Quantum random walk search on satisfiability problems

Stephan Hoyer

Swarthmore College
Department of Physics and Astronomy

April 1, 2008

Adviser: David A. Meyer
Department of Mathematics
University of California San Diego

Abstract

Using numerical simulation, we measured the performance of several potential quantum algorithms, based on quantum random walks, to solve Boolean satisfiability (SAT) problems. We develop the fundamentals of quantum computing and the theory of classical algorithms to indicate how these algorithms could be implemented. We also discuss the development of quantum random walks and the principles that go into creating them, presenting existing work on search algorithms using coined discrete-time quantum random walks. Although our quantum algorithms do not solve SAT problems faster than may be done classically, as a step toward that objective, we present the first example of a quantum walk on a directed graph that has a significant increase in speed over the analogous classical walk.

Contents

Chapter 1

Introduction

1.1 Quantum computing

Quantum computing offers a physically based model for computation that may be fundamentally more powerful than the computational model used for computers today. Originally suggested by theoretical physicists Richard Feynman and David Deutsch in the 1980's, quantum computing remained little more than a curiosity until Peter Shor's discovery of a polynomial time integer factorization algorithm in 1994 and Lov Grover's subsequent development of a quantum search algorithm in 1996 [13, 12, 29, 14]. The discovery of these practical algorithms launched a flurry of interest which continues to this day. Fundamental theoretical difficulties with quantum noise and error correction have been resolved, meaning that in principle, these computers can be built and operated efficiently to actually realize their potential [23]. Accordingly, today the race is on to build scalable quantum computers. Many scientists now believe that almost inevitably such machines will be built, however many years that may be down the line.

But even when they are finally built, the hardware for quantum computers will require software, and as yet this software—algorithms to run on quantum computers—remains very limited. Shor's factoring algorithm and Grover's search algorithm remain the foundations of the two main families of useful quantum algorithms. There is the additional likelihood of efficiently simulating real world quantum mechanical systems for chemistry, but these algorithms remain remarkably limited compared to the set of algorithms still most efficiently run on classical computers. Finding new and innovative quantum algorithms is hard, because classical intuitions no longer hold. Regardless, this task is important both to justify the expense and effort that will be required to build such machines, and also because the realm of quantum algorithms still seems to hold untapped potential. One such family of algorithms of recent interest and potential are those based on so-called quantum random walks [5].

1.2 Quantum random walks

The classical random walk is one of the most basic concepts of probability. When formulated with a finite number of states, it consists of randomly choosing paths between vertices on a graph, and many randomized algorithms for computing are such walks. These randomized algorithms are significant because they are often some of the simplest and fastest known ways to solve hard problems [22]. Important applications include solving a variety of graph theory problems, the fastest known tests of primality, and solving boolean satisfiability problems. Similarly, random walks and Brownian motion are at the core of statistical mechanics. Random walks have a clear appeal to both physicists and computer scientists: they are a powerful tool for both describing physical phenomena and writing algorithms.

Not surprisingly then, the quantum random walk can be developed from both physical and computational directions as well. A *quantum walk* is defined by the quantum mechanical movement of a particle on a graph such that if its position were measured continuously, the probability distribution would be a classical random walk. These walks also arise as the simplest versions of quantum cellular automata, and as it turns out, include a discrete version of the Dirac equation of relativistic quantum mechanics [19]. These walks have very different properties from classical walks, as they do not converge to limiting distributions and potentially spread much faster. As with classical walks on classical computers, quantum walks could be run directly and efficiently on quantum computers.

Our focus is on the algorithmic applications of discrete time quantum walks. The frequent faster spreading of quantum walks offers some possibility to improve the performance of randomized algorithms by running them as quantum walks instead, but doing so with most randomized algorithms of interest is difficult as there is no direct translation. We consider several techniques to do so using the problem of boolean satisfiability (SAT), of central interest to theoretical computer science, as a conceptual and quantitative framework to guide our investigations.

1.3 Organization

We begin discussing the classical theory of algorithms as necessary to understand both quantum computing in general and the specific problems we are attempting to solve (Chapter 2). We follow with an introduction to the foundations of quantum computing (Chapter 3), and introduce the notion and power of the quantum random walk (Chapter 4). Specific techniques for speeding up satisfiability problems in the form of quantum walk search are explored in Chapters 5 and 6.

Chapter 2

Classical algorithms

This chapter lays the groundwork in computer science necessary to understand quantum computing and the challenges in creating quantum algorithms. We review logic gates and computational complexity, and then consider in depth the problem of boolean satisfiability for which we hope to find a quantum algorithm.

2.1 Logic gates

There are a number of equivalent theoretical models for classical computation, but the fundamental model used by all modern computers is that of bits and logic gates. The fundamental unit of information is a bit, either 1 or 0. Larger blocks of information are represented by combinations of bits, so bits strung together encode individual letters, documents, pictures, sound and so forth. Thus data is represented by a string of n bits denoted $\{0,1\}^n$.

Each operation on a classical computer then may be considered as mapping $\{0,1\}^n \to \{0,1\}^m$ for some n and m. These operations are called logic gates. Classical computers rely on the principle that any logic gate can be constructed from repetition of a small finite set of logic gates that only act on a few bits, so all computation can be done with binary logic. Furthermore, each of these operation can be built from a small set of fundamental one or two bit operations such as NOT, AND and OR. One such set is given by the first four entries in Table 2.1. These sets are called universal. Note the inclusion of the FANOUT gate. This is often not raised to the level of a logic gate, as in a digital circuit duplicating bits is as simple as attaching another wire. But the same sort of wiring does not work in a quantum circuit, so it is best to make all operations explicit. It is an easy exercise to show that NAND (NOT AND) with appropriate dummy inputs can simulate each of the first three gates, so as it turns out, merely the operations NAND and FANOUT form a universal gate set.

Gate	Input $\to$ Output
AND	$(a,b) \to ab$
XOR	$(a,b) \to a \oplus b$
NOT	$a \to a \oplus 1$
FANOUT	$a \to (a,a)$
NAND	$(a,b) \to ab \oplus 1$

Table 2.1: Classical logic gates defined by their action on $a, b \in \{0,1\}$. $\oplus$ denotes addition modulo 2.

2.2 Computational complexity

Stating the computational resources that an algorithm requires is more sophisticated than it may seem. It is easy to use a stopwatch to see how fast a program runs, but such a metric is not very useful. It is dependent on the particular hardware on which the program is run, and the computers of today vary widely in speed, and are all overwhelmingly faster than those used decades ago. Instead, we could count the number of fundamental operations required, but this measure is still not universally meaningful because the same algorithm may tackle problems of widely varying difficulty and which problems are good benchmarks may change over time. Imagine we have a route-finding algorithm that takes a map and destinations as input and outputs an optimal driving plan. The measure of how long it takes to find a route from Philadelphia to New York might be a useful basis for comparison, but this is not why we wrote the algorithm in the first place. We want to know roughly how long the algorithm takes to find optimal routes based upon the map provided and the destinations. In particular, we are interested in how the time requirements for calculation scale in terms of quantitative measures like the number of junctions and the distance between destinations. Finally, we usually do not care how hard the easy cases are. If the same algorithm will be solving problems at vastly different scales of difficulty at comparable frequencies, it is the hard problems that determine the computational requirements.

Accordingly, computer scientists quantify the scaling of an algorithm by describing how characteristics like the time or memory space required scale as a function of input size n as it grows asymptotically large. A resource requirement given by $f(n)$ is described as $O(g(n))$ if $f(n) \leq c \cdot g(n)$ for all inputs as $n \to \infty$ with some constant c. Likewise it is described as $\Omega(g(n))$ if $f(n) \geq d \cdot g(n)$ for all input as $n \to \infty$ with some constant d, and $\Theta(g(n))$ if both $O(g(n))$ and $\Omega(g(n))$. Accordingly $O(g(n))$ gives an upper bound on the requirement in the worst case of input parameters of size n, whereas $\Omega(g(n))$ gives a lower bound in the best case. This notation gives a useful definition of speed and space requirements for algorithms independent of physical implementations. The difference in speed for an algorithm proceeding at some simple multiple of the speed of another is usually not interesting, because faster computers with identical design scale in performance in the same way. Since again the hardest cases are usually of the most interest, the notation using $O(f(n))$, labeled "big O" notation, is the most common basis for comparison between algorithms.

Problems are divided into computational complexity classes generally by the time requirements of their best possible algorithms using big O notation. The two main complexity classes used to classify the difficulty of problems are P and NP. These classify problems known as decision problems, those with only yes or no answers. This may seem overly restrictive, but nearly any problem can be rewritten as a decision problem. For instance, we could restate our route optimal route finding algorithm as the question of whether or not there is a route from A to B of less than X total miles. P stands for the class of such problems that can be solved in deterministic polynomial time, that is in time $O(\text{poly}(n))$ where $\text{poly}(n)$ is any polynomial of n. NP stands for the class of problems to which proposed solutions can be verified in polynomial time. A proposed solution gives a full description of a potential answer, so we can quickly check the answer to the decision problem. For instance, given a route between A and B, we can quickly check if it is less than X miles. Any problem in P is in the class NP as well, since we can merely run the entire algorithm to test any proposed solution. Problems in NP but not P are usually considered to run in exponential time, even though strictly there exist rarely encountered functions which are asymptotically larger than any polynomial but smaller than any exponential.

More colloquially, P can be considered as the class of "easy" problems whereas NP in general includes the class of hard problems as well. This breakdown makes sense because of the observation usually referred to as Moore's law, that the power of computers grows exponentially over time, which has remarkably held true for the most part since when Moore first stated it over four decades ago. Since $t \to \infty$, $\exp(t) > \text{poly}(t)$ for all exponential and polynomial functions, so eventually a polynomial requirement will always be less than an exponential one. Furthermore, an exponential increase in computational resources means that all problems that can be solved in polynomial time can eventually be done easily for nearly any input size, whereas problems that are exponentially hard will always remain hard.

Another subset of NP, termed NP-complete, is the set of all problems to which a solution can efficiently be used to solve any other problem in NP. Similar to the distinction between P and NP, "efficient" in a technical sense means that a solution to the original problem would only need to be repeated some at most some polynomial (sub-exponential) number of times to create a mapping to the other problem. Since any NP-complete can be efficiently mapped onto other hard problems, in some sense this is class of "hardest" problems.

One of the most significant open questions in computer science and mathematics is whether P is equal to NP. To show P=NP, it would suffice to find one example of an NP-complete problem that is also in P, as this problem could be adapted to every other problem in NP. Such a result would have revolutionary implications for computer science, as then nearly all hard problems could be solved with relative ease. Accordingly, nearly all complexity theorists believe that P≠NP.

The power of quantum computers comes because there are a number of quantum algorithms fundamentally faster than the best known classical algorithms, in the sense of big O asymptotic notation. There are problems that computer scientists believe cannot be solved classically in sub-exponential time, but can be solved in polynomial time on a quantum computer, the most notable example of these problems being integer factorization. The best classical algorithms take exponential time to factor large, but on a quantum computer, using Shor's algorithm, integer factorization can be done in polynomial time [29]. But importantly, integer factorization is not NP-complete. If so, quantum computers could solve nearly any computational problem easily. Quantum information theorists do not expect to find any such problems either, as this would imply in some sense a lack of bounds in the computational power of physical systems [1]. Quantum computers are good, but not that good.

2.3 Satisfiability problems

The problem of boolean satisfiability (SAT) is one of the archetypal NP-complete problems. A satisfiability formula is given by any logic gate or combination of gates that takes n bits to 1 bit. Such a formula is said to be satisfiable if there exists a truth assignment—a set of bit values assigned to each variable—such that the entire statement evaluates to 1. The SAT problem is then the question of whether or not a given satisfiability statement is satisfiable. Clearly this is a decision problem, and since it can be checked in polynomial time if a proposed truth assignment evaluates to 1 or 0, this problem is in NP. It should not be surprising that SAT is NP-complete, since as we already noted logic gates form the fundamental building blocks of modern computers. Thus the statement that SAT is NP-complete is equivalent to noting that any hard problem can be run with roughly equivalent efficiency on the hardware of a modern computer.

As it turns out, there is far more flexibility in SAT formulas than is required for NP-completeness. A useful subset of SAT formulas are those that can be written in k conjunctive normal form (k-CNF). A formula in k-CNF consists of literals defined as one of n variables or their negation, of which groups of k are combined by OR to form clauses, with m clauses combined by AND. As a concrete example, consider the following formula:

$$\underbrace{\overbrace{(a \text{ OR } \neg b \text{ OR } c)}^{k\text{ literals per clause}} \text{ AND } (\neg a \text{ OR } b \text{ OR } d) \text{ AND } (\neg a \text{ OR } a \text{ OR } \neg c)}_{m\text{ clauses with literals chosen from } n \text{ variables } (a,b,c,d,\ldots) \text{ and their negations } (\neg a = \text{NOT } a)} .$$

The problem of k-satisfiability (k-SAT) consists of all SAT formulas in k-CNF. As it turns out, k-SAT is also NP-complete for $k \geq 3$. Since 3-SAT is NP-

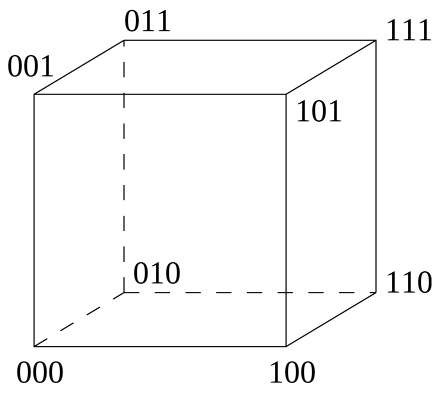


Figure 2.1: The binary hypercube in 3 dimensions. Vertices correspond to each bit array of length n, and edges connect paths that only differ in one bit.

complete and given by such a prescribed form, it is one of the simplest regular forms of SAT that maintains the full complexity of the problem. Also, since numerous other problems have been shown to be NP-complete by noting their equivalence to 3-SAT, many consider it the canonical NP-complete problem. In fact, the term “SAT” often refers directly to 3-SAT.

Remarkably, even though 3-SAT is NP-complete, 2-SAT with its very similar form is in P. This means that 2-SAT is actually an easy problem. We will prove this by presenting a simple random walk algorithm due to Christos Papadimitriou [24]:

```
Guess an initial truth assignment at random.
While the SAT formula is unsatisfied:
  1. Choose an unsatisfied clause at random.
  2. Flip the value of the variable associated with a random literal
     in that clause.
```

Note that a clause is said to be unsatisfied just as for a complete formula, if it evaluates to 0. We will spend a lot of time analyzing this algorithm because it will also be a focus of our interest in random walk search procedures.

Before we present a proof of the speed of this algorithm, there are a few things worth observing about it. First, note that we can represent any truth assignment as a position on the binary hypercube in n-dimensions, the n-dimensional space where there are exactly two possible values $\{0, 1\}$ in each dimension, as was perhaps alluded to by referring to a string of bits as $\{0, 1\}^n$. A binary hypercube is shown is in Fig. 2.1.

But more specifically to this problem, note that the algorithm given here for 2-SAT is a random walk algorithm that proceeds along the edges of this binary hypercube. In particular, the algorithm proceeds by picking a random starting vertex, and then flipping the value of one variable at a time. In fact, this 2-SAT algorithm can be considered as exactly a random walk proceeding along a directed graph where edges are taken from the set of those appearing

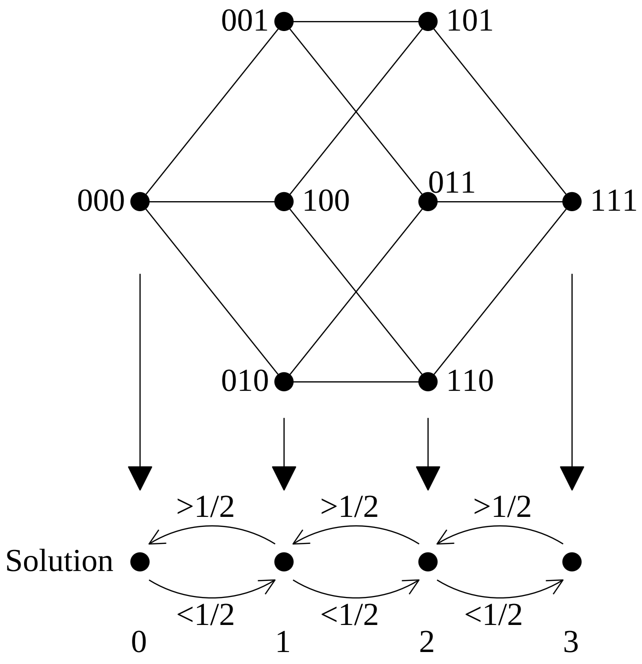


Figure 2.2: Reduction of hypercube walk to the line. Vertices are grouped by their Hamming distance from a solution, here chosen arbitrarily at 000.

on the binary hypercube, with possibly redundant paths in the case that a variable appears in multiple unsatisfied clauses.

