Features of Quantum Random Walks on Multiplex Networks and Other Topologies

Robert Speksnijder

Supervisors: Dr. J.L.A. Dubbeldam, Prof. dr. J.M. Thijssen

July 22, 2023

Abstract

Quantum random walks are the quantum analogs of classical random walks and appear to be promising tools to design fast quantum algorithms. Therefore it is important to study their time-related features and see how these differ compared to the classical case. For the discrete-time quantum walk on the line it has been shown that the probability to be absorbed by an absorbing boundary equals $\frac{2}{\pi}$ in contrast to the classical case where this probability equals 1, hence a quantum walk may continue forever without getting absorbed. It is also shown that mixing times of discrete-time quantum walks on the hypercube scale with n, the dimension of the hypercube, which is faster than $\mathcal{O}(n\log(n))$ for the classical random walk. So the quantum walk might offer a slight speed-up compared to the classical case. Finally, it is shown that the mixing times of the continuous-time quantum walk on a 2-layer multiplex graph depend on the eigenvalue gaps of the corresponding Laplacian matrix L. When the strength of the connections between the layers of a multiplex graph becomes very large, the eigenvalues of the Laplacian matrix converge. Thus the mixing times of the continuous-time quantum walk on 2-layer multiplex graphs converge.

Contents

1 Introduction

Classical random walks play an important role in theoretical computer science as well as in other disciplines such as physics, e.g. to simulate the motion of molecules, and financial theories, for example to predict stock prices. In computer science random walks find their use in a wide range of algorithms. They are used in PageRank algorithms[PBMW98] where random walks walk among web pages to calculate their importance. They also find their applications in algorithms that solve k-SAT problems[Sch99], which are problems about deciding whether Boolean variables can be assigned true or false values such that a Boolean expression of a specific form is true. For example, the problem of deciding whether the variables x_1, x_2, x_3 can be assigned truth values such that $(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\bar{x_1} \vee x_2)$ is true is a 2-SAT problem, since the Boolean expression is a conjunction of clauses that contain exactly 2 literals. Random walks are also implemented in algorithms that solve graph connectivity problems[MR95]. They can for example be used to decide whether there exists a path between two vertices of a given graph.

These are all examples of how classical random walks are used in algorithms implemented on classical computers. Over the last few decades however, the field of quantum information theory has seen a surge of interest. Much research is being done on designing efficient quantum algorithms to be implemented on quantum devices. To design these algorithms new sets of tools are needed. Quantum random walks appear to be one such set of tools that could be of great use in the setting of quantum computing.

Quantum random walks are the analogues of classical random walks. A particle walks on the vertices of a graph, but unlike the classical case the particle can be in a superposition of position states. There exist both discrete- and continuous-time versions of the walk. In both cases an initial particle state is transformed by a unitary operator; therefore the evolution of the particle state is deterministic. Randomness is only introduced through measurement of the particle state and thus this model is often simply called a 'quantum walk'.

Quantum walks have several features that indicate them to be useful tools to build quantum algorithms that can outdo their classical counterparts. One important feature is that the standard deviation of the position distribution of the quantum walk on a line is proportional to the elapsed time T, in contrast to the classical case where the standard deviation is proportional to $\sqrt{T}$[POR19]. Quantum walks thus travel quadratically faster. It has also been shown that the hitting time of a particle performing a quantum walk on a hypercube traveling from one corner to its opposite corner is polynomial in the dimension of the hypercube[Kem05]. The classical hitting time, on the other hand, is exponential in its dimension. The quantum random walk thus offers an exponential speed-up in this particular case.

An example of the implementation of quantum walks in quantum algorithms is a recently found spatial search algorithm that is quadratically faster than any classical algorithm[ACNR22]. There also exists a quantum version of the PageRank algorithm based on quantum walks[PMD11].

In this report temporal features of both discrete- and continuous-time quantum walks on different types of graphs, specifically multiplex graphs, are studied and compared to features of classical random walks.

The report can roughly be divided into three parts. In the first two chapters the discrete quantum walk on the line is defined and the probability that a particle performing a quantum walk gets absorbed by an absorbing boundary is studied. It turns out that a quantum walk on the line may be performed for an infinite amount of time without getting absorbed by the boundary. Then in chapter 4 the discrete quantum walk on graphs is defined and another time-related quantity is studied. In contrast to the absorption time, we would now like to study a quantity that is always finite. Hence mixing times of the quantum walk on the hypercube are considered. Finally, in chapter 5 we define the continuous-time quantum walk and study mixing times of the continuous quantum walk on multiplex graphs. In Appendix A the most important pieces of code can be found that are used in this report.

2 The Discrete Quantum Random Walk on the Line

In this chapter we will define the discrete quantum random walk on a line and compare it to the classical random walk. We will first recall the definition of the classical discrete random walk and then define the analogous quantum version. It will be explained that the discrete quantum walk describes a time evolution of a particle living in a Hilbert space $\mathcal{H} = \mathcal{H}_c \otimes \mathcal{H}_p$, the tensor product of the coin space and the position space. The basis states of the coin space $\mathcal{H}_c$ give the direction in which the particle wants to walk, left or right, and the basis states of the position space $\mathcal{H}_p$ give the position of the particle. The particle state evolves by repeatedly applying the operator $U = S \cdot (C \otimes I)$, which is the consecutive application of a coin operator, changing the direction of the particle, and the shift operator, moving the particle to the left or right based on its direction.

2.1 The Classical Random Walk

The classical discrete random walk on a line can be described as a particle living in a one-dimensional space, a line, moving randomly within this set over time[GA17]. We consider both time and space to be discrete, thus we observe the particle at points $0, 1, 2 \ldots$ in time, and for each point in time we consider the particle's location to be given by one of the elements of $\mathbb{Z}$, the integers. The particle moves over this line as follows. Given an initial position, at each time step the particle moves to the right with probability $0 < p < 1$ or to the left with probability $q = 1 - p$. So when S_t denotes the position of the particle at time t

$$S_{t+1} = \begin{cases} S_t + 1 & \text{with probability } p, \\ S_t - 1 & \text{with probability } 1 - p. \end{cases} \tag{2.1}$$

If $p = q = 1/2$ the random walk is called symmetric.

The probability distribution of the particle's location as a function of time can be computed and is plotted for different times in figure 1[1]. One can see that the distribution is centered around 0 and that the standard deviation in position scales with $\sqrt{t}$[POR19]. This is in contrast to the quantum case as we will see in the next section.

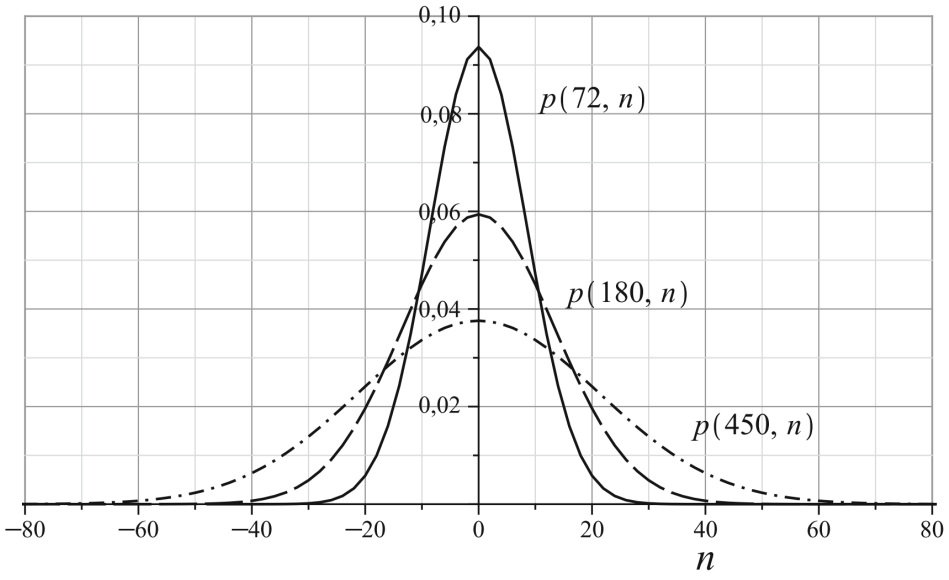

Figure 1: Figure taken from [POR19]. The probability that the particle performing the random walk is found at position n is plotted for different times. The probability distribution is seen to be centered around its initial position 0 and its standard deviation scales with $\sqrt{t}$. Since for even time steps the probability to be at odd positions is 0, the curves drawn are actually the envelopes of the probability distribution.

[1] Since the probability to be at positions of parity unequal to that of time t equals 0, the curves in the figure are actually the envelopes of the probability distributions.

2.2 The Discrete Quantum Walk

We will define the discrete quantum random walk analogously to the classical random walk. Again we consider discrete times and again the particle is located at discrete points in a one-dimensional space. So we define the Hilbert space $\mathcal{H}_p$ as the space spanned by basis states $\{|i\rangle : i \in \mathbb{Z}\}$ and call this space the position space. We would like the quantum random walk to incorporate quantum effects such as the particle being in a superposition of position states. Naively, we could therefore define the particle as living in $\mathcal{H}_p$ and define the quantum walk by the following transformation applied to every basis state of $\mathcal{H}_p$ for each time step

$$|i\rangle \to \frac{1}{\sqrt{2}}|i-1\rangle + \frac{1}{\sqrt{2}}|i+1\rangle . \tag{2.2}$$

However, one can see that this transformation is not unitary. The state $\frac{1}{\sqrt{2}}|-1\rangle + \frac{1}{\sqrt{2}}|1\rangle$ will for example be mapped to $\frac{1}{2}|-2\rangle + |0\rangle + \frac{1}{2}|2\rangle$, which is not normalized.

To define a quantum walk that is unitary, we need to introduce an extra degree of freedom that indicates in which direction the particle wants to walk. Let $\mathcal{H}_c$ be the Hilbert space, which we will call the coin space, spanned by the elements of the set $\{|\uparrow\rangle, |\downarrow\rangle\}$. Now the particle performing the walk lives in the Hilbert space $\mathcal{H} = \mathcal{H}_c \otimes \mathcal{H}_p$. So the state of the particle is given by a superposition of the basis states $\{|\uparrow\rangle \otimes |i\rangle, |\downarrow\rangle \otimes |i\rangle : i \in \mathbb{Z}\}$. A basis state $|\uparrow\rangle \otimes |i\rangle$ represents the particle being located at position i and wanting to take a step to the right. We thus define the unitary shift operator S as follows

$$S|\uparrow\rangle \otimes |i\rangle = |\uparrow\rangle \otimes |i+1\rangle , \quad S|\downarrow\rangle \otimes |i\rangle = |\downarrow\rangle \otimes |i-1\rangle . \tag{2.3}$$

Since we want the particle to walk in both directions, we need to apply another unitary operator before the shift operator that works on the coin space and sends each basis state of the coin space to a superposition of $|\uparrow\rangle$ and $|\downarrow\rangle$. We call such an operator a coin operator C in analogy to the coin-flip in the classical random walk. An example of a coin operator is the Hadamard coin defined by

$$H|\uparrow\rangle = \frac{1}{\sqrt{2}}|\uparrow\rangle + \frac{1}{\sqrt{2}}|\downarrow\rangle , \quad H|\downarrow\rangle = \frac{1}{\sqrt{2}}|\uparrow\rangle - \frac{1}{\sqrt{2}}|\downarrow\rangle , \tag{2.4}$$

or in matrix form when $|\uparrow\rangle = \begin{pmatrix}1\\0\end{pmatrix}$, $|\downarrow\rangle = \begin{pmatrix}0\\1\end{pmatrix}$

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}. \tag{2.5}$$

A full step of the quantum random walk is then defined by the transformation U given by

$$U|\psi\rangle = S \cdot (C \otimes I)|\psi\rangle . \tag{2.6}$$

When we for example start in the state $|\uparrow\rangle \otimes |0\rangle$ and apply the transformation U with H as coin operator C, we get for the first three time steps

$$\begin{aligned}
U|\uparrow\rangle \otimes |0\rangle &= S \cdot (C \otimes I)|\uparrow\rangle \otimes |0\rangle = S\frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle) \otimes |0\rangle = \frac{1}{\sqrt{2}}|\downarrow\rangle \otimes |-1\rangle + \frac{1}{\sqrt{2}}|\uparrow\rangle \otimes |1\rangle ,\\
U^2|\uparrow\rangle \otimes |0\rangle &= U\frac{1}{\sqrt{2}}(|\downarrow\rangle \otimes |-1\rangle + |\uparrow\rangle \otimes |1\rangle) = \frac{1}{2}(-|\downarrow\rangle \otimes |-2\rangle + (|\uparrow\rangle + |\downarrow\rangle) \otimes |0\rangle + |\uparrow\rangle \otimes |2\rangle),\\
U^3|\uparrow\rangle \otimes |0\rangle &= \frac{1}{2\sqrt{2}}(|\downarrow\rangle \otimes |-3\rangle - |\uparrow\rangle \otimes |-1\rangle + (2|\uparrow\rangle + |\downarrow\rangle) \otimes |1\rangle + |\uparrow\rangle \otimes |3\rangle).
\end{aligned} \tag{2.7}$$

Note that for the Hadamard coin U sends a basis state of $\mathcal{H}$ to the left and right with prefactors of equal magnitude squared, hence the probability to find the particle one step to the right of its initial position upon measurement is equal to the probability of finding it one step to the left of its initial position. This is similar to the symmetric classical random walk. A coin operator with this property of sending the basis states of $\mathcal{H}_c$ to a superposition of $|\uparrow\rangle, |\downarrow\rangle$ with prefactors of equal magnitude is called a balanced coin.

Even though we used a balanced coin, we can see that already after three steps the distribution is not symmetric anymore unlike the classical random walk. This asymmetry remains for larger time steps; in figure 2 the probability distribution of the location of the particle after 100 steps is shown.

The reason for this asymmetry is destructive interference caused by the coin operator. When H, see equation 2.5, is applied to $|\downarrow\rangle$ a minus sign is introduced, but when H is applied to $|\uparrow\rangle$ there is no change in phase. This causes more destructive interference for left-traveling paths, while it causes constructive interference for right-traveling paths. We can also see that the distribution is peaked towards the right end of the distribution. This peak is always present for large time scales. The particle is most likely to be found away from the origin as if it is traveling to the right; the quantum walk has a ballistic behaviour.

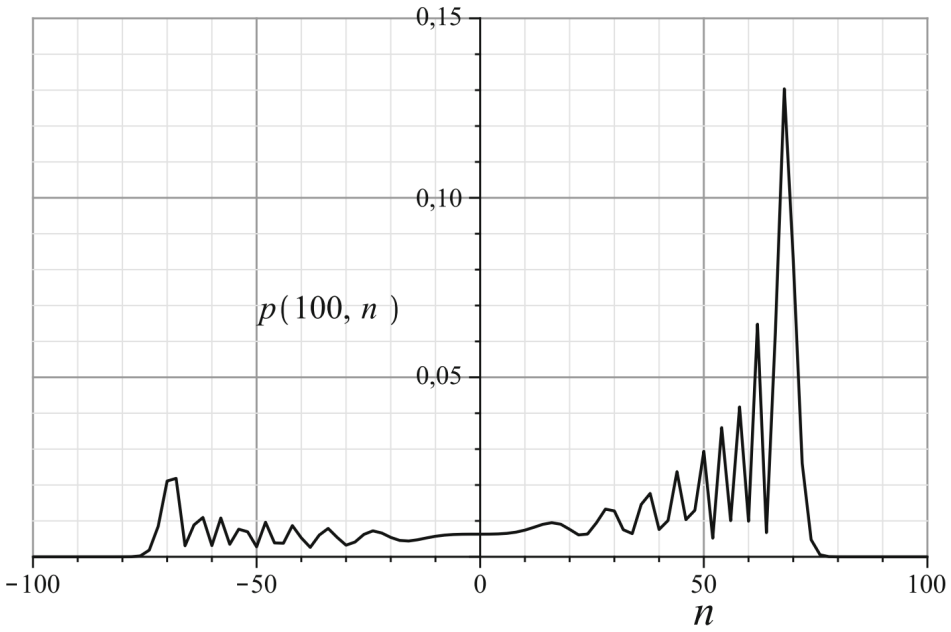

Figure 2: Figure taken from [POR19]. The probability distribution of the particle's position upon measurement after 100 time steps with initial state $|\uparrow\rangle \otimes |0\rangle$. The distribution is asymmetric unlike in the classical case.

