



Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Computer Science and Information Theory

Simulation of quantum walks on a classical computer

Scientific Students' Association Report

Author:

Viktória Nemkin

Advisor:

dr. Katalin Friedl

2021

Contents

Kivonat	i
Abstract	ii
1 Introduction	1
2 Classical random walks	3
3 Quantum computing	5
3.1 The postulates of quantum mechanics	5
4 Quantum walks	10
4.1 Formulating the Quantum coin	10
4.1.1 Hadamard coin	10
4.1.2 Grover coin	11
4.1.3 Fourier coin	12
4.2 Quantum walks on the line	12
4.2.1 State space	13
4.2.2 Evolution	13
4.2.3 Measurement	16
4.3 Generalization of Quantum Walks	16
4.3.1 Generalization using multiple independent 2 dimensional coins	16
4.3.2 Generalization using a single higher dimensional coin	20
5 Simulator software	23
5.1 Currently available solutions	23
5.2 Architecture	23
5.3 Language choice	24
5.4 High level design	24
5.4.1 Graph models	25
5.4.2 Simulators	26

5.4.3	Running, configuration and result collection	26
5.4.4	Availability	26
6	Results	27
6.1	Walks on the line	27
6.2	Walks on the grid	29
6.3	Walks on hypercube	34
7	Conclusion	38
	Acknowledgements	39
	Bibliography	40

Kivonat

Az utóbbi években egyre nagyobb figyelem összpontosul a kvantuminformatikára. Olyan globális vállalatok, mint az IBM, a Google, a Microsoft és az Amazon jelentős összegeket fektetnek kutatásba, hardver- és szoftverfejlesztésekbe ezen a területen, míg az Európai Unió és Magyarország számos olyan programot indított, melyek a kvantuminformatikai kutatások fellendítését célozzák meg.

A jelenlegi kvantumszámítógépekben elérhető qubitek (kvantumbitek) mennyisége még csekély, de sokan úgy vélik hogy a jövőben ez a szám növekedni fog. Az első olyan, a gyakorlatban is hasznos kvantumalgoritmusok, amiket ezeken a processzorokon futtatni tudunk majd, várhatóan azok lesznek, melyek takarékosan bánnak a rendelkezésre álló qubitekkel. A kvantumséta, mely a klasszikus véletlen bolyongás általánosítása kvantumos esetben, pontosan ilyen algoritmus. Mivel a qubit igénye a gráf csúcsszámában logaritmikus, így ez egy érdekes módszernek ígérkezik akár a közeljövőre nézve is. A kvantumséták erejét bizonyítja, hogy a Grover keresés (mely több kvantumalgoritmus alapját képezi) is értelmezhető ezek egy speciális fajtájának.

Dolgozatomban leírom a kvantumséták matematikai alapjait, részletezve a megvalósítás szempontjából fontos pontokat, melyek a szakirodalomban kisebb hangsúllyal szerepelnek. Ismertetem az általam írt szimulátor program architektúrális felépítését és működését, továbbá a futtatott szimulációim eredményeit.

A szimulátor programot Python 3 nyelven írtam, a Stratégia tervezési minta alapján. A szakirodalomban tipikusan használt gráfokat beépítetten támogatja, melyek kombinálásával tetszőlegesen bonyolult reguláris gráf előállítható, ez az előállítás képezi a kvantumséta alapját is. A szoftver reguláris gráfokon történő kvantum és klasszikus séták szimulációját teszi lehetővé, az eredményekről pedig egy részletes report fájlt generál. A kvantumos séták esetében a séta tulajdonságai a valószínűségek generálásához felhasznált érmétől is függenek, melyet többféleképpen is lehet definiálni. A program beépítetten tartalmazza az Hadamard-, a Grover- és a Fourier-érméket, de felépítéséből adódóan könnyen bővíthető tetszőleges érmével is.

Szimulációim segítségével összehasonlítottam a klasszikus és a kvantum séták viselkedését, továbbá kimutattam az elméleti szakirodalom alapján elvárt kvantumos jellegzeteségeket, az Hadamard-séta ballisztikus természetét és a kvantumséták ciklikus tulajdonságát.

Abstract

In recent years, there has been an increasing focus on quantum informatics. Influential global companies such as IBM, Google, Microsoft, and Amazon have invested significant amounts into studying and developing hardware and software for this sector, while the European Union and Hungary have launched several programs to accelerate quantum research.

Current technology is yet to produce a significant number of qubits (quantum bits) in a quantum processor, but many believe the amount will increase over the years. The first practical quantum algorithms to be run on these processors are likely to be the ones that use qubits sparingly. Quantum walking, the generalized version of classical random walking, is exactly this kind of algorithm. The number of qubits required to run a quantum walk on a graph is logarithmic in the number of vertices, making it a promising technique for the near future. Furthermore, Grover's search algorithm (a basis for many quantum algorithms) can be viewed as a special case of quantum walks, which illustrates the potential power of this method.

In my dissertation, I present the mathematical framework for quantum walks, detailing the points critical for implementation, which are given less emphasis in the literature. I describe the architecture and capabilities of the simulator program I have written and the conclusions of the simulations I have run.

I developed the software using Python 3, based on the Strategy design pattern. It supports graphs commonly found in the literature while also providing a method for combining them, facilitating experimentation on several kinds of regular graphs. This composition is also the foundation of the quantum walk. It can simulate classical and quantum walks on the same graphs and produce a report file detailing the results. In the quantum case, the characteristics of the walk are also dependent on the type of coin used to generate the probabilities, which can be defined in several ways. The program includes the Hadamard, Grover, and Fourier coins and can easily be extended with others.

Running several simulations, I compared the behavior of classical and quantum walks and demonstrated the quantum characteristics expected from the theoretical literature, the ballistic nature of the Hadamard walk, and the cyclic property of quantum walks.

Chapter 1

Introduction

Classical random walks are well-known tools for describing different stochastic processes. Many real-life scientific approaches rely on these methods, including stock price movement prediction, natural language processing, Brownian motion description; and evolution, population and disease outbreak models. Other algorithms utilize random walks to gain speed or combat the search space's scale, most notably Google's Page Rank algorithm and various recommender systems. [12].

In recent years quantum computing has been gaining traction amongst researchers and computer scientists. While there exists a wide variety of quantum algorithms to be explored, I specifically targeted quantum walks due to their reasonable hardware requirements and promising features. Quantum walks provide a quadratic speedup compared to their classical counterparts and display behaviour, such as the ballistic nature and the cyclic property that their classical equivalent does not, while only requiring logarithmic space, which due to the various limitations of physical qubit realization is particularly valuable. [9]

Since currently, the publicly available quantum computers can operate with only around 5-10 qubits, I created a simulator software that runs on a regular computer to experiment with the algorithm. I am hopeful that the time for feasibly switching to quantum hardware is just around the corner.

During my research, I have observed a significant lack of software engineering perspective on this subject. Most research papers are written by physicists who are well acquainted with the details of quantum mechanics with a heavy focus on physics-related formulas and functional descriptions. It has been a strenuous process to gather the motivation and justification of certain implementation choices that seem to be the standard for someone in the inner circle but strange for me, just getting started with the topic.

This report aims to provide a comprehensible introduction to quantum walking from the software engineer's perspective, spending extra effort on implementation-specific details and mathematical proofs missing from the available literature. In contrast with the functional descriptions, I describe the algorithms using linear algebra, allowing for a more natural way to implement the simulation.

The rest of this report is structured in the following way: In Chapters 2 and 3, I briefly introduce classical random walks and quantum computing, using only the necessary formulas and focusing on the details employed later in the report. In Chapter 4, I introduce quantum walking in a bottom-up approach, starting from the simplest form and then generalizing it. Contrary to many authors, I use linear algebra exclusively to describe

each step since implementation on a universal quantum computer requires the definition to come in the form of unitary transformations.

Section 4.3 discusses two of the generalization techniques found in [9]. I present my improvement to one of these methods, which I have proven to have the equivalent result but remove an exponential memory requirement from the implementation. The other method in [9] uses a constraint about the evolution operator, for which I have not found proof in the literature. Here, I present my more generalized version of this constraint and the proof I have given.

In Chapter 5, I describe the architecture and implementation details of my simulator software, then in Chapter 6, I present the results obtained from my simulation runs.

Chapter 2

Classical random walks

Before introducing quantum computation and specifically quantum walks, I first overview classical random walks, based on the book *Probability and Computing*, written by Michael Mitzenmacher and Eli Upfal [8].

A *random walk* is a stochastic process modeled by a particular type of Markov chain. While a variety of Markov chains exist, in this work, I use the following definition exclusively.

Definition 2.1 (Markov chain). A discrete time stochastic process X_0, X_1, X_2, \dots on a finite state space A is a Markov chain if it has the Markov property:

$$P(X_k = a_k \mid X_{k-1} = a_{k-1}, \dots, X_0 = a_0) = P(X_k = a_k \mid X_{k-1} = a_{k-1}) \quad \forall a_0, \dots, a_k \in A.$$

Without loss of generality, we can assume, that $A = \{0, 1, \dots, n\}$.

If the Markov chain is homogenous (time-invariant), the probability of moving from state $i \in A$ to state $j \in A$ is independent of k , and thus can be shortened the following way:

$$P(X_k = j \mid X_{k-1} = i) = p_{j \leftarrow i} = p_{j,i} \quad \forall k \in \mathbb{Z}^+.$$

Where $p_{j,i}$ is called the *transition probability* between states i and j .

The *transition matrix* \mathbf{P} is formed by the transition probabilities.

$$\mathbf{P}[j, i] = p_{j,i}$$

It follows, that for each column in \mathbf{P} , the sum is 1.

$$\sum_{j=0}^n \mathbf{P}[j, i] = 1 \quad \forall i \in \{0, \dots, n\}$$

Let the *probability distribution* of the process in the t -th step be π_t . Then, π_t can be computed from the starting distribution π_0 using \mathbf{P} .

$$\pi_t = \mathbf{P}^t \pi_0$$

The *stationary distribution* (π) of the Markov chain is a distribution that does not change with a transition, i.e. $\pi = \mathbf{P}\pi$.

Markov chains can be represented using graphs. A directed, weighted graph $G(V, E)$ with weight function $w : E \rightarrow [0, 1]$ represents a Markov chain, if $V = A$ and $w(i, j) = \mathbf{P}[j, i]$. If $\mathbf{P}[j, i] = 0$, then $\{i, j\} \notin E$.

A random walk on graph G starts from $X_0 = a_0$ and visits the vertices of the graph according to the states of the Markov-chain: $X_1 = a_1, X_2 = a_2, \dots$.

Frequently studied characteristics of random walks are *hitting time* [12] and *mixing time* [8]. Informally, hitting time describes how quickly can a vertex be reached from another vertex in the graph, while mixing time expresses how fast the walk reaches the stationary distribution, where the starting vertex can no longer be identified.

Definition 2.2 (Hitting time). Let $h_{j,i}$ be the expected number of steps before node j is visited in a random walk starting from node i . Then, $h_{j,i}$ is given by the following recursive formula:

$$h_{j,i} = \begin{cases} 1 + \sum_{k \in A} p_{j,k} h_{k,i} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

Definition 2.3 (Mixing time). The smallest time index of the Markov chain, where the total variational distance between the current and the stationary distribution is not greater than a given ε . This measure still depends on the starting distribution π_0 , so we take the maximum over all of the possible π_0 distributions.

$$m(\varepsilon) = \max_{\pi_0} \{ \min \{ t : \sum_{j=0}^n |\pi_t[j] - \pi[j]| \leq \varepsilon \} \}$$

Chapter 3

Quantum computing

Algorithms in quantum computing are derived from the postulates of quantum mechanics. These fundamental rules define how a quantum computer operates, and therefore they are essential for any discourse on quantum algorithms.

3.1 The postulates of quantum mechanics

This introduction is based on the following books: Quantum Computing and Communications by Sándor Imre and Ferenc Balázs [2], Quantum Computing by Mika Hirvensalo [5] and Quantum Walks and Search Algorithms by Renato Portugal [9].

Postulate I. State space

The state of an isolated physical system can be described using a unit length *state vector* in a Hilbert space (i.e. complex linear vector space), or *state space*, equipped with an inner product.

Definition 3.1 (Qubit). A state vector in the 2 dimensional Hilbert space (H_2) is a qubit. The base vectors in this space are

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \text{ and } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

A generic qubit is written in the form

$$a|0\rangle + b|1\rangle = \begin{pmatrix} a \\ b \end{pmatrix}$$

where $a, b \in \mathbb{C}$ and $|a|^2 + |b|^2 = 1$.

