution follows a power law (see Fig. 2.8). That is, the probability that a randomly selected node has degree k is:

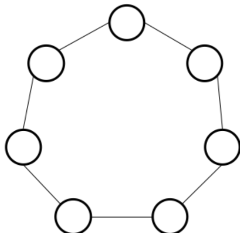


Figure 2.7: Graph for C_7

$$P(k) \sim k^{-\gamma}$$

where γ is its degree exponent that ranges between 2 and 3. It implies that few nodes have great connectivity and most nodes have only a few links. These numerous small nodes are held together by a few highly connected hubs [7].

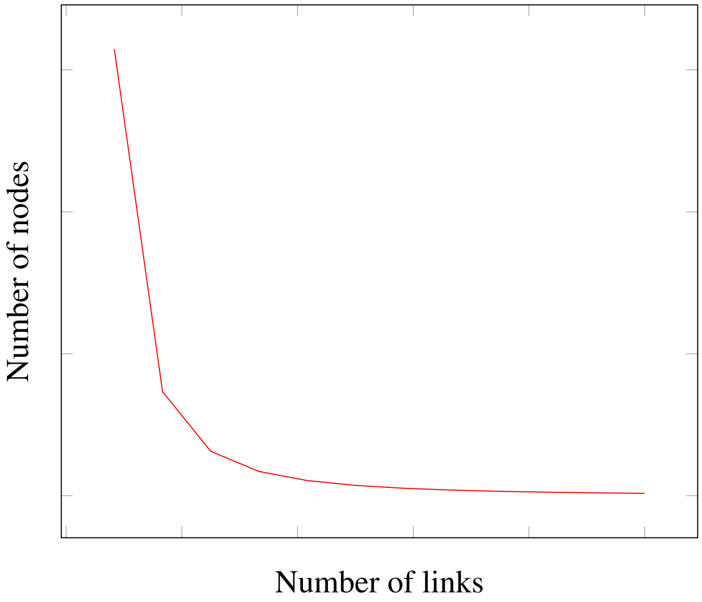


Figure 2.8: Power law distribution

Many biological, social, and technological systems are described by complex networks whose nodes represent individuals or elements, and links simulate the interactions among them [12]. Many of these networks follow the scale-free property, which in turn results in a power law distribution for the probability that a node of the network has a determined number of connections to other nodes [6].

A particular example of a scale free network is the internet and the WWW which have been extensively studied because of the technological and economic relevance. While the terms WWW and the Internet are regularly used interchangeably, we point out that they refer to different systems. The WWW is an information network whose nodes are documents and links are URLs. In contrast, the internet is an infrastructural network, whose nodes are computers called routers and whose links correspond to physical connections, like optical cables or wireless links [7]. The World Wide Web is a directed network whose nodes are documents and the links are the uniform resource locators (URLs) that allow us to surf with a click from one web document to the other.

The WWW is a directed network, hence each document is characterized by an out-degree k_{out}, representing the number of links that point from the documents, and an in-degree k_{in}, representing the number of documents that point at the selected document. So, we distinguish two degree distributions: the probability $p_{k_{out}} \sim k^{-\gamma_{out}}$ that a randomly selected document points to k_{out}, web documents, and the probability $p_{k_{in}} \sim k^{-\gamma_{in}}$ that a randomly chosen node has k_{in} web documents pointing to it. In this case, both probabilities can be approximated by a power law. The scale-free property applies to both in and out degrees [3].

In order to study systems that follow a power law using quantum walks, we require a formalism that supports directed graphs. In this thesis, we investigate the properties of the quantum PageRank algorithm that works on real world networks such as the WWW. In order to study these systems a suitable formalism is required, the Szegedy (Ch. 4.1.2) formalism is utilized.

Chapter 3

A concise introduction to Quantum Mechanics

In this section we summarize the core background of quantum mechanics [21, 36] for quantum computing that includes important definitions employed in the study of quantum algorithms. A major purpose of this section is clarify the mathematical notations and terminology. Then, we proceed to define the postulates of quantum mechanics.

Quantum mechanics represent the most complete and reliable description of the world at a microscopic level and beyond known so far. It is also the foundation of quantum computation and quantum information [41]. We start by introducing a general mathematical formalism based on linear algebra.

The basic elements in linear algebra are vector spaces. The elements of a vector space are vectors. The vector space of interest is $\mathbb{C}^n$, a space defined by n-tuples, $(z_1, \ldots, z_n)$, of complex numbers. There is an addition operation which takes pairs of vectors to generate other vectors. Also, in a vector space, there is a scalar multiplication operation. We define the notions of vector addition and scalar multiplication as follows.

- Vector addition. This is a binary operation that takes a pair of vectors $\mathbf{x}, \mathbf{y} \in \mathbb{V}$ to produce a another vector $\mathbf{x} + \mathbf{y} \in \mathbb{V}$.
- Scalar multiplication. This operation takes a vector $\mathbf{x} \in \mathbb{V}$ and a scalar $c \in \mathbb{F}$ to produce another vector $c\mathbf{x}$.

Definition 1 *Let $\mathbb{V}$ be a set associated to a field $\mathbb{F}$. The elements of $\mathbb{V}$ are called vectors. The elements of $\mathbb{F}$ are known as scalars. Set $\mathbb{V}$, together with a field $\mathbb{F}$ and the operations known as vector addition and scalar multiplication, is known as a vector space iff it satisfies the following axioms.*

1. **Closure under addition**.
   $\forall \mathbf{x}, \mathbf{y} \in \mathbb{V} \implies \mathbf{x} + \mathbf{y} \in \mathbb{V}$.
2. **Commutativity of addition**.
   $\forall \mathbf{x}, \mathbf{y} \in \mathbb{V} \implies \mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$.
3. **Associativity of addition**.
   $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{V} \implies \mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$.
4. **Additive identity**.
   $\exists ! \mathbf{0} \in \mathbb{V}$ such that $\forall \mathbf{x} \in \mathbb{V} \implies \mathbf{x} + \mathbf{0} = \mathbf{0} + \mathbf{x} = \mathbf{x}$.
5. **Additive inverse**.
   For each $\mathbf{x} \in \mathbb{V}, \exists ! -\mathbf{x} \in \mathbb{V}$ such that $\mathbf{x} + (-\mathbf{x}) = -\mathbf{x} + \mathbf{x} = \mathbf{0}$.
6. **Closure under multiplication**.
   $\forall \mathbf{x} \in \mathbb{V}, \alpha \in \mathbb{F} \implies \alpha \mathbf{x} \in \mathbb{V}$.
7. **Commutativity of multiplication**
   $\forall x, y \in \mathbb{F} \implies xy = yx$.
8. **Associativity of multiplication**
   $\forall \mathbf{x} \in \mathbb{V}, \alpha, \beta \in \mathbb{F} \implies \alpha(\beta \mathbf{x}) = (\alpha\beta)\mathbf{x}$.
9. **Multiplicative identity**.
   $\exists \in \mathbb{F}$, such that, $\forall \mathbf{x} \in \mathbb{V} \implies \mathbf{x}1 = 1\mathbf{x} = \mathbf{x}$.
10. **Multiplicative inverse**.
    For each $x \in \mathbb{F} - \{0\}$ $\exists x^{-1} \in \mathbb{F}$ such that $xx^{-1} = x^{-1}x = 1$.
11. **Distributivity of multiplication over addition**
    $\forall \mathbf{x} \in \mathbb{V}, \alpha, \beta \in \mathbb{F} \implies (\alpha + \beta)\mathbf{x} = \alpha \mathbf{x} + \beta \mathbf{x}$.
    $\forall \mathbf{x}, \mathbf{y} \in \mathbb{V}, \alpha \in \mathbb{F} \implies \alpha(\mathbf{x} + \mathbf{y}) = \alpha \mathbf{x} + \alpha \mathbf{y}$.

Now, we define important mathematical preliminaries that will allow us to introduce the central concepts in quantum computing.

Definition 2 ***Functional.*** *Let $\mathbb{V}$ be a vector space over a field $\mathbb{F}$. A linear functional is a linear function $f : \mathbb{V} \to \mathbb{F}$*

Definition 3 ***Inner-product vector space.*** *An inner product vector space $\mathbb{V}$, is a complex vector space, equipped with an inner product $\langle \cdot | \cdot \rangle : \mathbb{V} \times \mathbb{V} \to \mathbb{C}$*

Definition 4 ***Complete inner-product vector space.*** *An inner-product vector space* $\mathbb{V}$ *is denoted as complete if for any sequence* $\{\mathbf{a}_i\}_{i=1}^{\infty}, \mathbf{a}_i \in \mathbb{V}$ *with the property* $\lim_{i,j\to\infty} ||\mathbf{a}_i - \mathbf{a}_j|| = 0$*, there is a unique element* $\mathbf{b} \in \mathbb{V}$ *such that* $\lim_{i,j\to\infty} ||\mathbf{b} - \mathbf{a}_j|| = 0$

Now, we define an important vector space, the Hilbert space.

Definition 5 ***Hilbert Space.*** *A Hilbert space* $\mathcal{H}$ *is a complete complex inner-product vector space.*

An example of a Hilbert space is $\mathbb{C}^2(\mathbb{C})$, the complex bidimensional vector space defined over the field of complex numbers.

$$\mathbb{C}^2(\mathbb{C}) = \left\{ \binom{a}{b} | a, b \in \mathbb{C} \text{ and scalars } \alpha \in \mathbb{C} \right\}$$

Any vector in the vector space can be written as a linear combination $|v\rangle = \sum_i a_i |v_i\rangle$ of vectors in that set. A spanning set for a vector space is a set of vectors $|v_1\rangle, \ldots, |v_n\rangle$.

The standard notation of quantum mechanics for a vector in a vector space is known as *Dirac notation*, also known as *Bra-Ket* notation. This notation is a convenient way to represent vectors and operations in linear algebra. The formal definition of Ket and Bra is as follows.

Definition 6 ***Dirac notation.*** *Let* $\mathcal{H}$ *be a Hilbert space. A vector* $|\psi\rangle \in \mathcal{H}$ *is denoted* $|\psi\rangle$ *and is referred as* **ket**. *The corresponding linear functional is denoted* $\langle\psi|$ *and is referred as* **bra**. *Thus,* $\langle\cdot|$ *can be seen as an operator that maps each state* ϕ *into a functional* $\langle\phi|$ *such that* $\langle\phi|(|\psi\rangle) = \langle\phi|\psi\rangle$*. We define* $|\psi\rangle^\dagger \equiv \langle\psi|$

Ket. Let $\mathcal{H}$ be a Hilbert space. A vector $\psi \in \mathcal{H}$ is denoted by $|\psi\rangle$ and it is referred to as a ket. When written using matrix notation, it is customary to write kets as column vectors.

For instance, let $\mathcal{H} = \mathbb{C}^2$ and let us choose the vector basis $\{|0\rangle, |1\rangle\}$, where

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

Then, every element $|\psi\rangle \in \mathcal{H}$ can be written as

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

Bra. Formally speaking, Bras are functionals (i.e. functions of vector spaces into corresponding fields) and in practice, they can be thought of as row vectors:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \text{ if and only if } \langle\psi| = \alpha^* \langle 0| + \beta^* \langle 1|$$

where $\alpha, \beta, \alpha^*, \beta^* \in \mathbb{C}$. α^* and β^* are the complex conjugate of α and β, respectively and $\langle 0| = (1, 0)$ and $\langle 1| = (0, 1)$

In general, if $\mathcal{H}$ is an n-dimensional Hilbert space then a ket $|\psi\rangle \in \mathcal{H}$ can be represented as an n-dimensional column vector. Its corresponding bra $\langle\psi| \in \mathcal{H}^*$ can be seen as an n-dimensional row vector. We represent this idea in the following example. In Dirac notation:

$$\mathbf{v} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix} = |\mathbf{v}\rangle$$

as well as for the dual vector of $|v\rangle$, the bra:

$$\overline{\mathbf{v}^T} = \begin{bmatrix} v_0^* & v_1^* & \dots & v_n^* \end{bmatrix} = \langle \mathbf{v}|$$

where $\overline{v}$ and v_i^* are two ways to denote the complex conjugate of v.

An inner product is a function which takes as input two vectors $|a\rangle$ and $|b\rangle$ from a vector space and produces a complex number as output.

Definition 7 ***Inner Product.*** *Let $\mathbb{V}(\mathbb{C})$ denote a vector space $\mathbb{V}$ defined over the set of complex numbers $\mathbb{C}$. Also, let $|a\rangle \,.\, |b\rangle \in \mathbb{V}(\mathbb{C})$. We define the inner product function as follows.*

$$\langle\cdot|\cdot\rangle : \mathbb{V} \times \mathbb{V} \to \mathbb{C}$$

with the following properties

1. $\forall\, |a\rangle \in \mathbb{V} \implies (|a\rangle\, , |a\rangle) \geq 0$ and $(|a\rangle\, , |a\rangle) = 0 \iff |a\rangle = 0$
2. $\forall\, |a\rangle\, , |b\rangle \in \mathbb{V} \implies (|a\rangle\, , |b\rangle) = (|b\rangle\, , |a\rangle)^*$
3. $\forall\, |a\rangle\, |b\rangle_i \in \mathbb{V}, \alpha_i \in \mathbb{C}, i \in \mathbb{N} \implies (|a\rangle\, , \sum_i \alpha_i\, |b\rangle_i) = \sum_i \alpha_i(|a\rangle\, , |b\rangle_i)$

For instance, the inner product in $\mathbb{C}^n$ is

$$\langle a|b\rangle = \overline{\mathbf{a}^T}\mathbf{b} = \begin{bmatrix} a_1^* & a_2^* & \dots & a_n^* \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = a_0^* b_0 + a_1^* b_1 + \ldots + a_n^* b_n = \sum_{i=1}^{n} a_i^* b_i$$

which is the usual row-column matrix multiplication, where a_i^* is the complex conjugate of $a_i, \forall i\; n\{1, \ldots, n\}$

Using the Dirac notation, and considering $|\phi\rangle, |\psi\rangle \in \mathbb{C}^2$ the inner product is denoted as

$$(|\phi\rangle, |\psi\rangle) = \langle\phi| |\psi\rangle = \langle\phi|\psi\rangle$$

The inner product on the vector basis introduced before we obtain

$$\begin{aligned}\langle 0|0\rangle = \langle 1|1\rangle = 1,\\ \langle 0|1\rangle = \langle 1|0\rangle = 0.\end{aligned}$$

We need to define one more operation, the outer product. To do so, first we define the concept of Linear Operators.

Definition 8 ***Linear operator.*** *A linear operator between vectors spaces $\mathbb{V}$ and $\mathbb{W}$ is defined as any function $\hat{A} : \mathbb{V} \to \mathbb{W}$ which is linear in its inputs,*

$$\hat{A}\left(\sum_i \alpha_i |\psi_i\rangle\right) = \sum_i \alpha_i \hat{A} |\psi_i\rangle$$

To illustrate the previous definition let's consider the Adjoint/Hermitian conjugate operator

Definition 9 ***Adjoint/Hermitian Conjugate Operator.*** *Let $\hat{A} : \mathcal{H} \to \mathcal{H}$ be a linear operator that induces the map $|\psi\rangle \to |\psi'\rangle$. The operator $\hat{A}^\dagger$, known as $\hat{A}$ dagger, the adjoint of $\hat{A}$ or the Hermitian Conjugate of $\hat{A}$, induces the map $\langle\psi| \to \langle\psi'|$ on the corresponding bras.*

In other words,

$$\begin{aligned}\hat{A}|\psi\rangle = |\psi'\rangle\\ \langle\psi| \hat{A}^\dagger = \langle\psi'|\end{aligned}$$

In matrix notation, this operation, $\hat{A}^\dagger$, is $(A^t)^*$ where t denotes transposition and $*$ denotes complex conjugation.

The outer product, denoted $|v\rangle\langle u|$ or $|v\rangle \otimes \langle u|$, is known as the tensor or Kronecker product of $|v\rangle$ with the conjugate transpose of $|u\rangle$. The result in contrast with the inner product is not a scalar but a matrix:

$$|v\rangle\langle u| = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix} \begin{bmatrix} u_0 & u_1 & \dots & u_n \end{bmatrix} = \begin{bmatrix} v_0 u_0^* & v_0 u_1^* & \dots & v_0 u_n \\ v_1 u_0^* & v_1 u_1^* & \dots & v_1 u_n \\ \vdots & \vdots & \ddots & \vdots \\ v_n u_0^* & v_n u_1^* & \dots & v_n u_n^* \end{bmatrix}$$

The outer product is crucial in describing linear transformation between vectors spaces [53]. Dirac notation proves to be elegant and simple to describe transformation matrices like the one depicted above. Let $|\psi\rangle, |a\rangle \in \mathcal{H}_1$ and $|\phi\rangle \in \mathcal{H}_2$ then the outer product is the linear operator from $\mathcal{H}_1$ to $\mathcal{H}_2$ defined by

$$(|\phi\rangle, \langle\psi|)\,|a\rangle \equiv (\langle\psi|a\rangle)\,|\phi\rangle$$

As a result, the summation of outer products is also a linear operator.

We encounter the tensor product in most operations in quantum computation. The tensor product between two vectors $|u\rangle \otimes |v\rangle$ is used so frequently that is simplified to $|u\rangle\,|v\rangle$ or $|uv\rangle$. A tensor product of a vector with itself n times is denoted as $|v\rangle^{\otimes n}$. We show the convenience of using this notation in the following example:

$$|u\rangle\,|v\rangle = |uv\rangle = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_m \end{bmatrix} \otimes \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_0 \cdot v_0 \\ u_0 \cdot v_1 \\ \vdots \\ u_0 \cdot v_n \\ u_1 \cdot v_0 \\ \vdots \\ u_{m-1} \cdot v_n \\ u_m \cdot v_n \\ \vdots \\ u_m \cdot v_n \end{bmatrix}$$

Tensor products are helpful since they are a mathematical abstractions that describes interactions between two quantum systems.

3.1 Postulates of Quantum Mechanics

In this subsection we give a concise description of the basic postulates of quantum mechanics. These postulates provide a connection between the physical world and the mathematical formalism of quantum mechanics.

The first postulate of quantum mechanics lays the foundation in which quantum mechanics takes place.

State Space

Postulate 1 *Any isolated physical system has an associated Hilbert space $\mathcal{H}$, known as the state space of the system. The physical system is completely described by its* state vector*, which is a unit vector,* $|\psi\rangle \in \mathcal{H}$*, in the system's state space.*

The simplest quantum mechanical system is the qubit. In quantum computing, information is stored, manipulated and measured in the form of qubits. A qubit is an element of a two-dimensional state space $|\psi\rangle \in \mathcal{H}^2$, thus a qubit $|\psi\rangle$ may be written in general form as

$$|\psi\rangle = \alpha\,|p\rangle + \beta\,|q\rangle$$

where $\alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1$ and $\{|p\rangle\,, |q\rangle\}$ is an arbitrary basis spanning $\mathcal{H}^2$. The special states $|0\rangle$ and $|1\rangle$ are known as *computational basis states*, and form an orthonormal basis for this vector space.

Postulate 1 implies that any vector $|v\rangle$ in the vector space can be written as a linear combination $|v\rangle = \sum_i \alpha_i\,|v_i\rangle$ of vectors in that set. This is known as the superposition principle. For example, the state

$$\frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

is a superposition of the states $|0\rangle$ and $|1\rangle$ with amplitude $1/\sqrt{2}$ for the state $|0\rangle$, and amplitude $-1/\sqrt{2}$ for the state $|1\rangle$

To perform operations in quantum computing, we take qubits as registers. The state space of a quantum register of size n is represented as a linear combination of n basis vectors, each of length 2^n as follows:

$$|\psi_n\rangle = \sum_{i=0}^{2^n-1} a_i\,|i\rangle \tag{3.1}$$

As an example, a three-qubit register would have the following expansion:

$$|\psi_3\rangle = a_0\,|000\rangle + a_1\,|001\rangle + a_2\,|010\rangle + a_3\,|011\rangle + a_4\,|100\rangle + a_5\,|101\rangle + a_6\,|110\rangle + a_7\,|111\rangle$$

or in the vector form, using the computational basis [53]

$$|\psi_2\rangle = a_0\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + a_1\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix} + a_2\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix} + a_3\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix} + a_4\begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix} + a_5\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix} + a_6\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix} + a_7\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$$

Quantum Evolution

Postulate 2 *The evolution of a closed quantum system with state vector $|\Psi\rangle$ is described by a unitary operator. That is, the state of the system at time t_1 is related to the state of the system at time t_2 by the unitary operator $\hat{U}$ which depends only on times t_1 and t_2. Some examples of Unitary operators acting on a single qubit that are important in quantum computation and quantum information includes the Pauli matrices, the Hadamard gate (as described in section 5.2).*

$$|\Psi(t_2)\rangle = \hat{U}\,|\Psi(t_1)\rangle$$

Postulate 2 describes the change of a quantum mechanical system with time. It describes the mathematical properties that an evolution operator must have. An alternative way to write a quantum evolution is in terms of the Schrödinger equation, it describes quantum evolution in continuous time:

$$i\hbar\frac{\partial\,|\psi(t)\rangle}{\partial t} = \hat{\mathbf{H}}\,|\psi(t)\rangle$$

where $\hbar = h/2\pi$, h is the Planck's constant ($6.62607004 \times 10^{-34} J \cdot s$) and $\hat{\mathbf{H}}$ is a Hermitian operator known as the Hamiltonian of the system. If the Hamiltonian is known then we fully understand the dynamics of the system whose evolution is described by the Schrödinger equation.

Quantum Measurement

Postulate 3 *Let $|\psi\rangle \in \mathcal{H}^n$ be the state of a quantum system immediatly before measurement. Also, let $\{a_i\}$ be the set of measurement outcomes and $\hat{M}_{a_i} = \langle i|i\rangle$ be the set of measurement operators built using basis $B = \{|i\rangle\}$, where $i \in \{0, 1, \ldots, n-1\}$. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that result a_i occurs is given by*

$$p(a_i) = \langle\psi|\hat{M}^{\dagger}_{a_i}\hat{M}_{a_i}|\psi\rangle$$

and the post-measurement quantum state that corresponds to measurement outcome a_i is given by

$$\frac{\hat{M}_{a_i}\,|\psi\rangle}{p(a_i)}$$

where $\hat{M}^{\dagger}_{a_i}$ is the result of applying the dagger operator † to the projection operator $\hat{M}_{a_i}$.

Measurement operators satisfy the completeness equation, $\sum_a M_{a_i}^\dagger M_{a_i} = I$. The completeness equation expresses the fact that probabilities sum to one: $1 = \sum_m p(a_i) = \sum_a \langle\psi|M_{a_i}^\dagger M_{a_i}|\psi\rangle$.

As an example, let us consider a measurement on a single qubit in the computational basis. This a measurement with two outcomes defined by the two measurements operators $M_0 = |0\rangle\langle 0|$, $M_1 = |1\rangle\langle 1|$. Observe that each measurement operator is Hermitian, thus the completeness relation is obeyed, $I = M_0^\dagger M_0 + M_1^\dagger M_1 = M_0 + M_1$. Suppose the state being measured is

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

.

Then the probability of obtaining measurement outcome 0 is

$$\begin{aligned} p(a_0) = \langle\psi|M_{a_0}^\dagger M_{a_0}|\psi\rangle &= (\frac{1}{\sqrt{2}}\langle 0| + \frac{1}{\sqrt{2}}\langle 1|)[(|0\rangle\langle 0|)(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle)] \\ &= (\frac{1}{\sqrt{2}})^2\langle 0|0\rangle + (\frac{1}{\sqrt{2}})^2\langle 1|0\rangle \\ &= \frac{1}{2} \end{aligned}$$

Now, we compute the post-measurement quantum state $|\psi\rangle_{pm}^{a_0}$

$$|\psi\rangle_{pm}^{a_0} = \frac{\hat{M}_{a_0}}{\sqrt{p(a_0)}} = \frac{\frac{1}{\sqrt{2}}|0\rangle}{\frac{1}{\sqrt{2}}} = |0\rangle$$

So, the post measurement state is $|\psi\rangle_{pm}^{a_0} = |0\rangle$. Applying the same procedure for the state 1 we obtain a probability $p(a_1) = \frac{1}{2}$, and computing the post-measurement quantum state $|\psi\rangle_{pm}^{a_1} = |1\rangle$.

Composite Quantum Systems

Consider a qubit $|\psi\rangle \in \mathcal{H}^2$. We know from postulate 3 that measuring $|\psi\rangle$ will produce one of two mutually exclusive outcomes, either 0 or 1. For composite systems, consider $|\psi\rangle_1 \in \mathcal{H}_1$ where $|\psi\rangle_1 = a|0\rangle + b|1\rangle$ and $|\psi\rangle_2 \in \mathcal{H}_2$ where $|\psi\rangle_2 = c|0\rangle + d|1\rangle$. If we measure both qubits at once, the possible results are: 0 and 0, 0 and 1, 1 and 0, 1 and 1. The proposed mathematical representation of s system composed by two qubits is

$$|\phi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$$

where $|\phi\rangle$ is the result of mixing $|\psi\rangle_1$ and $|\psi\rangle_2$, $\alpha_0, \alpha_1, \alpha_2, \alpha_3 \in \mathbb{C}$ and $\sum_i ||\alpha_i||^2 = 1$. $|\phi\rangle$ is an example of a method that describes multipartite quantum system: the tensor product.

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

Postulate 4 *The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. If we have* n *quantum systems,* $|\psi\rangle_1, |\psi\rangle_2, \ldots, |\psi\rangle_n$ *then the joint state of the total system is* $|\psi\rangle_T = |\psi\rangle_1 \otimes |\psi\rangle_2 \otimes \ldots \otimes |\psi\rangle_n$.

As metioned before, we may write $|a\rangle \otimes |b\rangle$ as $|ab\rangle$ or $|a, b\rangle$. In general, the tensor product of $|a\rangle$ with itself n times may be written as $|a\rangle \otimes |a\rangle \otimes \ldots \otimes |a\rangle = |a\rangle^{\otimes n}$. Consider $|101\rangle$, it represents an integer value of 5:

$$\begin{aligned} |101\rangle &= |1\rangle \otimes |0\rangle \otimes |1\rangle \\ &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}^T \end{aligned}$$

Quantum registers are an extension of quantum bits. Understanding how quantum registers work and its mathematical description allow us to think about how the state of a quantum register can evolve over time.

Chapter 4

Quantum walks

Quantum walks, the quantum mechanical counterpart of random walks [45, 51], were originally designed to model quantum phenomena [2] as well as a mathematical tool for building quantum algorithms that has been shown to constitute an universal model of quantum computation [16, 57]. There are two kinds of quantum walks: discrete and continuous quantum walks. The main difference between these two sets is the timing used to apply corresponding evolution operators. In the case of discrete quantum walks, the corresponding evolution operator of the system is applied only in discrete time steps, while in the continuous quantum walk case, the evolution operator can be applied at any time.

Two popular models of discrete-time quantum walk are coined quantum walks and Szegedy's quantum walks. In both models, time runs in discrete steps, hence $t \in \mathbb{N}$. The discrete time quantum walk on the line was one of the first quantization models of classical random walks. The generalization of regular random walks on finite graphs was introduced in Ref. [1]. Using a different quantization procedure, Szegedy [54], proposed a new coinless discrete time quantum walk model on bipartite graphs (Fig. 2.5). In [62] and [48], it is shown the equivalence of the the coined quantum walk and the Szegedy's quantum walk. As for the continuous-time quantum walk, it evolves according to the Schrödinger equation. In this thesis, we focus on discrete-time quantum walks.

In the next subsections, we center our attention to describe the discrete model as it represents a good approach for the problem stated. Finally, we make an analysis of the equivalence of the Szegedy's quantum walk and the coined quantum walk [62, 48] by giving explicit relationship between corresponding evolution operators.

