Evaluation of Boolean Formulas with Restricted Inputs

by

Bohua Zhan

Submitted to the Department of Physics
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Physics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author ....[illegible]............
Department of Physics
May 7, 2010

Certified by ....[illegible]............
Edward Farhi
Professor
Thesis Supervisor

Accepted by ....[illegible]............
David Pritchard
Physics Thesis Coordinator

Evaluation of Boolean Formulas with Restricted Inputs

by

Bohua Zhan

Submitted to the Department of Physics
on May 7, 2010, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Physics

Abstract

In this thesis, I will investigate the running time of quantum algorithms for evaluating boolean functions when the input is promised to satisfy certain conditions. The two quantum algorithms considered in this paper are the quantum walk algorithm for NAND trees given by Farhi and Gutmann [2], and an algorithm for more general boolean formulas based on span programs, given by Reichardt and Špalek [6]. I will show that these algorithms can run much faster on a certain set of inputs, and that there is a super-polynomial separation between the quantum algorithm and the classical lower bound on this problem. I will apply this analysis to quantum walks on decision trees, as described in [3], giving a class of decision trees that can be penetrated quickly by quantum walk but may not be efficiently searchable by classical algorithms.

Thesis Supervisor: Edward Farhi
Title: Professor

Acknowledgments

I would like to thank my advisor, Prof. Edward Farhi, for his guidance throughout this research project. This work is done in collaboration with Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Avinatan Hassidim, and Shelby Kimmel. I would not have come this far without their numerous inputs in discussions over the past year. I would also like to thank Andrew Lutomirski and Huy Ngoc Nguyen, with whom I discussed at various points during this research and received many helpful comments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Evaluation of boolean functions is a major class of problems considered in the theory of computation. In particular, we may consider those boolean functions F that can be written nicely as a tree. The leaves of the tree correspond to inputs. That is, we consider an input to F as a function associating each leaf a to some $r_a \in [0, 1]$. Then we associate each node b in the tree to some $r_b \in [0, 1]$, by requiring that $r_b = f_b(r_{b_1}, \ldots, r_{b_c})$ for every non-leaf node b in the tree with child nodes $b_1, \ldots, b_c$, where $f_b : [0, 1]^c \to [0, 1]$ is a fixed boolean function. The output of F is the value associated to the root. We may further simplify the situation by requiring each f_b to be the same function f, and that all leaves are at the same height. The function F is then fixed by fixing f and a height n. Throughout the paper we will let the height n be the number of edges in any path from the root to the leaves. Equivalently this is the number of times the function f is composed.

As examples, we will mainly consider two such boolean functions: NAND trees and 3-MAJ trees, but many of our results are more general. The function f for the NAND tree is the NAND gate, with $f(r_1, r_2) = 0$ if $r_1 = r_2 = 1$, and 1 otherwise. We may interpret a NAND tree with height n as a two-player game with exactly n turns, with the turn alternating between the two players. At each turn the player moving has two choices. An input to the function corresponds to an assignment of each end position of the game to either win or loss for the player moving. Then the value at each node in the tree corresponds to whether the player moving will win given best play, and the output of the overall function F is whether the player moving first will win with best play. For this reason NAND trees are also called game trees.

The function f for the 3-MAJ tree returns the value of a majority of its three inputs. That is, $f(r_1, r_2, r_3) = 0$ if and only if at least two of r_1, r_2, r_3 are zero.

There are several widely-used classical models of computation for constructing algorithms to evaluate these trees, and examining the cost of these algorithms. We give three examples. In all three examples the cost of evaluating a leaf (or equivalently querying an input value) is 1, and all other computations are free.

Model 1: the computation is deterministic. No mistakes are allowed. The cost is the number of evaluations maximized over all inputs.

Model 2: the computation is probabilistic. No mistakes are allowed. The cost is the expected number of evaluations on a fixed input, maximized over all inputs.