Theorem. *If its formula is satisfiable, the 2-SAT algorithm given above terminates in $O(n^2)$ steps.*

Proof. In the worst case scenario that this formula is satisfiable, there is exactly one satisfying truth assignment, and the initial guess is as far away from that solution as possible. We can group truth assignments by how many bits they have that are different from the solution. This measure of the number of different bits is known as the Hamming distance. This grouping by Hamming distance from $x_0 = 000$ is shown in Fig. 2.2. Since each possible move along the hypercube is a change of the value of one bit, each either increments or decrements the Hamming distance from the solution, so we can consider this algorithm as a random walk along the line given by this set of distance values as well. If there are n variables, then this walk is along the line $[0, n]$.

At each position along the line, there is at least a $1/2$ probability of moving closer to the solution, because in each unsatisfied clause, there are at most two variables, at least one of which must have a different value from its value in the solution. The walk only terminates when it reaches the end of the line at distance $x = 0$ from the solution. At the other end, it reflects from the position of maximum distance $x = n$ to the position $x = n - 1$ with certainty.

We can write down a recurrence relation for the number of expected steps E_x at position x until the walk terminates in terms of the number of expected steps left at adjacent positions as

$$E_x = \frac{E_{x-1}}{2} + \frac{E_{x+1}}{2} + 1, \quad \text{for } 0 \leq x < n \tag{2.1}$$

as in the worst case there is at most a 1/2 chance of moving to a position at greater distance from the solution at the next step, at which point one more step has been made. We also have the constraints

$$E_0 = 0 \tag{2.2a}$$
$$E_n = E_{n-1} + 1 \tag{2.2b}$$

as at $x = 0$ the walk terminates and there is only one possible move at $x = n$. Since the recurrence relation involves two previous terms, clearly these two initial conditions completely specify E_x. Without delving into the theory of recurrence relations, we can guess the unique solution

$$E_x = n^2 - (n - x)^2, \tag{2.3}$$

which can be easily verified to satisfy Eqs. (2.1) and (2.2). In the worst case, we start position $x = n$, so algorithm is expected to terminate in n^2 steps. By repeating this algorithm some constant number of times, we can make the probability of failure in the case that a satisfying solution exists arbitrarily small. Then if after $O(n^2)$ steps the algorithm does not terminate, with arbitrary certainty the formula is unsatisfiable. □

The polynomial speed of this algorithm relies on the fact that there is always at least probability 1/2 of moving closer to the solution at the next subsequent step so that Eq. (2.1) holds. If the constraint is any lower bound than 1/2, then the algorithm turns out to run in exponential time. Accordingly, the algorithm is slower for 3-SAT because the minimum probability of success is 1/3 if only one variable in a clause has the wrong value. The run time of this algorithm is thus very precariously dependent on the particular structure of the graph it performs a random walk on.

We call the formula $f(x)$ that evaluates the satisfiability statement for a given truth assignment an Oracle function because it tells us when we have reached a solution to our "search" problem over all possible arrays of binary values. This algorithm only requires checking the Oracle $O(n^2)$ times. This makes it notably more efficient than brute force search, which would require evaluating the Oracle at each position on the binary hypercube, 2^n positions in all, to check each possible solution. We have gained a substantial improvement in speed by performing search on a graph instead of the unstructured problem. Faster algorithms for 2-SAT exist than solve the problem in as little as $O(n)$ steps, but this algorithm is remarkably simple and can be expressed as a random walk.

The primary reason why this 2-SAT algorithm is of significance is because it is similar to a very fast algorithm for 3-SAT. Since 2-SAT is such an easy problem, it is generally only of interest because of its connection to 3-SAT. 3-SAT is a problem of substantial interest to computer scientists for which there have been ongoing efforts to find fast algorithms. As for 2-SAT, the naive brute-force search of each possible solution runs in 2^n steps, but better algorithms exist. One of the very best to date is due to Uwe Schöning [26]. At its core is the same procedure as the 2-SAT algorithm by Papadimitriou, however the loop is only repeated $3n$ times before the algorithm is run again in its entirety with a new guess of an initial truth assignment. After some number of repetitions taking no more than $O((4/3)^n)$ steps in total, the algorithm will terminate successfully with arbitrarily high constant probability if a satisfying truth assignment exists. Note that the fact that the algorithm repeats is not actually any different from the 2-SAT algorithm, as both only terminate successfully after some number of steps with some probability. Reaching arbitrary certainty in either case would require repetition, although in the case of 2-SAT it would suffice to merely run continue the same run of the algorithm for longer as well. Although Schöning's algorithm is no longer remains the very fastest for 3-SAT, it remains among the best, and many superior algorithms are still based off of it.

2.4 Evaluating k-SAT algorithms

Evaluating new algorithms for 2-SAT and 3-SAT by numerical simulation instead of analytical techniques requires measuring speed on some set of formulas instead of establishing some abstract upper bound. But k-SAT problems can vary widely in difficulty, and it is easy to cook up formulas that are either satisfiable or unsatisfiable for all truth assignments. The tricky cases are those that *a priori* cannot be categorized as likely satisfiable or unsatisfiable. This should correspond to a set of formulas for which on average it takes a long time to find a solution, giving some rough estimate of the upper bound.

The simplest way to find k-SAT formulas to test other than taking them directly from real world problems would be to generate them uniformly at random with n variables and m clauses. If we are interested in measuring the performance of a k-SAT algorithm at different values of n, then for given n and k, we need to somehow pick the number of clauses m. For randomly generated formulas, it is clearly far more likely for there to be multiple solutions if m is small since there is a smaller set of constraints, and as m increases randomly generated formulas should be less and less likely to have any solutions at all. This transition between likely satisfiable and likely unsatisfiable formulas can be considered as sort of numerical phase transition occurring at some critical ratio $\alpha = n/m$ of the number of variables to the number of clauses, and there is an actually an extensive literature examining these sorts of transitions from the perspective of statistical mechanics [9]. Randomly generated formulas at this phase transition are neither more likely to be satisfiable or not, so these are the hardest cases. Since some of the best 3-SAT algorithms run slowest

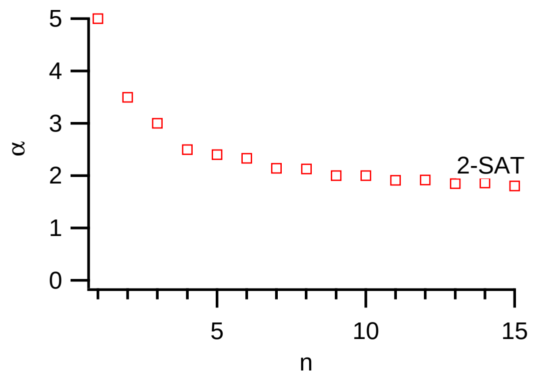


Figure 2.3: Location of phase transition for 2-SAT problems at small n in terms of the parameter $\alpha = m/n$. As $n \to \infty$, $\alpha = 1$ for 2-SAT and $\alpha \approx 4.25$ for 3-SAT.

on randomly generated formulas at the transition, it makes sense to use this transition when evaluating the performance of algorithms numerically [15]. These algorithmic phase transitions have been studied and classified in the same sorts of ways as with physical transitions. For example, 3-SAT has a first-order (discontinuous) transition in terms of the number of clauses m, whereas the transition with 2-SAT is second-order (continuous). It turns out that in the limit as $n \to \infty$, for 2-SAT this transition is at $\alpha = 1$ and for 3-SAT at about $\alpha = 4.25$, but the transition moves to this limit relatively slowly. Accordingly, for use in later numerical trials, we give the location of the phase transition for 2-SAT in Fig. 2.3.

Another way to find hard cases is the obvious measure of counting the number of solutions. Then k-SAT formulas with only one solution should be the hardest to solve. This is most dramatically true for a random brute-force approach, although in practical instances, we can imagine that multiple solutions may be clustered structurally, so multiple solutions could be nearly as hard to find as one solution for a classical random walk or systematic search. There are two obvious ways to generate such statements randomly. One option is add clauses one by one until a only one solution remains or the formula is unsatisfiable, in which case we backtrack one step. The alternative is to randomly generate formulas at the phase transition and discard them unless they have one solution. The distinction is not very important, and accordingly we will choose the second method, if only because it seems more analogous to the more general technique.

Chapter 3

Quantum computing

3.1 Physical motivations

The potential of quantum computing is alluded to by the difficulty of simulating quantum systems on a classical computer. For example, consider a system of 100 particles each with two possible states. If this system were classical, we could write down its state just with 100 zeros or ones. But if this system were quantum mechanically, recording its state would involve writing down 2^{100} complex numbers, more numbers to keep track of than the number of atoms in the visible universe, estimated at roughly 10^{80}. Accordingly there is clearly no way that a classical computer could calculate exactly any sort of non-trivial evolution of such a system.

This suggests a natural question leading to quantum computing, originally by Richard Feynman, of whether such hard to simulate systems could be harnessed directly to solve other hard problems of interest [13]. One obvious and very important application would be simulating quantum systems themselves, already of central interest to many chemists and biophysicists. But in a broader sense, quantum simulation alone is a very limited of application. More interesting is the question of whether more general computational needs, as we use classical computers for, can be better met by quantum devices.

In particular, in this chapter we present a general model of computing with quantum systems that is believed to give as much computational power as possible with quantum systems. The answer to the question of whether quantum computers are fundamentally more powerful than classical machines seems to be a resounding yes. Some such algorithms for quantum search will be identified in Chapter 5, although the algorithms providing the most compelling evidence for this answer are not the focus of this thesis. For further details about other quantum algorithms and implementing quantum computing, a good reference is the introduction to quantum computing and information by Nielsen and Chuang [23].

3.2 Quantum mechanics on qubits

Quantum computing builds from very basic concepts in quantum mechanics. The gate model of quantum computing that we will construct here relies only on a small subset of these laws. Here we will review the foundations, although portions of this thesis may not be accessible to those without further familiarity with quantum mechanics.

The evolution of quantum states is deterministic, but the process of measuring a quantum system is stochastic. The state of a quantum system is given by a vector written $|\psi\rangle$ in a complex vector space. Each dimension of the vector space corresponds to possible result from measuring the system, such as the location of a particle. If ψ_i indicates the projection of $|\psi\rangle$ onto the ith dimension, then measurement of the ith result has probability $|\psi_i|^2$. "Measurement" is an ambiguous concept, corresponding to a result indicated by a detector in the lab. It is the only way that classical information may be extracted from quantum systems: there is no way to determine the exact state of such a system. Since a detector always registers a value, these probabilities must sum to one. This corresponds to the requirement that the state $|\psi\rangle$ must have length $\langle\psi|\psi\rangle = 1$, as $\langle\psi|$ indicates the conjugate transpose $|\psi\rangle^\dagger$. Since quantum mechanics is linear, transitions between quantum states are given by length preserving transformations. These correspond to left multiplication by a unitary matrices U, that is those such that $UU^\dagger = U^\dagger U = I$.

Quantum computing uses systems built from the natural quantum analog of the bit, the *qubit*. A qubit is any two dimensional quantum system with orthonormal states labeled $|0\rangle$ and $|1\rangle$. Accordingly it can be measured in either of the classical states 0 or 1. But qubits can also be in any *superposition* of $|0\rangle$ and $|1\rangle$, in the state $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$ with $|a|^2$ and $|b|^2$ corresponding to the probabilities of measuring 0 and 1. Larger systems are built from systems that could describe multiple qubits, with basis states given by the tensor product of the basis vectors for the original qubits. If two qubits are specified by independent states $|\psi_1\rangle$ and $|\psi_2\rangle$, then a combined system is given by the tensor product $|\psi_1\rangle \otimes |\psi_2\rangle$. Accordingly systems formed by two qubits are a four dimensional with basis states $|0\rangle\otimes|0\rangle$, $|0\rangle\otimes|1\rangle$, $|1\rangle\otimes|0\rangle$ and $|1\rangle\otimes|1\rangle$. These states are generally referred to with the tensor product symbol omitted, as $|00\rangle$ and so forth, corresponding to a result from measurement. Thus a quantum system upon measurement gives a result with the same number of classical bits as there are qubits. More generally, larger systems are built in the analogous fashion with simply more qubits.

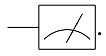
Quantum algorithms consist of these qubits and unitary operations, called *quantum gates*, that are applied to them. As the first step of an algorithm, the qubits are initialized in some uniform starting state, usually in the state $|00\ldots0\rangle$. The qubits are processed through quantum gates, and at the end of any algorithm, the state of the system is measured so that a classical result may be obtained. There is no particular quantum system such as an atom or photon that a qubit represents—rather, as with classical bits, a qubit is an abstract building block that could refer to any quantum system. As with classical computers, we can abstract away the details of how these machines are built. There are many proposals for implementations that are for the most part theoretically equivalent.

Because quantum mechanical systems are linear, we can put the initial state of our system in any superposition of input states. So in a single evaluation of a unitary operation on the superposition of all possible input states, we can obtain a quantum mechanical state that is some combination of results that classically could only be obtained from evaluating each possible result separately. Accordingly it might seem that quantum computers can run every possible input at the same time. This is a frequent misconception, but quantum states cannot be known absolutely as they can only be obtained by the physical process of measurement. Thus this sort of naive parallelism gives no more information on average in one evaluation than by choosing the input to an equivalent classical circuit at random. However, this does allude to the where the power of quantum computing arises. Sometimes the clever use of these superpositions allows information to be obtained at substantially faster speeds than is known or possible classically.

3.3 Quantum gates

As with classical computing, it is generally not helpful to consider quantum algorithms as a single unitary transformation. Accordingly, as with classical operations and logic gates, we generally break down unitary operations into a series of sub-operations known as quantum gates.

For clarity, such operations are often written in a form known as quantum circuit notation. Quantum circuit notation is used to represent the action of a quantum system on a set of qubits, like a classical circuit acts on bits. Qubits are indicated by lines and multiple qubits may be indicated by slashes through those lines. Operations are indicated by labeled boxes intersecting qubits. As is written for classical circuits, time always proceeds from left to right. Also, as the final step in a quantum algorithm, measurement is often indicated explicitly by the icon



The operation U applied to a qubit $|\psi\rangle$ is written in quantum circuit notation as

$$|\psi\rangle \;-\!\boxed{U}\!-.$$

Overall, this framework is especially useful because it keeps a physical implementation of quantum devices in mind.

Since quantum transitions are given by matrices, we can define any operation by considering its action on a set of basis vectors. So to construct a quantum NOT gate X, we want a qubit starting in with a definite value of 0 or 1 to be in the opposite state after applying X. Thus within an unmeasurable phase factor $e^{i\phi}$, we require $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. If we make the associations

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

then X must be given by

$$X = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}, \tag{3.1}$$

the Pauli σ_x matrix. But unlike the classical case, this gate is far more powerful as it can act on any linear combination $a|0\rangle + b|1\rangle$.

Of course, quantum gates can do more than merely copy classical operations—any unitary matrix can serve as a quantum gate. Another standard one-qubit operation is the Hadamard transform

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}. \tag{3.2}$$

The Hadamard transform takes the state $|0\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, and so is useful for creating a superposition of input states.

In the standard convention, the tensor product of definite single qubit states $|i_1\rangle, \ldots, |i_n\rangle$ where each i_k is 0 or 1, is written as $|i_1\rangle \otimes |i_2\rangle \otimes \ldots \otimes |i_n\rangle = |i_1 i_2 \ldots i_n\rangle$. Such states are ordered in this "computational basis" in ascending numerical order with $i_1 i_2 \ldots i_n$ taken as the binary representation of a number, which is often written in base ten. Each state can be identified directly with the number for which it is the binary representation. This computational basis provides a natural ordering and basis for writing unitary transformation of these combined states. Arbitrary spaces of any finite dimension m can be constructed merely by requiring that the subspace with $i_1 i_2 \ldots i_n \geq m$ starts with zero probability amplitude. We can now see that the Hadamard transform acting on each qubit individually, denoted $H^{\otimes n}$, takes the state $|0\rangle$ to a superposition of all possible states:

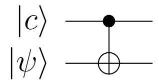
$$\begin{aligned}
H^{\otimes n}|0\rangle &= H|0\rangle \otimes \ldots \otimes H|0\rangle && (3.3)\\
&= \frac{1}{\sqrt{2^n}}(|0\rangle + |1\rangle) \otimes \ldots \otimes (|0\rangle + |1\rangle) && (3.4)\\
&= \frac{1}{\sqrt{2^n}}(|0\ldots 0\rangle \otimes \ldots \otimes |1\ldots 1\rangle) && (3.5)\\
&= \frac{1}{\sqrt{2^n}}(|0\rangle + |1\rangle + |2\rangle + \ldots + |2^n - 1\rangle). && (3.6)
\end{aligned}$$

This feature makes the Hadamard transform a common initialization step in quantum algorithms.