Definition 3.2 (Superposition). A quantum system is said to be in *superposition*, if its state vector is a linear combination of multiple basis states.

For example $a|0\rangle + b|1\rangle$ is in a superposition of the basis states $|0\rangle$ and $|1\rangle$, with probability amplitudes a and b .

Postulate II. Evolution

The time evolution of an isolated physical system is described using unitary transformation, which depends only on the starting and finishing time of the evolution.

A *quantum algorithm* is a sequence of unitary operators applied to an initial state.

Definition 3.3 (Unitary matrix). \mathbf{U} is unitary if $\mathbf{U}^\dagger = \mathbf{U}^{-1}$ [4].

The following definitions are equivalent:

1. \mathbf{U} 's rows form an orthonormal basis of \mathbb{C}^n .
2. \mathbf{U} 's columns form an orthonormal basis of \mathbb{C}^n .
3. \mathbf{U} is an isometry: it is injective and preserves length.
4. \mathbf{U} preserves the inner product.

Postulate III. Measurement

A quantum measurement is defined by a set of measurement operators $\{\mathbf{M}_m\}$, where m stands for the possible results of the measurement. The probability of measuring m if the system is in state $|v\rangle$ is

$$P(m|v) = \langle v | \mathbf{M}_m^\dagger \mathbf{M}_m | v \rangle.$$

The state of the system after measuring m is then

$$|v'\rangle = \frac{\mathbf{M}_m |v\rangle}{\sqrt{\langle v | \mathbf{M}_m^\dagger \mathbf{M}_m | v \rangle}}.$$

The set of measurement operators have to satisfy the following *completeness relation*:

$$\sum_m \mathbf{M}_m^\dagger \mathbf{M}_m = I,$$

due to

$$1 = \sum_m P(m|v) = \sum_m \langle v | \mathbf{M}_m^\dagger \mathbf{M}_m | v \rangle.$$

Projective measurement

To distinguish a set of orthonormal states $\{|\varphi_m\rangle\}$, the corresponding measurement operators can be produced as $\mathbf{P}_m = |\varphi_m\rangle\langle\varphi_m|$, with the following properties.

Property 3.1 (\mathbf{P}_m is self adjoint).

$$\mathbf{P}_m^\dagger = \mathbf{P}_m$$

Since

$$\mathbf{P}_m^\dagger = (|\varphi_m\rangle\langle\varphi_m|)^\dagger = \langle\varphi_m|^\dagger|\varphi_m\rangle^\dagger = |\varphi_m\rangle\langle\varphi_m| = \mathbf{P}_m.$$

Property 3.2 (\mathbf{P}_m is a projection).

$$\mathbf{P}_m\mathbf{P}_m = \mathbf{P}_m$$

Since

$$\mathbf{P}_m\mathbf{P}_m = (|\varphi_m\rangle\langle\varphi_m|)(|\varphi_m\rangle\langle\varphi_m|) = |\varphi_m\rangle(\langle\varphi_m|\varphi_m\rangle)\langle\varphi_m| = |\varphi_m\rangle 1 \langle\varphi_m| = |\varphi_m\rangle\langle\varphi_m| = \mathbf{P}_m.$$

Property 3.3 (The \mathbf{P}_m are orthogonal).

$$m \neq n \Rightarrow \mathbf{P}_m\mathbf{P}_n = \mathbf{0}$$

Since

$$\mathbf{P}_m\mathbf{P}_n = (|\varphi_m\rangle\langle\varphi_m|)(|\varphi_n\rangle\langle\varphi_n|) = |\varphi_m\rangle(\langle\varphi_m|\varphi_n\rangle)\langle\varphi_n| = |\varphi_m\rangle 0 \langle\varphi_n| = \mathbf{0}.$$

From these properties follows, that the probability of measuring m in case of a projective measurement is

$$P(m|v) = \langle v|\mathbf{P}_m^\dagger\mathbf{P}_m|v\rangle = \langle v|\mathbf{P}_m\mathbf{P}_m|v\rangle = \langle v|\mathbf{P}_m|v\rangle = \langle v|\varphi_m\rangle\langle\varphi_m|v\rangle = |\langle\varphi_m|v\rangle|^2.$$

For example, the value of a qubit can be any unit length vector in H_2 , however when we measure it, we will receive one of the base vectors of H_2 . For $a|0\rangle + b|1\rangle$ we measure 0 with probability $|a|^2$ and 1 with probability $|b|^2$.

Postulate IV. Composite systems

The state space of an isolated composite physical system is the *tensor product* of the state spaces of the individual components. The current state vector of the composite system is the *tensor product* of the current state vectors of the individual systems.

If V_1, \dots, V_n are the state spaces of the individual systems, then $V_1 \otimes \dots \otimes V_n$ is the composite state space and for $|v_1\rangle \in V_1, \dots, |v_n\rangle \in V_n$ state vectors, $|v_1\rangle \otimes \dots \otimes |v_n\rangle = |v_1, \dots, v_n\rangle$ is the state vector of the composite system.

Definition 3.4 (Tensor product). The tensor product $\mathbf{A} \otimes \mathbf{B}$ of matrix $\mathbf{A}_{(r \times s)}$ and matrix $\mathbf{B}_{(t \times u)}$ is of size $(rt \times su)$ and is defined as follows [4]:

$$\text{For } \mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1s} \\ a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & \ddots & a_{rs} \end{pmatrix}, \text{ and } \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1u} \\ b_{21} & b_{22} & \dots & b_{2u} \\ \vdots & \vdots & \ddots & \vdots \\ b_{t1} & b_{t2} & \ddots & b_{tu} \end{pmatrix}$$

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1s}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \dots & a_{2s}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1}\mathbf{B} & a_{r2}\mathbf{B} & \ddots & a_{rs}\mathbf{B} \end{pmatrix}$$

and has the following properties:

Property 3.4 (Associativity).

$$(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$$

Property 3.5 (Mixed product property). If the corresponding matrices are compatible, then

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD}),$$

and as an immediate consequence, we obtain

$$(\mathbf{A} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{B}) = \mathbf{A} \otimes \mathbf{B}.$$

Definition 3.5 (Quantum register). The composite system of n qubits is a *quantum register*, having the composite state space

$$H_2^{\otimes n} = H_2 \otimes H_2 \otimes \dots \otimes H_2$$

and for $|q_{n-1}\rangle \in H_2, \dots, |q_0\rangle \in H_2$ individual state vectors, the composite state vector is

$$|q_{n-1}\rangle \otimes \dots \otimes |q_0\rangle = |q_{n-1}, \dots, q_0\rangle.$$

Definition 3.6 (Entangled state). Any state consisting of multiple qubits, that is not decomposable, i.e. that can not be written in the form of a composite system of qubits is *entangled*.

For example, the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is entangled, since it can not be written in the form

$$(a_0 |0\rangle + a_1 |1\rangle) \otimes (b_0 |0\rangle + b_1 |1\rangle) = a_0 b_0 |00\rangle + a_0 b_1 |01\rangle + a_1 b_0 |10\rangle + a_1 b_1 |11\rangle$$

since that would require from $a_0 b_0 = a_1 b_1 = \frac{1}{\sqrt{2}}$, for all coefficients to be non-zero and from $a_0 b_1 = a_1 b_0 = 0$ for either a_0 or b_1 and either a_1 or b_0 to be zero, which is a contradiction.

Chapter 4

Quantum walks

In classical random walks, the walker moves from the current vertex via one of its outgoing edges, chosen randomly, weighted by the edge weights. This random choice can be interpreted as a (generalized) coin toss.

To formulate a quantum version of graph walking, we define the quantum coin, which will replace the classical concept of randomness with quantum superposition.

4.1 Formulating the Quantum coin

I used Renato Portugal's Quantum Walks and Search Algorithms [9] book as a reference for the different types of coins presented in this section.

A *quantum coin* is a quantum system, which behaves according to the postulates of quantum mechanics. It has a current state, represented by a state vector in a Hilbert space and a unitary time evolution operator, describing a coin toss.

After tossing the coin, the resulting coin state chooses the next step of the quantum walker. If there are d outgoing edges to choose from, then the coin's state space must have d orthonormal basis states, each corresponding to one of the possible edges. If the current state is one of the basis states, then the walker moves in that direction. However, in the quantum world, the coin can also be in a superposition, consisting of multiple basis states. This means that the walker will simultaneously move in all corresponding directions and occupy more than one vertex at the same time, resulting in the walker spreading over the graph in a superposition.

For the d dimensional coin state, the corresponding coin flip operator is a $(d \times d)$ dimensional unitary matrix. Based on what the transition operator is, several types of coins can be defined. The following ones are typically used in quantum walks.

4.1.1 Hadamard coin

The Hadamard coin is the most commonly used quantum coin. It is defined by the Hadamard-matrix as a transition operator:

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

If the starting coin state is $|0\rangle$, then flipping the coin once results in the following state:

$$\mathbf{H}|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

If we measured the above coin, the probability of measuring 0 is

$$P(0 | \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)) = \left| \frac{1}{\sqrt{2}} \right|^2 = \frac{1}{2}.$$

Similarly, if the starting coin state is $|1\rangle$, then flipping the coin once results in the following state:

$$\mathbf{H}|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle.$$

The probability of measuring 1 here is similarly

$$P(1 | \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)) = \left| -\frac{1}{\sqrt{2}} \right|^2 = \frac{1}{2}.$$

An unexpected feature of this coin comes from the fact, that the Hadamard-matrix is Hermitian (self-adjoint), i.e. $\mathbf{H}^\dagger = \mathbf{H}$, while also unitary, i.e. $\mathbf{H}^\dagger = \mathbf{H}^{-1}$, which results in $\mathbf{H}^{-1} = \mathbf{H}$, thus $\mathbf{H}\mathbf{H} = \mathbf{I}$. This means, that after flipping the coin twice without measuring it, it will return the coin state to its origin. For example, starting from $|0\rangle$:

$$\mathbf{H}^2|0\rangle = \mathbf{H} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{2}(|0\rangle + |1\rangle + |0\rangle - |1\rangle) = |0\rangle.$$

After the second flip, the probability of measuring $|0\rangle$ is 1, due to the destructive interference between the two $|1\rangle$ probability amplitudes, demonstrating a significant contrast between classical and quantum walks.

Definition 4.1 (2ⁿ dimensional Hadamard-coin). A 2ⁿ dimensional Hadamard-coin operator can be created by taking the tensor product of the 2 dimensional Hadamard-coin n times: $\mathbf{H}^{\otimes n}$.

4.1.2 Grover coin

The Grover coin originates from Grover's search algorithm, where it is applied as the diffusion operator.

Let $|D\rangle$ be the following state:

$$|D\rangle = \mathbf{H}^{\otimes n} |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle.$$

Using $|D\rangle$, the Grover coin is the following unitary matrix:

$$\mathbf{G} = 2 |D\rangle \langle D| - \mathbf{I}.$$

If $N = 2^n$, then \mathbf{G} unrolls to the following representation:

$$\mathbf{G} = \begin{pmatrix} \frac{2}{N} - 1 & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & \frac{2}{N} - 1 & \cdots & \frac{2}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} - 1 \end{pmatrix}.$$

4.1.3 Fourier coin

In contrast to the Hadamard and Grover coins, the Fourier coin can be of any size, not just a power of 2. A size N Fourier-coin, F_N is defined by the matrix of the Quantum Fourier Transform:

$$\mathbf{F}[k, l] = \frac{1}{\sqrt{N}} \omega^{kl}$$

where ω is the N -th root of unity,

$$\omega = e^{\frac{2\pi i}{N}}.$$

\mathbf{F} unrolls to the following representation:

$$\mathbf{F} = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix}$$

4.2 Quantum walks on the line

Kempe introduces quantum walks from a physicist's perspective in [6] using a particle characterised by its position on the line $|x\rangle$ and its spin state $|s\rangle$.

4.2.1 State space

Spin state

The spin state is in H_2 with the basis states spin up and down:

$$\begin{aligned} |\uparrow\rangle &= |0\rangle, \\ |\downarrow\rangle &= |1\rangle. \end{aligned}$$

The spin state vector is then given by:

$$|s\rangle = s_0 |\uparrow\rangle + s_1 |\downarrow\rangle.$$

Position state

At the start of the walk the particle is at the origin $|0\rangle$ and the walking lasts for N steps. The position state is in $H_{(2N+1)}$ with the following basis vectors corresponding to the possible positions on the line.

$$\{|-N\rangle, |-(N-1)\rangle, \dots, |-1\rangle, |0\rangle, |1\rangle, \dots, |N-1\rangle, |N\rangle\}$$

I index the basis states using negative numbers to match the labels on the axis.