4.1 Discrete-time Quantum walks

4.1.1 Coined based quantum walk

A coined discrete quantum walk is composed of two quantum systems known as the walker and the coin, evolution operators for both quantum systems and a set of measuring operators.

The walker is a quantum system living in a Hilbert space of infinite but countable dimension $\mathcal{H}_P$ while the coin lives in a 2-dimensional Hilbert space $\mathcal{H}_C$. We denote the walker as $|\text{position}\rangle \in \mathcal{H}_P$, and the valid states for position includes the computational basis states $|i\rangle_P$, as well as any superposition in the form $\sum_i \alpha_i |i\rangle_P$. The walker is usually initialized at the origin, i.e., $|\text{position}\rangle_{initial} = |0\rangle_P$.

The coin is a quantum system living in a Hilbert space, $|\text{coin}\rangle \in \mathcal{H}_C$. The coin may take the canonical basis states $|0\rangle$ and $|1\rangle$ as well any superposition of these basis states. A general normalized state of the coin may be written as $|\text{coin}\rangle = a\,|0\rangle_C + b\,|1\rangle_C$, where $|a|^2 + |b|^2 = 1$.

A single step in the dynamics of a coined quantum walk consists of the following two operations:

a) A coin toss, which in the context of quantum mechanics is equivalent to applying a Unitary operator $\hat{C}$ to the quantum coin.

b) A shift operator $\hat{S}$ which is a bipartite Unitary operator to be applied to both coin and walker in order to quantum mechanically diffuse the walker over the graph upon which the quantum walk is being run.

The total state of the system $|\psi\rangle \in \mathcal{H}$ is described by the tensor product of the walker Hilbert space $\mathcal{H}_P$ and the coin Hilbert space $\mathcal{H}_C$, i.e. $\mathcal{H}_t = \mathcal{H}_P \otimes \mathcal{H}_C$.

We write the step operator $\hat{U}$ for the coined quantum walk as shown in Eq. (4.1):

$$\hat{U} = \hat{S} \otimes \hat{C} \tag{4.1}$$

As described above, the coin operator $\hat{C}$ is applied first, followed by the shift operator $\hat{S}$. Considering a initial state $|\psi_0\rangle$ we obtain the state of the system at time t by applying the step operator t times, according to Eq.(4.2)

$$|\psi_t\rangle = \hat{U}^t\,|\psi\rangle_{initial} \tag{4.2}$$

At some point, we must perform a measurement operation (see Section 3.1) in order to know the outcome of the walk. To do so, we define a set of observables related to the basis states defined in the coin and walker operator. For instance, we perform a measurement on the coin using the observable

$$\hat{M}_c = \alpha_0 \left|0\right\rangle_c \left\langle 0\right| + \alpha_1 \left|1\right\rangle_c \left\langle 1\right|$$

.

Then, a measurement is performed on the position states of the walker by using the operator

$$\hat{M}_p = \sum_i \left|i\right\rangle_p \left\langle i\right|$$

Coined quantum walk on an unrestricted line. In this case, we run a quantum walk on line, i.e. a one dimensional graph with all nodes connected only to two other adjacent nodes (Fig. 4.1).

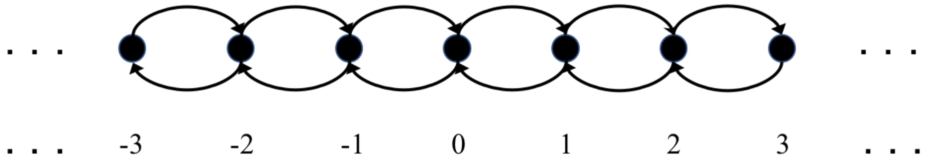


Figure 4.1: An unrestricted walk on a line.

The walker's position on the line is described by a vector $|n\rangle$ in a Hilbert space $\mathcal{H}_P$ of infinite but countable dimension. We take as basis of $\mathcal{H}_P$ the computational basis $\{|n\rangle : n \in \mathbb{Z}\}$.

The Hilbert space of the system is $\mathcal{H} = \mathcal{H}_C \otimes \mathcal{H}_P$, where $\mathcal{H}_C$ is the two-dimensional Hilbert space associated with the coin whose computational basis is $\{\left|0\right\rangle, \left|1\right\rangle\}$.

As stated above, we apply an evolution operator to the coin state followed by a conditional shift operator to the total quantum system. The effect of applying the coin operator is a superposition of the coin state.

The coin operator most frequently used is the Hadamard operator, presented in Eq. 4.3 and with matrix representation given by Eq. (4.4)

$$\hat{H} = \frac{1}{\sqrt{2}}(\left|0\right\rangle_c \left\langle 0\right| + \left|0\right\rangle_c \left\langle 1\right| + \left|1\right\rangle_c \left\langle 0\right| - \left|1\right\rangle_c \left\langle 1\right|) \tag{4.3}$$

$$\hat{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{4.4}$$

For the conditional shift operator, we use a suitable Unitary operator for allowing the walker to go one step forward if the accompanying coin state is one of the two basis states (e.g. $|0\rangle$) or one step backwards if the accompanying coin state is the other basis state (e.g. $|1\rangle$). The complete description of this linear operator is

$$\hat{S} = |0\rangle_c \langle 0| \otimes \sum_{n=-\infty}^{\infty} |n+1\rangle_p \langle n| + |1\rangle_c \langle 1| \otimes \sum_{n=-\infty}^{\infty} |n-1\rangle_p \langle n| \tag{4.5}$$

Altogether, the operator on the total Hilbert space is $\hat{U} = \hat{S} \cdot (\hat{C} \otimes \hat{\mathbb{I}}_p)$ and the mathematical representation of the discrete quantum walk after t steps is as in Eq. 4.2, where

$$|\psi\rangle_{initial} = |\text{position}\rangle_{initial} \otimes |\text{coin}\rangle_{initial} .$$

Let us suppose we have an initial state $|\psi\rangle_0 = |0\rangle_c \otimes |0\rangle_p$ with Eq. (4.3) and Eq. (4.5). The first step of the quantum walk, consisting of applying $\hat{H}$ to the coin state followed by the shift operator $\hat{S}$, is computed as follows

$$\begin{aligned}
|\psi\rangle_0 &= |0\rangle_c \otimes |0\rangle_p \\
|\psi\rangle_1 &= \hat{S}(\hat{H} \otimes \hat{\mathbb{I}})(|0\rangle_c \otimes |0\rangle_p) \\
&\xrightarrow{\hat{H}\otimes\hat{I}} \frac{|0\rangle_c + |1\rangle_c}{\sqrt{2}} \otimes |0\rangle_p \\
&\xrightarrow{\hat{S}} \frac{1}{\sqrt{2}}(|0\rangle_c \otimes |1\rangle_p + |1\rangle_c \otimes |-1\rangle_p)
\end{aligned}$$

Applying the same procedure, $|\psi\rangle_2$ would be

$$\begin{aligned}
|\psi\rangle_2 &= (\frac{1}{2}|0\rangle_c + 0|1\rangle_c)|2\rangle_p + (\frac{1}{2}|0\rangle_c + \frac{1}{2}|1\rangle_c)|0\rangle_p + (0|0\rangle_c - \frac{1}{2}|1\rangle_c)|-2\rangle_p \\
&= \frac{1}{2}(|0\rangle_c|2\rangle_p + (|0\rangle_c + |1\rangle_c)|0\rangle_p - |1\rangle_c|-2\rangle_p)
\end{aligned}$$

As we can see, after the first step, the position of the particle is a superposition of $n = 1$ and $n = -1$, generated by the coin operator. We say that the coin $\hat{H}$ is unbiased since the probability to go to the right is equal to the probability to go to the left. If we apply the step operator over and over again without intermediate measurement, the quantum correlations between different positions generate constructive or destructive interference, creating the behavior characteristic of quantum walks. In this case, the probability distribution is not the normal distribution and the standard deviation is $O(t)$ in contrast with the standard deviation of the unrestricted classical random walk on a line, which is $O(\sqrt{t})$ [47, 57].

Coined Quantum Walks on non-regular Graphs. Based on Ref. [47] and [48] we give a mathematical description that includes directed edges using non-regular graphs (or irregular graphs). The notation used until now is referred as the coin-position notation where the walker steps on vertices. Now, we introduce a useful notation for describing arbitrary graphs, the arc notation. If v and v' are adjacent, we make a distinction between labels, where (v, v') means from v to v' and label (v', v) means from v' to v. In this case, the physical interpretation of the walker dynamics is not a walking on vertices, instead the walker steps on arcs.

Let $\vec{G}(V, A)$ be a directed graph such that (v, v') and (v', v) are in $A(\vec{G})$. The Hilbert space associated with the coined walk on $\vec{G}$ is spanned by the arc set as follows

$$\mathcal{H}^{2|E|} = span\{|v, v'\rangle : (v, v') \in A(\vec{G})\} \tag{4.6}$$

where $2|E|$ is associated with two arcs of $\vec{G}$. The notation $|v, v'\rangle$ is called arc notation. The evolution operator of the coined quantum walk on $\vec{G}$ is

$$\hat{U} = \hat{S} \otimes \hat{C}$$

where $\hat{S}$ is the flip-flop shift operator defined by

$$\hat{S}\,|v, v'\rangle = |v', v\rangle$$

and $\hat{C}$ is the coin operator defined by

$$\hat{C} = \bigoplus_{v \in V} \hat{C}_v$$

where $\hat{C}_v$ is a d(v)-dimensional unitary matrix and $d(v)$ is the degree of v. To write $\hat{C}$ as a direct sum, we are decomposing $\mathcal{H}^{2|E|}$ as

$$\mathcal{H}^{2|E|} = \bigoplus_{v \in V} span\{|v, v'\rangle : (v, v') \in A(\vec{G})$$

For example, the coin operator for Fig. 2.1 (which is a non-regular graph) would be

$$\hat{C}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \hat{C} & 0 & 0 \\ 0 & 0 & \hat{H} & 0 \\ 0 & 0 & 0 & \hat{H} \end{bmatrix}$$

where $\hat{H}$ is the Hadamard gate and

$$\hat{C} = \frac{1}{3}\begin{bmatrix} -1 & 2 & 2 \\ 2 & -1 & 2 \\ 2 & 2 & -1 \end{bmatrix}$$

The shift operator is

$$\hat{S} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Is straightforward to check that

$$\hat{C}' = 2\sum_{j=0}^{3} |\alpha\rangle_j \langle\alpha|_j - I$$

$$\hat{S} = 2\sum_{j=0}^{3} |\beta\rangle_j \langle\beta|_j - I$$

A free scale network can be considered as a non-regular graph since we have different degree nodes. In addition to this discussion, in [37], Montanaro defines a condition in which a discrete time quantum walk can be defined on directed graphs. The procedure for such goal is the partition of graph G into subgraphs that meet the following condition, G_1 and G_2 are reversible subgraphs.

Definition 10 *An arc $a \to b$ is called reversible if there is a path from b to a. A graph whose arcs are all reversible is also reversible; otherwise it is called irreversible.*

Corollary 1 *A discrete-time quantum walk can be defined on a finite graph G if and only if G is reversible.*

According to [37], it is possible to define a partially quantum walk that maintains some quantum coherence in the reversible portions of the graph.

Lemma 1 *Let G^{rev} be the subgraph of G whose arcs consist of all the reversible arcs of G. Then G^{rev} is reversible.*

The following process is a possible way of defining a walk on irreversible graphs. First, we consider G integrated by the connected components of G^{rev} and patched together with irreversible arcs. A set of quantum walks is produced, each corresponding to one component of G^{rev}. Then, the irreversible arcs are simulated by replacing them with undirected edges. If such an edge is traversed by the walker, a different walk operator is used to ensure that it cannot be traversed in the opposite direction.

For example, consider a pair of vertices v_1 and v_2 that are in different reversible subgraphs of G (called G_1 and G_2), and consider an irreversible arc $v_1 \to v_2$. This arc can be simulated by the following two-steps process. First, perform a measurement to determine if the walker is in C_1 or C_2. Then, if it is in C_1, we perform one step of a quantum walk defined on the graph consisting of C_1 augmented with an undirected edge $v_1 \leftrightarrow v_2$. Alternatively, if the walker is in C_2, we perform one step of a walk only defined on the graph C_2. This ensures that the irreversible arc cannot be traversed in the wrong direction.

4.1.2 Szegedy Quantum Walk

Szegedy's quantum walk takes place on a symmetric bipartite digraph. A symmetric bipartite digraph is obtained by a duplication process of the underlying digraph (just as in Fig. 2.5). The underlying digraph defines a classical Markov chain and the Szegedy's model is considered as the quantized version of this Markov chain [47].

To describe the duplication process, consider a bipartite graph with identical sets X and Y of equal cardinalities. Let x and y be vertices of X and Y, respectively. The edges $\{x_i, x_j\}$ of the underlying graph, connects the adjacent vertices x_i and x_j, corresponds to two edges $\{x_i, y_j\}$ and $\{y_i, x_j\}$ in the bipartite graph.
Consider a connected bipartite graph $\Gamma(X, Y, E)$, as in Fig. 2.5. The biadjacency matrix of Γ (see equation 2.4) is

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$$

where A allow us to define transition matrix P as a probabilistic mapping from X to Y with entries p_{xy}. Matrix A^T, similarly, defines Q as a probabilistic mapping from Y to X with entries q_{yx}.

Consider the transition matrix P (or stochastic matrix) of the Markov chain with entries p_{ij} for $i, j \in \mathcal{S}$ and define, for the bipartite graph, the transitions p_{xy} and q_{yx} as the inverse of the degree of vertex x and y respectively if edges x and y are adjacent, otherwise $p_{xy} = q_{yx} = 0$. The entries p_{xy} and q_{yx} satisfy:

$$\sum_{y \in Y} p_{xy} = 1 \;\; \forall x \in X,$$
$$\sum_{x \in X} q_{yx} = 1 \;\; \forall y \in Y.$$

Note that $p_{xy} = q_{yx}$ and are symmetric since the bipartite graph is undirected and there is an correspondence between X and Y.

The quantum walk on the bipartite graph has an associated Hilbert space $\mathcal{H}^{mn} = \mathcal{H}^m \otimes \mathcal{H}^n$ where $m = |X|$ and $n = |Y|$. The computational basis of which is $\{|x, y\rangle : x \in X, y \in Y\}$. Instead of using probability matrices P and Q of the classical random walk, which entries are p_{xy} and q_{yx}, we define operators $A : \mathcal{H}^n \to \mathcal{H}^{n^2}$ and $B : \mathcal{H}^n \to \mathcal{H}^{n^2}$, known as half projectors, as follows

$$A = \sum_{x \in X} |\phi_x\rangle \langle x| , \tag{4.7}$$
$$B = \sum_{y \in Y} |\psi_y\rangle \langle y| , \tag{4.8}$$

The quantization method (P, Q) proposed by Szegedy starts by creating two operators on the Hilbert space with basis states $|x\rangle$, $|y\rangle$, where $x \in X$ and $y \in Y$, are defined as transition vectors as follows

$$|\phi_x\rangle = \sum_{y \in Y} \sqrt{p_{xy}} \, |x\rangle \, |y\rangle \tag{4.9}$$
$$|\psi_y\rangle = \sum_{x \in X} \sqrt{q_{yx}} \, |x\rangle \, |y\rangle \tag{4.10}$$

These transition vectors are defined on a vector space $H_A \otimes H_B$ with basis states $\{|x\rangle\}$ and $\{|y\rangle\}$ correspond to H_A and H_B respectively [59]. Another way to express Eq. (4.7) and Eq. (4.8) is multiplying the xth column of matrix A by the xth vector of the computational basis. Hence, the columns of matrix A correspond to vectors $|\phi_x\rangle$ and the columns of matrix B are the vectors $|\psi_y\rangle$.

Then we have

$$A^T A = I_n$$
$$B^T B = I_n$$

So, if $|\phi\rangle$ is a unit vector in $\mathcal{H}^n$, then $A|\phi\rangle$ is a unit vector in $\mathcal{H}^{n^2}$. The equations imply that the actions of A and B preserve the norm of vectors.

The product in reverse order is:

$$AA^T = \sum_{x \in X} |\phi_x\rangle \langle\phi_x| \tag{4.11}$$

$$BB^T = \sum_{y \in Y} |\psi_y\rangle \langle\psi_y| \tag{4.12}$$

So, we define the projectors

$$\Pi_A = AA^T \tag{4.13}$$

$$\Pi_B = BB^T \tag{4.14}$$

Eqs. (4.11, 4.12) shows that Π_A projects a vector in $\mathcal{H}^{n^2}$ on subspace $\mathcal{A} = span\{|\psi_x\rangle : x \in X\}$ and Π_B projects on subspace $\mathcal{B} = span\{|\psi_y\rangle : y \in Y\}$.

Definition 11 *An involution, or a reflection is an operation that is its own inverse. An operator is an involution if $A^2 = \mathbf{1}$. Involutions are operators of the form*

$$R = 2\mathbf{P} - \mathbf{1} \tag{4.15}$$

where $\mathbf{P}$ is some projector $(P^2 = \mathbf{1})$.

After obtaining the projectors, we can define the associated reflection operators, which are

$$\mathcal{R}_A = 2\Pi_A - I_{n^2}$$
$$\mathcal{R}_B = 2\Pi_B - I_{n^2}$$

This means that $\mathcal{R}_A$ leaves invariant any vector in $\mathcal{A}$, that is, if $|\psi\rangle \in \mathcal{A}$, then $\mathcal{R}_A|\psi\rangle = |\psi\rangle$. On the other hand, $\mathcal{R}_A$ inverts the sign of any vector orthogonal to $\mathcal{A}$that is, if $|\psi\rangle \in \mathcal{A}^\perp$, then $\mathcal{R}_A|\psi\rangle = -|\psi\rangle$. Geometrically, this is a reflection through $\mathcal{A}$, as if $\mathcal{A}$ is the mirror and $\mathcal{R}_A|\psi\rangle$ is the image of $|\psi\rangle$. The same is true for $\mathcal{R}_B$ with respect to subspace $\mathcal{B}$.

Performing a product of reflection provides the evolution operator

$$\begin{aligned} W &= (2AA^\dagger - \mathbf{1})(2BB^\dagger - \mathbf{1}) \\ &= (2\Pi_A - \mathbf{1})(2\Pi_B - \mathbf{1}) \\ &= \mathcal{R}_A\mathcal{R}_B \end{aligned}$$

A major advantage in using Szegedy's formalism over discrete and continuous time quantum walks lies in its ability to define a unitary quantum walk on directed and weighted graphs [35]. This fact will allow us to define a quantum walk on a directed network in which the quantum PageRank takes place.

For example, for the bipartite double cover in Fig. 2.5, let's consider the example provided by [62]. Let's suppose that the amplitude of edge $|2,1\rangle$ is $c_{2,1}$, the amplitude of edge $|2,3\rangle$ is $c_{2,3}$ and the amplitude of edge $|2,4\rangle$ is $c_{2,4}$. They are all incident to vertex $2 \in X$, and their average is

$$\bar{c}_2 = \frac{c_{2,1} + c_{2,3} + c_{2,4}}{3}$$

When operator $\mathcal{R}_A$ is applied, each of the three amplitudes are inverted about this mean, therefore

$$\begin{aligned} c_{2,1} &\to 2\bar{c}_2 - c_{2,1} \\ c_{2,3} &\to 2\bar{c}_2 - c_{2,3} \\ c_{2,4} &\to 2\bar{c}_2 - c_{2,4} \end{aligned}$$

the $\mathcal{R}_A$ operator does the same at each vertex in X, and $\mathcal{R}_B$ applies the same operation to vertices in Y.

4.2 Equivalence of discrete time quantum walk and Szegedy's quantum walk

Given the relevance in our study of discrete time quantum walks, in [62] and [48] Wong and Portugal, provide details on the equivalence of the coined quantum walks and Szegedy quantum walk.

In [62], Wong provides a proof for the fact that one step of Szegedy's quantum walk is equivalent to two steps of the coined quantum walk. A precise relationship between the equivalence of individual operators is provided. It shows that Szegedy's first reflection operator, is equal to the Grover diffusion coin flip of the coined quantum walk, while Szegedy's second reflection operator is equal to the "flip-flop" shift, which causes the particle to hop and turn around as $\hat{S}\,|a,b\rangle = |b,a\rangle$.

In the coined quantum walk, the walk is performed on the vertices of the original graph, using an additional coin degree of freedom. Thus, the number of directions in which a walker at the vertex v can point is $\deg(v)$, so the Hilbert space is $\mathbb{C}^{\sum_v \deg(v)}$. A particle that is at vertex a and points towards vertex b the state is written as $|a,b\rangle$. Hence, the computational basis is

$$\{|a,b\rangle : a,b \in \{1,\ldots,N\}, a \sim b\}$$

where $x \sim y$ denotes that vertices x and y are adjacent.

The evolution operator of the coined quantum walk is defined by repeated applications of $\hat{U} = \hat{S} \otimes \hat{C}$, just as Eq. 4.1. Using the Grover's diffusion coin which is a permutation-symmetric operator defined as

Definition 12 *Let $\mathcal{H}$ be an n-dimensional Hilbert space and $|i\rangle$ be the canonical basis for $\mathcal{H}$ and $|s_a\rangle = \frac{1}{\sqrt{2}} \sum_{i=0}^{n-1} |i\rangle$. Then the Grover operator is*

$$\hat{G} = 2 \sum_{a=1}^{N} |s_a\rangle \langle s_a| - I, \tag{4.16}$$

where

$$|s_a\rangle = \frac{1}{\sqrt{\deg(a)}} \sum_{b \sim a} |a, b\rangle \tag{4.17}$$

is the state of a particle at vertex a uniformly pointing towards each of its neighbors. Then, for each vertex a, $\hat{G}$ reflects the internal coin state across the equal superposition $|s_a\rangle$. In relation with the inversion about the mean of Grover's algorithm, for each vertex a, $\hat{G}$ inverts the amplitude of each coin state at a about the average amplitude of coin states at a.

For the shift $\hat{S}$, consider the flip-flop shift, which causes the particle to hop and then turn around. For example, a particle at vertex a pointing to vertex b jumps to vertex b and points at vertex a, so $\hat{S}|a, b\rangle = |b, a\rangle$.

The proof given by Wong states that Szegedy's quantum walk is equivalent to the coined quantum walk, with two applications of the coined operator equal to one application of Szegedy's, i.e. $W = U^2$, includes precise relationships between the individual operators of the walks. To begin with, the Hilbert spaces of the two quantum walks are identical:

- The Szegedy's quantum walk evolves in $\mathbb{C}^{2|E|}$ and the coined quantum walk evolves in $\mathbb{C}^{\sum_v \deg(v)}$, since $\sum_v = deg(v) = 2|E|$, we have $\mathbb{C}^{2|E|} = \mathbb{C}^{\sum_v \deg(v)}$.

Szegedy's walker on the edge connecting vertices $i \in X$ with $j \in Y$ is equivalent to a coined particle at vertex i pointing towards vertex j. In both walks, there is a bijection between basis states, we observe the same basis vector $|i, j\rangle$, denoting the same quantum state. This bijection allows us to reinterpret Szegedy's walk $W = R_2 R_1$ from its original form on the edges of the bipartite graph. Let's consider first R_1. In the bipartite graph, R_1 goes through each vertex in X and inverts its edges about their average at the vertex. In the coined quantum walk, R_1 goes through each vertex in the directed graph, inverting its outgoing amplitudes about their average at the vertex [62]. These outgoing amplitudes are the coin states at each vertex, that are precisely the Grover diffusion coin $\hat{G}$, so

$$R_1 = \hat{G} \tag{4.18}$$

Now, let's consider R_2. In the bipartite graph, R_2 goes through each vertex in Y and inverts its edges about their average at the vertex. In the coined quantum walk, the interpretation of R_2 is going through each vertex of the directed graph and inverts its incoming amplitudes about their average at the vertex. This is equivalent to the flip-flop shift followed by the Grover coin and another flip-flop shift, i.e.,

$$R_2 = \hat{S}\hat{G}\hat{S} \tag{4.19}$$

Combining the results of R_1 and R_2 we have

$$W = R_2 R_1 = \hat{S}\hat{G}\hat{S}\hat{G} = \hat{U}^2 \tag{4.20}$$

thus proving that two applications of the coined quantum walk are exactly equivalent to one application of Szegedy's quantum walk.

In [48], Portugal goes further, establishing that the coined quantum walk model share a large class of quantum walks instances. Also, the shift operator must be Hermitian and the coin operator must be an orthogonal reflection. The class of orthogonal reflections includes the Grover and the Hadamard coins. It concludes that there are Szegedy's quantum walks which cannot be converted into the coined formalism using the standard and non-regular flip-flop coined quantum walks.

Chapter 5

A concise introduction to QISKIT

The IBM Q Experience is an online platform that gives users access to a set of IBM's prototype quantum processors via the cloud. Users are allowed to interact with a quantum processor through the quantum circuit model of computation, which requires applying a set of quantum gates on qubits using a GUI called the quantum composer, writing quantum assembly language code [19] or through Qiskit [4].

The IBM QX architecture is integrated by IBM QX2 composed of 5 qubits, IBM QX3 composed of 16 physical qubits, and a revised versions of this 5-qubit and 16- qubit backends named IBM QX4 and IBM QX5 respectively. When executing quantum circuits or algorithms on these architectures, coupling restrictions have to be satisfied. The user first has to decompose all non-elementary quantum operations (Toffoli gate, SWAP gate, or Fredkin gate) to the elementary operations $U(\theta, \phi, \gamma)$ and CNOT. Moreover, two qubit gates cannot be arbitrarily placed in the architecture but are restricted to prescribed pairs of qubits only according to the architecture [60].

For these reasons, in this work we focus on Qiskit for classical simulation. Qiskit is a Python-based open-source software development kit (SDK) used to create algorithms for a quantum computer. It allows us to work with noisy intermediate-scale quantum computers (NISQ) [49] at the level of pulses, circuits, and algorithms [4].

The use of IBM's platform has proved useful to scientists and researchers for the development of quantum algorithms in different areas. In [13], Cervera explores the use of IBM's platform, in quantum physics, for the exact simulation of a one dimensional transverse Ising spin chain. Also, in [26] Xizuan *et al* make use of IBM's Qiskit quantum simulator and a quantum processor of five qubits to demonstrate a general quantum algorithm that evolves in open quantum dynamics. Recently, researchers have used quantum computing to advance the area of neural networks [55]. In [64], Zoufal *et al* employ a qGAN (quantum Generative Adversarial Network) distribution learning and loading method with

Qiskit and test it using a quantum simulation as well as actual quantum processors provided by the IBM Q Experience platform. Furthermore, the authors employ quantum simulation to demonstrate the use of the trained quantum channel in a quantum finance application. In addition to finance applications, Ref. [52], uses a 20-qubit quantum processor (IBM Q Tokio) to price options and portfolios of options on a gate-based quantum computer using amplitude estimation, an algorithm which provides a quadratic speedup compared to classical Monte Carlo methods. Similarly, a quantum algorithm that analyzes financial risk is presented in [61]. It employs quantum amplitude estimation to price securities and evaluate risk measures such as 'Value at Risk' and 'Conditional Value at Risk' on a gate-based quantum computer. Another example is the use of quantum simulation in chemistry, as in references [30, 23].

Beyond the extensive documentation available, some useful resources are for introduction to quantum algorithms using Qiskit [18, 29, 60] and for an introduction to quantum computing [41, 39].

5.1 Programming quantum circuits using Qiskit

Qiskit is made up of elements that work together to enable quantum computing. The tool is packed in four libraries named after the four classical elements, terra, aqua, aer, and ignis.

- Terra. Terra provides a bedrock for composing quantum programs at the level of circuits and pulses, to optimize them for the constraints of a particular device, and to manage the execution of batches of experiments on remote-access devices. Terra defines the interfaces for a desirable end-user experience, as well as the efficient handling of layers of optimization, pulse scheduling and backend communication.

