Some Upper and Lower Bounds regarding Query Complexity

by

Rui Peng Liu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics & Optimization

Waterloo, Ontario, Canada, 2017

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Query complexity is one of the several notions of complexity defined to measure the cost of algorithms. It plays an important role in demonstrating the quantum advantage, that quantum computing is faster than classical computation in solving some problems. Kempe showed that a discrete time quantum walk on hypercube hits the antipodal point with exponentially fewer queries than a simple random walk [K05]. Childs et al. showed that a continuous time quantum walk on "Glued Trees" detects the label of a special vertex with exponentially fewer queries than any classical algorithm [CCD03], and the result translates to discrete time quantum walk by an efficient simulation.

Building on these works, we examine the query complexity of variations of the hypercube and Glued Tree problems. We first show the gap between quantum and classical query algorithms for a modified hypercube problem is at most polynomial. We then strengthen the query complexity gap for the Glued Tree label detection problem by improving a classical lower bound technique; and we prove such a lower bound is nearly tight by giving a classical query algorithm whose query complexity matches the lower bound, up to a polylog factor.

Acknowledgements

I would like to thank my supervisor, Dr. Ashwin Nayak, for his continuous support and insights. Without his guidance, this work would not have been possible.

I would also like to thank Dr. Michele Mosca and Dr. Jon Yard for agreeing to be the readers of this thesis.

Dedication

To my parents.
Thank you for all of your support along the way.

Table of Contents

List of Figures

List of Symbols

$\mathbb{N}$ the set of natural numbers, i.e., $\{1, 2, 3, \ldots\}$

$[n]$ the set of natural numbers bounded above by n, i.e., $\{1, 2, 3, \ldots, n\}$

$\aleph_0$ the cardinality of $\mathbb{N}$, i.e., $|\mathbb{N}|$

$\mathbb{Z}_2$ the unique finite field of 2 elements, namely $\{0, 1\}$

$\mathbb{Z}_2^n$ the n-dimensional vector space over $\mathbb{Z}_2$

$\mathbb{R}_+$ the set of nonnegative real numbers

$\mathbb{R}_{++}$ the set of positive real numbers

$O(\cdot)$ big O notation, $f \in O(g)$ if $f(n) \leq cg(n)$ for functions $f, g : \mathbb{N} \to \mathbb{N}$, positive constant c, and n large enough

$o(\cdot)$ little o notation, $f(x) \in o(g(x))$ if $\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$ for functions f, g

$\Omega(\cdot)$ big Omega notation, $f \in \Omega(g)$ if $f(n) \geq cg(n)$ for functions $f, g : \mathbb{N} \to \mathbb{N}$, positive constant c, and n large enough

$\Theta(\cdot)$ big Theta notation, $f \in \Theta(g)$ if $f \in O(g)$ and $f \in \Omega(g)$ for functions $f, g : \mathbb{N} \to \mathbb{N}$

Chapter 1

Introduction

Ever since the idea of quantum computation was conceived in the 80s, researchers have been active in demonstrating the quantum advantage, that quantum computation is much faster than any classical computation in solving some problems. The computations were abstracted into algorithms, and we care about the complexity of those algorithms, where complexity measures the resources used by the algorithms.

Several complexity classes were defined, and query complexity is arguably one of the most successful models, for it has been used to show quantum computation may achieve an exponential speed-up over classical computation. For example, Simon's problem is the first theoretical problem ever studied that exhibits an exponential query complexity gap; indeed, any classical algorithm solving Simon's problem must make $\Omega(2^{n/2})$ queries, where the best quantum algorithm solving it makes only $\Theta(n)$ queries. Following the work of Simon, Shor presented the integer factorization algorithm, demonstrating a potential exponential time complexity gap for a practical problem the first time.

However, both problems are based on the quantum Fourier transform; in fact, most of the quantum algorithms designed in the 90s have some flavour of quantum Fourier transform. It made people wonder whether there is another type of quantum algorithm that does not invoke quantum Fourier transform at all. Since quantum algorithms, in some sense, are generalizations of randomized algorithms, researchers cast their eyes onto random walks. As a quantization of random walk, quantum walk was defined and heavily studied[1].

Kempe showed in [K05] that on the hypercube, a discrete time quantum walk reaches the antipodal point in $O(n)$ queries, which is exponentially faster than $\Omega(2^n)$, the expected

[1]Ironically, quantum Fourier transform plays an important role in quantum walk type algorithms.

hitting time of simple random walk on the hypercube of dimension n. However, there is a $O(n^3)$ deterministic algorithm solving the same problem. Later in [CCD03], Childs, Cleve, Deotto, Farhi, Gutmann and Spielman presented the Glued Tree label detection problem with an oracle of input size N, and showed that any classical algorithm solving the problem uses at least $\Omega(N^{1/6})$ queries, which is exponentially slower than a continuous time quantum walk, though the precise classical query complexity was left open. They also showed that the continuous time quantum walk can be simulated by a discrete time quantum walk with a polylog overhead (see also [CGM09] and [C09]), thus the result for the Glued Tree label detection problem also applies to some discrete time quantum walk.

Is it possible to modify those problems so that the variations exhibit query complexity gaps that are larger than the largest gap known so far? We first define a class of graphs with the hypercube being a special case, then we reason why such generalization may slow down the most efficient quantum query algorithms. We also adopt similar ideas from the algorithm solving the hypercube problem to show there is an efficient deterministic query algorithm solving the modified hypercube problem, hence the query complexity gap for this class of problems is at most polynomial in n, the dimension of the hypercube.

Next, we show for the Glued Tree problem, no classical algorithm would use fewer than $\Omega(N/\log(N))$ queries even if N, the input size of the given oracle, is reduced down to the number of nodes on the graph; thus, we may redefine the input size of the oracle, which strengthens the query complexity gap from $N^{1/6}/\text{polylog}(N)$ to $N^{1/2}/\text{polylog}(N)$. We then present a $O(N\ \log N)$ deterministic algorithm whose complexity matches the $\Omega(N/\log N)$ lower bound of the classical query complexity, up to a log factor, thereby proving the classical query complexity is nearly tight.

1.1 Outline of this Thesis

In chapter 2, we review some relevant topics on algorithms and random walks. We start with a summary of different types of algorithms, then introduce the oracle setting and query complexity, which is the main theme of this thesis. We then proceed to build the knowledge of random walk on graphs; starting with elementary probability, we develop the notions of markov chains, random walks on line and the hypercube, and we end the chapter with a discussion on hitting time.

In chapter 3, we turn the hypercube hitting time problem into a query problem, namely a label detection problem, which can be solved by an efficient quantum query algorithm. We introduce a class of related problems and show there is an efficient deterministic query algorithm for all of them.

In chapter 4, we modify the Glued Tree label detection problem. By improving a classical lower bound technique, we may shrink the input size of the problem by a polynomial factor, thus strengthening the query complexity gap by a polynomial factor. We also come up with a classical query algorithm in which the query complexity matches the lower bound of the classical query complexity of the problem, thereby proving the query complexity of the problem is tight, up to a factor logarithmic in the input size.

In chapter 5, we summarize our results, discuss ongoing research on query complexity, and end this thesis with open problems.

Chapter 2

Preliminaries

2.1 Algorithm

2.1.1 Introduction to Algorithms

Let us begin with a review of the concept of algorithm. We are mainly concerned with randomized algorithms and deterministic algorithms in this thesis, but we also make some reference to quantum algorithms. As opposed to quantum algorithms, randomized algorithm and deterministic algorithms are sometimes referred to as *classical* algorithms.

According to the textbook [CLR09], an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

We can also view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relationship. The algorithm describes a specific computational procedure for achieving the input/output relationship.

We are interested in the following three types of algorithms,

1. Deterministic Algorithm

   A deterministic algorithm always returns the same output for a fixed input. Given a problem, we say a deterministic algorithm solves the problem if the algorithm always produces the correct output. (i.e., one that satisfies the input-output relation)

2. Randomized Algorithm

   A randomized algorithm makes random choices in intermediate steps according to some probability distribution. As a consequence, the output needs not always be the same. Note that deterministic algorithm is a special case of randomized case, since we may set the probability distributions to have values 1 and 0 only.

   Given a problem, we say a randomized algorithm solves the problem if for each input, the algorithm produces the correct output with high probability, i.e., probability at least 2/3. (Note this definition is not exact, but it is good enough for the purpose of this thesis, the reader is referred to [MR95] for a thorough treatment of the topic).

3. Quantum Algorithm

   A quantum algorithm takes advantage of quantum mechanics, namely exploiting the entanglement of states and interference phenomenon. Note quantum algorithm reduces to a randomized algorithm if there is no interference at all. The reader is referred to [KLM07] for a formal treatment of topics on quantum computing.

Search algorithm is one of the most famous and ubiquitous algorithms, below is a common problem solved by search algorithms.

Example 1 (Search Algorithm)**.** *Consider a subset of a shuffled deck of* 52 *playing cards. We want to find "ace of spades" among the cards if it is in the subset. Until we find the card, we may either search for it from the top to the bottom, or draw cards randomly. The former is a deterministic algorithm, because for a fixed subset, we always find "ace of spades" in the same location, if it is in the subset; the latter is a randomized algorithm, because the number of draws needed depends greatly on how "lucky" we are.*

2.1.2 Query Model

Suppose the input x of the problem is given in a black box (also called an oracle), and we may access the input only by querying the oracle O. The input to O can be an index, a label, etc., and we learn something about x when we feed some input into O.

Formally, suppose we want to evaluate $f(x)$ for some function $f : \{0,1\}^n \to \{0,1\}$, $n \in \mathbb{Z}$, and input $x = (x_i)_{i\in[n]} \in \{0,1\}^n$. Given the oracle $O : [n] \to \{0,1\}$ by $O(i) = x_i$, we may learn about i-th bit of x by querying $O(i)$.

Definition 2. *The query complexity of an algorithm is a function $c : \mathbb{Z} \to \mathbb{Z}$ such that $c(n)$ is the maximum number of queries made by the algorithm on input of a fixed size n.*

The classical (or quantum, resp.) query complexity of a problem is the infimum of query complexity of all classical (or quantum, resp.) algorithms solving the problem. We further define the query complexity gap to be the ratio of quantum query complexity versus classical query complexity.

Note we do not count the intermediate steps between the queries, hence the algorithm is allowed to take a huge number of steps, as long as it makes a minimum number of queries.

Let N denote the total number of inputs of an oracle, then we say an algorithm is efficient if the query complexity is $O(polylog(N))$.

Example 3 (OR Function)**.** *Consider the function*

$$f : \mathbb{Z}_2^N \to \{0, 1\}$$

$$f(x) = \vee_{i\in[N]}\ x_i = \begin{cases} 0 & x = 000\ldots0 \\ 1 & otherwise \end{cases}$$

Any search problem can be modelled as evaluating an OR function. Indeed, suppose we want to search for some "marked" elements among N elements, and we label the elements from 1 to N, then it corresponds to evaluating f with the input x, where

$$x_i = \begin{cases} 1 & element\ i\ is\ marked \\ 0 & otherwise \end{cases}.$$

Associated with the problem is an oracle O that takes indices $i \in [N]$ as input, then returns $O(i) = x_i$ as output. In order to evaluate f, it is not always necessary to read every index of x; for example, say $x = 100\ldots0$, then we may query only the first index to learn $x \neq 000\ldots0$, and produce the output $f(x) = 1$ in just 1 query.

It can be shown that any classical algorithm solving this problem must make at least $\Omega(N)$ queries, and any quantum algorithm solving this problem must make at least $\Omega(\sqrt{N})$ queries. Since there is a $O(\sqrt{N})$ quantum query algorithm, the query complexity gap is $\sqrt{N}$ versus N, a quadratic separation.

We end this section with a proof of the classical lower bound of evaluating the OR function. The quantum lower bound can be found in [BBB98]. The reader may need to read the next section for topics on probability first.

Lemma 4. *Any classical algorithm that solves the OR function f on N bits makes $\Omega(N)$ queries.*

Proof. We make use of Yao's Principle (more details can be found in Section 2.2.2. of [MR95]). In short, Yao's Principle states that the average cost of a randomized algorithm is bounded below by the expected cost of any deterministic algorithm that solves the problem with probability at least $2/3$ under an input distribution, up to constant factor. In our case, the cost is the number of queries.

To apply the principle, we only need to provide a distribution of inputs that is hard for the deterministic algorithms, i.e., under such distribution of inputs, any deterministic algorithm needs to make many queries in order to produce the correct result $2/3$ of the time.

Let X be a random variable of inputs, and consider the distribution

$$P(X = x) = \begin{cases} 1/2 & x = 000\ldots0 \\ 1/(2N) & x \text{ has exactly one 1-entry} \\ 0 & \text{otherwise} \end{cases}.$$

By rearranging the indices, we assume WLOG that a deterministic algorithm queries the indices in the order $1, 2, 3, \ldots$. When the algorithm makes $t = o(N)$ queries, the probability the algorithm has discovered a 1-entry is $t/N \to 0$ as $N \to \infty$.