The position state vector is then given by:

$$|x\rangle = \sum_{i=-N}^N x_i |i\rangle.$$

Composite state

The composite state of the system, according to [PostulateIV] is then

$$|x\rangle \otimes |s\rangle.$$

4.2.2 Evolution

The particle travels on the line based on its current spin state:

- If the current spin state is $|0\rangle$, the particle moves to the left, i.e. from position $|i\rangle$ the particle travels to position $|i-1\rangle$.
- If the current spin state is $|1\rangle$, the particle moves to the right, i.e. from position $|i\rangle$ the particle travels to position $|i+1\rangle$.

This step is realised with the unitary matrix \mathbf{S} which operates on the complete state of the system, $|x\rangle \otimes |s\rangle$ and is assembled from a left and a right shift operator acting on $|x\rangle$ and another operator for checking $|s\rangle$ compiled using tensor product.

Definition 4.2 (Left shift operator). To move from position $|i\rangle$ to the left ($|i-1\rangle$) the position vector is multiplied with the following \mathbf{L} matrix, containing 1's above the diagonal. To keep \mathbf{S} unitary, an unused transition must be added in the lower left corner (see Theorem 4.1).

$$\mathbf{L} = |N\rangle\langle -N| + \sum_{i=-(N-1)}^N |i-1\rangle\langle i| = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & & & \ddots & 1 \\ 1 & 0 & \cdots & & 0 \end{pmatrix} \quad (4.1)$$

For a given basis vector $|j\rangle$ only one of the summands in \mathbf{L} is non-zero, where $i = j$, resulting in the required shift being performed.

$$\mathbf{L}|j\rangle = |j-1\rangle\langle j|j\rangle = |j-1\rangle$$

Definition 4.3 (Right shift operator). To move from position $|i\rangle$ to the right ($|i+1\rangle$) the position vector is multiplied with the following \mathbf{R} matrix, containing 1's under the diagonal. To keep \mathbf{S} unitary, an unused transition must be added in the top right corner (see Theorem 4.1).

$$\mathbf{R} = |-N\rangle\langle N| + \sum_{i=-N}^{N-1} |i+1\rangle\langle i| = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ 1 & \ddots & & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix} \quad (4.2)$$

For a given basis vector $|j\rangle$ only one of the summands in \mathbf{R} is non-zero, where $i = j$, resulting in the required shift being performed.

$$\mathbf{R}|j\rangle = |j+1\rangle\langle j|j\rangle = |j+1\rangle$$

Shift operator

Using matrixes L and R operating on the position register $|x\rangle$ only, we construct a unitary operator S , which operates on the composite state of the system, $|x\rangle \otimes |s\rangle$, executing matrix \mathbf{L} on $|x\rangle$ only when $|s\rangle = |0\rangle$ and matrix \mathbf{R} only when $|s\rangle = |1\rangle$.

$$\mathbf{S} = \mathbf{L} \otimes |0\rangle \langle 0| + \mathbf{R} \otimes |1\rangle \langle 1| \quad (4.3)$$

The execution logic is as follows:

$$\begin{aligned} \mathbf{S}(|x\rangle \otimes |s\rangle) &= \\ (\mathbf{L} \otimes |0\rangle \langle 0| + \mathbf{R} \otimes |1\rangle \langle 1|)(|x\rangle \otimes |s\rangle) &= \\ (\mathbf{L} \otimes |0\rangle \langle 0|)(|x\rangle \otimes |s\rangle) + (\mathbf{R} \otimes |1\rangle \langle 1|)(|x\rangle \otimes |s\rangle) &= \dots \end{aligned}$$

using [TensorMixedProduct]:

$$\begin{aligned} \dots = \mathbf{L} |x\rangle \otimes (|0\rangle \langle 0|s\rangle) + \mathbf{R} |x\rangle \otimes (|1\rangle \langle 1|s\rangle) &= \\ |x-1\rangle \otimes s_0 |0\rangle + |x+1\rangle \otimes s_1 |1\rangle &= \\ s_0 |x-1, 0\rangle + s_1 |x+1, 1\rangle. \end{aligned}$$

- If the spin state was $|s\rangle = |0\rangle$ at the beginning, then $s_0 = 1$ and $s_1 = 0$, which means that the resulting system state is $|x-1, 0\rangle$, which means that the particle shifted to the left, as designed.
- If the spin state was $|s\rangle = |1\rangle$ at the beginning, then $s_0 = 0$ and $s_1 = 1$, which means that the resulting system state is $|x+1, 1\rangle$, which means that the particle shifted to the right, also as intended.

Furthermore, the spin state can be any mixed state $s_0 |0\rangle + s_1 |1\rangle$ as well. In this case the particle will shift *both* to the left and to the right, at the same time. When measured, the particle can be found in position $|x-1\rangle$ with probability $|s_0|^2$ and in position $|x+1\rangle$ with probability $|s_1|^2$.

In quantum graph walks, the walker can simultaneously explore multiple parallel paths in the graph, at the same time. With good design, this behaviour can be used to search the graph faster than in classical random graph walks.

Coin operator

To inject the quantum superposition into the walk, the particle's spin state is transformed using any 2 dimensional unitary matrix between shift operations. The Hadamard, Grover and Fourier coins mentioned earlier are commonly used as coin operators.

For any C operator on the coin register, the unitary transform for the composite system is defined as follows:

$$\hat{\mathbf{C}} = \mathbf{I} \otimes \mathbf{C}$$

since the coin operator does not modify the position register.

Evolution operator

Combining the shift operator and the coin operator together, we obtain the following evolution operator, defining one step of the quantum walk on the line. The step consists of flipping the coin once, then applying the shifting the walker's position accordingly, as follows:

$$\mathbf{U} = \mathbf{S}\hat{\mathbf{C}} = \mathbf{S}(\mathbf{I} \otimes \mathbf{C})$$

4.2.3 Measurement

To measure the probability of the particle being at position $|i\rangle$, the projective measurement operator acting on $|x\rangle$ is defined as $\mathbf{P}_i = |i\rangle \langle i|$, in accordance with [PostulateIIIProjective].

Since the coin register need not be measured, we apply the identity operator on it, using $\mathbf{P}_i \otimes \mathbf{I}$ on the complete system to measure the particle's current position.

The probability of finding the particle in position i is:

$$P(i|x) = \langle x, s | \mathbf{P}_i \otimes \mathbf{I} | x, s \rangle = \dots$$

using [TensorMixedProduct]:

$$\dots = \langle x | \mathbf{P}_i | x \rangle \langle s | \mathbf{I} | s \rangle = \langle x | \mathbf{P}_i | x \rangle 1 = \langle x | \mathbf{P}_i | x \rangle = \langle x | i \rangle \langle i | x \rangle = |\langle i | x \rangle|^2 = |x_i|^2$$

4.3 Generalization of Quantum Walks

After presenting quantum walking on the line, I review and extend two approaches to generalize it in this chapter.

1. **Section 4.3.1: Use multiple two-dimensional coins:** In [9], Renato Portugal shows the generalization of quantum walks on a line to a two-dimensional grid, using a method with 2 two-dimensional coins. In this work, I prove how his method reduces to effectively two synchronous independent walks on the x and y axes. Then I improve his technique by generalizing to arbitrarily large dimensional grids in a more memory-efficient way than what would naturally follow from his description.
2. **Section 4.3.2: Use a single higher dimensional coin:** In [9], Renato Portugal describes the generalization of a quantum walk on a line to an arbitrary undirected graph and gives the necessary condition for creating the unitary transition matrix without proof. In this work, I generalize to directed graphs and give proof of the generalization of the condition using directed graphs.

4.3.1 Generalization using multiple independent 2 dimensional coins

In [9], Renato Portugal defines the following method for Quantum Walking on a 2D grid:

Let the position state of the walker be $|x, y\rangle$ and the two coins $|c_x\rangle$, acting on the x coordinate and $|c_y\rangle$, acting on the y coordinate of the walker.

The shift operator moves the walker on the grid diagonally, according to the current state of the two coins, described by

$$\mathbf{S} |x, y\rangle |c_x\rangle |c_y\rangle = |x + (-1)^{c_x}, y + (-1)^{c_y}\rangle |c_x\rangle |c_y\rangle, \quad (4.4)$$

and the coin operator leaves the position state in place while flipping both coins at the same time, described by

$$\hat{\mathbf{C}} = \mathbf{I} \otimes \mathbf{C}_4 = \mathbf{I} \otimes (\mathbf{C}_2 \otimes \mathbf{C}_2). \quad (4.5)$$

Issues with this method

In 4.4, we can see how the matrix \mathbf{S} will quickly increase in size, as further dimensions are added to the equation. In 2D, if the walker takes N steps, the size of \mathbf{S} is

$$(2N + 1)^2(2N + 1)^2 2^2 2^2 = 16(2N + 1)^4 = O(N^4).$$

To increase the dimension count, one would naturally append more coordinates to the composite position state and add further coins, for example in 3D \mathbf{S} would become

$$\mathbf{S} |x, y, z\rangle |c_x\rangle |c_y\rangle |c_z\rangle = |x + (-1)^{c_x}, y + (-1)^{c_y}, z + (-1)^{c_z}\rangle |c_x\rangle |c_y\rangle |c_z\rangle,$$

For a dimension count d , the size of \mathbf{S} is exponential in d .

$$((2N + 1)^2(2^2))^d = (4(2N + 1)^2)^d = O(N^{2d}).$$

My improvements

Since in 4.4 the coordinates of the walker are updated independently by the separate coins, I was able to disassemble \mathbf{S} into smaller matrices, using the properties of the tensor product.

To do this, in what follows, I first define \mathbf{S} in 2D explicitly (as opposed to the implicit definition in 4.4, stating only how \mathbf{S} updates the state of the system). I will be using the matrices \mathbf{L} defined by Equation (4.1) and \mathbf{R} defined by Equation (4.2).

Notice that \mathbf{R} increases the walker's coordinates on the line, while \mathbf{L} decreases them. This means, that on the y axis \mathbf{R} acts by moving the walker up, and \mathbf{L} acts by moving the walker down.

When the coins are in the state $|c_x, c_y\rangle = |0, 0\rangle$, the walker moves up and to the right. This movement is captured by $\mathbf{S}_{0,0}$, acting on the position state:

$$\mathbf{S}_{0,0} = \mathbf{R} \otimes \mathbf{R}$$

The other three \mathbf{S}_{c_x, c_y} matrices are defined similarly:

$$\mathbf{S}_{0,1} = \mathbf{R} \otimes \mathbf{L}$$

$$\mathbf{S}_{1,0} = \mathbf{L} \otimes \mathbf{R}$$

$$\mathbf{S}_{1,1} = \mathbf{L} \otimes \mathbf{L}$$

Then, I assemble \mathbf{S} using $\mathbf{S}_{0,0}$, $\mathbf{S}_{0,1}$, $\mathbf{S}_{1,0}$ and $\mathbf{S}_{1,1}$ the following way:

- $\mathbf{S}_{0,0}$ acts only when $|c_x, c_y\rangle = |0, 0\rangle$,
- $\mathbf{S}_{0,1}$ acts only when $|c_x, c_y\rangle = |0, 1\rangle$,
- $\mathbf{S}_{1,0}$ acts only when $|c_x, c_y\rangle = |1, 0\rangle$, and finally
- $\mathbf{S}_{1,1}$ acts only when $|c_x, c_y\rangle = |1, 1\rangle$.

