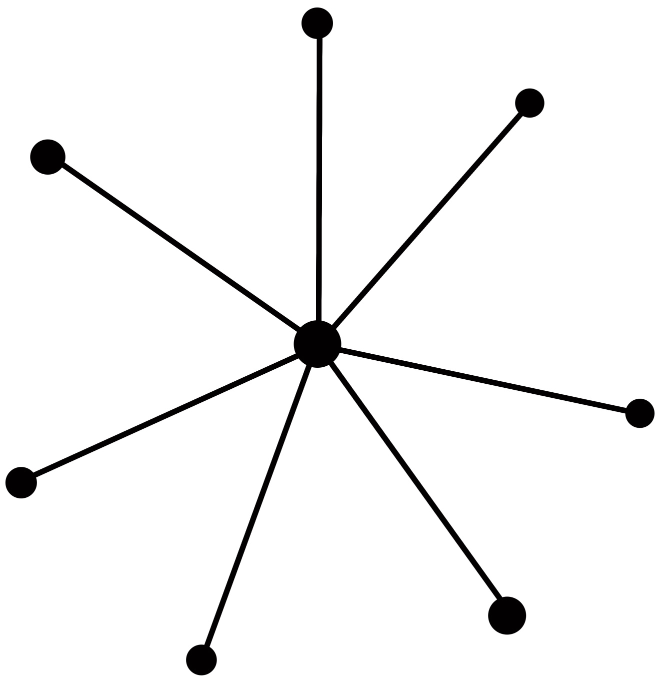
An alternating phase-walk on a star graph with percolation

Ralph Steller

Bachelor Thesis



Supervisor: Dr. Miriam Blaauboer
09/07/2025

Delft University of Technology

Abstract

Since their theoretical introduction in 1980, quantum computers have progressed significantly in both theory and experimentation. Quantum computers have the potential to solve certain problems significantly faster than their classical counterparts. One particularly promising area is search algorithms. In 1996, Lov Grover introduced a quantum search algorithm that laid the foundation for numerous variants. Among them, the alternating phase-walk stands out as a method for searching an element in an ordered list, which is modeled as a graph. However, a major challenge in practical quantum computing remains quantum decoherence.

The objective of this project was to model decoherence as bond percolation on a star graph during an alternating phase-walk, and to investigate its resulting effects. The introduction of decoherence transformed the search algorithm from a deterministic process into a probabilistic one, as the probability of measuring the marked state is no longer guaranteed to be 100% on every run of the algorithm. The most significant results were observed when varying the walk time in the continuous-time quantum walk (CTQW), both with and without a varying initial state. Here, the optimal time t_{opt} and its corresponding maximum average value $\mu_{\max}$ are presented for each value of p.

Static Initial State

- $N = 3$ **star graph:**
  $[p = 0.1,\ t_{\text{opt}} = 2.46\,\tau_3,\ \mu_{\max} = 0.67]$
  $[p = 0.5,\ t_{\text{opt}} = 0.62\,\tau_3,\ \mu_{\max} = 0.91]$
  $[p = 0.9,\ t_{\text{opt}} = 0.37\,\tau_3,\ \mu_{\max} = 1.00]$
- $N = 7$ **star graph:**
  $[p = 0.1,\ t_{\text{opt}} = 2.21\,\tau_7,\ \mu_{\max} = 0.56]$
  $[p = 0.5,\ t_{\text{opt}} = 0.59\,\tau_7,\ \mu_{\max} = 0.96]$
  $[p = 0.9,\ t_{\text{opt}} = 0.34\,\tau_7,\ \mu_{\max} = 0.99]$
- $N = 14$ **star graph:**
  $[p = 0.1,\ t_{\text{opt}} = 5.50\,\tau_{14},\ \mu_{\max} = 0.55]$
  $[p = 0.5,\ t_{\text{opt}} = 0.62\,\tau_{14},\ \mu_{\max} = 0.96]$
  $[p = 0.9,\ t_{\text{opt}} = 0.35\,\tau_{14},\ \mu_{\max} = 0.99]$

Varying Initial State

- $N = 3$ **star graph:**
  $[p = 0.1,\ t_{\text{opt}} = 2.30\,\tau_3,\ \mu_{\max} = 0.68]$
  $[p = 0.5,\ t_{\text{opt}} = 0.37\,\tau_3,\ \mu_{\max} = 0.70]$
  $[p = 0.9,\ t_{\text{opt}} = 0.36\,\tau_3,\ \mu_{\max} = 0.99]$
- $N = 7$ **star graph:**
  $[p = 0.1,\ t_{\text{opt}} = 2.30\,\tau_7,\ \mu_{\max} = 0.57]$
  $[p = 0.5,\ t_{\text{opt}} = 0.37\,\tau_7,\ \mu_{\max} = 0.66]$
  $[p = 0.9,\ t_{\text{opt}} = 0.33\,\tau_7,\ \mu_{\max} = 0.99]$
- $N = 14$ **star graph:**
  $[p = 0.1,\ t_{\text{opt}} = 6.42\,\tau_{14},\ \mu_{\max} = 0.58]$
  $[p = 0.5,\ t_{\text{opt}} = 0.39\,\tau_{14},\ \mu_{\max} = 0.60]$
  $[p = 0.9,\ t_{\text{opt}} = 0.34\,\tau_{14},\ \mu_{\max} = 0.98]$

Table of Contents

1 Introduction

In 1980, Paul Benioff published a paper that marked the beginning of a new field of research [1]. Benioff used the rules governing the quantum world to conceptualize a Quantum Turing Machine, a mathematical model for a computer based on quantum mechanics. This laid the foundation for an entirely new interdisciplinary field known as Quantum Computing. Since then, researchers have focused on how to build such machines, scale them, and develop the programs they might run.

Quantum computers operate fundamentally differently from classical computers. While classical computers use transistors and Boolean logic, quantum computers use qubits and operate based on principles from linear algebra. This difference enables quantum computers to solve certain problems that are intractable for classical machines. Notable fields where quantum computers have shown potential include cryptography, where Shor's algorithm can break current encryption schemes; chemical engineering—for simulating molecular interactions and aiding drug discovery [2]; and artificial intelligence, where quantum systems may improve training and optimization processes [3].

In this project, we focus on quantum search algorithms. Unlike classical search algorithms, which require checking each entry in an unsorted database sequentially, quantum search algorithms provide a significant speedup. Specifically, we examine a variant of a quantum search algorithm developed by Lov Grover. First published in 1996 [4], Grover's algorithm has since been experimentally implemented on IBM quantum computers [5], nuclear magnetic resonance (NMR) systems [6, 7, 8], and optical systems [9, 10]. The algorithm has also been improved, modified, and generalized in various ways [11, 12, 13, 14, 15, 16].

One such variant applies to searching on graphs, which are mathematical structures consisting of vertices (nodes) and edges (connections). For example, searching for a file on your computer can be modeled as navigating a graph: folders and files are vertices, and accessing a folder is represented by an edge. Grover's spatial variant, known as the alternating phase-walk, is designed for search tasks on such graphs [17]. It operates by transforming a wave function whose basis states correspond to the graph's vertices, such that the probability of measuring the desired (marked) node is amplified.

This project centers on the alternating phase-walk and its key component, the Quantum Walk. Although we use the quantum walk primarily as a tool for searching in this context, it also has broader applications in quantum information processing, multi-particle simulations, and other quantum algorithms [18].

However, building reliable quantum computers presents significant challenges. These systems are highly sensitive to environmental disturbances and must be cooled to extremely low temperatures to reduce errors. Nonetheless, noise remains a persistent problem. This phenomenon is known as quantum decoherence. It can disrupt the transmission of information, thereby ruining the search algorithm. In the context of the alternating phase-walk on graphs, we can model this with bond percolation, where the edges of the graph can be absent for a certain amount of time.

The aim of this project is to study the behavior of the alternating phase-walk on a special type of graph called the star graph, under the effects of percolation. To achieve this, we will first discuss the theoretical background: Quantum Walks, Grover's algorithm, the alternating phase-walk, and bond percolation. With the theory in place, we will describe how to simulate an alternating phase-walk, and then analyze the simulation results by varying key parameters of the system.

2 Theory

2.1 Quantum walks

In this section, we discuss the theory of quantum walks, the quantum analogues of classical random walks. We consider both the discrete-time and continuous-time formulations.

2.1.1 Classical discrete-time Markov chains & discrete-time quantum walks

To introduce classical discrete-time Markov chains, we begin with the example of a random walk. Consider a line with N points, or vertices, arranged along it. A coin is placed on one of these vertices as the starting position. At each time step, the coin is flipped, and depending on the outcome, it moves either to the left or to the right—to the neighboring vertex. This process is repeated multiple times (e.g., a dozen steps), and the final position is recorded. After each walk, the coin is reset to its original starting position, and the process is repeated many times. Over time, a distribution of final positions emerges, showing the probability of the coin ending up at each vertex after a given number of steps. A classical discrete-time Markov chain describes this kind of stochastic process, where the transition probabilities between states are fixed and do not depend on the previous steps.

Now that we have introduced the classical random walk, we turn to its quantum analogue—the quantum walk. In the quantum case, the physical coin is replaced by a quantum system described by a wavefunction in a Hilbert space. This quantum system could be, for example, the spin of an electron. The requirement is that it can be measured in two distinct states, analogous to the two outcomes of a coin flip. We call this quantum two-state system a *qubit.* The state space of the qubit is a two-dimensional Hilbert space, which we refer to as the *face Hilbert space* $\mathcal{H}_f$. It has an orthonormal basis corresponding to the two measurable states, which we denote by $|H\rangle$ and $|T\rangle$, analogous to heads and tails. The wavefunction of the qubit is a linear combination(a superposition) of these two basis states:

$$|C\rangle = \alpha\,|H\rangle + \beta\,|T\rangle\,, \quad \text{where } \alpha, \beta \in \mathbb{C},\ |\alpha|^2 + |\beta|^2 = 1. \tag{1}$$

Where α, β describe the amplitudes of the wavefunction, where squaring either α or β gives us the probability of measuring either $|H\rangle$ or $|T\rangle$.