We can make the distribution symmetric by changing the initial particle state to $\frac{|\uparrow\rangle - i|\downarrow\rangle}{\sqrt{2}} \otimes |0\rangle$. For 100 steps the distribution is plotted in figure 3. We now see two peaks, one at each end of the distribution. Similar to the asymmetric case, the quantum walk shows ballistic behaviour.

The standard deviation in position as a function of time can be computed and turns out to be proportional to T, the number of time steps, unlike the classical random walk that has standard deviation proportional to $\sqrt{T}$[POR19]. A particle performing a quantum walk thus travels away from the origin and spreads quadratically faster than the classical random walk.

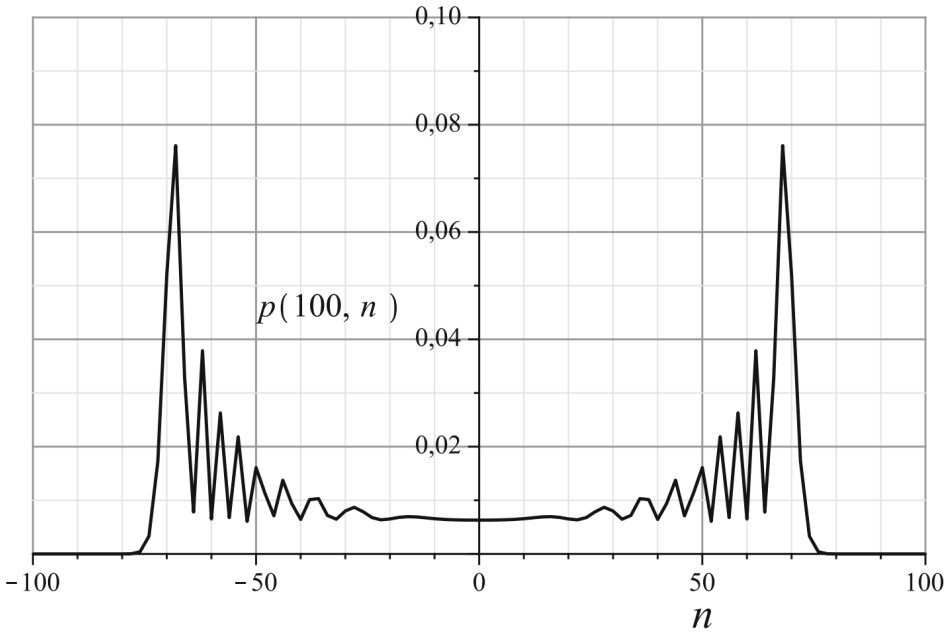

Figure 3: Figure taken from [POR19]. The probability distribution of the particle's position upon measurement after 100 time steps. The distribution is symmetric and has two peaks, one at the left edge and another at the right edge.

3 Absorption Probability of a Semi-Infinite Quantum Walk

In this chapter we will consider the probability that a particle performing a quantum walk on the line is eventually absorbed by an absorbing boundary placed at the origin. There is no absorbing boundary located to the right of the origin. The particle can travel infinitely far in one direction, but it can never reach negative locations. The quantum walk performed in this way is semi-infinite. It turns out that the absorption probability of the quantum walk equals $\frac{2}{\pi}$. The particle performing the quantum walk thus has a finite probability to escape the boundary. This is unlike the classical case where the absorption probability equals 1.

3.1 Absorption Probability of the Classical Random Walk

We consider a symmetric random walk with our particle initialized in position 1. The random walk terminates as soon as the particle reaches the origin. We want to find the probability

$$p_\infty = \text{probability that the particle gets absorbed at location 0 eventually.} \tag{3.1}$$

It turns out that $p_\infty = 1$ in the classical case as shown by the following proof that can be found in [Kem03]. Starting in 1 the particle has probability of $\frac{1}{2}$ to step to position 0, otherwise we step to position 2. The probability to reach position 0 from 2 eventually equals the probability to reach position 1 from 2 eventually times the probability to reach position 0 from 1 eventually. Since the probability to reach position 1 from 2 eventually equals p_∞, we get the following equation

$$p_\infty = \frac{1}{2} + \frac{1}{2}p_\infty^2. \tag{3.2}$$

Solving this equation gives us that $p_\infty = 1$ in the classical case. The particle cannot escape from the absorbing boundary.

3.2 Absorption Probability of the Quantum Walk

Suppose we start in the state $|\uparrow\rangle \otimes |1\rangle$. One step of the quantum walk is now defined by applying evolution operator U, see equation 2.6, with the Hadamard coin, see equation 2.4, as coin operator and a subsequent local measurement in the origin on the state $|\downarrow\rangle \otimes |0\rangle$[2]. When the particle is found in the origin, the walk terminates. When the particle is not found in the origin, we again apply U and perform a measurement in the origin.

We want to find the probability

$$p_\infty = \text{probability that the particle gets absorbed at location } |0\rangle \text{ eventually.} \tag{3.3}$$

In [ABN+01] the following theorem is proven.

Theorem 3.1. $p_\infty = \frac{2}{\pi}$

The proof has been replicated below. The result has also been numerically verified with the code in Appendix A.1.

Proof. It must hold that

$$p_\infty = \sum_{t=1}^{\infty} \left\| \Pi U[(I-\Pi)U]^{t-1} |\uparrow\rangle \otimes |1\rangle \right\|^2, \tag{3.4}$$

where $\Pi = |\downarrow\rangle \otimes |0\rangle \langle\downarrow| \otimes \langle 0|$. The t^{th} term of this sum equals the probability that after t time steps the particle is absorbed for the first time.

To find p_∞ we consider all paths via which we can walk to $|\downarrow\rangle \otimes |0\rangle$. Every path can be represented by a tuple $(a_1, a_2, \ldots, a_t)$ with $a_i \in \{-1, 1\}$ and $\sum_{i \le j} a_i \ge 0$ for $j < t$ and $\sum_{i=1}^{t} a_i = -1$. So all paths of $t \in \mathbb{N}$ time steps are represented in

$$A_t = \{(a_1, a_2, \ldots, a_t) \in \{-1,1\}^t : \text{ for all } 0 \le j < t, \sum_{i=1}^{j} a_i \ge 0 \text{ and } \sum_{i=1}^{t} a_i = -1\}. \tag{3.5}$$

[2]Note that it is not necessary to measure the state $|\uparrow\rangle \otimes |0\rangle$, since this state can only be reached from location $|-1\rangle$.

We want to partition each A_t into the paths that correspond to $|\downarrow\rangle \otimes |0\rangle$ getting a positive prefactor A_t^+ and those that give $|\downarrow\rangle \otimes |0\rangle$ a negative prefactor A_t^-. The paths in A_t^+ will have prefactor $\left(\frac{1}{\sqrt{2}}\right)^t$ since we apply the Hadamard coin operator at each step. The elements in A_t^- will have prefactor $-(\frac{1}{\sqrt{2}})^t$. We can then write the absorption probability as the following infinite series

$$p_\infty = \sum_{t=1}^{\infty} \frac{(\#(A_t^+) - \#(A_t^-))^2}{2^t}. \tag{3.6}$$

This follows from the fact that we have to sum over all possible number of steps to reach the barrier and that the paths with positive and negative prefactors interfere destructively. If we define

$$A_t^+ = \{x = (a_1, \ldots, a_t) \in A_t : \#\{i : a_i = a_{i+1} = -1\} \text{ is even}\} \tag{3.7}$$

and A_t^- as

$$A_t^- = \{x = (a_1, \ldots, a_t) \in A_t : \#\{i : a_i = a_{i+1} = -1\} \text{ is odd}\} \tag{3.8}$$

we see that we indeed get our desired partition.

We attempt to find the sum in equation 3.6 by defining the following generating function for $(\#(A_t^+) - \#(A_t^-))$

$$f(z) = \sum_{t=1}^{\infty} (\#(A_t^+) - \#(A_t^-)) z^t. \tag{3.9}$$

We will now show that

$$f(z) = z - z(zf(z) + (zf(z))^2 + (zf(z))^3 + \cdots), \tag{3.10}$$

by showing that $-z(zf(z))^k$ for $k \geq 1$ equals an analogous generating function. Namely,

$$-z(zf(z))^k = \sum_{t=1}^{\infty} c_t z^t = \sum_{t=1}^{\infty} (\#(A_{t,k}^+) - \#(A_{t,k}^-)) z^t. \tag{3.11}$$

$A_{t,k}^+$ here equals the set containing all paths, t steps long, starting off in the state $|\uparrow\rangle \otimes |1\rangle$ and ending in state $|\downarrow\rangle \otimes |0\rangle$, such that our final state has a positive prefactor and such that we visit $|1\rangle$ k times after leaving our initial position. $A_{t,k}^-$ contains all paths of t steps from our initial state to $|\downarrow\rangle \otimes |0\rangle$, such that we visit our initial position k times, but this time our final state must have negative prefactor.

We will prove equation 3.11. First we write out $-z(zf(z))^k$,

$$-z(zf(z))^k = \sum_{t_1=1}^{\infty} \sum_{t_2=1}^{\infty} \cdots \sum_{t_k=1}^{\infty} -(\#(A_{t_1}^+) - \#(A_{t_1}^-))(\#(A_{t_2}^+) - \#(A_{t_2}^-)) \cdots (\#(A_{t_k}^+) - \#(A_{t_k}^-)) z^{t_1+t_2+\cdots+t_k+k+1}. \tag{3.12}$$

For this expression to be equal to the right side of equation 3.11, we need to show that the coefficients in equation 3.12 equal the coefficients c_t. Firstly, we need to show that $c_t = 0$ for $t \leq 2k$. This is indeed the case, because we have to step from $|1\rangle$ to $|2\rangle$ and back again exactly k times and after our kth visit to $|1\rangle$ we have to go to $|0\rangle$. So there are no paths of length shorter than $2k+1$, hence our coefficients c_t for $t \leq 2k$ are zero. For $t \geq 2k+1$ we want that

$$c_t = \#(A_{t,k}^+) - \#(A_{t,k}^-) = \sum_{\substack{t_1, t_2, \cdots, t_k \\ \sum t_j = t-k-1}} -(\#(A_{t_1}^+) - \#(A_{t_1}^-))(\#(A_{t_2}^+) - \#(A_{t_2}^-)) \cdots (\#(A_{t_k}^+) - \#(A_{t_k}^-)). \tag{3.13}$$

We will show that equation 3.13 is true for $k = 1$ first. We will show that

$$\#(A_{t,1}^+) - \#(A_{t,1}^-) = -(\#(A_{t-2}^+) - \#(A_{t-2}^-)). \tag{3.14}$$

For $k = 1$, we can split each path up in three parts. The first step from $|1\rangle$ to $|2\rangle$, the middle part which consists of a path in A_{t_1} with $t_1 = t-2$ from $|2\rangle$ to $|1\rangle$, and the last step from $|1\rangle$ to $|0\rangle$. Note that for the path to end in $|\downarrow\rangle \otimes |0\rangle$ with positive prefactor, the middle part of the path must be an element

of $A^-_{t_1}$, since the last step reverses the sign. For the path to end with a negative prefactor, the middle part of the path must be an element of $A^+_{t_1}$. So indeed $\#(A^+_{t,1}) = \#(A^-_{t-2})$ and $\#(A^-_{t,1}) = \#(A^+_{t-2})$.

We will now prove equation 3.13 for general k. A path from $|1\rangle$ to $|0\rangle$, that visits $|1\rangle$ k times after leaving can be seen as composed of $2k+1$ paths. Namely, 1 step from $|1\rangle$ to $|2\rangle$, followed by a path of length t_1 back to $|1\rangle$, again 1 step from $|1\rangle$ to $|2\rangle$, followed by a path of length t_2 back to $|1\rangle$ and so on, until we visit $|1\rangle$ for the kth time followed by the last step to $|0\rangle$. Note that $\sum t_j = t-k-1$ must hold. Every path that steps back from $|2\rangle$ to $|1\rangle$ in t_i steps can be seen as an element of either $A^+_{t_i}$ or $A^-_{t_i}$. If the number of paths stepping back from $|2\rangle$ to $|1\rangle$ that reverses sign is odd, the total path will end with positive prefactor, since the last step to $|0\rangle$ reverses the sign. So the number of paths in $\#(A^+_{t,k})$ equals the sum over all products $\prod_{\sum t_j = t-k-1}^{t_1,\dots,t_k} \#(A^{+/-}_{t_j})$ with odd number of factors $\#(A^-_{t_j})$. In equation 3.12 every such product is included with positive sign. Similarly, the number of paths in $\#(A^-_{t,k})$ equals the sum over all products $\prod_{\sum t_j = t-k-1}^{t_1,\dots,t_k} \#(A^{+/-}_{t_j})$ with even number of factors $\#(A^-_{t_j})$. All these products are included in equation 3.12 with negative sign. Since there are no other terms included, the equality of equation 3.11 must hold.

Now we can show that equation 3.10 holds by writing

$$f(z) = (\#(A^+_1) - \#(A^-_1))z + \sum_{t=2}^{\infty}(\#(A^+_t) - \#(A^-_t))z^t = z + \sum_{t=2}^{\infty}\sum_{k=1}^{\infty}(\#(A^+_{t,k}) - \#(A^-_{t,k}))z^t$$
$$= z + \sum_{t=1}^{\infty}\sum_{k=1}^{\infty}(\#(A^+_{t,k}) - \#(A^-_{t,k}))z^t = z + \sum_{k=1}^{\infty}\sum_{t=1}^{\infty}(\#(A^+_{t,k}) - \#(A^-_{t,k}))z^t = z + \sum_{k=1}^{\infty} -z(zf(z))^k. \tag{3.15}$$

This of course equals

$$f(z) = z + -z^2 f(z)\frac{1}{1-zf(z)}. \tag{3.16}$$

Solving for $f(z)$ gives

$$f(z) = \frac{1+2z^2-\sqrt{1+4z^4}}{2z}. \tag{3.17}$$

Now this expression is very similar to the generating function of the Catalan numbers. So

$$\#(A^+_t) - \#(A^-_t) = \begin{cases} 1 & \text{if } t = 1, \\ (-1)^{k+1}C_k & \text{t} = 4k+3, \\ 0 & \text{else}, \end{cases} \tag{3.18}$$

where $C_k = \frac{k}{k+1}\binom{2k}{k}$ is the kth Catalan number.

We can use this to rewrite the sum in equation 3.6 as follows

$$p_\infty = \sum_{t=1}^{\infty}\frac{(\#(A^+_t) - \#(A^-_t))^2}{2^t} = \frac{1}{2} + \frac{1}{8}\sum_{k=0}^{\infty} C_k^2 2^{-4k}. \tag{3.19}$$

It can be proved by induction that

$$\sum_{k=0}^{N} C_k^2 2^{-4k} = (16N^3 + 36N^2 + 24N + 5)C_N^2 2^{-4N} - 4. \tag{3.20}$$

Since $C_N(\frac{2^{2N}}{N^{3/2}\sqrt{\pi}})^{-1} \longrightarrow 1$ as N goes to infinity, we get

$$\sum_{k=0}^{\infty} C_k^2 2^{-4k} = \frac{16}{\pi} - 4. \tag{3.21}$$

Plugging this into equation 3.19 we find the desired result

$$p_\infty = \frac{2}{\pi}. \tag{3.22}$$

□

4 The Discrete Quantum Random Walk on the Hypercube

In this chapter we will define the discrete quantum walk on graphs and subsequently perform the quantum walk on the hypercube. Whereas in the last chapter we studied a temporal quantity that may be infinite, namely the time it takes for a particle to be absorbed by an absorbing boundary, we would now like to study a quantity that is always finite. Hence the mixing time on the hypercube is considered. It is seen that this quantity scales linearly with the dimension n of the hypercube, unlike the classical case where the mixing time is of order $\mathcal{O}(n \log n)$[MPAD08].

4.1 The Discrete Quantum Walk on a Graph

In this section we define the quantum random walk on a graph. Similar to the case of the quantum random walk on a line, our particle lives in a Hilbert space that is a tensor product of a coin and position space $\mathcal{H}_C \otimes \mathcal{H}_P$. The position space $\mathcal{H}_P$ is spanned by the basis states giving the location of the particle and the coin space $\mathcal{H}_C$ is spanned by the basis states that can be thought of as giving the direction in which the particle wants to walk. Again, given an initial state, the quantum walk is performed by repeatedly applying a coin operator and then a shift operator to the particle state. Let us define this more formally.