Another gate of central importance in quantum computing is the controlled not gate C-NOT. Acting on two qubits, it is given in matrix form by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.7}$$

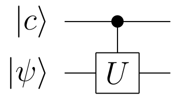
Its action is to NOT the value of the second bit if the first bit is 1. In quantum circuit notation, it is written



where $|\psi\rangle$ is the controlled qubit and $|c\rangle$ is the controlling qubit. As it turns out, the C-NOT gate and the set of single qubit unitary operations forms a universal gate set for quantum computing: any unitary transformation can be created by their combination [23]. More generally, it can be useful to consider the controlled-U gate written in block form as

$$\begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix}, \tag{3.8}$$

where if the first bit is 1, an arbitrary unitary operation U is applied to remaining qubits. In quantum circuit form, the controlled-U gate is written as



where $|c\rangle$ is the control qubit acting on the qubits $|\psi\rangle$. Along these lines, the controlled not gate is also the controlled-X gate. Also, instead of the symbol $\bullet$, the symbol $\circ$ can be used to indicate a controlled not changing the second bit if the first bit is in the state $|0\rangle$.

The process of building an arbitrary quantum circuit is done starting from a basic set of gates that allow for any quantum calculations with arbitrarily small bounded error. Such a complete gate set is given by the Hadamard gate, the C-NOT, and two other single-qubit gates, labeled the phase gate and $\pi/8$ gate. Thus hardware implementations of quantum computing only need to use a small set of reproducible gates. Not all unitary operations can necessarily be simulated *efficiently* by a particular gate set, but schemes can be identified that suffice for arbitrary quantum algorithms.

3.4 Reversible computation

Quantum mechanical transitions are unitary. But logic gates from classical computing such as AND are not reversible, never mind something that could be made unitary. How could such operations be made quantum mechanical?

Input			Output		
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Table 3.1: Truth table for the Toffoli gate.

The answer lies in reversible computing, as every logic gate in classical computing can be made reversible, and then be extended such that they may be implemented on a quantum computer.

Without loss of generality we can make any irreversible operator reversible by saving the results of the operation in a different bit. For instance, the operation AND: $(a, b) \to ab$, can be made reversible by taking $(a, b, 0) \to (a, b, ab)$. We can extend this principle to create a gate known as the *Toffoli gate* which provides a reversible universal basis for computation. This gate is defined by its action on three bits

$$(a, b, c) \to (a, b, c \oplus ab),$$

where $\oplus$ denotes addition modulo 2. Consider applying the Toffoli gate twice: $(a, b, c) \to (a, b, c \oplus ab) \to (a, b, c \oplus ab \oplus ab) = (a, b, c)$. Thus the Toffoli gate is its own inverse, so it is reversible. The truth table for this gate is shown in Table 3.1. Since a NAND $b = 1 \oplus ab$ is given by the third bit in a Toffoli gate for $c = 1$, and a Toffoli gate can also do the operation FANOUT for $b = 1$ and $c = 0$, the Toffoli gate can simulate a universal gate set, and thus is a universal gate itself.

As it takes a constant number of Toffoli gates to simulate any other fundamental logic gates, reversible computation is as fast as irreversible computation within a constant factor, a slowdown irrelevant in terms of computational complexity. As the Toffoli gate is reversible, it can be written out in full form as a $2^3 \times 2^3$ unitary matrix providing definite transitions between the states $|abc\rangle$. This matrix can be directly constructed from looking at Table 3.1: it is the "doubly controlled-not" gate, where the third bit is flipped only if both of the first two bits are 1. More generally, it is clear that we can make any reversible classical operation taking bit values $j \to k$ quantum by mapping the basis vectors $|j\rangle \to |k\rangle$. Then as the Toffoli gate is a quantum gate, all reversible classical computation and thus all classical computation may be run on a quantum computer.

This detail is important for several reasons. First of all, it shows that classical computing is a strict subset of quantum computing, so a quantum computer would be sufficient for all computational needs. Further, even though all classical computation could just be run on a classical computer directly, the theoretical ability to mock up quantum versions of an any classical circuit is important when considering minor adaptations of otherwise classical algorithms. This is important for the implementation of quantum walks as discussed in the following chapter.

Chapter 4

Quantum random walks

4.1 Markov chains and random walks

Before considering quantum random walks, it is worthwhile to recall the formalism of classical random walks, embodied in the notion of a Markov chain. Consider any system undergoing random transitions between a discrete set of possibilities. Let a column vector p with each term positive and summing to one be a probability distribution over these states. For each possible state, there is corresponding column vector that gives the probability of transition to each other state. Accordingly transitions to another state via a random process can be described by a Markov transition matrix M where the ijth entry is given by

$$M_{ij} = \Pr(i|j), \tag{4.1}$$

and the state after the random process is given by[1]

$$p' = Mp. \tag{4.2}$$

Repeated applications of the random process can be determined by repeated left multiplication by M.

For example, for the random walk on a line with equal probability of moving left and right, the transition matrix is given by the band diagonal matrix

$$M = \begin{pmatrix} \ddots & & & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & & \\ & \frac{1}{2} & 0 & \frac{1}{2} & & \\ & & \frac{1}{2} & 0 & \frac{1}{2} & \\ & & & & & \ddots \end{pmatrix}. \tag{4.3}$$

[1]The standard convention of mathematicians for Markov chains uses right multiplication, but instead we use an equivalent formulation with left multiplication to emphasize their connection with changes of state in quantum mechanics.

Thus a particle at position n goes to positions $n-1$ and $n+1$ each with probability $\frac{1}{2}$.

In general, Markov chains greatly aide the analysis of complicated random systems. For instance, stationary distributions can be identified immediately as the eigenvectors of M with eigenvalue 1. They are a widely used tool in probability theory, but for our purposes it will be sufficient to note their existence and form for comparison to quantum walks.

4.2 Defining a quantum walk

Equation (4.2) looks remarkably similar in form to the quantum mechanical evolution of a state $|\psi\rangle$ under the unitary operator U,

$$|\psi'\rangle = U\,|\psi\rangle\,, \tag{4.4}$$

with the new normalization condition $\langle\psi|\psi\rangle = 1$. Evolution after each additional time steps proceeds again by left multiplication by U. But if projective measurement onto basis state is performed after each application of U, the unitary transition U becomes a stochastic process that can be described by a Markov chain, with a transition matrix given by entries

$$M_{ij} = U^*_{ij}U_{ij} = |U_{ij}|^2, \tag{4.5}$$

corresponding to the ijth entry U_{ij} of U. This "classical limit" is the natural way in which a quantum operation suggests a random walk.

We call such an quantum system when thought of in the context of the random walk that is its classical limit a *quantum random walk*. Originally considered in continuous time and space in 1993 [4], quantum random walks in discrete space have been the focus of subsequent work because they amenable to use in quantum algorithms. Although our focus is on quantum walks discrete in both time and space, there is significant literature concerning quantum walks that evolve in continuous time. They will not be discussed here, but they have similarly powerful algorithmic applications, including exponential to polynomial speedup for some contrived problems [10]. Thus in some sense—not very usefully—any quantum system can be considered as such a walk, so in practice systems are labeled quantum walks to emphasize their connection with classical walks.

The difficulty with this definition is that a quantum mechanical operation induces a random walk, not the other way around. As the standard challenge in quantum computing is to speed up classical algorithms, we would like to be able to directly find quantum analogs of classical walks. Any sort of random walk with discrete positions marked on a graph is a natural candidate. The focus of research in this subfield has been to discover quantum versions of classical walks of interest and examine the distinctions between quantum and classical versions, all with an eye toward "speeding up" algorithms based on such walks.

We will introduce quantum walks and their uses in the remainder of this chapter. For further details and more applications, we recommend that the interested reader consult review articles by Ambainis [5] and Kempe [16].

4.3 Quantum walks on a line

The simplest classical random walk is a particle on a discrete line in one dimension. At each step, a coin is flipped and the particle moves left or right with equal probability. The transition matrix for this walk was given previously in Equation (4.3), and the probability distribution from this walk is well known to proceed as a binomial distribution. The probability of finding the particle at position n at time T $p(n,T)$ is zero if $n+T$ is odd and otherwise given by

$$p(n,T) = \frac{1}{2^T}\binom{T}{\frac{1}{2}(n+T)}, \quad \text{if } n+T \text{ even} \tag{4.6}$$

where the deviation from the direct binomial distribution $\binom{T}{n}$ arises because the walking particle moves left or right dependent on a coin flip, instead of merely moving on each successful flip. In any case the distribution for this walk is identical to the binomial distribution centered at the origin adding zeros at alternate positions. Over time, the average position $\langle n\rangle = 0$ and the variance, the average distance squared to the mean, is given by $\sigma^2 = \langle(n-\langle n\rangle)^2\rangle = \langle n^2\rangle = T$, as can be show by straightforward algebra.

Since so much of the representative power of the classical random walk is found on the line, the line is a natural first candidate for finding a quantum random walk as well. Following the definition given in the proceeding section, we would like to find a quantum walk on the line with a random walk as its classical limit.

The most obvious way to define a quantum walk on a line is to choose as a set of basis states $|n\rangle$, $n \in \mathbb{Z}$ representing each integer. Then we would like to find a unitary operator U that acts as a classical walk under measurement. The operation should act identically on each position state $|n\rangle$ by altering the position with some probability. Defining its action on each basis vector $|n\rangle$, then the walk must be given by some U such that

$$U|n\rangle = a|n-1\rangle + b|n\rangle + c|n+1\rangle\,, \tag{4.7}$$

where $a, b, c \in \mathbb{C}$. However, defining a non-trivial quantum walk in this manner is not possible, as the following theorem shows.

Theorem. *If this operation U as defined above is unitary, then exactly one of the following holds (a simple case following the general proof given by Meyer [19]):*

1. $|a|^2 = 1$, $b = c = 0$

2. $|b|^2 = 1,\ a = c = 0$

3. $|c|^2 = 1,\ a = b = 0$

Proof. Written in matrix form, we have

$$U = \begin{pmatrix} \ddots & a & & & \\ & b & a & & \\ & c & b & a & \\ & & c & b & \\ & & & c & \ddots \end{pmatrix}.$$

Since the columns of U form an orthonormal set, we have

$$\begin{aligned} c^* a &= 0 \\ a^* b + b^* c &= 0 \\ |a|^2 + |b|^2 + |c|^2 &= 1. \end{aligned}$$

From the first two equations, at least two of a, b and c must equal zero, and by the last equation one of the cases 1-3 is fulfilled. □

Accordingly if a walk is defined as in Equation (4.7) at each step a particle makes a definite step left or right, or stays in the same position. These are not interesting cases, but fortunately, non-trivial quantum walks can be easily defined using only slightly more complicated operations. There are multiple ways in which this could be done; here we present only the most straightforward and extensible method, known as the "coined" quantum walk.

The walk is performed on the Hilbert space $\mathcal{H}_s \otimes \mathcal{H}_c$, the tensor product of Hilbert space $\mathcal{H}_s$ of discrete position states $|n\rangle$ with another 2-dimensional space $\mathcal{H}_c$ labeled the coin space. We label a set of orthonormal basis vectors in $\mathcal{H}_c$ as $|-\rangle$ and $|+\rangle$. Thus the coin space is equivalent to the particle having spin-$\frac{1}{2}$. Two unitary operations are then defined which are performed in succession at each time step.

1. The "coin flip" operator C acts on $\mathcal{H}_c$:

$$C\,|-\rangle = a\,|-\rangle + b\,|+\rangle \tag{4.8a}$$
$$C\,|+\rangle = c\,|-\rangle + d\,|+\rangle \tag{4.8b}$$

2. The "shift" operator S applies a conditional shift to the states $|n\rangle$ dependent on the coin state:

$$S\,|n,-\rangle = |n-1,-\rangle \tag{4.9a}$$
$$S\,|n,+\rangle = |n+1,+\rangle \tag{4.9b}$$

The evolution of the walk in one time step is thus given by $U = S \cdot (I \otimes C)$. The operator $I \otimes C$ indicates the identity operation on the position subspace and the operation C on the coin subspace, and is unitary for any unitary $C = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$. The operator S is also clearly unitary since it is norm preserving as it merely exchanges the amplitudes associated with basis vectors. Accordingly the evolution U is unitary as well.

The coin flip operator C may be specified by any unitary matrix. Standard choices include the Hadamard transformation

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4.10}$$

and the "balanced" coin

$$Y = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}. \tag{4.11}$$

Both these operators exactly reproduce the classical random walk in the classical limit when projective measurement is applied after each set to collapse the state $|\psi\rangle$ to a state with definite number $|n\rangle$, since any state specified by $|n\rangle$ transitions to either $|n-1\rangle$ or $|n+1\rangle$ with equal probability. Now consider the evolution of an initial state $|\psi\rangle = |0,+\rangle$ using the Hadamard transformation H as the coin operator:

$$|\psi\rangle \to \frac{1}{\sqrt{2}} |-1,-\rangle - \frac{1}{\sqrt{2}} |1,+\rangle \tag{4.12a}$$

$$\to \frac{1}{2} |-2,-\rangle - \frac{1}{2} |0,-\rangle + \frac{1}{2} |0,+\rangle + \frac{1}{2} |2,+\rangle \tag{4.12b}$$

$$\to \frac{1}{2\sqrt{2}} |-3,-\rangle + \frac{1}{2\sqrt{2}} |-1,+\rangle + \frac{1}{\sqrt{2}} |-1,-\rangle - \frac{1}{2\sqrt{2}} |1,-\rangle + \frac{1}{2\sqrt{2}} |3,+\rangle \tag{4.12c}$$

After the first two evolutions, the probability of measuring the particle at any position remains the same as for a classical random walk. But on the third application of U, positive moving amplitudes at $n = 1$ interfere destructively simultaneously with constructive interference at $n = -1$. There is a probability $5/8$ of the particle being measured in the state $|-1\rangle$ but only a probability $1/8$ at $|1\rangle$. Quantum effects have entered our random walk, and all future evolution of the walk will show even more interference effects.

Graphs of the probability distribution of these walks initialized in the state $|\psi\rangle = |0,-\rangle$ as measured after T steps are shown in Figure 4.1. The Hadamard matrix is real, but does not treat $|-\rangle$ and $|+\rangle$ state symmetrically and thus leads to an asymmetric walk as shown when started in a state initially moving strictly in one direction. The initial state $|\psi\rangle = |0,+\rangle$ leads to the same probability distribution reflected over the vertical axis, so the initial state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0,-\rangle + i\,|0,+\rangle)$ with the coin H leads to the symmetrical probability distribution shown in Figure 4.2, since H has all real values, so the imaginary

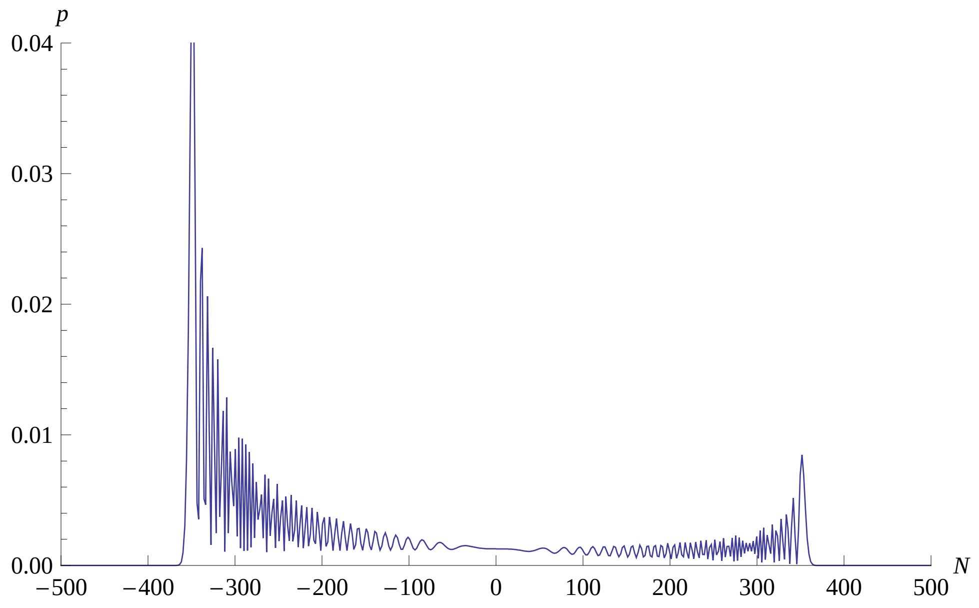


Figure 4.1: Probability of position measurement for the unbalanced 1D quantum walk after $T = 500$ time steps. Only the probabilities for even N is shown as $p = 0$ for all odd N. Generated by the coin operator H with initial state $|\psi\rangle = |0, -\rangle$. Initial state $|\psi\rangle = |0, +\rangle$ generates the same graph reflected over the vertical axis.

probability amplitude propagates independently from the real portion. The "balanced" coin Y is so called because it leads to the exact same symmetrical distribution even when starting in an unbalanced state such as $|0, -\rangle$.