Model 3: the computation is probabilistic, and the algorithm is allowed to make mistakes with bounded probability, say less than 1/3. This means the algorithm makes mistakes with probability less than 1/3 on any given input. The cost is the number of evaluations, maximized over all inputs and choices of the algorithm.

It is clear that for Model 3, taking the maximum number of evaluations is not very different from taking the maximum over all inputs of the expected number of evaluations, since the number of evaluations can exceed the expected number by a large factor only infrequently, so the algorithm can afford to guess randomly if this happens and only add a small value to its probability of error. When comparing to quantum algorithms that can make errors with small probability, Model 3 appears to be the most appropriate.

For the NAND tree, the obvious algorithm is shown to be the best classically, with cost $O(N^{\log_2[(1+\sqrt{33})/4]}) = O(N^{0.754})$ [7], where $N = 2^n$ is the number of inputs. However, there are quantum algorithms, first devised by Farhi, Goldstone, and Gutmann, that can evaluate a NAND tree in time $O(\sqrt{N})$ [2]. The original algorithm, which will be described below, is in the quantum walk model. It is written in the standard query model by Ambainis in [1], still with $O(\sqrt{N})$ running time.

The classical lower bound for evaluating 3-MAJ trees is still unknown. In particular, the obvious algorithm is known to be suboptimal. Reichardt and Špalek [6], using a concept called "span programs", first gave a quantum algorithm with running time $O(2^n)$. They also showed in the same paper that this is the best possible. It is faster than the currently best lower bound of $\Omega((7/3)^n)$ for classical algorithms.

In this paper, we will consider the problem of evaluating boolean functions when its inputs are restricted to a certain set. We will show that the quantum algorithm can be much faster on this set of inputs, and that there is a super-polynomial separation between the cost of classical and quantum algorithms for this problem. This investigation originated from attempts to find on which inputs can the quantum algorithm for NAND trees run quickly, and which decision trees can be penetrated quickly by a quantum walk (see [3] and below). However, it can be viewed with more generality in the context of span programs, which we will do here.

In the next two sections, we will describe the algorithm given in [2] for NAND trees, and the more general algorithm given in [6] using span programs. In chapter 2, we will describe the restriction on inputs and analyze the cost of quantum algorithms for the new problem. In chapter 3, we will analyze classical algorithms on this problem, deriving a lower bound on the cost of any algorithm in Model 3 above. In chapter 4, we give an application of our results to quantum walk through decision trees. We conclude in chapter 5 with possible directions of future research.

1.1 Evaluation of NAND Trees

In this section we will describe the algorithm given in [2] for evaluating NAND trees. The algorithm is given in the quantum walk model. In this model, each case of the problem is associated with a graph. This is then associated with a quantum system, where each node in the graph corresponds to a vector in the standard basis, and the Hamiltonian H is directly read from the graph. In this section we will let H be the adjacency matrix of the graph. Afterwards we will consider weighted graphs (that is, every edge (i, j) is given a weight v_{ij}), and we let $H_{ij} = v_{ij}$ if there is an edge (i, j), and $H_{ij} = 0$ otherwise.

After initializing the system to a certain state, the system evolves according to the Hamiltonian for a certain time t, then the answer to the problem is found by a measurement on the system. The equations of motion bear some similarity to that of the classical random walk through a graph, hence the name quantum walk.

The algorithm in the quantum walk model is as follows. Consider the NAND tree as a graph. For each leaf with value 1, add another node to the graph and connect it only to that leaf. Next, make a 'runway' consisting of a line of $2M+1$ nodes numbered $-M$ to M, where M is a large integer to be determined. The final graph is formed by joining the root of the binary tree (denoted by $|\text{root}\rangle$) to the node numbered 0 (denoted by $|r=0\rangle$) on the runway. See Figure 1-1 for an example.