The position of the qubit is also described by a wavefunction, now in position Hilbert space $\mathcal{H}_p$. Here $\{|n\rangle\,|n \in \mathbb{Z}\}$ will serve as the basis.

$$|P\rangle = \sum A_n\,|n\rangle \quad A_n \in \mathbb{C}, \sum |A_n|^2 = 1 \tag{2}$$

Here, A_n are the amplitudes.

To describe the whole system, we have to combine the two Hilbert spaces with an operation called a tensor product, written as $\mathcal{H} = \mathcal{H}_f \otimes \mathcal{H}_p$, with corresponding wavefunction $\psi = |C\rangle \otimes |P\rangle$. The big difference to the random walk is that the qubit is not actually walking on a line. The quantum walk is just a way of representing the transformation of the wavefunction, as if the coin is being flipped and walking on a line. This is important, as later in the theory section, the vertices and edges are not rooted in physical space and represent something else, but we can use the same model to describe them. The first transformation is what is going to 'flip' the coin, called the Hadamard matrix.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3}$$

However, now that we are working in a combined space, we also have to transform the position, but we are only transforming the wavefunction in face space. Thus, we transform it with an identity operator, leaving the wavefunction of position space the same.

$$F = H \otimes I_p \tag{4}$$

Then the position wavefunction is transformed based on $|H\rangle$ and $|T\rangle$:

$$W = |H\rangle\langle H| \otimes \sum |n+1\rangle\langle n| + |T\rangle\langle T| \otimes \sum |n-1\rangle\langle n| \tag{5}$$

Here we can see that $|H\rangle$ and $|T\rangle$ remain unchanged, but the position wavefunction is move to the left or to the right. The wavefunction evolves by flipping it first, then walking is written as the multiplication of the two operators. [19]

$$U = W(H \otimes I_p) \tag{6}$$

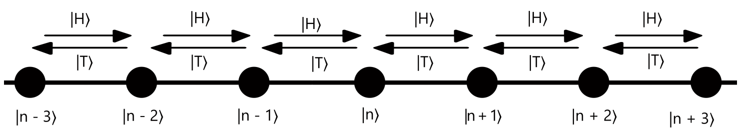


Figure (1) A discrete quantum walk. The arrows represent how $|H\rangle$ at position n will move to the right and $|T\rangle$ to the left.

2.1.2 Classical continuous-time Markov chains & continuous-time quantum walks

Quantum Walk on a Star Graph

We now extend the quantum walk model by modifying the discrete-time system and applying to a star graph (see Fig. 2) instead of a line. In this configuration, we arrange the vertices such that there are N outer vertices connected to a single central vertex. These vertices form the basis states of the position Hilbert space:

$$\{|c\rangle, |1\rangle, |2\rangle, |3\rangle, \ldots, |N\rangle\},$$

where $|c\rangle$ denotes the central vertex and $|i\rangle$ for $i = 1, 2, \ldots, N$ represent the outer vertices.

As in the earlier discrete-time case, the quantum walker's state evolves over this Hilbert space. The wavefunction can evolve such that amplitude flows from the center to the outer vertices and vice versa. However, in this model, we simplify the system by removing the explicit coin space and associated coin operators. Instead, the walk is governed directly by the evolution of amplitudes over time. We denote the time-dependent amplitudes at each vertex by $\phi_i(t)$, where $i = 0, 1, 2, \ldots, N$ and $\phi_0(t)$ corresponds to the central vertex. These amplitudes can be collected into a column vector:

$$\mathbf{p}(t) = \begin{pmatrix} \phi_c(t) \\ \phi_1(t) \\ \phi_2(t) \\ \vdots \\ \phi_N(t) \end{pmatrix}$$

This vector represents the full quantum state of the walker in position space at time t.

As before, we define our walking operator:

$$S_N = \sum_{x=1}^{N} |c\rangle\langle x| + |x\rangle\langle c| \tag{7}$$

We are so allowed to represent it in matrix form, called the adjacency matrix:

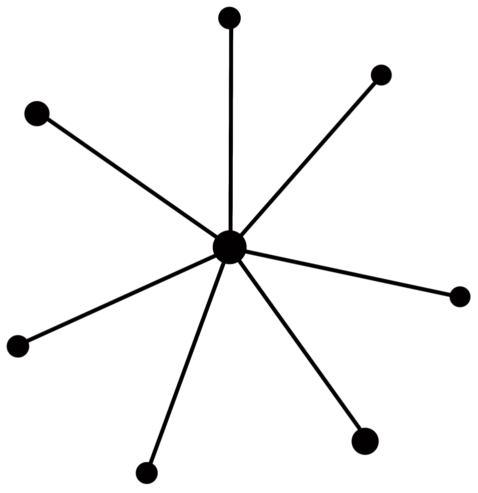


Figure (2) An example of a starnode graph with seven outer nodes and one center node.

$$S_N = \begin{pmatrix} 0 & 1 & 1 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix} \tag{8}$$

The probability distribution for the next time step is defined as:

$$\mathbf{p}_{t+1} = S_N \mathbf{p}_t \tag{9}$$

For continuous time, we replace the adjacency matrix (8) with a generator matrix [19, 20], and make the timesteps infinitesimally small. The generator matrix is just another way to represent the adjacency matrix (8)

$$H_{ij} = \begin{cases} 1 & i = 0, j > 0 \\ 1 & i > 0, j = 0 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

transforming equation (9) into a differential equation

$$\frac{dp_i(t)}{dt} = -\sum_j H_{ij} p_j(t) \tag{11}$$

by solving it you acquire:

$$\mathbf{p}(t) = e^{-Ht}\mathbf{p}(0) \tag{12}$$

Using this idea for a CQW, the generator matrix becomes the Hamiltonian of the system. By multiplying the generator matrix with i, it guarantees that Eq. 13 is unitary and hermitian.[19, 20]

$$U(t) = e^{-iHt} \tag{13}$$

2.2 Grover's algorithm

Searching for an item in an unordered list involves checking each element one by one until the target is found. For a list with N elements, the item is found, on average, after checking $\frac{N}{2}$ elements. This means the number of operations required grows linearly with the size of the list, which we denote as $O(N)$. Other algorithms may grow at different rates, such as $O(N^2)$, $O(2^N)$, or even $O(N!)$, where a small increase in input size can lead to a dramatic increase in computational cost. In contrast, our quantum search algorithm can search an unordered list in $O(\sqrt{N})$ time, offering a significant improvement over classical approaches.

Quantum computers currently do not yet possess the computational power of classical computers, so we will not be using them to search for files or perform large-scale computations. Instead, we scale the problem down to the fundamental components of a quantum system—qubits. In Eq. 1, the qubit was used to represent a quantum coin; here, it serves as the quantum analog of a classical bit.
Consider a quantum computer with three qubits. As discussed in Subsection 2.1.1, the wavefunctions of individual qubits are combined using the tensor product. This gives rise to a superposition over all possible $2^3 = 8$ basis states:

$$|\Psi\rangle = c_0\,|000\rangle + c_1\,|001\rangle + c_2\,|010\rangle + c_3\,|011\rangle + c_4\,|100\rangle + c_5\,|101\rangle + c_6\,|110\rangle + c_7\,|111\rangle \tag{14}$$

The idea of the algorithm is to transform the wave function that starts with uniform superposition into a wave function where the greatest amplitude lies with the marked element, a technique called amplitude amplification. As with classical computers it does this with logic gates, however now we have quantum gates, which are designed to transform the wavefunction. We shall not delve into the details of quantum gates, only the mathematical transformation they carry out.
The algorithm works in two parts that are iterated in pairs: The oracle and the diffuser. Both are a combination of quantum gates in order to perform a specific transformation. First we have the oracle, which inverts the amplitude for the given marked node, which is mathematically notated as:

$$U_f = I - 2\,|\omega\rangle\,\langle\omega| \tag{15}$$

Here $|\omega\rangle$ stands for the marked element. Next is the diffuser, where all the amplitudes are inverted around the average amplitude of all the states. In mathematical terms we reflect our wavefucntion around another wavefunction where the amplitudes are equal, just like we had at the start. $|s\rangle = \frac{1}{N}\sum^{N}|x\rangle$, we define this as:

$$U_w = 2\,|s\rangle\,\langle s| - I \tag{16}$$

As the amplitude of the marked state is inverted, the amplitudes of the non-marked elements remain largely unaffected, since they are initially very similar in magnitude. Only the marked element, now phase-flipped, gets significantly amplified. If the pair of operators (oracle and diffuser) is applied repeatedly, specifically for $\left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$ iterations, the probability of measuring the marked bit string becomes equal to or greater than $1 - \frac{1}{N}$ [19].

2.2.1 The alternating phase-walk: Grover's algorithm spatial variant

Instead of representing an unsorted list, we now associate each state of the wavefunction with a vertex on a star graph. For example, consider a star graph with 7 outer nodes and 1 central node, giving a total of 8 vertices. Using a 3-qubit quantum register, we can map the basis states as follows: $|000\rangle = |c\rangle$ represents the central node, while the outer nodes are labeled as $|001\rangle = |1\rangle, |010\rangle = |2\rangle, \ldots, |111\rangle = |7\rangle$. Grover's algorithm is then adapted to operate on this structured configuration. Specifically, the standard diffuser operator is replaced with a CTQW, as defined in Eq. 13. This modification follows the approach of Childs and Goldstone's spatial search algorithm [21], and is further explored in the context of alternating phase walks [17]. To implement the alternating phase walk, we apply the CTQW operator repeatedly, inverting the sign of the Hamiltonian at each step.

$$U_{grover} = \prod_{i=1}^{P} U_w[(-1)^i t_{N,P}] U_f \tag{17}$$

here t_N, p is defined as the optimal walk time for the CTQW.

$$t_{N,P} = \frac{2}{\sqrt{N}} \arcsin\left(\sqrt{N} \sin\left(\frac{\pi}{2(1+2P)}\right)\right) \tag{18}$$

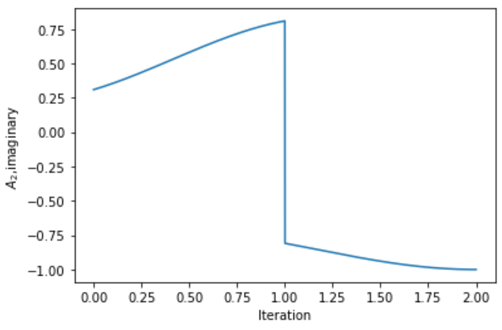
and the amount of pairs of operators or iterations we must apply.

$$P \geq \left\lceil \frac{\pi}{4}\sqrt{N} - \frac{1}{2} \right\rceil \tag{19}$$

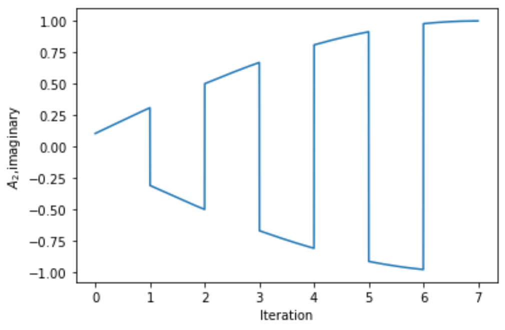
The initial state $|s\rangle$ is not an equal superposition of all the states, as in our original Grover's algorithm. Instead, it is constructed using a CTQW.

$$|s\rangle = U_w(t_{N,P}/2)\,|c\rangle \tag{20}$$

According to Qu et al., when we apply Eq. (18), (19), and (20), we measure the marked node a 100% every time we perform the algorithm, making it deterministic[22].