These walks appear very different from their classical analogs. They do not converge to a Gaussian or any other static distribution as $t \to \infty$ and have frequent amplitude oscillations due to self-interference. It should not be surprising that the central limit theorem does not hold for quantum walks, because unlike Markov chains, they are always reversible. In particular, it is apparent and can be shown analytically [7] that the value is approximately constant over the interval $[-\frac{T}{\sqrt{2}}, \frac{T}{\sqrt{2}}]$. Thus, to calculate the variance σ^2, as again $\langle n \rangle = 0$,

$$\sigma^2 = \langle n^2 \rangle = \frac{1}{2T/\sqrt{2}} \int_{-T/\sqrt{2}}^{T/\sqrt{2}} x^2 dx = \frac{T^2}{6}. \tag{4.13}$$

Thus while the expected distance from the origin σ for the classical walk is $\Theta(\sqrt{T})$, for the quantum walk it is $\Theta(T)$. This is a quadratic improvement in speed and is our first example of how a quantum walk propagates faster than its classical analog.

The analytical proof for the only approximate probability distribution of this walk is quite involved, but the result is easily evident from numerical simulations. Such simulations are easy to setup and evaluate. This is a recurring theme in the analysis of quantum walks and suggests numerical simulation as

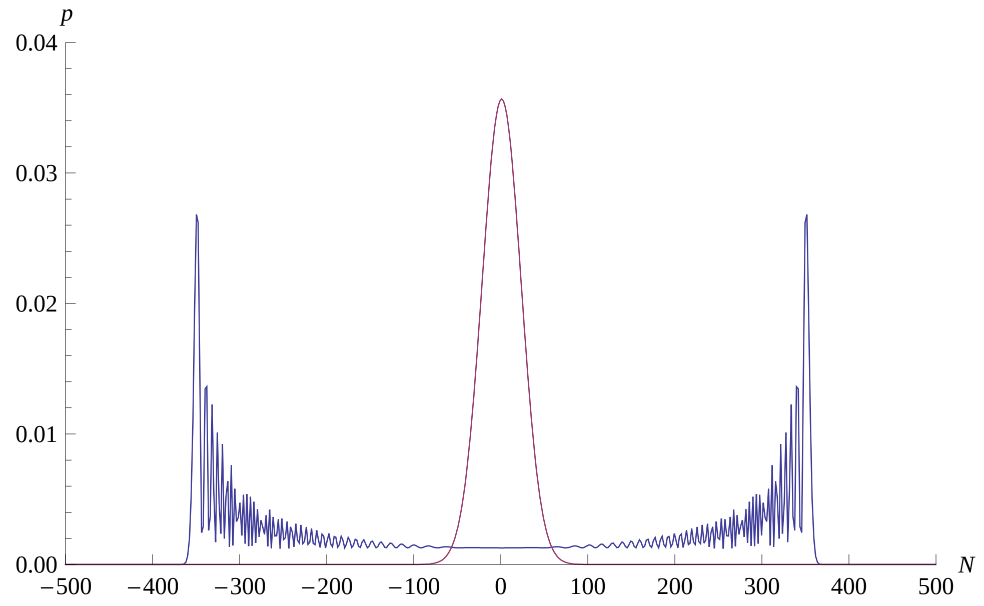


Figure 4.2: Probability of measurement at a position for the balanced 1D quantum walk and the classical 1D walk. Likewise only the probabilities for even N are shown. The classical walk is in the form of Gaussian, and the quantum walk is approximately constant with quantum interference effects, and is clearly spreading far faster than a Gaussian. The quantum walk was generated using by the coin operator Y in initial state $|\psi\rangle = |0,-\rangle$ or by H with initial state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0,-\rangle + i\,|0,+\rangle)$.

the primary method of attack for these problems. Once a result is obtained numerically, then it may be well worth attempting to prove analytically.

4.4 Quantum walks on regular graphs

A quantum random walk like that on the line can be defined for arbitrary undirected graphs by using different position and coin spaces. We first define it for a regular graph, one with the same number of edges adjacent to each vertex, and then extend it to an arbitrary graph. As with the quantum walk on the line, we perform the walk on the tensor product of a position space and a coin space $\mathcal{H}_s \otimes \mathcal{H}_c$. Again, the position space has discrete values corresponding to each node of the graph. Now the coin space is of dimension n for the n paths out of each vertex, which are labeled by edges corresponding to the path leaving a vertex. At each vertex, the edges are ordered by some index i. Then this space is given by the set of adjacent vertex-edge pairs $|v, i\rangle$ as basis vectors, referring to vertices v and the adjacent edges with index i. This is shown for an arbitrary regular graph in Fig. 4.3. In our quantum walk, the particle no longer only be located at vertices, but now we can think of the coin space as directly associated with edges on the graph, so the positions the particle can occupy are at each adjacent vertex-edge pair.

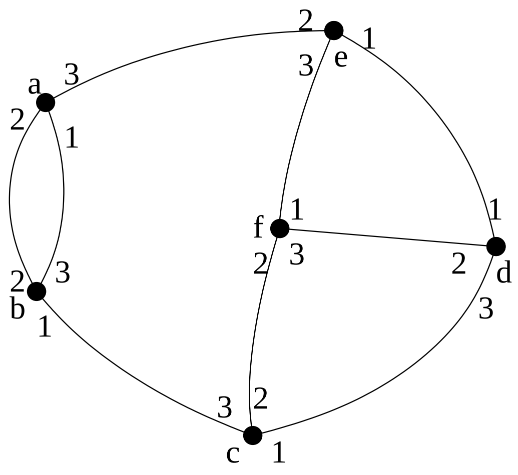


Figure 4.3: A regular graph. The position space has letters a-f corresponding to each vertex and numbers 1-3 corresponding to indexes for the edges at each vertex.

The operator constituting a step of the random walk U is defined again by the composite of a coinflip $I \otimes C$ and a shift S. The coin operator is again a unitary operator on the subspace $\mathcal{H}_c$, and the shift is defined by its action on each basis vector,

$$S |v, i\rangle = |v', i'\rangle , \tag{4.14}$$

where v' is the other vertex attached to edge marked by i, and i' is the labeling for the index associated with the same edge marked by i. There are many ways in which the index i may be ordered at each vertex. As in Fig. 4.3, it may be chosen arbitrarily, but more generally, when possible, it is chosen in some way that reflects the symmetry of the graph, or in which i and i' may be labeled the same. For instance on a square lattice, indexes $\uparrow$, $\rightarrow$, $\downarrow$ and $\leftarrow$ would be chosen corresponding to the direction of each path.

In general, the coin operator C may also be chosen arbitrarily, but two such choices are most common. The first such coin is the discrete Fourier transform (DFT). Recall that the Fourier Transform $X(f)$ is given by

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i f t} dt. \tag{4.15}$$

Then the discrete Fourier transform $\vec{X} \in \mathbb{C}^n$ performs the same transformation as the Fourier Transform on the discrete set of n data points $\vec{x} \in \mathbb{C}^n$ by

$$X_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j e^{-2\pi i j k/n} \tag{4.16}$$

with the substitutions $t \to j$, $dt \to 1$ and $f \to k/n$. The factor $1/\sqrt{n}$ in front is a normalization constant so that $\vec{x}$ and $\vec{X}$ have the same length, making the DFT transformation unitary, as can be easily verified. Then we can re-express the DFT as mapping basis vectors $|j\rangle$ to $|k\rangle$ as with

$$|j\rangle \to \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} e^{2\pi i j k/n} |k\rangle, \tag{4.17}$$

which in matrix form in the standard basis gives us,

$$DFT = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \gamma & \gamma^2 & \cdots & \gamma^{n-1} \\ 1 & \gamma^2 & \gamma^4 & \cdots & \gamma^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \gamma^{n-1} & \gamma^{2(n-1)} & \cdots & \gamma^{(n-1)(n-1)} \end{pmatrix}, \tag{4.18}$$

where $\gamma = e^{2\pi i/n}$ is the nth root of unity. It turns out that the DFT gate can in general be constructed easily out of simple gates for use in a quantum computer [23]. This gate makes it easy to find the periodicity of functions, and is the basis of a whole family of quantum algorithms based off Shor's revolutionary algorithm for integer factorization in time $O(\text{poly})$ in contrast to the best classical algorithm running in time $O(\exp)$. But this operation is also remarkably useful for quantum walks. In some sense it is the natural generalization of the Hadamard transformation (Eq. (4.10)), as the Hadamard transform is the discrete Fourier transform with $n = 2$. In the classical limit this coin gives an equal probability of a transition from any state to any other, since each matrix element has equal absolute value, and this operation is the unique such choice, up to an irrelevant phase factor and permutation of the rows and columns. This coin lets us exactly reproduce a random walk on a graph with equal probability of moving along each edge from a vertex, but it introduces phase differences dependent upon the path. These phases break the symmetry of the original random walk, just as in the case of the Hadamard coin H vs. the balanced coin Y (Eq. (4.11)), but for $n > 2$ there is not an easy way to account for this lack of balance with a difference initial state.

The second coin of prominent use is an extension of the balanced coin Y. It is also, perhaps not coincidently, an operation of central importance in the other main family of quantum algorithms. Physically, a particle in a quantum walk can be considered to be moving through the graph and scattering off a vertex. If the coin matrix corresponds to a physical operation of scattering, then the resulting amplitude distribution should respect the symmetry of the interaction. So the balanced coin reflects the symmetry of a random walk on a line in a way that the Hadamard coin does not by treating the transitions from positive to negative moving particles and vice-versa symmetrically. These are also the sort of coins that yield quantum walks corresponding to physical process, such as discrete versions of the Schrodinger and Dirac equations. For example, a quantum walk on a square lattice, on the basis $\{|\uparrow\rangle, |\rightarrow\rangle, |\downarrow\rangle, |\leftarrow\rangle\}$ corresponding to shifts in the direction of the arrow, one would expect a coin of the form

$$C = \begin{pmatrix} a & c & b & c \\ c & a & c & b \\ b & c & a & c \\ c & b & c & a \end{pmatrix}, \tag{4.19}$$

as particles arriving in any direction at a vertex should have the same amplitudes associated with reflection (a), transmission (b) or deflection to the left or right at 90 degrees (c).

For a general graph without any additional structure, there is only one feature that breaks the symmetry of the problem: the edge along which a probability amplitude representing a particle arrives. Then coins respecting this symmetry would be matrices of the form

$$C = \begin{pmatrix} a & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{pmatrix} \tag{4.20}$$

at each vertex, with some constant amplitude a for the diagonal elements and b for the off-diagonal elements. The constant off-diagonal elements associate the same amplitude with each possible edge change. Using the requirement that this matrix must be unitary, we can place restrictions on a and b. By the orthonormality of the columns,

$$|a|^2 + (n-1)|b|^2 = 1 \tag{4.21a}$$
$$a^*b + b^*a + (n-2)|b|^2 = 0. \tag{4.21b}$$

Let $\alpha = |a|$ and $\beta = |b|$ be the magnitudes of a and b. Then action of the coin will clearly only depend upon the phase difference Δ between a and b as any overall phase can be factored out of the matrix, and phase factors are not measurable. Then as $a^*b + b^*a = 2\alpha\beta\cos\Delta$, we have

$$\alpha^2 + (n-1)\beta^2 = 1 \tag{4.22a}$$
$$2\alpha\beta\cos\Delta + (n-2)\beta^2 = 0. \tag{4.22b}$$

For $n = 2$, there are three solutions: the identity, the Pauli matrix σ_x and symmetric coin Y from Eq. (4.11). Solving these equations for α and β yields a continuum of solutions for $n > 2$. These are given by

$$\alpha = \left(1 + \frac{4(n-1)\cos\Delta}{(n-2)^2}\right)^{-1/2} \tag{4.23a}$$
$$\beta = \frac{2\alpha\cos\Delta}{n-2}. \tag{4.23b}$$

These solutions are shown graphically in Fig 4.4. As $\Delta \to \pi/2$, that is as a and b become orthogonal vectors in the complex plane, $\beta \to 0$, so the coin is some multiple of the identity. As $\Delta \to 0$, a and b becomes parallel vectors, yielding another solution that can be written as a real matrix (clearly a and b will have to differ in sign). Making an arbitrary sign choice, the case $\Delta = 0$ simplifies to the diffusion operator G from Grover's search algorithm,

$$G = \begin{pmatrix} -1+\frac{2}{n} & \frac{2}{n} & \cdots & \frac{2}{n} \\ \frac{2}{n} & -1+\frac{2}{n} & \cdots & \frac{2}{n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{n} & \frac{2}{n} & \cdots & -1+\frac{2}{n} \end{pmatrix}, \tag{4.24}$$

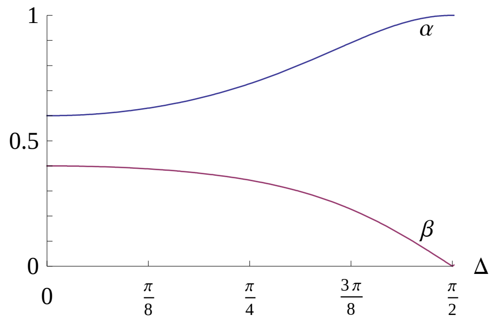


Figure 4.4: Magnitudes of diagonal elements α and non-diagonal elements β as a function of the phase difference Δ with $n = 5$.

which can also be written

$$G = -I + 2\,|\psi\rangle\langle\psi|\,, \tag{4.25}$$

where $|\psi\rangle = \frac{1}{\sqrt{n}}\sum_{i=0}^{n}|i\rangle$ is the equal superposition over all states. This choice of "Grover's coin" ensures that amplitudes spreads through a quantum walk at the maximum rate, as using the identity operator as a coin means that the walk does not move at all. Clearly as n increases, all these solutions including Grover's coin tend strictly toward the identity operator, but this cannot be avoided, and as it turns out, does not mean that this coin is necessarily less powerful than the classical "fair" coin with equal transition probabilities. On general graphs, in addition to being more physically realistic, these sorts of coins are more likely to creating interesting quantum walks, as respecting the symmetry of the graph allows quantum interference to yield non-classical results. Another advantage of this particular coin is that the matrix has substantial symmetry that could make analytical analysis substantially easier.

Not surprisingly, these more general quantum walks share qualitative and quantitative features with the quantum walk on the line. A simple example is the quantum walk on a square lattice. Using Grover's coin operator G and associating the indexes at each vertex with shifts on the lattice up, down, left and right, we obtain the symmetric picture shown in Fig. 4.5. Again, the walk spreads much faster than a Gaussian and shows quantum interference effects.

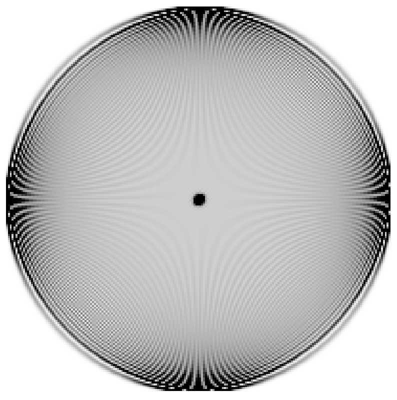


Figure 4.5: Probability distribution for quantum walk on a 2D lattice. The walk was generated using the coin G and run for 200 time steps. Only alternate positions are shown as the others have zero probability. The axes run from -200 to 200. Generated (with permission) from a Maple script by Pemantle [25].