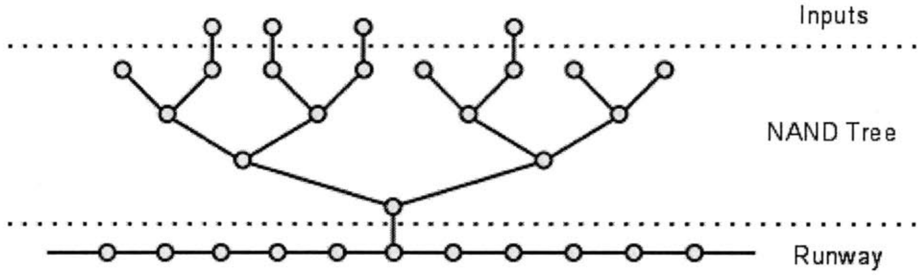

Figure 1-1: Graph corresponding to a NAND tree of height 3.

The initial state is a wave packet on the left branch of the runway, moving to the right and having a narrow peak in the energy spectrum at $E=0$. As the system evolves, part of the packet will transmit through $|r=0\rangle$, and the other part will be reflected back. The following equation relating the transmission coefficient to a parameter of the NAND tree is shown in [2].

$$T(E)=\frac{2i\sin\theta}{2i\sin\theta+y(E)}, \tag{1.1}$$

where

$$E=-2\cos\theta,\quad y(E)=\frac{\langle\text{root}|E\rangle}{\langle r=0|E\rangle},$$

and $|E\rangle$ is the exact eigenstate of the system with energy E. Here we assume that the runway is essentially infinite. Given any $\langle r=0|E\rangle$, one can see by counting equations and unknowns that for generic values of E, the equation $H|E\rangle = E|E\rangle$ fixes $\langle b|E\rangle$ for every node b in the tree (that is, above $|r=0\rangle$). Moreover the value $\langle \text{root}|E\rangle$ fixed this way is proportional to $\langle r=0|E\rangle$. So the quantity $y(E)$ can be defined even when the runway is finite.

The function $y(E)$ can be computed as follows. First, for every node b in the tree, let b' be the parent of b (the parent of $|\text{root}\rangle$ is $|r=0\rangle$). Define

$$y_b(E) = \frac{\langle b|E\rangle}{\langle b'|E\rangle}.$$

Then if b_1, b_2 are child nodes of b, the equation $H|E\rangle = E|E\rangle$ turns into the recursive relation

$$y_b(E) = \frac{1}{E - y_{b_1}(E) - y_{b_2}(E)}$$

and $y_b(E) = 1/E$ if b is a leaf.

Using this recursive relation, one can compute $y(E) = y_{\text{root}}(E)$. By induction from the leaves to the root, one can show that $y_b(E) \to 0$ as $E \to 0$ if b has value 1, and $y_b(E) \to \pm\infty$ as $E \to 0$ otherwise. In particular the behavior of $y(E)$ as $E \to 0$ indicates the value of the root. Using equation 1.1, one can see that $T(E) \approx 0$ at $E \approx 0$ if the root has value 0, and $T(E) \approx 1$ otherwise. Thus we can obtain the value at the root by measuring how much of the initial wave packet transmitted through $|r=0\rangle$. In order for this to work, we need $|y(E)|$ to be far from 1 for the range of E in the packet (either near zero or very large depending on the value of the root). From the range of E around 0 for which this is true, we can find how narrow the wave packet needs to be in energy space, which in turn gives the size of the packet in position space, which gives the length of the runway and evolution time required. So the cost of the algorithm is inverse the size of range of E for which $y(E)$ is far from 1. It is shown in [2] (by a method similar to the one described in the next section) that the size of this range is at least $\frac{1}{16\sqrt{N}}$, where $N = 2^n$ is the number of leaves. The cost is therefore $O(\sqrt{N})$, which is shown in the paper to be the best possible.