- Aer. To really speed up development of quantum computers we need better simulators, emulators and debuggers. Aer helps us understand the limits of classical processors by demonstrating to what extent they can mimic quantum computation. Furthermore, we can use Aer to verify that current and near-future quantum computers function correctly. This can be done by stretching the limits of simulation, and by simulating the effects of realistic noise on the computation.

- Ignis. Ignis is dedicated to fighting noise and errors and to forging a new path. This includes better characterization of errors, improving gates, and computing in the presence of noise. Ignis is meant for those who want to design quantum error correction codes, or who wish to study ways to characterize errors through methods such as tomography, or even to find a better way for using gates by exploring dynamical decoupling and optimal control.

- Aqua. To make quantum computing live up to its expectations, we need to find real-world applications. Aqua is where algorithms for NISQ computers are built. These algorithms can be used to build applications for quantum computing. Aqua is accessible to domain experts in chemistry, optimization, finance and AI, who want to explore the benefits of using quantum computers as accelerators for specific computational tasks, without needing to worry about how to translate the problem into the language of quantum machines.

The workflow of constructing a quantum algorithm in Qiskit consists in three high-level steps: build, execute and analyze. We build the circuit using quantum gates, then we must specify on what backend we want to run them and finally a measurement gives the results.

Our available options for simulation are the backends handled by Aer. Qiskit Aer is a high-performance simulator for Qiskit Terra that provides a highly configurable noise model for studying quantum computing in the NISQ regime. Aer includes the following simulator backends:

- The QASM Simulator is the main Qiskit Aer backend. This backend emulates ideal and noisy multi-shot execution of a quantum circuit on a real device an returns measurement counts. It executes a Qiskit `QuantumCircuit` and returns a count dictionary containing the final values of any classical registers in the circuit. The circuit may contain gates measure, reset, conditionals, and other advanced simulator options.

- The State Vector Simulator. The `StatevectorSimulator` simulates the ideal single-shot execution of a quantum circuit and returns the final quantum state vector of the device at the end of simulation.

- The Unitary simulator allows ideal single-shot simulation of the final unitary matrix implemented by an ideal quantum circuit.

These backends are found in the Aer provider with the names `qasm_simulator`, `statevector_simulator` and `unitary_simulator` respectively.

As an example, let us consider the following code that generates a simple quantum system $|\Psi\rangle = |011\rangle$

```
from qiskit import QuantumRegister, QuantumCircuit, Aer, execute

qr = QuantumRegister(1)
qc = QuantumCircuit(q)

qc.iden(qr[0])
qc.x(qr[1])
qc.x(qr[2])

job = execute(qc, 'statevector_simulator)
result = job.result()
result.get_statevector

qc.draw(output='mpl')

qc1.measure(qr1, c1)
oq.Wavefunction(qc, systems=[3], column= True, show_systems=[True])

plot_histogram(result)
```

This is a quantum system of three qubits, in state $|011\rangle$. We begin by importing the main classes needed for the construction of our circuit, described in the following lines.

- `QuantumCircuit`: is a class considered as the set of instructions of the quantum system. When designing larger and more complex algorithms, we store operations into `QuantumCircuits`, which can be called upon by simulators to run them later.

- `QuantumRegister`: this class supports our qubits. We can have a certain number of qubits and perform gate operations specifying the qubit with a Quantum Register index location.

- `execute`- this is a function that runs our quantum algorithm. The number of times the circuit is run can be specified via the `shots` argument of the `execute` method, the default number of shots is 1024.

- **Aer** this is a class that handles classical simulator backends. Qiskit Aer is a high performance simulator framework for quantum circuits. It provides several backends to achieve different simulation goals.

Let us segment our code and explain its main components. To start with, consider the registers created.

```
qr = QuantumRegister(3)
c1 = ClassicalRegister(3)
qc = QuantumCircuit(qr, c1)

qc.iden(qr[0])
qc.x(qr[1])
qc.x(qr[2])
```

The first line is creating a `QuantumRegister` of three qubits, and is called '`qr`'. In the following line, we create a `ClassicalRegister` that will allow us to store the bits of the measurement results. Then, we create a `QuantumCircuit` called '`qc`' (quantum circuit) that takes as argument the registers created. Lastly, we apply three gates to the quantum registers. First, the identity operator is applied to the first qubit register using the function `iden` and we specify the application of this operator on index position `q[0]` (the first index entry is 0). Then, we apply an x gate on the second and third quantum registers respectively.

The instructions of our `QuantumCircuit` go through two more classes before finally coming out as a printable wavefunction.

```
job = execute(qc, 'statevector_simulator')
result = job.result()
result.get_statevector()
```

The created `QuantumCircuit` is just a set of instructions. We have to define the backend on which we will run our quantum circuit. In this example we use the `statevector_simulator`. Using the quantum circuit, we create a `job` via `execute`, then a `result` is created from that job and finally the results are displayed via the function `get_statevector` defined in the `Result` class, which prints our wavefunction as an array. We obtain the following result

```
{'110': 1024}
```

We can visualize the operations we are performing in the registers as a circuit, using `qc.draw()` which results in Fig. 5.1. In this circuit, the qubits are put in order, with

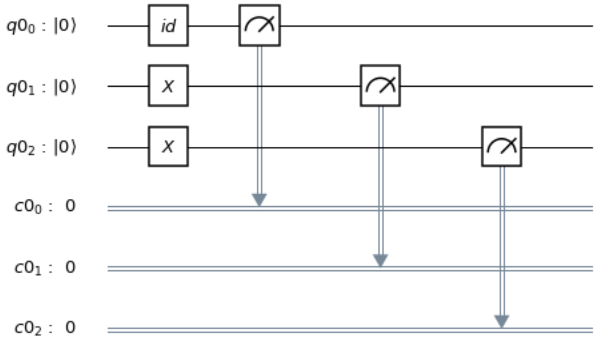


Figure 5.1: Quantum circuit for the $|011\rangle$ state.

qubit zero at the top and qubit two at the bottom. The circuit is read left to right (meaning that gates that are applied earlier in the circuit show up further to the left).

Now, we make use of a special function that will allow us to see our wavefunction in a ket notation and avoid dictionary-like printed results. For this reason, we import the function `Wavefunction` from `Our_Qiskit_Functions` provided by [29]. This function takes several parameters useful for better visual representation of the states in ket notation.

Finally, we are able to visualize the measurement results as a histogram as in Fig. 5.2.

```
plot_histogram(result)
```

The `plot_histogram` has some options to adjust the output graph [4]. We summarize the arguments it takes below.

- `legend`. It provides a label for the execution. It takes a list of strings use to label each execution's results. This is mostly useful when plotting multiple execution results in the same histogram.
- `sort`. This argument orders the bars in the histogram. It can be set to either ascending order with `asc` or descending order with `desc`.
- `color`. To adjust the color of the bars, `color` either takes a string or a list of strings for the colors to use for the bars for each execution.
- `figsize`. It takes a tuple of the size in inches to modify the size of the output figure.

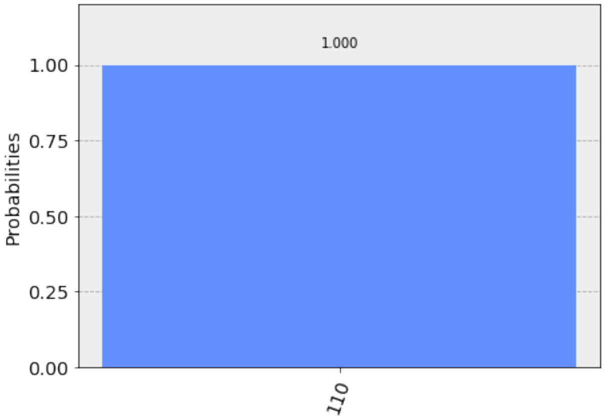


Figure 5.2: Histogram

The following lines summarise a basic template for creating a quantum algorithm on Qiskit:

1. Define how many qubits are needed
2. store them in a `QuantumRegister`
3. create a `QuantumCircuit` using all of the qubits in the quantum register
4. apply gate operations, measurements, barriers.

5.2 Quantum Gates for Quantum circuit design

Most quantum algorithms are best studied in the language of quantum circuits. The main quantum algorithms are presented in terms of quantum gates that evolve the quantum system. This model provides a discrete set of components which describe computational procedures and enable us to quantify the cost of an algorithm in terms of the total number of gates and the circuit depth. Nielsen and Chuang [41] provide an introduction to the quantum circuit model in which we implement our Szegedy quantum walk.

This section provides a review of the main quantum gates available in Qiskit, some of which represent the main operations applied to our circuit model. It starts with the description of single-qubit gates, Paulli gates, rotation gates, and controlled operations. Finally, we discuss the implementation of higher order control gates that are crucial to the quantum PageRank algorithm.

Single-Qubit Gates

Qiskit provides a set of single-qubit gates: u gates, identity gate, Pauli gates, Clifford gates, $C3$ gates, and standard rotation gates. We present these operators as matrix representations.

Quantum gates/operations are normally represented as matrices. A gate which acts on a single qubit is represented by a 2×2 unitary matrix U. In these terms, the action of a quantum gate is that of a matrix multiplying a vector representing the quantum state, that is:

$$|\psi'\rangle = U\,|\psi\rangle$$

where $|\psi\rangle = \begin{bmatrix}\alpha \\ \beta\end{bmatrix}$, is the column vector equal to $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$. Since global phase is undetectable, that is $|\psi\rangle = e^{i\varphi}\,|\psi\rangle$, we only require two real numbers to describe a single qubit quantum state.

Phase. If $re^{i\varphi}$ is a complex number, $e^{i\varphi}$, is called phase.

Global Phase. Consider the following states:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \qquad e^{i\varphi}\,|\psi\rangle = e^{i\varphi}\,|0\rangle + e^{i\varphi}\,|1\rangle$$

both states are indistinguishable since $|\alpha|^2 = |e^{i\varphi}\alpha|^2$, and $|\beta|^2 = |e^{i\varphi}\alpha|^2$, so it makes no difference during measurement.

A convenient representation of a qubit state $|\psi\rangle$ is:

$$|\psi\rangle = cos(\frac{\theta}{2})\,|0\rangle + e^{i\varphi}sin(\frac{\theta}{2})\,|1\rangle$$

for some angles $\theta \in [0, \pi]$ and $\varphi \in [0, 2\pi]$. From this, there is a clear one-to-one correspondence between qubit states and points on a unit sphere. From, this fact we define the general unitaries to perform any calculation in Qiskit.

u gates

A general unitary is implemented in Qiskit using the $u3$ gate is defined as

$$U(\theta, \phi, \lambda) = u3(\theta, \phi, \lambda)$$

where $U(\theta, \phi, \lambda)$ is

$$U(\theta, \phi, \lambda) = \begin{bmatrix} cos(\theta/2) & -e^{i\lambda}sin(\theta/2) \\ e^{i\phi}sin(\theta/2) & e^{i\lambda+i\phi}cos(\theta/2) \end{bmatrix}$$

This is the most general form of a single qubit unitary.

The $u2(\phi, \lambda) = u3(\pi/2, \phi, \lambda)$ has the matrix form

$$u2(\phi, \lambda) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\phi+\lambda)} \end{bmatrix}$$

The $u1(\lambda) = u3(0, 0, \lambda)$ gate has the matrix form

$$u1(\lambda) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix},$$

which is useful to apply a quantum phase.

The $u0(1) = u3(0, 0, 0)$ gate is the identity matrix. It has the matrix form

$$u0(1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Identity gate The identity gate is $id = u0(1)$

Pauli Gates

X: bit-flip gate The bit-flip gate X is defined as:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = u3(\pi, 0, \pi)$$

Y, bit- and phase-flip gate The Y gate is defined as

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = u3(\pi, \pi/2, \pi/2)$$

Z: phase-flip gate The phase flip gate Z is defined as:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = u1(\pi)$$

Clifford gates

Hadamard gate

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = u2(0, \pi)$$

Standard Rotations

The standard rotation gates are those that define rotations around the Pauli set $P = \{X, Y, Z\}$. They are defined as

$$R_P(\theta) = exp^{(-i\theta P/2)} = \cos(\theta/2)I - i\sin(\theta/2)P$$

Rotation around X-axis, $R_x(\theta)$. A rotation gate where the initial and final states can be represented as θ rotation around the x-axis on a Bloch sphere.

$$R_x(\theta) = \begin{bmatrix} cos(\frac{\theta}{2}) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix} = u3(\theta, -\pi/2, \pi/2)$$

Rotation around Y-axis, $R_y(\theta)$. A rotation gate where the initial and final states can be represented as θ rotation around the y-axis on a Bloch sphere.

$$R_y(\theta) = \begin{bmatrix} cos(\frac{\theta}{2}) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix} = u3(\theta, 0, 0)$$

Rotation around Z-axis $R_z(\theta)$. A rotation gate where the initial and final states can be represented as θ rotation around the z-axis on a Bloch sphere.

$$R_z(\theta) = \begin{bmatrix} e^{\frac{-i\theta}{2}} & 0 \\ 0 & e^{\frac{i\theta}{2}} \end{bmatrix} = u1(\phi)$$

Controlled operations on qubits

The standard multi-qubit gates consist of two qubit gates and three qubits gates. The two qubit gates are: controlled Pauli gates, controlled Hadamard gate, controlled rotation gates, controlled phase gate, controlled $u3$ gate and the swap gate. The three qubit gates are: Toffoli gate and Fredkin gate. In particular each gate uses a 'target qubit' and a 'control qubit'. The role of the control qubit is to determine whether or not a particular operation is applied to the target qubit.

Two-qubit gates

In general, a controlled two-qubit gate, C_U acts to apply the single-qubit unitary U to the second qubit when the state of the first qubit is in $|1\rangle$. Suppose U has a matrix representation

$$U = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix}$$

Controlled Pauli Gates

Controlled-X (or, CNOT) gate . The controlled-not gate flips the target qubit when the control qubit is in state $|1\rangle$.

Controlled Y gate This gate applies the Y gate to the target qubit if the control qubit is the most significant bit (MSB).

Controlled Z (or, controlled Phase) gate The controlled Z gate flips the phase of the target qubit if the control qubit is 1.

Controlled Hadamard gate

Apply H to the target qubit if the control qubit is $|1\rangle$.

$$C_H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

Controlled rotation around Z-axis

A rotation around Z-axis on the target qubit is performed if the control qubit is $|1\rangle$.

$$C_{R_z}(\lambda) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\lambda/2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\lambda/2} \end{bmatrix}$$

Controlled phase rotation Perform a phase rotation if both qubits are in the $|11\rangle$ state.

$$C_{u1}(\lambda) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\lambda} \end{bmatrix}$$

Controlled $u3$ rotation The controlled $u3$ rotation is performed by

$$C_{u3}(\theta,\phi,\lambda) \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i(\phi+\lambda)/2}\cos(\theta/2) & 0 & -e^{-i(\phi-\lambda)/2}\sin(\theta/2) \\ 0 & 0 & 1 & 0 \\ 0 & e^{i(\phi-\lambda)/2}\sin(\theta/2) & 0 & e^{i(\phi+\lambda)/2}\cos(\theta/2) \end{bmatrix}$$

SWAP gate. The SWAP gate causes two qubits to exchange states.

$$\begin{aligned} \textbf{SWAP}\,|00\rangle &\rightarrow |00\rangle \\ \textbf{SWAP}\,|01\rangle &\rightarrow |10\rangle \\ \textbf{SWAP}\,|10\rangle &\rightarrow |01\rangle \\ \textbf{SWAP}\,|11\rangle &\rightarrow |11\rangle \end{aligned}$$

The matrix notation of this operator is

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Controlled-X (or, controlled-NOT) gate. The controlled-not gate flips the target qubit when the control qubit is in the state $|1\rangle$.

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

5.2.1 Three-qubit gates

CCNOT (Toffoli Gate.) The Toffoli gate flips the third qubit if the first two qubits (LSB) are both $|1\rangle$:

$$|abc\rangle \rightarrow |bc \oplus a\rangle \otimes |b\rangle \otimes |c\rangle$$

$$\textbf{CCNOT}\,|010\rangle \to |010\rangle$$
$$\textbf{CCNOT}\,|101\rangle \to |101\rangle$$
$$\textbf{CCNOT}\,|110\rangle \to |111\rangle$$
$$\textbf{CCNOT}\,|111\rangle \to |110\rangle$$

In matrix notation

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The effect of this gate is equivalent to an X gate on the states $|110\rangle$ and $|111\rangle$.

5.3 Higher order control gates

As we have seen so far, Qiskit comes with a handful set of control gates. While the gates provided by Qiskit are sufficient as a universal set [29] it is helpful to define multiple-controlled gates. Qiskit does not provide an explicit implementation of higher order control gates. An example of a multiple qubit conditioning, is the Toffoli gate, where two qubits condition the application of the X gate. However, we require to implement higher order control gates. The strategy that we employ to construct higher order control gates is based in the use of *ancilla qubits*. In general, we define the controlled operation $C^n(U)$ by the equation

$$C^n(U)\,|x_1x_2\dots x_n\rangle\,|\psi\rangle = |x_1x_2\dots x_n\rangle\,U^{x_1x_2\dots x_n}\,|\psi\rangle$$

where U is a k qubit unitary operator and $x_1x_2\dots x_n$ in the exponent of U means the product of the bits $x_1, x_2, \dots, x_n$. That is, the operator U is applied to the last k qubit if the first n qubits are all equal to one, otherwise nothing is done. Such conditional operations are useful for our circuit, as illustrated in figure 5.3. Here, we present a special circuit notation for this task.

The implementation of $C^n(U)$ gates using our available gates in Qiskit involves the use of Toffoli gates. The circuit is divided into three stages, and make use of $(n-1)$ working qubits (ancilla qubits) as in Fig. 5.4.

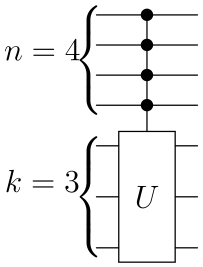


Figure 5.3: Circuit representation for the $C^n(U)$ operation, where U is a unitary operator on k qubits for $n = 4$ and $k = 3$

In essence, the ancilla qubits will allow us to temporarily store information about the control qubits, and ultimately determine whether or not to apply the operation. First, we need all of the ancilla qubits to be initialized in the state $|0\rangle$. In order for our CCNOT gate to change the target ancilla in the state $|1\rangle$, (only when control qubits are in the $|1\rangle$ state), the target qubit must be initially in the $|0\rangle$ state. Then, U is applied if and only if all of c_1 through c_n are set to the corresponding controls. Finally, the last part of the circuit just reverses the steps of the first stage, returning all the work qubits to their initial state, $|0\rangle$. The combined result therefore, is to apply the unitary operator U to the target qubit, if and only if all the controls bits c_1 through c_n are set as desired. This procedure is illustrated in Fig. 5.4.

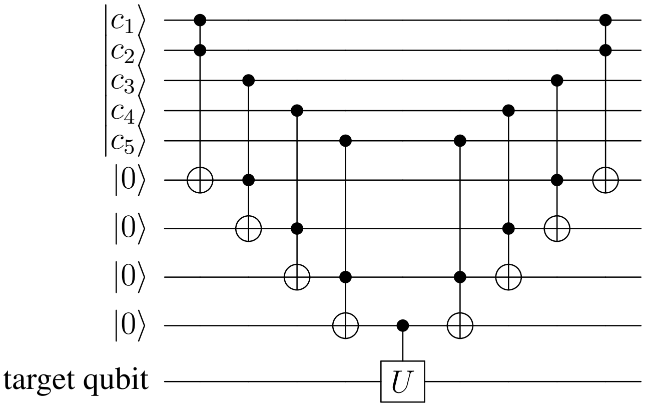


Figure 5.4: Network implementing the $C^n(U)$ operation for $n = 5$

As we will see in section 6.2.1 we require gates controlled by three qubits, hence we will need two ancilla or auxiliary qubits to implement our circuit.

5.4 Circuit implementation of Quantum Walks

We want to express a given quantum walk in terms of a quantum circuit in order to avoid the lack of scalability of the quantum walk model, i.e., as the size of the Hilbert space scales exponentially, the number of physical vertices required for the quantum walk also scales exponentially [33]. Douglas and Wang [20] show that quantum walks can be clasically implemented in a time that scales polynomially with the size of the state space. Therefore, an efficient quantum implementation of quantum walks is considered to be one in which the required resources scales logarithmically with the size of the state space [20].

In designing quantum circuits for quantum walks, it is possible to have many valid quantum circuits that would implement the required quantum walk on a given graph [35]. A general principle for efficient quantum circuit implementation of quantum walks, stated by Douglas and Wang [20], affirm that for the possibility of exponential speedups, the number of elementary gates required to perform the walk need to grow logarithmically with the size of the state space [20]. In addition, an efficient quantum circuit is possible for those graphs with a high degree of symmetry [20]. Hence, the only way to obtain a symmetric graph from a directed graph is to consider the double bipartite graph as in Eq. 2.1, used in the Szegedy formalism.

We provide an example of a quantum circuit implementing a quantum walk along an undirected 16-cycle presented in Fig. 5.5. The evolution operator in the coined model $\hat{U} = \hat{S} \otimes \hat{C}$ is implemented as follows. For the coin operator, we implement $\hat{C}$ using a Hadamard gate acting on a qubit as a subnode that controls the shifting operators. For the shift operator $\hat{S}$, we perform a cyclic permutation of the node states as $S = Incr. \left|1\right\rangle + Decr. \left|0\right\rangle$. This permutation is achieved via 'increment' and 'decrement' gates made up of generalized CNOT gates shown in Fig. 5.6 and 5.7. Later in Sec. 7, we will use these gates as left and right permutations respectively.

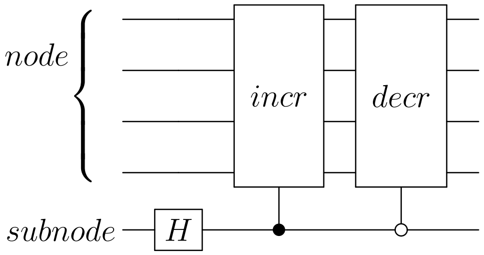


Figure 5.5: Quantum circuit implementing a quantum walk along a 16-length cycle.

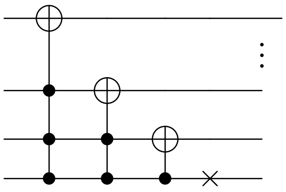


Figure 5.6: Increment on n qubits

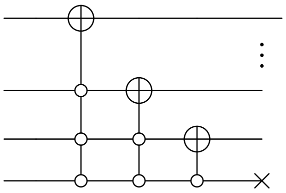


Figure 5.7: Decrement

5.4.1 Szegedy Circuit

The Szegedy quantum walk runs repeating t times the circuit with the operator $\hat{U}(t)$. We can establish a partial equivalence between the step operator for the Szegedy quantum walk and the coined quantum walk as explained by Wong [62] and Portugal [48]. For the Szegedy walk, we base our implementation in the theoretical framework for the development of efficient quantum circuits provided by Loke and Wang in [35]. The corresponding quantum circuit that reproduces the unitary evolution operator of the quantum walk is presented in Eq. (5.1) .

$$U_{walk} = \mathcal{S}(I - 2\Pi) = \mathcal{S}\mathcal{R} \tag{5.1}$$

We reproduce the main ideas in the Szegedy implementation in the following lines.

- First, implementing the swap operator $\mathcal{S}$ in a quantum circuit is straightforward. It consists of swap gates applied between the two registers as in Fig. 5.8.
- In order to implement the reflection operator $\mathcal{R}$, a diagonalization procedure using a unitary operator $\hat{U}$ is carried out.

$$U\mathcal{R}U^{\dagger} = U(2\Pi - I)U^{\dagger} = 2U\Pi U^{\dagger} - I$$

The Szegedy walk operator results in

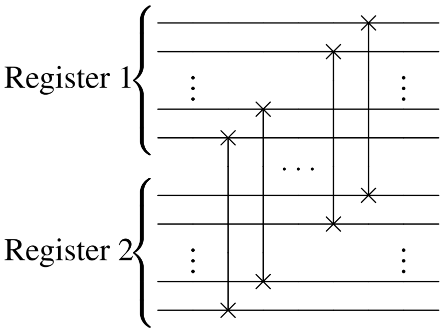


Figure 5.8: Quantum circuit implementation of the swap operator $\mathcal{S}$.

$$U_{walk} = \mathcal{S}U^{\dagger}DU.$$

Fig. 5.9 shows the general quantum circuit for implementing U_{walk}

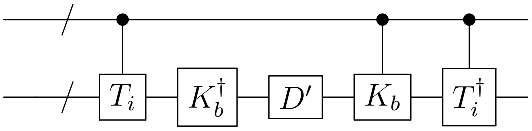


Figure 5.9: General Quantum circuit implementation of $\hat{U}_{walk}$

A complete circuit using this formalism is presented in Fig. 6.3.

5.5 Qiskit experience

In this section, we discuss the advantages and main drawbacks of the Qiskit platform. Qiskit is aimed for researchers, teachers, developers and enthusiasts [60]. The platform helps users to easily design and execute their own applications on real quantum computers and state-of -the-art digital simulators of quantum algorithms and protocols. The work presented in this thesis is developed using the Qiskit version 0.11.2. We provide an overview of different aspects involving advantages and disadvantages from two perspectives:

- the user's perspective
- the developer perspective

To start with, we know that quantum gates perform operations on qubits. These qubits can be represented by two basis states $|0\rangle$ or $|1\rangle$, but also in a superposition of both states. We employ circuit diagrams where qubits are represented by horizontal lines and gates pass through these lines. The diagram defines the order in which quantum gates are applied to qubits, from left to right. The IBM Q Experience allows to edit and write quantum algorithms through a graphical interface, and run them on actual IBM Q hardware. Using the IBM QX architectures it is possible to define elementary single operation $U(\theta, \psi, \lambda) = R_z(\phi)R_y(\theta)R_z(\lambda)$. The IBM QX architectures available publicly are: Yorktown(5 qubits), Vigo (5 qubits), Ourense (5 qubits), Essex (5 qubits), Burlington (5 qubits), London (5 qubits), Melbourne (14 qubits). In addition, the classical simulator, the qasm simulator, allows to simulate up to 32 qubits with no coupling restrictions.

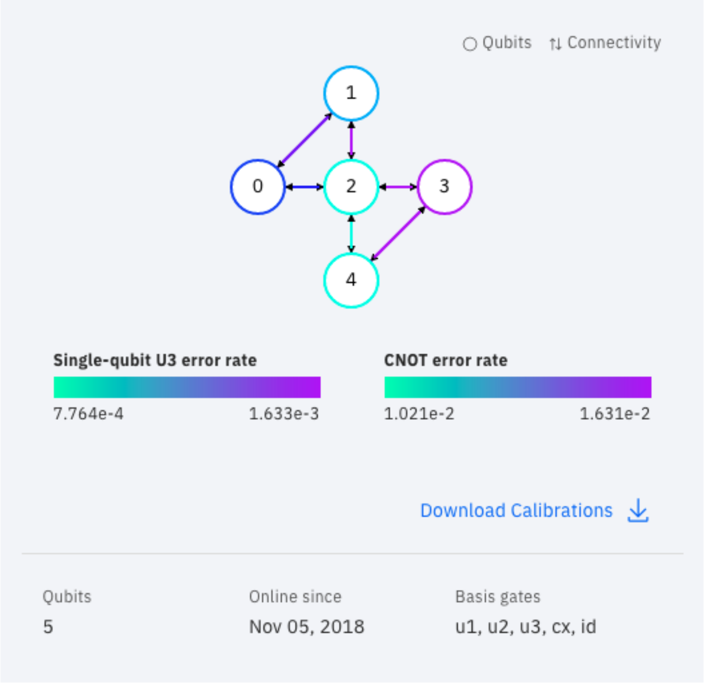
In order to execute a quantum algorithm on these architectures, we must satisfy coupling restrictions. First, all non-elementary quantum operations (Toffoli, Swap, Fredkin gate) are decomposed into the elementary operations $U(\theta, \phi, \lambda)$ and CNOT. In addition, gates operating on two qubits, like CNOT, are connected in a restricted manner given by the architecture. The restrictions are given by the so called *coupling-map* illustrated in Fig. 5.10 which lays out the architecture of the IBM QX2 Yorktown. Each node in the architecture represents a qubit and the arrows reflect the direction in which an operation can be performed. As an example, the CNOT, is defined from physical qubit Q_i to qubit Q_j, with control qubit Q_i and target qubit Q_j. This limitation is a major challenge in the design of any quantum algorithm on real quantum hardware.