Consider a d-regular graph. Let $\mathcal{H}_P$ be the position Hilbert space spanned by the vertices of the graph. Thus $\{|v_i\rangle : i = 1, 2, \ldots, N\}$ spans $\mathcal{H}_P$ for a graph with N vertices.

To define the coin space, for each vertex, we label the d incident edges by $1, 2, \ldots, d$ on the side of the vertex. So each edge has two, possibly distinct, labels, one for each of its endpoints, and for each vertex the labels of the incident edges are all distinct. See figure 4 for an example of a correct labeling[3]. Now, when the particle is at a certain vertex, given by a basis state of $\mathcal{H}_P$, and it wants to walk in one of the d directions, all we have to do is specify the label corresponding to that direction. Thus we define the coin space $\mathcal{H}_C$ to be the Hilbert space spanned by the set of the d labels $\{|j\rangle : j = 1, \ldots, d\}$.

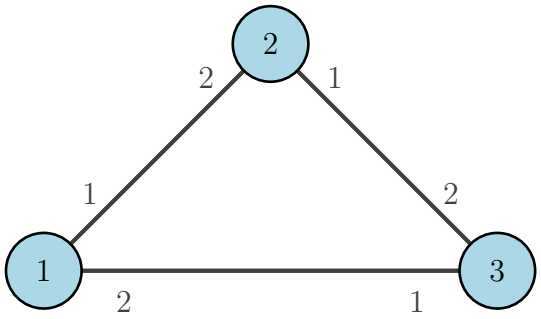

Figure 4: This figure shows a graph with a correct labeling. When our particle is at a certain vertex, say vertex 1 and wants to walk to vertex 3, we say that it is in the state $|2\rangle \otimes |1\rangle$, since the label of the edge pointing in the direction of vertex 3 has label 2 on the side of vertex 1.

As mentioned above, our particle performing the quantum random walk lives in the Hilbert space $\mathcal{H} = \mathcal{H}_C \otimes \mathcal{H}_P$. A basis state $|j\rangle \otimes |v\rangle$ corresponds to the particle being at vertex v, pointing in the direction of the edge with label j on v's side.

Let us call e_v^j the edge (v, w) with label j on v's side. We can now define the shift operator S by

$$S\,|j\rangle \otimes |v\rangle = |j\rangle \otimes |w\rangle\,, w \in e_v^j. \tag{4.1}$$

When the particle is originally at vertex v it moves along the edge labeled with label j to vertex w.

The coin operator C can be any d-dimensional unitary operator acting on the coin space. An example of a balanced coin, a coin that gives equal probability for the particle to be found in each of

[3]It must be noted that there is an extra condition on the labeling such that a quantum random walk can be performed. This issue is addressed in Appendix B.

the d directions upon measurement, is the DFT-coin defined as:

$$DFT\text{-}coin = \frac{1}{\sqrt{d}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{d-1} \\ 1 & \omega^2 & \omega^{2(2)} & \dots & \omega^{2(d-1)} \\ & & \vdots & & \\ 1 & \omega^{d-1} & \omega^{2(d-1)} & \dots & \omega^{(d-1)(d-1)} \end{pmatrix}, \tag{4.2}$$

where $\omega = e^{\frac{2\pi i}{d}}$ and d the degree of the regular graph. One can clearly see that every coin basis state $|j\rangle$ will be sent to an equally weighted superposition of all coin basis states.

The quantum random walk of T steps is defined by the transformation U^T, where

$$U = S \cdot (C \otimes I) \tag{4.3}$$

is the consecutive application of the coin and shift operator.

In this paragraph we have defined the quantum random walk on regular graphs. We can easily perform quantum random walks on graphs that are not regular by adding self-loops to the vertices that have a degree smaller than the maximum degree of the graph, see figure 5.

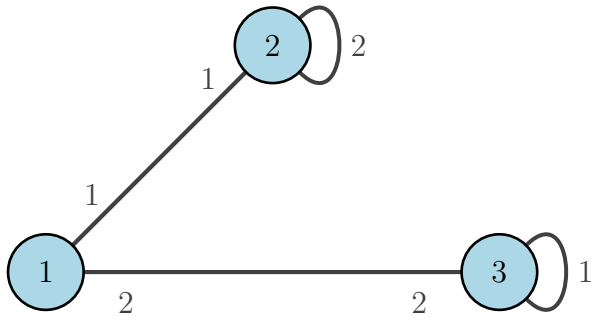

Figure 5: A graph with a correct labeling is shown. Self-loops were added to vertices 2 and 3 such that for every vertex there are $d = 2$ directions.

4.2 Quantum Random Walk on a Hypercube

In this section, we will look at the quantum random walk on a hypercube. An n-dimensional hypercube graph is a simple graph that has 2^n vertices. Each vertex corresponds to an n-bit string, see figure 6. The Hamming weight of a vertex is the number of ones in its n-bit string. The Hamming distance between two vertices equals the number of bits that one needs to flip to convert the n-bit string of one vertex into the n-bit string of the other. Two vertices are connected by an edge if their Hamming distance equals 1. Note that the n-dimensional hypercube is n-regular. In figure 6, a 3D-hypercube is shown.

To define a quantum random walk on a hypercube, we have to label all the edges of the hypercube at both its endpoints, such that for each vertex all incident edges have a distinct label. It turns out that in the special case of a hypercube we can label each edge by one label, i.e. both endpoints get the same number. The labeling is defined by the following rule: when an edge connects two vertices whose n-bit strings differ in position i, assign this edge the label i.

A Python script was written that could perform quantum random walks on general n-dimensional hypercubes, see Appendix A.2. In figure 7 the 2D-hypercube for 8 steps of the quantum random walk is shown, where the vertex colour indicates the probability to find the particle at that vertex upon measurement of the particle location. The darker the vertex, the more likely it is to find the particle there. The initial state of the particle was $\frac{1}{\sqrt{2}}(|1\rangle \otimes |(00)\rangle + |2\rangle \otimes |(00)\rangle)$. Looking at the figure, one might expect that the particle state oscillates over time and has period 8. This turns out to be indeed the case. By writing the shift and coin operators as matrices, we can write their consecutive application

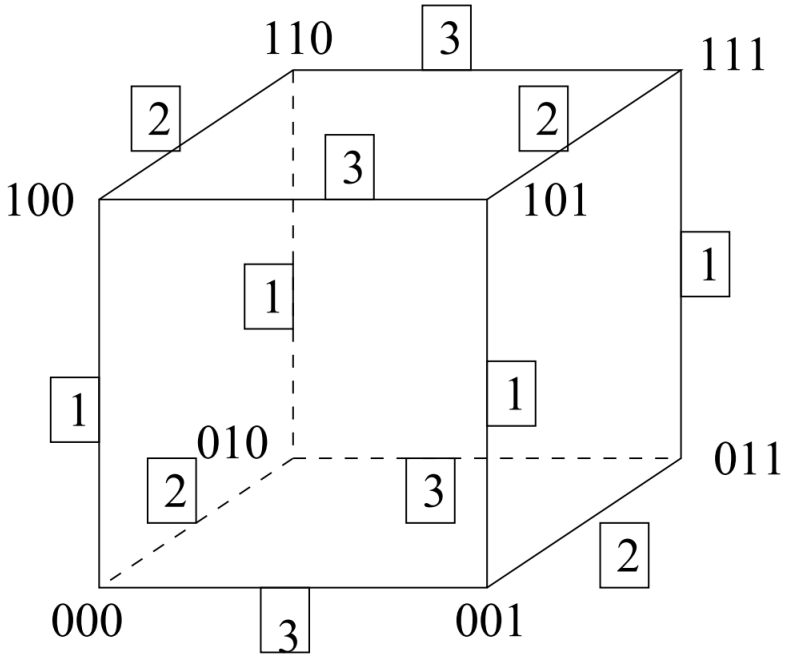

Figure 6: This figure was taken from [p. 8][Kem03]. It shows a 3D-hypercube where each vertex is labeled by a 3-bit string. The edges between the vertices are labeled with the numbers $1, 2, 3$. An edge is labeled with the number i, if the edge connects two vertices whose bit strings differ only in the ith position.

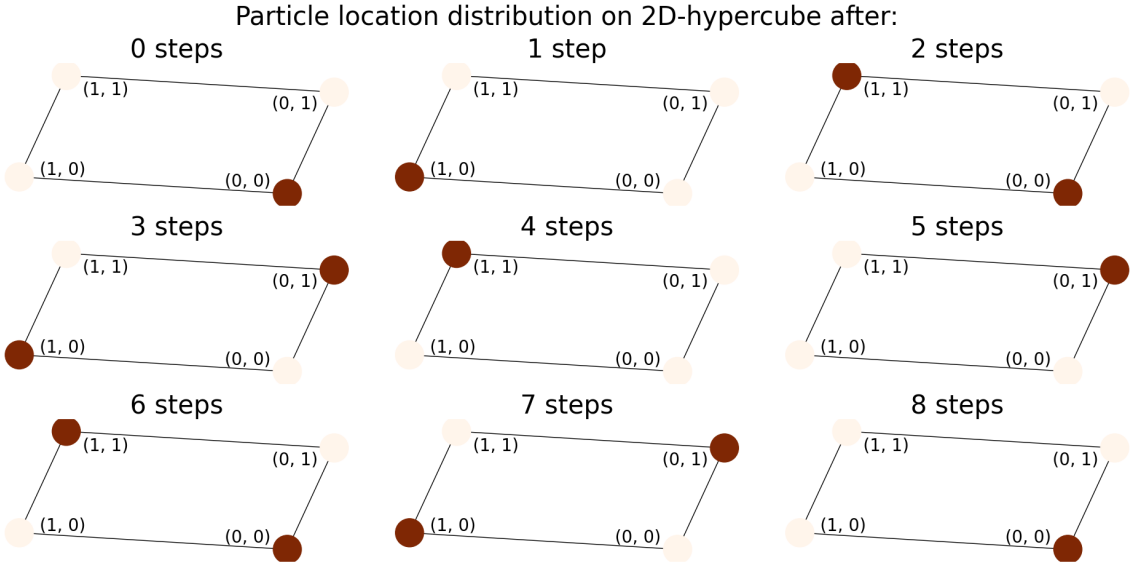

Figure 7: A quantum random walk on a 2D-hypercube was performed with the DFT-coin as coin operator. The probability to find the particle at a certain vertex is indicated by the intensity of the colour of the vertex. The darker the vertex, the more likely it is to find the particle there. The initial position and the first 8 steps are shown. One can see the oscillatory behaviour of the quantum random walk on the hypercube. The particle state oscillates with period 8.

as a matrix as well. By using Mathematica the following eigenvalues for this larger matrix were found $\{-\frac{1-i}{\sqrt{2}}, -\frac{1+i}{\sqrt{2}}, \frac{1+i}{\sqrt{2}}, \frac{1-i}{\sqrt{2}}, -1, -1, 1, 1\}$. Now since our initial state can be written as a superposition of eigenvectors $\sum_{i=1}^{t} c_i \vec{v}_i$, after $8k$ steps we end up in state $\sum_{i=1}^{t} \lambda^{8k} c_i \vec{v}_i = \sum_{i=1}^{t} c_i \vec{v}_i$. If we start our

quantum walk in the nodes *(0,0)* and *(1,1)*, so in the state $\frac{1}{\sqrt{4}}(|1\rangle\otimes|(0,0)\rangle+|2\rangle\otimes|(0,0)\rangle+|1\rangle\otimes|(1,1)\rangle+|2\rangle\otimes|(1,1)\rangle)$, we can even get a quantum walk oscillating with a period of 2, since the eigenvectors of this initial state correspond to the eigenvalues 1 and -1.

One can also perform the quantum walk on the hypercube with different coin operators. One such coin operator is the Grover coin.

$$\textit{Grover-coin} = \begin{pmatrix} \frac{2}{d}-1 & \frac{2}{d} & \frac{2}{d} & \cdots & \frac{2}{d} \\ \frac{2}{d} & \frac{2}{d}-1 & \frac{2}{d} & \cdots & \frac{2}{d} \\ & & \vdots & & \\ \frac{2}{d} & \frac{2}{d} & \frac{2}{d} & \cdots & \frac{2}{d}-1 \end{pmatrix}, \tag{4.4}$$

where d is the degree of the regular graph. The Grover coin is permutation symmetric, which means that interchanging the vertices of equal Hamming distance does not change the random walk.

In figure 8 we perform a quantum walk on a 2D-hypercube with the Grover-coin. The probability distribution of the location of a particle upon measurement is shown for 8 steps of the quantum walk. The initial state is the same as in figure 7. As expected by the permutation symmetry of the Grover coin, vertices of equal Hamming weight have equal probability to find the particle on them. For this case one can see that the period of the quantum walk is 4, since the eigenvalues are given by $\{-1,-1,-i,-i,i,i,1,1\}$.

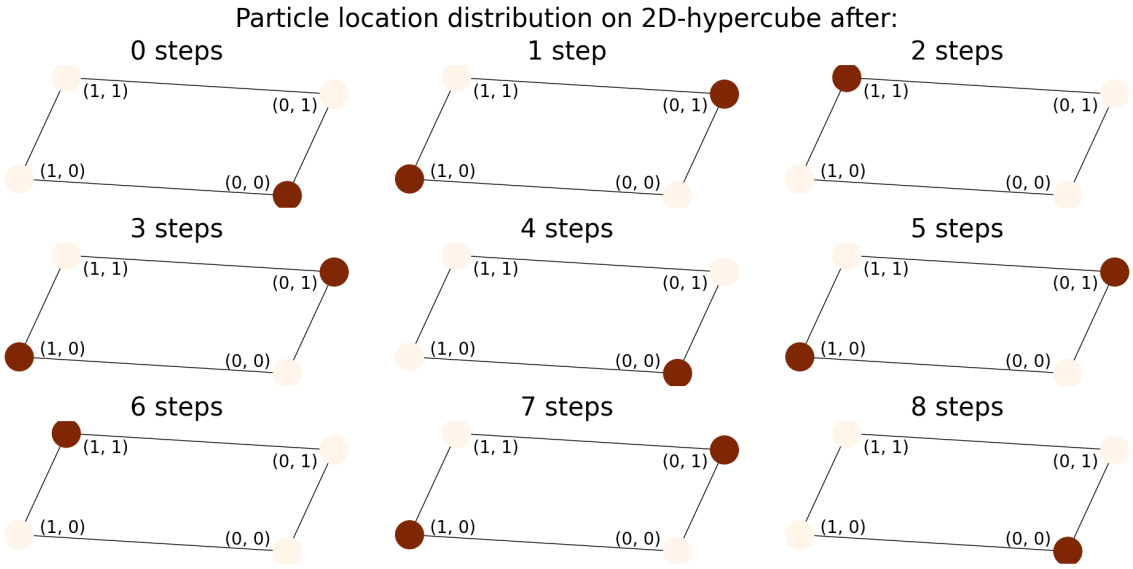

Figure 8: A quantum random walk on a 2D-hypercube, now performed with the Grover-coin as coin operator. The probability to find the particle at a certain vertex is again indicated by the intensity of the colour of the vertex. The darker the vertex, the more likely it is to find the particle there. The initial position and the first 8 steps are shown. The quantum walk oscillates and its period is 4 steps.

As we go to hypercubes of higher dimensions, the quantum walks performed on these graphs will not be periodic anymore. However, it turns out that all quantum walks on any finite graph will be quasi-periodic as described by the quantum mechanical version of the Poincaré recurrence theorem[BL57][Sch78]. The proof given below was taken from [Sch78].