4.5 Walks on arbitrary undirected graphs

We can expand the definition of a random walk on any regular graph to undirected graphs in general with ease. This can be done by now referring explicitly to the set $|v, e\rangle$ of adjacent vertex edge pairs as the set of basis states, as shown

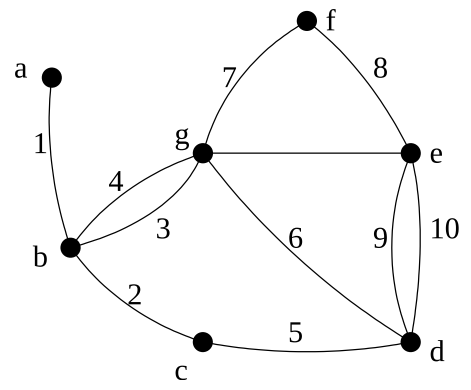


Figure 4.6: A undirected graph. The set $|v, e\rangle$ has an element for each adjacent vertex edge pair with vertices a-f and edges 1-10.

in Fig. 4.6. The indexes i are replaced by more specific edges e. Since there may be different numbers of edges corresponding to a specific vertex, we can no longer refer to the space on which the walk is performed in the separable form $\mathcal{H}_s \otimes \mathcal{H}_c$. Coin operators C_n with different dimensionality n are applied to the subspace spanned by each vertex with a different number of adjacent edges. These operators C_n can be any unitary operators such as the ones described in the previous section. Instead of applying the coin operator $I \otimes C$, we use its natural generalization C', where C' is a new operator that can be written in block diagonal form in a basis with the basis states ordered by vertex,

$$C' = \begin{pmatrix} C_{n_a} & & & \\ & C_{n_b} & & \\ & & C_{n_c} & \\ & & & \ddots \end{pmatrix}, \tag{4.26}$$

where each n_v refers to the number of loops at vertex v. Then the unitary walk step is given by $U = S \cdot C'$.

4.6 Quantum walks on quantum computers

Quantum walks are an interesting and beautiful exercise in mathematics but they are also a tool that can be used for quantum algorithms. Just as classical walks can be run efficiently on classical computers, so can quantum walks be run efficiently on quantum computers. In fact, the proof of this second result follows directly from the first. If a classical walk can be run efficiently on a classical computer, then it is done so by some classical circuit defined in terms of logic gates, with the addition of some random number generator as input to a transition matrix. The circuit can be made reversible with only a constant decrease in speed by use of the Toffoli gate, for instance, as detailed in Sec. 3.4. A difference of any constant multiple is irrelevant because quantum

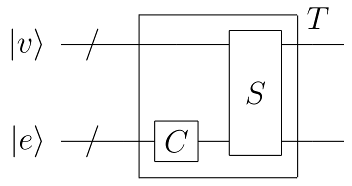


Figure 4.7: Quantum circuit implementation of a quantum walk on a regular graph. The walk operation U is outlined by the box marked by T, to indicate that the circuit inside is repeated sequentially T times. There are qubits representing the vertices $|v\rangle$ and the edges $|e\rangle$, and the gates marked C and S are the coin and shift operations.

and classical hardware will differ massively in speed and design anyways. Then in place of using a stochastic transition between states, we place the "coin flip" operator directly into the quantum circuit instead.

More concretely, we can also create a quantum circuit diagrams for an arbitrary quantum walk that show directly how they could be implemented on a quantum computer. Such a diagram is shown in Fig. 4.7. For simplicity, it shows a quantum walk on a regular graph; the implementation of a general undirected graph is similar, although it is harder to draw the circuit clearly.

Chapter 5

Quantum walk search

In previous chapters we have presented both random walk search based algorithms for k-SAT and quantum random walks as a family of algorithms that can be faster than classical random walks. The challenge discussed in this chapter and the next is to find faster quantum versions of these randomized algorithms.

5.1 Quantum search

Searching a database is a basic and essential task for computers. In the classical world, finding a particular entry in an unsorted database of N items always takes on average checking $N/2$ entries. So classical database search runs in time $O(N)$. In contrast, Lov Grover's search algorithm can search any unstructured database by with only $O(\sqrt{N})$ queries of an Oracle $f(x)$ [23]. This result is remarkable because almost every important algorithm in classical computing can be phrased as a search of some sort, even if it works as naively as by checking every possible solution. The quadratic speedup offered by Grover's algorithm has been proved to be the best possible result for general search.

But real databases have their entries stored in physical media. It is far easier to flip to the next page of this thesis than to head to the library and check out another book, and it takes more effort to flip two pages than one page. Moreover, information may be transmitted no faster than the speed of light, and actual quantum circuits will take time to look up entries. So as a new model, consider search of a database on a graph with one entry at each point. Then instead of reading all entries simultaneously, we imagine a "quantum robot" moving through the database in the form of operations that can only be applied locally [8]. If this robot is running Grover's algorithm, it will take M times longer to run the search algorithm, where M is average distance it needs to move between vertices. It still only requires only $O(\sqrt{N})$ queries, but between each evaluation it takes M steps to move to the next vertex. More precisely, the Oracle operation itself now takes $O(M)$ steps to evaluate. Then as a square grid of N elements has distance $O(\sqrt{N})$ between elements, running Grover's search on this structured database in fact takes order $\sqrt{N} \times \sqrt{N} = N$ steps, no better than classical search [11].

Quantum walks resolve this issue by providing a means to perform local search on structured databases often just as fast as Grover's algorithm. The quantum walk by analogy has robots as cellular automata stuck at each cell who can only pass quantum states off to their neighbors. These search algorithms are remarkably simple and have been implemented for a variety of graphs reaching the theoretical limit of $O(\sqrt{N})$ performance or close to it for many graphs. Aaronson and Ambainis have shown that for 3 or more dimensions, quantum search on a grid can be performed in time $O(\sqrt{N})$, although their best algorithm for search on the 2 dimensional lattice is in time $O\left(\sqrt{N}\log^{3/2} N\right)$ [2].

5.2 Binary hypercube search

Quantum walk search works on many graphs, but search on most graphs lacks an algorithmic application. Recall the binary hypercube in n-dimensions, as introduced in the discussion of satisfiable algorithms in Sec. 2.3. We can consider this graph as a database with $N = 2^n$ entries. One vertex x_0 chosen arbitrarily is marked as the solution to our search. Classically, starting from any initial vertex, it takes $O(2^n)$ steps until a solution is reached with high probability. Since the maximum distance between any two vertices is $M = n$, Grover's algorithm could search the hypercube in $O(n\sqrt{2^n})$ steps. The following algorithm, due to Shevi, Kempe and Whaley [28], performs the spatial search in $O(\sqrt{2^n})$ steps.

Since the hypercube is regular, we can choose a basis by the vertices and an index keeping track of each edge. The set of basis vectors is of the form $|x, i\rangle \in \mathcal{H}_s \otimes \mathcal{H}_c$, where each vertex is a string of bits x, and the indexes i correspond to the ith variable being flipped between the two adjacent vertices. In general, one arbitrary vertex x_0 is marked as the solution. The algorithm is given as follows:

1. The system is initialized in the equal superposition state $\frac{1}{\sqrt{n2^n}}\sum_{x,i}|x,i\rangle$, as can be obtained by performing a Hadamard transform on each qubit representing the space $\mathcal{H}_s$ and $\mathcal{H}_c$.

2. The operation $U = S \cdot C'$ is applied for $\frac{\pi}{2}\sqrt{2^n}$ steps.

   (a) At unmarked vertices, the coin C_0 is used. At the marked vertex, the coin C_1 is used. Then the operation C' is given by

   $$C' = I \otimes C_0 + |x_0\rangle\langle x_0| \otimes (C_1 - C_0). \tag{5.1}$$

   Let $C_0 = G$ (Grover's operator from Eq. (4.24)) and $C_1 = -I$.

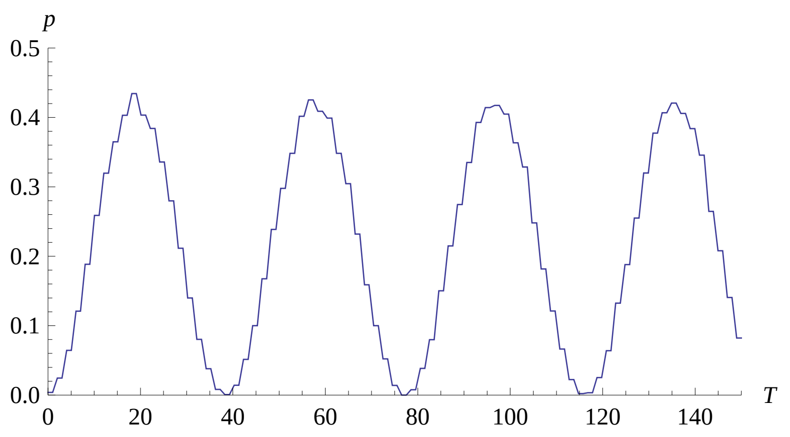


Figure 5.1: Probability of measurement at solution vs. time for quantum walk search on the 8-dimensional hypercube. The probability of measurement clearly builds up periodically, reaching a maximum after about 20 time steps.

   (b) The shift operation $S : |x, i\rangle \rightarrow |x^i, i\rangle$ is applied, where x_i indicates the bit string x with the ith bit flipped. This is the normal shift operation.

3. Measurement is performed on the states $|x\rangle$. With some probability $\frac{1}{2} - O(1/n)$, the state $|x_0\rangle$ is measured, so by repeating this algorithm some constant number of times we have the result with arbitrarily high probability.

The proof of this result is quite non-trivial and is too involved to go into here, but as with the classical 2-SAT algorithm it involves reducing this walk over the hypercube to the line. The build up of probability at the solution is illustrated in Fig. 5.1.

Numerically, the same speedup can be obtained for different arbitrary choices of coins C_0 and C_1. Essentially the same procedure also works on many other regular graphs such as a hexagonal grid or a cubic lattice. In these symmetrical systems, we can understand these quantum walks working as a search procedure because the marked coin C_1 perturbs a symmetrical superposition over all states of a symmetrical graph. Then the system can only change in a manner that respects the symmetry of the quantum walk procedure. After waiting some appropriate number of repetitions, the position of particle on the graph is measured. Each of these quantum random walks perform search on graphs significantly faster than their classical counterparts, although they do not beat the quadratic improvement offered by Grover's algorithm for unstructured search.

Since they are quantum walks, these structured quantum search algorithms can also be implemented, of course, on a quantum computer. A quantum circuit for the binary hypercube search algorithm using an Oracle function $f(x)$ is shown in Fig. 5.2. This Oracle function $f(x)$ equals 1 if $x = x_0$

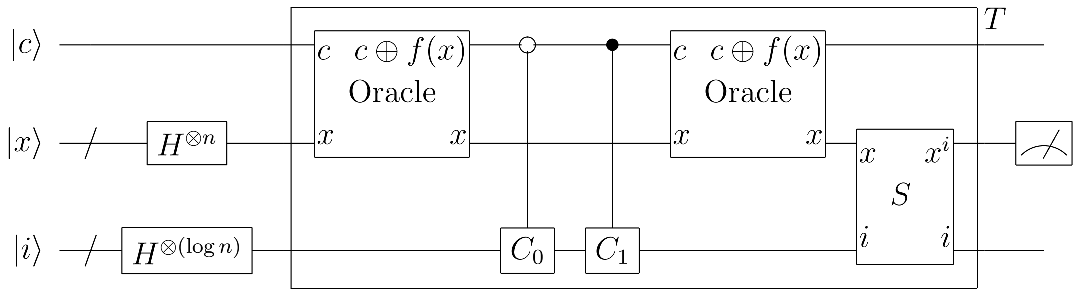


Figure 5.2: Quantum circuit implementation of binary hypercube search algorithm. The control qubit $|c\rangle$ is initialized in the state $|0\rangle$ as are the qubits referred to by the labels $|x\rangle$ and $|i\rangle$. The position states $|x\rangle$ are stored in a set of n-qubits and the coin index $|i\rangle$ can without loss of generality be represented by $\log n$ qubits since it needs to have n distinct states. The Oracle circuit flips the value of $|c\rangle$ if $x = x_0$ and is run again after the controlled operations to reset the value of $|c\rangle$. The walk step U marked by the box is repeated $T = O(\sqrt{2^n})$ times.

and 0 otherwise, and is the standard way in which a particular element is marked in quantum algorithms. Writing the Oracle as circuit element suggests the approach we will use for satisfiability problems in which case the Oracle function actually involves the evaluation of a circuit, the formula for the SAT problem. The Oracle is in fact called twice for each iteration to reverse its action on the control qubit, but again some reduction in speed by a constant multiple is irrelevant in a broader sense.

5.3 Converting hypercube search to k-SAT

Our challenge now is to adapt this quantum search procedure to solve satisfiability problems. However, there are a significant differences between the hypercube search problem and the k-SAT random walk algorithms. The k-SAT algorithms run on directed graphs with possibly redundant paths, on a subset of the binary hypercube. It is also possible and common to have multiple solutions. Furthermore, several steps in the procedure such as the initialization step rely on the symmetric nature and known structure of the graph at each vertex. None of these challenges is necessarily insurmountable but a quantum version of this walk will clearly be far less trivial on the hypercube. As a general principle, even without proof or any particular reason, we can safely assume that no altering of the graph in the process of making it suitable for quantum walks would improve the speed of the classical algorithm, or the approach would already have been taken. Under this principle, we will consider two main approaches. The first is explained in the remainder of this chapter and the second in Chapter 6.

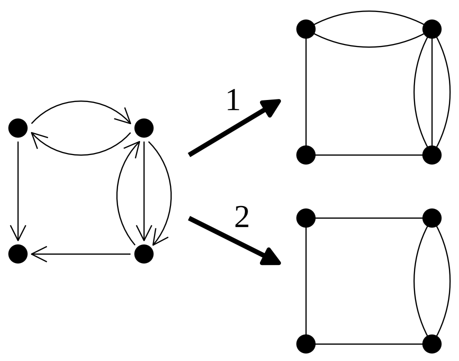


Figure 5.3: Methods for making a directed graph undirected. Method 1 replaces each directed path with an undirected path and method 2 places undirected paths along an edge equal to the larger of the number of paths in each direction.

The easiest of these difficulties to resolve are redundant and missing paths. Instead of using the quantum walk algorithm for a regular graph, we use general algorithms that can use different size coins corresponding to any number of edges at a vertex. This means that there needs to some sort of "edge Oracle" in the quantum circuit that stores which edges are accessible to each vertex.

Whether the truth assignment indicated at a vertex is a solution is checked by an Oracle function given by the classical circuit that would evaluate the SAT formula. Then multiple solutions are not any additional trouble to mark with a special coin. The trouble, as is obvious numerically, is that having multiple solutions can very easily destroy the power of the marking coin since they would vastly change the symmetry of the graph. One way to deal with the possibility of multiple solutions is to ignore it by only testing formulas with one solution. This corresponds to one heuristic for hard problem instances, as discussed in Sec. 2.4. Classically, formulas with multiple solutions should converge faster, and it is impossible to distinguish these cases a priori when presented with an actual formula to solve. This approach is not entirely satisfying, but even an algorithm for k-SAT that only works on these instances could be notable.

The main remaining difficulty is the classical random walks proceeds on directed graphs, but all of our quantum walk algorithms so far only work on undirected graphs. There are two possible responses to this dilemma. We can either adjust the graph in some way to make it undirected, or consider a very different model based on some sort of "directed quantum walk." This later choice is the one made in Chapter 6.

Here we consider making the graph undirected, and again there are two obvious choices: either (1) doubling up each directed path with a directed path in the opposite direction, or (2) merely adding additional paths to an edge in the direction with less paths, until the number of paths in each direction is equal. These options are shown in Fig. 5.3. The later choice seems to be better for a two marginal reasons. First, it alters the classical algorithm less, as there are less additional paths added into the graph, so the transition probabilities in the classical limit would be altered by a lesser amount. Secondly, it would also have marginally less significant resource requirements for implementation, both on a quantum device and for simulating performance on a classical computer. Nonetheless, we will look at both approaches.

Unfortunately, both of these approaches, and any based on an undirected walk, would not work in the classical limit. This is because the random walk SAT algorithms we have examined rely on solution vertices having no paths out to trap random walk attempts there. When the graph is made undirected, the structure does not force the particle to stay at potential solutions, as undirected graphs always have the same number of edges in and out of each vertex. It appears that our quantum search algorithms and the classical structured search for SAT problems may rely on antithetical principles.

Still, we will evaluate the merit of this procedure experimentally. Now that we have an undirected graph on which to perform a quantum walk, we consider the walk using the general formalism for undirected walks discussed in the previous chapter. Instead of using the same coins C_0 and C_1, we use the coins of the same form scaled at each vertex to the number of attached edges. We also need to initialize graph in some uniform state that is not dependent upon the structure, as the walk possessing non-local knowledge of the location of each path would require already having solved the problem. To do so, we create an equal superposition at each vertex such that the probability of the walk starting at each vertex is the same, even if the edges from distinct vertices start with different values.