To use the hardware backends, users are required to register on the web platform. This way, users obtain a token that will allow them to run quantum algorithms on real devices. Furthermore, users obtain credits which limit the number of times that the quantum processor can be used. In the following example we load the QX4 architecture.

```python
from qiskit import IBMQ
IBMQ.load_accounts()
ibmqx4 = IBMQ.get_backend('ibmqx4')
```

Qiskit covers the full interaction with the IBM Q hardware. In Sec. 5.1 we provide a review of the Qiskit toolset. Using qiskit, anyone can implement quantum operations depending of the level of complexity of the circuit. In general, the workflow in Qiskit is smooth as described at the beginning of this section. The graphics to visualize the circuits are functional as well as the graphics to visualize the results. As a result, users will find a very intuitive platform.

One of the major hurdles to overcome is the use of multiple control qubits described in Sec. 5.3. The strategy to implement multiple controls used in this thesis was the use



Qubits
5

Online since
Nov 05, 2018

Basis gates
u1, u2, u3, cx, id

Figure 5.10: Yorktown quantum processor.

of ancilla qubits. In this work, we reduce the use of CNOT and Toffoli gates by reusing the control sequence common to the controlled gates. This is a manual design, there is no automatic procedure in Qiskit to implement multiple controlled gates.

From the developer perspective, the efforts are in optimization techniques since the corresponding quantum states grow exponentially with respect to the number of qubits employed. In mathematical terms, a quantum circuit is a sequence of matrix-vector multiplication, using the initial state vectors, and the operations defined in the circuit. For example, applying a a CNOT operation to a two qubit system in state $|10\rangle$, results in:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \equiv |11\rangle$$

In order to optimize simulation of quantum circuits new approaches are being developed. Besides simulation, mapping a quantum circuit to a specific architecture constitute a problem. We can use Hadamard and SWAP gates to change the mapping of the logical qubits, however, using these gates to satisfy coupling restrictions increase the probability of errors.

We can take advantage of classical simulators to design circuits, and analyze the results. If the simulation results are promising, an execution on a real quantum device can be performed using the backends used for that purpose. There is a lot of room for improvement of the Qiskit platform. It is noticeable the fast pace of improvement and the rapid development of the platform. This advancement is thanks to contributions of the community. As an open source project, Qiskit uses the Github platform, hence making contributions easy. Once a contribution is done the Qiskit community review, test and analyze the contribution. A review of the main gate-based quantum software platforms is presented in [32].

Chapter 6

Classical and Quantum PageRank

Google's ranking system has had a tremendous influence on the development of the Internet. In short, PageRank's thesis states that a webpage is important if it is pointed to by other important pages. As we will see the PageRank importance scores are actually the stationary values of a Markov chain, and consequently Markov theory explains many interesting properties of the model used by Google [31]. Our goal of this section is to explain the core concepts of the PageRank algorithm. We base our discussion on references [11, 10, 42, 31].

6.1 Google Pagerank

As seen in section 2.2, the World Wide Web is an interconnected web of pages that follows the scale free network property. The basic idea behind the ranking of pages is a notion of page authority which is independent from page content.

Let us suppose the web of interest contains n pages, each page indexed by an integer k, $1 \leq k \leq n$. An example of a web composed of four pages is illustrated in Fig 6.1.

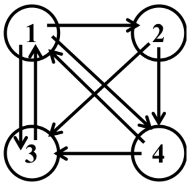


Figure 6.1: Representation of a web with four pages.

An arrow from A to B indicates a link from page A to B, producing a *directed graph* (defined in Section 2).

Definition 13 *Let $r_k(P_i)$ be the PageRank of page P_i at iteration k. Then the PageRank of page P_i, is the sum of the PageRank of all pages pointing into P_i*

$$r_{k+1}(P_i) = \sum_{P_j \in B_i} \frac{r_k(P_j)}{outdeg(P_j)} \tag{6.1}$$

where $outdeg(P_j)$ is the outdegree of page P_j and B_i is the set of pages backlinking or pointing into page P_i. A link from a page to itself is not counted.

Consider r_k as nonnegative and $r_k(P_j) > r_k(P_i)$ indicates that page j is more important than page k. According to Eq. 6.1 we have for Fig. 6.1:

- $P_1 = P_3/1 + P_4/2$ since pages 3 and 4 are backlinks for page 1, and page 3 contains only one link, while page 4 contains 2 links.
- $P_2 = P_1/3$ since page 1 points to page 2, and page 1 contains 3 outlinks to different pages.
- $P_3 = P_1/3 + P_2/2 + P_4/2$ since pages 1, 2 and 4 are backlinking for page 3, and page 1, 2 and 4 contain 3, 2 and 2 links respectively.
- $P_4 = P_1/3 + P_2/2$ since pages 1 and 2 are backlinking for page 4 and page 1 and 2 contain 3 and 2 links respectively.

These linear equations can be written as $\mathbf{Ax} = \mathbf{x}$, where $\mathbf{x} = [P_1 P_2 P_3 P_4]^T$. Henceforth, we treat the Pagerank problem as a linear algebra problem.

For a web of n pages, Eq. 6.1 produces a matrix $\mathbf{A}$ with $A_{i,j} = 1/n_j$ if page j links to page i, and $A_{i,j} = 0$ otherwise. The jth column of $\mathbf{A}$ then contains n_j nonzero entries, each equal to $1/n_j$, and the column thus sums to 1. We denote $\mathbf{A}$ as a transition matrix (see Eq. 2.2) for the web in Fig. 6.1

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

The process for calculating the PageRank is initiated with the ranking at the first iteration as $r_0(P_i) = 1/n$ for all pages P_i and repeated until convergence. Applying Eq. 6.1 to the tiny web of Fig. 6.1 gives the following values for the PageRank after few iterations shown in Table 6.1.

Iteration 0	Iteration 1	Iteration 2	Rank at iteration 3
$r_0(P_1) = 1/4$	$r_1(P_1) = 3/8$	$r_2(P_1) = 9/16$	1
$r_0(P_2) = 1/4$	$r_1(P_2) = 1/12$	$r_2(P_2) = 1/36$	4
$r_0(P_3) = 1/4$	$r_1(P_3) = 1/3$	$r_2(P_3) = 4/9$	2
$r_0(P_4) = 1/4$	$r_1(P_4) = 5/24$	$r_2(P_4) = 25/144$	3

Table 6.1: First iterations using Eq. 6.1 on Fig. 6.1

The web ranking problem is considered as the standard problem of finding an eigenvector for a square matrix. The eigenvalues λ and eigenvectors $\mathbf{x}$ of a matrix $\mathbf{A}$ satisfy the equation $\mathbf{Ax} = \lambda\mathbf{x}$, with $\mathbf{x} \neq \mathbf{0}$.

To explain the modifications to the transition matrix, Brin and Page use the notion of a *random surfer*. In this depiction, a web surfer navigates randomly the hyperlink structure of the web. In the long run, the amount of time spent by the web surfer on a given page is the measure of relative importance of that page. Unfortunately, the random surfer is caught in dangling nodes, i.e., nodes without outgoing links. A web with dangling nodes produces a matrix $\mathbf{A}$ which contains one or more columns of all zeros. To fix this problem Brin and Page make an adjustment called *stochastic adjustment* because the 0^T rows of the transition matrix are replaced with $1/n$, where n is the number of nodes. Thereby making $\mathbf{A}$ stochastic. Consequently, the random surfer can hyperlink to any page at random.

$$\mathbf{S} = \mathbf{A} + \mathbf{a}(1/n)$$

where $\mathbf{a} = 1$ if page i is a dangling node and 0 otherwise. $\mathbf{A}$ is the adjacency matrix.

This adjustment guarantees that $\mathbf{S}$ is stochastic. However, by itself it does not guarantee the convergence results desired. Brin and Page used another adjustment named *primitivity adjustment.* With this adjustment, the resulting matrix is stochastic and primitive. A primitive matrix is both irreducible and aperiodic (aperiodicity plus irreducibility implies primitivity). These conclusions motivate the following definitions, used in the study of Markov chains.

Definition 14 ***Stochastic matrix.*** *A square matrix $\mathbf{A}_{n\times n}$ is called a column stochastic matrix, if all of its entries are nonnegative and the entries in each column sum to* 1. *We make a distinction between row and column- stochastic.*

Definition 15 ***Primitivity matrix.*** *A matrix $\mathbf{A}$ is defined to be a primitive matrix when $\mathbf{A}$ is nonnegative irreducible matrix that has only one eigenvalue, $r = \rho(\mathbf{A})$ on its spectral circle.*

Definition 16 ***Irreducible matrix.*** *A square matrix* $\mathbf{A}$ *is irreducible if and only if its directed graph is strongly connected. In other words,* $\mathbf{A}$ *is irreducible if and only if for each pair of indices* (i, j) *there is a sequence of entries in* $\mathbf{A}$ *such that* $a_{ik_1}a_{k_1k_2}\dots a_{k_tj} \neq 0$.

Definition 17 *An aperiodic Markov chain is an irreducible chain whose transition probability matrix* $\mathbf{P}$ *is a primitive matrix.*

Therefore, the stationary vector of the chain (which is the PageRank vector in this case) exists, is unique, and can be found by a simple power iteration.

The theory of Markov chains is suitable for the PageRank problem. With Markov theory we make adjustments to the PageRank equation that allow convergence, i.e. desirable results.

In particular, we know that for any starting vector, the method (known as Power method) applied to a Markov matrix $\mathbf{A}$ converges to a unique positive vector called the *stationary vector*, as long as $\mathbf{A}$ is stochastic, irreducible and aperiodic [31]. Therefore, the PageRank convergence problems that arise from sinks and cycles, can be eliminated if $\mathcal{A}$ is modified so that the Markov matrix share the desired properties.

The argument for the primitivity adjustment is based on the fact that a random surfer can restart the browsing by entering a new destination in the browser's URL line. To model this activity mathematically, Brin and Page proposed a new matrix

$$\mathbf{G} = \alpha\mathbf{S} + (1-\alpha)\mathbf{E} \tag{6.2}$$

where $\mathbf{S}$ is the stochastic matrix, α is a scalar between 0 and 1. $\mathbf{G}$ is called the **Google matrix.** In this model, α is a parameter that controls the proportion of time the random surfer follows the hyperlink. Suppose $\alpha = 0.6$, then the random surfer follows the hyperlink structure of the web 60% of the time and the other 40% of the time the random surfer teleports to a new random page. This situation is represented as the teleporting matrix $\mathbf{E} = 1/n\mathbf{e}\mathbf{e}^T$.

Now, we make a special emphasis in the Markov chain analysis of the PageRank algorithm. Is important to note that the mathematical component of Google's Pagerank vector is the stationary distribution of a discrete-time, finite state Markov chain. We summarize the concepts involved in Markov chains and random walks:

- A *stochastic process* is a set of random variables $\{X_t\}_{t=0}^{\infty}$ having a set of state space $\{S_1, S_2, \dots, S_n\}$. X_t represents the state of the process at time t. The process of surfing the web is represented by a state space corresponding to the set of all web pages and the random variable X_t is the web page being viewed at time t. We consider the Markov chain process as discrete-time.

- A *Markov chain* is a stochastic process that satisfies the Markov property $P(X_{t+1} = S_j)|X_t = S_{i_t}, X_{t-1}, \dots, X_0 = S_{i_0}) = P(X_{t+1} = S_j|X_t = S_{i_t})$ for each $t = 0, 1, 2, \dots$. The notation $P(E|F)$ denotes the conditional probability that event E occurs given event F occurs.
  - We can view the Markov property as memoryless in the sense that the state of the chain depends only on the current state and not on the past history of the chain. For instance, the web surfing is a Markov chain where the next page visited does not depend on the pages previously visited in the past- the choice depends only on the current page.
  - This kind of chain is referred to as a *random walk* on the link structure of the web.
- The *transition probability* $p_{ij} = P(X_t = S_j|X_{t-1} = S_i)$ is the probability of being in state S_j at time t given that the chain is in state S_i at time $t-1$. In few words, it is the probability of moving from S_i to S_j at time t.
- A *stationary Markov chain* is a chain in which transition probabilities do not change with time, i.e., $p_{ij}(t) = p_{ij}$ for all t.

6.2 Quantum PageRank Algorithm with Szegedy Walks

The quantum PageRank algorithm proposed in [43] and [44] is a quantization process of the Google PageRank algorithm. Using Szegedy's formalism (described in Sec. 4.1.2) of the discrete-time quantum walk, we are able to quantize (apply quantum theory) the Markov chain corresponding to a classical random walk described in the previous section. This section is based on [35, 43, 44, 34].

Classically, for an N-node graph, such a process is described by an order $N \times N$ matrix P of transition probabilities, where each entry P_{jk} denotes the transition probability from node k to node j.

The average probability of the wave function at certain node is considered as the measure of relative importance of that node.

Consider a directed graph (representing a network) described by a connectivity matrix C, where $C_{i,j} = 1$ if there is a link $j \to i$. Then the patched connectivity matrix S is defined as:

$$S_{i,j} = \begin{cases} 1/N & outdeg(j) = 0 \\ C_{i,j}/outdeg(j) & otherwise, \end{cases} \tag{6.3}$$

where $outdeg(j) = \sum_{i=1}^{N} C_{i,j}$ is the out-degree of vertex j. The Google matrix is:

$$G = \alpha S + \frac{1-\alpha}{N} J \tag{6.4}$$

where $\alpha \in [0,1]$ is the damping parameter (typically chosen to be 0.85 and J is the all-1s matrix. Taking the transition matrix as $P = G$, we can define the Szegedy walk operator U_{walk} given by Eq. (5.1). Then the instantaneous quantum PageRank of the jth vertex is given by Eq. (6.5)

$$Q(j,t) = |\, \langle j|_2 \, U_{walk}^{2t} \, |\psi_0\rangle^2 \,| \tag{6.5}$$

where $\langle j|_2 = (|j\rangle_2)^\dagger$ and $|j\rangle_2$ is the jth standard basis vector of the second Hilbert space $\mathcal{H}_2$.

The quantum walk is initialised as

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} |\psi_j\rangle \tag{6.6}$$

that is, the initial state $|\psi\rangle_0$ is taken to be an equal superposition over the $|\psi_j\rangle$. The average quantum PageRank for a vertex j, over some number of steps T, is defined as:

$$\langle Q(j) \rangle = \frac{1}{T} \sum_{t=0}^{T-1} Q(j,t) \tag{6.7}$$

It can be shown to converge for sufficiently large T. This quantity is called the quantum PageRank of a graph. The goal is to simulate the walk operator U_{walk} using the Google matrix G as the transition matrix.

While quantum walks on undirected graphs have been well studied, extending the framework presented in the quantum PageRank algorithm to include directed quantum walks is non-trivial due to the requirements of unitary and reversibility of the walk [34]. We point out some of the key ideas when extending the classical framework into the quantum formalism.

Is important to mention that unitarity of the quantum walk is maintained since G (Eq. 6.2) is stochastic [34], moreover information on the directionality of the network is preserved in G.

6.2.1 Szegedy circuit for the quantum PageRank algorithm

Examples of quantum circuits for Szegedy's quantum walks are presented in [15], [35], [33]. References [43], [44] use Szegedy's quantum walk as a quantization procedure of the classical Google PageRank algorithm, here we reproduce the quantum circuit by Thomas Loke for the quantum PageRank in Fig. 6.3.

An example of computing the PageRank on directed graphs is given by 6.2, the vertices of the graph can be partitioned into subsets of equivalent vertices as $Z = Z_1 \cup Z_2 \cup Z_3$ where $Z_1 = \{1, 2, 3, 4\}$, $Z_2 = \{4, 5\}$, $Z_3 = \{6, 7\}$

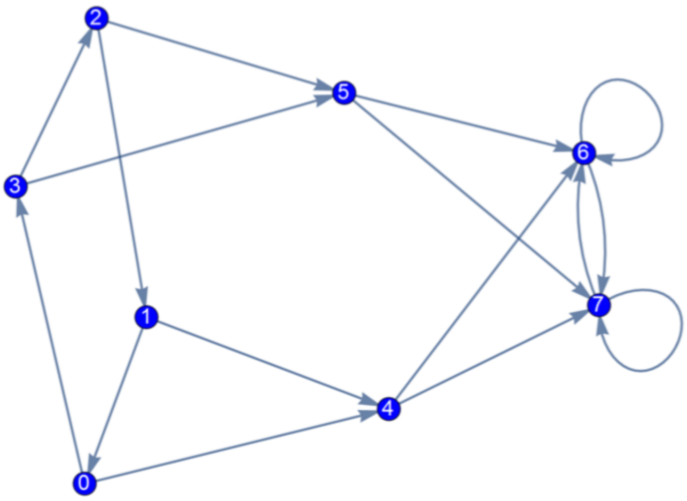


Figure 6.2: A directed graph

The connectivity matrix given by:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{6.8}$$

The process starts setting the basis states as $|b_1\rangle = |b_2\rangle = |b_3\rangle = |0\rangle$.

- For the set Z_1, we take a reference state as $|\phi_0\rangle$ written as $|\phi_0\rangle = \{\sqrt{\beta}, \sqrt{\gamma_1}, \sqrt{\beta}$,-$\sqrt{\beta}, \sqrt{\beta}, \sqrt{\beta}, \sqrt{\beta}, \sqrt{\beta}\}$ where $\beta = \frac{1-\alpha}{8}$ and $\gamma = \alpha + \beta$.
  The required transformations $T_{1,y}$ for $y \in Z_1$ that does $T_{1,y} : |\phi_y\rangle \to |\phi_0\rangle$ can be identified as $T_{1,y} = L^y$
- For the set Z_2, the reference state is selected as $|\phi_4\rangle$, which can be written as $|\phi_4\rangle = \{\sqrt{\gamma_2}, \sqrt{\gamma_2}, \sqrt{\beta}, \sqrt{\beta}, \sqrt{\beta}, \sqrt{\beta}, \sqrt{\beta}, \sqrt{\beta}, \sqrt{\beta}\}$ where $\gamma_2 = \frac{\alpha}{2} + \beta$. The required transformation $T_{2,y}$ for $y \in Z_2$ that does the analogous transformation is simply $T_{2,4} = I$ and $T_{2,5} = L^2$.
- For the set Z_3, the reference state is $|\phi_6\rangle$, which can be written as $|\phi_6\rangle = \{\sqrt{\beta}$,-$\sqrt{\beta}, \sqrt{\beta}, \sqrt{\beta}, \sqrt{\gamma_3}, \sqrt{\gamma_3}, \sqrt{\gamma_3}, \sqrt{\gamma_3}, \sqrt{\gamma_3}\}$, where $\gamma_3 = \frac{\alpha}{4} + \beta$. Since $|\phi_6\rangle = |\phi_7\rangle$ no transformation are required.

For circuit details see ref. [35]. The implemented circuit is show in fig. 6.3.

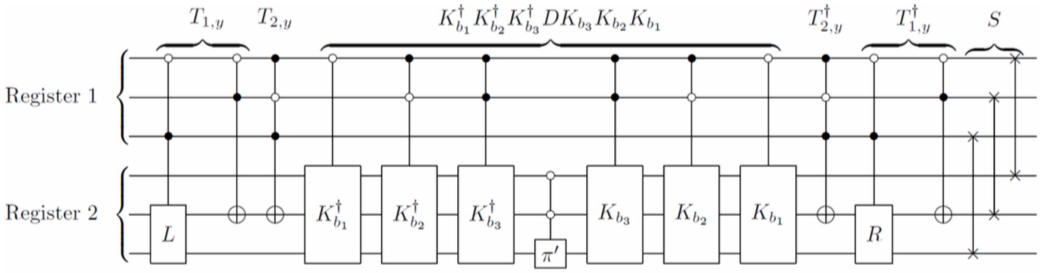


Figure 6.3: Complete circuit for U_{walk}

where L and R operators are defined as in figure 5.6 and 5.7; K_{b_1}, K_{b_2} and K_{b_3} are represented as in Fig. 6.4, 6.5 and 6.6, respectively.

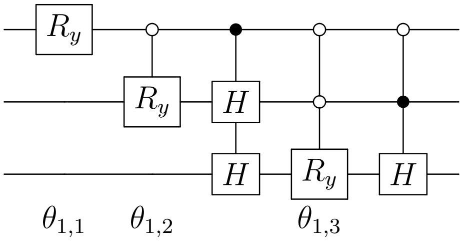


Figure 6.4: K_{b_1}

We implement the controlled gates, based on the discussion of multiple controlled gates in section 5.3. Our proposal consist in controlling individually each gate. We assume

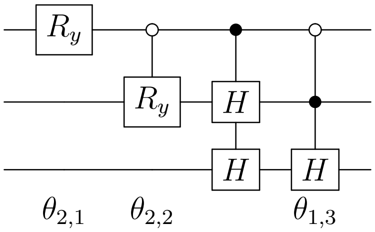


Figure 6.5: K_{b_2}

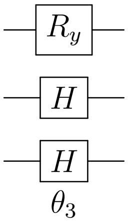


Figure 6.6: K_{b_3}

that controlling a composed-quantum gate is equivalent to controlling individually all the gates composing it. So, for the $T_{1,y}$ segment in Fig. 6.3, we propose an implementation as in Fig. 6.7, where the labels q0 and q1 correspond to register 1 and register 2 respectively and the label anc correspond to the ancilla qubits. In this case, we have an operation with 3 controls, so we require two ancilla qubits (see discussion in Sec. 5.3). In Qiskit we implement this operation as in Fig. 6.8.

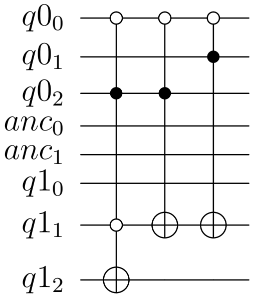


Figure 6.7: Controlled gate corresponding to $T_{1,y}$

The same procedure is applied to the rest of the controlled gates. In Appendix A, we provide the diagram circuit for each of the implemented gates.

In the next section we implement and simulate each of the operations required for the Szegedy quantum walk.

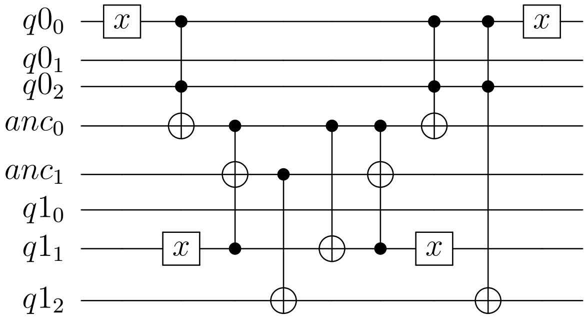


Figure 6.8: Qiskit implementation of controlled gate corresponding to $T_{1,y}$

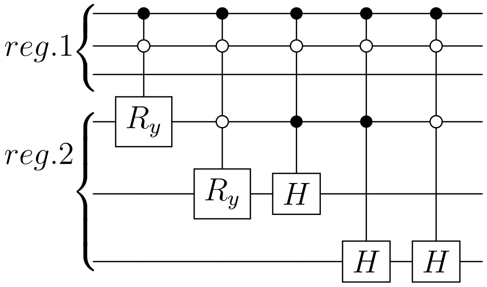


Figure 6.9: Circuit for K_{b_2}

Chapter 7

Simulations

In this section, we present our main results for the simulations of the quantum circuit presented in Fig. 6.3. We start by setting the layout where we program our quantum circuit. We then proceed to prove the correct implementation of the operations in the quantum PageRank algorithm. For this task, we utilize the `statevector_simulator` to show the corresponding vector states and `qasm_simulator` to obtain measurement results.

Let us consider the following code

```
from qiskit import QuantumRegister, QuantumCircuit, Aer, execute

#Simulator for viewing the wave function of our quantum system
S_simulator = Aer.backends(name='statevector_simulator')[0]

q0 = QuantumRegister(3, 'q0')
q1 = QuantumRegister(3, 'q1')
anc = QuantumRegister(2, 'a')

qwc = QuantumCircuit(q0, q1, anc)
```

which is a quantum circuit composed by 3 registers, `q0`, `q1` and `anc`, according to the Szegedy formalism and using an additional register for the ancilla qubit. We use the backend `statevector_simulator` and store in the variable `S_simulator` to see the wave function of our system. We denote the `QuantumCircuit` as ’qwc’ (quantum walk circuit). These definitions are the basic template for creating our quantum algorithm in Qiskit. First, we define the number of qubits for both registers and the additional ancilla register. Second, we store them in a `QuantumRegister` and finally, we create a `QuantumCircuit` using the quantum registers.

A quantum walk on 4 nodes

To understand the dynamics of a quantum walk, we provide an implementation of a discrete-time quantum walk on a graph with 4 nodes as presented in Ref. [18].

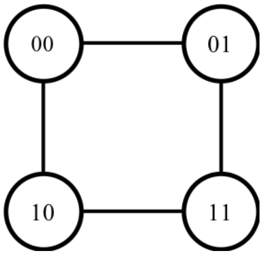


Figure 7.1: A 4-node graph. The starting vertex is labeled as 00.

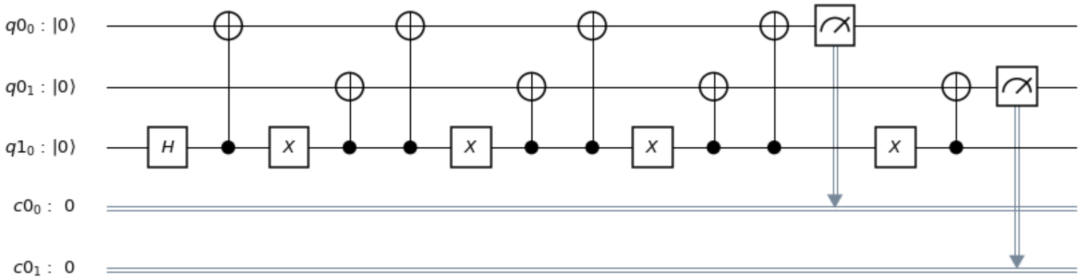


Figure 7.2: Quantum walk circuit implementation on a graph with four nodes

The quantum walk consists of a walk around a 4-node graph (Fig. 7.1). It starts at vertex 00. In the first step, the walker moves to vertex 01 and 10 with equal probability. Then it moves to state 11. The circuit representation of this walk in Qiskit is presented in Fig. 7.2. We provide the corresponding implementation in Qiskit below.