(a) 2 iterations steps for a 7 node star graph



(b) 7 iterations for a 7 node star graph

Figure (3) The evolution of the imaginary amplitude of $|2\rangle$, the marked node, during an alternating phase-walk on a 7-node star graph as a function of the iteration count. The amplitude grows over a certain time t and then flips at the beginning of the next iteration. Setting P to either 2 (the minimum number of iterations required for $N = 7$) or 7 results in the amplitude A_2 reaching 1.

2.3 Percolation theory & Decoherence

Lastly, we will discuss the theory of percolation, which is a powerful mathematical model that describes a wide range of phenomena and processes in physics, geography, biology, and many other fields. Percolation theory has several variants, each designed to model different types of systems. Examples of its applications include flow through porous materials, such as the diffusivity of gases [23] or electrical conductivity [24]. It also contributes to our understanding of protein structures [25].One important model is Fortuin–Kasteleyn (FK) percolation, which generalizes and unifies two models used to describe the behavior of ferromagnets. This model explains how ferromagnets undergo a phase transition and lose their magnetism when heated above a certain temperature [26]. Percolation theory can also be used to model the spread of disasters, such as the propagation of fire in a forest [27], and the transmission pattern of COVID-19 [28].

At its core, percolation theory describes the topology of graphs at microscopic and mesoscopic scales [29]. In this project, we focus on bond percolation. Below, we define bond percolation and explain its application to our quantum system and search algorithm.

When a quantum system is exposed to its external environment, it undergoes a process known as quantum decoherence. In general, decoherence refers to the loss of quantum information to the environment [30]. This phenomenon can manifest itself in various ways; however, in this context, decoherence is modeled as a modification to the continuous-time quantum walk (CTQW), which in turn alters the evolution of the wavefunction. Specifically, we represent decoherence by randomly removing edges from the graph, thereby disrupting the system's coherence. This type of model is known as bond percolation.

Each edge is associated with an independent Bernoulli random variable X_i, which takes the value 1 with probability p (indicating the presence of an edge), and 0 with probability $1-p$ (indicating the absence of an edge). The probability p is referred to as the *percolation parameter*. These variables are called independent because the realization of any X_i does not affect the probability distribution of the others. As a result, the original adjacency matrix in Equation (10) is replaced with a new matrix containing these random variables:

$$S_N = \begin{pmatrix} \prod_{i=1}^{N}(1-X_i) & X_1 & X_2 & \cdots & X_N \\ X_1 & 1-X_1 & 0 & \cdots & 0 \\ X_2 & 0 & 1-X_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ X_N & 0 & 0 & \cdots & 1-X_N \end{pmatrix} \tag{21}$$

Looking at the matrix in Equation (21), when $X_i = 0$, the corresponding diagonal entry is 1, meaning that the outer vertex retains its amplitude during that step. When all edges are absent (i.e., all $X_i = 0$), the central node holds the entire amplitude.

Bond percolation can manifest itself in two forms: static and dynamic. In static percolation, the graph is altered once at the beginning, and the continuous-time quantum walk (CTQW) proceeds over a fixed time interval. In dynamic percolation, the total walk time is divided into equal time steps Δt, with the graph potentially changing at each step. Since the matrix in Equation (21) varies with each time step, we cannot exponentiate a sum of matrices, as they generally do not commute. Instead, we must multiply the exponentials at each step as follows:

$$U_w(t) = e^{-iS_n^{(1)}\Delta t} e^{-iS_n^{(2)}\Delta t} e^{-iS_n^{(3)}\Delta t} \dots$$

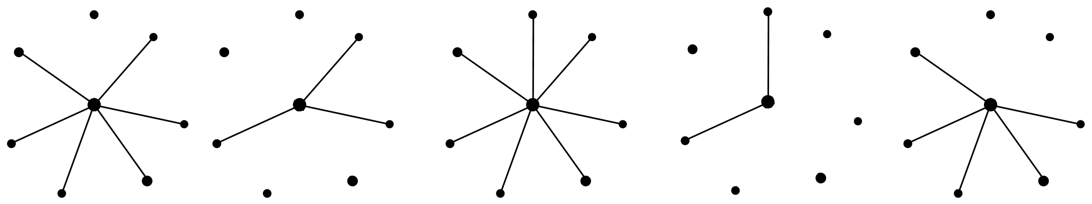


Figure (4) An example of what dynamical percolation could look like at a certain p value. Going from left to right the bond could be absent one time step and then be present the next.

3 Simulation

In this section, the process is outlined for how we are going to analyze the percolation star graph.

3.1 Methodology

To analyze the effects of decoherence in our system, we consider two categories of parameters: those inherent to the algorithm itself and those related to percolation dynamics. The algorithmic parameters include the number of outer vertices, the identity of the marked node, the number of evolution operator pairs used within the alternating phase-walk, and the number of times the algorithm is executed to allow for statistical analysis. Additionally, we account for the walk time of a quantum walk.

Regarding percolation, we introduce a percolation parameter that governs the probability of edge presence in the underlying graph. Furthermore, we consider the number of time steps over which the graph undergoes dynamical changes, representing the temporal aspect of percolation.

We will simulate the alternating phase-walk using Python, systematically varying the aforementioned parameters to observe their influence on the probability of measurement of the marked element P_ω.

To analyze the resulting data, we will apply statistical tools such as the Shapiro-Wilk test to assess normality [31] and the Savitzky-Golay filter for smoothing and trend extraction. The filter takes a window of fixed size over the data and fits a polynomial over it by a method of linear least squares and slides that window over the data[32].

The star graph is also known as a periodic graph, meaning there exists a time $\tau > 0$ for a quantum walk where for every vertex $v \mid \langle v| U_w(\tau) |v\rangle \mid = 1$. Solving the $U_w(t) |v\rangle$ we get:

$$\tau_N = \frac{2\pi}{\sqrt{N}} \tag{22}$$

This is going to be our base time.

3.2 Python Code

A simulation of the alternating phase-walk was implemented in Python; see Listing 1 and 2 in the appendix.

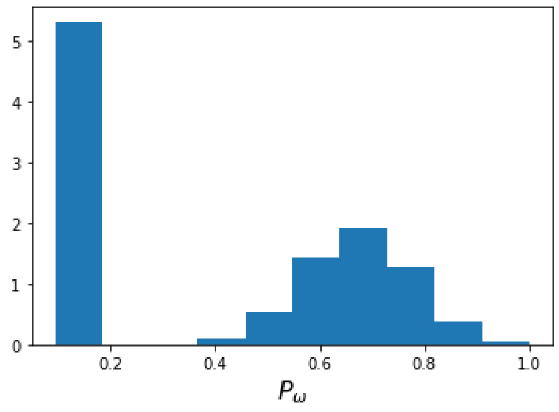
The code consists of two primary functions: one to generate the adjacency matrix (21)(with or without percolation), and another to perform the quantum walk. Percolation is simulated as follows: a bitstring of length N is generated using the `random.choices` function, where each bit is randomly selected as 0 or 1 based on the given percolation parameter. This bit string is inserted into the 0th row of an $(N+1) \times (N+1)$ matrix. Its transpose is placed in the 0th column. The inverted bitstring is then placed along the diagonal, excluding the $(0,0)$ element. Finally, the $(0,0)$ entry is set based on whether all bits in the string are zero (i.e., whether the center node is fully disconnected). This results in the construction of the percolation-modified adjacency matrix as defined in Eq. (21).

4 Results

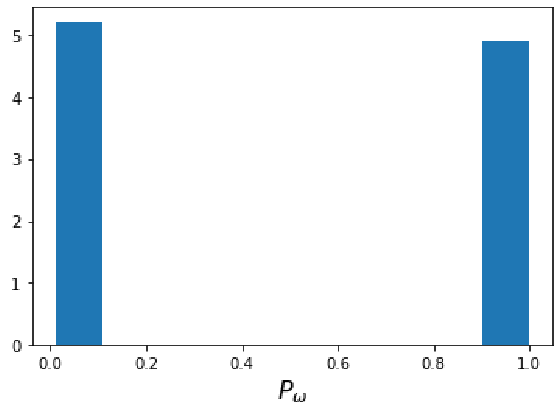
All results were produced by running the algorithm 200 times, and setting Δt at $\frac{1}{100}$ of the walk time.

4.1 Static percolation

We will first start with static percolation and how that affects the alternating phase-walk. This will be done on a N=7 star graph with optimal time $t_{7,2}$



(a) the amount of iterations set at 2.



(b) the amount of iterations set at 7.

Figure (5) Distribution of the probability of measuring $|\omega\rangle$ under static percolation with $p = 0.5$, shown for a graph with $N = 7$ nodes.