Using the same method as in Equation (4.3) I arrive at:

$$\begin{aligned} \mathbf{S} = & \mathbf{S}_{0,0} \otimes |0, 0\rangle \langle 0, 0| + \\ & \mathbf{S}_{0,1} \otimes |0, 1\rangle \langle 0, 1| + \\ & \mathbf{S}_{1,0} \otimes |1, 0\rangle \langle 1, 0| + \\ & \mathbf{S}_{1,1} \otimes |1, 1\rangle \langle 1, 1| \end{aligned}$$

After substituting the \mathbf{S}_{c_x, c_y} matrices in:

$$\begin{aligned} \mathbf{S} = & (\mathbf{R} \otimes \mathbf{R}) \otimes |0, 0\rangle \langle 0, 0| + \\ & (\mathbf{R} \otimes \mathbf{L}) \otimes |0, 1\rangle \langle 0, 1| + \\ & (\mathbf{L} \otimes \mathbf{R}) \otimes |1, 0\rangle \langle 1, 0| + \\ & (\mathbf{L} \otimes \mathbf{L}) \otimes |1, 1\rangle \langle 1, 1| \end{aligned}$$

Let us name the coin state $|0\rangle$ heads and the coin state $|1\rangle$ tails. Then, using the following equalities and introducing matrices \mathbf{H} and \mathbf{T} , as shorthands:

$$\begin{aligned} |0, 0\rangle \langle 0, 0| &= (|0\rangle \langle 0|) \otimes (|0\rangle \langle 0|) = \mathbf{H} \otimes \mathbf{H} \\ |0, 1\rangle \langle 0, 1| &= (|0\rangle \langle 0|) \otimes (|1\rangle \langle 1|) = \mathbf{H} \otimes \mathbf{T} \\ |1, 0\rangle \langle 1, 0| &= (|1\rangle \langle 1|) \otimes (|0\rangle \langle 0|) = \mathbf{T} \otimes \mathbf{H} \\ |1, 1\rangle \langle 1, 1| &= (|1\rangle \langle 1|) \otimes (|1\rangle \langle 1|) = \mathbf{T} \otimes \mathbf{T} \end{aligned}$$

I arrive at:

$$\begin{aligned}
\mathbf{S} = & (\mathbf{R} \otimes \mathbf{R}) \otimes (\mathbf{H} \otimes \mathbf{H}) + \\
& (\mathbf{R} \otimes \mathbf{L}) \otimes (\mathbf{H} \otimes \mathbf{T}) + \\
& (\mathbf{L} \otimes \mathbf{R}) \otimes (\mathbf{T} \otimes \mathbf{H}) + \\
& (\mathbf{L} \otimes \mathbf{L}) \otimes (\mathbf{T} \otimes \mathbf{T})
\end{aligned}$$

Then, using [TensorMixedProduct] I inflate the equation with (appropriately sized) \mathbf{I} matrices:

$$\begin{aligned}
\mathbf{S} = & ((\mathbf{R} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{R})) \otimes ((\mathbf{H} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{H})) + \\
& ((\mathbf{R} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{L})) \otimes ((\mathbf{H} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{T})) + \\
& ((\mathbf{L} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{R})) \otimes ((\mathbf{T} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{H})) + \\
& ((\mathbf{L} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{L})) \otimes ((\mathbf{T} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{T}))
\end{aligned}$$

At this point, I introduce a few aliases to make the equation more manageable. Notice, how for example $\mathbf{I} \otimes \mathbf{R}$ is acting on the 2D position state, but only updating the y coordinate to move the walker upward. This gives a way to naturally define $\mathbf{S}_{\text{up}} = \mathbf{I} \otimes \mathbf{R}$, and similarly:

$$\begin{aligned}
\mathbf{S}_{\text{right}} &= \mathbf{R} \otimes \mathbf{I} \\
\mathbf{S}_{\text{left}} &= \mathbf{L} \otimes \mathbf{I} \\
\mathbf{S}_{\text{up}} &= \mathbf{I} \otimes \mathbf{R} \\
\mathbf{S}_{\text{down}} &= \mathbf{I} \otimes \mathbf{L}
\end{aligned}$$

At the same time, for example $\mathbf{I} \otimes \mathbf{H}$ is acting on the composite coin state, but only checking if the second coin's state is heads, which is the coin for the y axis. This gives a way to naturally define $\mathbf{H}_y = \mathbf{I} \otimes \mathbf{H}$, and similarly:

$$\begin{aligned}
\mathbf{H}_x &= \mathbf{H} \otimes \mathbf{I} \\
\mathbf{T}_x &= \mathbf{T} \otimes \mathbf{I} \\
\mathbf{H}_y &= \mathbf{I} \otimes \mathbf{H} \\
\mathbf{T}_y &= \mathbf{I} \otimes \mathbf{T}
\end{aligned}$$

Substituting all of the aliases:

$$\begin{aligned}
\mathbf{S} = & (\mathbf{S}_{\text{right}} \mathbf{S}_{\text{up}}) \otimes (\mathbf{H}_x \mathbf{H}_y) + \\
& (\mathbf{S}_{\text{right}} \mathbf{S}_{\text{down}}) \otimes (\mathbf{H}_x \mathbf{T}_y) + \\
& (\mathbf{S}_{\text{left}} \mathbf{S}_{\text{up}}) \otimes (\mathbf{T}_x \mathbf{H}_y) + \\
& (\mathbf{S}_{\text{left}} \mathbf{S}_{\text{down}}) \otimes (\mathbf{T}_x \mathbf{T}_y)
\end{aligned}$$

Then, using [TensorMixedProduct] again, I arrive at:

$$\begin{aligned} \mathbf{S} = & (\mathbf{S}_{\text{right}} \otimes \mathbf{H}_x)(\mathbf{S}_{\text{up}} \otimes \mathbf{H}_y) + \\ & (\mathbf{S}_{\text{right}} \otimes \mathbf{H}_x)(\mathbf{S}_{\text{down}} \otimes \mathbf{T}_y) + \\ & (\mathbf{S}_{\text{left}} \otimes \mathbf{T}_x)(\mathbf{S}_{\text{up}} \otimes \mathbf{H}_y) + \\ & (\mathbf{S}_{\text{left}} \otimes \mathbf{T}_x)(\mathbf{S}_{\text{down}} \otimes \mathbf{T}_y) \end{aligned}$$

Then using the distributive property of matrix multiplication with respect to matrix addition I finally arrive at:

$$\mathbf{S} = ((\mathbf{S}_{\text{right}} \otimes \mathbf{H}_x) + (\mathbf{S}_{\text{left}} \otimes \mathbf{T}_x))((\mathbf{S}_{\text{up}} \otimes \mathbf{H}_y) + (\mathbf{S}_{\text{down}} \otimes \mathbf{T}_y))$$

We can see from the equation above, that \mathbf{S} is actually the product of two shift operators, one only acting on the x coordinate using the first coin's state, the other acting only on the y coordinate, according to the second coin's state.

$$\begin{aligned} \mathbf{S}_x &= (\mathbf{S}_{\text{right}} \otimes \mathbf{H}_x) + (\mathbf{S}_{\text{left}} \otimes \mathbf{T}_x) \\ \mathbf{S}_y &= (\mathbf{S}_{\text{up}} \otimes \mathbf{H}_y) + (\mathbf{S}_{\text{down}} \otimes \mathbf{T}_y) \end{aligned}$$

$$\mathbf{S} = \mathbf{S}_x \mathbf{S}_y$$

This proves, that the walk on the 2D grid decomposes into two independent line walks on the axes, since \mathbf{S}_x only touches the x coordinate and the first coin, while \mathbf{S}_y only touches the y coordinate and the second coin and there is no entanglement between the registers of the x and y axes. Using this fact, we can simply simulate two independent quantum walks on the line in parallel, or sequentially, using the same registers, which vastly decreases the memory needs of the algorithm. Running in parallel, the memory needs is now $d(4(2N+1)^2) = O(dN^2)$, or running sequentially $(4(2N+1))^2 = O(N^2)$, however the latter uses dN steps, instead of N , and d measurements, instead of 1.

4.3.2 Generalization using a single higher dimensional coin

Using this method, we generalize quantum walking to arbitrary directed graphs. To do so, we first generalize to regular graphs, then discuss how non-regular graphs can be regularized.

In a d -regular graph, the vertices have d neighbours, meaning the walker must choose from d possible directions at every step. Thus the coin is d dimensional. Using this idea as a starting point, we can reverse engineer the generalized walk from the walk on the line by starting from the evolution operator, which assumes nothing about the given graph or coin.

$$\mathbf{U} = \mathbf{S}\hat{\mathbf{C}} = \mathbf{S}(\mathbf{I} \otimes \mathbf{C})$$

In this equation, \mathbf{C} can be any d -dimensional unitary matrix. The definition of \mathbf{S} requires more thought, since \mathbf{S} has to encode the graph's structure, while also satisfying [PostulateII].

In one dimension, \mathbf{S} was defined the following way:

$$\mathbf{S} = \mathbf{L} \otimes |0\rangle\langle 0| + \mathbf{R} \otimes |1\rangle\langle 1|.$$

$|0\rangle\langle 0|$ and $|1\rangle\langle 1|$ were matrices that checked the current state of the coin and "activated" the transition \mathbf{L} or \mathbf{R} accordingly. In d dimensions, the coin has d possible states, i.e. "sides"

$$\{|0\rangle, |1\rangle, \dots, |d-1\rangle\},$$

which means \mathbf{S} will be constructed using d transition matrices, describing the graph's structure

$$\mathbf{S} = \mathbf{S}_0 |0\rangle\langle 0| + \mathbf{S}_1 |1\rangle\langle 1| + \dots + \mathbf{S}_{d-1} |d-1\rangle\langle d-1|.$$

In the 1 dimensional case \mathbf{L} and \mathbf{R} described stepping to the left and to the right, which are the directed edges of the line graph and $\mathbf{L} + \mathbf{R}$ is the adjacency matrix of the line graph. To generalize this, $\mathbf{S}_0 + \mathbf{S}_1 + \dots + \mathbf{S}_{d-1}$ is going to be the adjacency matrix of the d -regular graph.

The question is how do we construct the matrices $\mathbf{S}_0, \mathbf{S}_1, \dots, \mathbf{S}_{d-1}$ from a given adjacency matrix of a d -regular graph? It turns out, that in order for \mathbf{S} to satisfy [PostulateII], there are strict rules on how these \mathbf{S}_i matrices can be defined.

It seems to be a well-known fact in the literature, that for d -regular undirected graphs with a valid edge coloring using d colors, a possible choice for the sides of the coin correspond to the colorsets of the edges. This means, that the S_i adjacency matrix contains all of the edges that have the i th color assigned to them, in both directions (i.e. S_i is symmetric). This is also mentioned in [9], however no proof is given in this book or any other books and articles I have found during research.

In the following section, I present a more generalized theorem for directed d -regular graphs formulated and proven by me. Then I discuss how the special case of the theorem for undirected graphs gives the edge coloring as a result.

Theorem 4.1. Given a coined quantum walk on a directed, d -regular graph G , in the shift operator of the walk: $\mathbf{S} = \sum_{i=0}^{d-1} \mathbf{S}_i \otimes |i\rangle\langle i|$, assuming the \mathbf{S}_i are nonnegative, real matrices, it follows that they are **permutation** matrices.

Proof.

According to [PostulateII], \mathbf{S} must be unitary.

According to [Unitary] the columns of \mathbf{S} form an orthonormal basis. This means, that the inner product of any two different columns is 0.

Let \mathbf{S} be a matrix of size $(N \times N)$. Then

$$(\mathbf{S} |k\rangle)^\dagger (\mathbf{S} |j\rangle) = 0 \quad \forall j \neq k, 0 \leq j, k \leq N.$$

Since both the \mathbf{S}_i matrices and the $|i\rangle \langle i|$ matrices contain only non-complex, non-negative values, this means that \mathbf{S} contains also only non-complex, non-negative values. Thus the only way the inner product of two different columns can be 0 is if the columns don't contain non-zero values in the same row.

From this observation follows, that for each individual S_i matrix, no two different columns can contain non-zero values in the same row. If there were two different columns in S_i , that contained non-zero values in the same row, then that would result in $\mathbf{S}_i \otimes |i\rangle \langle i|$ containing two different columns containing non-zero values in the same row, which would result in \mathbf{S} containing two different columns containing non-zero values in the same row (since no matrices contain negative or complex values), which is a contradiction.

Using [Unitary], the rows of \mathbf{S} also form an orthonormal basis. With similar reasoning, it can be proven that for each individual S_i matrix, no two different rows can contain non-zero values in the same column.

From these two observations follows, that each matrix \mathbf{S}_i contains exactly one non-zero value in each row and also each column. Adding the fact, that the rows and columns of \mathbf{S} are normalized, means that this non-zero value must always be a 1, resulting in the \mathbf{S}_i being **permutation** matrices.

□.

When this theorem is applied to undirected graphs, the adjacency matrices \mathbf{S}_i can be constructed to be symmetric (since the graph's adjacency matrix is also symmetric) and in this case they correspond to a valid edge coloring using d colors (since if $\mathbf{S}_i[j, k] = 1$, then also $\mathbf{S}_i[k, j] = 1$ due to symmetry, and the $\{k, j\}$ edge has the color i , while no other edges of vertex j or k have the i th color, since \mathbf{S}_i is a permutation matrix).