Theorem 4.1 (Discrete-time finite-dimensional quantum recurrence theorem)**.** *Consider a finite-dimensional Hilbert space H and a unitary operator U. Let $|\psi_i\rangle$ be a wave function evolving in time under the operator U, such that at time step T, $|\psi_T\rangle = U^T|\psi_0\rangle$. Then for every $\epsilon > 0$ there exist infinitely many positive integers N, such that*

$$||\psi_{n+N}\rangle - |\psi_n\rangle| < \epsilon, \quad n \in \mathbb{Z}.$$

Proof. Since U is unitary and H is finite-dimensional, we can write the states $|\psi_n\rangle$ and $|\psi_{n+T}\rangle$ as linear combinations of orthonormal eigenstates $\{|\phi_i\rangle : i = 1, \ldots, D\}$ of U,

$$|\psi_n\rangle = \sum_{i=1}^{D} c_i |\phi_i\rangle$$

and

$$|\psi_{n+T}\rangle = \sum_{i=1}^{D} c_i e^{-i\theta_i T} |\phi_i\rangle .$$

So

$$||\psi_{n+T}\rangle - |\psi_n\rangle|^2 = \left| \sum_{i=1}^{D} c_i (e^{-i\theta_i T} - 1) |\phi_i\rangle \right|^2 = 2 \sum_{i=1}^{D} |c_i|^2 (1 - \cos(\theta_i T))$$

We must therefore show that there exist infinitely many positive numbers $T = N$, such that

$$2 \sum_{i=1}^{D} |c_i|^2 (1 - \cos(\theta_i N)) < \epsilon,$$

as is done in Appendix C. □

This result is in contrast to the behaviour of classical random walks on graphs that often reach a stationary distribution eventually[POR19].

4.3 Mixing Times on the Hypercube

In section 3.2 we saw that the time it takes for a particle performing a quantum walk to get absorbed may be infinite. For the quantum walk on the hypercube we would like to study a quantity that reflects the temporal behaviour of the walk as well, but is always finite. We will thus consider the mixing time of the quantum walk on the hypercube. For the classical random walk on the hypercube, the probability distribution of the particle location converges to a stationary distribution. The mixing time of the classical walk is the time it takes the probability distribution to get close to this limiting distribution. The quantum walk, however, does not converge to a stationary distribution, hence the mixing time must be defined in a different way. Since the quantum walk is (quasi-)periodic, it would not be surprising if the time average of the probability distribution of the location of the particle would converge. This is indeed the case as shown below and we will use this fact to define the mixing time of discrete quantum walks on graphs. It turns out that the mixing time of the discrete quantum walk on the hypercube is proportional to n, the dimension of the hypercube. This is faster than for the classical random walk where the mixing time is of order $\mathcal{O}(n \log n)$[MPAD08].

We first want to show that the time average of the probability distribution of the particle location converges. Since the probability to be found at vertex v at time t is the sum $\sum_{j=1}^{d} |(\langle j| \otimes \langle v|) U^t |\Psi_0\rangle|^2$ with d the degree of the graph, we only have to show that

$$\frac{1}{N} \sum_{n=0}^{N} |(\langle j| \otimes \langle v|) U^n |\Psi_0\rangle|^2 \tag{4.5}$$

converges for $N \longrightarrow \infty$. We can write an initial state $|\Psi_0\rangle$ as

$$|\Psi_0\rangle = \sum_{\lambda_i} d_i |\lambda_i\rangle , \tag{4.6}$$

where $|\lambda_i\rangle$ is the eigenstate of evolution operator U corresponding to eigenvalue $\lambda_i = e^{-i\theta_i}$. Similarly we can write

$$|j\rangle \otimes |v\rangle = \sum_{\lambda_i} c_i |\lambda_i\rangle . \tag{4.7}$$

So

$$\frac{1}{N} \sum_{n=0}^{N} |(\langle j| \otimes \langle v|) U^n |\Psi_0\rangle|^2 = \frac{1}{N} \sum_{n=0}^{N} \left| \sum_{\lambda_i} \bar{c_i} \langle \lambda_i| U^n \sum_{\lambda_i} d_i |\lambda_i\rangle \right|^2 . \tag{4.8}$$

Since

$$\sum_{\lambda_i} \bar{c}_i \langle \lambda_i | U^n \sum_{\lambda_i} d_i |\lambda_i\rangle = \sum_{\lambda_i} \bar{c}_i d_i e^{-i\theta_i n}, \tag{4.9}$$

the sum in equation 4.5 becomes

$$\frac{1}{N} \sum_{n=0}^{N} \sum_{\lambda_i} \sum_{\lambda_j} \bar{c}_i c_j d_i \bar{d}_j e^{-i(\theta_i - \theta_j)n}. \tag{4.10}$$

As $N \longrightarrow \infty$, all terms with $\theta_i \neq \theta_j$ average out to zero and the sum converges to a constant

$$\sum_{\substack{\lambda_i, \lambda_j \\ \lambda_i = \lambda_j}} \bar{c}_i d_i c_j \bar{d}_j. \tag{4.11}$$

Now that we know that the time average of the probability distribution converges, we will define the mixing time of the quantum walk for a given initial state. Given the time t, denote the average probability distribution up to time t by $\bar{P}_t$. Denote the limit of $\bar{P}_t$ as $t \longrightarrow \infty$ by $\vec{\pi}$.

Definition 4.1 (ϵ-mixing time discrete quantum walk)**.** The ϵ-mixing time of a discrete quantum walk on a graph with limiting average distribution $\vec{\pi}$ for a given initial state is given by

$$M_\epsilon = min\{T : \forall t \geq T, \left\| \bar{P}_t - \vec{\pi} \right\| < \epsilon\},$$

where $\|\cdot\|$ is the 1-norm[4].

For different values of ϵ and dimensions n of the hypercube, the mixing time was computed, see figure 9, using the code that can be found in Appendix A.3. The Grover coin was used as coin operator and the initial state was given by $\frac{1}{\sqrt{n}} \sum_{j=1}^{n} |j\rangle \otimes |(00\ldots0)\rangle$. The mixing time seems to scale linearly with n. This is indeed the case, in contrast to the classical case where the mixing time is of order $\mathcal{O}(n \log n)$, as is confirmed numerically in [MPAD08].

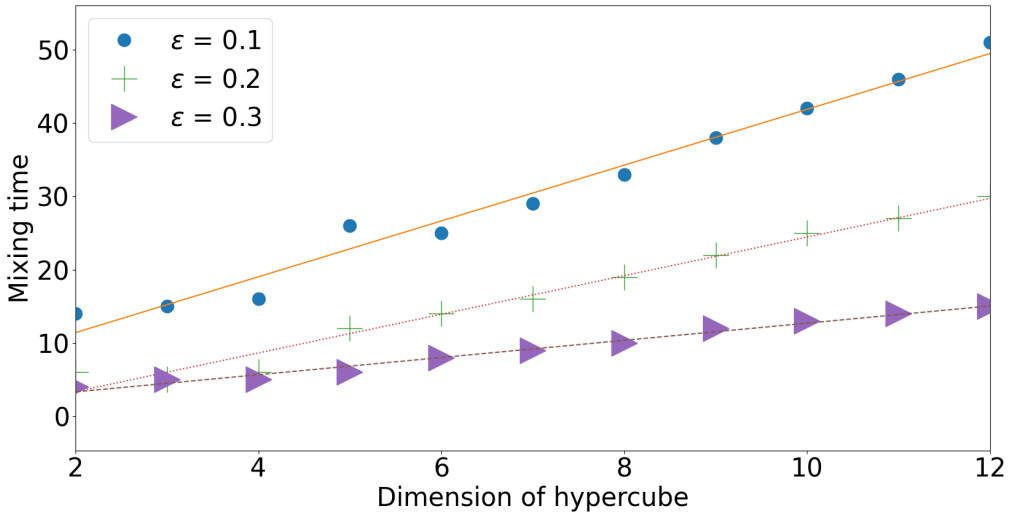

Figure 9: ϵ-mixing times for different values of ϵ and dimensions n of the hypercube. The initial state is given by $\sum_{j=1}^{n} |j\rangle \otimes |(00\ldots0)\rangle$. It can be seen that the mixing time on the hypercube is of order $\mathcal{O}(n)$, which is faster than the classical case where the mixing time is of order $\mathcal{O}(n \log n)$[MPAD08].

[4]Since $\|\vec{x}\|_p \leq \|\vec{x}\|_1, p \geq 1$ for all vectors $\vec{x}$ in a finite dimensional vector space, this definition gives a larger value of T than if any other p-norm would have been used.

5 Mixing Times of the Continuous-Time Quantum Walk on 2-Layer Multiplex Graphs

In this chapter we consider the continuous-time quantum walk and its mixing time on multiplex graphs. First we take a look at the continuous-time Markov chain in section 5.1 and then define the continuous quantum walk analogously in section 5.2. Then in section 5.3 the eigenvalues of the Laplacian of multiplex graphs are analysed. This analysis is then used in section 5.4 to consider the behaviour of mixing times of continuous-time quantum walks on multiplex graphs.

5.1 The Continuous-Time Markov Chain

We consider a continuous-time classical Markov chain first. Consider a graph G with vertex set V containing N vertices. Let $\vec{p}(t)$ be the probability distribution of the particle location at time t. When the particle is at vertex i, it will jump to neighbouring vertex j with transition rate γ_{ij}. So the probability to jump from vertex i to vertex j is γ_{ij} per unit time. We take $\gamma_{ij} = \gamma_{ji}$ and $\gamma_{ij} = 0$ if vertex i and j are not connected by an edge. We can then write down a differential equation for the probability distribution $\vec{p}(t)$

$$\frac{dp_i(t)}{dt} = \sum_{j=1}^{N} \gamma_{ij}(p_j(t) - p_i(t)). \tag{5.1}$$

If we define the Laplacian matrix as follows

$$L_{ij} = \begin{cases} \sum_{k=1}^{N} \gamma_{ik} & \text{if } i = j, \\ -\gamma_{ij} & \text{if } i \neq j, \end{cases} \tag{5.2}$$

equation 5.1 can be rewritten as

$$\frac{d\vec{p}(t)}{dt} = -L\vec{p}. \tag{5.3}$$

The solution to this equation is

$$\vec{p}(t) = e^{-Lt}\vec{p}(0). \tag{5.4}$$

Since L is a normal matrix, there exists an orthonormal basis of eigenvectors $\{\vec{v_j} : j = 1 \ldots, N\}$. We can write $\vec{p}(0) = \sum_{j=1}^{N} c_j\vec{v_j}$. Then $\vec{p}(t)$ equals $\sum_{j=1}^{N} e^{-\lambda_j t}c_j\vec{v_j}$, where λ_j is the eigenvalue of L corresponding to $\vec{v_j}$. Note that for the eigenvalues of L it holds that $0 = \lambda_1 < \lambda_2 \leq \ldots \leq \lambda_N$, since L is positive semidefinite and only $\vec{v_1} = \vec{1}$ has eigenvalue 0. Thus as $t \to \infty$ the probability distribution $\vec{p}(t)$ converges to a stationary distribution and the time this takes is proportional to $\frac{1}{\lambda_2}$, the inverse of the second smallest eigenvalue of L, in the worst case.

5.2 The Continuous-Time Quantum Walk

The continuous-time quantum walk is defined analogously to the continuous-time classical Markov chain. Again consider the graph G with vertex set V of size N. Let $\mathcal{H}$ be the Hilbert space spanned by the vertices of the graph. So $\mathcal{H}$ is spanned by the set $\{|v_i\rangle : v_i \in V\}$. For the continuous-time quantum walk we do not need a coin space and the state of the particle $|\Psi(t)\rangle$ consists only of a superposition of basis states $|v_i\rangle$.

Now we consider the following differential equation for the particle state

$$\frac{d\,|\Psi(t)\rangle}{dt} = -iH\,|\Psi(t)\rangle\,, \tag{5.5}$$

with H a Hermitian operator. This equation looks very similar to the Schrödinger equation with $\hbar$ set equal to 1. When we choose $H = L$ the Laplacian operator, we get an equation very similar to equation 5.3, only with $\vec{p}(t)$ replaced by the particle state and the right side multiplied by i. The solution of equation 5.5 is very similar in form to the solution of the classical Markov Chain, namely

$$|\Psi(t)\rangle = e^{-iHt}\,|\Psi(0)\rangle\,. \tag{5.6}$$

We take this as the definition of the continuous quantum walk. Given a Hermitian operator H, the Laplacian L for our purposes, the initial particle state is transformed to the particle state at time t

by applying the unitary operator e^{-iHt}. When we write our particle state as a column vector, we can again write down the particle state as a superposition of eigenstates of H. When $|\Psi(0)\rangle = \sum_{j=1}^{N} c_j \vec{v_j}$, $|\Psi(t)\rangle = \sum_{j=1}^{N} e^{-i\lambda_j t} c_j \vec{v_j}$, where λ_j is the eigenvalue of L corresponding to $\vec{v_j}$.

5.3 Laplacian Eigenvalues of 2-Layer Multiplex Graphs

In this section we consider the eigenvalues of the Laplacian of a special type of graph, namely a multiplex graph. A multiplex graph is made up of multiple layers and has the property that vertices in different layers can only be connected if they are each other's counterparts. See figure 10 for an example. The strength of the connections between the layers can be varied and we want to study the behaviour of the eigenvalues of the Laplacian as we vary this strength. We will replicate the analysis of the eigenvalues of L for different strengths as is done in [GDGGG+13]. Since for the continuous-time Markov chain the probability distribution $\vec{p}(t)$ converges to a stationary distribution in time proportional to $\frac{1}{\lambda_2}$, we will pay extra attention to the second smallest eigenvalue of the Laplacian. The results of this section will also be applied to the continuous quantum walk in the next section.

Let us first explain the concept of a multiplex graph. A multiplex graph is a special type of multilayer graph. The vertices of different layers are replicates of each other. So for every two layers, a vertex in one layer has a counterpart in the other. The way the vertices are connected within a layer however, differs for different layers. The layers themselves are connected by edges between the vertices that are each other's counterparts. See figure 10 for an example of a 2-layer multiplex graph.

The different layers of the multiplex graph can for example represent different transportation networks (e.g., the bus, the subway)[Bar11] or social networks(e.g., Facebook, Threads)[GDGGG+13]. When the connections between the layers have a different strength compared to the connections within the layers, for example when the value of the transition rate γ varies, multiplex graphs can be useful tools to analyse these networks.

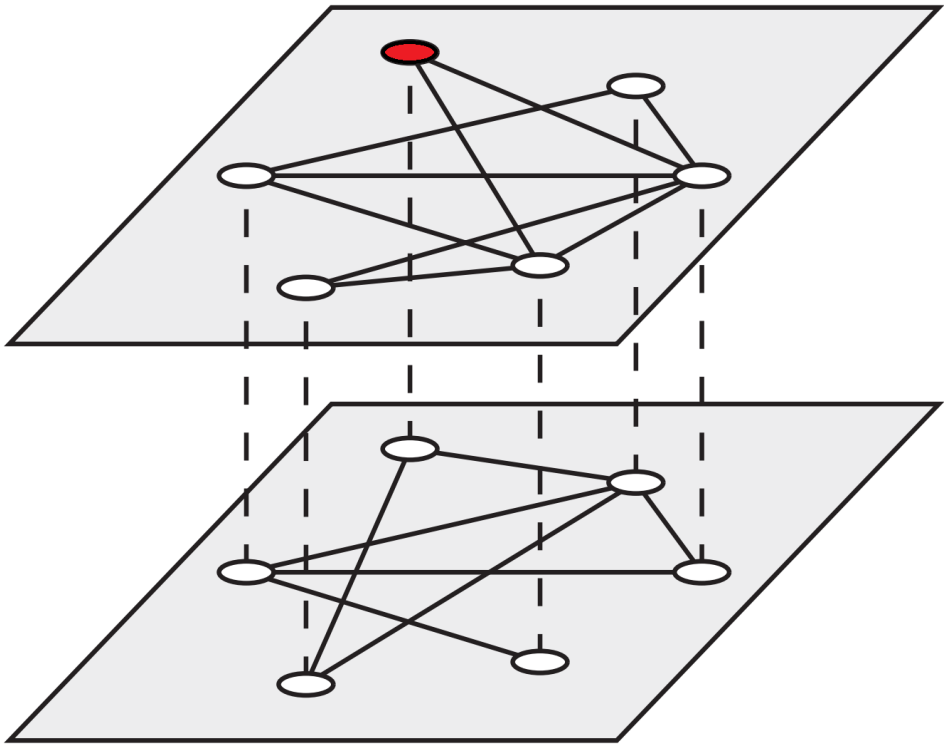

Figure 10: An example of a multiplex graph made up of two layers taken from [GDGGG+13]. The vertices in both layers are copies of each other, but the connections within the layers are different. The connections between the layers are formed by the edges connecting duplicate vertices. In section 5.4 the red vertex is taken as the starting point of a quantum walk for which the mixing time is computed.

We will only consider 2-layer multiplex graphs. Let N be the number of vertices in each layer.