1.2 Span Programs

The use of span programs in quantum algorithms is introduced by Reichardt and Špalek in [6]. We will discuss in detail only the algorithm and the analysis of its running time, for the special case where each input is used only once. This is sufficient for the rest of the paper since it includes the case of NAND and 3-MAJ trees. We will motivate the construction from the point of view of quantum walks.

To generalize the algorithm in the last section to evaluate other boolean formulas, in particular those that can be written as a tree with boolean function f at each node, one can try to construct subgraphs, or 'gadgets', with the following properties. Let c be the number of inputs to f. The gadget should have $c + 1$ outgoing edges, each connected to a different vertex inside the gadget. This is shown schematically in Figure 1-2.

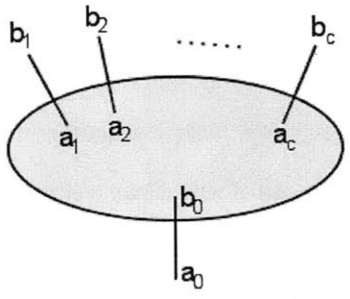

Figure 1-2: Schematic diagram of a gadget.

The edge shown at the bottom of the figure corresponds to the output x_0, while the c edges at the top are inputs $x_1, \ldots, x_c$. To each input and output, we can define

$$y_i(E) = \frac{\langle b_i|E\rangle}{\langle a_i|E\rangle}, \quad 0 \leq i \leq c \tag{1.2}$$

similar to before. The function y_i indicates the value x_i in the sense that we require

$$\text{as } E \to 0, \qquad \begin{aligned} y_i(E) &\to 0 \quad \text{if } x_i = 1, \\ y_i(E) &\to \pm\infty \quad \text{if } x_i = 0. \end{aligned} \tag{1.3}$$

Once $y_i(E), 1 \leq i \leq c$ and $\langle a_0|E\rangle$ are fixed, by counting equations and unknowns one can see that for generic E the vector $|E\rangle$ is uniquely determined inside the gadget by the equation $H|E\rangle = E|E\rangle$. Moreover the value of $\langle b_0|E\rangle$ thus determined is proportional to $\langle a_0|E\rangle$. So $y_0(E)$ is a function of $y_i(E), 1 \leq i \leq c$. For the gadget to correspond to a boolean function $f : [0,1]^c \rightarrow [0,1]$, we require that given $y_i(E), 1 \leq i \leq c$ satisfying condition 1.3, then $y_0(E) \rightarrow 0$ also does with $x_0 = f(x_1, \ldots, x_c)$.

We can join these gadgets together to evaluate a function F as follows: let each non-leaf node b in the tree correspond to a gadget P_b, and if node b is the i'th child node of b' in the original tree, then identify the edge (a_0, b_0) in P_b with the edge (a_i, b_i) in P'_b. The leaves of the original tree just correspond to the edges (a_i, b_i) in the last level of gadgets. An input to F amounts to removing some of the edges (a_i, b_i) corresponding to leaves. For consistency, the edge (a_i, b_i) is kept if and only if the input x_i is 0. Then $y_i(E) = 1/E$ if $x_i = 0$ and $y_i(E) = 0$ otherwise, so condition 1.3 is satisfied. By induction $y(E)$ at the root indicates the output of F.

In order to have some bound on the running time of the algorithm, we can set further conditions on $y_0(E)$ as a function of $y_i(E), 1 \leq i \leq c$. Define s_i such that:

$$s_i(E) = \frac{y_i(E)}{E} \text{ if } x_i = 1 \text{ and } s_i(E) = \frac{1}{y_i(E)E} \text{ if } x_i = 0. \tag{1.4}$$

Roughly speaking, the smaller the values of s_i, the slower $y_i(E)$ approaches order unity from either 0 or $\pm\infty$. We would like a condition such as $|s_0(E)|$ is at most a constant multiple of $s = \max_{1 \leq i \leq c} |s_i(E)|$.