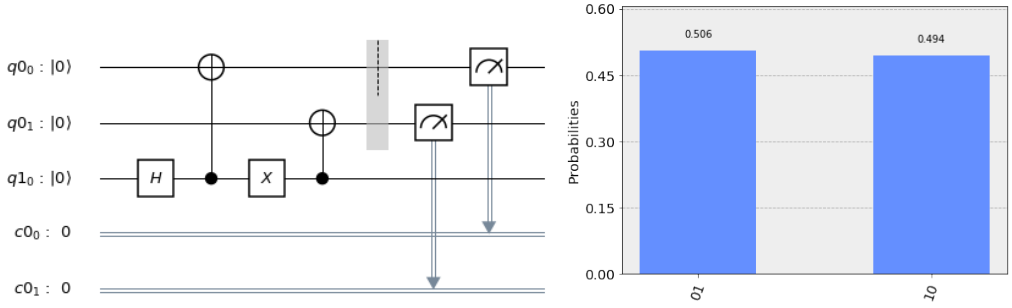
```
#definitions of registers and ancilla qubits
q0 = QuantumRegister(2, 'q0')
```

```
q1 = QuantumRegister(1, 'q1')
c0 = ClassicalRegister(2, 'c0')

qwc = QuantumCircuit(q0, q1, c0)
```

```
#coin operator
qwc.h(q1[0])
#quantum walk
def runQWC(qwc, times):
    for i in range(times):
        qwc.cx(q1[0], q0[0])
        qwc.x(q1[0])
        qwc.cx(q1[0], q0[1])
        #qwc.barrier()
    return qwc

step = 3

qwc = runQWC(qwc, step)
qwc.barrier()

qwc.measure(q0, c0)
```

For this example, we define two registers. The first register is the place where the shift operator acts. In this case, a CNOT gate acts as the shift operator on qubit `q[0]` and qubit `q[1]`. The coin operator on qubit `q[2]`, is a Hadamard gate. Now we show the results on each step.

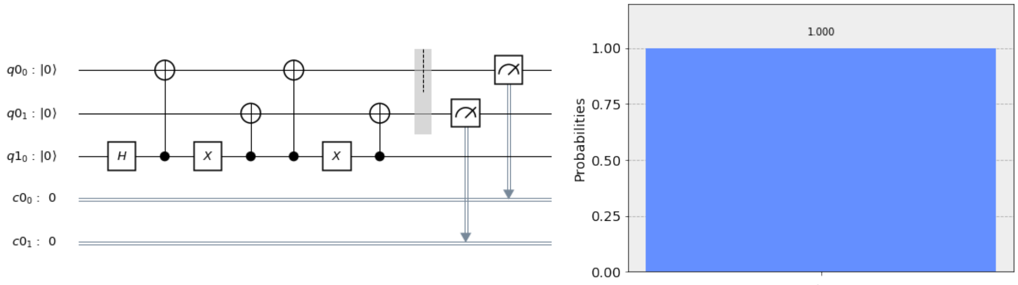
1. Step 1:
    - One step of the walk. Circuit Fig. 7.3a
    - Results Fig. 7.3b
2. Step 2:
    - Two steps of the walk. Circuit Fig. 7.4a
    - Results Fig. 7.4b
3. Step 3:



(a) One step of the walk.

(b) Step 1 chooses 01 or 10 with equal probability.

Figure 7.3: One-step quantum walk on 4 nodes.



(a) Two steps on the 4 node graph.

(b) Step 2 chooses 11.

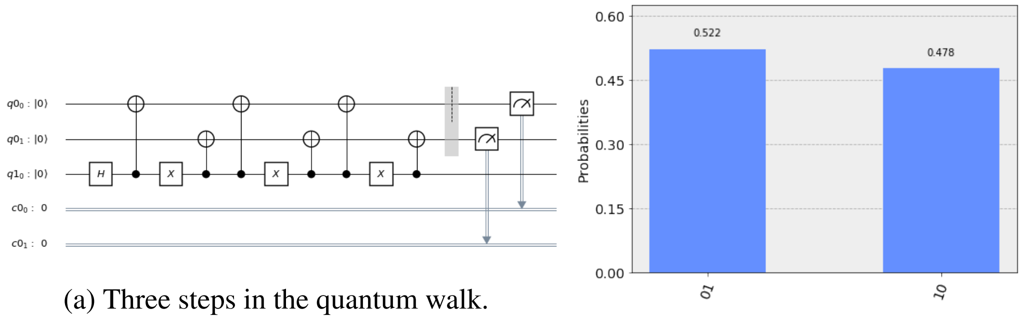
Figure 7.4: Two-step quantum walk on 4 nodes.

- Three steps of the quantum walk. Circuit Fig. 7.5a
- Results Fig. 7.5b

Initial template

To start constructing our circuit we start by providing the Qiskit layout. Eight qubits are being set up and numbered as $q0, q1, anc$, consisting of 3, 3 and 2 qubits respectively. The measure operation is applied to the qubits which extracts an output of 0 or 1.

Qubits are always initialized to give 0 as an output. To see this fact, we obtain a histogram showing that the result is in fact `000000`. The reason for showing the results as a histogram is because the randomness nature of quantum computers. In this case, since we are not performing operations we obtain `000000` in both registers with certainty.



(a) Three steps in the quantum walk.

(b) Step 3 goes from node 11 and chooses 01 and 10.

Figure 7.5: Three-step quantum walk on 4 nodes.

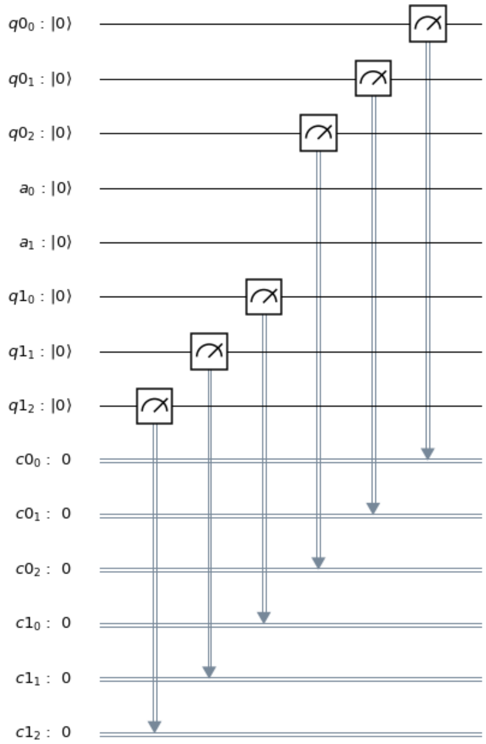


Figure 7.6: Circuit

As mentioned in Sec. 5, the results are going to be obtained from a quantum simulator, which is classical computer doing what a quantum computer would do. Now we simulate the initial condition on which the circuit starts to operate.

State Superpositions

A qubit can exist in a continuum of states between $|0\rangle$ and $|1\rangle$, until it is observed [41]. Applying a Hadamard gate to either a $|0\rangle$ or $|1\rangle$ qubit, will produce a quantum state that, if observed, will be a 0 or 1 with equal probability. From Ch. 3 and Postulate 4, we know that the state space of the composite system is a tensor product of each of the systems composing it. Hence, the initial superposition is given by $\hat{H} \otimes \hat{H} \otimes \hat{H}$ acting on initial qubits in states $|0\rangle$. We obtain a superposition state resulting from 3 qubits, as presented in Fig. 7.7. We present a mathematical analysis of this state in superposition.

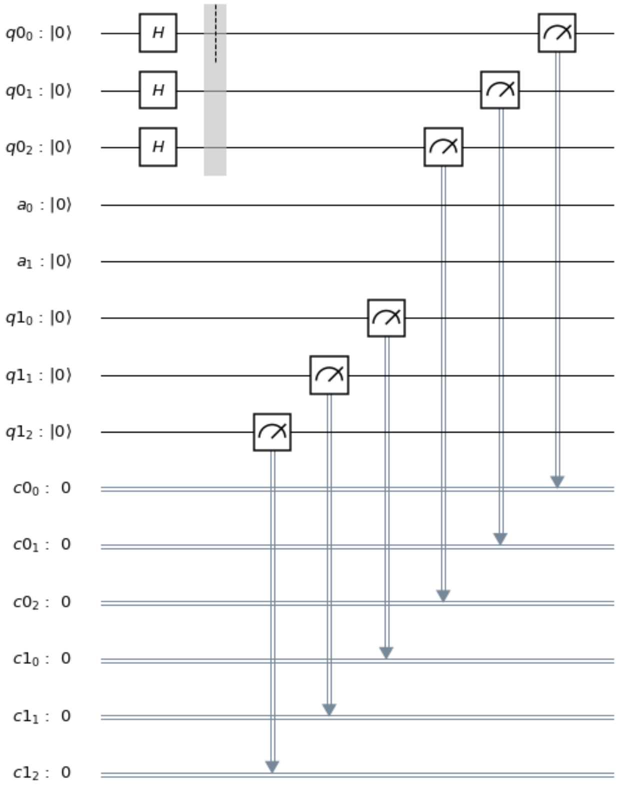


Figure 7.7: Initial superposition

The mathematical representation of the superpositions of the three qubits in register `q0` is as follows:

$$\begin{aligned}
|\psi_t\rangle &= \hat{H}\,|0\rangle \otimes \hat{H}\,|0\rangle \otimes \hat{H}\,|0\rangle \\
&= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \\
&= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle\,|0\rangle + |0\rangle\,|1\rangle + |1\rangle\,|0\rangle + |1\rangle\,|1\rangle}{2}\right) \\
&= \frac{1}{2\sqrt{2}}\left(|0\rangle\,|00\rangle + |0\rangle\,|01\rangle + |0\rangle\,|10\rangle + |0\rangle\,|11\rangle + |1\rangle\,|00\rangle + |1\rangle\,|01\rangle\,|1\rangle\,|10\rangle + |1\rangle\,|11\rangle\right)
\end{aligned}$$

Therefore, for each state we have the following probabilities:

$$\left|\frac{1}{2\sqrt{2}}\right|^2 = \frac{1}{\left(2^{3/2}\right)^2} = \frac{1}{2^3} = \frac{1}{8} = 0.125$$

Applying a measurement process, as explained in Sec. 3 and Postulate 3, results in probabilities given by $|\alpha_x|^2$, with qubits states after the measurement being $|x\rangle$. The condition that probabilities sum to one is stated by the normalization condition $\sum_{x\in\{0,1\}^3} |\alpha|^2 = 1$, where the notation $\{0,1\}^3$ means 'the set of strings of lenght 3 with each letter being zero or one'.

In general, a system of n qubits is composed of a computational basis of the form $|x_1x_2\ldots x_n\rangle$, so a quantum state of such system is specified by 2^n amplitudes. In this case we have $n = 3$, therefore, we have 8 amplitudes. In fact, we observe a measurement performed in Qiskit in Fig. 7.8. The results given by the simulator produces values around 0.125 as calculated before.

For visualizing and printing the wave function of the system we use the function called `Wavefunction` from an additional python file: `Our_Qiskit_Functions`, available at [29]. This additional python file will allow us to to see the states of our system in a standard ket notation.

The wave function printed below shows an equal superposition of eight states with amplitudes $\frac{1}{2\sqrt{2}}$ or 0.35355. Here, the numbers attached to the states are the system's amplitudes, not probabilities. The probabilities in Fig. 7.8, are the observables of the system, but amplitudes are the inner workings. Hence, each state has amplitude of $\frac{1}{2\sqrt{2}}$, which when squared results in all states having probability of $\frac{1}{8}$ or 0.125.

```
0.35355 |000>
0.35355 |100>
```

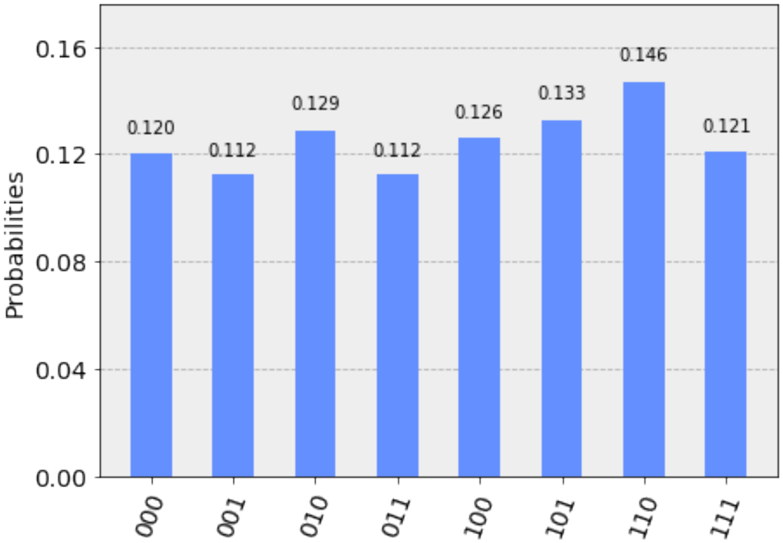


Figure 7.8: Measurement results of an equal superposition of 3 qubits.

```
0.35355 |010>
0.35355 |110>
0.35355 |001>
0.35355 |101>
0.35355 |011>
0.35355 |111>
```

These eight states comes from the mathematical calculations made previously. They were obtained using the backend designed for that purpose, the `statevector_simulator`.

Cyclic permutations

It is important to notice that the theoretical framework for the construction of quantum circuits of Szegedy quantum walks in Ref. [33, 35] is applied to the class of Markov chains where the transition matrix (see Eq. 2.2) is described by a cyclic permutation. Now, we define this concept.

Definition 18 ***Cyclic permutation.*** *A permutation which shifts all elements of a set by a fixed offset, where the elements shifted off to the end are added back to the beginning. A cyclic permutation of one place to the left would produce* $\{a_2, a_3, \ldots, a_n, a_1\}$ *for a given* *set of elements* $\{a_1, a_2, \dots, a_n\}$*. Similarly a one place cyclic permutation to the right will produce* $\{a_n, a_1, a_2, \dots a_{n-1}\}$.

From Def. (18), we extend the idea to permutation matrices.

Definition 19 ***Permutation matrix.*** *A permutation matrix is a matrix obtained by permuting the rows of an* $n \times n$ *identity matrix according to some permutation offset. Every row and column therefore contains precisely a single* 1 *with* 0*'s everywhere else, and every permutation corresponds to a unique permutation matrix. There are therefore* $n!$ *permutation matrices of size* n*, where* $n!$ *is a factorial.*

Let R be the one-element right rotation operator such that $R[c_1, c_2, c_{N-1}, c_N]^T = [c_N, c_1, c_2 ..., c_{N-1}]$ and $L = R^\dagger$ is the one-element left-rotation operator [33]. In correspondence with notation, we know that state $|00\rangle$ can be represented as a vector

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

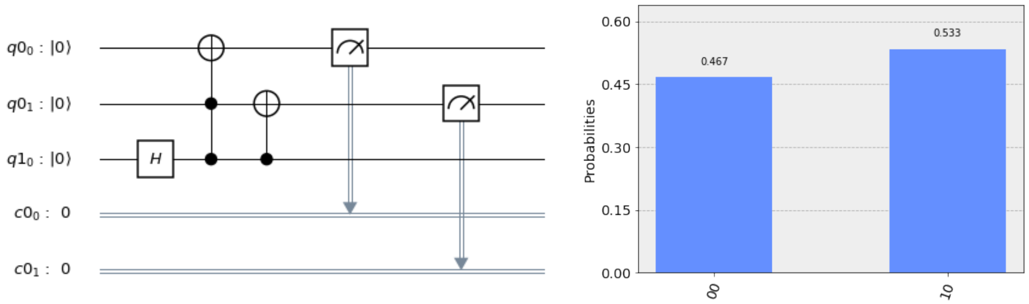
We show the R operator in Fig 5.6 and the L operator in Fig. 5.7. In fact, we are able to simulate correctly both operations using Qiskit. We show the procedure in 2 qubits, as in Fig. 7.9. A one-step cyclic permutation is composed by a Hadamard gate in an additional qubit and two gates, the first one a Toffoli gate and the second one a CNOT gate (see Sec. 5.2).

Applying the right permutation on the starting state $|00\rangle$, i.e, $R\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$, will produce a $|01\rangle$ state or $\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$.

The process of applying a right permutation repeatedly is as follows

$$R\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow R\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow R\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \rightarrow R\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \rightarrow R\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Fig. 7.9a shows the circuit for one-place cyclic permutation to the right operator R and its result on Fig. 7.9b. Similarly, a sequence of two operators for one-place permutation to the right will move the element two places to the right, reflected in the state $|10\rangle$ as in Fig. 7.10. Applying this operation consecutively starting from state $|00\rangle$ will result in $|00\rangle \rightarrow |01\rangle \rightarrow |10\rangle \rightarrow |11\rangle \rightarrow |00\rangle$ and so on. The results show a superposition state due to the nature of the Hadamard gate.

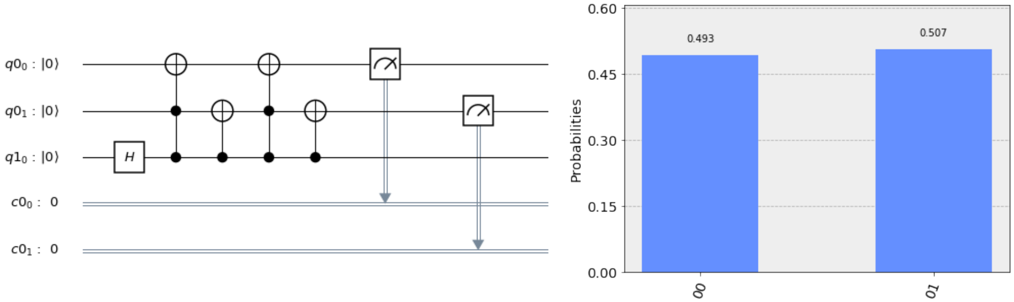


(a) Circuit for one-place cyclic permutation to the right.

(b) Measure results on one-place cyclic permutation to the right.

Figure 7.9: One-place cyclic permutation to the right.

Note that the states in the results are printed in reverse order such that the qubit 0 (the top register) is stored in the classical register index 0. In classical computing, the rightmost index is defined as the least significant bit (LSB). Hence, in Qiskit we consider the states such that qubit 0 (the top register) is the leftmost index. We use the `Wavefunction` function to print correctly the states.



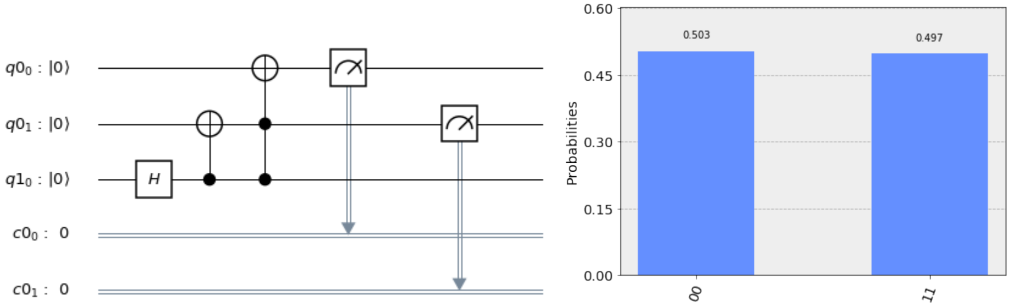
(a) Application of the R operator cyclic permutation circuit.

(b) Results of two-place cyclic permutation.

Figure 7.10: Two-place cyclic permutation to the right.

For the one-element left-rotation operator L (Fig. 5.7), we simulated the operation to perform the correct procedure. To provide a solution to the problem of controlling the operations with 0's entries instead of 1's we take advantage to the fact that $L = R^{\dagger}$. Hence we propose the following circuit, switching the order of the Toffoli gate and CNOT gate as in Fig. 7.11a.

We proceed to show this operation. Starting from state $|00\rangle$ we obtain $L\,|00\rangle =$

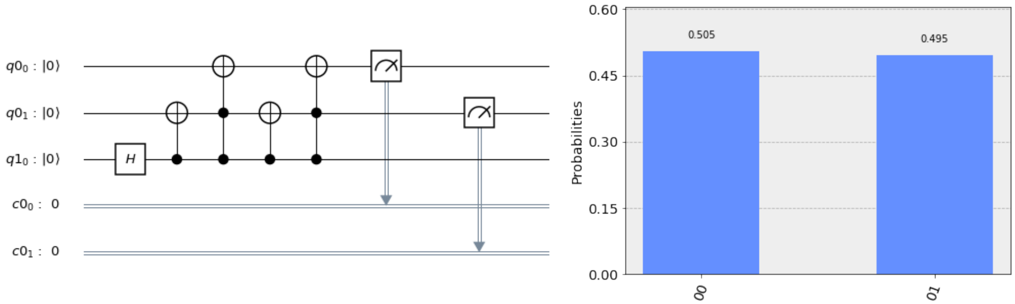


(a) L operator cyclic permutation circuit.

(b) Results of one-place cyclic permutation.

Figure 7.11: One-place cyclic permutation to the left results.

$|11\rangle$, or equivalently $L\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^T = \begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}^T$, shifting the 1 element to the left. Similarly, applying the same operator we obtain, $|00\rangle \rightarrow |11\rangle \rightarrow |10\rangle \rightarrow |01\rangle \rightarrow |00\rangle$, and so on.



(a) Circuit of two-place cyclic permutation to the left, L operator.

(b) Results of two-place cyclic permutation to the left.

Figure 7.12: Two-place cyclic permutation to the left.

The process of applying the left permutation repeatedly is as follows:

$$L\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \rightarrow L\begin{bmatrix}0\\0\\0\\1\end{bmatrix} \rightarrow L\begin{bmatrix}0\\0\\1\\0\end{bmatrix} \rightarrow L\begin{bmatrix}0\\1\\0\\0\end{bmatrix} \rightarrow L\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

which is represented by the following states:

$$L\,|00\rangle \rightarrow |11\rangle : L\,|11\rangle \rightarrow |10\rangle\,; L\,|10\rangle \rightarrow |01\rangle\,; L\,|01\rangle \rightarrow |00\rangle\,.$$

Multiple controlled gates

To implement our circuit we are required to apply gates controlled by multiple qubits, as described in Sec. 5.3. While Qiskit provides an universal set of gates, it does not provide a tool to perform operations on multiple qubits. However, we can define our own procedure to allow multiple controlled operations. This is direct procedure using Toffoli and CNOT gates, and making use of ancilla qubits.

To begin with, let us consider a 3-qubit controlled U gate operation as in Fig. 7.13. In this example, the U gate is controlled by the state $|100\rangle$. For simplicity we consider U to be an X gate.

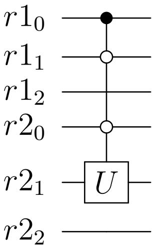


Figure 7.13: 3-controlled gate.

To implement Fig. 7.13, we make use of $(N-1)$ ancilla qubits as explained before. The conditional on the control qubits being set to zero, are implemented applying an X operation before the qubits conditioning the Toffoli gate (see Fig. 7.14). Now, we show the code representing the circuit in the following lines.

```
r1 = QuantumRegister(3, 'q0') #register1
r2 = QuantumRegister(3, 'q1') #register2
anc = QuantumRegister(2, 'a') #ancilla qubits
c0 = ClassicalRegister(3, 'c0')
c1 = ClassicalRegister(3, 'c1')

qctest = QuantumCircuit(r1, anc, r2, c0)

qctest.h(r1[0])
qctest.h(r1[1])
qctest.h(r1[2])
qctest.barrier()
print('---initial state---')
oq.Wavefunction(qctest, systems=[3,2,3], column= True, show_systems=[True, False, True]
qctest.x(r1[1])
qctest.ccx(r1[0], r1[1], anc[0])
qctest.x(r2[0])
qctest.ccx(anc[0], r2[0], anc[1])
qctest.cx(anc[1], r2[1])
qctest.ccx(anc[0], r2[0], anc[1])
qctest.x(r2[0])
qctest.ccx(r1[0], r1[1], anc[0])
qctest.x(r1[1])

print('---final state---')
oq.Wavefunction(qctest, systems=[3,2,3], column= True, show_systems=[True, False, True]

qctest.draw(output='mpl')
```

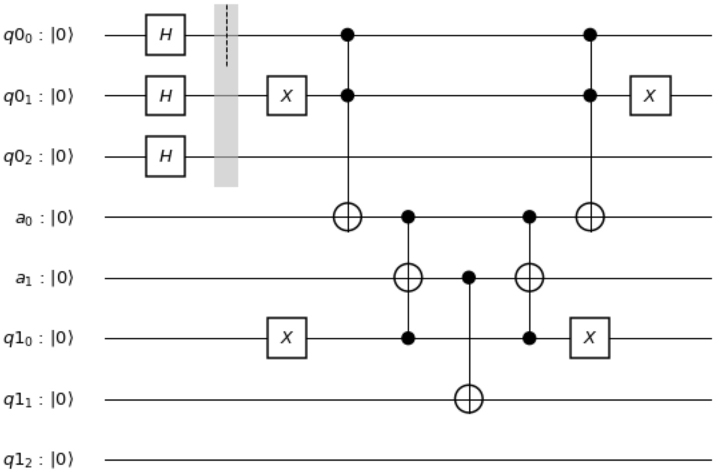


Figure 7.14: 3-controlled gate implemented in qiskit.

The results that produces the circuit are shown below:

```
---initial state---
0.35355 |000>|000>
0.35355 |100>|000>
0.35355 |010>|000>
```

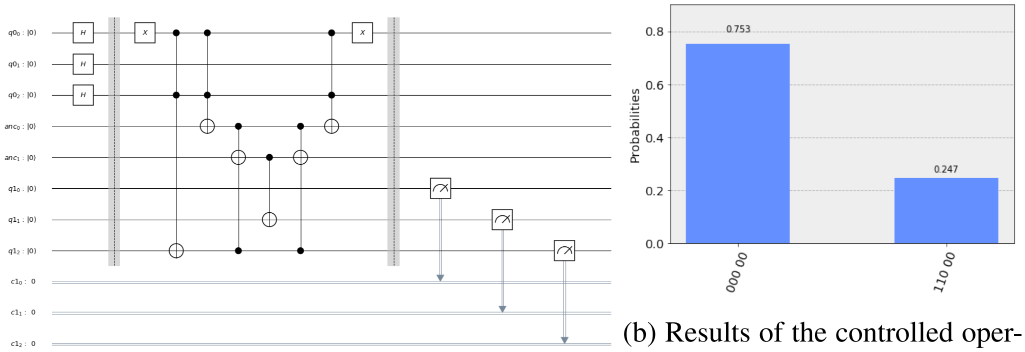
```
0.35355 |110>|000>
0.35355 |001>|000>
0.35355 |101>|000>
0.35355 |011>|000>
0.35355 |111>|000>

---final state---
0.35355 |000>|000>
0.35355 |010>|000>
0.35355 |110>|000>
0.35355 |001>|000>
0.35355 |011>|000>
0.35355 |111>|000>
0.35355 |100>|010>
0.35355 |101>|010>
```

The results of the proposed circuit have successfully picked out the control-state $|1\rangle|0\rangle|0\rangle$ and applied the NOT operation to the second qubit on the second register. We were able to effectively use the desired state $|100\rangle$ as the control state. The same procedure is applied to all multiple controlled qubit gates in our circuit. The `--initial state--` result shows a superposition for the first register, and the `--final state--` shows the NOT operation when controls are in state $|100\rangle$ respectively. The resulting circuit in Qiskit for this test circuit is shown in Fig. 7.14.

Controlled cyclic permutation

In the section about 'Cyclic permutations' we have shown the functioning of the cyclic permutation operation realized by R and L operators. Now, we want to control both operators with $|0\rangle$ and $|1\rangle$ controls. These operations are localized in the $T_{1,y}$ and $T_{1,y}^{\dagger}$ sections of the circuit (see Fig. 6.3).

Fig. 7.15a, shows the circuit proposed for controlling the cyclic permutation L-operator. First, we use Hadamard gates in the `q0` register to induce a superposition that will generate the 0 and 1 states to control our gates. The measurement operation is applied to the register of interest, `q1`. The implementation approach for this controlled-L operator is to control individually the gates composing the L operator as shown in Fig. 7.15a. We are able to obtain the desired results showed in Fig. 7.15b. However, it is important to notice that the probability results are diminished when using additional control states. We can conclude that adding an extra control to the circuit reduces the probability of observing the state by half. The reason behind this result is that the first control has a 50% chance to



(a) Controlled cyclic permutation circuit to the left.

(b) Results of the controlled operation of the cyclic permutation to the left.

Figure 7.15: Controlled cyclic permutation to the left.

pick up the correct state from the Hadamard gates, and the second control has the chance to select the correct state from that 50% of the first control.