Applying Vizing's theorem to d -regular graphs, the graphs can be categorized into two classes:

- **Class 1 d -regular graphs:** Their edge-chromatic number is d . The construction defined in this section works for these graphs.
- **Class 2 d -regular graphs:** Their edge-chromatic number is $d + 1$. [9] seems to state, that for these types of graphs, the position-coin notation does not give a way to construct a quantum walk on them (and the arc notation shall be used), however using Theorem (4.1) if we extend the method to directed graphs, it can be possible to do so. For example, the triangle is 2-regular, but its edge-chromatic number is 3. However, if we direct the edges both ways, we suddenly arrive at two cycles of length 3, for which the adjacency matrices are permutation matrices, allowing for the construction of a unitary evolution operator.

Chapter 5

Simulator software

In this chapter, I present the simulator software I wrote. I discuss the currently available solutions, why I chose to write the software, the architecture, the components and design patterns I used, the challenges I faced during the development and the solutions I found.

5.1 Currently available solutions

Since publicly accessible quantum computers currently only have around 5-10 qubits, it is not viable at the moment to run quantum walks on a real quantum computer. Hence why, when I started researching quantum walks, I quickly began looking into simulator software. While there are many of these currently available, most of them have at least one of the following issues:

1. Not maintained and developed anymore: the last commit was years ago.
2. Written in a low-level language, like C++, in a script-like fashion, with a prominent focus on memory and performance optimization while neglecting readability, modularity and extensibility.
3. Works exclusively on a specific type of graph, for example, n -dimensional lattices only.
4. Unable to compare and contrast classical and quantum walks on the same graph, running only quantum simulations.
5. Hard to understand as a novice.

There is no general, open-source solution available that is designed and developed using sound software engineering practices and an architecture that allows for experimentation with different kinds of graphs with both classical and quantum simulations available.

I intend my solution to be valuable for research purposes while also providing a readable open-source codebase for college students to study the algorithm.

5.2 Architecture

The architecture of my simulator program employs the Strategy design pattern, which is described in the following way:

”Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.” [3]

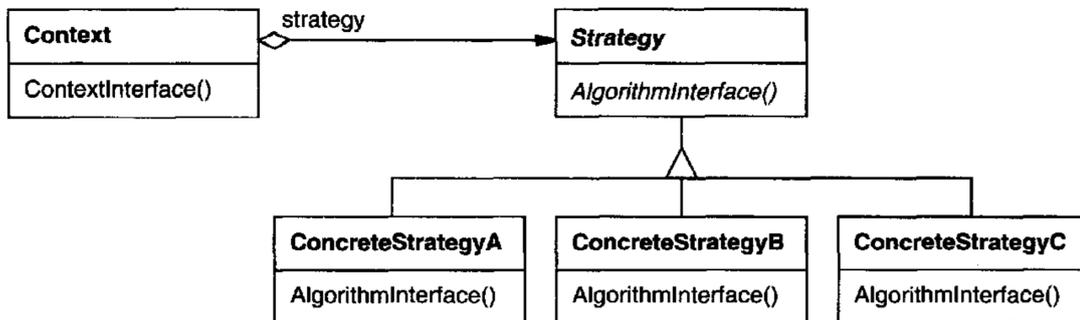


Figure 5.1: UML diagram for the Strategy design pattern from [3]

This is a great design pattern for research purposes since it facilitates experimentation with various algorithms for the same purpose. It also makes the code easily readable, as the Strategy interface provides an abstraction layer between the Context and the concrete implementation.

5.3 Language choice

With the specified goals and the architecture in mind, I needed a language that is object-oriented, easily readable by beginners and has extensive capabilities for using complex numbers, linear algebra and plotting. For these purposes, I choose the Python language. Python is concise, it reads like pseudocode and has libraries such as NumPy, SciPy and Matplotlib, and so on, covering all areas of data science. Furthermore, it is well-known and extensively used by researchers with no software engineering background, allowing for easier collaboration.

5.4 High level design

The source code of the software can be divided into three parts:

- Graph models
- Simulators
- Running, configuration and result collection

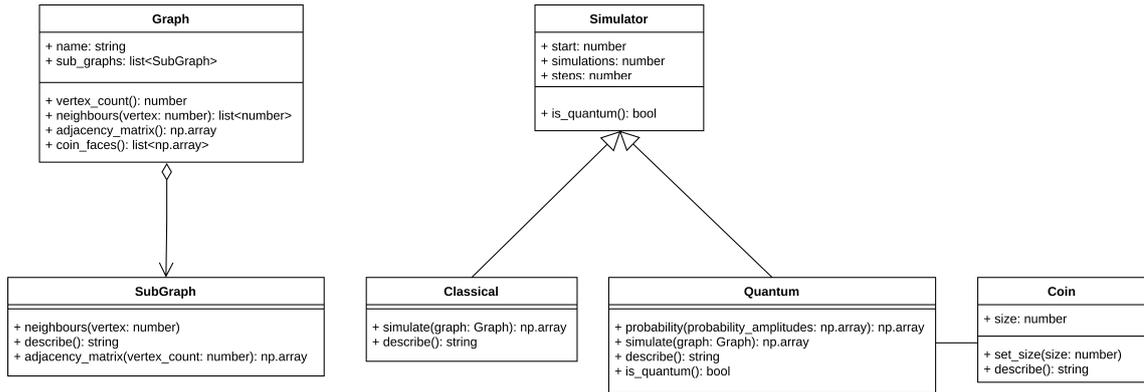


Figure 5.2: UML diagram for the Graph models and Simulators

On the UML diagram above, the SubGraph class is a Strategy, with the following ConcreteStrategy implementations:

- BinaryTree
- Bipartite
- Circle
- Grid
- Hypercube
- Path
- Random

Each of these employs an oracle that calculates neighbouring vertices on-the-fly.

Furthermore, the Simulator is also a Strategy, implemented by the Classical and the Quantum classes, the latter using the Coin Strategy, implemented by the Hadamard, Grover and Fourier (DFT) classes.

(For a cleaner diagram, I did not picture the SubGraph and Coin implementations.)

5.4.1 Graph models

I ran several experiments on various graphs while researching quantum graph walks, including paths, circles, bipartite graphs, hypercubes, and grids. Initially, I directly generated and stored their adjacency matrices, however, I quickly ran into memory scaling issues with this approach. Furthermore, in quantum research, graphs are typically built like 'Legos', glueing together a few common types, which was challenging to do with my original approach.

To combat these issues, I switched from the adjacency matrix representation to the oracle representation. The oracle is a function that returns the neighbours of a given vertex. Since I was using common graphs, I could calculate neighbouring indexes on-the-fly without storing anything about these graphs and only querying what is needed at the current step, dramatically reducing the memory requirements of the graph models.

5.4.2 Simulators

I implemented a classical and a quantum simulator class. The quantum simulator can currently simulate directed k regular graphs, however since the permutation matrix decomposition, or in the undirected case, the edge coloring of the matrix is an NP-complete problem, in the current setup, the graph oracle must be implemented in a way that returns the neighbours in the same color order for all inputs. Since the human programmer designs the oracle, this is not a critical limitation at the moment. I have implemented a check as a safety guard to ensure the resulting shift matrices are unitary in case an error is made while coding one of the oracles.

5.4.3 Running, configuration and result collection

Using the above classes, I developed a framework in which experimental runs can be configured very quickly. The results of the run are collected in an aggregated Latex document, using Matplotlib for creating various graphics. It contains the given graph, the named type of the subgraphs, the adjacency matrices, the distribution results of the simulations, including empirical hitting and mixing times and the eigenvalues and eigenvectors of the evolution operators. In the following chapter, I present several examples collected from these Latex reports of my experiments.

5.4.4 Availability

My simulator software is available under the open-source MIT license on my personal Github account, under the following link:

<https://github.com/nemkin/quantum>

Chapter 6

Results

In this chapter, I review classical and quantum walking on three specific graphs, for which interesting results can be observed. I discuss the evolution of the probability distributions and the hitting and mixing times for classical and quantum walks with the Hadamard, Grover and Fourier (DFT) coins.

6.1 Walks on the line

The first graph to be reviewed is the line (with 100 vertices), using the adjacency matrix below. It is important to note, that an extra edge has to be added to connect the two ends of the line (see Theorem (4.1) for details).

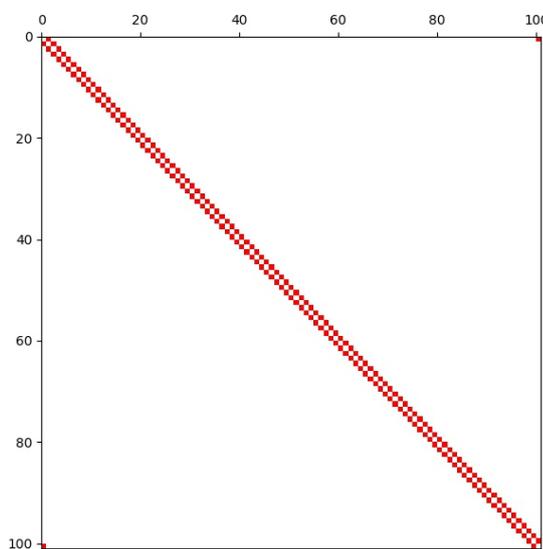


Figure 6.1: *Adjacency matrix of the line*

In the following pictures, we can see the changes in the probability distribution during the walk. The x axis contains the vertices, and the y axis contains the steps. The walker starts from the centre, and in the classical case, multiple runs are done to arrive at a probability distribution, while in the quantum case, a single walker is enough, as it spreads in superposition over the graph.

The ballistic nature of the walk can be seen from steps 0 to 50, where the bright yellow diagonals represent a strong probability concentration spreading to the two ends of the line. When the probability bumps reach the sides, they cross over and travel to the opposite ends.

From steps 50 to 200, we can see secondary, tertiary, and further yellow bumps travelling alongside the main ones. These reach the ends slower and cross over each other later. This results in a beautiful weaved pattern in the picture.

Since the line is a 2-regular graph, 2 dimensional coins are used. The 2 dimensional Hadamard-coin and Fourier-coin are identical, while the 2 dimensional Grover-coin results in the walker not moving away from the starting position, hence why only the Hadamard coin is shown in the distributions.

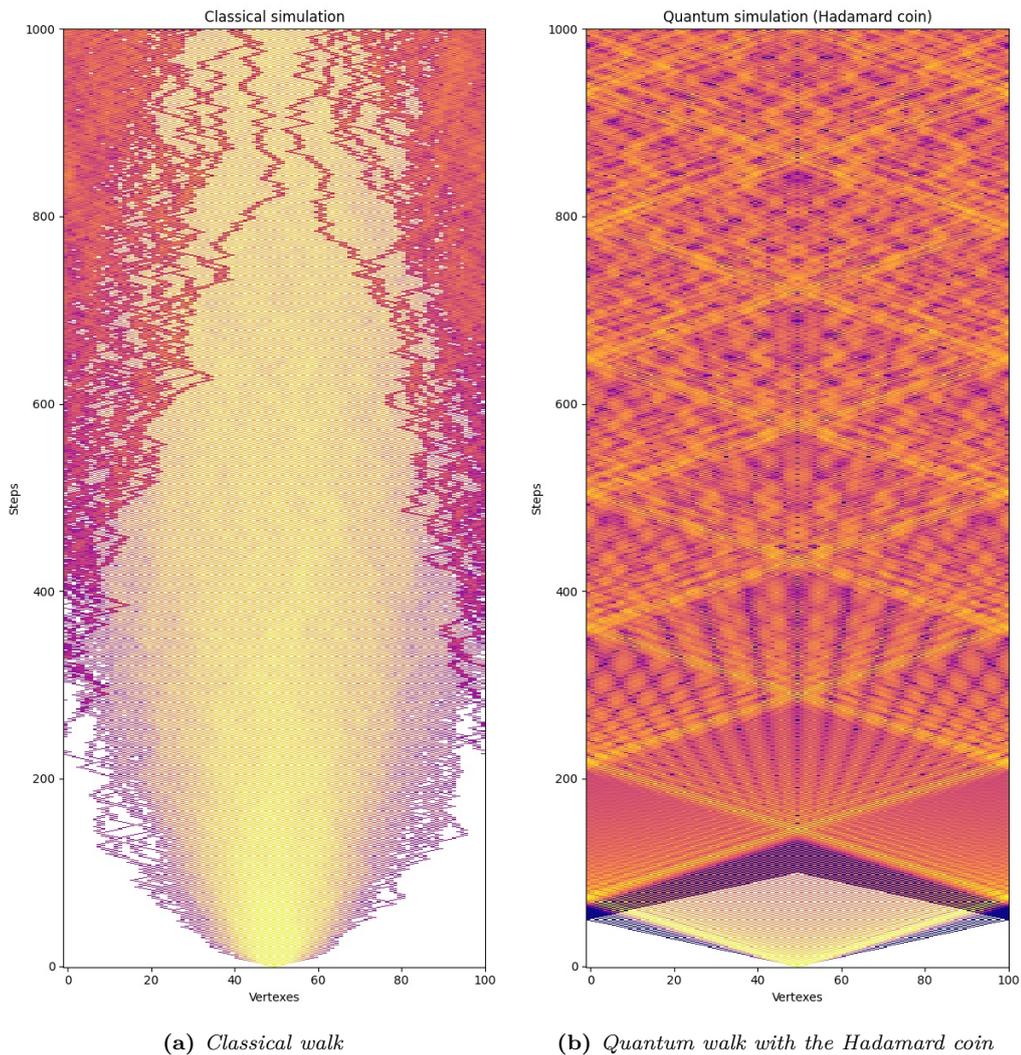


Figure 6.2: *Probability distribution of classical and quantum walks on the line*

I have empirically measured hitting and mixing times for the different types of walks. Hitting time is the expected number of steps to reach a specific vertex from the starting point. For this, I have plotted the number of steps it took to first reach a particular vertex from the starting point. Mixing time is the number of steps it takes before reaching the stationary distribution with ϵ error. For this, I have plotted the Euclidean difference between the walk's current and the end distribution.