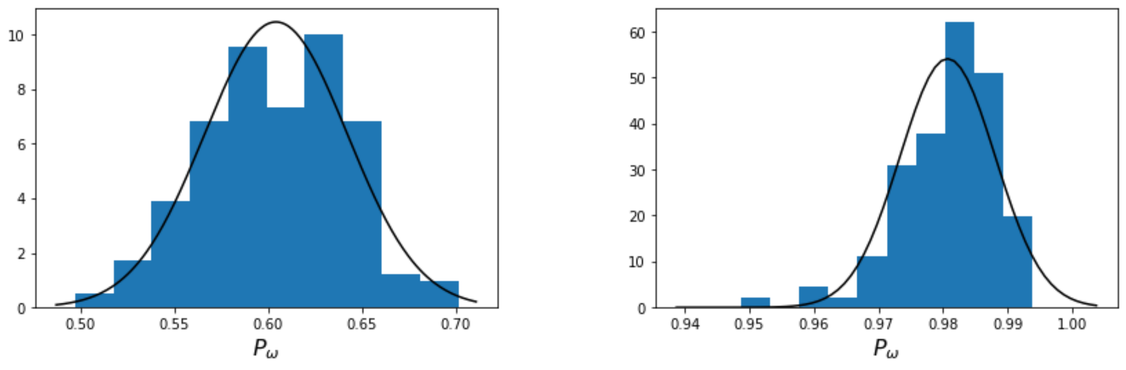
In Fig. 5, two different types of distributions emerge. In Fig. 5a, we observe that the marked node is either measured with a probability close to 0%, indicated by a single bar on the left, or with a broader distribution of measurement probabilities. Only after increasing the number of evolution operator pairs, does a Bernoulli distribution begin to appear, as shown in Fig. 5b. The singular bar on the left arises when the marked node becomes disconnected from the graph due to percolation; in such cases, it cannot be measured, resulting in a probability of the initial state. The transition from the broader distribution observed in Fig. 5a to the singular bar in Fig. 5b occurs because the graph effectively transforms into a smaller star graph after percolation. However, the initial state prepared for the original graph is not compatible with the smaller graph's structure. As a result, additional iterations of the quantum walk are necessary to adapt and partially compensate for this mismatch. Furthermore, we observe that the algorithm is no longer deterministic, as the probability of measuring the marked node becomes dependent on the percolation parameter p.

4.2 Dynamic Percolation

This section presents the results of dynamic percolation, with variations in the percolation parameter p, the star graph size N, and the walk time.

4.2.1 Percolation parameter vs P_ω

We will start with the distribution of P_ω. Here we run the alternate phase-walk on a 7 node star graph varying only p



(a) p=0.5 (b) p=0.9

Figure (6) The probability distribution and a normal fit(the black line) for the probability of measuring $|\omega\rangle$, for both p=0.5 and p = 0.9. on a N=7 star graph

When switching to dynamic percolation, we observe that the distribution changes from a Bernoulli distribution to a more continuous spread. Applying the Shapiro–Wilk test to both cases, we find that the distribution in Fig. 6a is likely to be normally distributed, with a p-value of 0.34. In contrast, Fig. 6b yields a p-value of $9.9 \cdot 10^{-5}$, leading to the rejection of the null hypothesis that the data follows a normal distribution.

The results that follow present the average values μ of the observed distribution and analyze how it varies with respect to the different parameters.

4.2.2 Percolation parameter vs μ

From the previous subsection, it was shown that switching to dynamic percolation results in a distribution of P_ω. Here, we examine the mean of this distribution and how it changes under dynamic percolation.

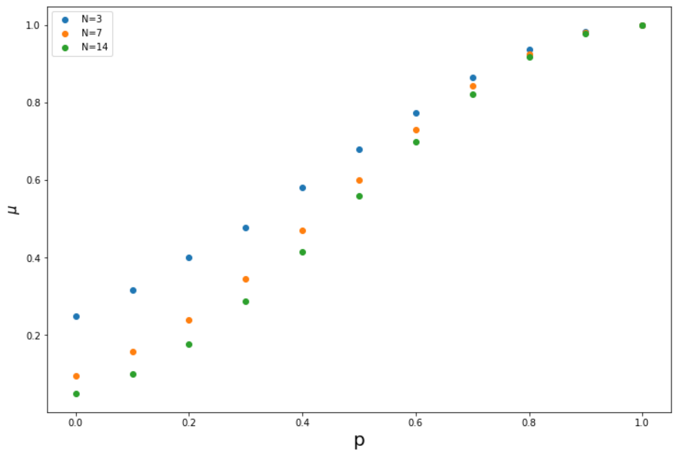


Figure (7) The percolation parameter p vs μ the average probability of finding the marked node, for the star graphs with outer nodes N=3,7,14.

At $p = 0$, the edges are absent at every time step, meaning the graph is entirely disconnected. In this case, the values of μ reflect the probabilities in the initial states. At the other extreme, when $p = 1$, the edges are always present, and all values converge to $\mu = 1$, without abscent bonds we get our original algorithm back. For all three N-star graphs considered, μ appears to level off as p approaches 1. This is likely due to the distribution becoming increasingly skewed, with the right tail compressing as more outcomes cluster near the maximum. However, this effect is not observed at $p = 0$, where the lower bound is governed by the initial state itself, reducing the impact of skewness.

4.2.3 Percolation parameter vs std

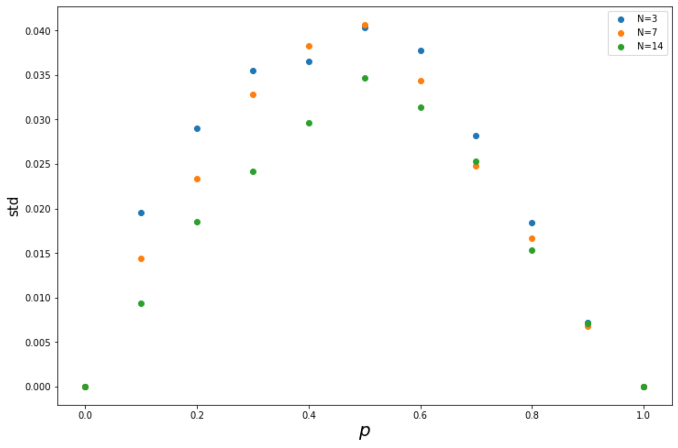


Figure (8) The percolation parameter p versus the standard deviation of the distribution of P_ω for star graphs with $N = 3$, 7, and 14 nodes.

Here we find that the standard deviation takes on a parabolic shape for different values of p. The peak of every Nth star graph is located at p=0.5.

4.2.4 N vs μ

Here the amount of nodes N was varied and how that affects the alternate phase-walk

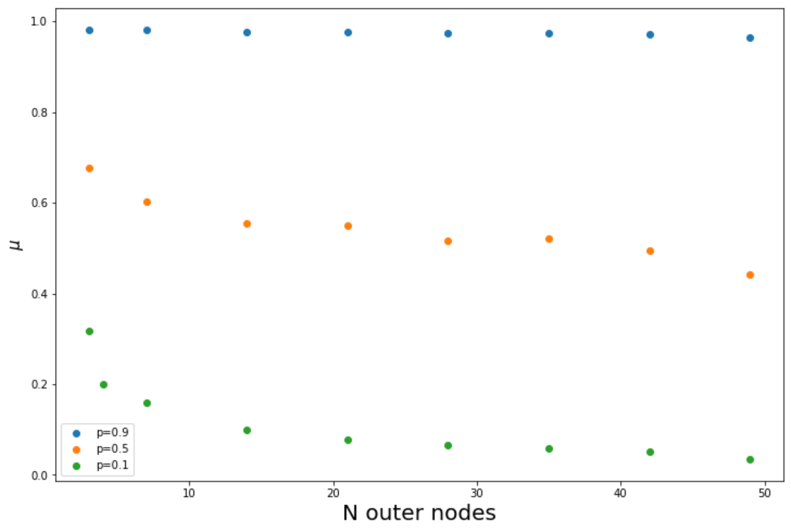


Figure (9) The number of outer nodes N versus μ, the average probability of finding the marked node, for different values of p.

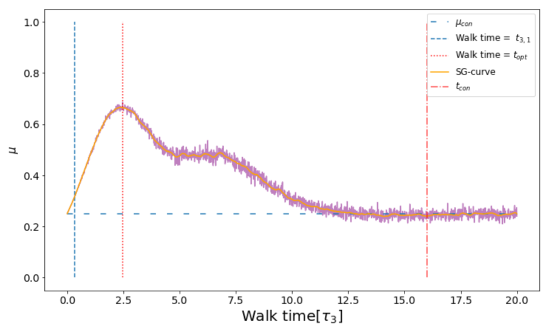
Increasing N seems to have an effect that μ converges to the value p.

4.2.5 Walk Time vs μ

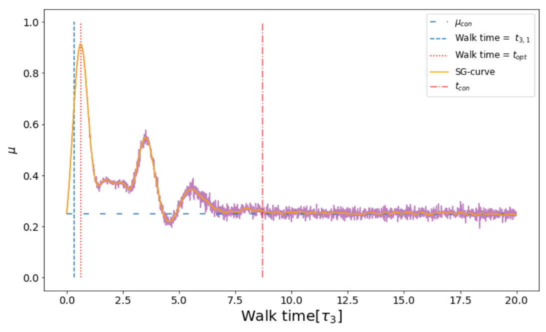
This subsection is dedicated to exploring how variations in the walk time of the quantum walk affect the average measurement probability μ. It is divided into two parts, based on how the initial state is constructed using a continuous-time quantum walk (CTQW). The first part examines the initial state independently of the walk time, while the second considers the case where the initial state depends on the walk time. Throughout these results, the walk time is scaled to τ_N.

For the purposes of this project, the initial state is always constructed in the absence of percolation. Percolation is applied only during the execution of the algorithm itself, not during the preparation of the initial state.

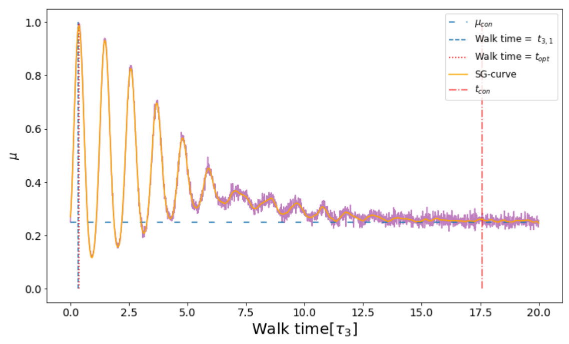
Static initial state These are the results of letting the initial state not vary by walk time, instead putting the value of $t_{N,P}$ from eq. 17.