Moreover, given that the algorithm has not discovered a 1-entry, the probability $f(X) = 1$ is

$$\frac{N - t}{2N(1 - t/N)} \to 1/2 \text{ as } N \to \infty.$$

Hence, we cannot be correct about the value of $f(X)$ $2/3$ of the time when $t = o(N)$, so any algorithm that produces the correct output $2/3$ of the time under this input distribution must make $\Omega(N)$ queries. □

2.2 Random Walk

Over the past several decades, researchers gradually realized the importance of probability theory in designing randomized algorithms. Being an essential part of the probability theory, random walk has been closely studied and employed in randomized algorithms.

In this section, we give a brief summary of the concepts involved in this thesis. We begin with a quick overview of elementary probability, then move onto Markov chains and random walk on graphs, and finish this section with a discussion on hitting times. One can learn more about topics on random walk in [B98], [L06], [R09], and [LPW08].

2.2.1 Quick Review of Elementary Probability

We first recall some key terms in probability theory; a detailed explanation can be found in [R09].

Let Σ denote a collection of events. Associated with Σ is a probability measure, a set function, $P : \Sigma \to [0, 1]$ such that $\forall$ events $E \in \Sigma$, we have

1. $0 \leq P(E) \leq 1$
2. $P(S) = 1$
3. For any sequence of mutually exclusive events $E_1, E_2, \ldots \in \Sigma$,

$$P(\cup_{n=1}^{\infty} E_n) = \sum_{n=1}^{\infty} P(E_n).$$

We say two events E and F are independent if $P(E \cap F) = P(E)P(F)$, where $E \cap F$ denotes the events that both E and F happen.

Sometimes, we are interested in the conditional probability, i.e., given event E happens, what is the probability that event F happens. Denote such event by $F \mid E$, then it satisfies the equation

$$P(F \mid E) = \frac{P(E \cap F)}{P(E)}.$$

The events are often described in terms of random variables. A random variable is a function $X : \Sigma \to S \subseteq \mathbb{R}$, and the events are of the form $X = x$ for some $x \in S$.

The expected value of a random variable X is a linear function such that

$$E(X) = \sum_{x \in S} xP(X = x).$$

Suppose $S \subseteq \mathbb{R}$, then for $t \in \mathbb{R}_{++}$, the expected value of $|X|$ is related to the tail probability $P(|X| \geq t) = \sum_{s \geq t} P(|X| = s)$ by the Markov Inequality,

$$P(|X| \geq t) \leq \frac{E(|X|)}{t}.$$

If X is a random variable whose range is a subset of $\mathbb{R}_+$, then the Markov Inequality is simply

$$P(X \geq t) \leq \frac{E(X)}{t}.$$

Say two random variables X and Y are independent if the events $X \leq t_1$ and $Y \leq t_2$ are independent $\forall t_1, t_2 \in \mathbb{R}$, then X and Y satisfies

$$E(XY) = E(X)E(Y).$$

2.2.2 Markov Chain

The type of random walk we are interested in can be captured by Markov chain, a model that describes how a system changes over time, and Markov chain is a special type of stochastic process. This section is a summary of the first three chapters of [L06].

Definition 5. *A stochastic process is a collection of random variables X_t indexed by time. In this thesis, we only consider discrete-time stochastic process, i.e., time will always be a subset of the nonnegative integers $\{0, 1, 2, \ldots\}$.*

The random variables X_t take values in a set that we call the state space. Again, we only consider discrete state space which can be either finite or countable, hence the state space is isomorphic to some subset of $\mathbb{N}$.

Given a discrete-time stochastic process, $X_t, t = 0, 1, 2, \ldots$, where X_t takes values in some set $S \subseteq \mathbb{N}, |S| = N \in \mathbb{N} \cup \{\aleph_0\}$. We call the possible values for X_t the states of the system. We define the probability measure P and let

$$P(X_0 = i_0, X_1 = i_1, \ldots, X_n = i_n)$$

denote the probability of the event ($X_0 = i_0$ and $X_1 = i_1$ and $\ldots$ and $X_n = i_n$). Equivalently, we could give the initial probability distribution

$$\phi^0 \in \mathbb{R}^N, \qquad \phi_i^0 = P(X_0 = i), i = 1, \ldots, N,$$

and the transition probabilities

$$q_n(i_n \mid i_0, \ldots, i_{n-1}) = P(X_n = i_n \mid X_0 = i_0, \ldots, X_{n-1} = i_{n-1}),$$

where $P(X_n = i_n \mid X_0 = i_0, \ldots, X_{n-1} = i_{n-1})$ denotes the probability of the event $X_n = i_n$ given the event ($X_0 = i_0$ and $X_1 = i_1$ and $\ldots$ and $X_{n-1} = i_{n-1}$). Now, it follows that

$$P(X_0 = i_0, \ldots, X_n = i_n) = \phi_{i_0}^0 q_1(i_1 \mid i_0) q_2(i_2 \mid i_0, i_1) \cdots q_n(i_n \mid i_0, \ldots, i_{n-1}).$$

The probability measure satisfies the Markov property if the present state of the system is all that is needed to make predictions of the behavior of a system in the future, i.e., the past history is irrelevant. Formally, $\forall i_0, i_1, \ldots, i_n \in S$,

$$P(X_n = i_n \mid X_0 = i_0, \ldots, X_{n-1} = i_{n-1}) = P(X_n = i_n \mid X_{n-1} = i_{n-1}).$$

Definition 6. *A Markov Chain is a stochastic process with the Markov property.*

In this thesis, all Markov Chains are time-homogeneous, i.e., transition probabilities do not depend on time. Formally,

$$q_n(i_n \mid i_0, \ldots, i_{n-1}) = P(X_n = i_n \mid X_0 = i_0, \ldots, X_{n-1} = i_{n-1}) = p(i_{n-1}, i_n),$$

then for this time-homogeneous Markov Chain, we may define a $N \times N$ transition matrix T such that $T_{ij} = p(i, j)$ for all entries (i, j). The matrix T is a stochastic matrix, i.e.,

$$0 \leq T_{ij} \leq 1, \qquad 1 \leq i, j \leq N$$

$$\sum_{j=1}^{N} T_{ij} = 1, \qquad 1 \leq i \leq N.$$

On the other hand, any matrix satisfying the two conditions can be the transition matrix for a Markov Chain.

Note that for a time-homogeneous Markov Chain,

$$\begin{aligned} P(X_0 = i_0, \ldots, X_n = i_n) &= \phi^0_{i_0} q_1(i_1 \mid i_0) q_2(i_2 \mid i_0, i_1) \cdots q_n(i_n \mid i_0, \ldots, i_{n-1}) \\ &= \phi^0_{i_0} p(i_0, i_1) p(i_1, i_2) \cdots p(i_{n-1}, i_n) \\ &= \phi^0_{i_0} T_{i_0 i_1} T_{i_1 i_2} \cdots T_{i_{n-1} i_n}. \end{aligned}$$

If we let $\phi^n \in \mathbb{R}^N$ be the vector such that $\phi^n_i = P(X_n = i)$, then

$$\begin{aligned} \phi^n &= (P(X_n = j))_j \\ &= (\sum_{i_0 \in [N]} \sum_{i_1 \in [N]} \cdots \sum_{i_{n-1} \in [N]} P(X_0 = i_0, \ldots, X_{n-1} = i_{n-1}, X_n = j))_j \\ &= (\sum_{i_0 \in [N]} \sum_{i_1 \in [N]} \cdots \sum_{i_{n-1} \in [N]} \phi^0 T_{i_0 i_1} T_{i_1 i_2} \cdots T_{i_{n-1} j})_j \\ &= \phi^0 T^n, \end{aligned}$$

which gives the probability distribution of the states of the system in n steps.

Below is a simple example of Markov Chain:

Example 7 (Two-state Markov Chain)**.** *Suppose the state of a phone is modelled by $X_t = 0$ (the phone is free at time t) and $X_t = 1$ (the phone is busy). We assume that during each time interval there is a probability p that a call comes in (for ease we will assume that no more than one call comes in during any particular time interval). If the phone is busy during that period, the incoming call does not get through. We also assume that if the phone is busy during a time interval, there is a probability q that it will be free during the next interval. Our model gives a Markov Chain with state space $S = \{0, 1\}$ and matrix*

$$T = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 1-p & p \\ q & 1-q \end{pmatrix} \end{array}$$

This matrix gives the general form for a transition matrix of a two-state Markov Chain. In order to specify the matrix one only needs to give the values of p and q.

2.2.3 Random Walk on Graphs

In this section, we illustrate two important random walks on graphs that are related to the next two chapters, namely random walk on line and simple random walk on hypercube. We first begin with an introduction to the notion of graph.

Definition 8. *A graph G is an ordered pair of sets (V, E) such that the edges E is a subset of the set of unordered pairs of the nodes V, where V is non-empty. We say $u, v \in V$ are neighbours of each other if $\{u, v\} \in E$. Denote $\mathcal{N}(u) = \{v \in V : \{u, v\} \in E\}$ to be the set of neighbours of $u \in V$.*

A finite graph is a graph where both V and E are finite; similarly for infinite graphs. In this thesis, we consider both finite and infinite graphs.

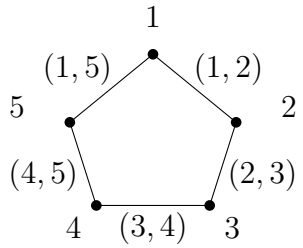
An automorphism of a graph $G = (V, E)$ is a bijective mapping $\tau : V \to V$ such that $\forall u, v \in V, \{u, v\} \in E \iff \{\tau(u), \tau(v)\} \in E$.

Example 9. *The finite graph $C_5 = (V, E)$ is given by*

$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{1, 5\}\}$$

Below is one representation of C_5



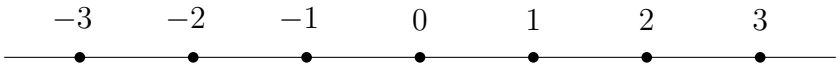
C_5 has many automorphisms, $\tau(i) = (i \mod 5) + 1$ is one of them.

Graphs are ubiquitous; one can easily model transportation problems, scheduling problems, etc. as graph problems. An interesting example is to model a maze as a graph; we start at the ENTRANCE, choose direction according to some probability distribution at each intersection, and how long does it take to reach the EXIT? This interesting case belongs to a huge class of process, namely random walk on graphs.

Random walk on graphs are merely time-homogeneous Markov Chains with the nodes being the states, and the associated transition matrix depends on the edges[1]. Let us illustrate the basic idea with two examples that are related to the following chapters.

[1]In fact, every reversible finite discrete-time Markov Chain can be represented as a random walk on some graph with self-loops allowed.

Example 10 (Random Walk on Line)**.** *A random walk on $\mathbb{Z}$ is a walk on the infinite graph $G = (\mathbb{Z}, E)$, where $\forall u, v \in \mathbb{Z}, (u, v) \in E \iff |u - v| = 1$*



The transition matrix T is given by

$$T_{uv} = \begin{cases} p & v = u + 1 \\ 1 - p & v = u - 1 \\ 0 & \textit{otherwise} \end{cases}$$

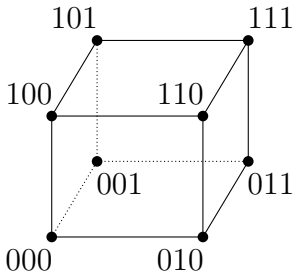
for some $p \in [0, 1]$, and $\forall u, v \in \mathbb{Z}$. Note that

$$0 \leq T_{uv} \leq 1, \qquad u, v \in V$$

$$\sum_{v \in V} T_{uv} = T_{u(u-1)} + T_{u(u+1)} = (1 - p) + p = 1, \qquad \forall u \in V,$$

so T is indeed a stochastic matrix.

Example 11 (Simple Random Walk on Hypercube)**.** *A hypercube is a finite graph $H^{(n)} = (\mathbb{Z}_2^n, E)$ such that $\forall u, v \in \mathbb{Z}_2^n, (u, v) \in E \iff h(u \oplus v) = 1$, where h is the Hamming weight*[2]*.*



The hypercube $H^{(3)} = (\mathbb{Z}_2^3, E)$

[2]For a given $u \in \mathbb{Z}_2^n$, $h(u)$ is the number of 1's in u, e.g., $h(10010) = 2$

For a simple random walk on the graph, the walk always moves to one of its neighbours uniformly at random. In the case of $H^{(3)}$, that translates to the transition matrix

$$T = \begin{array}{c} \\ 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{array} \begin{array}{c} \begin{array}{cccccccc} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \end{array} \\ \begin{pmatrix} 0 & 1/3 & 1/3 & 0 & 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 \\ 1/3 & 0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 1/3 & 1/3 & 0 & 0 & 0 & 0 & 1/3 \\ 1/3 & 0 & 0 & 0 & 0 & 1/3 & 1/3 & 0 \\ 0 & 1/3 & 0 & 0 & 1/3 & 0 & 0 & 1/3 \\ 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 1/3 \\ 0 & 0 & 0 & 1/3 & 0 & 1/3 & 1/3 & 0 \end{pmatrix} \end{array},$$

which is symmetric. In fact, every transition matrix associated with some simple random walk on a graph is symmetric.