Transformations

The K_b transformations are preparation routines for preparing the initial state $|\phi_0\rangle$ from some computational basis state $|n\rangle$. These preparations form the central part of our circuit. As mentioned in Sec. 4.1.2, Sec. 6.2.1, and Eq. 5.1, we want to implement the Szegedy walk operator for a given Markov chain, which we rewrite here

$$U_{walk} = \mathcal{S}(I - 2\Pi) = \mathcal{S}\mathcal{R}$$

In order to implement the reflection operator as explained in Ref. [33], we diagonalize it using a unitary operator U:

$$U\mathcal{R}U^\dagger = U(I - 2\Pi)U^\dagger = I - 2U\Pi U^\dagger$$

where $U = \sum_{i=0}^{N-1} |i\rangle\langle i| \otimes U_i$. Then expanding the expression above using the projector states in Eq. 4.9 and projector operator in Eq. 4.11 and 4.13 gives:

$$U\mathcal{R}U^\dagger = I - 2\sum_{i=0}^{N-1} |i\rangle\langle i| \otimes \left(U_i |\phi_i\rangle\langle\phi_i| U_i^\dagger\right)$$

U_i acts to transform $|\phi_i\rangle$ into a fixed computational basis state $|b\rangle$, i.e. $U_i|\phi_i\rangle = |b\rangle$ or conversely the inverse operator $U^\dagger$ generates $|\phi_i\rangle$ form $|b\rangle$ [33]. Then

$$U\mathcal{R}U^\dagger = I - 2I_N \otimes |b\rangle\langle b| = I_N \otimes (I - 2|b\rangle\langle b|) \equiv D.$$

which is a diagonal matrix implemented using a controlled-π gate (implemented as `pauliz`). Hence, [35] arrives to the Szegedy walk operator

$$U_{walk} = \mathcal{S}U^\dagger D U \tag{7.1}$$

Now, due to symmetries in the transition matrix P, it is possible to implement U efficiently. Consider $U_i = K_b^\dagger T_i$, such that $K_b^\dagger |\phi_r\rangle = |b\rangle$ and $T_i |\phi_i\rangle = |\phi_r\rangle$; K_b and T_i are both unitary operators and $|\phi_r\rangle$ is a chosen reference state. That is, $K_b^\dagger$ transform the state $|\phi_r\rangle$ into $|b\rangle$ or its inverse operation k_b prepares the state $|\phi_r\rangle$ from a computational basis state $|b\rangle$. In the same direction, T_i transforms $|\phi_i\rangle$ into $|\phi_r\rangle$. The above operations transform the ith column of P into the rth column of P (with square roots on both). The quantum circuit implementation is represented in Fig. 5.9.

If E is a matrix with columns related by cyclic permutations as defined before, the matrix G also has the same property. To avoid obtaining an equal PageRank in each vertex it is necessary to partition the vertex into subsets using the Google matrix G as the transition matrix.

As before, it is possible to construct the preparation routine K_b explicitly. The transformations $T_{1,y} : |\phi_0\rangle \rightarrow |\phi_0\rangle$ can be defined as a restricted cyclic permutation of the reference state $|\phi_0\rangle$.

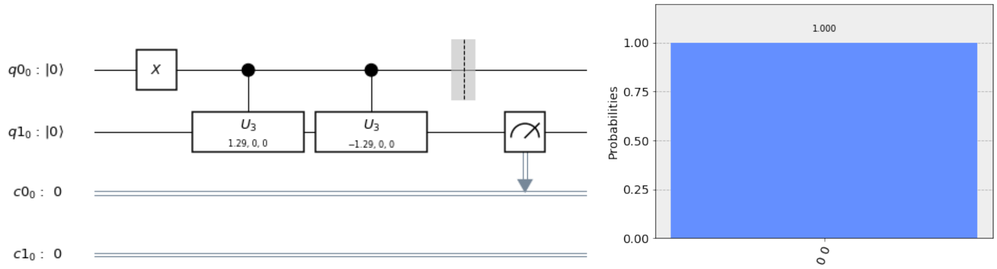
In our case, the vertices are partitioned into subsets of equivalent vertices $Z = Z_1 \cup Z_2 \cup Z_3$, where $Z_1 = \{0, 1, 2, 3\}$, $Z_2 = \{4, 5\}$ and $Z_3 = \{6, 7\}$.

Now, we provide simulations to prove that we have the correct implementation $UU^\dagger = I$ operation in Qiskit. We are required to implement this operation for K_{b_1}, K_{b_2}, K_{b_3} and its dagger operators (†). The K_b operators are mainly composed of R_y rotations and Hadamard gates as seen in Fig. 6.4, 6.5, 6.6.

Let us start with R_y rotations. It is not evident how to obtain the dagger operations for the rotations used in K_{b_1}, K_{b_2} and K_{b_3}. We start by proving that using a negative angle in this rotation we obtain its dagger equivalent. We know from Definition 9, that the dagger operator $U^\dagger$ is equivalent to applying the transpose and complex conjugate of U. The transpose operation does not change the gate, however, the complex conjugate operation changes the sign of the complex number. That is, $e^{i\theta} = \cos\theta + i\sin\theta$, and $e^{-i\theta} = \cos\theta - i\sin\theta$, where $\sin(-\theta) = -\sin(\theta)$ and $\cos(-\theta) = \cos(\theta)$.

Thus, applying $UU^\dagger = I$ would be equivalent to performing an identity operation. If $UU^\dagger$ acts on a qubit initialized in $|0\rangle$, we obtain the $|0\rangle$ state. If $U = R_y$, we apply $R_y R_y^\dagger$ to the $|0\rangle$ state as shown in Fig. 7.16. If $R_y R_y^\dagger$ acts on a $|1\rangle$ state we obtain the state $|1\rangle$.

The same operations are applicable to the Hadamard and controlled-Hadamard in K_{b_1}, K_{b_2} and K_{b_3}. Applying a dagger operator to H, does not change the gate. Hence,



(a) Controlled-R_y circuit using cu3 gate.

(b) Results of the controlled-$R_y R_y^\dagger$ operation.

Figure 7.16: $R_y R_y^\dagger = I$

$HH^\dagger = I$. The same analysis is applied for the controlled-Hadamard gates. We are able to show the correct implementation of inverse operations $K_{b_1}K_{b_1}^\dagger$, $K_{b_2}K_{b_2}^\dagger$ and $K_{b_3}K_{b_3}^\dagger$ are equal to the identity.

The central part of our circuit is a diagonal matrix D, implemented using a Pauli-z gate (described in Sec. 5), according to Eq. 7.1, with $D' = I_N - 2\left|b\right\rangle\left\langle b\right|$. It is controlled by $\left|0\right\rangle\left|0\right\rangle$ states. The result of applying this gate is a change in the phase of the state it is acting on. When applied to state $\left|0\right\rangle$, it does not change the phase. When applied to state $\left|1\right\rangle$, it changes the phase of the state. It is possible to observe it using the `statevector_simulator` and our function `Wavefunction`. We show the acting of the z gate below:

$$z\left|0\right\rangle = \left|0\right\rangle$$

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

$$z\left|1\right\rangle = -\left|1\right\rangle$$

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

Our proposal to implement this circuit is in Fig. 7.17 for the Pauli-z controlled gate D. The expected results are obtained with this circuit. We use the function `Wavefunction` since making a measurement on this state results in a positive number.

```
-----final state------
-1.0 |001>
```

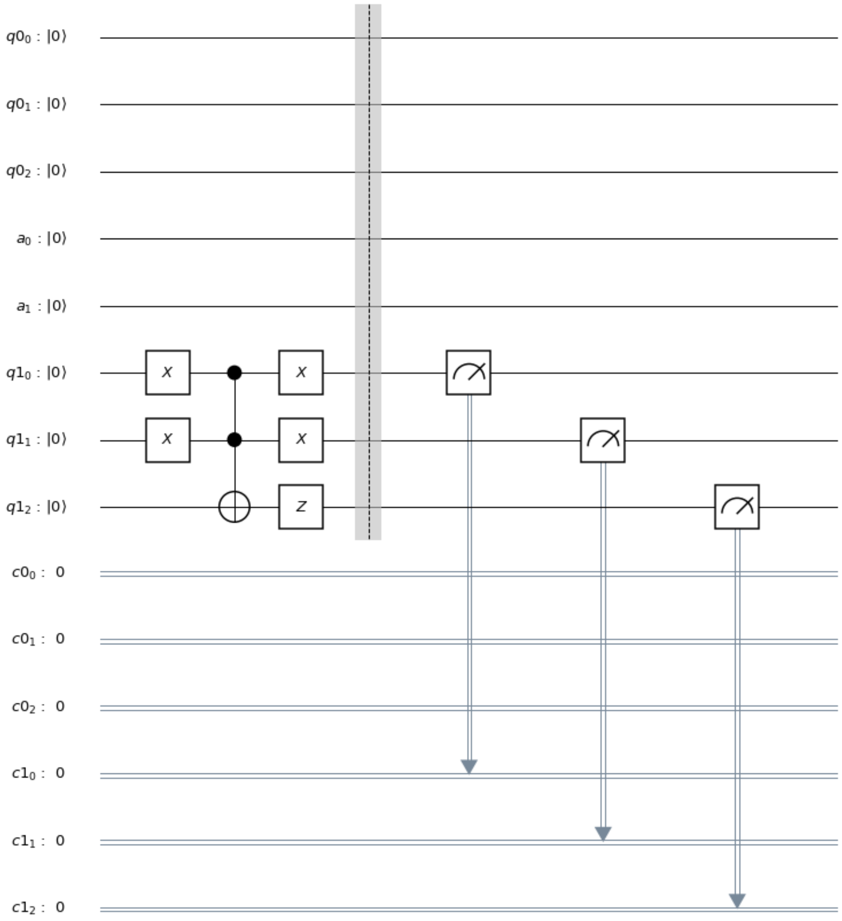


Figure 7.17: Circuit for Pauli-z controlled operation, D.

Szegedy quantum walk for the Quantum PageRank

Now, we are able to show the complete implementation of our circuit. The analysis provided before allows us to implement the circuit presented in Fig. 6.3 for the graph in Fig. 6.2. We present each of the gates discussed above in the following code. We provide circuit diagrams and functions definitions, in Appendix A.

```python
#definitions of registers and ancilla qubits
q0 = QuantumRegister(3, 'q0')
q1 = QuantumRegister(3, 'q1')
anc = QuantumRegister(2, 'a')
c0 = ClassicalRegister(3, 'c0')

qwc = QuantumCircuit(q0, anc, q1, c0)

#Quantum walk definition

#superposition
qwc.h(q0[0])
qwc.h(q0[1])
qwc.h(q0[2])
qwc.barrier()

def runQWC(qwc, times):
    for i in range(times):
        t1y(qwc, q0, q1, anc)
        t2y(qwc, q0, q1, anc)
        kb1dag(qwc, q0, q1, anc)
        kb2dag(qwc, q0, q1, anc)
        kb3dag(qwc, q0, q1, anc)
        pauliz(qwc, q0, q1, anc)
        kb3(qwc, q0, q1, anc)
        kb2(qwc, q0, q1, anc)
        kb1(qwc, q0, q1, anc)
        t2ydag(qwc, q0, q1, anc)
        t1ydag(qwc, q0, q1, anc)
        swap(qwc, q0, q1)
    return qwc
```

```

step = 1

qwc = runQWC(qwc, step)

qwc.barrier()
qwc.measure(q1, c1)
#qwc.draw(output='mpl', scale=0.9)

#qasm simulator
backend = Aer.get_backend('qasm_simulator')

outputstate = execute(qwc, backend).result().get_counts(qwc)

plot_histogram(outputstate)
```

We start the walk in a superposition state in order to obtain $|0\rangle$ and $|1\rangle$ states that will control our gates as explained before. A quantum barrier will allow to execute first the superposition. We define each controlled gate as functions defined in Appendix A.

Due to the randomness nature of the measurement we provide an expected result for a multiple step walk; the results obtained using the `qasm_simulator` are shown in Fig. 7.18 for the Szegedy quantum walk.

Using visualization options of the `plot_histogram` we are able to compare the differences in the probability results making a second measurement and plotting against the first one.

We obtain this result at step 7 of the quantum walk. This result makes sense since we are permuting the states which takes 7 steps to cover all states. In other words, the cyclic permutation takes 7 steps to include all states $|000\rangle \rightarrow |001\rangle \rightarrow |010\rangle \rightarrow |011\rangle \rightarrow |100\rangle \rightarrow |101\rangle \rightarrow |110\rangle \rightarrow |111\rangle$. We see that it converges at the desired results at step 7 and 14, more than that the results are not clear since more errors are introduced. Since we grouped similar nodes in set Z_1, Z_2 and Z_3 we consider the nodes within each set to be equivalent. Hence, we are able to provide the average quantum PageRank for the Szegedy walk in Fig. 7.20.

7.1 Quantum PageRank algorithm

As mentioned before, the quantum PageRank algorithm is a quantization procedure of the Markov chain encoded in the Google matrix. The quantization procedure is a transition

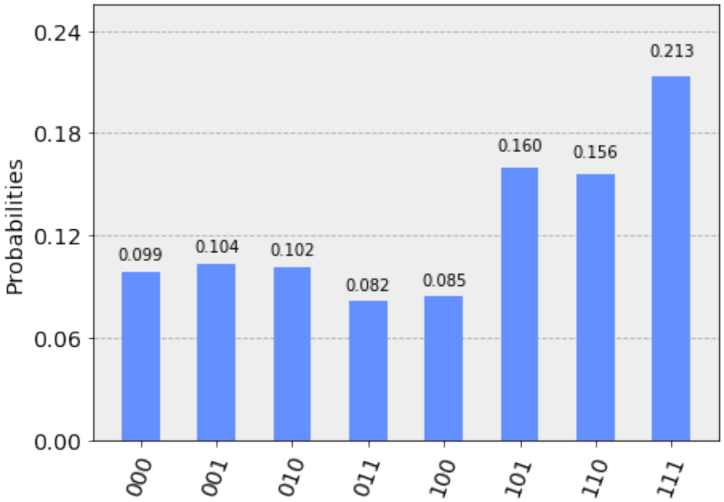


Figure 7.18: Histogram results of the Szegedy quantum walk.

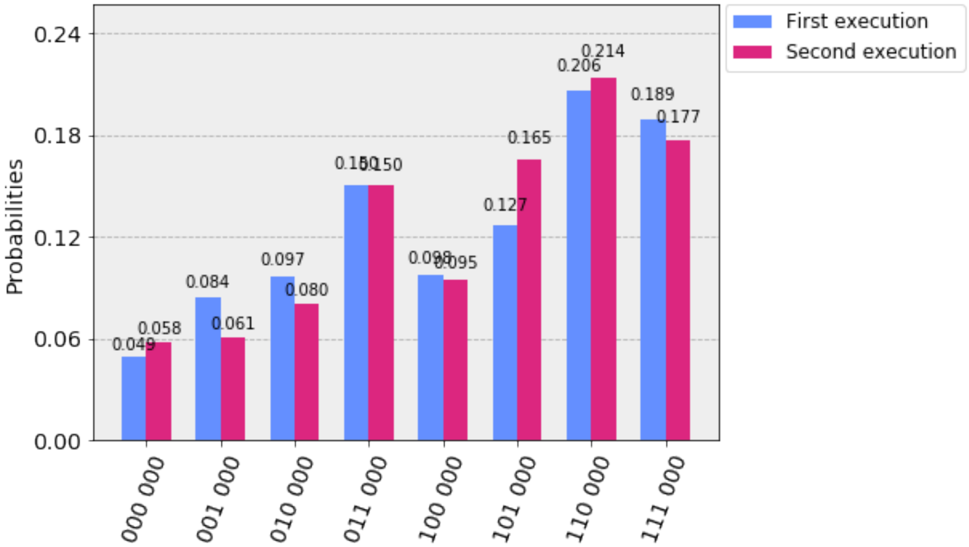


Figure 7.19: Szegedy walk results.

from a classical understanding of certain phenomena into an understanding in terms of quantum mechanics. As we have stated in this work, we utilised the framework provided by the Szegedy quantum walk to develop a quantum understanding of the classical PageRank algorithm. In general, a Szegedy quantum walk is performed on a duplicated Hilbert

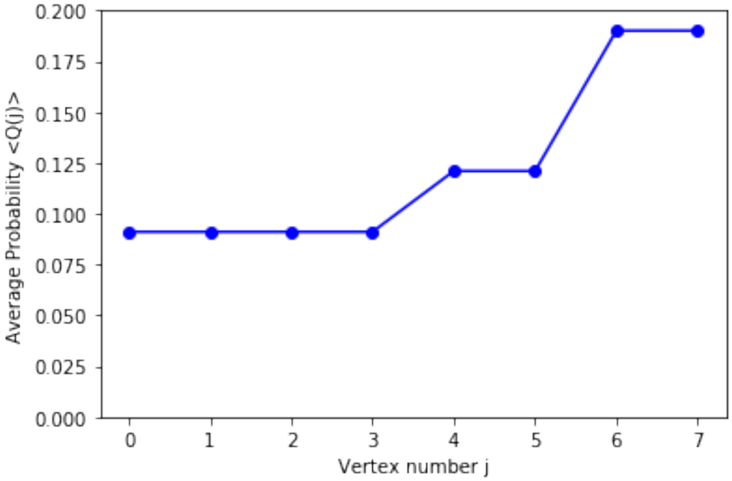


Figure 7.20: Average Quantum PageRank with Szegedy walk.

space $\mathcal{H}_1 \otimes \mathcal{H}_2$, represented in Fig. 7.21.

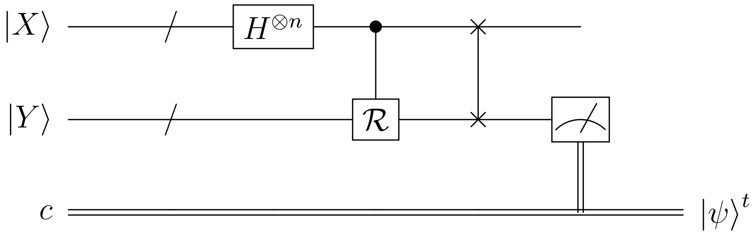


Figure 7.21: A general Szegedy quantum walk circuit.

From Sec. 4.1.2, we know that the Szegedy's model considers a connected bipartite graph $\Gamma(X, Y, E)$, where X, Y are disjoint sets of vertices and E is the set of non-directed edges. In the circuit model, we represent both sets of n qubits as $|X\rangle$ and $|Y\rangle$ as represented in Fig. 7.21. The controlled operations represent the mapping from the set X to the set Y and the swap operator the opposite direction, from Y to X. We indicate n number of qubits using the symbol / in registers $|X\rangle$ and $|Y\rangle$.

In Sec. 6.2 and 6.2.1, we define the general procedure to make possible the quantum PageRank. Now we summarize the algorithm.

1. Create a uniform superposition applying Hadamard gates to the first register,

$$|s\rangle = \frac{1}{\sqrt{2^N}} \sum_{i=0}^{N-1} |\psi_i\rangle$$

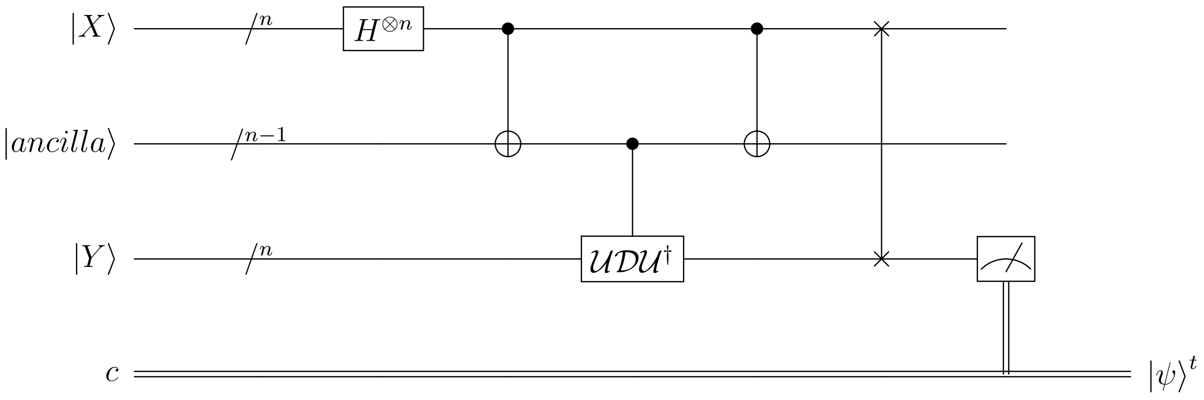


Figure 7.22: Szegedy circuit using ancilla qubits.

2. Implement the Szegedy walk operator U_{walk}, see Sec. 6.2.1
    - Diagonalize the $\mathcal{R}$ operator (in our case some symmetries allow to implement it efficiently). See Fig. 5.9.
    - Implement the swap operator $\mathcal{S}$. See Fig. 5.8.
3. Evolve the system repeating the circuit at least n times.
4. Measurement in the second Hilbert space. The instantaneous PageRank of the jth vertex is given by:
$$Q(j,t) = |\, \langle j|_2 \, U_{walk} \, |\Psi_0\rangle \,|^2$$
$\langle j|_2$ is the jth standard basis vector of the second Hilbert space $\mathcal{H}_2$

To implement the circuit we make use of an additional register. We use ancilla qubits for gates controlled by 3 or more control-qubits. We express this additional register in Fig. 7.22.

In comparison with the evolution operator from the coined model, $U(t)$, we implemented the step operator U_{walk} by repeating the circuit t times. Here, a single application of U_{walk} propagates the corresponding amplitude to adjacent vertices [35]. Recall that the Szegedy operator is given by

$$U_{walk} = \mathcal{S}(2\Pi - I) = \mathcal{S}\mathcal{R}$$

$\mathcal{S}$ and $\mathcal{R}$ correspond to the shifting and coin operator of a coined quantum walk. The operator $\mathcal{S}$ is easily implemented, whereas $\mathcal{R}$ represents a difficulty to implement. In this section we have shown, step-by-step how it is possible to implement a Szegedy walk for computing the Quantum PageRank.

Chapter 8

Discussion

We know from Sec. 6.1 that the problem of finding the PageRank of a website is equivalent to the problem of finding the eigenvector with eigenvalue one of a matrix. Also, we know that the hyperlink matrix H is given by the number of webpages indexed, and forms a square matrix. Using the hyperlink connectivity structure alone, it is a necessary but not sufficient condition to ensure convergence of the stationary vector r_k. To ensure a meaningful stationary state we go through a series of modifications of the original hyperlink structure. In the case of web pages having *dangling nodes*, the columns corresponding to dangling nodes are replaced with entries $1/N$, where N is the number of nodes. This strategy is equivalent to linking every dangling node to every node in the web including itself as shown in Fig 8.1.

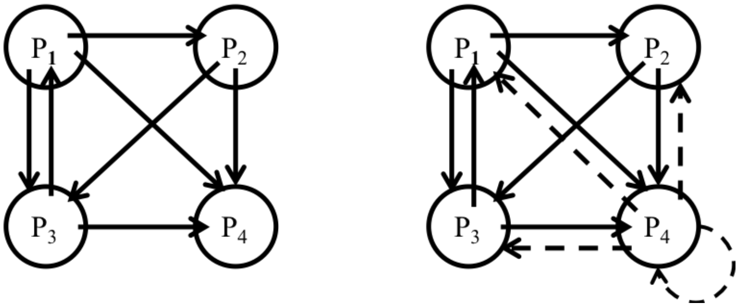


Figure 8.1: Linking every dangling node to every node in the web ensures ensures stochasticity.

By this modification, the possibly disconnected graph is 'patched' and becomes effectively connected. To ensure convergence we also require that the matrix E is primitive or regular, i.e. that there is an integer m such that E^m has only positive entries. The interpretation of the modified hyperlink matrix is the random walker diffusing on the network as a time step evolution operator. The assumption is that the walker will eventually arrive to any page using a path of at least m links.

The notion behind irreducibility is that subgraphs within the web do not link back and forth with each other, i.e there are no paths from a subgraph to another, leading to an irrelevant importance vector. Hence, we arrive to a key idea: in order to obtain the Pagerank vector the graph must be *strongly connected.* This fact is expressed in the Google matrix, Eq. 6.2.

The connection between the Google matrix and random walk is direct. We define a set of random variables $X_0, X_1, \ldots, X_T$, where T is the number of time steps. These variables take values on the set of nodes $\{P_i\}$ in the web. Translating the Google PageRank into the language of Markov chains we have:

$$G_{ij} = Pr(X^{(n+1)} = P_i | X^{(n)} = P_j) \tag{8.1}$$

The PageRank result is a stationary distribution of the Markov chain. This interpretation of the Google PageRank in terms of Markov chains, i.e random walks, is necessary to make a connection with the quantum equivalent.

In the quantum counterpart, the walker starts its dynamics with an initial position $|\psi_0\rangle$, and a defined evolution operator. Then a projection measurement is performed into the basis states encoding the node positions. We interpret the resulting probability distributions as the *instantaneous Quantum PageRank* of the node.

The quantum version of finding an unknown vector $\vec{x}$ as in the following linear equation: $A\vec{x} = \vec{b}$ is written as $A|x\rangle = |b\rangle$, where A is Hermitian. The matrix A and the states $|x\rangle$ and $|b\rangle$ can be expanded in terms of the eigenstates of A as

$$A = \sum_{j=1}^{N} \lambda_j \left|u_j\right\rangle \left\langle u_j\right|$$

$$A^{-1} = \sum_{j=1}^{N} \lambda_j^{-1} \left|u_j\right\rangle \left\langle u_j\right|$$

$$\left|b\right\rangle = \sum_{j=1}^{N} \beta_j \left|u_j\right\rangle$$

$$\text{where } \beta_j = \left\langle u_j | b\right\rangle$$

$$\text{and } \left|x\right\rangle \propto A^{-1}\left|b\right\rangle = \left(\sum_{k=1}^{N} \lambda_k^{-1} \left|u_k\right\rangle \left\langle u_k\right|\right) \left(\sum_{j=1}^{N} \beta_j \left|u_j\right\rangle\right) = \sum_{j=1}^{N} \frac{\beta_j}{\lambda_j} \left|u_j\right\rangle$$

where λ_j and $\left|u_j\right\rangle$ are the eigenvalues and eigenstates of A. The algorithm to solve this equation is described in [18]. If the matrix A is not Hermitian, we require

$$\begin{bmatrix} 0 & A \\ A^{\dagger} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \vec{x} \end{bmatrix} = \begin{bmatrix} \vec{b} \\ 0 \end{bmatrix}$$

Hence, it implies a bipartite graph just as the Szegedy walk. This shows that Szegedy walk represents an advantage in this type of problems.

The task of efficiently implementing a given quantum walk is in general a difficult problem. We have provided a methodology for implementing a large scale quantum circuit, using a Szegedy's walk for the quantum PageRank. To make it possible, we employed a quantum computing platform that allowed us to use the most basic building blocks for constructing bigger operations on qubits. Overall, Qiskit is a very intuitive platform to perform quantum computing. As it is developed we will see new capabilities in the near future. For our purposes, we used the classical simulator since we are restricted by the topology of the quantum processor. As an example of connectivity restriction, consider a 5 qubit quantum processor where the connection in two-qubit gates is only possible between neighboring qubits (see Fig. 8.2).

From the implementation procedure, we can observe that a good circuit design does not imply an efficient implementation (with the available technology). Requiring up to $N-1$ ancilla qubits just to perform on N control qubits may represent a higher cost to pay. Truthfully it is. Qubits are a scarse resource, hence making this general strategy a resource intensive to be practical in most cases. This is a problem to keep in mind.