We set the transition rate γ_1 within the layers equal to 1 and vary the transition rate γ_2 between the layers. When L_1, L_2 are the Laplacians of the layers when isolated,

$$L = \begin{pmatrix} L_1 & 0 \\ 0 & L_2 \end{pmatrix} + \gamma_2 \begin{pmatrix} I & -I \\ -I & I \end{pmatrix} \tag{5.7}$$

is the Laplacian of the multiplex graph. When $\gamma_2 = 0$, the set of eigenvalues of L is the union of the set of eigenvalues of L_1 and the set of eigenvalues of L_2. So $\lambda_2 = min\{\lambda_2^1, \lambda_2^2\}$, where $\lambda_2^{1,2}$ are the second smallest eigenvalues of L_1 and L_2 respectively.

When $\gamma_2 \neq 0$, $2\gamma_2$ is always an eigenvalue of L with corresponding eigenvector $\begin{pmatrix} \vec{1} \\ -\vec{1} \end{pmatrix}$. So for small values of γ_2, $\lambda_2 = 2\gamma_2$.

We now consider the case for γ_2 very large. Write $\epsilon = \frac{1}{\gamma_2}$, then

$$L = \gamma_2 \left[\begin{pmatrix} I & -I \\ -I & I \end{pmatrix} + \epsilon \begin{pmatrix} L_1 & 0 \\ 0 & L_2 \end{pmatrix} \right] = \gamma_2 \tilde{L}. \tag{5.8}$$

Note that $\begin{pmatrix} I & -I \\ -I & I \end{pmatrix}$ has the N-degenerate eigenvalues 0 and 2 with eigenvectors $\begin{pmatrix} \vec{u} \\ \vec{u} \end{pmatrix}$ and $\begin{pmatrix} \vec{u} \\ -\vec{u} \end{pmatrix}$ respectively. Thus as $\gamma_2 \longrightarrow \infty$, L has N eigenvalues $2\gamma_2$ diverging to infinity and N eigenvalues converging to some finite value as a result of the undetermined limit $(0 \cdot \infty)$. We will use perturbation theory to find these eigenvalues.

We propose that the eigenvalues and eigenvectors of $\tilde{L}$ have the following form

$$\tilde{\lambda}_i = \tilde{\lambda}_i^{(0)} + \epsilon \tilde{\lambda}_i^{(1)} + \mathcal{O}(\epsilon^2), \tag{5.9}$$

$$\vec{\tilde{v}}_i = \vec{\tilde{v}}_i^{(0)} + \epsilon \vec{\tilde{v}}_i^{(1)} + \mathcal{O}(\epsilon^2). \tag{5.10}$$

Since we only want to consider the eigenvalues that converge for $\gamma_2 \longrightarrow \infty$, we consider the following perturbed eigenvalues and -vectors

$$\tilde{\lambda} = 0 + \epsilon \tilde{\lambda}' + \mathcal{O}(\epsilon^2), \tag{5.11}$$

$$\vec{\tilde{v}} = \begin{pmatrix} \vec{u} \\ \vec{u} \end{pmatrix} + \epsilon \begin{pmatrix} \vec{u}'_1 \\ \vec{u}'_2 \end{pmatrix} + \mathcal{O}(\epsilon^2). \tag{5.12}$$

When we put this in the equation $\tilde{L}\vec{\tilde{v}} = \tilde{\lambda}\vec{\tilde{v}}$,

$$\epsilon \begin{pmatrix} L_1 \vec{u} + (\vec{u_1}' - \vec{u_2}') \\ L_2 \vec{u} + (-\vec{u_1}' + \vec{u_2}') \end{pmatrix} = \epsilon \lambda' \begin{pmatrix} \vec{u} \\ \vec{u} \end{pmatrix} + \mathcal{O}(\epsilon^2). \tag{5.13}$$

So we can write

$$(L_1 + L_2)\vec{u} = 2\tilde{\lambda}'\vec{u} + \mathcal{O}(\epsilon). \tag{5.14}$$

by adding the bottom of expression 5.13 to the top and multiplying by γ_2. Thus $\vec{u}$ must be an eigenvector of $L_1 + L_2$ and we expect the eigenvalues of L to converge to

$$\lambda = \tilde{\lambda}' = \frac{\lambda_s}{2}, \tag{5.15}$$

where λ_s is an eigenvalue of $L_1 + L_2$. So the second smallest eigenvalue of L converges to to the second smallest eigenvalue of $\frac{L_1+L_2}{2}$. Since it must hold that this eigenvalue is larger than $min\{\lambda_2^1, \lambda_2^2\}$, we see that for large γ_2, the convergence of $\vec{p}(t)$ in the multiplex graph is faster than in the layer $argmin_i\{\lambda_2^i : i = 1, 2\}$.

For the multiplex graph in figure 10, the eigenvalues of the Laplacian were computed using Mathematica and are plotted in figure 11. We see that as $\gamma_2 \longrightarrow \infty$ half of the eigenvalues converge to some finite value and the other half diverge like $2\gamma_2$ as expected.

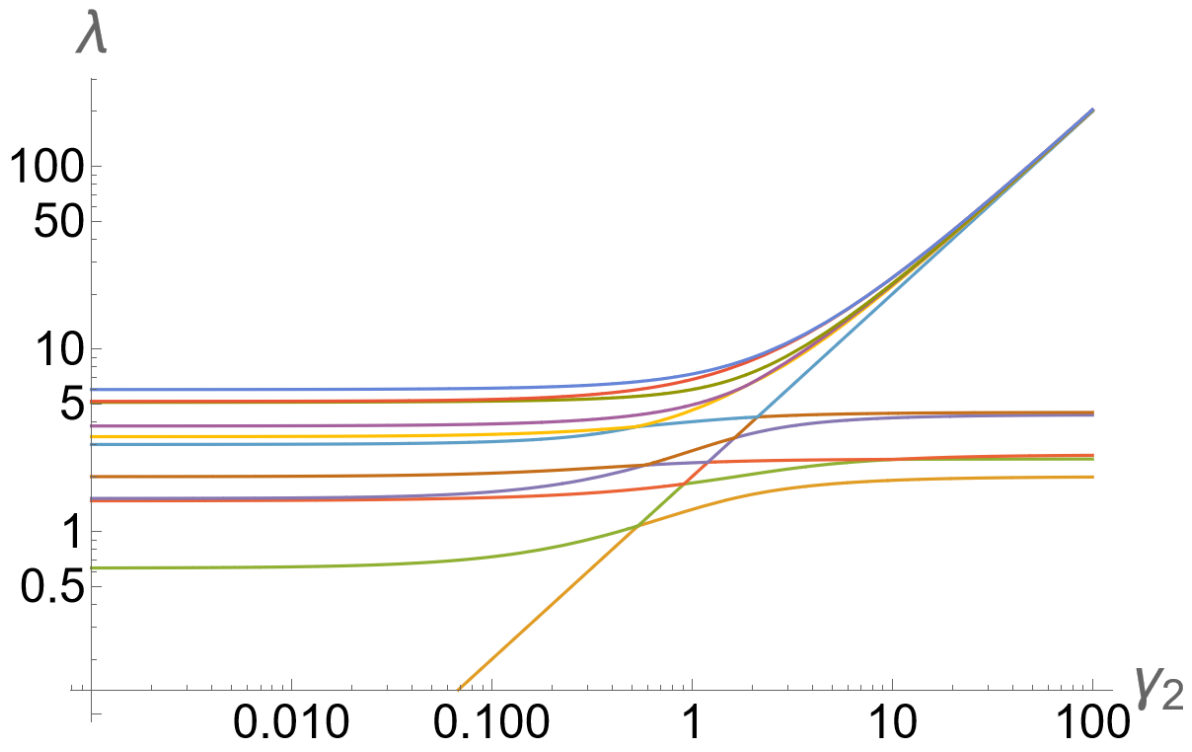

Figure 11: The eigenvalues of the multiplex graph in figure 10 are plotted for varying γ_2. We see that indeed half of the eigenvalues diverge to infinity and the other half converges.

In figure 12 λ_2 as a function of γ_2 is plotted using Mathematica. We see that $2\gamma_2$ is indeed a good approximation of λ_2 for small values of γ_2 and that the second smallest eigenvalue of $\frac{L_1+L_2}{2}$ is a good approximation for large values of γ_2. We can also see that in this case λ_2 of L is larger than the second smallest-eigenvalues of both L_1 and L_2.

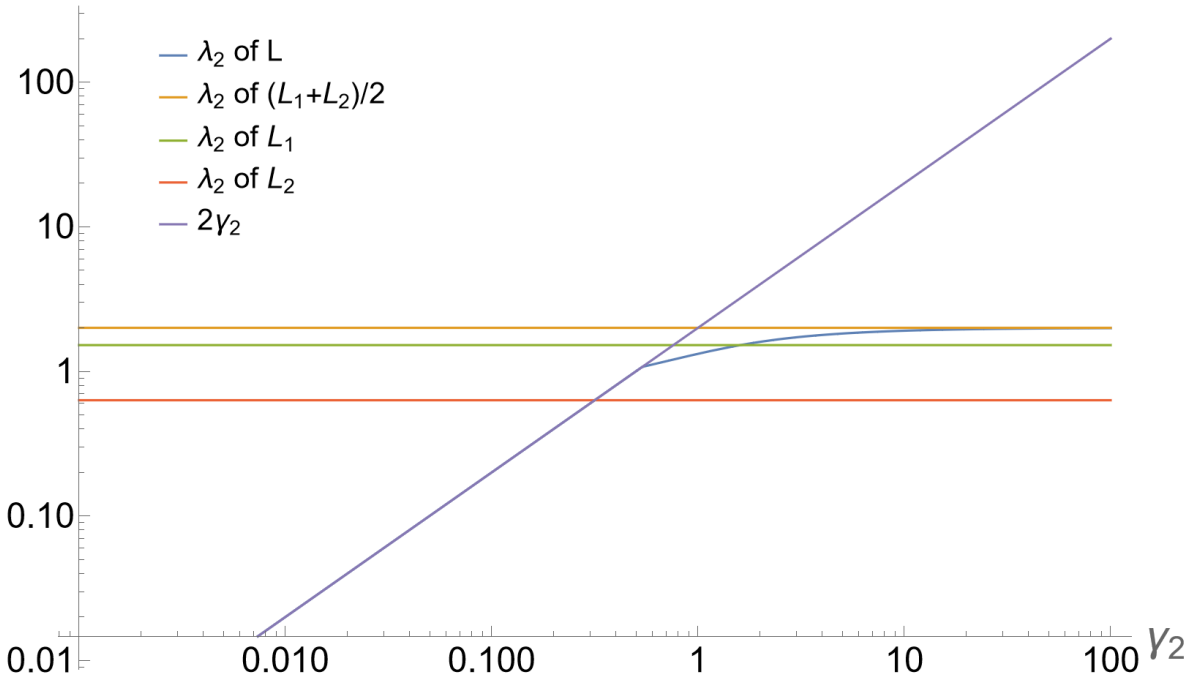

Figure 12: The smallest eigenvalue λ_2 of the Laplacian L is plotted as a function of γ_2 for the multiplex graph in figure 10. We see that $2\gamma_2$ is a good approximation for λ_2 for small values of γ_2 and that the second smallest eigenvalue of the $\frac{L_1+L_2}{2}$ is a good approximation for large values of γ_2.

5.4 Mixing Times of The Continuous Quantum Walk on 2-Layer Multiplex Graphs

Similar to the discrete quantum walk, the time average of the probability distribution of the continuous quantum walk converges. We will use this fact to define the mixing time of the continuous quantum walk and consider how this quantity behaves on multiplex graphs with varying γ_2. Specifically, we will consider multiplex graphs whose layers are generated by the Erdős–Rényi model.

To show that the time average of the probability distribution of the continuous quantum walk converges we will consider the integral

$$\frac{1}{T}\int_0^T \left|\langle v|\, e^{-iHt}\, |\Psi(0)\rangle\right|^2 dt \tag{5.16}$$

which equals the time average of the probability to find the particle in vertex v upon measurement from time 0 to time T. Again we expand $|v\rangle, |\Psi(0)\rangle$ in the orthonormal basis of eigenvectors of H.

$$\frac{1}{T}\int_0^T \left|\left(\sum_{\lambda_i} \bar{c}_i \langle\lambda_i|\right) e^{-iHt} \left(\sum_{\lambda_i} d_i |\lambda_i\rangle\right)\right|^2 dt = \frac{1}{T}\int_0^T \left|\sum_{\lambda_i} \bar{c}_i d_i e^{-i\lambda_i t}\right|^2 dt$$

$$= \frac{1}{T}\int_0^T \sum_{\lambda_i}\sum_{\lambda_j} \bar{c}_i d_i c_j \bar{d}_j e^{-i(\lambda_i-\lambda_j)t}\, dt = \sum_{\substack{\lambda_i,\lambda_j \\ \lambda_i=\lambda_j}} \bar{c}_i d_i c_j \bar{d}_j + \frac{1}{T}\sum_{\substack{\lambda_i,\lambda_j \\ \lambda_i\neq\lambda_j}} \frac{\bar{c}_i d_i c_j \bar{d}_j}{-i(\lambda_i-\lambda_j)}(e^{-i(\lambda_i-\lambda_j)T}-1), \tag{5.17}$$

which must converge to a constant value as $T \longrightarrow \infty$.

Denote the time average of the probability distribution from time 0 to t by $\bar{P}(t)$ and its limit by $\vec{\pi}$. We can then define the ϵ-mixing time for the continuous quantum walk similar to how it was defined for the discrete case.

Definition 5.1 (ϵ-mixing time continuous quantum walk)**.** The ϵ-mixing time of a continuous quantum walk on a graph with limiting average distribution $\vec{\pi}$ for a given initial state is given by

$$M_\epsilon = inf\{T : \forall t \geq T, \left\|\bar{P}(t) - \vec{\pi}\right\| < \epsilon\},$$

where $\|\cdot\|$ is the 1-norm.

Given the final expression in equation 5.17, we can write for a graph G with vertex set V such that for $v \in V$, $|v\rangle = \sum_{\lambda_i} c_i^v |\lambda_i\rangle$,

$$\left\|\bar{P}(T) - \vec{\pi}\right\| = \sum_{v\in V} \left| \frac{1}{T} \sum_{\substack{\lambda_i,\lambda_j \\ \lambda_i \neq \lambda_j}} \frac{\bar{c}_i^v d_i c_j^v \bar{d}_j}{-i(\lambda_i - \lambda_j)} (e^{-i(\lambda_i-\lambda_j)T} - 1) \right| \leq \frac{1}{T} \sum_{v\in V} \sum_{\substack{\lambda_i,\lambda_j \\ \lambda_i \neq \lambda_j}} \left| \frac{\bar{c}_i^v d_i c_j^v \bar{d}_j}{-i(\lambda_i - \lambda_j)} (e^{-i(\lambda_i-\lambda_j)T} - 1) \right|. \tag{5.18}$$

Then as in [CLR20] we can write

$$\left\|\bar{P}(T) - \vec{\pi}\right\| \leq \frac{1}{T} \sum_{\substack{\lambda_i,\lambda_j \\ \lambda_i \neq \lambda_j}} \sum_{v\in V} \left|\bar{c}_i^v c_j^v\right| \frac{\left|d_i \bar{d}_j\right|2}{|\lambda_i - \lambda_j|} \leq \frac{1}{T} \sum_{\substack{\lambda_i,\lambda_j \\ \lambda_i \neq \lambda_j}} \frac{\left|d_i \bar{d}_j\right|2}{|\lambda_i - \lambda_j|}, \tag{5.19}$$

since[5] $\left|\bar{c}_i^v c_j^v\right| = |\langle v|\lambda_i\rangle \langle\lambda_j|v\rangle| \leq \frac{1}{2}(|\langle v|\lambda_i\rangle|^2 + |\langle\lambda_j|v\rangle|^2)$ and $\sum_{v\in V} |\langle v|\lambda_i\rangle|^2 = 1$.

Hence we expect the ϵ-mixing time to be bounded by

$$\frac{1}{\epsilon} \sum_{\substack{\lambda_i,\lambda_j \\ \lambda_i \neq \lambda_j}} \frac{\left|d_i \bar{d}_j\right|2}{|\lambda_i - \lambda_j|}. \tag{5.20}$$

Thus whereas in the classical case the convergence rate of $\vec{p}(t)$ is determined by the second smallest eigenvalue, in the quantum case it is determined by all eigenvalue gaps $|\lambda_i - \lambda_j| \neq 0$ of the Laplacian.