2.2.4 Hitting Times

Often, we only have local information about the graph, and we are interested in traversing the graph by effectively using local information. For example, suppose we start at a node u, and we want to reach another node v; and the neighbours of nodes can be obtained only by querying an oracle. If we perform a simple random walk on the graph, how many queries does it take to reach v from u? Hence, it is natural to create the notion of hitting times.

Definition 12 (Hitting Times)**.** *Given a Markov Chain with state space S, the hitting time τ_A of a subset $A \subseteq S$ is defined to be the first time one of the nodes in A is visited by the chain. Formally, if $X_t, t = 0, 1, 2, \ldots$ is the random walk, then*

$$\tau_A := \min\{n \geq 0 : X_t \in A\}.$$

Observe that by the Markov Inequality,

$$P(\tau_A \geq t) \leq \frac{E(\tau_A)}{t}.$$

Take $t = 3E(\tau_A)$, we get

$$P(\tau_A \geq 3E(\tau_A)) \leq \frac{E(\tau_A)}{3E(\tau_A)} = \frac{1}{3}$$

Hence, the random walk reaches the desired nodes with high probability in $O(E(\tau_A))$ many steps. Suppose a step of the walk may be implemented by performing a query, then $O(E(\tau_A))$ is clearly a upper bound on the query complexity. Therefore, it is of great interest to investigate the expected hitting time of random walks on graphs. The Hypercube is one of the examples that demonstrate an exponential expected hitting time in the dimension.

Example 13 (Hypercube revisited)**.** *Given a hypercube $H^{(n)} = (\mathbb{Z}_2^n, E)$, suppose we start at $\mathbf{0} = 000\ldots0$, and we perform simple random walk, what is the expected hitting time of reaching $\mathbf{1} = 111\ldots1$, i.e., $E(\tau_{\mathbf{1}})$?*

Let us first simplify the problem to a random walk on line. Observe that every node $u \in \mathbb{Z}_2^n$ can be classified by the shortest distance between u and $\mathbf{0}$. Denote A_k to be the set of nodes at distance k to $\mathbf{0}$, then $A_0 = \{\mathbf{0}\}$ and $A_n = \{\mathbf{1}\}$. For each node $u \in A_k$, it has k many 1-entries, and $n-k$ many 0-entries; hence it has k many neighbours in A_{k-1}, and $n-k$ many neighbours in A_{k+1}, then a simple random walk performed at u moves to A_{k-1} with probability $\frac{k}{n}$, and to A_{k+1} with probability $\frac{n-k}{n}$. Therefore, it is natural to consider the following modified random walk on the line:

with the transition matrix

$$T_{kj} = T_{A_k, A_j} = \begin{cases} \frac{k}{n} & j = k-1 \\ \frac{n-k}{n} & j = k+1 \\ 0 & otherwise \end{cases}$$

where $k, j \in \{0, \ldots, n\}$

Let t_k denote the number of steps the modified walk starting at A_k takes to reach A_n the first time, then we have the following recurrence relation,

$$t_k = \begin{cases} 1 + t_1 & k = 0 \\ 1 + b_k t_{k-1} + (1 - b_k) t_{k+1} & 0 < k < n \\ 0 & k = n \end{cases}$$

where b_k is a Bernoulli random variable that takes value 0 with probability $\frac{n-k}{n}$, and 1 with probability $\frac{k}{n}$. Note that by definition t_0 is the same as $\tau_{\mathbf{1}}$.

Now, taking the expectation, we have

$$E(t_k) = \begin{cases} E(1+t_1) & k=0 \\ E(1+b_k t_{k-1} + (1-b_k)t_{k+1}) & 0<k<n \\ E(0) & k=n \end{cases}$$
$$= \begin{cases} 1+E(t_1) & k=0 \\ 1+\frac{k}{n}E(t_{k-1}) + \frac{n-k}{n}E(t_{k+1}) & 0<k<n \\ 0 & k=n \end{cases},$$

where the second equality follows from the independence of b_k *and* t_i*, linearity of expectation, and the fact that* $E(b_k) = \frac{k}{n}$*.*

It remains to solve the recurrence relation. Start by expanding the first few terms,

$$\begin{aligned} E(t_1) &= 1 + \frac{1}{n}E(t_0) + \frac{n-1}{n}E(t_2) \\ &= 1 + \frac{1}{n}[1+E(t_1)] + \frac{n-1}{n}E(t_2) \\ \implies E(t_1) &= \left(\frac{1}{n}+1\right)\frac{n}{n-1} + E(t_2) \end{aligned}$$

$$\begin{aligned} E(t_2) &= 1 + \frac{2}{n}E(t_1) + \frac{n-2}{n}E(t_3) \\ &= 1 + \frac{2}{n}\left[\left(\frac{1}{n}+1\right)\frac{n}{n-1} + E(t_2)\right] + \frac{n-2}{n}E(t_3) \\ \implies E(t_2) &= \left[\left(\frac{1}{n}+1\right)\frac{n}{n-1}\frac{2}{n} + 1\right]\frac{n}{n-2} + E(t_3) \end{aligned}$$

By induction,

$$E(t_k) = \mu_k + E(t_{k+1})$$

where

$$\mu_k = \begin{cases} 1 & k=0 \\ (\frac{k}{n}\mu_{k-1}+1)\frac{n}{n-k} & 1<k\leq n-1 \end{cases}$$

Hence,

$$\begin{aligned}
E(\tau_{\mathbf{1}}) &= E(t_0) \\
&= \sum_{k=0}^{n-1} \mu_k \\
&= \sum_{k=0}^{n-1} \sum_{i=0}^{k} \left(\frac{n}{n-i} \prod_{j=i+1}^{k} \left(\frac{j}{n}\right) \left(\frac{n}{n-j}\right) \right) \\
&= n \cdot \sum_{k=0}^{n-1} \sum_{i=0}^{k} \frac{1}{n-i} \prod_{j=i+1}^{k} \frac{j}{n-j} \\
&= n \cdot \sum_{k=0}^{n-1} \sum_{i=0}^{k} \frac{k!(n-(k+1))!}{i!(n-i)!} \\
&= n \cdot \sum_{k=0}^{n-1} \sum_{i=0}^{k} \frac{\binom{n}{i}}{(n-k)\binom{n}{k}} \\
&\geq \frac{n}{(n-(n-1))\binom{n}{n-1}} \cdot \sum_{i=0}^{n-1} \binom{n}{i} \\
&= 2^n - 1,
\end{aligned}$$

which is approximately the same as the number of the nodes on the graph.

Due to the exponential expected hitting time in the dimension, we wonder if we may possibly achieve an exponential query complexity gap on some query problem based on hypercube. We examine this possibility in great details in the next chapter.

Chapter 3

Search on Modified Hypercube

3.1 Hypercube

In the previous chapter, we have seen the hypercube and exponential expected hitting time, and we are interested in coming up with a related query problem that exhibits exponential query complexity gap between classical and quantum algorithms. One natural approach is to model a random walk step as a query, but first, we need to make some adjustment to the graph.

Consider the hypercube $H^{(n)} = (\mathbb{Z}_2^n, E)$. Call $v \in \mathbb{Z}_2^n$ the i-th neighbour of u if $v = u \oplus e_i$ (i.e., v is obtained by flipping i-th bit of u). Note that if v is the i-th neighbour of u, then u is also the i-th neighbour of v.

The next step is to relabel the nodes so that we cannot tell the distance between two nodes by merely examining the labels. Formally, consider the graph $G^{(n)} = (\mathbb{Z}_2^n, E)$, where $\{u, v\} \in E(H^{(n)}) \iff \{\sigma(u), \sigma(v)\} \in E(G^{(n)})$, for a fixed σ drawn from the permutation group S_{2^n} uniformly at random; in particular, u and v are neighbours of each other in $H^{(n)}$ if and only if $\sigma(u)$ and $\sigma(v)$ are neighbours of each other in $G^{(n)}$.

Associated with $G^{(n)}$ is an oracle O that takes (u, i) as inputs, then returns (v, x) where v is the i-th neighbour of u in $G^{(n)}$ (the fixed but unknown ordering of the neighbours of each node is determined by σ), and

$$x = \begin{cases} i & \sigma^{-1}(v) \neq \mathbf{1} \\ 0 & \sigma^{-1}(v) = \mathbf{1} \end{cases}.$$

This way, we always know when the label of $\sigma(\mathbf{1})$ is detected.

Problem 14. *Suppose the label $\sigma(\mathbf{0})$ is given, what is the number of queries made by the most efficient algorithm in detecting the label of $\mathbf{1}$?*

With a minor modification to the query model, a $O(n)$ quantum walk based algorithm was proposed by Kempe in [K05], which is exponentially faster than $\Omega(2^n)$, the expected hitting time of simple random walk on the hypercube, as shown in Example 13. Do all classical algorithms take exponentially many queries in solving this problem? This answer is no; a $O(n^3)$ deterministic algorithm is briefly discussed in the Appendix of [CCD03]. The idea is to start at $\sigma(\mathbf{0})$, keep track of the neighbours of the current node that are closer to $\sigma(\mathbf{0})$, then move onto the remaining neighbours, thereby ensuring that the algorithm always moves away from $\sigma(\mathbf{0})$ by 1 unit of distance in each step.

In the hope of making it harder for the algorithm to keep track of the neighbours that are at known distance from $\sigma(\mathbf{0})$, we modify the hypercube by adding more edges.

3.2 Modified Hypercube

Definition 15 (k-Modified Hypercube)**.** *A k-modified hypercube is the graph $H_k^{(n)} = (\mathbb{Z}_2^n, E)$, where $\{u, v\} \in E$ for $u, v \in \mathbb{Z}_2^n$ if and only if $v = u \oplus (\oplus_{i \in I} e_i)$ for some $I \subseteq [n]$ such that $0 < |I| \leq k$. We call u and v the I-th neighbours of each other if $v = u \oplus (\oplus_{i \in I} e_i)$.*

Note that the standard hypercube is simply $H_1^{(n)}$, and $H_k^{(n)}$ is constructed by connecting distinct nodes of distance no greater than k in $H_1^{(n)}$ by edges.

Consider the node relabeled graph $G_k^{(n)} = (\mathbb{Z}_2^n, E)$, where $\{u, v\} \in E(H_k^{(n)}) \iff \{\sigma(u), \sigma(v)\} \in E(G_k^{(n)})$, for a fixed σ drawn from the permutation group S_{2^n} uniformly at random. Associated with $G_k^{(n)}$ is an oracle O that takes (u, I) as input, then returns (v, X), where v is the I-th neighbour of $\sigma(u)$ (the fixed but unknown ordering of the neighbours of each node is determined by σ), and

$$X = \begin{cases} I & \sigma^{-1}(v) \neq \mathbf{1} \\ 0 & \sigma^{-1}(v) = \mathbf{1} \end{cases}.$$

This way, we always know when the label of $\mathbf{1}$ is detected.

Suppose the label $\sigma(\mathbf{0})$ is given, we are interested in the number of queries made by the most efficient algorithm in detecting the label of $\sigma(\mathbf{1})$. In the remaining section, we denote $\sigma(u)$ by u_σ.

Let us first make an interesting observation that this kind of modification may gradually slow down the most efficient quantum algorithm when k gets larger than 1. The main idea is to show we cannot distinguish between $\mathbf{1}$ and some other nodes, which reduces the detection problem into finding a unique marked element, then apply the lower bound we see in Example 3. Below is a lemma that applies to $k = 2$.

Let $Aut(G_2^{(n)})$ denote the group of automorphisms of $G_2^{(n)}$, and consider the following equivalence classes $[\cdot]$ defined on $Aut(G_2^{(n)})$,

$$[\tau_1] = [\tau_2] \iff \tau_1(\mathbf{0}_\sigma) = \tau_2(\mathbf{0}_\sigma) \text{ and } \tau_1(\mathbf{1}_\sigma) = \tau_2(\mathbf{1}_\sigma).$$

Moreover, it is not hard to see $|[\tau]| = |[\nu]|$ for all automorphisms τ and ν. In fact, $\forall \tau_1, \tau_2 \in [\tau], \tau_2 = \tau_1\chi$ for some $\chi \in [Id]$, where $[Id]$ is the equivalence class induced by the identity mapping; indeed, we have $\chi = \tau_1^{-1}\tau_2 \in [Id]$. Hence $|[\tau]| = |[Id]| = |[\nu]|$.