In the following pictures, we can see the classical hitting and mixing times. Since my classical simulator approximates the distribution by running multiple walkers, the hitting time is slightly asymmetric. We can see from comparing the classical and the quantum hitting times that the quantum walk spreads faster than the classical one.

On the classical mixing time, we can see that the walker has not reached the stationary distribution. This is because until the walker reaches the ends of the line, the graph is essentially bipartite with two different limiting distributions and only after crossing over to the other side can the walk spread uniformly. This can also be seen in the probability distribution image above. At around step 300, the colour of the image intensifies at the sides. Before that, the distribution alternates between odd and even indexes having 0 probability, resulting in a chessboard pattern of white and colorful rectangles.

The quantum walk is mixing much better, as can be seen by the mixing time and the distribution image as well.

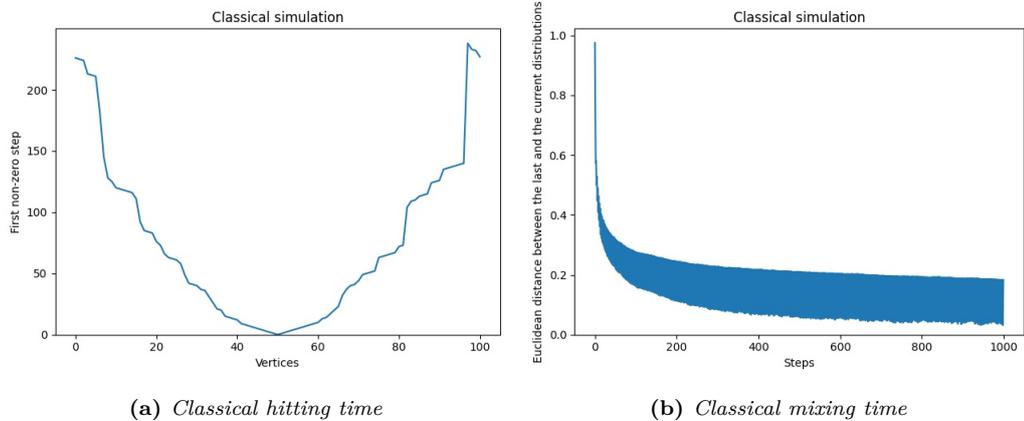


Figure 6.3: *Classical hitting and mixing times on the line*

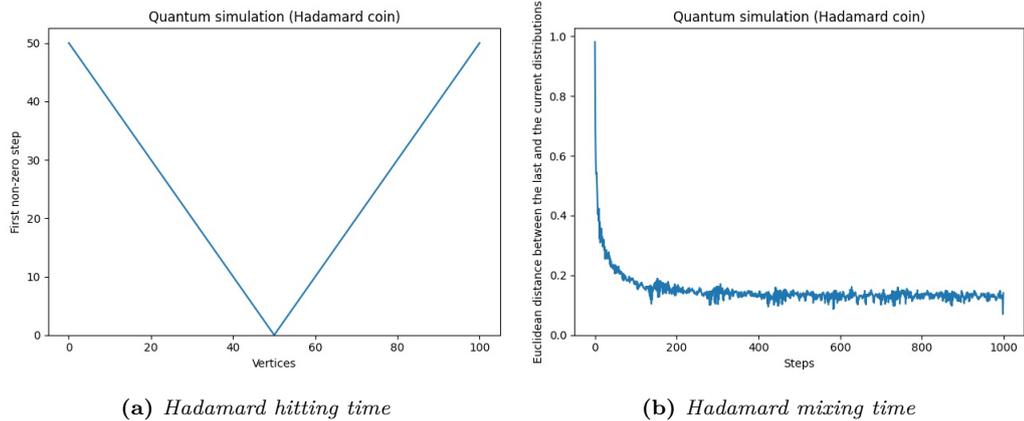


Figure 6.4: *Quantum (Hadamard) hitting and mixing times on the line*

6.2 Walks on the grid

The second graph reviewed is the 2 dimensional grid (with $4 \times 4 = 16$ vertices), using the adjacency matrix below.

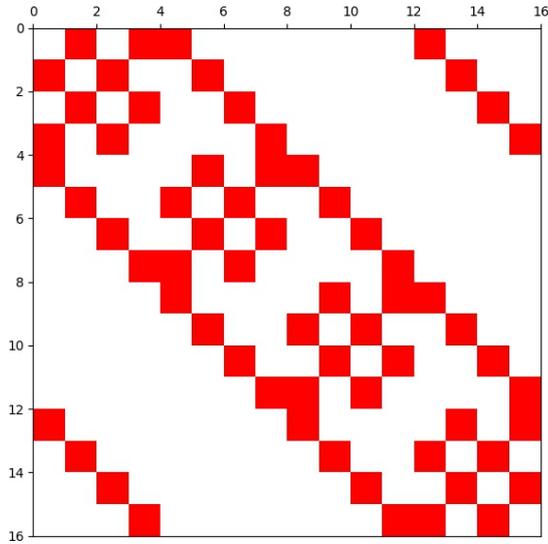
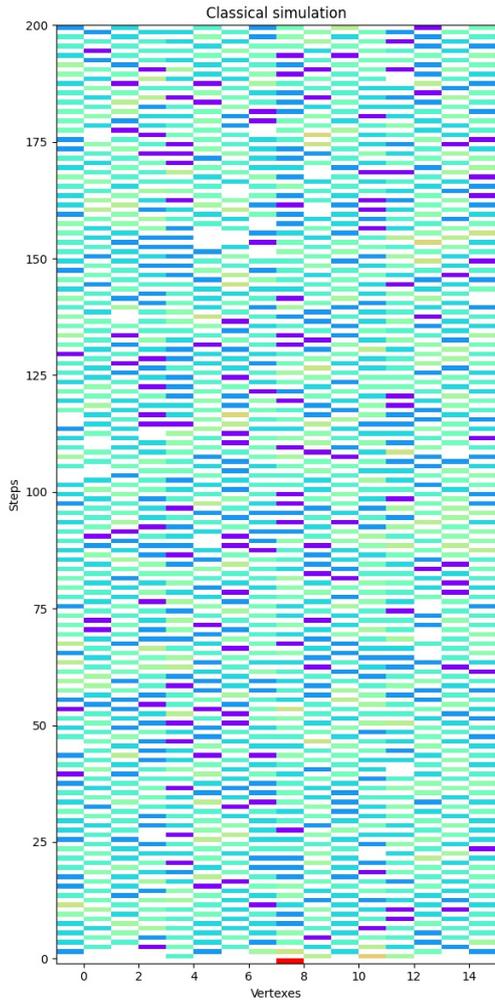


Figure 6.5: *Adjacency matrix of the grid*

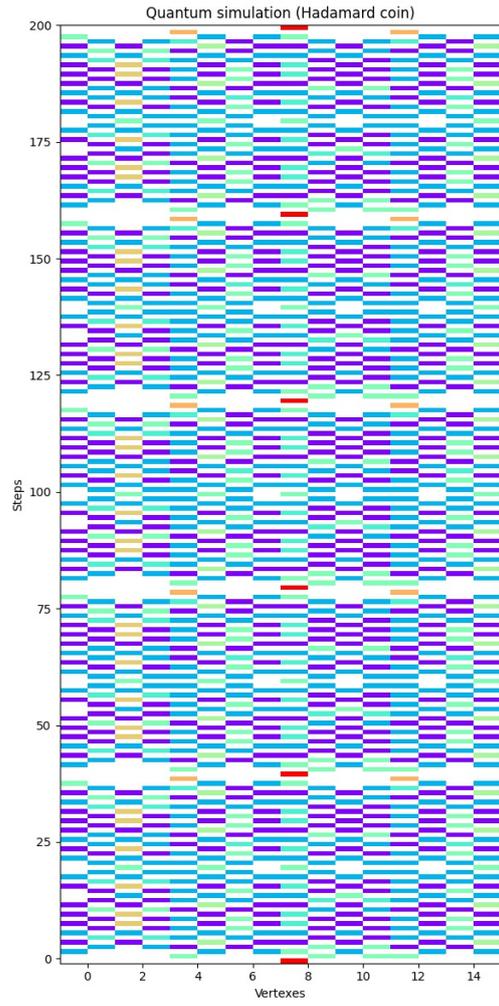
The following 4 images contain the classical, the quantum Hadamard, the quantum Grover and the quantum Fourier walks on the grid. The classical walk quickly spreads over the graph since all vertices are close to each other (as opposed to the line, where the maximum distance is large).

In the quantum case, using the Hadamard and Grover coins, an important quality of the quantum walks can be distinctly observed: quantum walks are periodic since the eigenvalues of the evolution operator are complex roots of unity. Furthermore, by choosing a vertex count that is a power of 2, I was able to create an evolution operator that has specific eigenvalues that result in the walker returning to its starting position with 100% probability (see the repeated red rectangles in the images), showing the cyclic nature of the quantum walk.

The Fourier coin mixes the state much better, resulting in no specific order in that image.

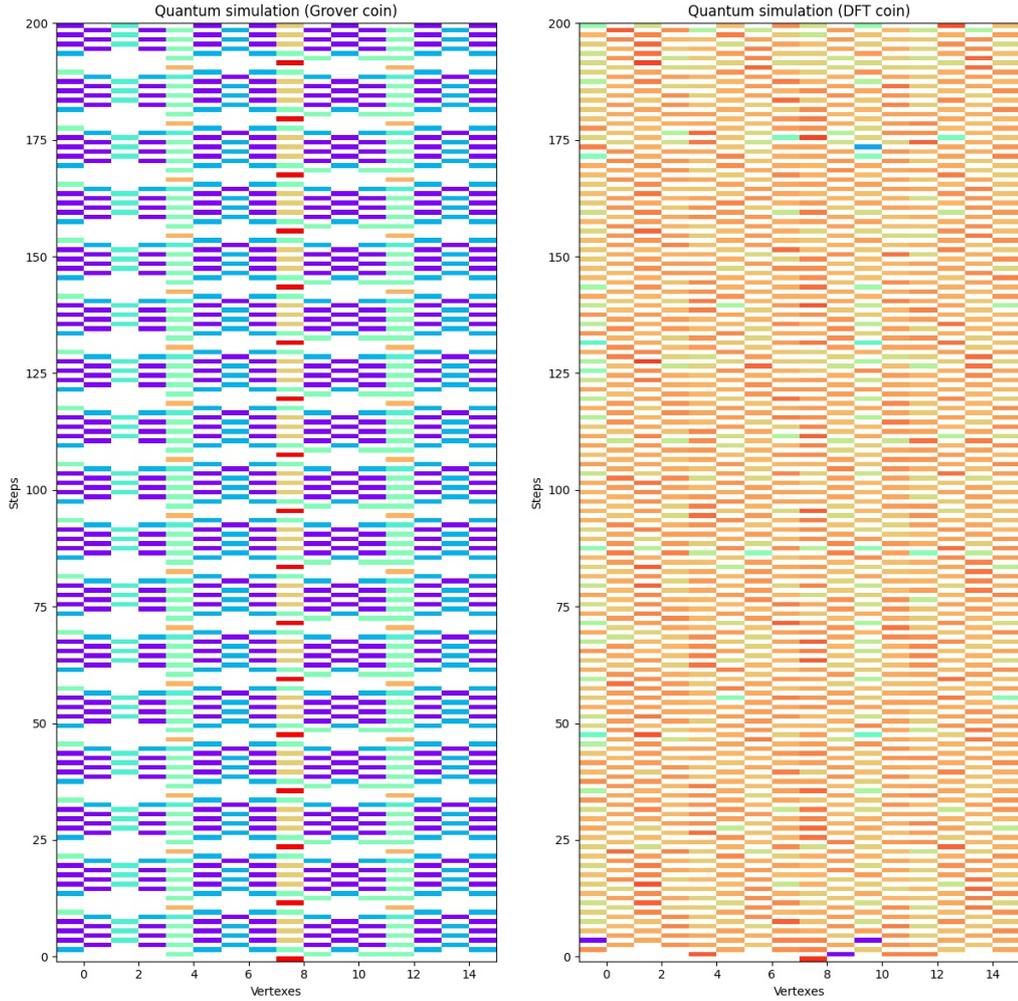


(a) Classical walk



(b) Quantum walk with the Hadamard coin

Figure 6.6: Probability distribution of classical and quantum walks on the grid



(a) Quantum walk with the Grover coin

(b) Quantum walk with the Fourier coin

Figure 6.7: Probability distribution of quantum walks on the grid

Interestingly, the hitting times of the 4 walks are similar. This is probably due to the fact, that the graph is small, which allows the classical walk to spread just as quickly as its quantum counterpart. We can see, that neither of the walks reached a stationary distribution. In the quantum case, we know that the walks are periodic, so there is no stationary distribution, while in the classical case, the graph is bipartite.