(a) $p = 0.1$



(b) $p = 0.5$

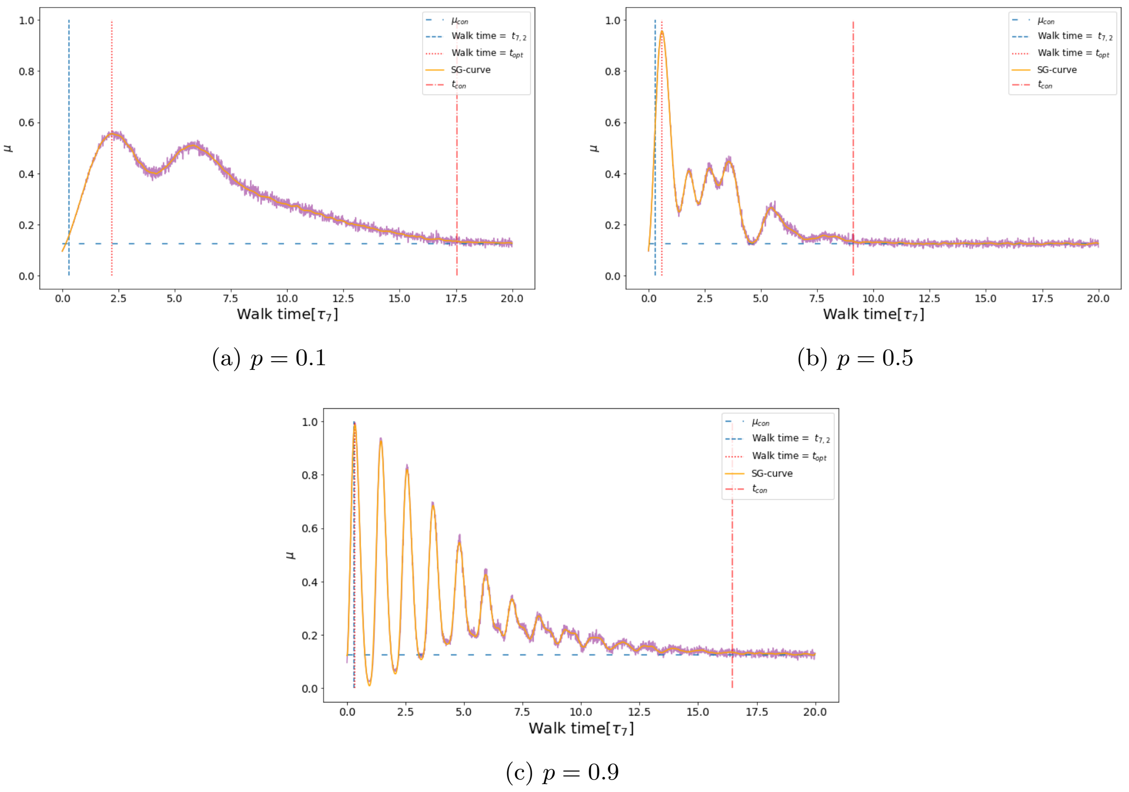


(c) $p = 0.9$

Figure (10) Walk time vs μ, for a N=3 star graph, with a static initial state. The walk time is extended up to 20 times the base time $\tau_3 = \frac{2\pi}{\sqrt{3}}$. In the plot, two vertical lines indicate the original optimal time in the absence of percolation, $t_{3,1}$, and the new optimal time, t_{opt}, corresponding to the maximum value of μ. A horizontal line marks the value to which μ converges, and an additional vertical line denotes the point at which this convergence occurs, t_{con}.The curve through the data is the Savitzky-Golay filter which smoothed out the data.

Table (1) Optimal time t_{opt} (where the maximum value $\mu_{\max}$ is reached), convergence time t_{con}, and the value of $\mu_{t_{3,1}}$ at the previous optimal time $t_{3,1}$ for an alternating phase-walk with a static initial state on an $N = 3$ star graph.

p	t_{opt} $[\tau_3]$	μ_{max}	$t_{con}[\tau_3]$	$\mu_{t_{3,1}}$
0.10	2.46	0.67	16.02	0.31
0.50	0.62	0.91	8.71	0.68
0.90	0.37	1.00	17.59	0.98

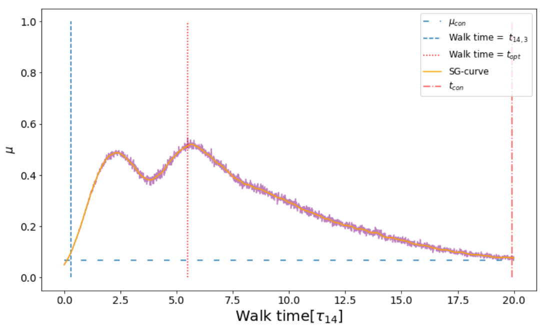


(a) $p = 0.1$ (b) $p = 0.5$ (c) $p = 0.9$

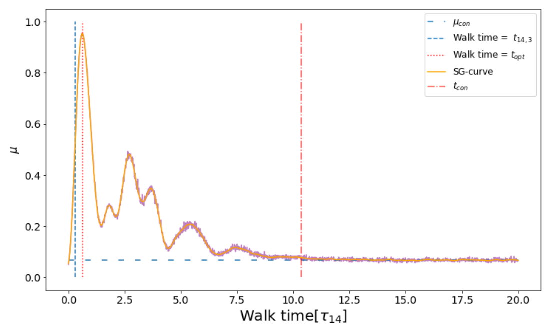
Figure (11) Walk time vs μ, for a N=7 star graph, with a static initial state. The walk time is extended up to 20 times the base time $\tau_7 = \frac{2\pi}{\sqrt{7}}$. In the plot, two vertical lines indicate the original optimal time in the absence of percolation, $t_{7,2}$, and the new optimal time, t_{opt}, corresponding to the maximum value of μ. A horizontal line marks the value to which μ converges, and an additional vertical line denotes the point at which this convergence occurs, t_{con}.The curve through the data is the Savitzky-Golay filter which smoothed out the data.

Table (2) Optimal time t_{opt} (where the maximum value $\mu_{\max}$ is reached), convergence time t_{con}, and the value of $\mu_{t_{7,2}}$ at the previous optimal time $t_{3,1}$ for an alternating phase-walk with a static initial state on an $N = 7$ star graph.

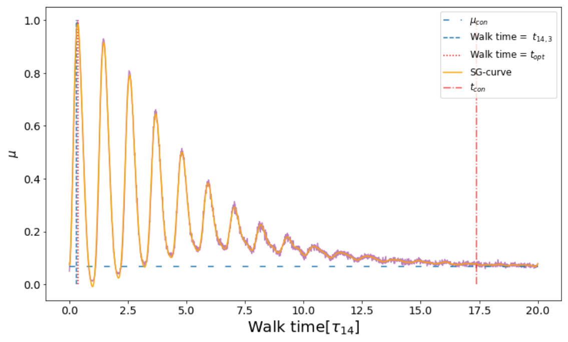
p	t_{opt} $[\tau_7]$	μ_{max}	$t_{con}[\tau_7]$	$\mu_{t_{7,2}}$
0.1	2.21	0.56	17.54	0.16
0.5	0.59	0.96	9.10	0.60
0.9	0.34	0.99	16.46	0.98



(a) $p = 0.1$



(b) $p = 0.5$



(c) $p = 0.9$

Figure (12) Walk time vs μ, for a N=14 star graph, with a static initial state. The walk time is extended up to 20 times the base time $\tau_{14} = \frac{2\pi}{\sqrt{14}}$. In the plot, two vertical lines indicate the original optimal time in the absence of percolation, $t_{14,3}$, and the new optimal time, t_{opt}, corresponding to the maximum value of μ. A horizontal line marks the value to which μ converges, and an additional vertical line denotes the point at which this convergence occurs, t_{con}.The curve through the data is the Savitzky-Golay filter which smoothed out the data.

Table (3) Optimal time t_{opt} (where the maximum value μ_{max} is reached), convergence time t_{con}, and the value of $\mu_{t_{14,3}}$ at the previous optimal time $t_{14,3}$ for an alternating phase-walk with a static initial state on an $N = 14$ star graph.

p	t_{opt} [τ_{14}]	μ_{max}	t_{con}[τ_{14}]	$\mu_{t_{14,3}}$
0.1	5.50	0.55	19.92	0.10
0.5	0.62	0.96	10.34	0.56
0.9	0.35	0.99	17.39	0.98

All points were calculated by taking $\frac{1}{100}$ of the base time τ_N and running the simulation for 2000 iterations, covering 20 full periods. The resulting data was smoothed using the Savitzky-Golay filter with a window length of 80 points and a polynomial order of 5. From the figures, we observe that for all values of N and p considered, the distributions converge after a certain period of time. The value to which they converge is approximately $\frac{1}{N+1}$, indicating that all states become equally probable.

We define the convergence time as the point, identified by counting backward, where the values of μ remain within 1% above $\frac{1}{N+1}$. This behavior contrasts with the non-percolated case, where increasing the walk time beyond the base time had no effect due to the periodicity of the graph. Here, however, the introduction of percolation breaks this periodicity and makes longer walk times meaningful.

A possible explanation for the case $p = 0.5$ is that it corresponds to the highest bond-fluctuation variance, leading to the most dynamic graph structure. In contrast, for $p = 0.1$ and $p = 0.9$, most bonds are either consistently absent or consistently present, leading to more stable (and thus predictable) behavior. All optimal times across different parameter configurations are summarized in Fig. 13.