In discrete time quantum walks, such as the coined model and Szegedy model, the shift operator must be Hermitian and the coin must be an orthogonal reflection, which

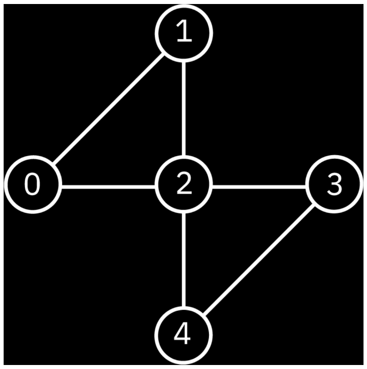


Figure 8.2: IBM Q 5 Tenerife.

is a unitary and Hermitian operator. The class of orthogonal reflections includes the Grover and Hadamard coins [48]. As a matter of fact, the Szegedy's scheme is considered a generalization of the Grover algorithm. Let us consider the Grover diffusion operator, $U_s = 2\left|s\right\rangle\left\langle s\right| - I$. Now, consider the Szegedy quantum walk operator, $W = (2AA^* - I)(2BB^* - I)$. In the Grover algorithm, we perform multiple 'Grover iterations' followed by a single measurement at the end [29]. Here, one Grover iteration is equivalent to a reflection about the average amplitude. In Szegedy, the operator $D' = I - 2\left|b\right\rangle\left\langle b\right|$ is the same reflection operation as in Grover. The key idea here, is the reflection operation. Geometrically a reflection involves the reflected object and the point or line where is reflected about. For example, consider Fig. 8.3, which illustrates a reflection of state $\left|\psi\right\rangle$ about the point of reflection which is the state $\left|0\right\rangle$. Mathematically, this reflection is equivalent to a sign flip of the $\left|1\right\rangle$ state. We perform this reflection using the Z gate illustrated in the circuit Fig. 6.3 as the D operator and correctly implemented in Sec. 7 about simulations. Reflecting the $\left|\psi\right\rangle_i$ about the state $\left|1\right\rangle$ will flip the sign of state $\left|0\right\rangle$ as in Fig. 8.4.

The interpretation of Grover's algorithm is that of a quantum walk on the edges of a complete graph [54] (see definition of complete graph in Sec. 2 and Fig. 2.6). Mathematically, $\mathcal{S}$ and $\mathcal{R}$ correspond to the shifting and coin operator of the coined model, respectively. Grover's work was an important factor in preparing the way for the quantum computing revolution that is still ongoing today since it was able to prove faster than its classical counterpart. As for the Grover and Szegedy quantum walk implementation procedure has taken time due to the technical challenges involved. As far as we know, there is not an implementation procedure for the Szegedy quantum walk with the existing technology. The first scalable version of a quantum computer did not appear until 2017. Recent research [22] has shown that Grover search algorithm is a natural occurring phenomenon, where electrons behave like a Grover search. The implications are profound. In the same

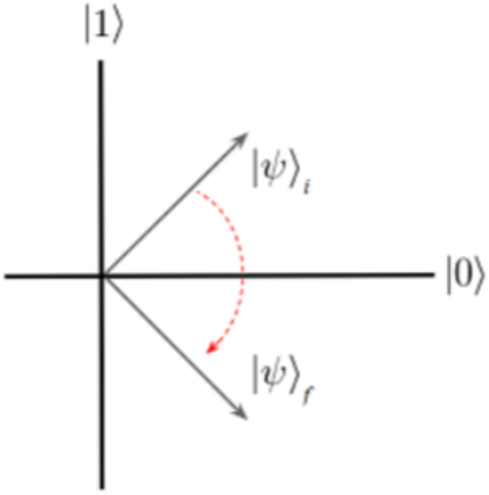


Figure 8.3: Reflection about state $|0\rangle$.

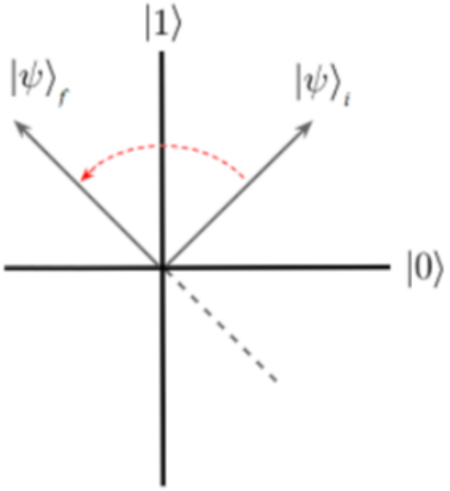


Figure 8.4: Reflection about state $|1\rangle$.

research, Grover algorithm is formulated as a quantum walk across a surface, in a two dimensional space [22].

Szegedy's quantum walk represent a good formalism to the problem of making a quantum walk on nonregular directed graphs. A discrete time quantum walk for the graph in Fig. 2.5, would require to define a coin operator as

$$\hat{C'} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C & 0 & 0 \\ 0 & 0 & H & 0 \\ 0 & 0 & 0 & H \end{bmatrix} \tag{8.2}$$

where $\hat{H}$ is the Hadamard gate and $\hat{C}$ (3-degree) is

$$\hat{C} = \frac{1}{3}\begin{bmatrix} -1 & 2 & 2 \\ 2 & -1 & 2 \\ 2 & 2 & -1 \end{bmatrix} \tag{8.3}$$

and for bigger node degrees we would require to replace each vertex with a coin of dimension d where d is the degree of that vertex. This strategy would require a large amount of extra qubits making it inefficient. In Szegedy quantum walk, we use a Hilbert space $\mathcal{H}_A \otimes \mathcal{H}_B$ which doubles the state space. In addition to double the state space of the Szegedy walk, our circuit requires $N-1$ ancillary qubits. There are several algorithms that implement a generic N-controlled gate. We distinguish between 3 distinct classes. The algorithms requiring $N-1$ ancilla qubits as used in this work. These algorithms have the best gate-count and circuit depth complexity. The algorithms requiring only 1 ancilla qubit, at the expense of a bigger gate-count and circuit-depth. Hybrid algorithms between the 2 previous classes: they need more than one ancilla qubit but less than $N-1$ and also have a gate complexity between the 2 previous classes. Our implementation of ancilla qubits shows a good strategy to follow.

The implementation of $U(t)$ is generated by repeating t times the circuit U_{walk}. In fact, this walk is carried out using a for loop which repeats the circuit t times and allows to propagate the amplitudes. We observed the desired results at step 7, which in fact is a cyclic permutation that includes all states.

The main drawback of Szegedy walk lies in the implementation of the reflection operator $\mathcal{R}$. This operator contains the adjacency information of the graph. The swap operator $\mathcal{S}$ is easily implemented. More research on diagonalizing and decomposing the operators must be carried on. The best known available techniques include QR decomposition and the sine-cosine decomposition [14] [38].

Chapter 9

Conclusions

Quantum computing is advancing at a rapid pace. In quantum computing, we start with a qubit, the basic unit of information. Thanks to its vectorial nature, a qubit can represent the vales 0 and 1 in the same equation. Fifty qubits can represent over one quadrillion values simultaneously and advanced simulation capabilities will be used to advance the development of quantum hardware [46]. Furthermore, quantum supremacy is in debate and considered as a real possibility. Whether quantum or classical, networks are a central discussion in the analysis of information. In this thesis, we have focused on an implementation procedure of a quantum PageRank algorithm using a quantum circuit formulation of the original quantum walk-based PageRank algorithm. Furthermore, we have implemented this quantum circuit-based PageRank algorithm on the IBM Q platform, a process in which we have focused on efficient implementation given the constraints of this platform. We aim to understand what this type of algorithm entails in a quantum computing platform. In order to provide a solid background to our studies we introduced several chapters in the field of graph theory, quantum mechanics, discrete quantum walks and a platform for quantum computation, Qiskit.

In our chapter on *Graph Theory* we set the basis to understand the dynamics of quantum walks. From this chapter we know that a quantum walk takes place on different types of graph. The network topology, such as a scale free network simulates the structure of the web. The web network is represented as a directed graph where nodes are pages and hyperlinks are represented by directed edges. We relate the position of the walker in the network as a basis state in the quantum formalism, such as $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ for a web of four nodes.

In chapter 3, *A concise introduction to Quantum Mechanics*, we discuss the postulates of quantum mechanics which form the basis of quantum computation. As previously stated, this chapter is written keeping in mind practitioners of other areas interested in quantum computing. Is important to mention that a solid background in linear algebra facilitates the understanding of the concepts described. From postulate one we know that Hilbert space is the state space of the system described by unit vectors. We may consider a system of n qubits with the computational basis states of the form $|x_1x_2\dots x_n\rangle$. A quantum state of such state is specified by 2^n amplitudes. For $n = 500$, this number is greater than the estimated number of atoms in the universe [41]. This is an enormous computational capability that computer scientists want to take advantage of.

To solve our problem of ranking, we have focused in the study of discrete time quantum walks which comprises the coined quantum walk and the Szegedy quantum walk. In chapter 4 about Quantum Walks, we describe the use of quantum walks as a mathematical tool for building quantum algorithms. The quantum walk constitutes a universal model of quantum computing. The quantum walk dynamics are driven by the unitary operator with no intermediary measurement. In the case of the coined quantum walk this operator is $\hat{U} = \hat{S} \otimes \hat{C}$. For the Szegedy quantum walk we have an operator defined by $W = (2AA^* - I)(2BB^* - I)$. As explained in Sec. 4.2, recent research shows that the Szegedy and the coined model have similarities that lead to the conclusion that the discrete time quantum walk and Szegedy's quantum walk are equivalent. Until now, this equivalence holds for quantum walks using the Grover and Hadamard coins.

A concise introduction to QISKIT is an introductory chapter to the computational platform in which we simulate the quantum PageRank circuit. We describe the workflow of constructing a quantum algorithms on this platform. We also provide a description of higher order control gates, i.e, gates controlled by n numbers of controls. For this task, we make use of ancilla qubits which allow to temporarily store information of the control qubit, and ultimately determine whether or not to apply the operation. The platform does not allow to implement a higher order controlled gate directly so we have to construct it using ancilla qubits. Our proposal consist in controlling individually each gate. The assumption is that controlling a composed-quantum gate is equivalent to controlling individually all the gates composing it. In Sec. 5.4, we discuss the circuit implementation of quantum walks, relevant to our simulation study.

From section 6, we know that a key idea of Google's Pagerank algorithm is the importance score (Eq. 6.1) and is obtained from the hyperlink matrix. The modified or 'patched' Google matrix is a direct representation of a Markov chain. The connection between Markov chain and random walks is notorious as explained. We define the terminology used in Markov chains. Also, we introduce the Szegedy circuit for the PageRank algorithm and describe the assumptions behind the quantum PageRank algorithm.

We have presented a comprehensive analysis and implementation of a quantum algorithm with the Szegedy quantum walk formalism. The biggest advantage of Szegedy's quantum walk lies in its ability to define a quantum walk on directed graphs at expense of another state space. The implementation procedure for quantum walks is developed using the quantum circuit model of computation taking advantage of a quantum platform provided by IBM. The results are given in Sec. 7 providing the simulations and operations performed. In particular, we were able to use a classical simulator, to obtain measurements of a Szegedy quantum walk. We were able to implement correctly controlled operations. Also, we implemented controlled cyclic permutations along with controlled gates composed of Hadamard and R_y rotations. We provided for those gates its dagger counterpart. We proposed an implementation of a reflection operator D. We were able to take full advantage of the capabilities of the Qiskit platform. As a result, we obtained the expected values reported in the literature.

Once the quantum circuit has been created we ran it in the available backends. As we can see, the qasm_simulator is a classical simulator. Hence the question of why we use a classical simulator to simulate quantum computers. The best notion to this issue is that quantum computing time is a scarce resource and we want to test as many times as possible to confirm that things work out. Also, we have a limitation in the current number of qubits and the connections between qubits in the available quantum processors are restricted.

Chapter 8, entitled *Discussion*, is our last chapter of our contribution. We present the main issues faced in the development of this thesis. We demonstrate the limitations of the research and the implications of the findings. Our work in this field has been inspired not only in how quantum mechanics can help in developing better algorithms but also to facilitate research on the most important open issues facing quantum computing today. Our future research in this direction would be to identify other classes of graphs in which the same formalism could apply. Also, to exploit the scheme provided here to apply a discrete quantum walk in Markov chains applications in Finance, Chemistry and Optimization problems. We also consider that a dynamic entity such as the WWW can be analyzed with dynamics quantum walks [25, 63]. Finally, this research is meant to be situated at an intersection of fields. Certainly, a central goal in the quantum computing community is to make easy for everyone to use quantum computers. This research will accomplish its mission if it encourages scientists in the intersection of fields to further advance their careers in this discipline.

Appendix A

Appendix

In this Appendix we include the code and circuit diagrams employed in the implementation of our quantum circuit. We start by writing the functions that define the operations. Then, we provide the circuits for each operation drawed by Qiskit.

```
#definitions

#t1,y
def t1y(qc, q0, q1, anc):
    qc.x(q0[0])
    qc.x(q1[1])
    qc.ccx(q0[0], q0[2], anc[0])
    qc.ccx(anc[0], q1[1], anc[1])
    qc.cx(anc[1], q1[2])
    qc.cx(anc[0], q1[1])
    qc.ccx(anc[0], q1[1], anc[1])
    qc.ccx(q0[0], q0[2], anc[0])
    qc.x(q1[1])
    qc.ccx(q0[0], q0[1], q1[1])
    qc.x(q0[0])
    return t1y

#t2y
def t2y(qc, q0, q1, anc):
    qc.x(q0[1])
    qc.ccx(q0[0], q0[1], anc[0])
    qc.ccx(q0[2], anc[0], anc[1])
    qc.cx(anc[1], q1[1])
    qc.ccx(q0[2], anc[0], anc[1])
    qc.ccx(q0[0], q0[1], anc[0])
    qc.x(q0[1])
    return t2y

#Angles
alpha = 0.85
beta = (1 - alpha)/8
gamma1 = alpha + beta
gamma2 = alpha/2 + beta
gamma3 = alpha/4 + beta

angle11 = np.arccos(np.sqrt((3*beta+gamma1)/(7*beta+gamma1)))
angle12 = np.arccos(np.sqrt((beta+gamma1)/(3*beta+gamma1)))
angle13 = np.arccos(np.sqrt(beta/(beta + gamma1)))
angle21 = np.arccos(np.sqrt((beta+gamma2)/(3*beta+gamma2)))
angle22 = np.arccos(np.sqrt(gamma2/(beta+gamma2)))
angle3 = np.arccos(np.sqrt(beta/(beta+gamma3)))

def kb1dag(qc, q0, q1, anc):
    qc.x(q0[0])
    qc.x(q1[0])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.ccx(anc[0], q1[1], anc[1])
    qc.ch(anc[1], q1[2])
    qc.ccx(anc[0], q1[1], anc[1])
    qc.x(q1[1])
    qc.ccx(anc[0], q1[1], anc[1])
    qc.cu3(-angle13, 0, 0, anc[1], q1[2])
    qc.ccx(anc[0], q1[1], anc[1])
    qc.x(q1[1])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.x(q1[0])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.ch(anc[0], q1[2])
    qc.ch(anc[0], q1[1])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.x(q1[0])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.cu3(-angle12, 0, 0, anc[0], q1[1])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.x(q1[0])
    qc.cu3(-angle11, 0, 0, q0[0], q1[0])
    qc.x(q0[0])
    return kb1dag

print('angle 11:',angle11)
print('angle 12:', angle12)
print('angle 13:', angle13)

#K_b2 dagger
def kb2dag(qc, q0, q1, anc):
    qc.x(q0[1])
    qc.ccx(q0[0], q0[1], anc[0])
    qc.x(q1[0])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.ch(anc[1], q1[2])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.x(q1[0])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.ch(anc[1], q1[2])
    qc.ch(anc[1], q1[1])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.x(q1[0])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.cu3(-angle22,0,0, anc[1], q1[1])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.x(q1[0])
    qc.cu3(-angle21, 0,0, anc[0], q1[0])
    qc.ccx(q0[0], q0[1], anc[0])
    qc.x(q0[1])
    return kb2dag

print('angle 21:', angle21)
print('angle 22:', angle22)

#k_b3 dagger
def kb3dag(qc, q0, q1, anc):
    qc.ccx(q0[0], q0[1], anc[0])
    qc.ch(anc[0], q1[2])
    qc.ch(anc[0], q1[1])
    qc.cu3(-angle3, 0 ,0, anc[0], q1[0])
    qc.ccx(q0[0], q0[1], anc[0])
    return kb3dag
print('angle 3:',angle3)

#Controlled Paulli z
def paulliz(qc, q0, q1, anc):
    qc.x(q1[0])
    qc.x(q1[1])
    qc.ccx(q1[0], q1[1], q1[2])
    qc.z(q1[2])
    qc.x(q1[1])
    qc.x(q1[0])
    return paulliz

def kb3(qc, q0, q1, anc):
    qc.ccx(q0[0], q0[1], anc[0])
    qc.cu3(angle3, 0 ,0, anc[0], q1[0])
    qc.ch(anc[0], q1[1])
    qc.ch(anc[0], q1[2])
    qc.ccx(q0[0], q0[1], anc[0])
    return kb3

def kb2(qc, q0, q1, anc):
    qc.x(q0[1])
    qc.ccx(q0[0], q0[1], anc[0])
    qc.cu3(angle21, 0,0, anc[0], q1[0])
    qc.x(q1[0])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.cu3(angle22,0,0, anc[1], q1[1])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.x(q1[0])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.ch(anc[1], q1[1])
    qc.ch(anc[1], q1[2])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.x(q1[0])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.ch(anc[1], q1[2])
    qc.ccx(anc[0], q1[0], anc[1])
    qc.x(q1[0])
    qc.ccx(q0[0], q0[1], anc[0])
    qc.x(q0[1])
    return kb2

def kb1(qc, q0, q1, anc):
    qc.x(q0[0])
    qc.cu3(angle11, 0, 0, q0[0], q1[0])
    qc.x(q1[0])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.cu3(angle12, 0, 0, anc[0], q1[1])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.x(q1[0])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.ch(anc[0], q1[1])
    qc.ch(anc[0], q1[2])
    qc.ccx(q0[0], q1[0], anc[0])
    #Ry
    qc.x(q1[0])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.x(q1[1])
    qc.ccx(anc[0], q1[1], anc[1])
    qc.cu3(angle13, 0, 0, anc[1], q1[2])
    qc.ccx(anc[0], q1[1], anc[1])
    qc.x(q1[1])
    qc.ccx(anc[0], q1[1], anc[1])
    qc.ch(anc[1], q1[2])
    qc.ccx(anc[0], q1[1], anc[1])
    qc.ccx(q0[0], q1[0], anc[0])
    qc.x(q1[0])
    qc.x(q0[0])
    return kb1

def t2ydag(qc, q0, q1, anc):
    qc.x(q0[1])
    qc.ccx(q0[0], q0[1], anc[0])
    qc.ccx(q0[2], anc[0], anc[1])
    qc.cx(anc[1], q1[1])
    qc.ccx(q0[2], anc[0], anc[1])
    qc.ccx(q0[0], q0[1], anc[0])
    qc.x(q0[1])
    return t2ydag

def t1ydag(qc, q0, q1, anc):
    qc.x(q0[0])
    qc.ccx(q0[0], q0[1], q1[1])
    qc.ccx(q0[0], q0[2], anc[0])
    qc.ccx(anc[0], q1[2], anc[1])
    qc.cx(anc[1], q1[1])
    qc.ccx(anc[0], q1[2], anc[1])
    qc.cx(anc[0], q1[2])
    qc.ccx(q0[0], q0[2], anc[0])
    qc.x(q0[0])
    return t1ydag

def swap(qc, q0, q1):
    qc.swap(q0[2], q1[2])
    qc.swap(q0[1], q1[1])
    qc.swap(q0[0], q1[0])
```

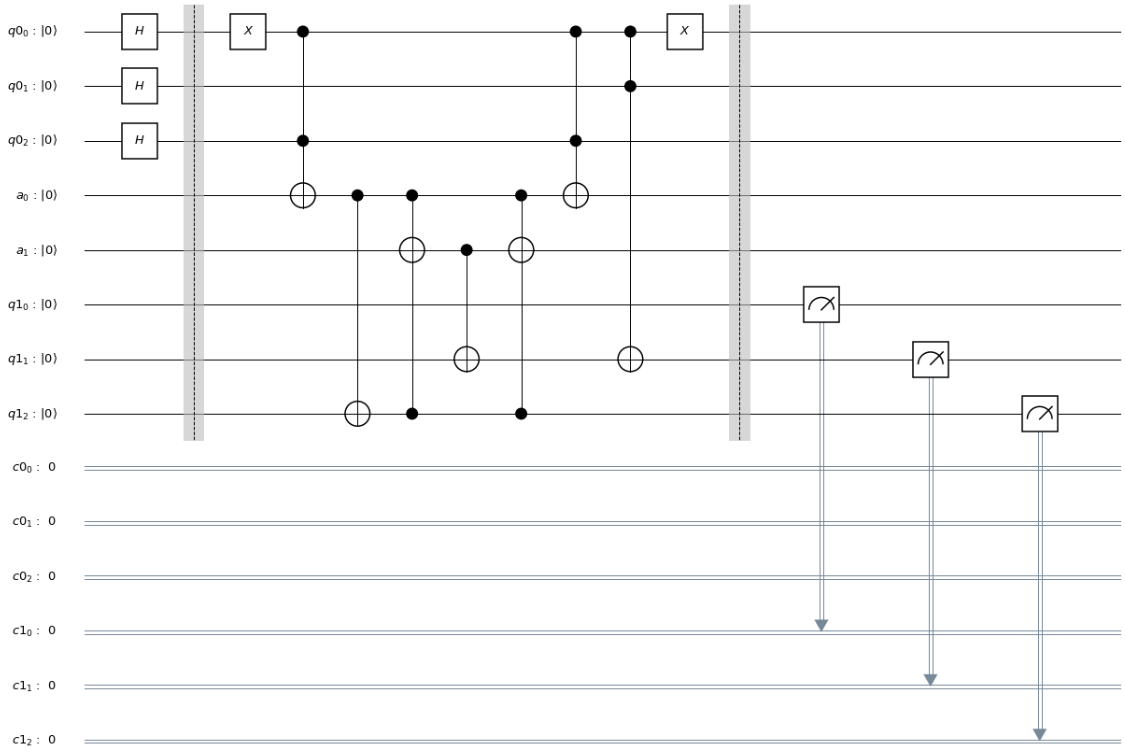


Figure A.1: Circuit corresponding to the $T_{1,y}$ transformation.

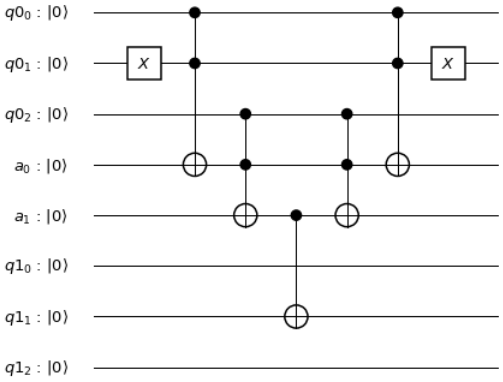


Figure A.2: Circuit corresponding to $T_{2,y}$.

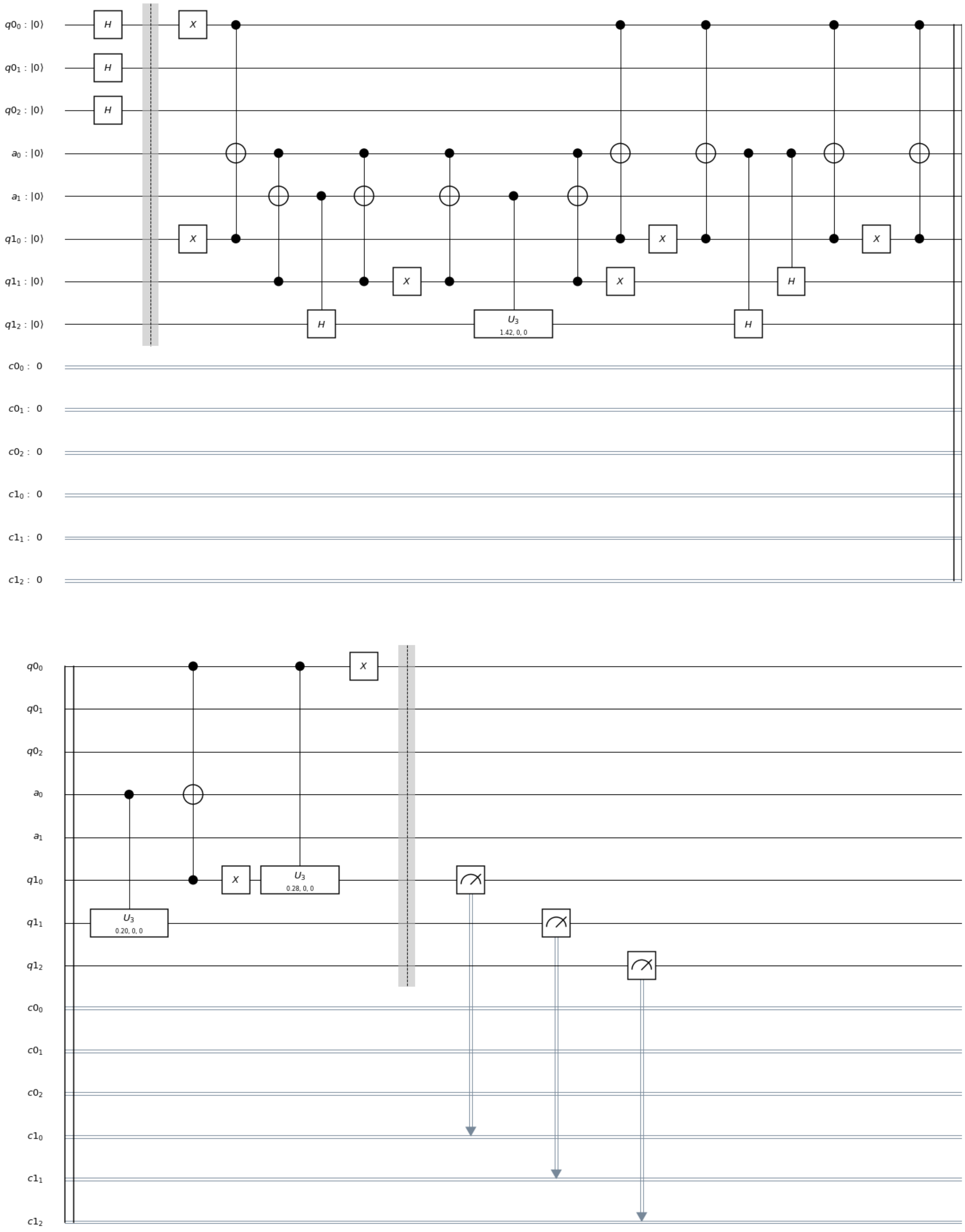


Figure A.3: $K_{b_1}^{\dagger}$ circuit.