Proposition 16. *Let n be even, then for each $i \in [n]$, $G_2^{(n)}$ has an equivalence class $[\tau]$ such that*

$$\tau(\mathbf{0}_\sigma) = \mathbf{0}_\sigma$$

but

$$\tau(\mathbf{1}_\sigma) = (\mathbf{1} \oplus e_i)_\sigma \neq \mathbf{1}_\sigma.$$

Proof. This proposition is easy to see if we work with the standard hypercube $H = H^{(n)}$ instead. The hypercube H can be seen as the graph obtained by connecting nodes u and $u \oplus e_j$, $\forall j \in [n]$.

Fix $i \in [n]$, consider the graph $H_i = (\mathbb{Z}_2^n, E_i), E_i = \{\{u, u \oplus e'_j\} \mid j \in [n]\}$, where

$$e'_j = \begin{cases} e_i & j = i \\ e_i \oplus e_j & j \neq i \end{cases}.$$

To see H_i is also a standard hypercube (i.e., isomorphic to H), we construct a bijection between H and H_i. Write the nodes $u \in \mathbb{Z}_2^n$ in the form

$$u = \oplus_{j \in I_u} e_j,$$

where $I_u = \{j : u_j = 1\}$ is the collection of indices of 1-entry of u.

Consider the mapping $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$,

$$f(u) = f(\oplus_{j \in I_u} e_j) = \oplus_{j \in I_u} e'_j.$$

Since for each $v \in \mathbb{Z}_2^n$, the decomposition $v = \oplus_{j \in I_v} e'_j$ is unique, $u = \sum_{j \in I_v} e_j$ is the unique node such that $f(u) = v$, hence f is indeed a bijection. In particular,

$$v = u \oplus e_j \iff f(v) = f(u) \oplus e'_j,$$

i.e., $\{u, v\} \in E \iff \{f(u), f(v)\} \in E_i$, therefore H_i is also a hypercube.

Now, for each u and v of distance no greater than 2 in H, we have $v = u \oplus_{j \in S} e_j$ for some $S \subseteq [n], 0 < |S| \leq 2$, then $v = u \oplus_{j \in S_i} e'_j$, where

$$S_i = \begin{cases} S \cup \{i\} & |S| = 1 \\ S \setminus \{i\} & |S| = 2 \end{cases}.$$

In other words, u and v are of distance no greater than 2 in H_i as well, hence both H and H_i induce the same modified hypercube $H_2^{(n)}$ and the nodes relabeled $G_2^{(n)}$, and the existence of automorphisms follows. □

Lemma 17. *Let n be even, then any quantum algorithm must make $\Omega(n^{1/2})$ queries in detecting the label of $\mathbf{1}_\sigma$ in $G_2^{(n)}$.*

Proof. As a consequence of the previous proposition, we cannot distinguish between $(\oplus_{j \in S} e_i)_\sigma$ and $(\oplus_{j \in S} e'_j)_\sigma$ in $G_2^{(n)}$ by only examining the structure of $G_2^{(n)}$; in particular, we cannot distinguish between

$$(\oplus_{j \in [n]} e_j)_\sigma = \mathbf{1}_\sigma$$

and

$$(\oplus_{j \in [n]} e'_j)_\sigma = (\mathbf{1} \oplus e_i)_\sigma \neq \mathbf{1}_\sigma.$$

Since the structure does not help, in order to detect $\mathbf{1}_\sigma$, we can only tell whether the label has been detected by reading the output of the oracle, hence the label detection problem is at least as hard as finding the marked element $\mathbf{1}_\sigma$ among $n + 1$ elements. As mentioned in Example 3, the lower bound for the quantum query complexity is $\Omega(n^{1/2})$. □

For $k > 2$, we speculate such indistinguishability still exists, since connecting nodes of further Hamming distance gives more freedom of choice of automorphisms.

3.2.1 An efficient classical algorithm

In this section, we see the idea from the Appendix of [CCD03] may be adapted to design an efficient classical query algorithm for the modified hypercube. Again, we demonstrate the idea for $k = 2$.

Algorithm 1 An efficient detection algorithm for $G_2^{(n)}$

1: Let w_1 be an arbitrary neighbour of $\mathbf{0}_\sigma$, and let $S_1 = \mathcal{N}(w_1) \cap (\{\mathbf{0}_\sigma\} \cup \mathcal{N}(\mathbf{0}_\sigma))$

2: **for** $i = 2$ to $\lfloor n/2 \rfloor$ **do**

3: Let w_i be an arbitrary neighbour of w_i that is not in S_{i-1}, i.e.,

$$w_i \in \mathcal{N}(w_{i-1}) \setminus S_{i-1}$$

4: $S_i \leftarrow \mathcal{N}(w_i) \cap (\cup_{u \in S_{i-1}} \mathcal{N}(u))$

5: Search for $\mathbf{1}_\sigma$ in $\{w_{\lfloor n/2 \rfloor}\} \cup \mathcal{N}(w_{\lfloor n/2 \rfloor})$.

The idea is to start at $\sigma(\mathbf{0})$, keep track of the neighbours of the current node that are closer to $\sigma(\mathbf{0})$, then move onto the remaining neighbours, thereby ensuring that the algorithm always moves away from $\sigma(\mathbf{0})$ by at least 1 unit of distance in each step.

For simplicity, we work with labels of the nodes in $H^{(n)}$. Before proving the correctness of the algorithm, we need a useful lemma.

Lemma 18. *Let $h(\cdot)$ denote the Hamming weight. $\forall 1 \leq i \leq \lfloor n/2 \rfloor$, we have*

$$2i - 1 \leq h(w_i) \leq 2i \tag{3.1}$$

and

$$S_i = \{u \in \mathcal{N}(w_i) : h(u) \leq 2i\} \tag{3.2}$$

Proof. It is straight forward to verify that (3.1) and (3.2) hold for $i = 1$. Suppose they hold for $i - 1 \in \mathbb{N}$.

Since $w_i \in \mathcal{N}(w_{i-1}) \setminus S_{i-1}$ and $S_{i-1} = \{u \in \mathcal{N}(w_{i-1}) : h(u) \leq 2(i-1)\}$, we must have $h(w_i) > 2(i-1)$. Also, the neighbours of any node u in $H^{(n)}$ is at most 2 Hamming distance away from u, hence $h(w_i) \leq h(w_{i-1}) + 2 \leq 2(i-1) + 2 = 2i$,

$$\implies 2i - 1 \leq h(w_i) \leq 2i.$$

which completes the induction on (3.1). It remains to show (3.2).

($\subseteq$) By definition, $S_i \subseteq \mathcal{N}(w_i)$. Also, $\forall u \in S_{i-1}$, we have $h(u) \leq 2(i-1)$, then $\forall v \in \mathcal{N}(u)$, $h(v) \leq 2i$, hence $S_i \subseteq \{u \in \mathcal{N}(u) : h(u) \leq 2i\}$.

($\supseteq$) Let $u^* \in \{u \in \mathcal{N}(w_i) : h(u) \le 2i\}$, then $u^* = w_i \oplus_{i\in I} e_i$ for some indices $0 < |I| \le 2$; we need to show $u^* \in S_i$.

Since $h(w_{i-1}) \ge 2(i-1) - 1 \ge 1$, we have $(w_{i-1})_\gamma = 1$ for some index γ. We consider the following two cases

1. Suppose $h(w_i) - h(w_{i-1}) = 2$, then $w_i = w_{i-1} \oplus e_\alpha \oplus e_\beta$ for some indices $\alpha \ne \beta$ and $\gamma \notin \{\alpha, \beta\}$, which implies $(w_{i-1})_\alpha = (w_{i-1})_\beta = 0$ and $(w_i)_\gamma = 1$.

   If $u^*_\alpha = u^*_\beta = 1$, then

   $$\{\alpha, \beta\} \cap I = \emptyset \implies h(u^* \oplus e_\alpha \oplus e_\beta) = h(u^*) - 2 \le 2i - 2 = 2(i-1),$$

   and

   $$\begin{aligned} & u^* \oplus e_\alpha \oplus e_\beta = w_{i-1} \oplus_{i\in I} e_i \\ \implies & u^* \oplus e_\alpha \oplus e_\beta \in \{u \in \mathcal{N}(w_{i-1}) : h(u) \le 2(i-1)\} = S_{i-1} \\ \implies & u^* = (u^* \oplus e_\alpha \oplus e_\beta) \oplus e_\alpha \oplus e_\beta \in S_i. \end{aligned}$$

   Else, $u^*_\alpha = u^*_\beta = 0$, then $\{\alpha, \beta\} = I$ and $u^* = w_{i-1}$. Since

   $$h(w_{i-1} \oplus e_\gamma) = h(w_{i-1}) - 1 \le 2(i-1),$$

   we have $w_{i-1} \oplus e_\gamma \in S_{i-1}$ by induction on (3.2). It follows that

   $$u^* = w_{i-1} = (w_{i-1} \oplus e_\gamma) \oplus e_\gamma \in S_i,$$

   Otherwise, WLOG, suppose $u^*_\alpha = 1$ but $u^*_\beta = 0$, i.e., $\alpha \notin I$ but $\beta \in I$.
   If $u^*_\gamma = 1$, then

   $$h(u^* \oplus e_\alpha \oplus e_\gamma) = h(u^*) - 2 \le 2i - 2 = 2(i-1).$$

   Since by induction on (3.2),

   $$u^* \oplus e_\alpha \oplus e_\gamma = w_{i-1} \oplus e_\gamma \oplus_{i\in I\setminus\{b\}} e_i \in S_{i-1},$$

   we have $u^* \in S_i$.
   Else, $u^*_\gamma = 0$, then

   $$\gamma \in I \implies I = \{\beta, \gamma\} \implies u^* \oplus e_\alpha = w_{i-1} \oplus e_\gamma \in S_{i-1},$$

   so $u^* \in S_i$.

2. Suppose $h(w_i) - h(w_{i-1}) = 1$, which, by (3.1), happens only when $h(w_i) = 2i-1$ and $h(w_{i-1}) = 2(i-1)$, then $w_i = w_{i-1} \oplus e_\lambda$ for some index $\lambda \in [n]$ and $(w_{i-1})_\lambda = 0$. Note that $\lambda \neq \gamma$ and $(w_i)_\gamma = 1$.

   First consider the case $u^*_\lambda = 1$.
   If $h(u^*) = 2i$, since $h(w_i) = 2i-1$, we must have $u^* = w_i \oplus e_\theta$ for some index $\theta \in [n]$ such that $(w_i)_\theta = 0$. Since $\gamma \neq \theta$, we have $u^*_\gamma = 1$ and

$$h(u^* \oplus e_\gamma \oplus e_\theta) = h(u^*) - 2 = 2i - 2 = 2(i-1),$$

   so by induction on (3.2),

$$u^* \oplus e_\gamma \oplus e_\theta = w_{i-1} \oplus e_\lambda \oplus e_\gamma \in S_{i-1} \implies u^* \in S_i.$$

   Else, $h(u^*) \leq 2i-1$, then

$$h(u^* \oplus e_\lambda) = h(u^*) - 1 \leq 2i - 2 = 2(i-1),$$

   and

$$u^* \oplus e_\lambda = w_{i-1} \oplus_{i \in I} e_i \in S_{i-1} \implies u^* \in S_i.$$

   Otherwise suppose $u^*_\lambda = 0$, then $\lambda \in I$ and $h(u^*) \leq h(w_i) = 2i-1$.
   If $u^*_\gamma = 1$, then

$$h(u^* \oplus e_\gamma) = h(u^*) - 1 \leq 2i - 2 = 2(i-1),$$

   so by induction on (3.2),

$$u^* \oplus e_\gamma = w_{i-1} \oplus e_\gamma \oplus_{i \in I \setminus \{\lambda\}} e_i \in S_{i-1} \implies u^* \in S_i.$$

   Else, $u^*_\gamma = 0$, then $I = \{\lambda, \gamma\}$, hence

$$u^* \oplus e_\lambda = w_{i-1} \oplus e_\lambda \oplus e_\gamma \in S_{i-1} \implies u^* \in S_i.$$

Therefore, (3.2) holds by induction.

□

In each step of the algorithm, the Hamming distance between $\mathbf{0}$ and w_i is at least $2i-1$. Since the Hamming distance between $\mathbf{0}$ and $\mathbf{1}$ is n, $v_{\lfloor n/2 \rfloor}$ is $\mathbf{1}$ or a neighbour of $\mathbf{1}$, and a brute force search guarantees the detection of $\mathbf{1}$.

Since $S_i \subseteq \mathcal{N}(w_i)$, S_i contains $O(n^2)$ nodes, which implies $\cup_{u \in S_{i-1}} \mathcal{N}(u)$ contains at most $O(n^4)$ nodes; since there are $O(n)$ iterations, the number of queries made by steps 2 to 4 is bounded above by $O(n^5)$. The last step takes $O(n^2)$ queries. Therefore, the algorithm runs in $O(n^5)$ queries.

Thus, the query complexity gap between classical and quantum algorithms for the detection problem on $G_2^{(n)}$ is at most polynomial. We present the Glued Tree problem in the next chapter, and that problem does have an exponential query complexity gap.