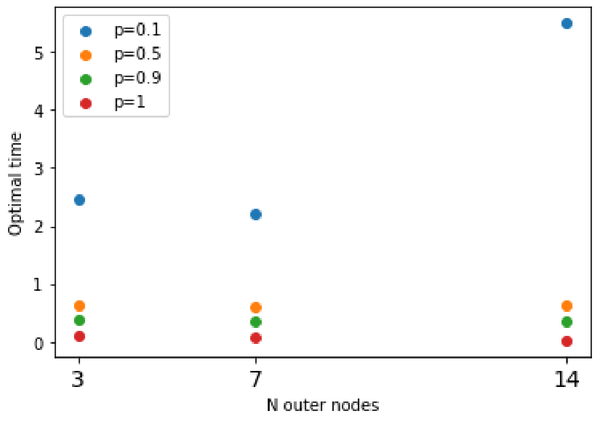
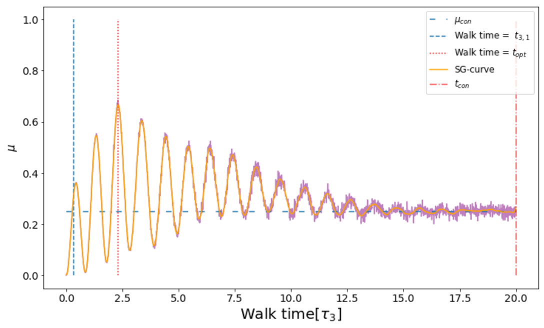


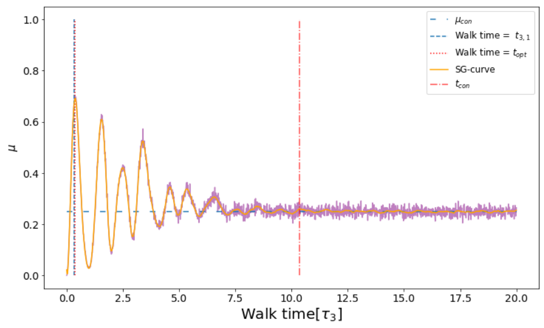
Figure (13) The optimal times for p = 0.1, 0.5, 0.9, 1 and for the nodes N=3, 7, 14 are scaled relative to the base time of their corresponding graph, with a static initial state.

From Fig 13. we can that as p increases, optimal times converge to the optimal time described with eq 18. As the percolation is increased more bonds will be present.

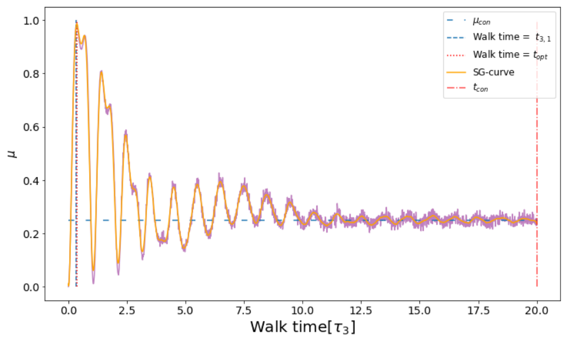
Initial state varied by walk time Here the initial state is varied by walk time. We will look at the same situation again for N = 3,7,14 and p = 0.1,0.5,0.9.



(a) $p = 0.1$



(b) $p = 0.5$

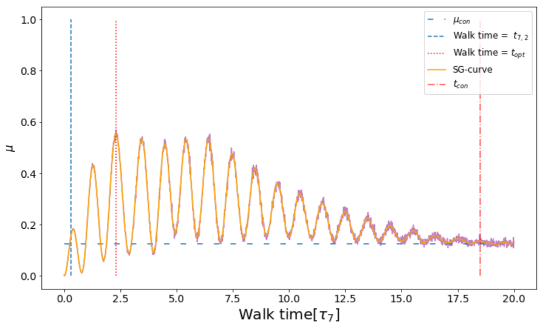


(c) $p = 0.9$

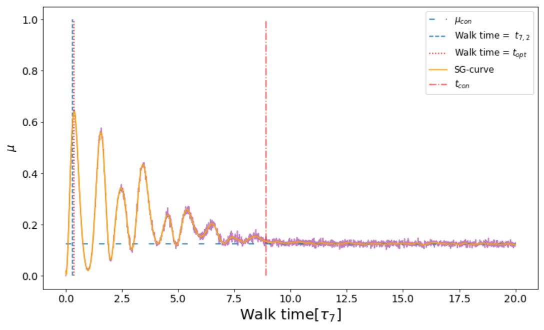
Figure (14) Walk time vs μ,for a N=3 star graph, with a varying initial state. The walk time is extended up to 20 times the base time $\tau_3 = \frac{2\pi}{\sqrt{3}}$. In the plot, two vertical lines indicate the original optimal time in the absence of percolation, $t_{3,1}$, and the new optimal time, t_{opt}, corresponding to the maximum value of μ. A horizontal line marks the value to which μ converges, and an additional vertical line denotes the point at which this convergence occurs, t_{con}.The curve through the data is the Savitzky-Golay filter which smoothed out the data.

Table (4) Optimal time t_{opt} (where the maximum value $\mu_{\max}$ is reached), convergence time t_{con}, and the value of $\mu_{t_{3,1}}$ at the previous optimal time $t_{3,1}$ for an alternating phase-walk with a varying initial state on an $N = 3$ star graph.

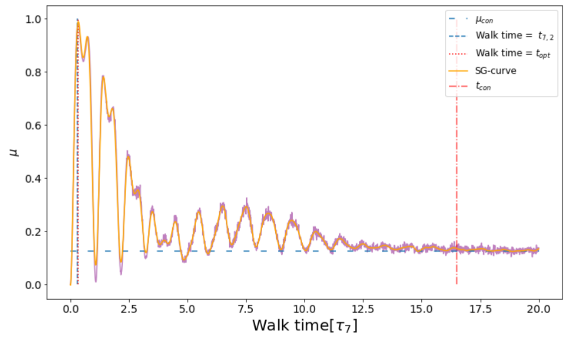
p	t_{opt} $[\tau_3]$	μ_{max}	$t_{con}[\tau_3]$
0.1	2.30	0.68	20.00
0.5	0.37	0.70	10.36
0.9	0.36	0.99	20.00



(a) $p = 0.1$



(b) $p = 0.5$

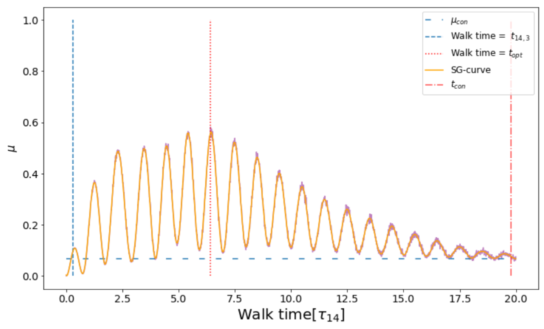


(c) $p = 0.9$

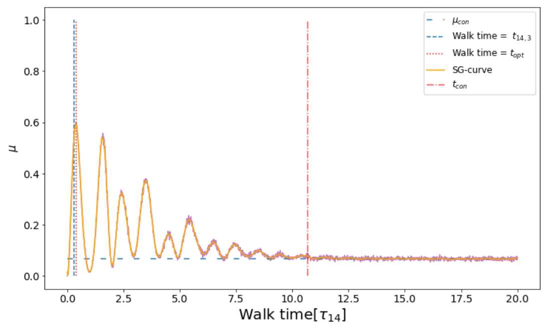
Figure (15) Walk time vs μ, for a N=7 star graph, with a varying initial state. The walk time is extended up to 20 times the base time $\tau_7 = \frac{2\pi}{\sqrt{7}}$. In the plot, two vertical lines indicate the original optimal time in the absence of percolation, $t_{7,2}$, and the new optimal time, t_{opt}, corresponding to the maximum value of μ. A horizontal line marks the value to which μ converges, and an additional vertical line denotes the point at which this convergence occurs, t_{con}.The curve through the data is the Savitzky-Golay filter which smoothed out the data.

Table (5) Optimal time t_{opt} (where the maximum value $\mu_{\max}$ is reached), convergence time t_{con}, and the value of $\mu_{t_{7,2}}$ at the previous optimal time $t_{7,2}$ for an alternating phase-walk with a varying initial state on an $N = 7$ star graph.

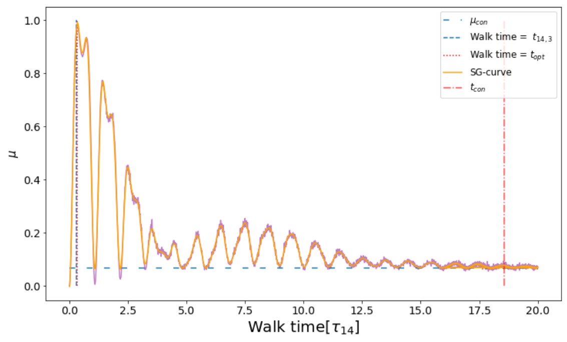
p	t_{opt} $[\tau_7]$	μ_{max}	t_{con} $[\tau_7]$
0.1	2.30	0.57	18.51
0.5	0.37	0.65	8.90
0.9	0.33	0.99	16.5



(a) $p = 0.1$



(b) $p = 0.5$



(c) $p = 0.9$

Figure (16) Walk time vs μ,for a N=14 star graph, with a varying initial state. The walk time is extended up to 20 times the base time $\tau_{14} = \frac{2\pi}{\sqrt{14}}$. In the plot, two vertical lines indicate the original optimal time in the absence of percolation, $t_{14,3}$, and the new optimal time, t_{opt}, corresponding to the maximum value of μ. A horizontal line marks the value to which μ converges, and an additional vertical line denotes the point at which this convergence occurs, t_{con}.The curve through the data is the Savitzky-Golay filter which smoothed out the data.

Table (6) Optimal time t_{opt} (where the maximum value μ_{max} is reached), convergence time t_{con}, and the value of $\mu_{t_{14,3}}$ at the previous optimal time $t_{14,3}$ for an alternating phase-walk with a varying initial state on an $N = 14$ star graph.

p	t_{opt} $[\tau_{14}]$	μ_{max}	$t_{con}[\tau_{14}]$
0.1	6.42	0.58	19.77
0.5	0.39	0.60	10.67
0.9	0.34	0.98	18.57

Switching to a varying initial state appears to have a noticeable impact on the behavior of μ. All graphs exhibit a sinusoidal pattern, with $p = 0.9$ for $N = 3, 7, 14$ showing dips in the upper amplitudes that were not present for a static initial state. Two aspects remain consistent with the static case: μ still converges to equal weight, and for $p = 0.9$ and $p = 0.5$, the new and old optimal times remain closely aligned.

Taking the values of t_{opt} from Tables 4, 5, and 6, we compile them into a single figure.