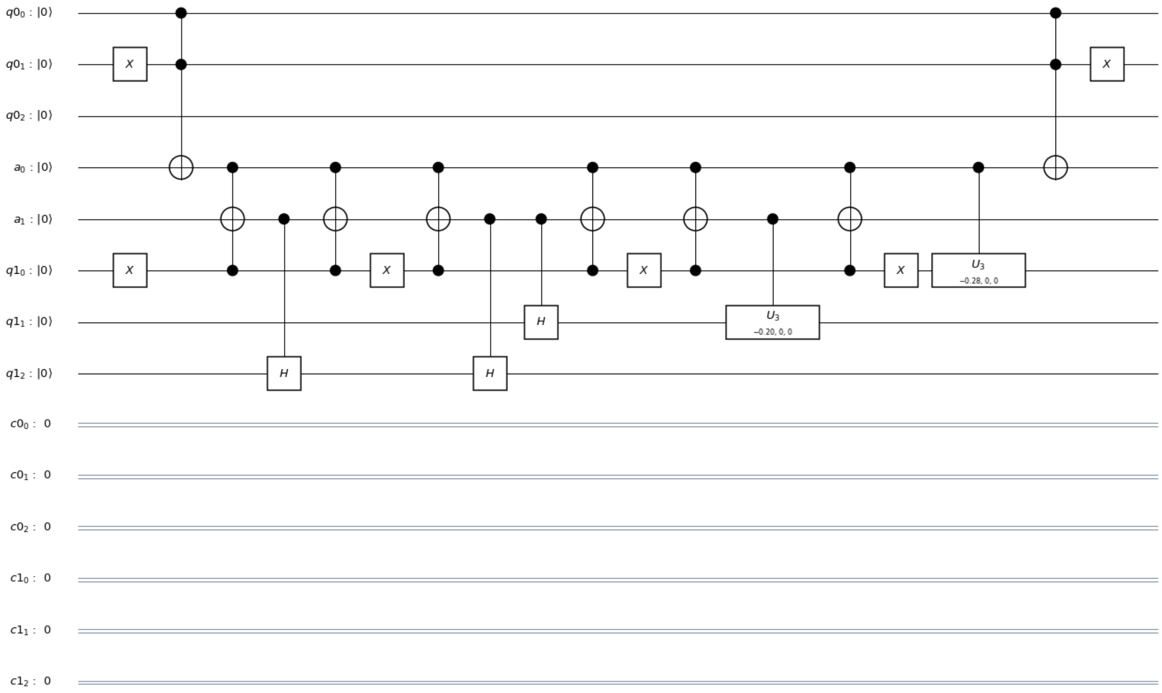


Figure A.4: Circuit implementing $K_{b_2}^{\dagger}$.

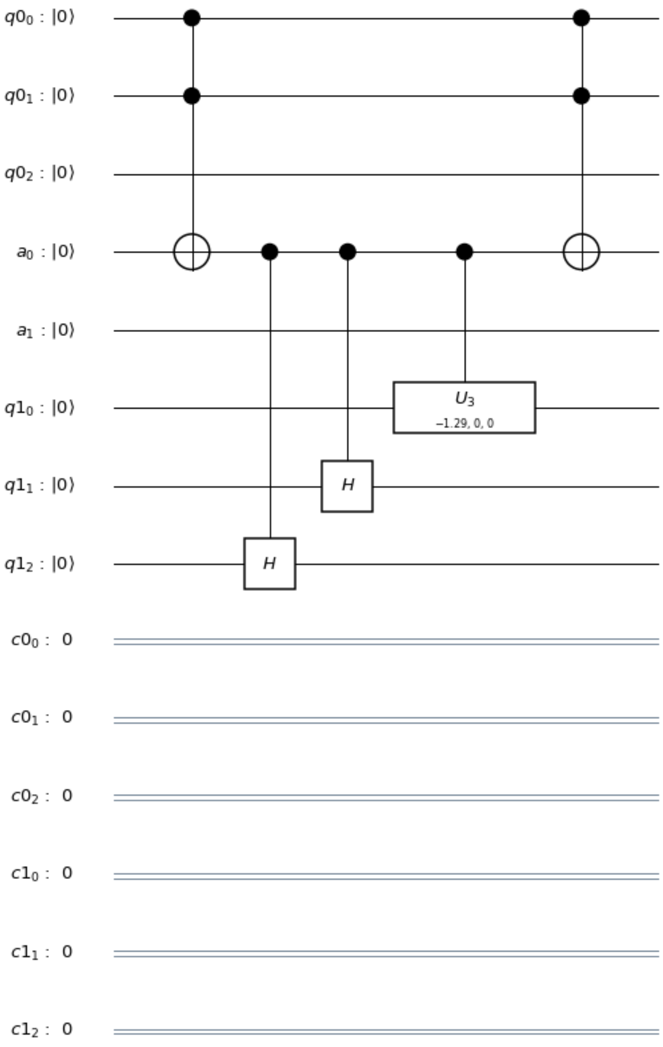


Figure A.5: Circuit implementing K_{b_3}.

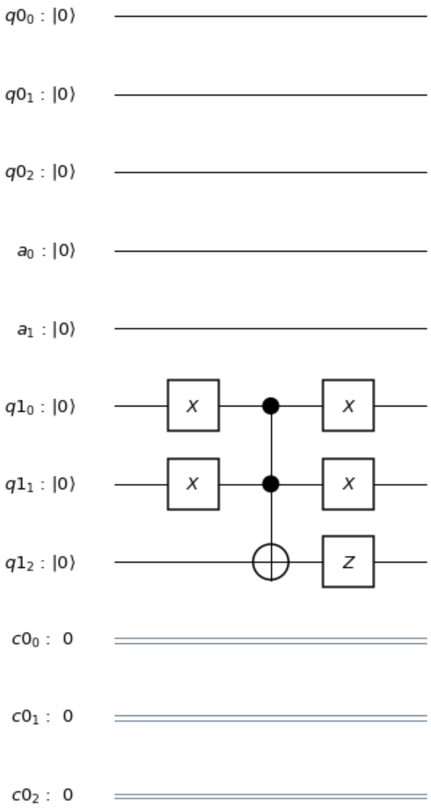


Figure A.6: Circuit implementation for Pauli-z operator.

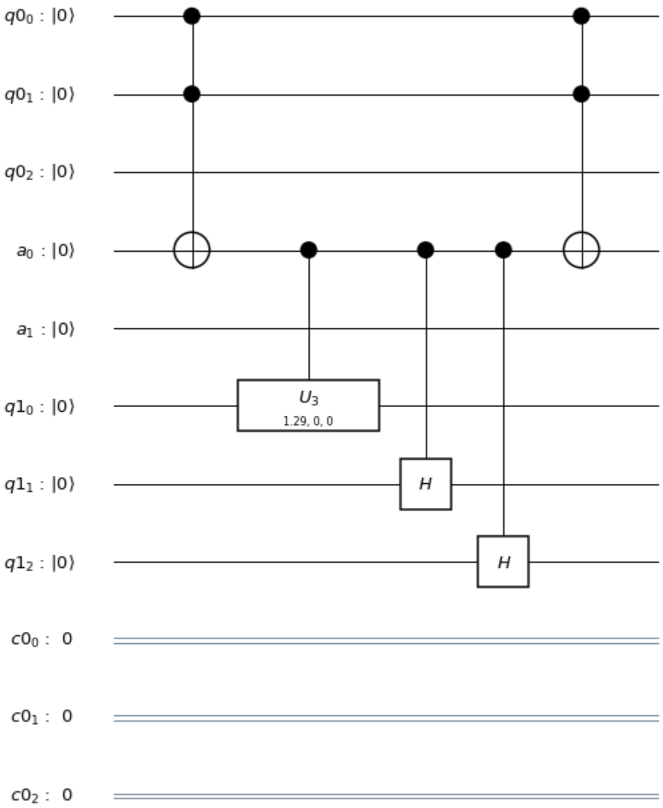


Figure A.7: K_{b_3} circuit.

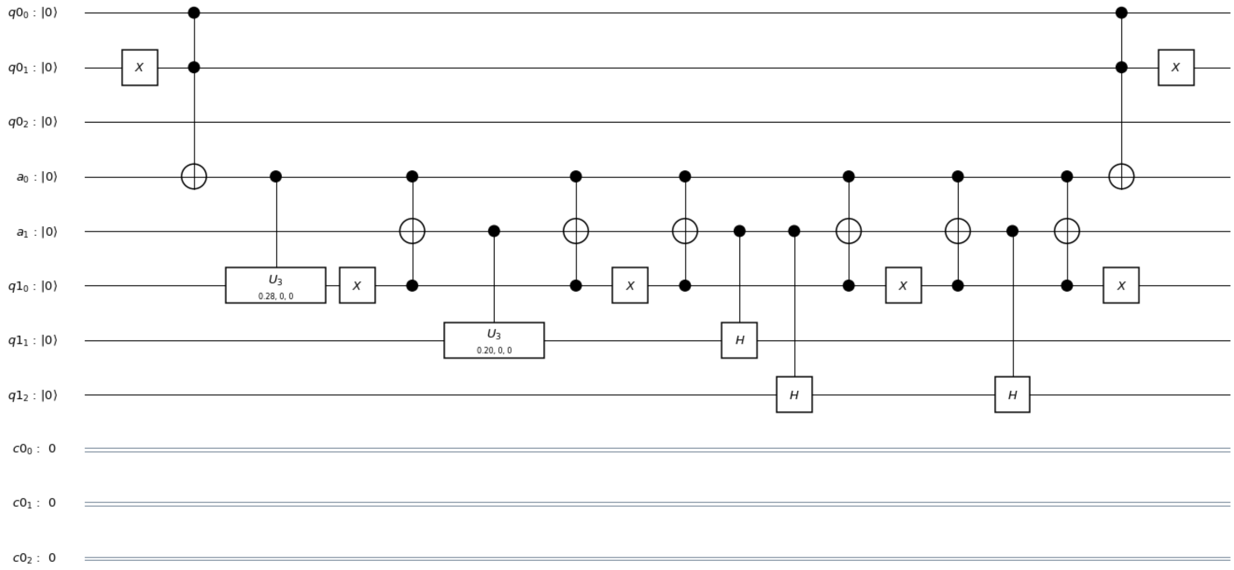


Figure A.8: K_{b_2} circuit.

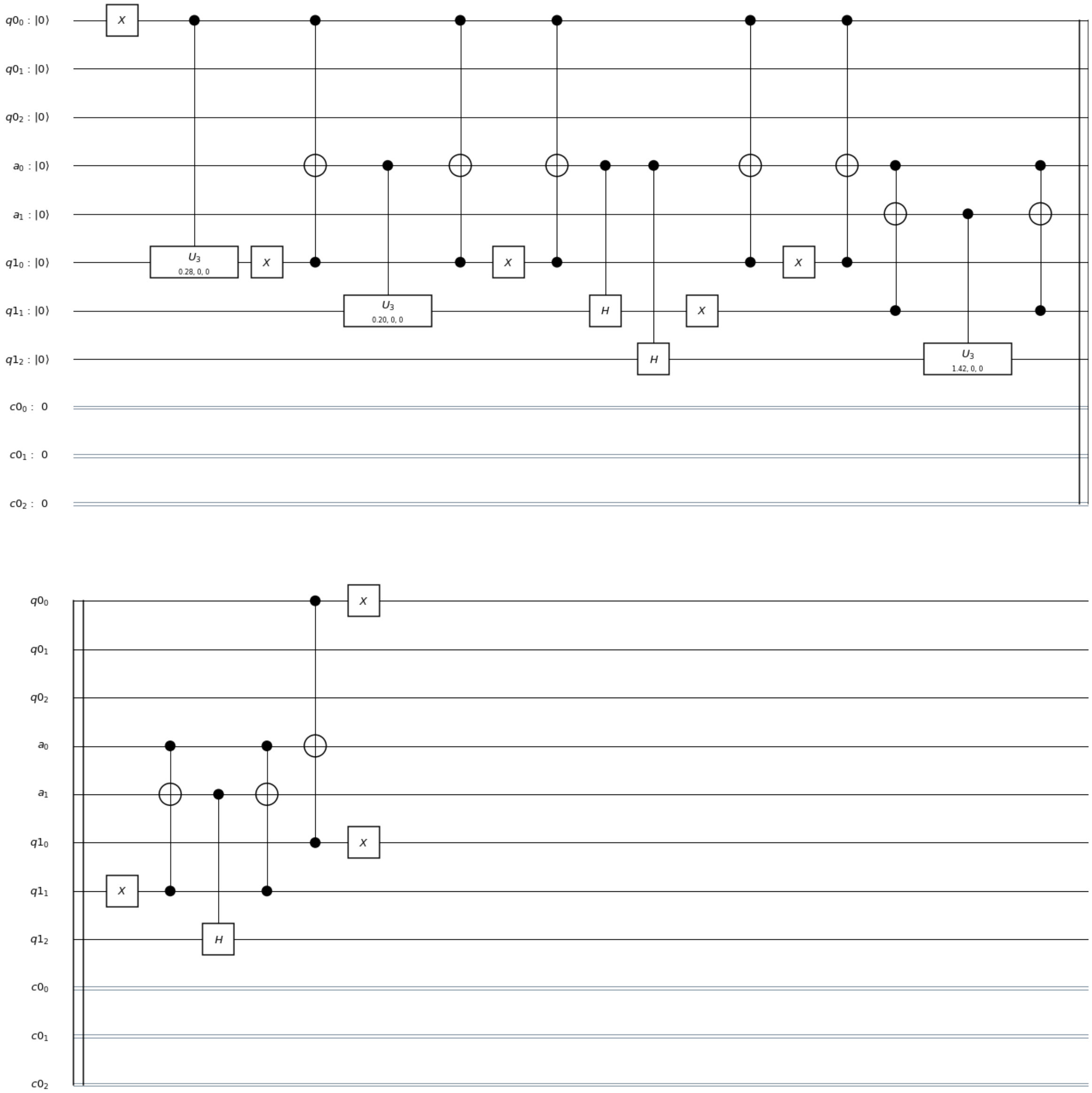


Figure A.9: K_{b_1} circuit-

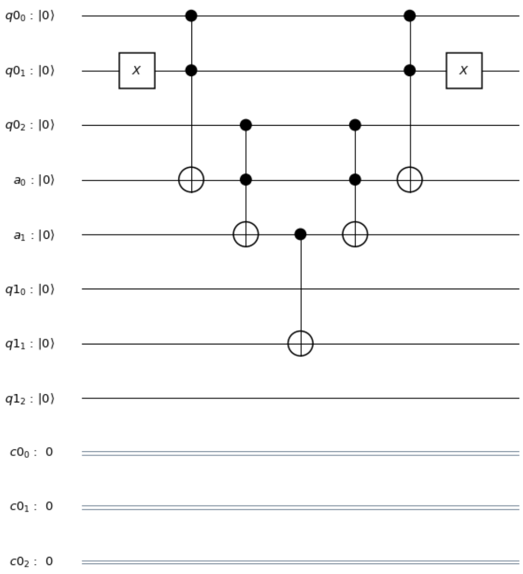


Figure A.10: $T_{2y}^{\dagger}$ circuit.

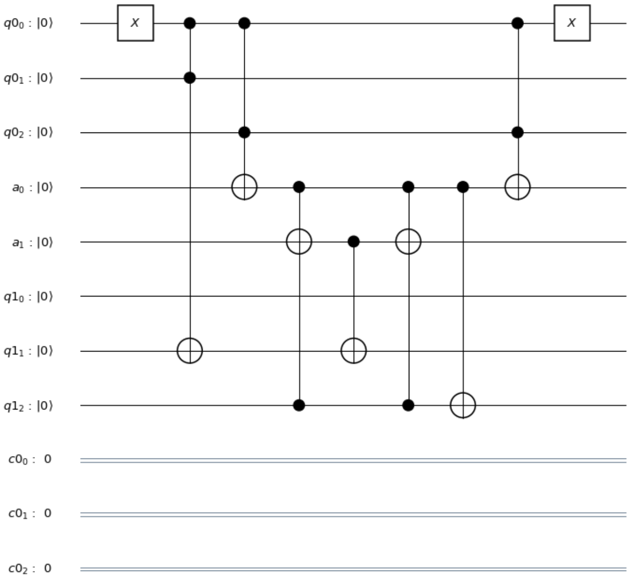


Figure A.11: $T_{1y}^{\dagger}$ circuit.

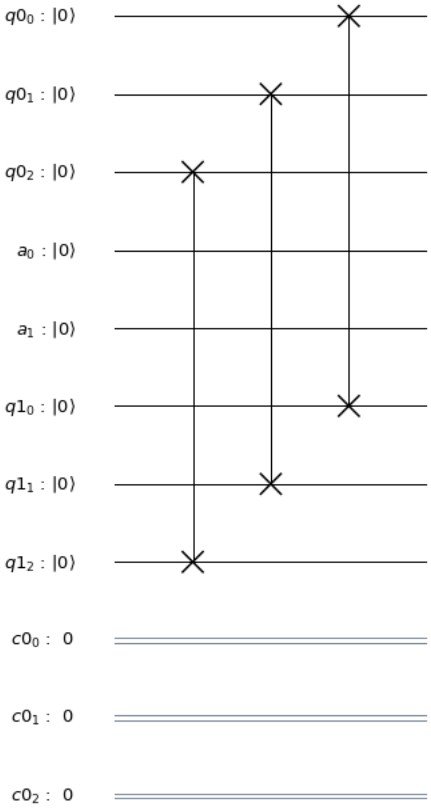


Figure A.12: Swap implementation.

Bibliography

[1] Aharonov, D., Ambainis, A., Kempe, J., and Vazirani, U. Quantum walks on graphs. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing* (2001), ACM, pp. 50–59.

[2] Aharonov, Y., Davidovich, L., and Zagury, N. Quantum random walks. *Phys. Rev. A 48* (Aug 1993), 1687–1690.

[3] Albert, R., Jeong, H., and Barabási, A.-L. Internet: Diameter of the world-wide web. *nature 401*, 6749 (1999), 130.

[4] Aleksandrowicz, G., Alexander, T., Barkoutsos, P., et al. Qiskit: An open-source framework for quantum computing, 2019.

[5] Bang-Jensen, J., and Gutin, G. Z. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

[6] Barabási, A.-L., and Bonabeau, E. Scale-free networks. *Scientific american 288*, 5 (2003), 60–69.

[7] Barabási, A.-L., et al. *Network science*. Cambridge university press, 2016.

[8] Biamonte, J., Faccin, M., and De Domenico, M. Complex networks: from classical to quantum. *arXiv preprint arXiv:1702.08459* (2017).

[9] Bollobás, B. *Modern graph theory*, vol. 184. Springer Science & Business Media, 2013.

[10] Brin, S., and Page, L. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems 30*, 1-7 (1998), 107–117.

[11] Bryan, K., and Leise, T. The $25,000,000,000 eigenvector: The linear algebra behind google. *SIAM review 48*, 3 (2006), 569–581.

[12] CALDARELLI, G. *Scale-free networks: complex webs in nature and technology*. Oxford University Press, 2007.

[13] CERVERA-LIERTA, A. Exact ising model simulation on a quantum computer. *Quantum 2* (2018), 114.

[14] CHEN, Y., AND WANG, J. Qcompiler: Quantum compilation with the csd method. *Computer Physics Communications 184*, 3 (2013), 853–865.

[15] CHIANG, C.-F., NAGAJ, D., AND WOCJAN, P. Efficient circuits for quantum walks. *arXiv preprint arXiv:0903.3465* (2009).

[16] CHILDS, A. M. Universal computation by quantum walk. *Physical review letters 102*, 18 (2009), 180501.

[17] CHILDS, A. M., CLEVE, R., DEOTTO, E., FARHI, E., GUTMANN, S., AND SPIELMAN, D. A. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing* (2003), ACM, pp. 59–68.

[18] COLES, P. J., EIDENBENZ, S., PAKIN, S., ADEDOYIN, A., AMBROSIANO, J., ANISIMOV, P., CASPER, W., CHENNUPATI, G., COFFRIN, C., DJIDJEV, H., ET AL. Quantum algorithm implementations for beginners. *arXiv:1804.03719* (2018).

[19] CROSS, A. W., BISHOP, L. S., SMOLIN, J. A., AND GAMBETTA, J. M. Open quantum assembly language, 2017.

[20] DOUGLAS, B., AND WANG, J. Efficient quantum circuit implementation of quantum walks. *Physical Review A 79*, 5 (2009), 052335.

[21] GRIFFITHS, D. J. *Introduction to quantum mechanics*. Pearson international edition (Pearson Prentice Hall, 2005), 2010.

[22] GUILLET, S., ROGET, M., ARRIGHI, P., AND MOLFETTA, G. D. The grover search as a naturally occurring phenomenon, 2019.

[23] HALDER, K., HEGADE, N. N., BEHERA, B. K., AND PANIGRAHI, P. K. Digital quantum simulation of laser-pulse induced tunneling mechanism in chemical isomerization reaction, 2018.

[24] HARRIS, J. M., HIRST, J. L., AND MOSSINGHOFF, M. J. *Combinatorics and graph theory*, vol. 2. Springer, 2008.

[25] HERRMAN, R., AND HUMBLE, T. S. Continuous-time quantum walks on dynamic graphs. *Phys. Rev. A 100* (Jul 2019), 012306.

[26] HU, Z., XIA, R., AND KAIS, S. A quantum algorithm for evolving open quantum dynamics on quantum computing devices, 2019.

[27] HUGUES-SALAS, E., NTAVOU, F., GKOUNIS, D., KANELLOS, G. T., NEJABATI, R., AND SIMEONIDOU, D. Monitoring and physical-layer attack mitigation in sdn-controlled quantum key distribution networks. *Journal of Optical Communications and Networking 11*, 2 (2019), A209–A218.

[28] IBM. Ibm q experience. `https://www.research.ibm.com/ibm-q/`, 2019 (accessed July 8, 2019).

[29] KOCH, D., WESSING, L., AND ALSING, P. M. Introduction to coding quantum algorithms: A tutorial series using qiskit. *arXiv preprint arXiv:1903.04359* (2019).

[30] KUMAR, S., SINGH, R. P., BEHERA, B. K., AND PANIGRAHI, P. K. Quantum simulation of negative hydrogen ion using variational quantum eigensolver on ibm quantum computer, 2019.

[31] LANGVILLE, A. N., AND MEYER, C. D. *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2011.

[32] LAROSE, R. Overview and comparison of gate level quantum software platforms. *Quantum 3* (2019), 130.

[33] LOKE, O. *Quantum circuit design for quantum walks*. PhD thesis, The University of Western Australia, 2017.

[34] LOKE, T., TANG, J., RODRIGUEZ, J., SMALL, M., AND WANG, J. B. Comparing classical and quantum pageranks. *Quantum information processing 16*, 1 (2017), 25.

[35] LOKE, T., AND WANG, J. Efficient quantum circuits for szegedy quantum walks. *Annals of Physics 382* (2017), 64–84.

[36] MILLER, D. A. *Quantum mechanics for scientists and engineers*. Cambridge University Press, 2008.

[37] MONTANARO, A. Quantum algorithms: an overview. *npj Quantum Information 2* (2016), 15023.

[38] MOTTONEN, M., AND VARTIAINEN, J. J. Decompositions of general quantum gates. *arXiv preprint quant-ph/0504100* (2005).

[39] NANNICINI, G. An introduction to quantum computing, without the physics. *arXiv:1708.03684* (2017).

[40] NEWMAN, M. E. The structure and function of complex networks. *SIAM review 45*, 2 (2003), 167–256.

[41] NIELSEN, M. A., AND CHUANG, I. Quantum computation and quantum information, 2002.

[42] PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab, 1999.

[43] PAPARO, G. D., AND MARTIN-DELGADO, M. Google in a quantum network. *Scientific reports 2* (2012), 444.

[44] PAPARO, G. D., MÜLLER, M., COMELLAS, F., AND MARTIN-DELGADO, M. A. Quantum google in a complex network. *Scientific reports 3* (2013), 2773.

[45] PEARSON, K. The problem of the random walk. *Nature 72* (1905), 294.

[46] PEDNAULT, E., GUNNELS, J. A., NANNICINI, G., HORESH, L., MAGERLEIN, T., SOLOMONIK, E., DRAEGER, E. W., HOLLAND, E. T., AND WISNIEFF, R. Breaking the 49-qubit barrier in the simulation of quantum circuits, 2017.

[47] PORTUGAL, R. *Quantum walks and search algorithms*. Springer, 2013.

[48] PORTUGAL, R. Establishing the equivalence between szegedy's and coined quantum walks using the staggered model. *Quantum Information Processing 15*, 4 (2016), 1387–1409.

[49] PRESKILL, J. Quantum computing in the nisq era and beyond. *Quantum 2* (2018), 79.

[50] QIANG, X., LOKE, T., MONTANARO, A., AUNGSKUNSIRI, K., ZHOU, X., O'BRIEN, J. L., WANG, J. B., AND MATTHEWS, J. C. Efficient quantum walk on a quantum processor. *Nature communications 7* (2016), 11511.

[51] RUDNICK, J., AND GASPARI, G. *Elements of the Random Walk: An introduction for Advanced Students and Researchers*. Cambridge University Press, 2010.

[52] STAMATOPOULOS, N., EGGER, D. J., SUN, Y., ZOUFAL, C., ITEN, R., SHEN, N., AND WOERNER, S. Option pricing using quantum computers, 2019.

[53] STRUBELL, E. An introduction to quantum algorithms. *COS498 Chawathe Spring 13* (2011), 19.

[54] SZEGEDY, M. Quantum speed-up of markov chain based algorithms. In *45th Annual IEEE symposium on foundations of computer science* (2004), IEEE, pp. 32–41.

[55] TACCHINO, F., MACCHIAVELLO, C., GERACE, D., AND BAJONI, D. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information 5*, 1 (2019), 26.

[56] VAN METER, R. *Quantum Networking*. Wiley-ISTE, 2014.

[57] VENEGAS-ANDRACA, S. E. Quantum walks: a comprehensive review. *Quantum Information Processing 11*, 5 (2012), 1015–1106.

[58] WEHNER, S., ELKOUSS, D., AND HANSON, R. Quantum internet: A vision for the road ahead. *Science 362*, 6412 (2018).

[59] WHITFIELD, J. D. Reflections in hilbert space ii: Szegedy's scheme for markov chain quantization. *Selected Lectures from WhitfieldâĂŹs homepage* (2012).

[60] WILLE, R., VAN METER, R., AND NAVEH, Y. Ibm's qiskit tool chain: Working with and developing for real quantum computers. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2019), IEEE, pp. 1234–1240.

[61] WOERNER, S., AND EGGER, D. J. Quantum risk analysis. *npj Quantum Information 5*, 1 (2019), 15.

[62] WONG, T. G. Equivalence of szegedy's and coined quantum walks. *Quantum Information Processing 16*, 9 (2017), 215.

[63] WONG, T. G. Isolated vertices in continuous-time quantum walks on dynamic graphs. *arXiv:1908.00507* (2019).

[64] ZOUFAL, C., LUCCHI, A., AND WOERNER, S. Quantum generative adversarial networks for learning and loading random distributions, 2019.

Curriculum Vitae

Master student was born in Tabasco, México. He earned a Nanotechnology and Molecular Engineering degree from Universidad de las Américas, Puebla in June 2013. He was accepted in the graduate program in Computer Science in August 2017.

Academic Qualifications

- August 2017–June 2019. MSc. in Computer Science. Monterrey Institute of Technology and Higher Education, State of México.
- May 2017–July 2017. Propaedeutic in Discrete Mathematics, Automata's theory and Data Structures. National Institute of Astrophysics, Optics and Electronics.
- August 2008–June 2013. BSc. Nanotechnology and Molecular Engineering. University of the Americas Puebla.

Experience

- September 2012–June 2013. Research project: National Institute of Astrophysics, Optics and Electronics, Puebla. Research at the microelectronics laboratory consisted in fabricating thin film transistors using microfabrication techniques such as, chemical vapor deposition, photolitography, etching and ellipsometry.
- January 2012– June 2012. Science Institute at Benemérita Universidad Autónoma de Puebla. Worked in a mass spectrometry laboratory as a mass spectrometer specialist. Conducted spectra analysis of organic and inorganic compounds, using fast atom bombardment (FAB) and electronic impact (EI).

Courses and Personal skills

- **Electives:** Data Analysis, Neural Networks, Quantum Computing, Computer Vision, Machine Learning.
- **Software:** IBM Qiskit, WEKA, TensorFlow, Keras, OpenCV, LaTeX, Tableau.
- Quantum computing 101, IBM Mexico Bootcamp. **Issued on:** 22 Jun 2018.

Figure A.13: https://bit.ly/2FFKqpX

This document was typed in using LaTeX 2_{ε}[a] by German Alamilla Peralta.

---

[a] The style file `phdThesisFormat.sty` used to set up this thesis was prepared by the Center of Intelligent Systems of the Instituto Tecnológico y de Estudios Superiores de Monterrey, Monterrey Campus