Chapter 4

Glued Trees Problem

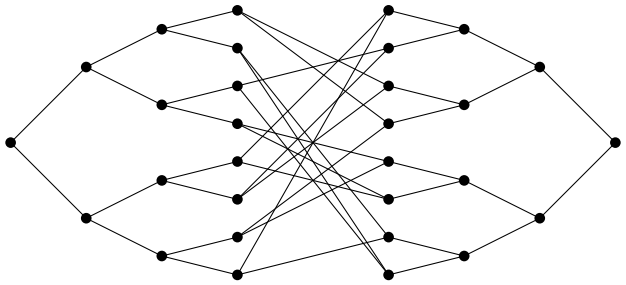
4.1 Introduction

In this chapter, we present a problem that exhibits an exponential query complexity gap, namely the Glued Trees problem; this problem was first proposed in [CCD03]. We introduce a slightly modified version of it, with the input size of the oracle reduced to the number of nodes, thereby strengthening the query complexity gap to $O(\text{polylog}(N))$ quantum queries to $\Omega(N^{1/2}/\log(N))$ classical queries. We also prove a tight $\Theta(N^{1/2})$ bound on the classical query complexity, up to a log factor.

Definition 19 (Glued Trees)**.** *An instance of Glued Trees of $2n+2$ levels is a graph $G_n = (V, E)$ constructed by joining two perfect binary trees of height n by a cycle of length $2 \cdot 2^n$ that alternates between the leaves of the two trees; the cycle is arbitrary, subject to this constraint.*

We call the root of the left tree ENTRANCE and the root of the right tree EXIT. Starting from the ENTRANCE, we label each level consecutively from 1 to $2n+2$.

Example 20. *An instance of Glued Trees of* 8 *levels, i.e.,* G_3*:*



with the leftmost node being the ENTRANCE and the rightmost node being the EXIT.

Since each perfect binary tree has $\sum_{i=0}^{n} 2^i = 2^{(n+1)} - 1$ nodes, there are $2 \cdot 2^{(n+1)} - 1 = 2^{(n+2)} - 2$ nodes in total, which matches the number of elements in $\mathbb{Z}_2^{(n+2)} \setminus \{000\ldots0, 111\ldots1\}$. Hence, we may take $V = \mathbb{Z}_2^{(n+2)} \setminus \{000\ldots0, 111\ldots1\}$, i.e., assign each node $v \in V$ a unique label from $\mathbb{Z}_2^{(n+2)} \setminus \{000\ldots0, 111\ldots1\}$, and it is a bijection; in particular, the ENTRANCE is always assigned $000\ldots01$, and the remaining nodes get labels from $\mathbb{Z}_2^{(n+2)} \setminus \{000\ldots0, 000\ldots01, 111\ldots1\}$ uniformly at random.

Associated with a Glued Tree is an oracle O that takes $v \in V$ as input, then returns the labels of neighbours of v, and it returns $111\ldots1$ if the EXIT is the input. We are interested in the number of queries made by the most efficient algorithm in detecting the label of the EXIT. In [CCD03], Childs et al. showed a quantum algorithm running in $O(\text{poly}(n))$ queries. In the next two sections, we show the classical query complexity for this problem is $\Theta(2^{n/2})$, up to a log factor.

Note that this is a different problem from the one defined originally in [CCD03]. Two major differences are

1. The input size is $O(2^{2n})$ in [CCD03], whereas it is $O(2^n)$ in this chapter. If we let $N = 2^n$, then the query complexity gap for this problem is at least $O(\text{polylog}(N))$ quantum queries to $\Omega(N^{1/2}/\log(N))$ classical queries, which is $N^{1/2}/\text{polylog}(N)$.

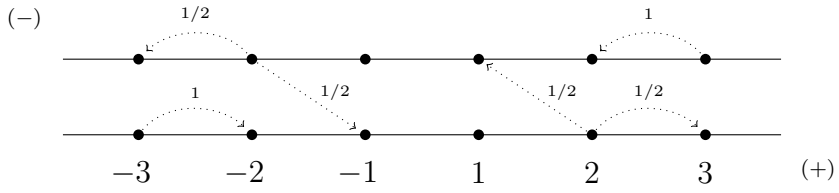
2. The oracle defined in [CCD03] included a colouring of the edges; this was mainly engaged to show the existence of an efficient quantum algorithm. We simplify the oracle here, since we are only concerned with the classical query complexity in this chapter. Our result still holds under their setting (with an edge colouring).

4.2 An $\Omega(2^{n/2}/n)$ Classical Lower Bound

In this section, we show any classical algorithm makes $\Omega(2^{n/2}/n)$ queries in order to detect the EXIT. Observe that in each query, any algorithm can either pick and query the label of an unknown node [1], or query a known node. We first demonstrate it is possible to simulate the process of randomly picking a unknown node by a special Tree Embedding process originating at the ENTRANCE. Hence any classical algorithm can be represented as some Tree Embedding process as described in [CCD03]. We show that this detects neither the EXIT nor a cycle with high probability in $o(2^{n/2})$ queries. First, let us study a modified random walk on line that lies at the heart of the simulation process.

4.2.1 Random Walk on $({}^{(\pm)}, \mathbb{Z} \setminus \{0\})$

Consider a random walk W on $({}^{(\pm)}, \mathbb{Z}\setminus\{0\})$ with initial probability distribution $P({}^{(-)}-1, 0) = P({}^{(+)}1, 0) = \frac{1}{2}$, where ${}^{(\pm)}k, m$ denotes the walk reaching k in m steps with the direction taken in the last step being ${}^{(\pm)}$. For example, $P({}^{(+)}5, 11)$ is the probability that the walk ends on 5 in 11 steps and the last direction the walk takes is right (i.e., the walk moves from 4 to 5 in the last step). In the language used in chapter 2, $P({}^{(\pm)}k, m) = P(W_m = ({}^{(\pm)}, k))$.



The nodes at the top are $({}^{(-)}, k)$, and the nodes at the bottom are $({}^{(+)}, k)$.

The walk has a probability transition matrix T such that

$$T({}^{(-)}k, {}^{(-)}\, k-1) = \begin{cases} \frac{1}{2} & k<0 \\ 1 & k>0 \end{cases} \qquad T({}^{(-)}k, {}^{(+)}\, k+1) = \begin{cases} \frac{1}{2} & k<0 \\ 0 & k>0 \end{cases}$$

$$T({}^{(+)}k, {}^{(-)}\, k-1) = \begin{cases} 0 & k<0 \\ \frac{1}{2} & k>0 \end{cases} \qquad T({}^{(+)}k, {}^{(+)}\, k+1) = \begin{cases} 1 & k<0 \\ \frac{1}{2} & k>0 \end{cases}$$

Note that when $k = 1$, then $k-1$ is replaced by -1; similarly for $k = -1$ and $k+1$.

[1] an unknown node is a node that has not been either the input or the output of any query; in particular, we always treat ENTRANCE as a known node.

Call $\{k : k < 0\}$ the left region and $\{k : k > 0\}$ the right region, then the random walk captures the property that

- If the walk takes a step to the left (or right, resp.) in the left region (or right region, resp.), then the next step it takes is either left or right with probability $1/2$ each.
- If the walk takes a step to the right (or left, resp.) in the left region (or right region, resp.), then it goes back to $({}^{(+)}, 1)$ (or $({}^{(-)}, -1)$, resp.) definitely.

The key property we prove in the end is that if we perform a m-step walk, then the probability of the walk landing on k or $-k, k > 0$ is

$$P(-k, m) = P(k, m) = \begin{cases} \frac{1}{2^{(|k|+1)}} & k \in [m] \\ \frac{1}{2^{(m+1)}} & k = m+1 \\ 0 & k > m+1 \end{cases}$$

where $P(k, m) = P({}^{(-)}k, m) + P({}^{(+)}k, m)$.

Lemma 21. *$\forall k > 1, m \in \mathbb{N}$, we have*

$P({}^{(+)}k, m) = \frac{1}{2}P({}^{(+)}k - 1, m - 1)$, $\quad P({}^{(-)}, k, m) = \frac{1}{2}P({}^{(-)}k - 1, m - 1)$,

$P({}^{(-)}{-k}, m) = \frac{1}{2}P({}^{(-)}{-k} + 1, m - 1)$, $\quad P({}^{(+)}{-k}, m) = \frac{1}{2}P({}^{(+)}{-k} + 1, m - 1)$.

Proof. $k > 1, m \in \mathbb{N}$

In the right region,

$$\begin{aligned} P({}^{(+)}k, m) &= P({}^{(+)}k - 1, m - 1)T({}^{(+)}k - 1, {}^{(+)}k) + P({}^{(-)}k - 1, m - 1)T({}^{(-)}k - 1, {}^{(+)}k) \\ &= \frac{1}{2}P({}^{(+)}k - 1, m - 1). \end{aligned}$$

Now, observe that the walk has the following probability distribution

$m = 0$: $\quad P({}^{(+)}1, 0) = P({}^{(-)}{-1}, 0) = 1/2$,

$m = 1$: $\quad P({}^{(+)}2, 1) = P({}^{(-)}{-1}, 1) = P({}^{(+)}1, 1) = P({}^{(-)}{-2}, 1) = 1/4$.

Hence, $P({}^{(-)}, k, 1) = \frac{1}{2}P({}^{(-)}k - 1, 0)$.

Next, $\forall m \geq 2$,

$$\begin{aligned} P(^{(-)}k, m) &= P(^{(-)}k+1, m-1)T(^{(-)}k+1, ^{(-)}k) + P(^{(+)}k+1, m-1)T(^{(+)}k+1, ^{(-)}k) \\ &= P(^{(-)}k+1, m-1) + \frac{1}{2}P(^{(+)}k+1, m-1) \end{aligned}$$

$$\implies P(^{(-)}k-1, m-1) = P(^{(-)}k, m-2) + \frac{1}{2}P(^{(+)}k, m-2) = P(^{(-)}k, m-2) + P(^{(+)}k+1, m-1).$$

Hence, to show

$$P(^{(-)}k, m) = \frac{1}{2}P(^{(-)}k-1, m-1),$$

it suffices to show

$$P(^{(-)}k+1, m-1) = \frac{1}{2}P(^{(-)}k, m-2),$$

or inductively (on m) that

$$P(^{(-)}k+m-1, 1) = \frac{1}{2}P(^{(-)}k+m-2, 0).$$

This is true, because by the probability distribution,

$$P(^{(-)}k+m-1, 1) = P(^{(-)}k+m-2, 0) = 0, \qquad \forall k > 1, m \in \mathbb{N}.$$

By symmytry, in the left region, we have

$$P(^{(-)}-k, m) = \frac{1}{2}P(^{(-)}-k+1, m-1),$$

$$P(^{(+)}-k, m) = \frac{1}{2}P(^{(+)}-k+1, m-1).$$

□

Observe that the walk can be decomposed into two random walks on $(^{\pm}, \mathbb{Z} \setminus \{0\})$ that are related to each other by symmetry, namely the one starting at $(^{(-)}, -1)$ and the one starting at $(^{(+)}, 1)$. Indeed, let W_l denote the walk starting at $(^{(-)}, -1)$ and W_r denote the walk starting at $(^{(+)}, 1)$, then we have $P(W_l =^{(sign)} k, m) = P(W_r =^{(-sign)} -k, m)$.

In odd steps, W_l (or W_r, resp.) has nonzero probability only on nodes that are odd levels away from -1 (or 1, resp.). Since there is an odd distance between 1 and -1, by parity arguments, those two walks occupy alternating levels so they do not overlap. The case is similar in even steps. Hence, we only need to analyze W_r, then apply the result to W_l.