Now we want to look at how the mixing time behaves on 2-layer multiplex graphs for varying γ_2. We expect that, analogous to the classical case, the mixing time converges as γ_2 goes to infinity, since the eigenvalues of the Laplacian converge and hence the values of the eigenvalue gaps. For the multiplex graph in figure 10, the sum $\sum_{\substack{\lambda_i,\lambda_j \\ \lambda_i \neq \lambda_j}} \frac{1}{|\lambda_i - \lambda_j|}$ as a function of γ_2 has been computed and plotted in figure 13 using Mathematica. The code can be found in Appendix A.4. We see that the sum is particularly large around 1 and converges for large values of γ_2. This is as expected if we consider figure 11. Around 1 the different eigenvalues are very close to each other, hence the sum grows very large. For large values of γ_2 the eigenvalues converge to the eigenvalues of $(L_1 + L_2)/2$, so the sum converges as well.

[5] $a^2 + b^2 \geq 2ab$ for any two real numbers a and b.

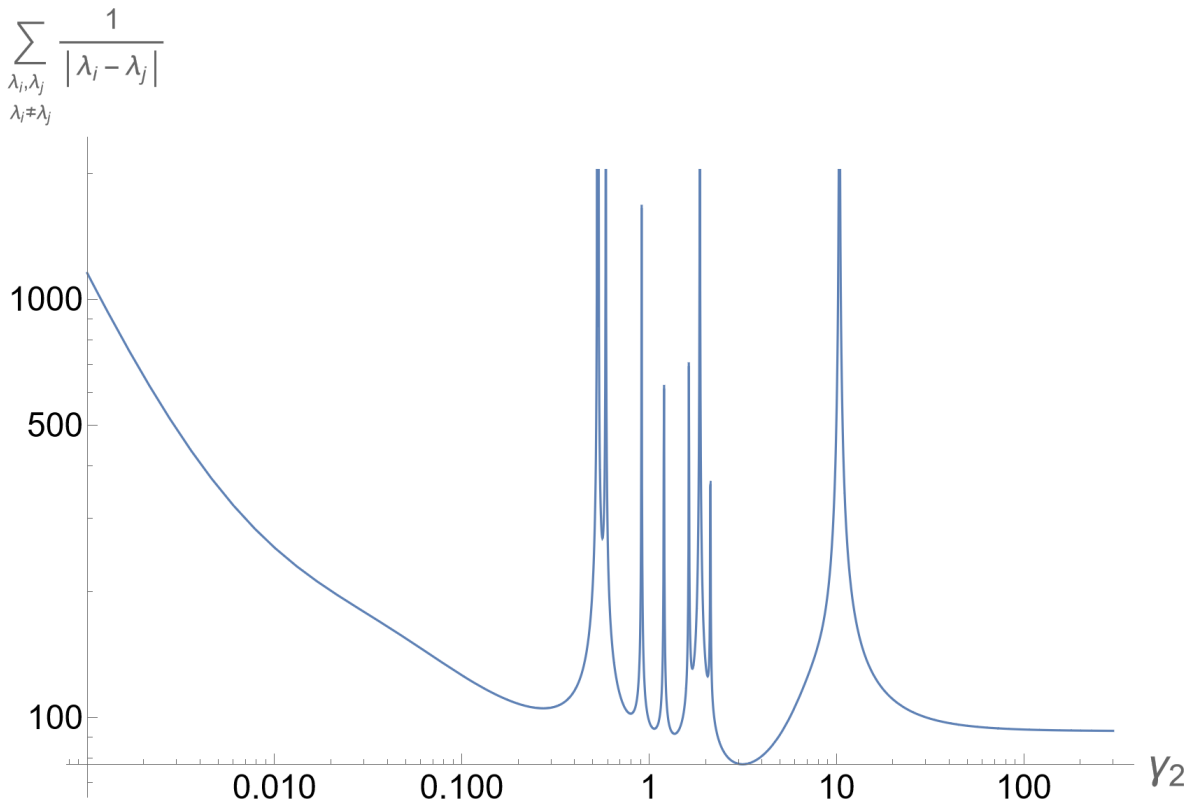

Figure 13: The sum $\sum_{\substack{\lambda_i,\lambda_j \\ \lambda_i \neq \lambda_j}} \frac{1}{|\lambda_i - \lambda_j|}$ has been computed as a function of γ_2 for the multiplex graph in figure 10. We see that the sum peaks around 1, where the eigenvalues of the Laplacian of the graph are very close to each other. We also see that for large γ_2 the sum converges as expected, since the eigenvalues converge as well.

In figure 14, the 0.1-mixing time of the quantum walk starting on the red vertex in figure 10 has been plotted. We see that the mixing time peaks in the same places around 1 as the sum does in figure 13. For large values of γ_2 the mixing time seems to converge as expected, however there are a few unexpected peaks that do not correspond to any peaks of the sum. These peaks might be caused by variations in the coefficients c_i, d_i of the expansions of the initial state $|\Psi(0)\rangle$ and vertex state $|v\rangle$ in terms of the orthonormal basis of eigenvectors as γ_2 changes. Note that the mixing time depends on the initial position. When we would have started on a different vertex than the red one, the behaviour of the mixing time would have been slightly different.

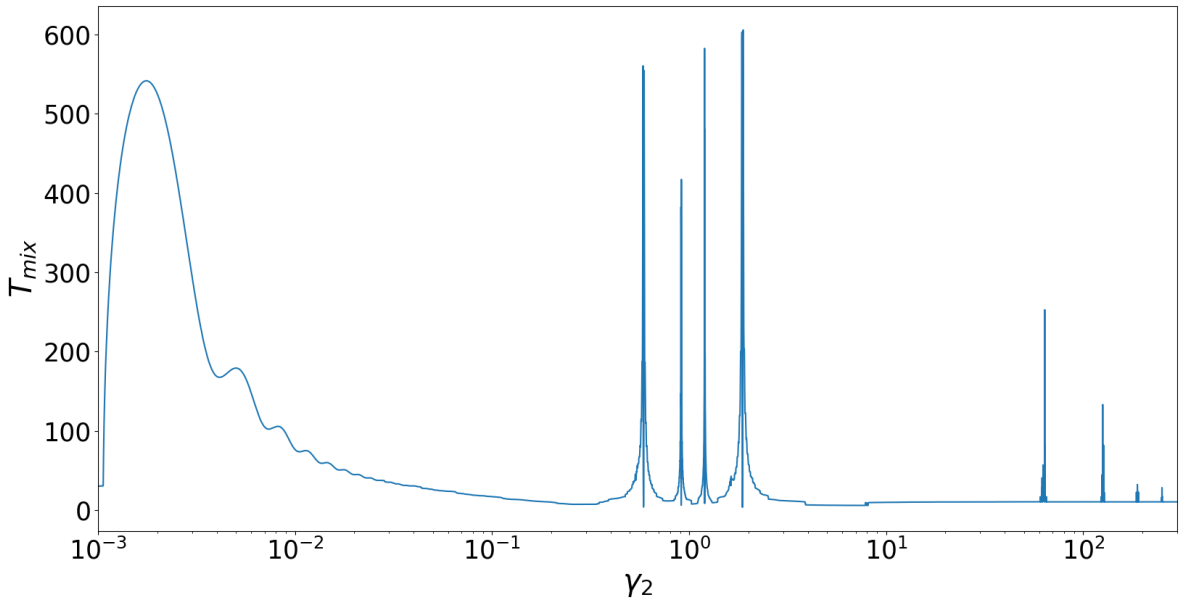

Figure 14: Mixing time of the continuous quantum walk on the graph in figure 10 starting on the red vertex for different values of γ_2. The peaks in mixing time around 1 correspond to the peaks in figure 13. The mixing time also seems to converge for large values of γ_2, however there are a few unexpected peaks.

The 0.05-mixing time has also been computed for 2-layer multiplex graphs where each layer is a graph generated by the $G(300, 1/2)$ Erdős–Rényi model. This model constructs a graph of 300 vertices and every two vertices are connected with probability $1/2$. Since this model generates a different graph each time it is called upon, for every value of γ_2 we compute the average of the 0.05-mixing times of 1000 multiplex graphs. These averages are plotted in figure 15 together with the standard deviation. The Python code used to compute these values can be found in Appendix A.6. Note that since the statistical properties of every vertex are the same, the starting vertex of each of the 1000 quantum walks is not expected to have much influence on the value of the average mixing time.

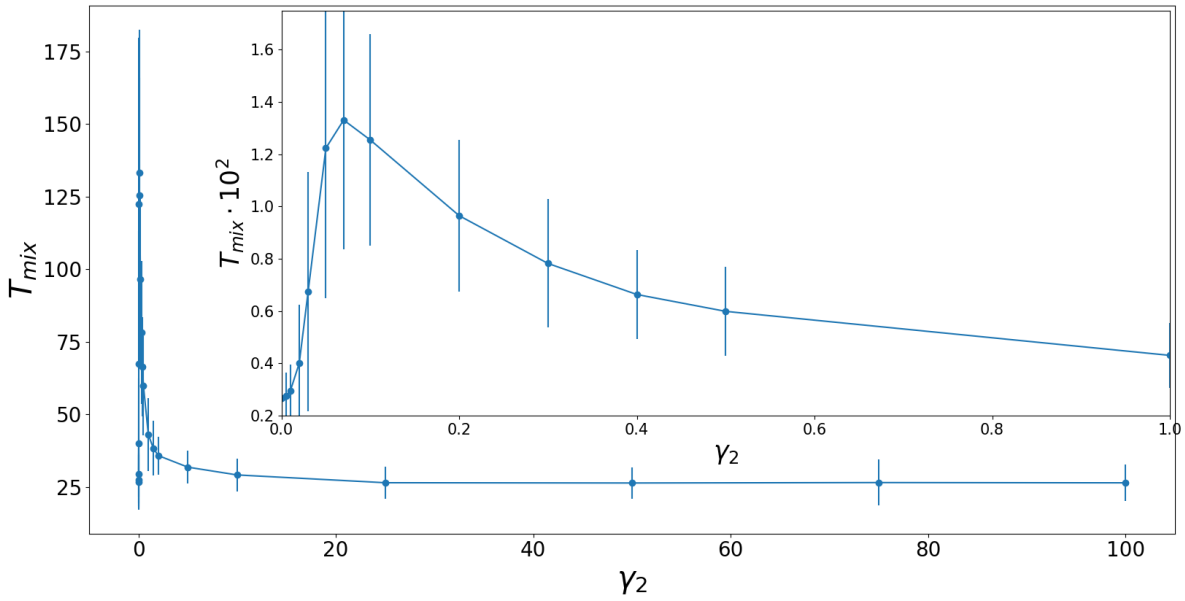

Figure 15: For different values of γ_2 the average of the 0.05-mixing times of 1000 multiplex graphs is shown. Every multiplex graph consists of 2 layers that are generated by the $G(300, 1/2)$ Erdős–Rényi model. The standard deviation is also plotted. We see that initially the average mixing time peaks and then converges. This is as expected since the eigenvalues of the Laplacian converge as described in the last section.

We see that the average mixing time initially increases rapidly. It then converges for large values of γ_2 as expected since the eigenvalues of the Laplacian L converge to the eigenvalues of $(L_1 + L_2)/2$. In this case, however, the matrices L_1 and L_2 are the random matrices

$$\begin{pmatrix} \sum_{i\neq 1} X_{1i} & -X_{12} & \cdots & -X_{1N} \\ -X_{21} & \sum_{i\neq 2} X_{2i} & \cdots & -X_{2N} \\ \vdots & & \ddots & \vdots \\ -X_{N1} & \cdots & -X_{N(N-1)} & \sum_{i\neq N} X_{Ni} \end{pmatrix} \tag{5.21}$$

where X_{ij} are independent $p = \frac{1}{2}$ Bernoulli random variables for $j > i$ and $X_{ij} = X_{ji}$. The matrix $L_1 + L_2$ then equals

$$\begin{pmatrix} \sum_{i\neq 1} Y_{1i} & -Y_{12} & \cdots & -Y_{1N} \\ -Y_{21} & \sum_{i\neq 2} Y_{2i} & \cdots & -Y_{2N} \\ \vdots & & \ddots & \vdots \\ -Y_{N1} & \cdots & -Y_{N(N-1)} & \sum_{i\neq N} Y_{Ni} \end{pmatrix} \tag{5.22}$$

with $Y_{ij} = X^1_{ij} + X^2_{ij}$ binomial $B(n = 2, p = 1/2)$ independent random variables for $j > i$ and $Y_{ij} = Y_{ji}$. In further research lower bounds on the eigenvalue gaps of these matrices could be investigated in terms of their size N to find upper bounds for the mixing time in the limits $\gamma_2 \longrightarrow 0$ and $\gamma_2 \longrightarrow \infty$.

6 Conclusions

For the discrete quantum walk on the line with an absorbing boundary it has been shown that the probability of the particle performing the quantum walk to be eventually absorbed equals $\frac{2}{\pi}$. The particle could therefore walk forever without getting absorbed. This is in stark contrast to the classical case where the probability to be eventually absorbed equals 1.

In chapter 4 it has been shown that quantum walks are quasi-periodic in contrast to classical random walks which often reach stationary distributions. It has also been shown that the mixing time of the discrete quantum walk on the hypercube scales linearly with the dimension n of the hypercube, unlike the classical random walk for which the mixing time is of the order $\mathcal{O}(n \log n)$. The quantum walk thus offers a slight speed-up compared to the classical random walk.

Finally, in chapter 5 it has been shown that the mixing times of continuous-time quantum walks on multiplex graphs are determined by the eigenvalue gaps of the Laplacian. This is in contrast to the classical case where only the value of the second smallest eigenvalue λ_2 is relevant. Similar to the classical case, we conclude that the mixing times converge as $\gamma_2 \longrightarrow \infty$, since the eigenvalues of the corresponding Laplacian matrix converge to the eigenvalues of the sum of the Laplacian matrices of the layers $(L_1 + L_2)/2$. When the two layers are graphs generated by the Erdős–Rényi model, $L_1 + L_2$ is a random matrix and further research could be conducted to derive bounds on the eigenvalue gaps of this graph such that bounds on the mixing time can be found.

A Code

In this part of the appendix the most important pieces of code used in this report are collected.