There is one major distinction with the choice of marking coins from the hypercube search algorithm. In the original hypercube search, the coin $C_1 = -G$ produces identical results to the coin $C_1 = -I$. This is because the walk is initialized in an entirely symmetric state, the amplitude on each vertex will always be some multiple a of the equal superposition state $|\psi\rangle$. Then using the form of G from Eq. (4.25),

$$-Ga\,|\psi\rangle = -(-I + 2\,|\psi\rangle\langle\psi|)a\,|\psi\rangle = a\,|\psi\rangle\,, \tag{5.2}$$

so the equal superposition $|\psi\rangle$ is an eigenvector of the operator $-G$ with eigenvalue -1. Since the walk is initialized in this eigenstate, the operator has the same action as the operator $-I$. In contrast, on these hypercube walks for k-SAT, the symmetry of the graphs has been thrown off in some way. Accordingly it is now possible that these two coin may result in different walks.

The full procedure for the algorithm is as follows:

1. Initialize walk in a uniform state equally likely to be at any vertex.
2. The operation $U = S \cdot C'$ is applied for t steps.
   (a) Using an edge Oracle from the classical algorithm, record the edges adjacent to each vertex from the classical walk.

(b) Combine marked directed edges into undirected edges, and count their multiplicity both along every edge of the hypercube and at each vertex.

(c) Using the solution Oracle, apply coins of the form C_0 at unmarked vertices, and C_1 at vertices that represent a satisfying truth assignment. These coins will of size as given by the multiplicity calculated in the previous step.

(d) Then apply the shift operator $S : |x, e\rangle \to |x', e\rangle$ where x' is the other vertex to which the edge e is assigned. This shift is dependent upon the recorded multiplicities.

3. Measure the position of the particle in the quantum walk.

4. Evaluate the SAT formula with the truth assignment from the position of the quantum walk with a classical circuit.

5. If the formula is unsatisfied and the quantum walk has been run less than r times, go back to step 1.

This algorithm uses some classical steps, but a quantum walk is used for the random walk procedure. As with the classical algorithms, it only completes if there is no satisfying truth assignment, but unlike the classical procedure, it cannot stop in the middle of the random walk procedure if it reaches the solution. If after evaluating the quantum walk r times for t steps each (time $T = rt$ in total) no satisfying assignment has been found, the algorithm should terminate stating that the formula is unsatisfying with some probability greater than $1 - \epsilon$, where ϵ is some maximum error rate. These counts r and t remain undetermined but will be evaluated numerically once we know how the probability of the quantum walk succeeding varies over time so that the total time the algorithm takes to run is minimized.

In principle, we can create a quantum circuit to run this algorithm only using at most $kmn = O(n^2)$ qubits to store the associated edges assignments. This is possible since each of n variables may only appear unsatisfied in at most km literals for any truth assignment. Also in principle, this walk could be implemented with only at most some constant decrease in speed and increase in storage requirements associated with making each operation reversible. We omit actually drawing a full quantum circuit implementation for this algorithm, as it would be quite complicated and unilluminating with making reversible some non-trivial operations such as those involved with counting the multiplicities of edges, especially when making the graph undirected.

5.4 Performance of probabilistic algorithms

To determine whether this search procedure works and to evaluate its performance, we need to be able to guess a time at which there is a high probability of measuring the walk at the solution. As with the original hypercube search, we expect the probability p of measuring the particle at the solution to build periodically. But as with the classical 3-SAT algorithm, running the walk reinitialized r times could be faster than only letting the same walk run once for a longer time. Subject to some upper bound ϵ on the probability of failing to categorize a SAT formula as satisfiable, we would like to minimize the total run time $T = rt$ as a function of the probability $p(t)$ that a satisfiable formula is measured at a satisfying truth assignment after t steps of each random walk. This will give us optimal choices of n and t, but in particular, we expect that the cutoff time t_c of "maximum returns" will be independent of ϵ as $\epsilon \to 0$. This would match the result obtained for the classical 3-SAT algorithm in Chapter 2 of $t = 3n$.

This algorithm terminates in the case that the formula is satisfiable upon the quantum walk successfully returning a satisfying truth assignment. The probability in one evaluation of the quantum walk of returning a non-satisfying truth assignment is $1 - p(t)$. Then the probability $\Pr(F)$ of failing to return a satisfying truth assignment each of r times is given by

$$\Pr(F) = [1 - p(t)]^r \,, \tag{5.3}$$

as each trial is independent. Using the result $T = rt$, we would like to minimize the total time T given that the probability of failure is less than some upper bound ϵ. Excluding the trivial case $t = 0$, we have the constraint

$$[1 - p(t)]^{T/t} \leq \epsilon, \tag{5.4}$$

which will be minimized in the case of equality. Simple algebraic manipulation shows that the total time T is given in terms of ϵ and t by

$$T(t) = \frac{t \log \epsilon}{\log [1 - p(t)]}, \tag{5.5}$$

which should be minimized as a function of t to find the optimal number of steps t_c for which to run each iteration of the walk. This equation allows easy determination of t_c from values of $p(t)$ obtained from simulation for some finite number of steps, and holds for any probabilistic algorithm for k-SAT of this design. Further, as is appropriate, this minimal value t_c is independent of ϵ as $\log \epsilon$ is a constant factor. Thus we can calculate the absolute time requirements for all k-SAT algorithms using Eq. (5.5) for any arbitrary choice of ϵ, so we choose $\epsilon = 1/e$ so that $\log \epsilon = -1$.

5.5 Quantum search on 2-SAT

We used the result of the previous section to numerically evaluate the performance of these adapted quantum search algorithms on 2-SAT. As a general procedure, we simulated the performance of these quantum walk search algorithms for 2-SAT for a number of formulas with n variables as chosen at random at the phase transition, as described in Sec. 2.4. We chose to try algorithms on 2-SAT first, both because classically it is an easier problem, and because it would be easier to simulate such algorithms for 2-SAT as well. This is because the difficulty of simulating a quantum walk scales with the number of states in the corresponding basis, and hard 2-SAT formulas have about 4 times fewer clauses than with 3-SAT formulas and a correspondingly smaller number of edges on their graphs.

We simulated the probability of measuring the walk at a satisfying truth assignment as a function of the time for which it was allowed to evolve undisturbed, and found the time t that minimized the total time T for each individual SAT formula. We also found the time t_c that minimized the time required to evaluate most SAT instances by minimizing the quantity

$$\langle T \rangle + c\sigma_T, \tag{5.6}$$

as a function of t, where c is some small constant (we chose 2), and σ_T is the standard deviation of T, used so that nearly all SAT instances will converge in the appropriate time. Then we considered the resulting total time T for each SAT formula using t_c, so that the algorithm would be likely to correctly classify each SAT formula as satisfiable. This procedure optimizes for the best candidate algorithm using the procedure from the previous section, with time $t = t_c$.

As shown in Fig. 5.4, on general 2-SAT instances at the phase transition, the walk performs abysmally. The total time required is exponential and far above the time required for the classical walk. The only consolation is that at least this algorithm performs better than the most naive, blind guess algorithm. Also as it turns out, our guess that $-G$ and $-I$ might have significantly different performance turned out to be unfounded. As these marking coins yielded almost completely identical performance, all results are all shown for clarity only using $-I$ as the marking coin. Fig. 5.5 shows another distinct effect, that the quantum walk performs worse on those 2-SAT formulas with the most satisfying truth assignments, exactly opposite the performance for the classical walk.

This performance leads us to consider how well this algorithm works strictly on formulas with only one solution. For these problems, we have another algorithm that provides a baseline comparison—the original quantum hypercube search. Unfortunately, as Fig. 5.6 shows, our modified algorithm actually performs worse than the straightforward hypercube search. The performance gap is small, suggesting that the process of making the graph undirected made the walk equivalent to a slight random perturbation of the hypercube search. This is similar to numerical results obtained by Krovi and Brun [18] for quantum search for a single element on hypercubes with minor a distortion formed by swapping paths such that the number of edges at each vertex remains con-

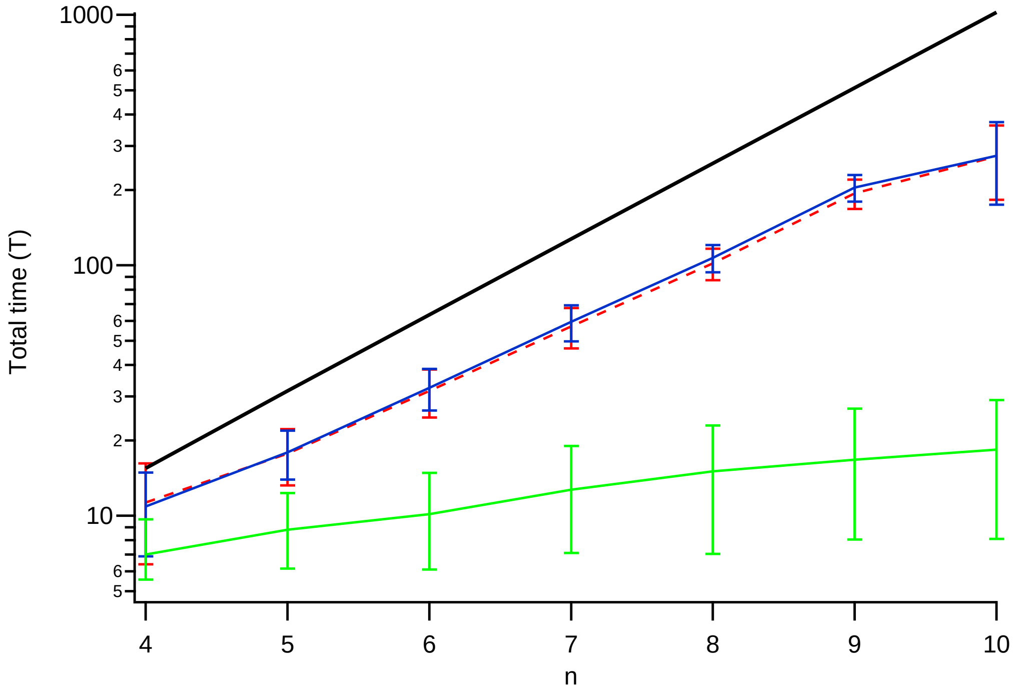


Figure 5.4: Scaled time requirement to solve 2-SAT formulas by quantum walk search. These values were determined from Eq. (5.5) using $p(t)$ as obtained numerically sampled over 100 formulas chosen at the phase transition. The classical run time from Eq. (5.5) is given by the green line that is a lower bound, and the thick black line above shows the performance from blind guessing. The dashed line shows the performance of the walk on the graph modified by method 1, and the solid line shows it with method 2. Note that the scaling of the graph is logarithmic, indicating that the quantum algorithms are exponentially slow and overwhelmingly slower than the classical algorithms.

stant. This means that these algorithms run in time slower than $O(2^{n/2})$, no improvement over the classical speed of $O(n^2)$.

Although it is interesting that these perturbed hypercube walk procedures still yielded quantum search procedures that worked, these search algorithms clearly did not respect enough of the structure of the 2-SAT problem to yield any algorithmic advantage. This suggests approaches based on directed quantum walks that may be able to preserve the structure of the problem, as are examined in the following chapter.

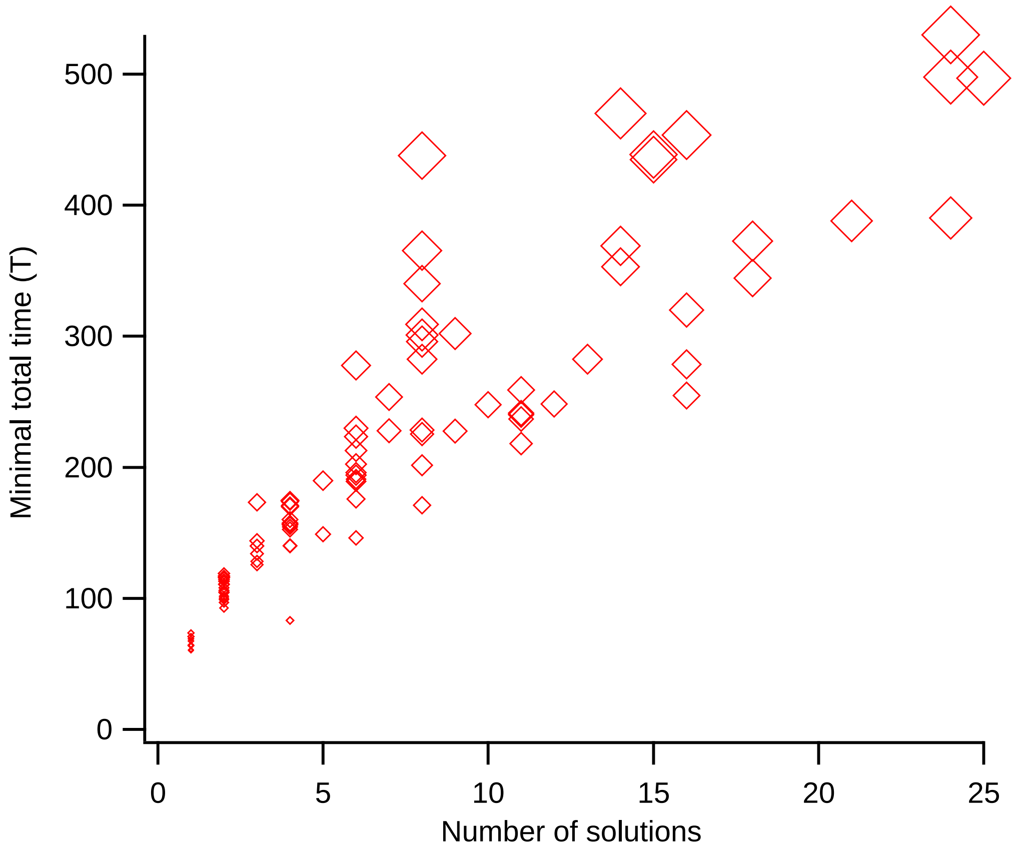


Figure 5.5: Minimal time required to solve 2-SAT formulas by quantum walk search vs. the number of satisfying truth assignments. The size of each diamond indicates the multiplicity at each point. These values were determined from Eq. (5.5) directly for each individual formula, and were taken from the data used to generated Fig. 5.4 with $n = 10$, so these formulas were at the phase transition. This plot clearly shows a strong correlation exactly opposite that of the classical algorithm: the presence of additional solutions makes the algorithm slower. Since these SAT formulas were also randomly generated at the phase transition, it shows that in general we cannot expect these formulas to only have a few solutions.

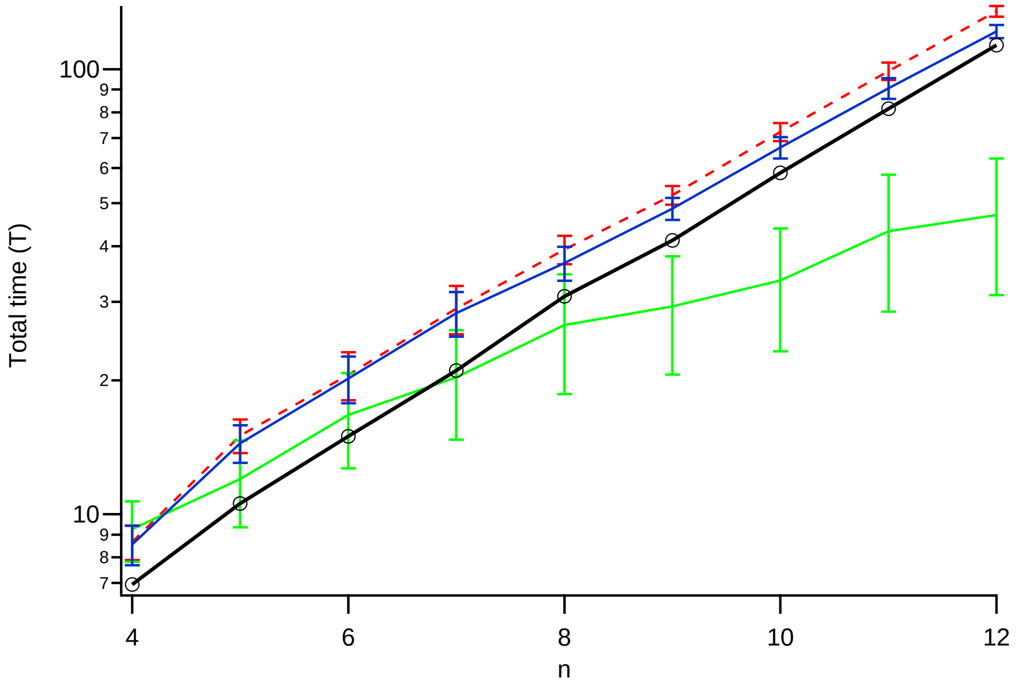


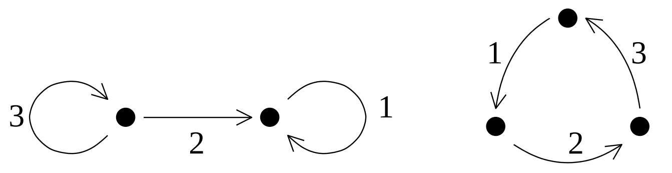
Figure 5.6: Scaled time requirement to solve 2-SAT formulas with one solution by quantum walk search. These values were determined from Eq. (5.5) using $p(t)$ as obtained numerically sampled over 100 formulas with one solution chosen at the phase transition for $n = 4 \ldots 10$, and 20 formulas each for $n = 11, 12$. The classical run time from Eq. (5.5) is given by the green line that is a lower bound. The thick line above shows the speed of the binary hypercube search. The dashed line shows the performance of the walk on the graph modified by method 1, and the solid line shows it with method 2. The same logarithmic scale is shown here as in Fig. 5.4. This graph clearly shows that when there is only one solution, the classical algorithm is slower and the quantum algorithms are faster. However, in each case using the unmodified quantum hypercube search procedure would have been faster than using the variation we developed.