Lemma 22. *Consider the walk starting at* $({}^{(+)}, 1)$ *with total probability* $1/2$. *Let* $m \in \mathbb{N}$, *then* $P({}^{(\pm)}k, m) = 0$ *except in the following cases:*

1.

$$P({}^{(+)}m+1, m) = P({}^{(+)}m-1, m) = P({}^{(-)}m-1, m) = P({}^{(-)}-m, m) = \frac{1}{2^{(m+1)}}.$$

2. $\forall$ *odd* $j \in [3, m-1]$

$$P({}^{(-)}m-j, m) = 2P({}^{(+)}m-j+2, m) + \frac{1}{2^{(m+1)}},$$

$$P({}^{(+)}m-j, m) = 2P({}^{(-)}m-j, m) - \frac{1}{2^{(m+1)}}.$$

3. $\forall$ *even* $l \in [2, m-1]$

$$P({}^{(+)}-m+l, m) = 2P({}^{(-)}-m+l-2, m) - \frac{1}{2^{(m+1)}},$$

$$P({}^{(-)}-m+l, m) = 2P({}^{(+)}-m+l, m) + \frac{1}{2^{(m+1)}}.$$

4. In particular, if $m \geq 2$ *is even, then*

$$P({}^{(-)}1, m) = P({}^{(-)}-2, m) = 2P({}^{(+)}-2, m) + \frac{1}{2^{(m+1)}},$$

$$P({}^{(+)}1, m) = 2P({}^{(-)}1, m) - \frac{1}{2^{(m+1)}}.$$

5. if $m \geq 3$ *is odd, then*

$$P({}^{(+)}-1, m) = P({}^{(+)}2, m) = 2P({}^{(-)}2, m) - \frac{1}{2^{(m+1)}},$$

$$P({}^{(-)}-1, m) = 2P({}^{(+)}-1, m) + \frac{1}{2^{(m+1)}}.$$

Proof. Observe that the walk has the following probability distribution

$m = 0:$ $\quad P({}^{(+)}1, 0) = 1/2,$

$m = 1:$ $\quad P({}^{(+)}2, 1) = 1/4, \qquad P({}^{(-)}-1, 1) = 1/4,$

$$
\begin{array}{lll}
m=2: & P(^{(+)}3,2)=1/8, & P(^{(+)}1,2)=1/8, \\
 & P(^{(-)}1,2)=1/8, & P(^{(-)}{-}2,2)=1/8, \\
m=3: & P(^{(+)}4,3)=1/16, & P(^{(+)}2,3)=1/16, \quad P(^{(+)}{-}1,3)=1/16, \\
 & P(^{(-)}2,3)=1/16, & P(^{(-)}{-}1,3)=3/16, \quad P(^{(-)}{-}3,3)=1/16, \\
m=4: & P(^{(+)}5,4)=1/32, & P(^{(+)}3,4)=1/32, \\
 & P(^{(+)}1,4)=5/32, & P(^{(+)}{-}2,4)=1/32, \\
 & P(^{(-)}3,4)=1/32, & P(^{(-)}1,4)=3/32, \\
 & P(^{(-)}{-}2,4)=3/32, & P(^{(-)}{-}4,4)=1/32.
\end{array}
$$

which clearly satisfies the relations. Hence, suppose it holds true for $n \in \mathbb{N}, 4 \leq n < m$. By the previous Lemma and induction,

1. $\forall m \geq 4$,

$$
\begin{aligned}
& P(^{(+)}m+1,m) = P(^{(+)}m-1,m) = P(^{(-)}m-1,m) = P(^{(-)}{-}m,m) \\
=& \frac{1}{2}P(^{(+)}m,m-1) = \frac{1}{2}P(^{(+)}m-2,m-1) = \frac{1}{2}P(^{(-)}m-2,m-1) = \frac{1}{2}P(^{(-)}{-}m+1,m-1) \\
=& \frac{1}{2^{(m+1)}}
\end{aligned}
$$

2. $\forall m-j>1$,

$$
\begin{aligned}
P(^{(-)}m-j,m) &= \frac{1}{2}P(^{(-)}m-1-j,m-1) \\
&= P(^{(+)}m+1-j,m-1) + \frac{1}{2^{(m+1)}} \\
&= 2P(^{(+)}m-j+2,m) + \frac{1}{2^{(m+1)}}
\end{aligned}
$$

$$
\begin{aligned}
P(^{(+)}m-j,m) &= \frac{1}{2}P(^{(+)}m-1-j,m-1) \\
&= P(^{(-)}m-1-j,m-1) - \frac{1}{2^{(m+1)}} \\
&= 2P(^{(-)}m-j,m) - \frac{1}{2^{(m+1)}}
\end{aligned}
$$

3. Similarly, $\forall -m+l<-1$,

$$P(^{(+)}-m+l,m)=2P(^{(-)}-m+l-2,m)-\frac{1}{2^{(m+1)}}$$

$$P(^{(-)}-m+l,m)=2P(^{(+)}-m+l,m)-\frac{1}{2^{(m+1)}}$$

It remains to consider the cases $m-j=1$ (so m is even) and $-m+l=-1$ (so m is odd). Note that exactly one of them holds in any step by parity argument.

Suppose $m-1$ is odd, then in step $m-1$, the walk is distributed on odd levels away from the node 1. By induction on 5., we have

$$P(^{(+)}-1,m-1)=P(^{(+)}2,m-1)=2P(^{(-)}2,m-1)-\frac{1}{2^m}$$

and

$$P(^{(-)}-1,m-1)=2P(^{(+)}-1,m-1)+\frac{1}{2^m}=2P(^{(+)}2,m-1)+\frac{1}{2^m}$$

then

$$\begin{aligned}
P(^{(-)}1,m) &= P(^{(-)}2,m-1)T(^{(-)}2,^{(-)}1)+P(^{(+)}2,m-1)T(^{(+)}2,^{(-)}1)\\
&= P(^{(-)}2,m-1)+\frac{1}{2}P(^{(+)}2,m-1)\\
&= \frac{1}{2}(2P(^{(+)}2,m-1)+\frac{1}{2^m}) && \text{by 2.}\\
&= \begin{cases} \frac{1}{2}P(^{(-)}-1,m-1) \\ 2(\frac{1}{2}P(^{(+)}2,m))+\frac{1}{2^{(m+1)}} \end{cases}\\
&= \begin{cases} P(^{(-)}-2,m) \\ 2P(^{(+)}3,m)+\frac{1}{2^{(m+1)}} \end{cases} && \text{by Lemma}
\end{aligned}$$

and

$$\begin{aligned} P({}^{(+)}1, m) &= P({}^{(+)}{-}1, m-1)T({}^{(+)}{-}1, {}^{(+)}1) + P({}^{(-)}{-}1, m-1)T({}^{(-)}{-}1, {}^{(+)}1) \\ &= P({}^{(+)}{-}1, m-1) + \frac{1}{2}P({}^{(-)}{-}1, m-1) \\ &= 2P({}^{(+)}2, m-1) + \frac{1}{2^{(m+1)}} \\ &= 2P({}^{(-)}1, m) - \frac{1}{2^{(m+1)}} \end{aligned}$$

thus completing the induction on 2 and 4. The cases 3 and 5 in which m is odd is similar. □

We are ready to prove the main property.

Theorem 23. *Suppose $m \in \mathbb{N}$ steps of the random walk are performed, then*

$$P(-k, m) = P(k, m) = \begin{cases} \frac{1}{2^{(|k|+1)}} & k \in [m] \\ \frac{1}{2^{(m+1)}} & k = m+1 \\ 0 & k > m+1 \end{cases}$$

where $P(k, m) = P({}^{(-)}k, m) + P({}^{(+)}k, m)$.

Proof. It suffices to show this property for the walk starting at $({}^{(+)}, 1)$ with total probability $1/2$, then extends to the random walk by symmetry.

$\forall m \in \mathbb{N}$, by the previous Lemma, we have

$$P(m+1, m) = P({}^{(+)}m+1, m) = \frac{1}{2^{(m+1)}}$$

$$P(m-1, m) = P({}^{(+)}m-1, m) + P({}^{(-)}m-1, m) = 2(\frac{1}{2^{(m+1)}}) = \frac{1}{2^m}$$

$$P(-m, m) = P({}^{(-)}{-}m, m) = \frac{1}{2^{(m+1)}}$$

$\forall$ odd $j \in [3, m-1]$,

$$\begin{aligned} P(m-j,m) &= P(^{(+)}m-j,m) + P(^{(-)}m-j,m) \\ &= 2(2P(^{(+)}m-j+2,m) + \frac{1}{2^{(m+1)}}) - \frac{1}{2^{(m+1)}} \\ &\quad + 2(2P(^{(-)}m-j+2,m) - \frac{1}{2^{(m+1)}}) + \frac{1}{2^{(m+1)}} \\ &= 4(P(^{(+)}m-j+2,m) + P(^{(-)}m-j+2,m)) \\ &= 4P(m-j+2,m) \end{aligned}$$

$\forall$ even $l \in [2, m-1]$,

$$\begin{aligned} P(-m+l,m) &= P(^{(+)}-m+l,m) + P(^{(-)}-m+l,m) \\ &= 2(2P(^{(+)}-m+l-2,m) + \frac{1}{2^{(m+1)}}) - \frac{1}{2^{(m+1)}} \\ &\quad + 2(2P(^{(-)}-m+l-2,m) - \frac{1}{2^{(m+1)}}) + \frac{1}{2^{(m+1)}} \\ &= 4(P(^{(+)}-m+l-2,m) + P(^{(-)}-m+l-2,m)) \\ &= 4P(-m+l-2,m) \end{aligned}$$

then apply induction we obtain the desired results on all nodes occupied by the walk starting at $(^{(+)}, 1)$. By symmetry argument, $P(-k,m) = P(k,m)$, thereby extending the results to nodes occupied by the walk originated from $(^{(-)}, 1)$ as well. $\square$

4.2.2 Simulate the Random Picking Process

Consider a Glued Tree of $2n+2$ levels, then there are $2^{n+2}-2$ nodes in total. Suppose the algorithm has queried t nodes, and it picks a node from the remaining $2^{n+2}-2-t$ unknown nodes uniformly at random, then each node gets picked with probability $\frac{1}{2^{n+2}-2-t}$.

Note that one is allowed to pick an unknown node according to some distribution other than the uniform distribution, but since each unknown node gets a label uniformly at random, we have $\forall$ unknown $v \in V$, $P($ label of v is a undiscovered label $z) = \frac{1}{2^{n+2}-2-t}$,

and

$$\begin{aligned} P(v \text{ gets picked}) &= \sum_{z \text{ undiscovered label}} P(v \text{ gets picked} \mid \text{label of } v \text{ is } z) P(\text{label of } v \text{ is } z) \\ &= \frac{1}{2^{n+2} - 2 - t} \sum_{z \text{ undiscovered label}} P(z \text{ gets picked}) \\ &= \frac{1}{2^{n+2} - 2 - t} \end{aligned}$$

which gives the same result as the uniform distribution.

We are ready to state the Simulation algorithm.

Algorithm 2 Simulating the random picking process

Start at the Entrance.
Let b be either 0 or 1 with probability 1/2 each
if $b = 0$ **then**
    Take a path $\mathcal{P}$ of length n to reach a node on level $n+1$ uniformly at random (i.e., querying and moving to a unvisited neighbour n times.)
    $z \leftarrow 0$
    for $i = 1$ to $n + 1$ **do**
        if $z = 0$ **then**
            Let b_i be either 0 or 1 with probability 1/2 each
            if $b_i = 0$ **then**
                Move backward by one step, stop when reaching the ENTRANCE
            else
                Move to a neighbour that is not on $\mathcal{P}$
                $z \leftarrow 1$
        else
            Move forward to one of the two neighbours with probability 1/2 each
else
    Take a path of length $n + 1$ to reach a node on level $n + 2$ uniformly at random
    for $j = 1$ to $n + 1$ **do**
        Move forward to one of the two neighbours with probability 1/2 each, stop when detecting the EXIT
Return the last node

This algorithm runs in at most $2(n+1)$ queries. To save some queries, instead of querying nodes in every iteration, the algorithm can be implemented by generating a set of directions from $\{backward, left, right\}$ to represent the sequence of moves, then query the nodes only when necessary.

Observe when one starts at the ENTRANCE and moves along the path of length $k \in [n+1]$, one always lands on level $k+1$. This structure is quite important, and it is exploited again later in designing the deterministic algorithm.

One can easily verify (though tedious) that starting in 4 and 17 of the algorithm, the first n steps of the algorithm is exactly a random walk on $(^{(\pm)}, \mathbb{Z} \setminus \{0\})$ with node -1 being level $n+1$, and node 1 being level $n+2$. In the $(n+1)$-th step, the probability differs only on levels 1, 2, 2n - 1, 2n, since the algorithm stops once reaching the ENTRANCE or the EXIT. It has the following probability distribution

$$P(n+1-k) = P(n+2+k) = \begin{cases} \frac{1}{2^{(k+2)}} & 0 \le k \le n-2 \\ \frac{1}{2^{(n+2)}} & k = n-1 \\ \frac{3}{2^{(n+2)}} & k = n \end{cases}$$

(A Matlab code is included in the Appendix for verification purpose)

Since we always choose uniformly at random, each node on the same level gets picked with the same probability. Hence, each node on level $n+1-k$ and $n+2+k$, $k \in \{0, \ldots, n-2\}$ gets picked with probability $\frac{1}{2^{(k+2)}} \frac{1}{2^{(n+1-k-1)}} = \frac{1}{2^{n+2}}$, where $2^{(n+1-k-1)}$ is the number of nodes on level $n+1-k$ and $n+2+k$; in particular, given the node is unknown and on level $\{3, \ldots, 2n\}$, the probability of this node getting picked follows a uniform distribution. It is the same as the true conditional probability distribution, because both are uniform distributions.