A.1 Absorption Probability Semi-Infinite Quantum Walk

In section 3.2 it is shown that the probability to be eventually absorbed by an absorbing boundary placed at the origin equals $\frac{2}{\pi}$ for the quantum walk on the line. This result was numerically verified by the following code.

```
import numpy as np
import math
import scipy.sparse as sparse

'''
We try to approximate the probability to be absorbed by the boundary placed at
0 eventually,sum ||Pi U ((I - Pi)U)^(t-1) |psi_0>||^2 , by using a finite sum
'''

#number of terms of sum
lengte_som = 1000

#Making operators
#Identity minus projection
IminPi = np.identity(2*(lengte_som + 2))
IminPi[lengte_som+2, lengte_som+2] = 0
#Projection
Pi = np.zeros((2*(lengte_som + 2), 2*(lengte_som + 2)))
Pi[lengte_som+2, lengte_som+2] = 1
#Hadamard coin
H = (1/np.sqrt(2))*np.array([[1, 1],[1, -1]])
I = np.identity(lengte_som + 2)
#Shift
S = np.block([[np.eye(lengte_som + 2, lengte_som + 2, -1),
               np.zeros((lengte_som + 2, lengte_som + 2))],
              [np.zeros((lengte_som + 2, lengte_som + 2)),
               np.eye(lengte_som + 2, lengte_som + 2,  1)]])

#U = S(CxI)
U = np.matmul(S, np.kron(H, I))
IminPiU = sparse.csr_matrix(np.matmul(IminPi, U))
PiU = sparse.csr_matrix(np.matmul(Pi, U))

#Initial state
psi_0 = np.zeros(2*(lengte_som + 2))
psi_0[1] = 1

#Summing
som = 0
for i in range(lengte_som):
    som += np.linalg.norm(sparse.csr_matrix.dot(PiU, psi_0), ord = 2)**2
    psi_0 = sparse.csr_matrix.dot(IminPiU, psi_0)



#Sum total
print('approximation:', som)
```

```
print('2/pi:', (2/np.pi))
```

A.2 Quantum Walk on the Hypercube

To perform a quantum walk on a Hypercube as done in section 4.2 the following code was used.

```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import math
import scipy.sparse as sparse

def WalkNsteps(G, degree, N_steps, particle, coin = 'DFT'):
    if coin == 'DFT':
        CoinxI = CoinOperatorDFT(degree, int(len(particle)/degree))
    elif coin == 'Grover':
        CoinxI = CoinOperatorGrover(degree, int(len(particle)/degree))
    #Shift operator
    S = sparse.csr_matrix(ShiftOperator(G, len(particle)))
    #Walking
    for i in range(0, N_steps):
        print(i)
        #Apply coin operator
        particle = np.dot(CoinxI, particle)
        #Now we apply the shift operator
        particle = sparse.csr_matrix.dot(S, particle)
    #Return the final particle state
    return particle

def ShiftOperator(G, DegreeTimesNodes):
    S = np.zeros((DegreeTimesNodes, DegreeTimesNodes))
    for j in range(DegreeTimesNodes):
        #The label corresponding to index j
        lab = math.ceil((j+1)/N_nodes)
        #The node corresponding to index j, when we call node (0,0,...,0)
        #0, node (0,0,..,0,1) 1 etc. So we count in binary.
        node = j % N_nodes
        noot = list(G)[node]
        #Looking for the neighbour on the other side of the edge with the right
        #label
        for nbr in range(0, len(G[noot])):
            neighbour = list(G[noot])[nbr]
            if G[noot][neighbour]['label'] == lab:
                #Now our node with label gets shifted to the neighbour
                #that is on the other end of the edge with the right label
                #We need to get the right index for our
                #neighbouring node
                nummer_neighbour = list(G.nodes()).index(neighbour)
                S[(lab - 1) * N_nodes + nummer_neighbour, j] = 1
    return S

#Making the DFT coin operator
def CoinOperatorDFT(degree, N_nodes):
    w = np.exp(2j*np.pi/degree)
    DFT_coin = np.zeros((degree, degree), dtype = complex)
    for i in range(0, degree*degree):
        #Computing row column for given i
```

```
            row = math.ceil((i+1)/degree)
            column = (i+1) % degree
            if column == 0:
                column = degree
            else:
                True
            #raising w (representing omega)  to the correct power
            #corresponding to that row and column
            DFT_coin[row-1][column - 1] = w**((row-1)*(column-1))
        DFT_coin = 1/np.sqrt(degree) * DFT_coin
        #Taking the kronecker product, because we have to take tensor products
        I = np.identity(N_nodes)
        DFTxI = np.kron(DFT_coin, I)
        return DFTxI

    def CoinOperatorGrover(degree, N_nodes):
        #Making the coin operator
        Grover_coin = np.full((degree, degree), 2/degree) - np.diag(np.ones(degree))
        I = np.identity(N_nodes)
        GroverxI = np.kron(Grover_coin, I)
        return GroverxI

    def NodeProbs(particle, degree):
        #Computing the probabilies to be at each node
        node_weights = np.zeros(N_nodes)
        for i in range(degree):
            node_weights += np.abs(particle[(i*N_nodes):((i+1)*N_nodes)])**2
        return node_weights

    ##############################################################################
    #Dimension of hypercube
    n_dim = 2
    #The degree
    degree = n_dim
    #The number of nodes
    N_nodes = 2**n_dim

    #Number of steps we take during the walk
    N_steps = 100

    #Which coin we use
    coin = 'Grover'

    #Making hypercube of dimension n
    G = nx.hypercube_graph(n_dim)

    ##############################################################################
    #Labeling the edges.
    #Running through all edges
    #i can be ((0,0,0),(1,0,0)) for example
    for i in G.edges():
        #Comparing the bits of the neighbouring nodes
        for j in range(len(i[0])):
            if i[0][j] != i[1][j]:
                #Labeling the edges by which bit flipped
                G.edges[i]['label'] = j + 1
```

```
#############################################################################################
#Initializing particle state
#Particle state, we run through nodes (0,0,..,0) up to (1,1,...,1)
#for each label 1 up to degree = n_dim
particle = np.zeros(degree * N_nodes, dtype = complex)
#initialize our particle uniformly over all labels and as being at node 0
particle[0] = 1/np.sqrt(2)
particle[N_nodes] = 1/np.sqrt(2)

#############################################################################################
#Now walking
#Drawing graph with probabilities
particle_final = WalkNsteps(G, degree, N_steps, particle, coin)
node_weights = NodeProbs(particle_final, degree)
```

A.3 Mixing Times on the Hypercube

To compute mixing times on the hypercube, see section 4.3, the following code was used. It makes use of some of the functions as found in the code in Appendix A.2.

```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import math
import scipy.sparse as sparse

def MixingTime(G, degree, N_times, particle, coin = 'DFT'):
    #Make coin operator
    if coin == 'DFT':
        CoinxI = sparse.csr_matrix(CoinOperatorDFT(degree,
                                int(len(particle)/degree)))
    elif coin == 'Grover':
        CoinxI = sparse.csr_matrix(CoinOperatorGrover(degree,
                                int(len(particle)/degree)))
    #Shift operator
    S = ShiftOperator(G, len(particle))
    #First compute average distribution
    particle_0 = np.copy(particle)
    av_dist = NodeProbs(particle_0, degree)
    for i in range(0, N_times):
        print('to stationary', i, degree)
        #Apply coin operator
        particle_0 = sparse.csr_matrix.dot(CoinxI, particle_0)
        particle_0 = sparse.csr_matrix.dot(S, particle_0)
        av_dist += NodeProbs(particle_0, degree)
    av_dist = av_dist/N_times

    #Compute mixing time
    particle_0 = np.copy(particle)
    #Time step
    step = 0
    #Number of steps we have been epsilon close
    steps_under = 0
    #Average distribution up till now
    av_dist_step = NodeProbs(particle_0, degree)
    #Distance
```

```
    epsilon = np.linalg.norm(av_dist_step - av_dist, ord = 1)
    if epsilon < 0.3:
        steps_under += 1
    while steps_under < 500:
        step += 1
        print('Convergence:', step)
        particle_0 = sparse.csr_matrix.dot(CoinxI, particle_0)
        particle_0 = sparse.csr_matrix.dot(S, particle_0)
        av_dist_step += NodeProbs(particle_0, degree)
        epsilon = np.linalg.norm(av_dist_step/(step) - av_dist, ord = 1)
        if epsilon < 0.3:
            steps_under += 1
        else:
            steps_under = 0
    mix_time = step - 499

    #Return the mixing time
    return mix_time
##############################################################################
#Mixing time
#all mixing times up to dimension
n = 12
MixTime = np.zeros(n-1)
for i in range(0, n-1):
    #Dimension of hypercube
    n_dim = i + 2
    #The degree
    degree = n_dim
    #The number of nodes
    N_nodes = 2**n_dim

    #Which coin we use
    coin = 'Grover'

    #Making hypercube of dimension n
    G = nx.hypercube_graph(n_dim)

##############################################################################
    #Labeling the edges.
    #Running through all edges
    #i can be ((0,0,0),(1,0,0)) for example
    for j in G.edges():
        #Comparing the bits of the neighbouring nodes
        for k in range(len(j[0])):
            if j[0][k] != j[1][k]:
                #Labeling the edges by which bit flipped
                G.edges[j]['label'] = k + 1

##############################################################################
    #Initializing particle state
    #Particle state, we run through nodes (0,0,..,0) up to (1,1,...,1)
    #for each label 1 up to degree = n_dim
    particle = np.zeros(degree * N_nodes, dtype = complex)
    #initialize our particle uniformly over all labels and as being at node 0
    for j in range(n_dim):
        particle[j*N_nodes] = 1/np.sqrt(n_dim)
```

```
##############################################################################
    MixTime[i] = MixingTime(G, degree, 10000, particle, coin)
```

A.4 Sum of Eigenvalue Gaps

The following Mathematica code was used to compute the sum $\sum_{\substack{\lambda_i,\lambda_j \\ \lambda_i \neq \lambda_j}} \frac{1}{|\lambda_i - \lambda_j|}$ in section 5.1 for the multiplex graph in figure 10.

```
L1 = {{2, 0, 0, 0, -1, -1}, {0, 3, 0, -1, -1, -1}, {0, 0, 2, 0, -1, -1},
    {0, -1, 0, 2, -1, 0}, {-1, -1, -1, -1, 5, -1}, {-1, -1, -1, 0, -1, 4}}
L2 = {{2, 0, -1, -1, 0, 0}, {0, 3, 0, -1, -1, -1}, {-1, 0, 2, -1, 0, 0},
    {-1, -1, -1, 4, -1, 0}, {0, -1, 0, -1, 2, 0}, {0, -1, 0, 0, 0, 1}}

fdist[gamma_] :=
 Module[{L, lambda},
  L = ArrayFlatten[{{(L1 + gamma*IdentityMatrix[6]), -gamma*IdentityMatrix[6]},
    {-gamma*IdentityMatrix[6], (L2 + gamma*IdentityMatrix[6])}}];
  Eigs = Eigenvalues[L];
  Som = 0;
  Do[Do[Som +=
     If[Abs[Eigs[[i]] - Eigs[[j]]] == 0, 0, 1/Abs[Eigs[[i]] - Eigs[[j]]]]
    , {j, 1, 12}]
   , {i, 1, 12}];
  Return[Som]
  ]
```

A.5 Mixing Times of a Small Multiplex Graph

To compute the mixing times of the continuous quantum walk starting in the red vertex of the graph in figure 10, the following Python code was used.

```
import networkx as nx
import numpy as np
from scipy.linalg import expm
import scipy.sparse as sparse

def MakeH(G_top, G_bottom, gamma2):
    #The 2 layers of the multiplex graph
    L1 = sparse.csr_matrix.todense(nx.laplacian_matrix(G_top))
    L2 = sparse.csr_matrix.todense(nx.laplacian_matrix(G_bottom))

    #Making off-diagonal part
    N_nodes_layer = len(G_top.nodes)
    OD = gamma2*np.identity(N_nodes_layer)
    #Making matrix H
    H = np.block([[L1 + OD, -OD],[-OD, L2 + OD]])
    return H

#Number of nodes per layer
N_nodes_layer = 6

#List of values of gamma2
gamma2list = np.logspace(-3, np.log10(300), 10000)

#Making top layer
G_top = nx.Graph()
```

```
G_top.add_nodes_from([1,2,3,4,5,6])
G_top.add_edges_from([(1,5),(1,6),(2,4),(2,5),(2,6),(3,5),(3,6),(4,5),(5,6)])

#Making bottom layer
G_bottom = nx.Graph()
G_bottom.add_nodes_from([1,2,3,4,5,6])
G_bottom.add_edges_from([(1,3),(1,4),(2,4),(2,5),(2,6),(3,4),(4,5)])


#Stepsize
dt = 0.05
#Steps we take to reach the average
N_steps = 20000

#Computing the mixing times for the different gamma2s
#index of for loop
ind = 0
#mix time array
mix_time = np.zeros(len(gamma2list))
for gamma2 in gamma2list:
    #Make Laplacian
    H = MakeH(G_top, G_bottom, gamma2)
    #Operator
    U_sp = expm(-1j*H*dt)
    #Computing average distribution
    particle = np.zeros(N_nodes_layer*2)
    particle[2] = 1
    particle_t = np.copy(particle)
    av_dist = np.abs(particle_t)**2
    for i in range(N_steps):
        particle_t = U_sp.dot(particle_t)
        av_dist += np.abs(particle_t)**2
        print(i, gamma2, N_nodes_layer)
    av_dist = av_dist/(N_steps+1)

    #Mixing time
    particle_t = np.copy(particle)
    #Step taken during loop to compute mixing time
    step = 0
    #average distribution to compare to av_dist
    sum_dist = np.abs(particle_t)**2
    #Distance between average distribution and stationary distribution
    epsilon = np.linalg.norm(sum_dist - av_dist, ord = 1)
    #Number of steps epsilon has been smaller than 0.1
    steps_under = 0
    if epsilon < 0.1:
        steps_under += 1
    while steps_under < 100:
        step += 1
        particle_t = U_sp.dot(particle_t)
        sum_dist += np.abs(particle_t)**2
        epsilon = np.linalg.norm(sum_dist/(step + 1) - av_dist, ord = 1)
        if epsilon < 0.1:
            steps_under += 1
        else:
            steps_under = 0
```

```
    #Save mixing time
    mix_time[ind] = step - 99
    ind += 1

mix_time = mix_time*dt
```

A.6 Mixing Times of a 2-Layer Erdős–Rényi Multiplex Graph

To compute the mixing time of a continuous quantum walk on 2-layer multiplex graphs where each layer is generated by the Erdős–Rényi model as described in section 5.4, the following Python code was used. The function MakeH() from the last section is necessary to run this code.

```
import networkx as nx
import numpy as np
from scipy.linalg import expm
import scipy.sparse as sparse

def MixTimeAverage(gamma2, N_nodes_layer):
    #List of average mixing times
    mix_average_list = np.zeros(N_graphs)
    for g in range(N_graphs):
        #We want a connected graph
        con = False
        #Making layers
        while con == False:
            #Can be quite slow
            G_top = nx.erdos_renyi_graph(N_nodes_layer, 1/2)
            G_bottom = nx.erdos_renyi_graph(N_nodes_layer, 1/2)

            #Making multilayer graph
            con = (nx.is_connected(G_top) and nx.is_connected(G_bottom))

        #Make Laplacian
        H = MakeH(G_top, G_bottom, gamma2)
        #Stepsize
        dt = 0.05
        #Steps we take
        N_steps = 10000
        #Operator
        U_sp = expm(-1j*H*dt)
        #Computing average distribution
        particle = np.zeros(N_nodes_layer*2)
        particle[0] = 1
        particle_t = np.copy(particle)
        av_dist = np.abs(particle_t)**2
        for i in range(N_steps):
            particle_t = U_sp.dot(particle_t)
            av_dist += np.abs(particle_t)**2
            print(i, g, gamma2, N_nodes_layer)
        av_dist = av_dist/(N_steps+1)

        #Mixing time
        particle_t = np.copy(particle)
        #Step taken during loop to compute mixing time
        step = 0
        #average distribution to compare to av_dist
        sum_dist = np.abs(particle_t)**2
```

```
        #Number of steps epsilon has been smaller than 0.05
        steps_under = 0
        #Distance between average distribution and stationary distribution
        epsilon = np.linalg.norm(sum_dist - av_dist, ord = 1)
        if epsilon < 0.05:
            steps_under += 1
        while steps_under < 100:
            step += 1
            particle_t = U_sp.dot(particle_t)
            sum_dist += np.abs(particle_t)**2
            epsilon = np.linalg.norm(sum_dist/(step + 1) - av_dist, ord = 1)
            if epsilon < 0.05:
                steps_under += 1
            else:
                steps_under = 0
        mix_time = step - 99
        mix_average_list[g] = mix_time

    #Return mean and standard deviation
    mix_average = np.mean(mix_average_list)
    mix_average_std = np.std(mix_average_list)
    return (mix_average, mix_average_std)

#Number of graphs to average over
N_graphs = 1000
#Number of nodes per layer
N_list = np.array([300])
#List of different values of gamma2
gamma2_list = np.array([0, 0.005, 0.01])

start = time()
#Array to store average mixing times in
data = np.zeros((len(gamma2_list), len(N_list)))
data_std = np.zeros((len(gamma2_list), len(N_list)))
#Finding all mixing times for different gammas and nodes
for g in range(len(gamma2_list)):
    for n in range(len(N_list)):
        res = MixTimeAverage(gamma2 = gamma2_list[g],
                             N_nodes_layer = N_list[n])
        data[g][n] = res[0]
        data_std[g][n] = res[1]
```

B Labeling of Discrete Quantum Walks on General Graphs

To perform a discrete quantum random walk on a graph one first needs to find a labeling as explained in section 4.1. In the case of a hypercube one can easily find a correct labeling by assigning an edge the label i, if the n-bit strings of its endpoints differ in position i. In this way we find a labeling with for each edge the same label at its endpoints. For general graphs it is not always possible to assign the same label to an edge at both its endpoints, since this could lead to two different edges incident to a vertex having the same label on the side of this vertex. See for example figure 16. Hence in general the two endpoints of an edge will have different labels. One might think that to label such more general

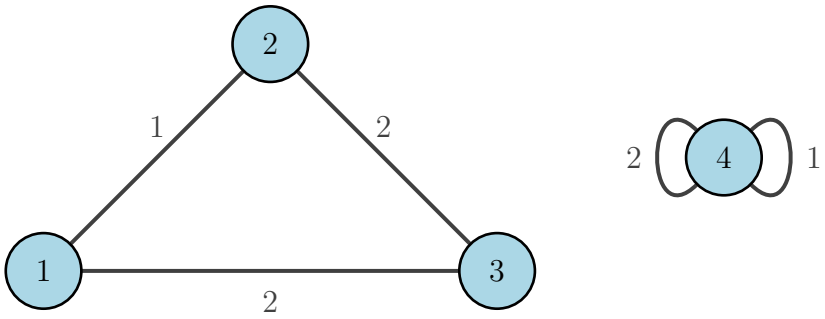

Figure 16: This figure shows a graph with a labeling that is not allowed. Both edges incident to vertex 3 have label 2 on the side of vertex 3.

graphs one can simply take each vertex of the graph and give all incident edges a distinct label on the side of that vertex. It turns out that this method of labeling the edges does not always give an allowed labeling, since the shift operator defined in equation 4.1 might not be unitary. To see this we can look at figure 17. Suppose we are in the state $\frac{1}{\sqrt{2}}(|1\rangle \otimes |2\rangle + |1\rangle \otimes |3\rangle)$, so we are in a superposition of being at node 2, label 1 and node 3, label 1. If we now apply the shift operator as defined in equation 4.1, we will end up in the state $\frac{2}{\sqrt{2}} |1\rangle \otimes |1\rangle$, which is not normalized.