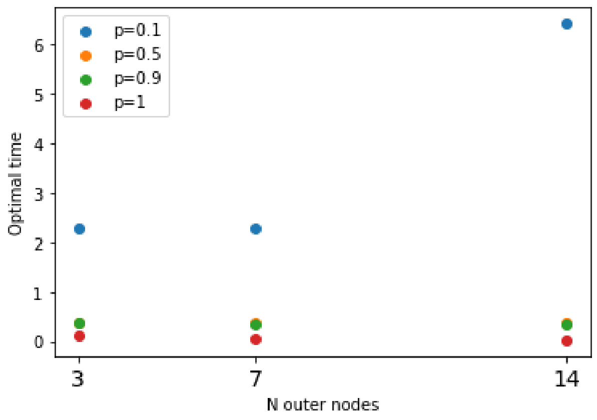


Figure (17) The optimal times for p = 0.1, 0.5, 0.9, 1 and for the nodes N= 3, 7, 14 are scaled relative to the base time of their corresponding graph, with a varying initial state.

With a varying initial state the optimal times for p= 0.5 and 0.9 converge faster to $t_{N,P}$.

5 Conclusion

Conclusion

The goal of this project was to model decoherence as bond percolation on a star graph while performing an alternating phase-walk, and to observe the resulting effects. Introducing decoherence transformed the search algorithm from a deterministic process into a probabilistic one, as the probability of measuring the marked state was no longer always equal to one.

In the case of static percolation, the probability of measuring the marked node was either 0 or 1 after enough iterations, rendering the search deterministic only a fraction p of the time. In contrast, dynamic percolation resulted in a distribution of probabilities for the measurement of the marked node at a given percolation parameter p. Varying p revealed an almost linear trend in μ.

The most significant result emerged when varying walk time. For a given p, choosing an optimal walk time t_{opt} significantly increased the probability of measuring the marked state from its old optimal time $t_{N,P}$. Two scenarios were considered: one with a static initial state and one with a varying initial state. The corresponding results for each case are summarized below.

Static Initial State

- **N = 3 star graph:** $[p = 0.1, t_{\text{opt}} = 2.46\tau_3,\ \mu_{\text{max}} = 0.67]$; $[p = 0.5, t_{\text{opt}} = 0.62\tau_3,\ \mu_{\text{max}} = 0.91]$; $[p = 0.9, t_{\text{opt}} = 0.37\tau_3,\ \mu_{\text{max}} = 1.00]$
- **N = 7 star graph:** $[p = 0.1, t_{\text{opt}} = 2.21\tau_7,\ \mu_{\text{max}} = 0.56]$; $[p = 0.5, t_{\text{opt}} = 0.59\tau_7,\ \mu_{\text{max}} = 0.96]$; $[p = 0.9, t_{\text{opt}} = 0.34\tau_7,\ \mu_{\text{max}} = 0.99]$
- **N = 14 star graph:** $[p = 0.1, t_{\text{opt}} = 5.5\tau_{14},\ \mu_{\text{max}} = 0.55]$; $[p = 0.5, t_{\text{opt}} = 0.62\tau_{14},\ \mu_{\text{max}} = 0.96]$; $[p = 0.9, t_{\text{opt}} = 0.35\tau_{14},\ \mu_{\text{max}} = 0.99]$

Varying Initial State

- **N = 3 star graph:** $[p = 0.1, t_{\text{opt}} = 2.30\tau_3,\ \mu_{\text{max}} = 0.68]$; $[p = 0.5, t_{\text{opt}} = 0.37\tau_3,\ \mu_{\text{max}} = 0.70]$; $[p = 0.9, t_{\text{opt}} = 0.36\tau_3,\ \mu_{\text{max}} = 0.99]$
- **N = 7 star graph:** $[p = 0.1, t_{\text{opt}} = 2.30\tau_7,\ \mu_{\text{max}} = 0.57]$; $[p = 0.5, t_{\text{opt}} = 0.37\tau_7,\ \mu_{\text{max}} = 0.66]$; $[p = 0.9, t_{\text{opt}} = 0.33\tau_7,\ \mu_{\text{max}} = 0.99]$
- **N = 14 star graph:** $[p = 0.1, t_{\text{opt}} = 6.42\tau_{14},\ \mu_{\text{max}} = 0.58]$; $[p = 0.5, t_{\text{opt}} = 0.39\tau_{14},\ \mu_{\text{max}} = 0.60]$; $[p = 0.9, t_{\text{opt}} = 0.34\tau_{14},\ \mu_{\text{max}} = 0.98]$

These results demonstrate that even when decoherence is introduced through percolation, the average probability of measuring the marked state can still be significantly improved by selecting the optimal walk time t_{opt} for a given percolation parameter and graph size.

Further research is needed to fully understand the effects of decoherence when modeled via bond percolation.

Bibliography

[1] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines," *Journal of Statistical Physics*, vol. 22, pp. 563–591, May 1980.

[2] F. Bova, A. Goldfarb, and R. G. Melko, "Commercial applications of quantum computing," *EPJ Quantum Technology*, vol. 8, p. 2, Jan 2021.

[3] M. Schuld, I. Sinayskiy, and F. Petruccione, "An introduction to quantum machine learning," *Contemporary Physics*, vol. 56, p. 172–185, Oct. 2014.

[4] L. K. Grover, "A fast quantum mechanical algorithm for database search," 1996.

[5] A. Mandviwalla, K. Ohshiro, and B. Ji, "Implementing grover's algorithm on the ibm quantum computers," in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 2531–2537, 2018.

[6] V. L. Ermakov and B. Fung, "Experimental realization of a continuous version of the grover algorithm," *Physical Review A*, vol. 66, no. 4, p. 042310, 2002.

[7] L. M. K. Vandersypen, M. Steffen, M. H. Sherwood, C. S. Yannoni, G. Breyta, and I. L. Chuang, "Implementation of a three-quantum-bit search algorithm," *Applied Physics Letters*, vol. 76, p. 646–648, Jan. 2000.

[8] J. A. Jones, M. Mosca, and R. H. Hansen, "Implementation of a quantum search algorithm on a quantum computer," *Nature*, vol. 393, p. 344–346, May 1998.

[9] P. G. Kwiat, J. R. Mitchell, P. D. D. Schwindt, and A. G. White, "Grover's search algorithm: An optical approach," *Journal of Modern Optics*, vol. 47, p. 257–266, Feb. 2000.

[10] B. Perez-Garcia, R. I. Hernandez-Aranda, A. Forbes, and T. Konrad, "The first iteration of grover's algorithm using classical light with orbital angular momentum," *Journal of Modern Optics*, vol. 65, no. 16, pp. 1942–1948, 2018.

[11] G. L. Long, "Grover algorithm with zero theoretical failure rate," *Physical Review A*, vol. 64, July 2001.

[12] D. Qiu, L. Luo, and L. Xiao, "Distributed grover's algorithm," 2022.

[13] F. M. Toyama, W. van Dijk, and Y. Nogami, "Quantum search with certainty based on modified grover algorithms: optimum choice of parameters," *Quantum Information Processing*, vol. 12, pp. 1897–1914, May 2013.

[14] G. Brassard, P. Høyer, M. Mosca, and A. Tapp, "Quantum amplitude amplification and estimation," 2002.

[15] D. Biron, O. Biham, E. Biham, M. Grassl, and D. A. Lidar, "Generalized grover search algorithm for arbitrary initial amplitude distribution," 1998.

[16] A. Gilliam, M. Pistoia, and C. Gonciulea, "Optimizing quantum search using a generalized version of grover's algorithm," 2020.

[17] S. Marsh and J. B. Wang, "A framework for optimal quantum spatial search using alternating phase-walks," *Quantum Science and Technology*, vol. 6, p. 045029, sep 2021.

[18] X. Qiang, S. Ma, and H. Song, "Quantum walk computing: Theory, implementation, and application," *Intelligent Computing*, vol. 3, Jan. 2024.

[19] R. Portugal, *Quantum walks and search algorithms.* 1 2018.

[20] J. Kempe, "Quantum random walks: An introductory overview," *Contemporary Physics*, vol. 44, p. 307–327, July 2003.

[21] A. M. Childs and J. Goldstone, “Spatial search by quantum walk,” *Phys. Rev. A*, vol. 70, p. 022314, Aug 2004.

[22] D. Qu, S. Marsh, K. Wang, L. Xiao, J. Wang, and P. Xue, “Deterministic search on star graphs via quantum walks,” *Phys. Rev. Lett.*, vol. 128, p. 050501, Feb 2022.

[23] B. Ghanbarian and A. G. Hunt, “Universal scaling of gas diffusion in porous media,” *Water Resources Research*, vol. 50, no. 3, pp. 2242–2256, 2014.

[24] J. Cai, W. Wei, X. Hu, and D. Wood, “Electrical conductivity models in saturated porous media: A review,” *Earth-Science Reviews*, vol. 117, 06 2017.

[25] D. Deb, S. Vishveshwara, and S. Vishveshwara, “Understanding protein structure from a percolation perspective,” *Biophys. J.*, vol. 97, pp. 1787–1794, Sept. 2009.

[26] H. Duminil-Copin, “100 years of the (critical) ising model on the hypercubic lattice,” 2022.

[27] Albinet, G., Searby, G., and Stauffer, D., “Fire propagation in a 2-d random medium,” *J. Phys. France*, vol. 47, no. 1, pp. 1–7, 1986.

[28] H. Deng, J. Du, J. Gao, and Q. Wang, “Network percolation reveals adaptive bridges of the mobility network response to covid-19,” *PLOS ONE*, vol. 16, pp. 1–18, 11 2021.

[29] M. Sahimi, “Applications of percolation theory,” 2023.

[30] X. Meng, J. Gao, and S. Havlin, “Concurrence percolation in quantum networks,” *Phys. Rev. Lett.*, vol. 126, p. 170501, Apr 2021.

[31] H.-M. Kaltenbach, “A concise guide to statistics,” 2012.

[32] M. Sadeghi, F. Behnia, and R. Amiri, “Window selection of the savitzky–golay filters for signal recovery from noisy measurements,” *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 8, pp. 5418–5427, 2020.