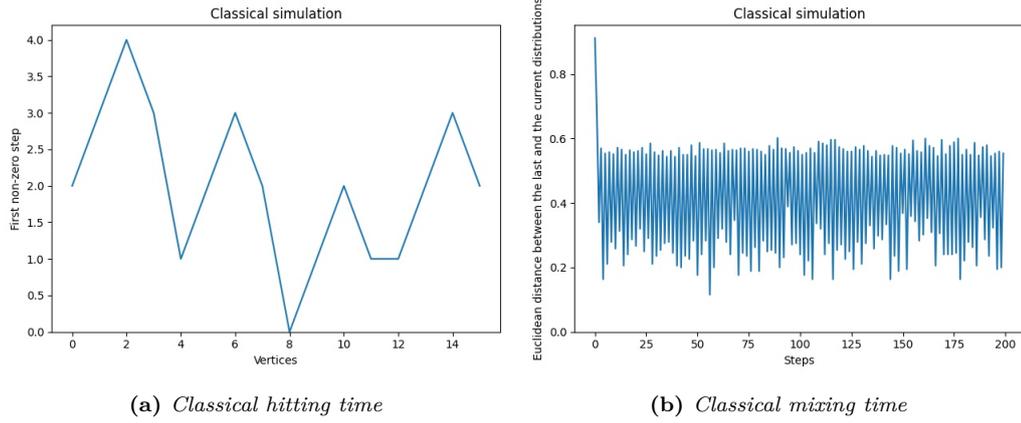


Figure 6.8: *Classical hitting and mixing times on the grid*

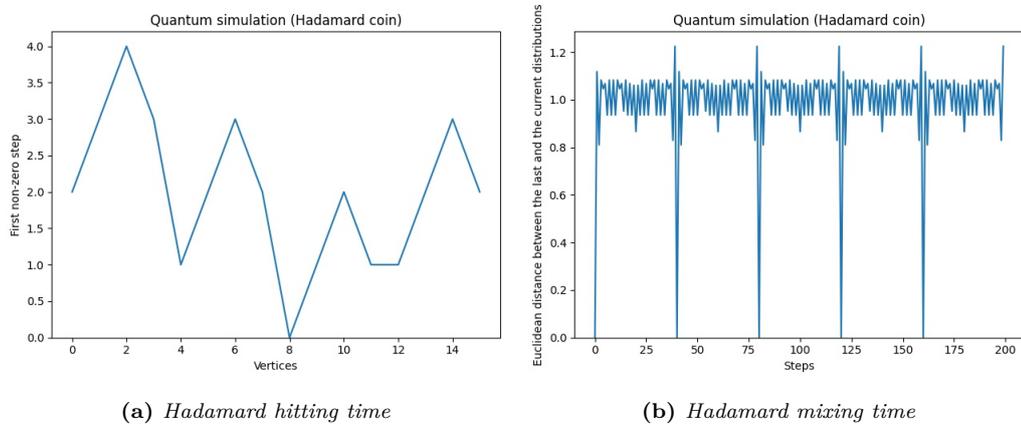


Figure 6.9: *Quantum (Hadamard) hitting and mixing times on the grid*

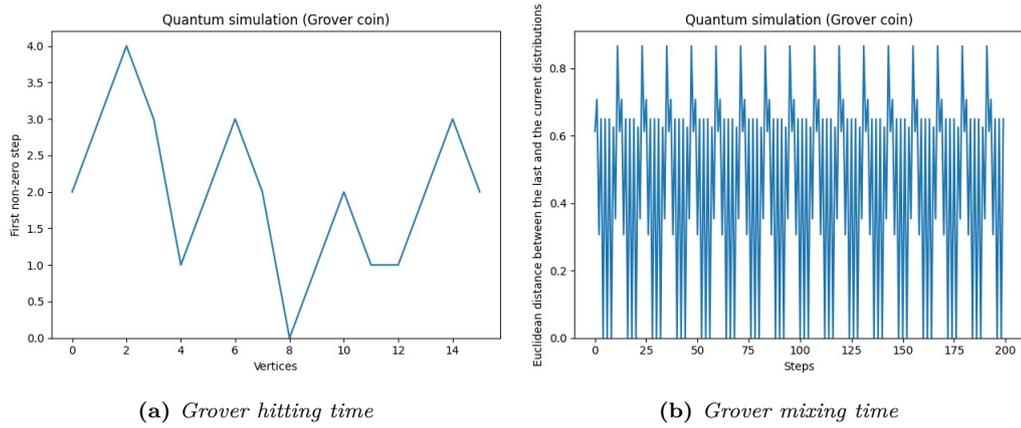


Figure 6.10: *Quantum (Grover) hitting and mixing times on the grid*

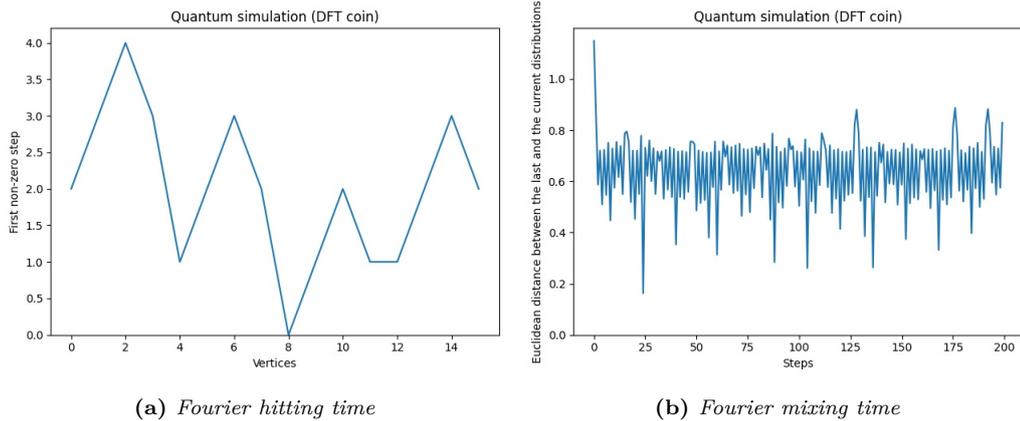


Figure 6.11: Quantum (*Fourier*) hitting and mixing times on the grid

6.3 Walks on hypercube

The third graph reviewed is the 4 dimensional boolean hypercube (with $2^4 = 16$ vertices), using the adjacency matrix below.

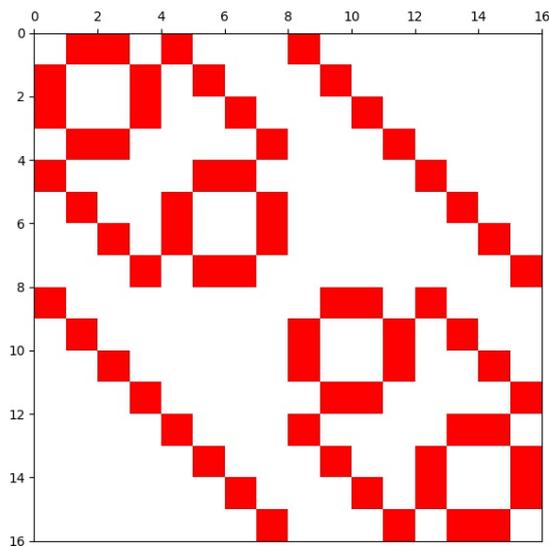
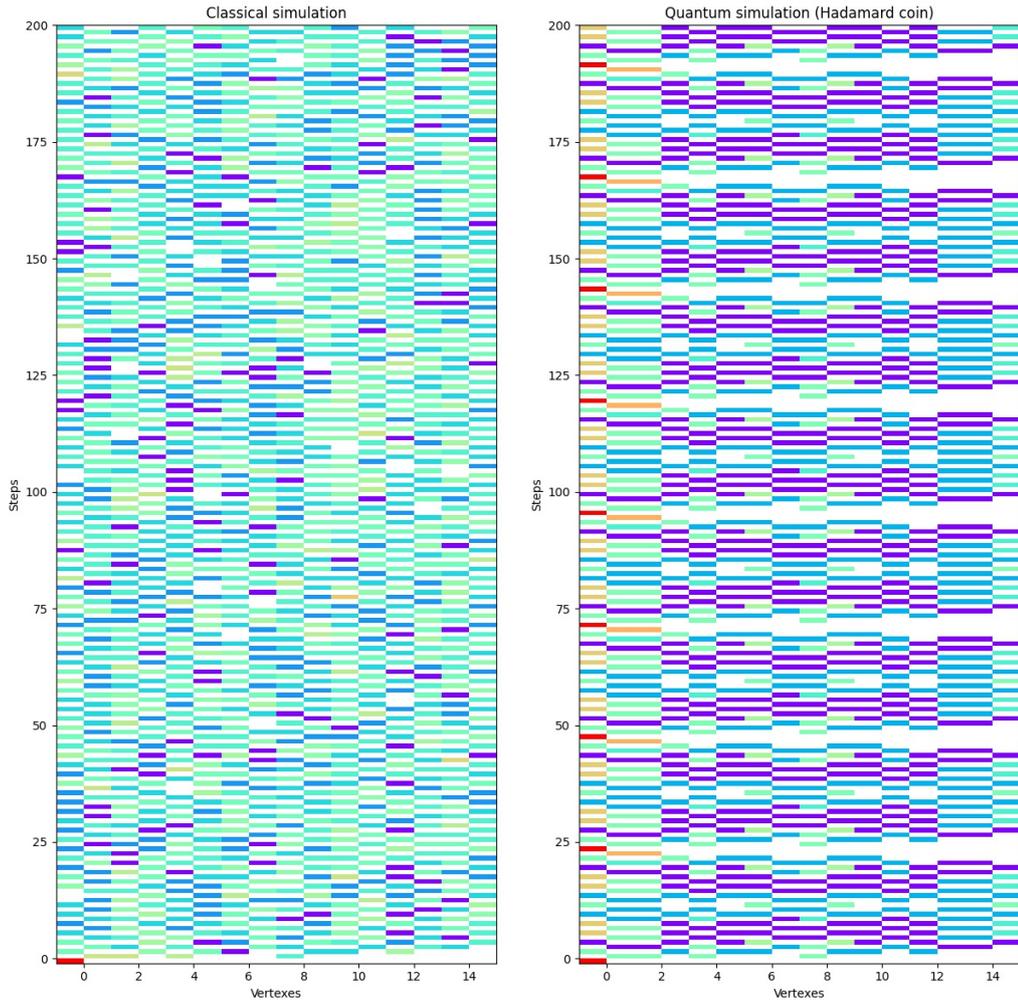