Suppose the number of nodes queried is $t = o(2^{n/2})$, we show next that even if we run the algorithm $o(2^{n/2})$ many times, the probability this algorithm returns a node not meeting the condition (i.e., node is either known or on level $\in \{1, 2, 2n+1, 2n+2\}$) is negligible when n is large enough. Indeed, in each round, the probability of the node not meeting the condition is at most $\frac{t+8}{2^{(n+2)}} \le \frac{1}{2^{(n/2)}}$, then the probability this algorithm ever returns a node not meeting the condition in $o(2^{n/2})$ many trials follows a geometric

distribution with tail probability bounded below by

$$\begin{aligned}(1-\frac{1}{2^{(n/2)}})^{o(2^{n/2})} &= (1-\frac{1}{2^{(n/2)}})^{s\cdot o(2^{n/2})/s} \\ &= [(1-\frac{1}{2^{(n/2)}})^{o(2^{n/2})}]^{1/s} \\ &\geq [(1-\frac{1}{2^{(n/2)}})^{2^{n/2}}]^{1/s} \\ &\geq (\frac{1}{e})^{1/s}\end{aligned}$$

which can be arbitrarily close to 1, since we may take large $s \in \mathbb{N}$ as n gets large.

The algorithm simulates the random picking process because

- Under the true distribution, when only $o(2^{n/2})$ nodes have been queried, the probability that a randomly picked unknown node is on level $l \in \{1, 2, 2n+1, 2n+2\}$ is $O(\frac{1}{2^n})$, which is negligible when n is large enough.
- the two conditional probability distributions are the same

It remains to show the algorithm is a special case of a bigger framework, namely the Tree Embedding Process, and prove a $\Omega(2^{n/2})$ lower bound for this process.

4.2.3 Lower Bound for Tree Embedding

For any algorithm that starts at the Entrance and queries only known nodes, the oracle always returns two unknown nodes unless the algorithm detects a cycle on the Glued Tree. Hence, before the algorithm detects a cycle, we may think of the algorithm embedding a binary tree into the Glued Tree rooted at ENTRANCE. This section is devoted to show any Tree Embedding process that detects either a cycle or the EXIT with high probability must make $\Omega(2^{n/2})$ queries.

The Tree Embedding was first defined in [CCD03]. Later [FZ03] gave a $\Omega(2^{n/2})$ lower bound assuming the input size is exponentially larger than the number of nodes (which eliminates the possibility of picking a random unknown node). For completeness, we adapt and include the proof here.

For a rooted binary tree T, define an embedding of T into G to be a function π from the nodes of T to the nodes of G such that $\pi(\text{ROOT}) = \text{ENTRANCE}$ and for all nodes u and v that are neighbours in T, $\pi(u) = \pi(v)$ are neighbours in G. We say that an embedding of T is proper if $\pi(u) \neq \pi(v)$ for $u \neq v$. We say that a tree T exits under an embedding π if $\pi(v) = \text{EXIT}$ for some $v \in T$ (i.e., the embedding detects the EXIT).

Algorithm 3 Tree Embedding [CCD03]

1: Given a binary tree T, label the ROOT of T as 0, and label the other nodes of T with consecutive integers so that if node i lies on the path from the root to node j then $i < j$.
2: Set $\pi(0) = \text{ENTRANCE}$.
3: Let i and j be the neighbours of 0 in T.
4: With probability 1/2 set $\pi(i) = u$ and $\pi(j) = v$, and with probability 1/2 set $\pi(i) = v$ and $\pi(j) = u$.
5: **for** $i = 1, 2, 3, \ldots$ **do**
6: **if** node i is not a leaf, and $\pi(i)$ is not EXIT or ENTRANCE **then**
7: Let j and k denote the children of node i, and l denote the parent of node i
8: Let u and v be the neighbours of $\pi(i)$ in G other than $\pi(l)$
9: With probability 1/2 set $\pi(i) = u$ and $\pi(j) = v$, and with probability 1/2 set $\pi(i) = v$ and $\pi(j) = u$

The algorithm for simulating random picking process is merely a special case of the Tree Embedding process; indeed, in the Simulation algorithm, if $b = 0$, then the algorithm embeds a path, and if $b = 1$, the algorithm embeds either a path or a tree.

We are ready to show the proof that bounds the query complexity of Tree Embedding process from below.

Lemma 24. *[FZ03] Any Tree Embedding process, on average of all possible random cycles, can detect neither a cycle nor the EXIT with high probability in $o(2^{n/2})$ queries.*

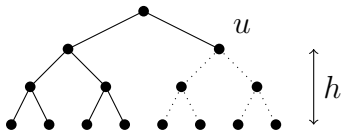
Proof. Let T be a tree with t nodes, with image $\pi(T)$ in G_n under the random embedding of π. For any nonroot node $u \in T$, let $p(u) \in T$ be the parent of u.

We assume that in G_n, the ENTRANCE is at level 1 and the EXIT is at level $2n + 2$, and the middle layer is between levels $n + 1$ and $n + 2$. Thus both binary trees have height n. To reach the EXIT from the column $n + 1$, π has to move right n times in a row, which has probability 2^{-n}. Since there are at most t tries on each path of T, and there are at most t such paths, the probability of finding the EXIT, $P(T \text{ exits on } G_n)$, is bounded by $t^2 \cdot 2^{-n}$.

Now assume that $\pi(T)$ does not detect the EXIT. It is easy to see that

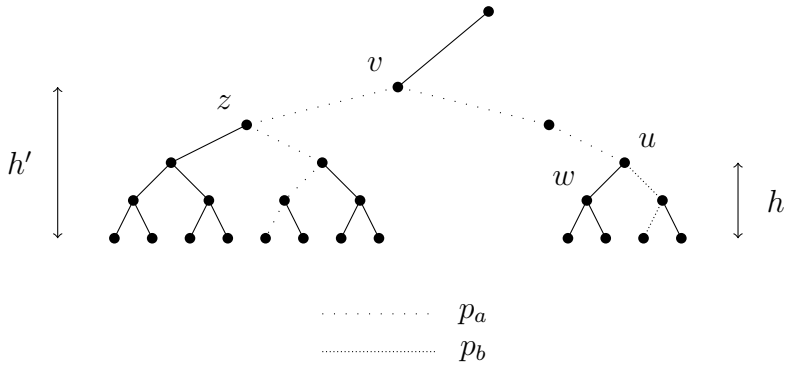
$$\begin{aligned} P(\pi \text{ is improper}) &= \sum_{a,b \in T} P[\pi(a) = \pi(b) \text{ and } \pi(p(a)) \neq \pi(p(b))] \\ &= \sum_{a,b \in T} \sum_{u \in V(G_n)} P[\pi(a) = \pi(b) = u \text{ and } \pi(p(a)) \neq \pi(p(b))] \end{aligned}$$

We calculate $P(\pi(a) = \pi(b) = u)$ first for fixed $a, b \in T$ and $u \in G_n$. Let the height of u be the distance h from u to a leaf of the subtree that u is in, i.e., if u is at level $l \leq n+1$, then $h = n + 1 - l$, and if u is at level $l \geq n + 2$, then $h = l - (n + 2)$.



Look at the two paths p_a and p_b in T from the root to a and to b, respectively. Either both paths go through the middle layer of G_n, or only one path goes through the middle layer (at least one path must go through the middle layer). Assume first that both paths go through the middle layer.

Assume WLOG that p_b goes through the middle layer at least as many times as p_a. From the last time p_a goes through the middle layer, in order to reach u it must first reach some ancestor of u, and let v be the highest ancestor [2].



[2] v is the node that is closest to the ENTRANCE if $l \leq n + 1$, or the EXIT if $l \geq n + 2$.

If v is u, then p_a and p_b each takes an edge below u (refer to the graph). There are two scenarios, namely p_a taking the left edge and p_a taking the right edge. Suppose p_a takes the left edge, then when it crosses the middle layer, it may take any leaf of the subtree that w is in, and there are at most $2^{(h-1)}$ many of them that have not been queried. Since T has t nodes, the process makes at most t queries, hence there are at least $2^n - t$ many nodes on level $n+1$ which p_a can possibly take, so the probability of p_a landing on the subtree that w is in is bounded above by $\dfrac{2^{(h-1)}}{2^n - t}$, on average of all possible random cycles.

After p_a lands on the subtree, it must go up by h levels to reach u, and the probability of choosing the edges to go up is $1/2$ for each level, hence the probability of p_a getting to u is bounded above by $2^{-h} \cdot \dfrac{2^{(h-1)}}{2^n - t} = \dfrac{1}{2(2^n - t)}$. The case is similar for p_b. Therefore, the probability of p_a and p_b meeting at u is at most $2(\dfrac{1}{2(2^n - t)})^2 = \dfrac{1}{2(2^n - t)^2}$. Note that when u is on level $n+1$ or $n+2$, then the above probability is $\dfrac{1}{(2^n - t)^2}$.

If v is not u. Let the height of v be h'. We may adopt the same argument as in the previous paragraph by considering the subtrees induced by z and u instead, then the probability of p_a reaching v and p_b reaching u are bounded above by $\dfrac{1}{2(2^n - t)}$ and $\dfrac{1}{2^n - t}$, respectively. Since the probability of p_a moving from v to u is $2^{(h-h')}$, the probability of p_a and p_b meeting at u is at most $\dfrac{2^{(h'-h)}}{2(2^n - t)^2}$. Summing over all possible v's, we get

$$\sum_{h'=h}^{n} \frac{2^{(h-h')}}{2(2^n - t)^2} = \frac{1}{2(2^n - t)^2} \sum_{m=h-n}^{0} 2^m \leq \frac{1}{(2^n - t)^2}$$

or $\dfrac{3}{2(2^n - t)^2}$ if u is on level $n+1$ or $n+2$. Summing over all possible u's, we obtain the upper bound $\dfrac{3(2^{(n+2)} - 2)}{2(2^n - t)^2}$.

Now assume that only one path, say p_b goes through the middle layer, and that $\pi(a) = \pi(b) = u$. The probability of p_a getting to u is $2^{(h-n)}$, and the probability of p_b getting to u

is at most $\frac{1}{2^n - t}$. Summing over all possible u's on the binary containing the ENTRANCE,

$$\sum_u \frac{2^{(h-n)}}{2^n - t} = \sum_{h=1}^{n} 2^{(n-h)} \frac{2^{(h-n)}}{2^n - t} = \frac{n+1}{2^n - t}$$

where the first equality follows from the fact that there are $2^{(n-h)}$ many u's with height h.

Hence,

$$\begin{aligned} P(\pi \text{ is improper}) &= \sum_{a,b \in T} P[\pi(a) = \pi(b) \text{ and } \pi(p(a)) \neq \pi(p(b))] \\ &= \sum_{a,b \in T} \sum_{u \in V(G_n)} P[\pi(a) = \pi(b) = u \text{ and } \pi(p(a)) \neq \pi(p(b))] \\ &\leq \sum_{a,b \in T} \frac{3(2^{(n+2)} - 2)}{2(2^n - t)^2} + \frac{n+1}{2^n - t} \\ &\leq \frac{t^2}{2^n - t}(\frac{3(2^{(n+2)} - 2)}{2(2^n - t)} + n + 1) \end{aligned}$$

which can be smaller than any constant probability for $t = o(2^{n/2})$ and large n.

□

Since we may simulate random picking process with $2(n+1)$ queries, we may simulate any classical algorithms making t queries by Tree Embedding process with at most $2t(n+1)$ queries. Hence, we obtain the following corollary.

Corollary 25. *Any classical algorithms that detect either a cycle or the EXIT with high probability must make $\Omega(2^{n/2}/n)$ queries.*

4.3 An $O(n2^{n/2})$ Deterministic Algorithm

The lower bound $\Omega(2^{n/2}/n)$ is almost tight as we show in this section. We present a $O(n2^{n/2})$ classical deterministic query algorithm. To overcome the randomness in the middle layer, one natural approach is to try "every possible option". We first incorporate this idea in designing the following deterministic algorithm.

Algorithm 4 The first approach

1: Start at the ENTRANCE, query all nodes on level $l \in [\lfloor 2n/3 \rfloor]$ and denote the set of nodes by U.
2: Follow any path from U to reach an arbitrary node v_0 on level $n+2$.

3: **for** $i = 1$ to $\lceil n/3 \rceil$ **do**
4: Denote the two unvisited neighbours of v_{i-1} by v_i^0 and v_i^1, respectively.
5: Form a "perfect binary tree"[3] T_i of height $\lceil n/3 \rceil + i$ rooted at v_i^0 (i.e., first query the neighbours of v_i^0 that are not v_{i-1}, then repeat the same process to the neighbours until the furthest node from v_i^0 is at distance $\lceil n/3 \rceil + i$).
6: **if** $T_i \cap U = \emptyset$ **then**
7: $v_i \leftarrow v_i^0$
8: **else**
9: $v_i \leftarrow v_i^1$

10: Form a "perfect binary tree" T of height $\lfloor 2n/3 \rfloor$ rooted at $v_{\lceil n/3 \rceil}$; the EXIT is on T.