Chapter 6

Directed quantum walks

6.1 Defining a directed walk

Classically, it is easy to perform any random walk with directed paths. Finding quantum directed walks for arbitrary directed walks is tricker. To get a sense of the difficulty involved, we will examine walks on the two trivial graphs shown below:



If the paths at a vertex for the first graph are selected uniformly at random, then the transition matrix is given by something like

$$M = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1/2 \\ 0 & 0 & 1/2 \end{pmatrix} \tag{6.1}$$

where the path goes from the 1 to the 0 state. Finding a quantum version of this walk seems almost certainly hopeless. There is no way a unitary matrix can have this transition matrix as its classical limit. In contrast, consider the three cycle graph with three vertices, each with an edge pointing to the next. Its transition matrix looks like

$$M = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{6.2}$$

which in fact is already a unitary matrix, too. So some random walks can be easily made quantum, and some cannot.

The constraint that a graph is *Eulerian* is sufficient for performing a quantum walk. A directed graph is Eulerian if each vertex has the same number

$$\begin{pmatrix} a & b & c & \cdots \\ a & b & c & \cdots \\ a & b & c & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Figure 6.1: No matrix of this form is unitary.

of edges in and out, as in our second example. If we want to have some unitary map that reassigns the amplitudes associated with n incoming edges $|e_{\text{in}}\rangle$ at a vertex then clearly we must reassign them to n outgoing edges $|e_{\text{out}}\rangle$, as unitary matrices are square. If we identify as a basis each directed edge, then a directed quantum walk can be written in similar form to the general undirected walk as

$$U = S \cdot C', \tag{6.3}$$

where C' is a coin flip operator as before and the shift operator S directly maps directed edges to an assigned next edge. In the undirected walk, there is an obvious next edge that does not break the symmetry of the operation: the edge that was just traversed in the opposite direction, but no such convenient choice exists for directed walks. This means that Grover's operator G is no longer a preferred coin operation, as we do not want any probability of returning to the same state at all, so the symmetry must be broken for transition to an arbitrary new edge. Accordingly the pairing of incoming and outgoing edges from each vertex could have a significant effect on the characteristics of a walk. There is no way to eliminate this symmetry breaking through a convenient choice of operation, as transition matrices that associate the same amplitude to each output cannot be unitary, as shown in Fig. 6.1. At best, we can pair incoming and outgoing edges at a vertex by some natural pairing that reflects the symmetry of the graph or the problem it is trying to solve.

The question of what directed graphs can be made quantum in general has been the focus of several papers [20, 27]. Specifically, the constraint known as *reversibility* has been shown to be necessary and sufficient for defining a quantum walk on a direct graph [20]. The reversibility requirement is that if there an edge from $a \to b$, there must also be some path from b back to a. But when quantum walks are defined on arbitrary reversible graphs, the transition probabilities in the classical limit to do not necessarily match that for the original graph. This fails in the objective of preserving the classical algorithmic properties of the graph. Further, the proof of this result relies upon the ability to use cycles through the graph in creating the coin space. This is entirely useless when the entire point of the algorithm is to understand the global nature of the graph.

These directed walks still only work on a quite limited subset of graphs, but they are a substantial increase over the set of undirected graphs. We will return to the task of generalizing quantum walks to any graph in Sec. 6.4 after seeing how far the use of Eulerian graphs can take us.

6.2 Directed walk on the line with loops

Before we proceed further with directed quantum walks, it is instructive to consider an example to demonstrate that they hold potential similar to that seen for undirected quantum walks. This was actually fairly difficult to find, because easy to visualize classical directed walks are already generally so fast. For instance, when propagating along a line, even if there is only some small probability p of moving forward, the expected position of the classical walk after T steps is pT. But the maximum speed for any conceivable quantum walk is T, since in each step we require that the walk operation be local. Both already cover the distance T in $O(T)$ time, so there is no room for meaningful quantum speedup. This is true for other regular repeating graphs, as well, in cases where undirected walks can spread quadratically faster. There is no room for directed walks to spread faster.

Fortunately, these cases of directed walks are not very interesting for algorithmic purposes, either, because the graphs can already be traversed in linear time. It does mean, however, that we will need a new way to show quantum speedup. A natural candidate is some measure of the complexity of the graph, the number of paths that would move a particle closer to the solution instead of remaining at the same distance or moving further away. Then we could hope to show improved speed as the graph gets more complicated and a classical walk would slow down.

Accordingly, we present a novel example of a toy directed quantum walk that shows remarkable quantum speedup. Consider the directed random walk along the line with $n-1$ self-loops at each vertex, as shown in Fig. 6.2. We can alternatively consider these self-loops at each vertex as additional small dimensions providing extra space for the walk to slow down. The classical speed of the walk can be easily scaled by adjusting n. At each vertex there is a probability $1/n$ of moving to the next vertex, so the expected position propagates a distance T/n in time T. Thus the classical walk proceeds in time $\Theta(1/n)$.

In the quantum version of this walk, we initialize the state at a position along the line and apply the n dimensional discrete Fourier transform (Eq. (4.18)) as the local coin operation at each vertex. We pair incoming and

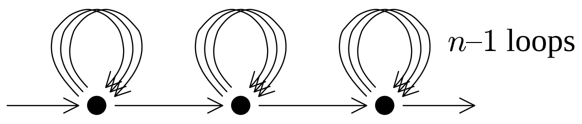


Figure 6.2: Directed walk on the line with $n-1$ self-loops at each vertex.

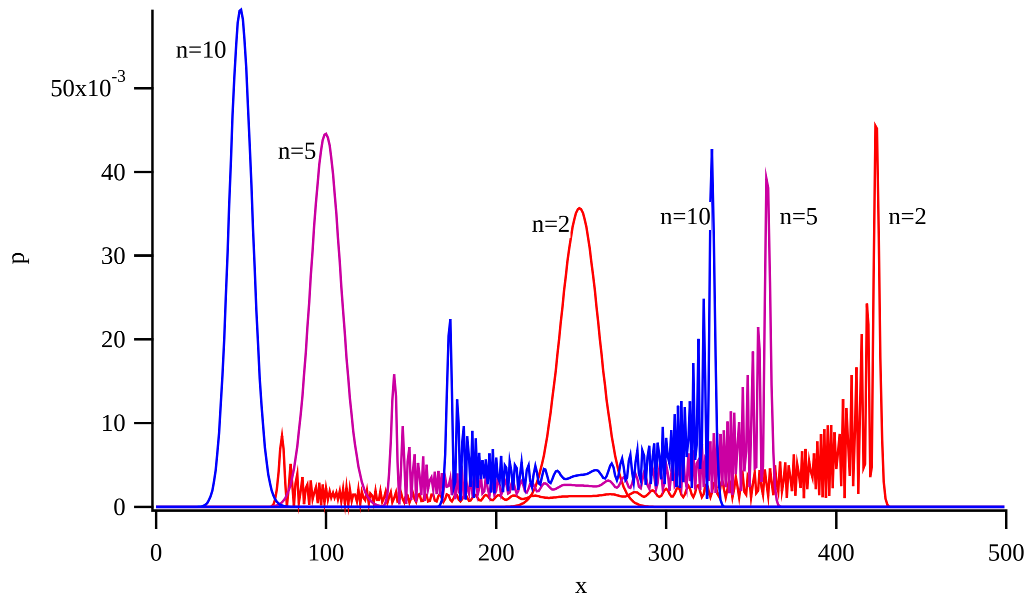


Figure 6.3: Classical vs. quantum directed walks with loops after 500 time steps, for increasing number of loops n. The classical walks are those with the Gaussian shape and the quantum walks are those exhibiting the characteristic interference pattern. The classical proceed on average a distance $500/n$, whereas the quantum walks converge to proceeding a constant distance.

outgoing edges in the only non-symmetry breaking manner, by associating self-loops with themselves and pairing the incoming and outgoing paths along the line. Starting in a definite state at one edge along the line, numerical results show that as $n \to \infty$, the expected position of this walk propagates T steps forward in time $T/2$, independent of n. This is in time $\Theta(1)$. Superimposed pictures of the walk after 100 steps for increasing n are shown in Fig. 6.3. The speed of the walk over time is shown in Fig. 6.4 clearly converging to 1/2 step forward per time step. The speed of this walk has been verified numerically for up to $n = 1000$.

This walk proceeds quickly by strongly exploiting the symmetry of its setup. We can understand how it works by considering the effects of the discrete Fourier transform in two steps. Starting with the $|0\rangle$ state, it is transformed to the equal superposition

$$|\psi\rangle = \frac{1}{\sqrt{n}}(|0\rangle + |1\rangle + \ldots + |n-1\rangle).$$

The inverse discrete Fourier transform is given by

$$|j\rangle \to \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} e^{-2\pi i jk/n} |k\rangle\,, \tag{6.4}$$

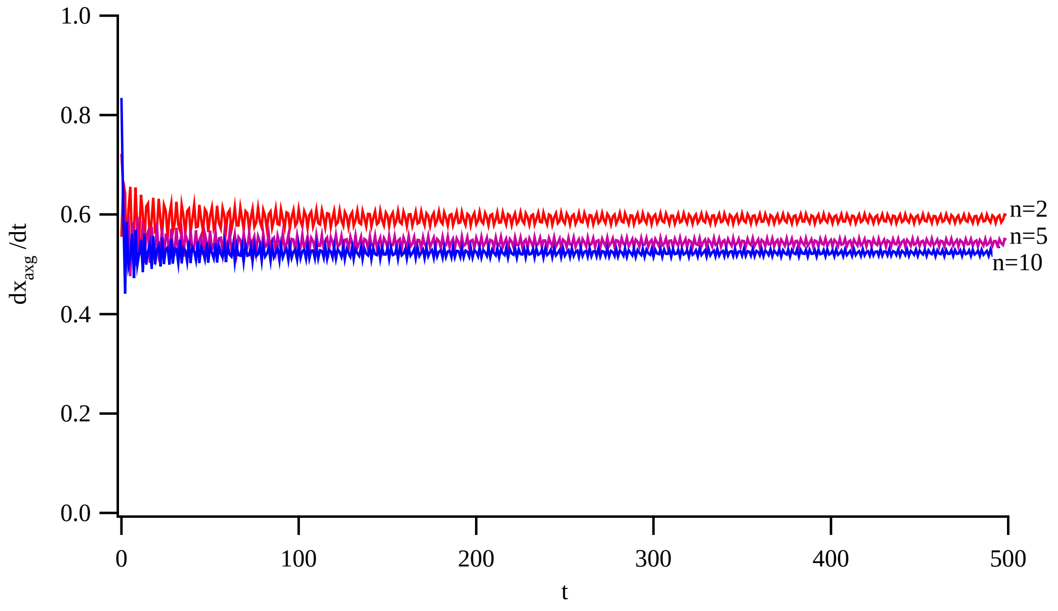


Figure 6.4: $d\langle x\rangle/dt$ as a function of n and T for the directed walk on the line with loops. The descending lines are at greater n.

which if applied would return the state of the walk to $|0\rangle$. Then we may as well specify the inverse discrete Fourier transform of $|\psi\rangle$ by the requirement

$$|j\rangle \rightarrow \frac{1}{\sqrt{n}}\,|0\rangle\,.$$

But the amplitude mapped to the state $|0\rangle$ by the direct discrete Fourier transform is exactly the same. Accordingly on $|\psi\rangle$, both the DFT and its inverse have the same action. Thus after a second time step, the system is returned to the $|0\rangle$ state. To a good approximation, this is exactly the action of the unitary time step operation in our directed quantum walk, except the amplitude associated with the state $|0\rangle$ after the first step is sent on to the next vertex. As $n \rightarrow \infty$, the amplitude associated with $|0\rangle$ in this first step goes to zero, so the approximation becomes precise, and after 2 time steps the particle proceeds forward one step.

The speed of this walk depends highly on the symmetric association of incoming and outgoing edges. If these edges are paired randomly at each vertex, the quantum walk slows down to the classical speed. Accordingly the walk needs to have some local sense of where the solution is in order to proceed faster. This makes the improvement in speed less remarkable, as of course the classical walk could proceed much faster, too, if it knew the way to the solution.

Numerical investigations have shown that similar results seem to hold even when the loops are made into finite dimensions. These are the first results to suggest that quantum walks on directed graphs may yield useful quantum algorithms, and suggest that this approach may have merit, although the speedup we have found is different and perhaps harder to obtain than that with undirected quantum walks.

6.3 Returning to k-SAT

Of course, the graphs from k-SAT problems are not necessarily Eulerian, so we cannot yet run a directed walk on them. Reflecting the difficulty of SAT, there are no nice restrictions on the number of edges in and out of each vertex on graphs corresponding to SAT formulas. Along the lines developed in Chapter 5, a natural first thought is to create some sort of edge Oracle that conditionally fixes the graph by adding in additional directed edges as necessary to make them Eulerian. But such a procedure is not helpful here, because adding edges to fix the graph at each vertex simultaneously would not fix the graph in one step. The fix of adding an new edge to or away from one vertex would throw off the count of in and out edges at the adjacent vertex. Any quantum walk search algorithm needs to be local, as querying the entire graph to fix it up is equivalent to solving the problem directly. Even worse, there is no way for probability to build up at solutions if there are paths leaving them, so the idea of conserving the number of paths in and out of every vertex is doomed at those vertices representing solutions. With this conundrum, we are left to consider the options for non-unitary quantum walk operations.

6.4 General quantum operations

Recall that the statistical ensemble of state vectors $|\psi_i\rangle$ with probabilities p_i is described by a density operator ρ given by

$$\rho = \sum_i p_i \, |\psi_i\rangle\langle\psi_i| \, . \tag{6.5}$$

An ensemble of density operators ρ_i with probabilities p_i is given by $\sum_i p_i\rho_i$. Density operators in general are the set of positive semi-definite matrices with trace 1. Pure states are the set of density operator that can be represented exactly as a single state vector $|\psi\rangle$, that is, if a density operator has one non-zero eigenvalue. States that are not pure are called mixed and represent classical ensembles of pure states. We evolve the state ρ by an operation $\mathcal{E}$ in the usual fashion by the use of a unitary operator U such that

$$\mathcal{E}(\rho) = U\rho U^\dagger. \tag{6.6}$$

These unitary operators classify the full range of quantum transitions in closed systems without measurement.

But this is not the full range of manners in which one quantum state may be mapped onto another. Consider a quantum black box that maps arbitrary quantum systems using physical processes. Inside, it can store any sort of ancillary quantum systems, to which we can apply a joint operation with our system of interest. The black box lets out another quantum system, but it must not necessarily even have the same size as the original system. For example, it could take two qubits as input and throw away one, only outputting the remaining qubit. It turns out that in general we can express any such black box $\mathcal{E}$, termed a general quantum operation or super operator, as

$$\mathcal{E}(\rho) = \sum_i E_i \rho E_i^\dagger, \tag{6.7}$$

where $\{E_i\}$ is any set of complex matrices satisfying

$$\sum_i E_i^\dagger E_i \leq I, \tag{6.8}$$

where the $\leq$ sign indicates that $I - \sum_i E_i^\dagger E_i$ must be a positive-semidefinite operator. For an operator from an n dimensional state to an m dimensional state, only at most mn operator elements E_i are needed to describe it. Also, just as a combination of pure states is not uniquely specified by a density operator, a set of operator elements is not uniquely specified by an quantum operation.