Figure 6.12: Adjacency graph of the hypercube

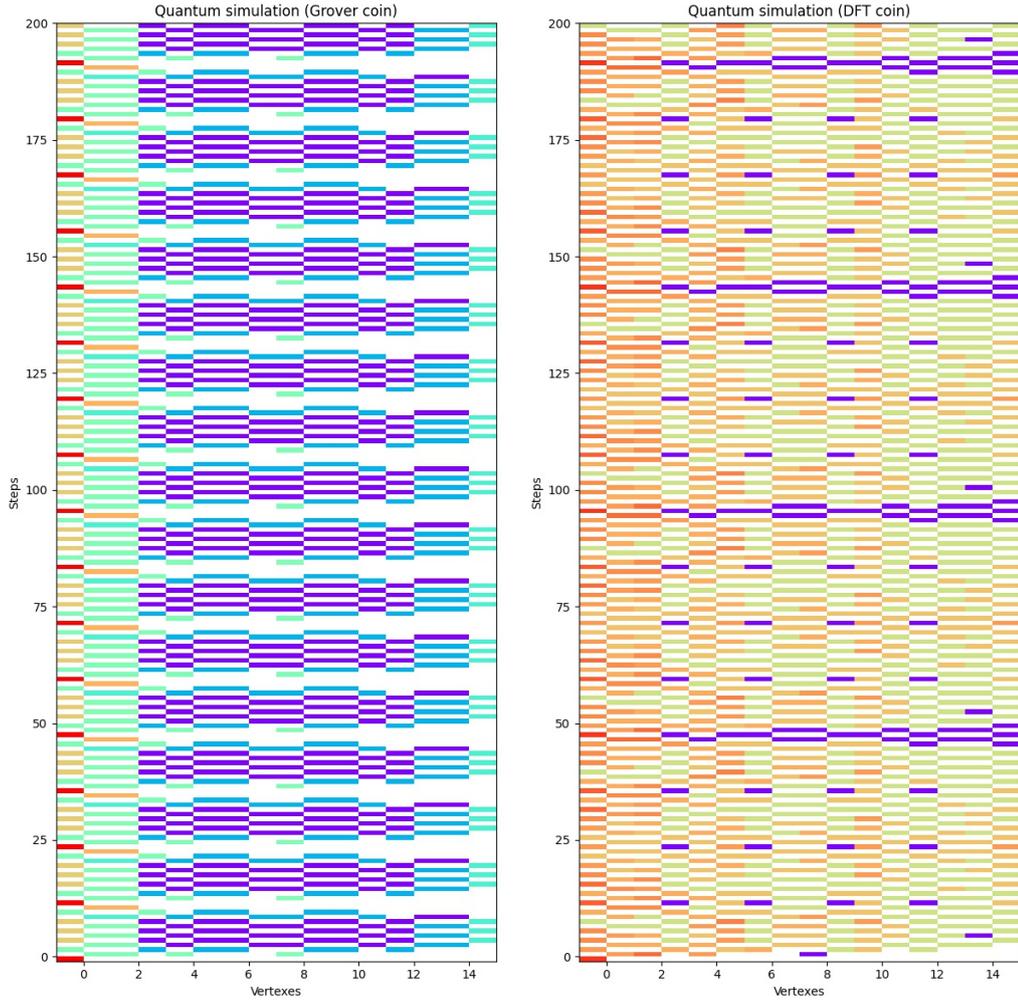
Similarly to the grid, the walks are recurrent (cyclic) in the Hadamard and Grover case. Interestingly in this case the periodicity can be visibly observed with the Fourier coin, however the walk does not return to its original starting point with 100% probability. This is due to the eigenvalues of the evolution operator just being slightly off, so there is no small exponent for which the evolution operator is the identity.



(a) Classical walk

(b) Quantum walk with the Hadamard coin

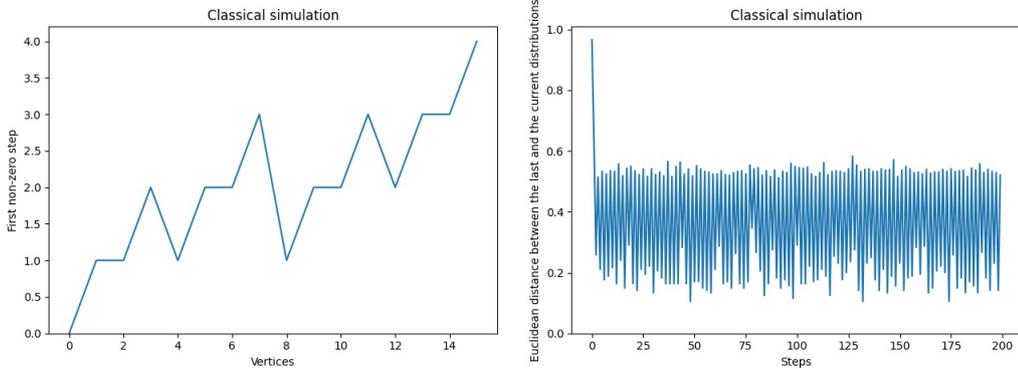
Figure 6.13: Probability distribution of classical and quantum walks on the hypercube



(a) Quantum walk with the Grover coin (b) Quantum walk with the Fourier coin

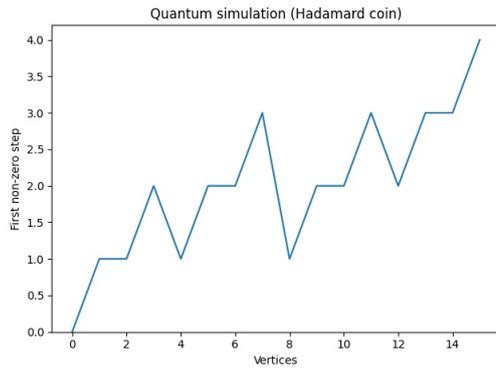
Figure 6.14: Probability distribution of quantum walks on the hypercube

Similarly to the grid, the hitting times are identical, since the graph is small and the classical walk’s disadvantage is not visible. The hypercube is also bipartite, resulting in no stationary distribution for the classical walk either.

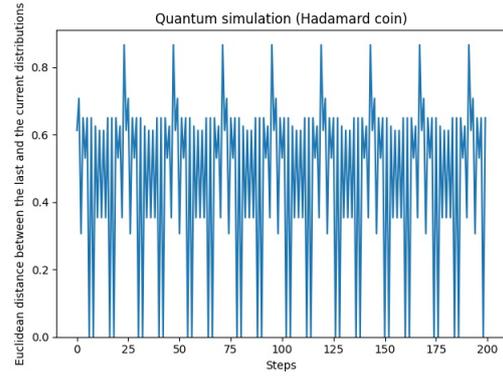


(a) Classical hitting time (b) Classical mixing time

Figure 6.15: Classical hitting and mixing times on the hypercube

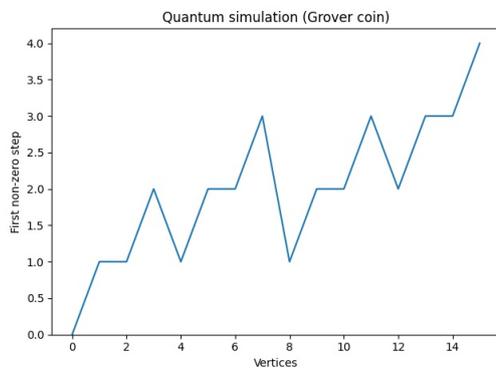


(a) Hadamard hitting time

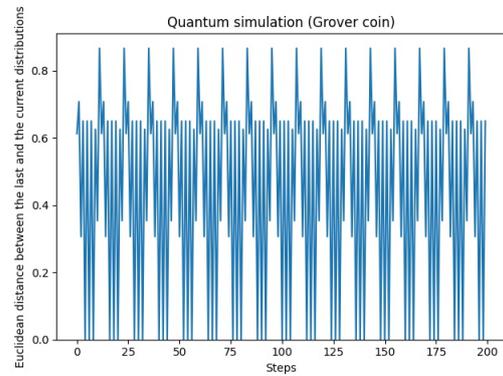


(b) Hadamard mixing time

Figure 6.16: Quantum (Hadamard) hitting and mixing times on the hypercube

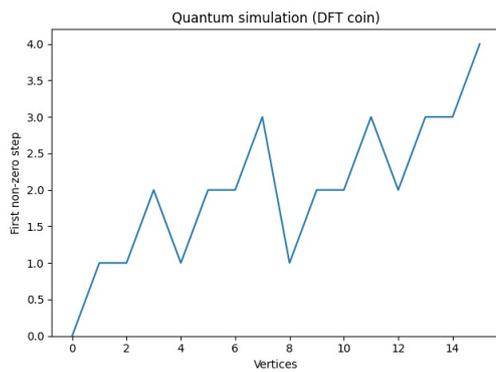


(a) Grover hitting time

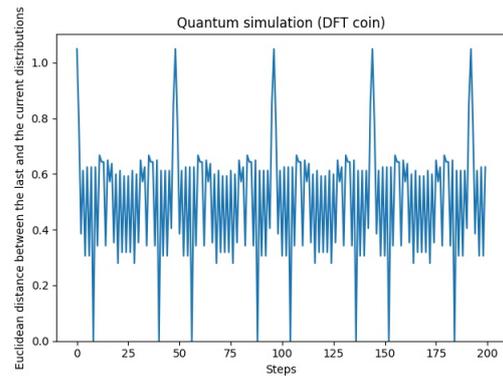


(b) Grover mixing time

Figure 6.17: Quantum (Grover) hitting and mixing times on the hypercube



(a) Fourier hitting time



(b) Fourier mixing time

Figure 6.18: Quantum (Fourier) hitting and mixing times on the hypercube

Chapter 7

Conclusion

During my research and development, I have read several excellent sources [1, 6, 9, 10, 12] and lecture notes, and found [9] to be a comprehensive general introduction. I reformulated its descriptions to the language of matrices in an explicit manner, which matches both classical random walk descriptions and universal quantum computing hardware requirements. I believe these are easier to understand for someone with a college-level software engineering background who is new to the subject.

Besides this, I gave a generalized requirement for constructing quantum random walks on d -regular graphs employing the position-coin notation and improved the memory requirement of n -dimensional lattice walks. I presented my proofs for both of these advancements.

Furthermore, I implemented a simulator software in Python, employing an architectural pattern that makes it straightforward to understand and extend the codebase. This software is available under the open-source MIT license on my personal Github account, under the following link:

<https://github.com/nemkin/quantum>

The software is still under heavy development. In the future, I would like to revise the report generation since the pdf format has proven to be too rigid. Instead of Latex, I believe a static website with an organized link hierarchy and the possibility for user interaction would prove much more helpful. Moreover, I am interested in extending my software's capabilities by implementing other quantum walk models. For example, inspired by the work of Ambainis, Szegedy [11] designed a different kind of approach to quantum walks by creating a general method to quantize classical Markov chains.

Furthermore, I plan to study applications of quantum walking, such as the MNRS quantum walk based search algorithm [7] which improves various aspects of many previous walk based algorithms. I aim to apply quantum walk based algorithms to bioinformatics research, such as medicine development.

Acknowledgements

Firstly, I would like to thank my advisor, dr. Katalin Friedl, for her commitment to spending a significant amount of time with me, discussing my research both during the school year and over the summer. Without her supervision and mathematical expertise, this effort would not have been achievable.

Secondly, I would like to thank the Quantum Information National Laboratory for providing me with a quantum research scholarship, which motivated me to work strenuously over the past months and submit this report to the Scientific Students' Associations Conference this year.

Bibliography

- [1] Yakir Aharonov, Luiz Davidovich, and Nicim Zagury. Quantum random walks. *Physical Review A*, 48:1687–1690, 1993.
- [2] Ferenc Balázs and Sándor Imre. *Quantum computing and communications: an engineering approach*. Wiley, Hoboken, N.J., 2013. ISBN 9781118725474.
- [3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley professional computing series. Addison-Wesley, 1995. ISBN 9780201633610.
- [4] Paul R. Halmos. *Finite-dimensional vector spaces*. Undergraduate texts in mathematics. Springer-Verlag, New York, 1974. ISBN 9780387900933.
- [5] Mika Hirvensalo. *Quantum computing*. Springer, Berlin; New York, 2001. ISBN 9783540407041.
- [6] Julia Kempe. Quantum random walks: An introductory overview. *Contemporary Physics*, 44(4):307–327, 2003.
- [7] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.
- [8] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, 2005. ISBN 9780521835404.
- [9] Renato Portugal. *Quantum Walks and Search Algorithms*. Quantum Science and Technology. Springer, Berlin; New York, 2018. ISBN 9783319978130.
- [10] Miklos Santha. Quantum walk based search algorithms. In Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation*, pages 31–46, Berlin, Heidelberg, 2008. Springer. ISBN 9783540792284.
- [11] Mario Szegedy. Quantum speed-up of markov chain based algorithms. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 32–41, Los Alamitos, CA, USA, 2004. IEEE Computer Society.
- [12] Feng Xia, Jiaying Liu, Hansong Nie, Yonghao Fu, Liangtian Wan, and Xiangjie Kong. Random walks: A review of algorithms and applications. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 4(2):95–107, 2020.