Observe that starting at the ENTRANCE, one always lands on level $l+1$ if one takes a path of length $l \in [n+1]$. Hence, the first step can be seen as quering all paths of length $\lfloor 2n/3 \rfloor - 1$, and there are $\sum_{i=1}^{\lfloor 2n/3 \rfloor} 2^i = O(2^{2n/3})$ nodes on those paths.

For v_{i-1} on level $n+1+i$, the unvisited neighbours v_i^0 and v_i^1 are on levels $n+i$ and $n+2+i$. Steps 5 to 9 ensure we always move to the node on level $n+2+i$; indeed, suppose v_i^0 is on level $n+i$ and v_i^1 is on level $n+2+i$, then the distance between v_i^0 and U is $n+i-\lfloor 2n/3 \rfloor = \lceil n/3 \rceil + i$, whereas the distance between v_i^1 and U is $\lceil n/3 \rceil + i + 2$. Hence, a perfect binary tree rooted at v_i^0 has common element with U, but it is not the case for v_i^1, so we may distinguish between v_i^0 and v_i^2, and move to the node on level $n+2+i$.

For each $i \in [\lceil n/3 \rceil]$, at most $2^{\lceil n/3 \rceil + i}$ queries are needed to form the perfect binary tree T_i, then the total number of queries needed is

$$\sum_{i=1}^{\lceil n/3 \rceil} 2^{\lceil n/3 \rceil + i} = O(2^{2n/3}).$$

By induction, $v_{\lceil n/3 \rceil}$ is on level $n+2+\lceil n/3 \rceil$, so the EXIT is at distance $(2n+2) - (n+2+\lceil n/3 \rceil) = \lfloor 2n/3 \rfloor$ away from $v_{\lceil n/3 \rceil}$, hence is on the perfect binary tree T of height $\lfloor 2n/3 \rfloor$. The total number of queries needed to detect the EXIT is therefore $O(2^{2n/3})$.

[3] The procedure does not always produce a tree

We can actually do better than the first approach. Suppose again that v_i^0 is on level $n+i$ and v_i^1 is on level $n+2+i$, and one follows a path of length $i-1$ starting at v_i^0, then one lands on level $n+1$ definitely, whereas one lands on level $m > n+1$ if starting at v_i^1 instead. Exploiting this structure gives an $O(n2^{n/2})$ algorithm. The same algorithm was discovered independently by Shalev Ben-David and disclosed in a footnote in [AA15].

Algorithm 5 An $O(n2^{n/2})$ classical algorithm

1: Start at the ENTRANCE, query all nodes on level $l \in [\lfloor n/2 \rfloor]$ and denote the set of nodes by U.
2: Follow a random path to reach an arbitrary node v_0 on level $n+2$.

3: **for** $i = 1$ to $\lceil n/2 \rceil$ **do**
4: Denote the two unvisited neighbours of v_{i-1} by v_i^0 and v_i^1, respectively.
5: Starting at v_i^0, follow a random path of length $i-1$ to reach a node u_i.
6: Form a "perfect binary tree" T_i of height $\lceil n/2 \rceil + 1$ rooted at u_i
7: **if** $T_i \cap U = \emptyset$ **then**
8: $v_i \leftarrow v_i^0$
9: **else**
10: $v_i \leftarrow v_i^1$

11: Form a "perfect binary tree" T of height $\lfloor n/2 \rfloor$ rooted at $v_{\lceil n/2 \rceil}$; the EXIT is on T.

If v_i^0 is the node on level $n+i$, then u_i is on $n+1$, and step 6 ensures we always query a node w on level $n+1-\lceil n/2 \rceil - 1 = \lceil n/2 \rceil$, i.e., w in U. If v_i^0 is on level $n+2+i$, then the tree formed in step 6 is disjoint from U. Hence, together with steps 7 to 10, we can always identify and move to the node on a higher level. After $\lceil n/2 \rceil$ iterations, we are on level $n+2+\lceil n/2 \rceil$, which is at distance $2n+2-(n+2+\lceil n/2 \rceil) = \lfloor n/2 \rfloor$ to the EXIT, so the EXIT is discovered in step 11.

This algorithm takes $O(2^{n/2})$ queries in step 1, 6, and 11. Since there are $O(n)$ iterations, the total number of queries is $O(n2^{n/2})$.

Chapter 5

Conclusion

In this thesis, we have mainly studied the classical query complexity of detection problems on modified Hypercube and Glued Tree.

- For the search problem on the modified Hypercube $H_k^{(n)}$ where k is a constant, the classical query complexity is at most $O(\text{polylog}(N))$, and we presented some evidence that the lower bound of the quantum query complexity may increase as we add more edges to the graph.
- For the search problem on the Glued Trees with input size N, we consider a variant with input size roughly equal to the number of nodes by introducing a simulation process; we strengthened the query complexity lower bound. We also presented a classical algorithm, which shows the classical query complexity bound is tight, up to a log factor.

  In summary, the classical query complexity has an $\Omega(N^{1/2}/\log(N))$ lower bound, and an $O(\log(N)N^{1/2})$ upper bound, hence the complexity is $\Theta(N^{1/2})$, tight up to a log factor. The query complexity gap is $O(\text{polylog}(N))$ quantum queries to $\Omega(N^{1/2}/\log(N))$ classical queries.

We only examined algorithms on graphs in this thesis, but query complexity gap is not limited to just graphs. There are many other types of problems that exhibit large query complexity, and some of the gaps are even exponentially larger than the one achieved by the Glued Trees. The largest query complexity gap known so far is 1 versus N, where N is the input size; such separation is achieved by the Fourier Sampling problem, and the reader

can learn more about the topic in [AC17]. On the other hand, Aaronson and Ambainis showed in [AA15] that for any partial Boolean functions which can be solved by a quantum algorithm in k queries, where k is a constant, the most efficient classical algorithm for it has query complexity bounded above by $O(N^{1-1/2k})$.

We end this thesis with some related open problems

- For partial Boolean functions, is the query complexity gap at most $O(N^{1/2})$?
- Is there any graph reach problem that exhibits a query complexity gap of $O(\text{polylog}(N))$ versus N?

Reference

[AA15] S. Aaronson and A. Ambainis, "Forrelation: A Problem that Optimally separates Quantum from Classical Computing". Proceedings of the forty-seventh annual ACM, Pages 307-316. ACM, 2015.

[AB09] S. Arora and B. Barak, "Computational Complexity: A Modern Approach". Cambridge University Press; 2009.

[ABB16] A. Ambainis, K. Balodis, A. Belovs, T. Lee, M. Santha and J. Smotrovs, "Separations in query complexity based on pointer functions". Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, Pages 800-813. ACM, 2016.

[ABK16] S. Aaronson, S. Ben-David and R. Kothari, "Separations in query complexity using cheat sheets". Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, Pages 863-876. ACM, 2016.

[AC17] S. Aaronson and L. Chen, "Complexity-Theoretic Foundations of Quantum Supremacy Experiments". 32nd Computational Complexity Conference (CCC 2017), Pages 22:1-22:67.

[AS16] N. Alon and J. Spencer, "The probabilistic Method, Fourth Edition". Wiley, 2016.

[B98] B. Bollobas, "Modern Graph Theory". Springer-Verlag, 1998.

[BBB98] C. Bennett, E. Bernstein, G. Brassard and U. Vazirani, "Strengths and Weaknesses of Quantum Computing". SIAM Journal on Computing; Volume 26 Issue 5, Oct. 1997, Pages 1510-1523.

[BBC98] R. Beals, H. Buhrman, R. Cleve, M. Mosca and R. de Wolf, "Quantum Lower Bounds by Polynomials". Journal of the ACM; Volume 48 Issue 4, July 2001, Pages 778-797.

[Bd02] H. Buhrman and R. de Wolf, "Complexity measures and decision tree complexity : a survey". Theoretical Computer Science – Complexity and logic; Volume 288 Issue 1, 9 October 2002, Pages 21-43.

[C09] A. Childs, "On the Relationship Between Continuous- and Discrete-Time Quantum Walk". Communications in Mathematical Physics; Volume 294, Issue 2. Springer-Verlag, 2009, Pages 581-603.

[CCD03] A. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann and D. Spielman, "Exponential algorithmic speedup by a quantum walk". Proceedings of the thirty-fifth annual ACM symposium on Theory of computing. ACM, 2003, Pages 59-68.

[CGM09] R. Cleve, D. Gottesman, M. Mosca, R. Somma and D. Yonge-Mallo, "Efficient discrete-time simulations of continuous-time quantum query algorithms". Proceedings of the forty-first annual ACM symposium on Theory of computing. ACM, 2009, Pages 409-416.

[CLR09] T. Cormen, C. Leiserson, R. Rivest and C. Stein, "Introduction to Algorithms", Thrid Edition. The MIT Press, 2009.

[FZ03] S. Fenner and Y. Zhang, "A note on the classical lower bound for a quantum walk algorithm". quant-ph/0312230v1, 2003.

[HLS07] P. Hoyer, T. Lee and R. Spalek, "Negative weights make adversaries stronger". Proceedings of the thirty-ninth annual ACM symposium on Theory of computing. ACM, 2007, Pages 526-535.

[K05] J. Kempe, "Discrete Quantum Walk Hit Exponentially Faster". Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques, Pages 354-360. Springer-Verlag, 2005.

[KLM07] P. Kaye, R. Laflamme and M. Mosca, "An Introduction to Quantum Computing". Oxford University Press; 2006.

[L06] G. Lawler, "Introduction to Stochastic Processes", Second Edition. Chapman and Hall/CRC, 2006.

[LP17] R. Lyons and Y. Peres, "Probability on Trees and Networks". Cambridge University Press; 2017.

[LPW08] D. Levin, Y. Peres and E. Wilmer, "Markov Chains and Mixing Times". ACM; 1 edition, 2008.

[MR95] R. Motwani and P. Raghavan, "Randomized Algorithms". Cambridge University Press; 1 edition, 1995.

[R09] S. Ross, "Introduction to Probability Models", Tenth Edition. Academic Press, 2009.

[R14] B. Reichardt, "Span Programs are Equivalent to Quantum Query Algorithms". SIAM Journal on Computing; Volume 43 Issue 3, 2014, Pages 1206-1219.

[S94] D. Simon, "On the power of quantum computation". Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Pages 116-123.

[S97] P. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". SIAM Journal on Computing; Volume 26 Issue 5, Oct. 1997, Pages 1484-1509.

[S04] M. Szegedy, "Quantum Speed-up of Markov Chain Based Algorithms". Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science. ACM, 2004, Pages 32-41.

Appendix A

Matlab Code for Simulation

This is one of many random walks we tried on the Glued Tree. The code was written first for checking the distribution, and once we found this good candidate, we proved its validity afterwards. The input and output are specified in the comment of the function. When interpreting the output, the reader should refer to the diagram provided in chapter section 4.2.1.

A.1 Code

```
function Dist = GTRP(W_type, W, steps)
%A function for Simulating the random picking process or Computing the
%probability distribution of random walk on (\pm, Z) in # steps
%
%W_type is the walk type, it can be either 'S' which stands for Simulation, or
%anyt other string, which returns the random walk
%
%W takes input and returns the walk starting at (-, -1) if W = 'Wl', (+, 1)
%if W = 'Wr' and mix of the two with probability 1/2 each if W is any
%other string

format rat

if W_type == 'S'
```

```
    n = steps;
else
    n = steps + 2;
end

%Define Transition matrix M
M = zeros(2 * (2 * n - 1));
M(2 * n, 2 * n) = 1;
M(2 * n - 1, 2 * n - 1) = 1;

for i = (2 - 1):(n - 1)
    M(i, i + 1) = 1;
end

for i = ((n + 1) - 1):((2 * n - 1) - 1)
    M(i, i + 1) = 1/2;
    M(i, i + (2 * n) - 1) = 1/2;
end

for i = ((2 * n - 1) + 2):((2 * n - 1) + n)
    M(i, i - (2 * n - 1)) = 1/2;
    M(i, i - 1) = 1/2;
end

for i = ((2 * n - 1) + (n + 1)):(2 * (2 * n - 1))
    M(i, i - 1) = 1;
end


%Output different results
I = eye(2 * (2 * n - 1));
if W == 'Wl'
    Dist_Temp = 1/2 * I(:, (2 * n - 1) + n);
elseif W == 'Wr'
    Dist_Temp = 1/2 * I(:, n);
else
    Dist_Temp = 1/2 * (I(:, (2 * n - 1) + n) + I(:, n));
end
```

```
if W_type == 'S'
    Dist_Temp = (Dist_Temp' * M^n)';
    Dist = [[(- n : -1) (1 : n)]'...
        [0; Dist_Temp(1:2 * n - 1)] + [Dist_Temp(2 * n:2 * (2 * n - 1)); 0]];
else
    Dist_Temp = (Dist_Temp' * M^(n - 2))';
    Dist = [[(- n : -1) (1 : n)]' [0; Dist_Temp(1:2 * n - 1)]...
        [Dist_Temp(2 * n: 2 * (2 * n - 1)); 0]];
end

end
```