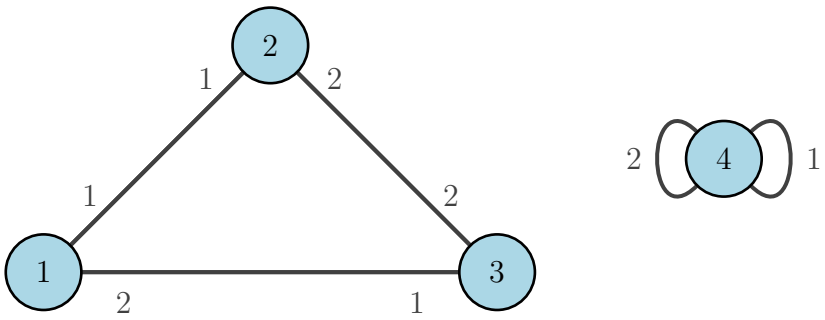

Figure 17: This figure shows a graph with a labeling that is not allowed. When applying the shift operator to the state $\frac{1}{\sqrt{2}}(|1\rangle \otimes |2\rangle + |1\rangle \otimes |3\rangle)$, we end up in the state $\frac{2}{\sqrt{2}} |1\rangle \otimes |1\rangle$, which has norm $\sqrt{2}$.

The problem with the labeling in figure 17 is that there is a vertex, vertex 1, with two incident edges that have the same label on the sides of its neighbours, vertices 2 and 3. In this way, two different states, same label, but different vertex, will be mapped to the same state by the shift operator, hence the shift operator is not unitary anymore. Thus, the condition on the labeling that guarantees the shift operator to be unitary is that for every vertex, the labels of its incident edges on the sides of its neighbours should be distinct.

It turns out that such a labeling can always be found. We can always rewrite a given graph, possibly with added self-loops, as a d-regular directed graph by replacing each edge between two distinct vertices by two directed edges and every self-loop by one directed edge, see for example figure 18. If we now can label all directed edges with labels $1, \ldots, d$, such that for each vertex all outgoing edges have a distinct label and all incoming edges have a distinct label, but an outgoing and incoming edge may have the same label, then we have found a correct labeling for our original undirected graph by mapping the label of the directed edge *(v,w)* to the undirected edge *(v,w)* on v's side.

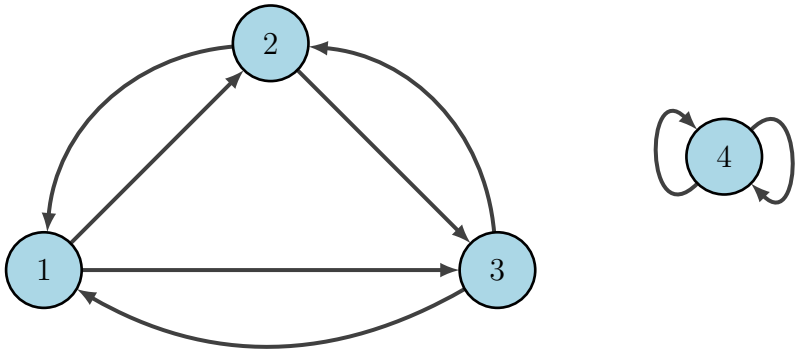

Figure 18: The graph of figure 17 is rewritten as a directed graph. To find a correct labeling of the original graph using labels $1, 2$, we must label for every vertex all outgoing edges with a distinct label and all incoming edges with a distinct label. An incoming and outgoing edge, however, may have the same label. There is a bijective mapping between the labeling of this graph and its undirected counterpart by assigning the label corresponding to a directed edge (v, w) to the undirected edge (v, w) on the side of v.

To find a correct labeling, we can rewrite this directed graph as a d-regular bipartite graph[hp]. First we double the vertices, so the first set of the bipartite graph contains all vertices of the original graph $\{v_i : i = 1, \ldots, N\}$ and the second set contains their duplicates $\{v'_i : i = 1, \ldots, N\}$. Now for every directed edge (v, w), draw an edge between v and w', such that there exists a one-to-one correspondence between the edges of the bipartite graph and the directed graph. See figure 19 for an example.

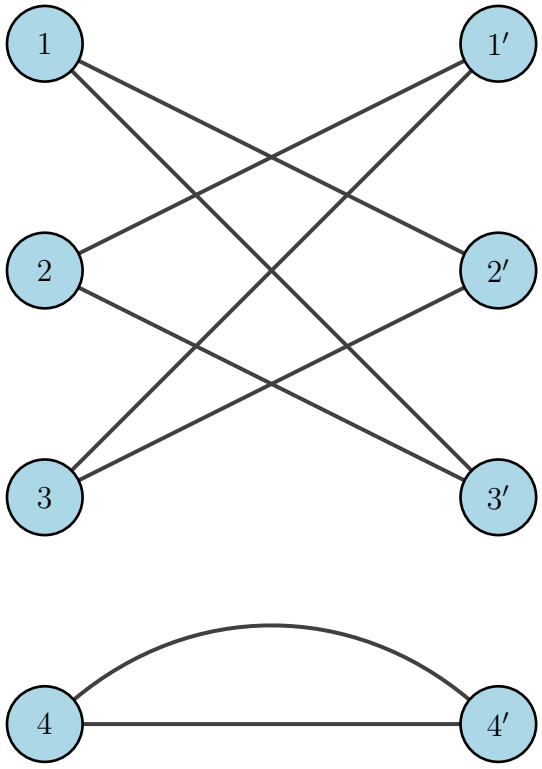

Figure 19: The graph of figure 18 is rewritten as a directed graph. The vertices are doubled and every directed edge (v, w) corresponds to an edge (v, w') in the bipartite graph. When we label the edges with the labels $1, 2$ in such a way that for every vertex all incident edges have a distinct label, we will have found a correct labeling for the directed graph and hence for our original graph.

If we can label the edges with d labels in such a way that for every vertex all incident edges have a distinct label, we have found a correct labeling for our directed graph and thus a correct labeling for our original graph. Indeed, for a vertex v in the directed graph all outgoing edges are incident to vertex v in the bipartite graph and must therefore have a different label. Similarly, all incoming edges correspond to the edges incident to vertex v' in the bipartite graph and must also have a distinct label. This problem of labeling the edges of a bipartite graph is a well-known edge-colouring problem and there exist fast algorithms that solve this problem in $\mathcal{O}(E \log(E))$[Alo03] or even $\mathcal{O}(E \log(d))$[COS01] time, where E and d are the number of edges and degree of the bipartite graph respectively. However, repeatedly applying the Hopcroft–Karp–Karzanov algorithm[HK73], which takes at most $\mathcal{O}(E\sqrt{V}d)$ time, might be easier, since it is already implemented in the NetworkX Python library.

C Completing the Proof of Theorem 4.1

To complete the proof of theorem 4.1, we must prove the following claim.

Claim C.1. *For any $\epsilon > 0$ and $r_i \in (0, \infty)$, $\theta_i \in (0, 2\pi)$, $i = 1, 2, \ldots, D$, there exist infinitely many positive integers N, such that*

$$2\sum_{i=1}^{D} r_i^2(1 - \cos(\theta_i N)) = \sum_{i=1}^{D} \left|r_i e^{-i\theta_i N} - r_i\right|^2 < \epsilon.$$

To do this, we will use Poincaré's recurrence theorem as proven in [BV13a].

Theorem C.1 (Poincaré's recurrence theorem)**.** *Let (X, Σ, μ) be a finite measure space. Let $f : X \to X$ be a measurable map and let μ be an f-invariant measure on X. For each set $E \in \Sigma$, we have*

$$\mu(x \in E : f^n(x) \in E \text{ for infinitely many values of } n) = \mu(E).$$

Now we can prove the claim.

Proof of claim C.1. Let $X = (0, 2r_1) \times (0, 2\pi) \times (0, 2r_2) \times \cdots \times (0, 2r_D) \times (0, 2\pi)$. Consider the measure space $(X, \mathcal{B}(X), \mu)$, where $\mathcal{B}(X)$ is the Borel σ-algebra and μ is the Lebesgue measure. Now consider the set

$$A = \{(c_1, \phi_1, \ldots, c_D, \phi_D) \in X : \sum_{i=1}^{D} \left|c_i e^{i\phi_i} - r_i\right| < \frac{\epsilon}{2}\}.$$

This set is open and thus an element of $\mathcal{B}(X)$; to see this, consider the standard metric on X and $\rho(x, y) = \sum_{i=1}^{D} |x_i - y_i|$ on $\mathbb{C}^D$. Since there is a continuous map $g : X \to \mathbb{C}^D$ defined by $g((c_1, \phi_1, \ldots, c_D, \phi_D)) = (c_1 e^{i\phi_1}, \ldots, c_D e^{i\phi_D})$ under which A is the preimage of an open ball, A must be open.

Now consider the map $f : X \to X$ defined by

$$f((c_1, \phi_1, \ldots, c_D, \phi_D)) = (c_1, \phi_1 - \theta_1 \bmod 2\pi, \ldots, c_D, \phi_D - \theta_D \bmod 2\pi).$$

Since this map only shifts points in X it preserves μ as shown in [BV13b]. Using theorem C.1, we conclude that there must be a point $x \in A$, such that there are infinitely many $N > 0$ for which it holds that $f^N(x) \in A$. In other words, there exists a vector $(c_1, \phi_1, \ldots, c_D, \phi_D)$, such that

$$\sum_{i=1}^{D} \left|c_i e^{i\phi_i} - r_i\right| < \frac{\epsilon}{2}$$

and

$$\sum_{i=1}^{D} \left|c_i e^{i(\phi_i - \theta_i N)} - r_i\right| < \frac{\epsilon}{2}.$$

So for infinitely many positive integers N

$$\sum_{i=1}^{D} \left|r_i e^{-i\theta_i N} - r_i\right|^2 \leq \sum_{i=1}^{D} \left|r_i e^{-i\theta_i N} - r_i\right| < \sum_{i=1}^{D} \left|r_i e^{-i\theta_i N} - c_i e^{i(\phi_i - \theta_i N)}\right| + \sum_{i=1}^{D} \left|c_i e^{i(\phi_i - \theta_i N)} - r_i\right| < \epsilon,$$

where the first inequality holds, because when $\epsilon \leq 1$, the terms of the sum must be less than or equal to 1 as well, and we only need to consider small ϵ. □

References

[ABN+01] Andris Ambainis, Eric Bach, Ashwin Nayak, Ashvin Vishwanath, and John Watrous. One-dimensional quantum walks. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, 12 2001.

[ACNR22] Simon Apers, Shantanav Chakraborty, Leonardo Novo, and Jérémie Roland. Quadratic speedup for spatial search by continuous-time quantum walk. *Phys. Rev. Lett.*, 129:160502, Oct 2022.

[Alo03] Noga Alon. A simple algorithm for edge-coloring bipartite multigraphs. *Information Processing Letters*, 85(6):301–302, 2003.

[Bar11] Marc Barthélemy. Spatial networks. *Physics Reports*, 499(1):1–101, 2011.

[BL57] P. Bocchieri and A. Loinger. Quantum recurrence theorem. *Phys. Rev.*, 107:337–338, Jul 1957.

[BV13a] Luis Barreira and Claudia Valls. *Dynamical Systems: An Introduction*, page 188. Springer Verlag, London, first edition, 2013.

[BV13b] Luis Barreira and Claudia Valls. *Dynamical Systems: An Introduction*, page 185. Springer Verlag, London, first edition, 2013.

[CLR20] Shantanav Chakraborty, Kyle Luh, and Jérémie Roland. How fast do quantum walks mix? *Physical review letters*, 124(5):050501, 2020.

[COS01] Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-coloring bipartite multigraphs in o(e logd) time. *Combinatorica*, 21(1):5–12, Jan 2001.

[GA17] Geoffrey R. Grimmett and Welsh D J A. *Random Walks*, page 167–168. Oxford University Press, 2017.

[GDGGG+13] S. Gómez, A. Díaz-Guilera, J. Gómez-Gardeñes, C. J. Pérez-Vicente, Y. Moreno, and A. Arenas. Diffusion dynamics on multiplex networks. *Physical Review Letters*, 110(2), 2013.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[hp] Fedor Petrov (https://mathoverflow.net/users/4312/fedor petrov). Directed edge-colouring. MathOverflow. URL:https://mathoverflow.net/q/232288 (version: 2016-02-27).

[Kem03] Julia Kempe. Quantum random walks: An introductory overview. *Contemporary Physics*, 44(4):307–327, 2003. Funding Information: The work on this article has drawn great benefits from fruitful discussions with Dorit Aharonov, Andris Ambai-nis, Andrew Childs, Richard Cleve, Eddy Farhi, Vivian Kendon, Neil Shenvi, Ben Tregenna, Umesh Vazirani and especially Peter Knight. JK's effort is sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-01-2-0524.

[Kem05] Julia Kempe. Discrete quantum walks hit exponentially faster. *Probability Theory and Related Fields*, 133(2):215–235, 2005.

[MPAD08] Franklin L Marquezino, Renato Portugal, Gonzalo Abal, and Raul Donangelo. Mixing times in quantum walks on the hypercube. *Physical Review A*, 77:042312, 4 2008.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bring order to the web. Technical report, Technical report, stanford University, 1998.

[PMD11] Giuseppe Davide Paparo and Miguel A. Martin-Delgado. Google in a quantum network. *Scientific Reports*, 2, 2011.

[POR19] RENATO PORTUGAL. *Quantum walks and search algorithms*. SPRINGER, 2019.

[Sch78] L. S. Schulman. Note on the quantum recurrence theorem. *Physical Review A*, 18(5):2379–2380, 1978.

[Sch99] Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, page 410, USA, 1999. IEEE Computer Society.

Lay Summary

Quantum mechanics has changed the way we look at our world. For most people it will be hard to grasp that in the quantum world a particle does not have a well-defined position but is in a combination of multiple positions, even though when we look at the particle, it takes on a specific position. Concepts such as these are confusing, yet they also hold much power and can be used to our own benefit. Quantum mechanical phenomena can be exploited in special types of computers called quantum computers. These devices are potentially capable of breaking widely used encryption schemes or search through databases much faster than any classical computer could. Such applications are based on quantum algorithms that can outdo classical algorithms on classical computers. To develop such new quantum algorithms, new sets of tools are needed that are not yet present in classical computing. Quantum random walks appear to be one such set of tools that can help in the development of quantum algorithms. They are the quantum versions of classical random walks, which are widely used in classical computing. In a classical random walk, a particle walks randomly over a line by repeatedly throwing up a coin. When the coin toss returns heads, the particle steps to the right; when the toss returns tails, the particle steps to the left. In a quantum random walk, the particle does not walk to the left or right, but walks both to the left and right as long as we don't look at it. Only when we look at the particle it takes on a specific position. It turns out that quantum random walks walk significantly faster than classical random walks and could therefore be useful in developing fast quantum algorithms. It is useful to know more about the tools you use and therefore it is important to examine more features of the quantum random walk, which is the subject of this report.