A Code for alternate phase walk

Listing (1) Python code for simulating a alternate phase-walk

```
import numpy as np
from scipy.linalg import expm
import matplotlib.pyplot as plt
import random
import math
import pandas as pd



#current_test_file = 'muvstime/timevsmuN3p10.xlsx'


N_nodes = 3  # amount of outer edges
selected_node =  2  #the node that needs to found
perc_prob = 0.9 #chance for percolation in percentages
graph_change_amount = 100 #100 #amount of times the graph changes in a
single CQW
Stat_runs = 200
count_pair = math.ceil(np.pi/4*np.sqrt(N_nodes)-1/2)

period_time = (2*np.pi)/np.sqrt(N_nodes)
tN_p = (2/np.sqrt(N_nodes)*np.arcsin(np.sqrt(N_nodes)*np.sin(np.pi
/(2*(1+2*count_pair)))))
print(tN_p/perio)
#print(period_time)


time_step = tN_p/graph_change_amount


##############################################################

"creating the CQW operator and the flip operator"


# Matrix for flipping the sign for the selected node
Uf = np.diag(np.ones(N_nodes+1))
Uf[selected_node,selected_node] = -1



def adjacency_matrix(p):
    "Creating a string of evolution operators to simulate dynamic/static
    percolation"

    #choosing a string from [0,1] to simulate a broken or intact
    connection to the center
    percolated_graph = random.choices([0,1], weights = [1-p,p], k =
    N_nodes)


    #inverting the string to put in the diagonal
    inverted_list = [1 - x for x in percolated_graph]


    Sn = np.zeros((N_nodes+1,N_nodes+1)) #initialize the matrix
    #check if there is percolation if any in the system
    #and allow the center to hold current value
    Sn[0,0] = percolated_graph.count(0) == N_nodes



    #putting the string in the matrix
    Sn[0,1::]  = percolated_graph
    Sn[1::,0]  = percolated_graph


    #putting the inverted string in the diagonal
    Sn = Sn + np.diag([0]+inverted_list)



    return Sn


def percolated_evolution(tsign):
    "Construction of the evolution matrix for a CQW"
    "Perccount is how many times the system changes within a single CQW"
    "tsign is for the flipping the sign in the exponential operator,
    depending on the iteration"
    Uw  = np.identity(N_nodes+1) # initialize the evolution operator for
    CQM


     # Time step before the next configuration of the graph

    for i in range(graph_change_amount):

        Sn = adjacency_matrix(perc_prob)


        # A partial evolution operator to add to string of evolution
        operators
        Uw_p = expm(-1j*time_step*Sn*(-1)**tsign)
        Uw = np.matmul(Uw_p,Uw) #multiplying the partial to the eventual
        evolution operator

    return Uw

################################
"preparing start vector"
# just the center vector
centervector = np.zeros(N_nodes+1)
centervector[0] = 1

#Same CQW matrix but just for the starting vector
SNstart = adjacency_matrix(1)



#tN_p
start_t = tN_p
UStart = expm(-1j*start_t/2*SNstart)
```

```
#making the s vector as in the paper
startvector = UStart.dot(centervector)

#print(startvector,"starting vector")



# my analytical answer
#checkstartvector = np.empty(N_nodes+1,dtype = np.complex_)
#checkstartvector[1::] =  -np.sin(np.pi/(2*(1+2*count_pair)))*1j
#checkstartvector[0] = np.sqrt(1-N_nodes*np.sin(np.pi/(2*(1+2*count_pair))
)**2)
#print(checkstartvector,"checking")



#####################

"Running the algorithm"

Grover_runs = []

for run in range(0,Stat_runs):
    #initialize the evolution operator for the Grover algorithm.
    U = np.identity(N_nodes+1)

    if (graph_change_amount == 1):
        "static percolation"
        Sn_static = adjacency_matrix(perc_prob)

        for iterat in range(1,count_pair+1):

            perc_U_static = expm(-1j*tN_p*Sn_static*(-1)**iterat)
            Upair_static = np.matmul(perc_U_static,Uf)
            U = np.matmul(Upair_static,U)

    else:
        "dynamic percolation"
        for iterations in range(1,count_pair+1):
            perc_U = percolated_evolution(iterations)
            Upair = np.matmul(perc_U,Uf)
            U = np.matmul(Upair,U)


    #applying the string of matrices to the startingvector
    final_vector = U.dot(startvector)

    #getting the probability distrubtion
    UCchance = final_vector*np.conjugate(final_vector)

    Grover_runs.append(UCchance[selected_node].real)



########################



"checking stats"
```

```
import statistics
mu = statistics.mean(Grover_runs)

std = statistics.stdev(Grover_runs)

#print(f"mu = {mu} and std = {std}")

from scipy import stats




shapiro_test = stats.shapiro(Grover_runs)

print(shapiro_test)


####################

# "Putting it all into a dataframe and in an excel file"


# ppp = pd.read_excel(current_test_file, index_col=0)


# ppp.loc[len(ppp)] = [N_nodes,mu,std,shapiro_test]
# ppp.to_excel(current_test_file)

# print(perc_prob,mu,std,shapiro_test)

##########################
"Printing the results "



# #printing the results
# x = [r'$\left|c\right\rangle$'] + [f"{i}" for i in range(1,N_nodes+1)]
# plt.xlabel("Nodes")
# plt.ylabel("Probability")
# plt.scatter(x,UCchance.real)
# plt.show()


plt.hist(Grover_runs,density=True)
plt.xlabel(r'$P_{\omega}$',fontsize= 16)
#plt.ylabel("Density (Manually Normalized)")


plt.show()


print(mu,'mu')
```

Listing (2) Python code for simulating a alternate phase-walk with varying walk time

```
import numpy as np
from scipy.linalg import expm
import matplotlib.pyplot as plt
import random
import math
import pandas as pd


current_test_file = 'muvstime/timevsmuN14p10.xlsx'

N_nodes = 14  # amount of outer edges
selected_node =  2  #the node that needs to found
perc_prob = 0.1 #chance for percolation in percentages
graph_change_amount = 100 #amount of times the graph changes in a single
CQW
Stat_runs = 200 # the statical runs
count_pair = math.ceil(np.pi/4*np.sqrt(N_nodes)-1/2)
tN_p = (2/np.sqrt(N_nodes)*np.arcsin(np.sqrt(N_nodes)*np.sin(np.pi
/(2*(1+2*count_pair)))))
period_time = (2*np.pi)/np.sqrt(N_nodes)



for time_add in range(1601,2001):
    walk_time = period_time*(time_add/100)
    time_step = walk_time/graph_change_amount

    print(time_add)
    ###############################################################

    "creating the CQW operator and the flip operator"


    # Matrix for flipping the sign for the selected node
    Uf = np.diag(np.ones(N_nodes+1))
    Uf[selected_node,selected_node] = -1



    def adjacency_matrix(p):
        "Creating a string of evolution operators to simulate dynamic/
        static percolation"

        #choosing a string from [0,1] to simulate a broken or intact
        connection to the center
        percolated_graph = random.choices([0,1], weights = [1-p,p], k =
        N_nodes)


        #inverting the string to put in the diagonal
        inverted_list = [1 - x for x in percolated_graph]


        Sn = np.zeros((N_nodes+1,N_nodes+1)) #initialize the matrix



        #check if there is percolation if any in the system
        #and allow the center to hold current value
        Sn[0,0] = percolated_graph.count(0) == N_nodes



        #putting the string in the matrix
        Sn[0,1::]  = percolated_graph
        Sn[1::,0]  = percolated_graph


        #putting the inverted string in the diagonal
        Sn = Sn + np.diag([0]+inverted_list)



        return Sn


    def percolated_evolution(tsign):
        "Construction of the evolution matrix for a CQW"
        "Perccount is how many times the system changes within a single
        CQW"
        "tsign is for the flipping the sign in the exponential operator,
        depending on the iteration"
        Uw  = np.identity(N_nodes+1) # initialize the evolution operator
        for CQM


         # Time step before the next configuration of the graph

        for i in range(graph_change_amount):

            Sn = adjacency_matrix(perc_prob)

            # A partial evolution operator to add to string of evolution
            operators
            Uw_p = expm(-1j*time_step*Sn*(-1)**tsign)
            Uw = np.matmul(Uw_p,Uw) #multiplying the partial to the
            eventual evolution operator

        return Uw

    ############################################################
    "preparing start vector"
    # just the center vector
    centervector = np.zeros(N_nodes+1)
    centervector[0] = 1

    #Same CQW matrix but just for the starting vector
    SNstart = adjacency_matrix(1)

    #tN_p
    start_t =  walk_time #tnp  # ##walk_time#(2/np.sqrt(N_nodes)*np.arcsin
    (np.sqrt(N_nodes)*np.sin(np.pi/(2*(1+2*count_pair)))))
    UStart = expm(-1j*start_t/2*SNstart)




    #making the s vector as in the paper
    startvector = UStart.dot(centervector)
```

```
#print(startvector,"starting vector")


################################################################

"Running the algorithm"

Grover_runs = []
for run in range(0,Stat_runs):
    U = np.identity(N_nodes+1) #initialize the evolution operator for
    the Grover algorithm.

    if (graph_change_amount == 1):
        "static percolation"
        Sn_static = adjacency_matrix(perc_prob)

        for iterat in range(1,count_pair+1):

            perc_U_static = expm(-1j*tN_p*Sn_static*(-1)**iterat)
            Upair_static = np.matmul(perc_U_static,Uf)
            U = np.matmul(Upair_static,U)

    else:
        "dynamic percolation"
        for iterations in range(1,count_pair+1):
            perc_U = percolated_evolution(iterations)
            Upair = np.matmul(perc_U,Uf)
            U = np.matmul(Upair,U)


    #applying the string of matrices to the startingvector
    final_vector = U.dot(startvector)

    #getting the probability distrubtion
    UCchance = final_vector*np.conjugate(final_vector)

    Grover_runs.append(UCchance[selected_node].real)


######################################################
"checking stats"

import statistics
mu = statistics.mean(Grover_runs)
std = statistics.stdev(Grover_runs)

#print(f"mu = {mu} and std = {std}")

from scipy import stats




shapiro_test = stats.shapiro(Grover_runs)

# print(shapiro_test)


#######################################################
```

```
"Putting it all into a dataframe and in an excel file"


perc_file = pd.read_excel(current_test_file, index_col=0)



perc_file.loc[len(perc_file)] = [walk_time,time_step,mu,std,
shapiro_test]
perc_file.to_excel(current_test_file)
```