This formalism is most commonly used to rigorously describe the phenomena of decoherence, as quantum systems become classical due to noise. Using non-unitary operators $\mathcal{E}$ in quantum algorithms does not give space for new quantum phenomena, but allows us to make algorithms more classical in a controlled and limited way. This suggests a strategy for extending directed quantum walks to general graphs in a manner in which they may still maintain some of their quantum properties.

It can be shown that Eq. (6.7) and (6.8) are a complete classification of all operators arising from the restrictions imposed by the following set of physical principles [23]:

1. $\operatorname{Tr}\mathcal{E}(\rho) \leq 1$, where the equality is strict unless the process is explicitly dependent upon the results of measurement. This inequality corresponds to the inequality in Eq. (6.8).

2. Applying the operation separately to a set of quantum states which are then mixed together should be equivalent to applying the operation to the states combined first:

$$\mathcal{E}\left(\sum_i p_i \rho_i\right) = \sum_i p_i \mathcal{E}(\rho_i). \tag{6.9}$$

3. If we consider ρ as part of any other quantum state described by a density operator, the final state should still be a density operator. This means that the operator $I \otimes \mathcal{E}$ should also map density operators to density operators, where I is the identity operator on any finite size space.

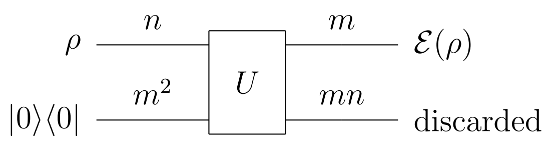


Figure 6.5: General quantum operations as unitary evolution of the system of interest and an environmental system. The numbers above the lines indicate the dimensionality of the quantum systems being manipulated.

When not dependent upon measurement results, these operators can also be described by a unitary operator acting on the combined system. Given a general quantum operation $\mathcal{E}$, we can describe its action by a unitary matrix U acting on $\rho \otimes |0\rangle\langle 0|$, where $|0\rangle\langle 0|$ is a label associated with some environmental system in its initial state, and then discarding all but some m dimensional state. We can explicitly write U acting on the pure state $|0\rangle \otimes |\psi\rangle$ as

$$U = \begin{pmatrix} [E_1] & \dots \\ [E_2] & \dots \\ \vdots & \dots \\ [E_{mn}] & \dots \end{pmatrix}, \tag{6.10}$$

where the E_i are a set of operator element corresponding to the operator and all but the first block column is filled in arbitrarily to make the matrix unitary. Each operator E_i corresponds to a particular basis state $|e_i\rangle$ in the final environmental system. Then discarding this environment corresponds to measuring all states by $|e_i\rangle$, obtaining the states $\frac{1}{\sqrt{p_i}}E_i |\psi\rangle$ each with probability $p_i = \langle\psi| E_i^\dagger E_i |\psi\rangle$. Schematically, this corresponds to the quantum circuit shown in Fig. 6.5. Since the density matrix for this new state is

$$\sum_i E_i |\psi\rangle\langle\psi| E_i^\dagger, \tag{6.11}$$

and we can write any density matrix as a sum of pure states, this procedure corresponds exactly to the operation $\mathcal{E}$.

This picture of general quantum operations is significant for two reasons. First of all, it shows that they can be specified by a unitary operation and adding only at most a certain number of temporary ancillary qubits. It demonstrates that in principle using such a general operation instead of a unitary transformation does not increase the permanent spatial requirements of a quantum algorithm and nor change the order of its run time.

Secondly, this result establishes shows that we can classically simulate quantum algorithms involving general quantum operations without reverting to the density matrix formalism. Simulating a quantum system in the density matrix formalism instead of with state vectors increases the memory requirements by a quadratic factor and increases the calculation requirements just as much. The simulation requirements to exactly calculate transitions of general quantum operators with density matrices could be even worse, since unlike unitary operators acting on independent subspaces, general quantum operations cannot be easily separated. Accordingly applying non-trivial general operations to density operators would involve exponentially larger matrix multiplication when simulating systems like quantum walks. By using quantum Monte-Carlo simulation with state vectors, we can model these algorithms far faster than from exact evaluation using density operators.

6.5 Directed quantum walks on any graph

We can now can write a quantum walk for any directed graph. Writing a single step of the walk as the composition of a shift and coin operation again, we have

$$U = S' \circ C', \tag{6.12}$$

where U is no longer necessarily unitary. The coin C' is of the form for general undirected graphs, but the shift S' must incorporates possibly changing the size of the spaces at each vertices, and thus must include some non-trivial general quantum operation. It also includes an explicit step pairing the possibly size changed amplitudes at a vertex with outgoing edges. In general, we can break down S' into the direct sum of shifts that happen at each vertex. At each vertex, there are n edges in and m edges out, leaving three possible cases. If $n = m$, then we can apply straightforward permutation, a unitary operation. If $n < m$, then we can also apply permutation, by classically choosing the vertices to be shifted onto at random. The hard case is for $n > m$, because then we need to somehow over-compress quantum states onto a smaller space. In general, we know this is not possible with perfect fidelity for arbitrary pure states, because quantum operations must be linear, and there are no invertible linear maps from a space of higher to lower dimensionality. At best, we can hope to compress the state in such a way that only some minimal amount of information is lost.

We now consider such choices of "over-compressing" operations in the case that there are more paths in n than out m. One of the easiest such operators to devise is to create an equal classical superposition of all outgoing states each time it is applied. This operation has operator elements given by

$$E_{ij} = \frac{1}{\sqrt{m}} |j\rangle\langle i| . \tag{6.13}$$

where $1 \leq i \leq n$ and $1 \leq j \leq m$. We can easily verify

$$\sum_{i,j} E_{ij}^{\dagger} E_{ij} = \frac{1}{m} \sum_{i,j} |i\rangle\langle j|j\rangle\langle i| = \sum_{i=1}^{n} |i\rangle\langle i| = I, \tag{6.14}$$

as required by Eq. (6.8). Then its action on a state ρ is given by

$$\begin{aligned}\sum_{i,j} E_{ij}\rho E_{ij}^\dagger &= \frac{1}{m}\sum_{i,j} |j\rangle\langle i|\rho|i\rangle\langle j| &(6.15)\\ &= \frac{1}{m}\sum_{i=1}^{n}\langle i|\rho|i\rangle \sum_{j=1}^{m}|j\rangle\langle j| &(6.16)\\ &= \frac{1}{m}\operatorname{Tr}(\rho) I &(6.17)\\ &= I/m, &(6.18)\end{aligned}$$

which is exactly the classical superposition of the states $|j\rangle$ with probability $1/m$ each. But classical superpositions do not provide room for quantum interference, so this is a poor candidate for our shift operation.

A more sophisticated transition operator picks m input states at random and maps them to output states, discarding $n-m$ states. This operator is given by operator elements

$$E_i = \frac{1}{\sqrt{\frac{m}{n}\binom{n}{m}}}\sum_{j=1}^{m}|j\rangle\langle i_j|\,, \tag{6.19}$$

where $|i_j\rangle$ indicates the jth basis vector from the ith choice of m out of n basis vectors for the input space. We can verify

$$\sum_{i\in C} E_i^\dagger E_i = \frac{n/m}{\binom{n}{m}}\sum_{i\in C}\sum_{j=1}^{m}|i_j\rangle\langle j|j\rangle\langle i_j| = \frac{n/m}{\binom{n}{m}}\sum_{i\in C}\sum_{j=1}^{m}|i_j\rangle\langle i_j| = I, \tag{6.20}$$

where C is the set of m choices of n, as on average each sum $\sum_{j=1}^{m}|i_j\rangle\langle i_j|$ contributes $\frac{m}{n}I$ and there are $\binom{n}{m}$ choices, so this is a general quantum operation. To get some limited sense of how this operator acts, consider its action reducing an arbitrary state in 3 dimensions to 2 dimensions,

$$\begin{pmatrix} a & b & c\\ e & f & g\\ h & j & k\end{pmatrix} \to \begin{pmatrix} a+\frac{1}{2}f & \frac{1}{2}(b+c+g)\\ \frac{1}{2}(e+h+j) & \frac{1}{2}f+k\end{pmatrix}. \tag{6.21}$$

In some sense the matrix elements of this resulting state look similar to the original. More broadly, it is hard to understand what this operator does, but at least its transitions clearly remain significantly quantum mechanical.

6.6 Quantum directed walks on 2-SAT

In the full procedure, we start the directed quantum walk in an initial state at an equal superposition of being at each vertex. Then we use a standard coin operator C' and a shift operator including one of the over-compression operators as given by Eq. (6.13) or Eq. (6.19). As we have no heuristic to pair in and out edges at any vertex, we resort choosing random pairings with the shift operation.

Unfortunately, when averaged over a sufficiently large number of runs, numerical results show that the average probability of any quantum walk converging to a solution after running the walk for any time t is exactly equal to the probability from simulating the classical walk with a Markov chain. Although the first transition operator is almost entirely useless, keeping the pairing between in and out edges at some random constant assignment over the course of quantum walk yields varied results at least for the second operator. But on average for every 2-SAT formula the performance of the classical and quantum walks are identically the same. By pairing in and out vertices randomly, we seem to have averaged over all quantum interference effects such that they are entirely unnoticeable.

6.7 Conclusions

Quantum random walks provide a simple model of quantum systems that emphasize the profound power of quantum interference. Constructive interference allows probability to build up faster in quantum walks than can be done classically. Intuitively, it is clear that in order for interference to build up in a manner that reflects the symmetry of a particular problem, the quantum walk itself must as much as possible reflect that symmetry. The examples of quantum walks we have discussed that are successfully faster than their classical versions respect most of the symmetry of the underlying graph obtained from the classical version, and this is an observation also consistent more generally with the most powerful quantum walks [18]. Our failure to find a way to respect symmetry when approaching SAT with directed quantum walks explains why these algorithms were not successful.

In our work with directed walks, we may also have been moving too fast in our attempts to solve satisfiability problems. We provided the first demonstration of the power of directed quantum walks (Sec. 6.2) and then immediately attempted to solve the very hard problem of satisfiability. Careful intermediate work would warranted, using directed quantum walks to solve more toy problems with intrinsically more accessible symmetry. A good starting point for further investigations would be to attempt to find more examples of useful quantum versions of directed walks on the hypercube or other structures with similarities to the graphs used in satisfiability problems, in a manner that is not as ambitious as attacking SAT directly. As it turns out, satisfiability problems may simply not have sufficient symmetry to be exploited by quantum walks, but further investigations toward these conclusions could still yield other useful insights.

In fact, hoping to spur these other insights was always the primary motivation for this work, as ironically a quantum algorithm for 3-SAT better than any classical algorithm already exists. It is essentially the best probabilistic algorithm for 3-SAT plugged directly into Grover's algorithm, which can make any algorithm that succeeds with bounded probability run in the square root of the time [6]. One might wonder in that case why we even looked for an improvement that could be expected at best to only match that performance, but even solving 3-SAT is only relevant because of its connection to a bigger classes of problems. The most important aspect of this work is the guidance it yields toward future quantum random walk based algorithms for problems that may not have such easy shortcuts, and, in a pinch, that guidance could be summarized in one word and essential concept: symmetry.

Acknowledgments

First and foremost, I would like to thank my research adviser last summer, David Meyer, under whom I performed most of this research. He introduced me to quantum computing and quantum walks, and was extremely helpful in guiding my research and providing feedback on this thesis.

I had a tremendously fun summer doing physics at UCSD thanks to my companions in the REU program. The comments and feedback from the fellow students in David's group were also hugely helpful, especially from Yi-Kai Liu who gave me some useful pointers on satisfiability problems. I would also like to thank my friend Eric Christiansen for convincing me to program in Python and for his (futile) attempts to understand quantum mechanics.

I am indebted to the UCSD Mathematics department Computing Support, which lent me a laptop when my own was stolen three weeks before the end of my work. For that matter, I can also thank the coincidence that I backed up all my work to a server only two days before.

Finally, I am grateful to my thesis readers at Swarthmore, John Boccio and Amy Bug, for reading this thesis with a critical eye even though it was outside their specialty, and also the entire physics department at Swarthmore for giving me a fantastic education.

This research was funded mostly by the NSF under the 2007 REU program at the University of California San Diego.

References

[1] Scott Aaronson. The limits of quantum computers. *Scientific American*, 298(3):62, Mar 2008.

[2] Scott Aaronson and Andris Ambainis. Quantum search of spatial regions. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science*, page 200, 2003.

[3] Dorit Aharonov, Andris Ambainis, Julia Kempe, and Umesh Vazirani. Quantum walks on graphs. In *STOC '01: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 50–59, July 2001, arXiv:quant-ph/0012090.

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

[5] Andris Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1:507–518, 2003, arXiv:quant-ph/0403120.

[6] Andris Ambainis. Quantum search algorithms. *SIGACT News*, 35(2):22–35, 2004, arXiv:quant-ph/0504012.

[7] Andris Ambainis, Eric Bach, Ashwin Nayak, Ashvin Vishwanath, and John Watrous. One-dimensional quantum walks. In *STOC '01: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 37–49, New York, NY, USA, 2001.

[8] Paul Benioff. Space searches with a quantum robot. *AMS Contemporary Math Series, Vol*, 305, 2002, arXiv:quant-ph/0003006.

[9] Bla Bollobs, Christian Borgs, Jennifer T. Chayes, Jeong Han Kim, and David B. Wilson. The scaling window of the 2-SAT transition. *Random Structures and Algorithms*, 18(3):201–256, 2001, arXiv:math/9909031.

[10] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. Exponential algorithmic speedup by quantum walk. In *STOC '03: Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 59–68, 2003, arXiv:quant-ph/0209131.

[11] Andrew M. Childs and Jeffrey Goldstone. Spatial search by quantum walk. *Phys. Rev. A*, 70:022314, 2004, arXiv:quant-ph/0306054.

[12] D. Deutch. Quantum theory, the Church-Turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London*, volume 400 of *A, Mathematical and Physical Sciences*, pages 97–117, 1985.

[13] Richard P. Feynman. Simulating physics with computers. *International Journal of Thoeretical Physics*, 21(6/7):467–488, Jun 1982.

[14] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC '96: Proceedings of the 28th annual ACM Symposium on the Theory of Computing*, pages 212–219, May 1996, arXiv:quant-ph/9605043.

[15] Tad Hogg. Solving random satisfiability problems with quantum computers. 2001, arXiv:quant-ph/0104048.

[16] Julia Kempe. Quantum random walks - an introductory overview. *Contemporary Physics*, 44(4):307–327, 2003, arXiv:quant-ph/0303081.

[17] Viv Kendon. Decoherence in quantum walks - a review. *Mathematical Structures in Computer Science*, 17(6):1169–1220, Dec 2007, arXiv:quant-ph/0606016.

[18] Hari Krovi and Todd A. Brun. Hitting time for quantum walks on the hypercube. *Phys. Rev. A*, 73:032341, 2006, arXiv:quant-ph/0510136.

[19] David A. Meyer. From quantum cellular automata to quantum lattice gases. *J. Stat. Phys.*, 85:551–574, 1996, arXiv:quant-ph/9604003.

[20] Ashley Montanaro. Quantum walks on directed graphs. *Quantum Information and Computation*, 7(1), 2007, arXiv:quant-ph/0504116.

[21] Cristopher Moore and Alexander Russell. Quantum walks on the hypercube. In *RANDOM '02: Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, pages 164–178, London, UK, 2002. Springer-Verlag, arXiv:quant-ph/0104137.

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

[23] Michael A. Nielsen and Isaac L. Chaung. *Quantum Computation and Quantum Information.* Cambridge University Press, Cambridge, UK, 2000.

[24] C.H. Papadimitriou. On selecting a satisfying truth assignment. In *FOCS '91: Proceedings of the 32nd Annual IEEE Symposium on the Foundations of Computer Science*, pages 163–169, October 1991.

[25] Robin Pemantle, Rajarshi Das, and Torin Greenwood. Quantum random walks archive. Available online at `http://www.math.upenn.edu/~pemantle/Summer2007/Archive.html`, Aug 2007.

[26] Uwe Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *FOCS '99: Proceedings of the 40th Annual IEEE Symposium on the Foundations of Computer Science*, pages 410–414, October 1999.

[27] Simone Severini. On the digraph of a unitary matrix. *SIAM J. Matrix Anal. Appl.*, 25(1):295–300, 2003, arXiv:math/0205187.

[28] Neil Shenvi, Julia Kempe, and K. Birgitta Whaley. Quantum random-walk search algorithm. *Phys. Rev. A*, 67(5):052307, May 2003, arXiv:quant-ph/0210064.

[29] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. In *FOCS '94: Proceedings of the 35th Annual IEEE Symposium the on Foundations of Computer Science*, pages 124–134, 1994, arXiv:quant-ph/9508027.