The paper [6] gives a construction of such a gadget based on span programs. A span program P corresponding to a boolean function $f : [0,1]^c \rightarrow [0,1]$ consists of a *target vector* t and *input vectors* $v_i, 1 \leq i \leq c$ such that $f(x_1, \ldots, x_c) = 1$ if and only if t is a linear combination of v_i, where the coefficient of v_i must be zero if $x_i = 0$. Here the vector space is over $\mathbb{C}$, although different fields are also used. For example, a span program for the 3-MAJ function is:

$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, v_I = \begin{pmatrix} \alpha & \alpha & \alpha \\ 1 & \omega & \omega^2 \end{pmatrix},$$

where $\alpha = \frac{1}{\sqrt{3}}$ and $\omega = e^{2\pi i/3}$. The above notation means $v_i, 1 \leq i \leq 3$ are the columns of the matrix v_I.

It is clear that by a linear transformation on both t and v_i, we can always make $t = (1, 0)^T$. The gadget is then constructed from v_i as follows: the nodes in the gadget include $b_0, a_1, \dots, a_c$ as shown in the figure, as well as auxillary nodes $b'_1, \dots, b'_d$, where $d + 1$ is the dimension of the vector space. The weight on each edge $(a_i, b_i), 0 \leq i \leq c$ is 1. The node b_0 is connected to each a_i with weight $(v_I)_{1i}$, and the nodes b'_j are connected to nodes a_i with weight $(v_I)_{j+1,i}$. Figure 1-3 shows the gadget from the span program for the 3-MAJ gate shown above.

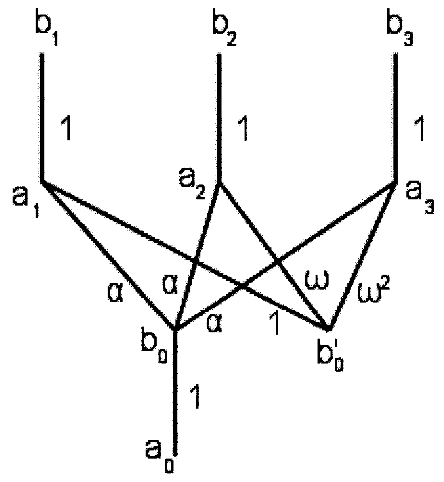

Figure 1-3: A gadget for the 3-MAJ gate.

First, we have the following.

Theorem 1.2.1 ([6], Thm. 2.5). *Consider the gadget constructed as above from a span program P corresponding to a boolean function f. Define $y_i(E), 0 \leq i \leq c$ as in equation 1.2. Then $y_i(E), 1 \leq i \leq c$ satisfying condition 1.3 implies $y_0(E)$ satisfy the same condition, with $x_0 = f(x_1, \dots, x_c)$.*

To give a bound on $|s_0|$ in terms of $s = \max_{1\leq i\leq c} |s_i|$, we need to use the concept of witness size. A *witness* can be considered as a certificate for whether the span program has a solution. Let V be the span of all $|v_i\rangle$ such that $x_i = 1$. If $f(x_1, \dots, x_c)$, then $t \in V$, so there exists a vector $|w\rangle$ such that $w_i = 0$ for all $x_i = 0$ and $v_I|w\rangle = |t\rangle$. Otherwise, $t \notin V$ implies the projection of $|t\rangle$ onto the complement of V is nonzero. So there is a vector w' such that $\langle t|w'\rangle = 1$ and $\langle v_i|w'\rangle = 0$ for all $x_i = 1$. We call either $|w\rangle$ or $|w'\rangle$ a witness for P and x_i.

Define $s_i(E)$ as in equation 1.4. Let $s_i = \sup_{E\in(0,E_0)} |s_i(E)|$ for some chosen E_0. Let $\tilde{z}$ be the diagonal matrix with s_i on the diagonal. Then the witness size is defined as the following:

Definition 1.2.2 ([6], Def. 3.6). *Let P be a span program corresponding to function f. If $x_0 = f(x) = f(x_1, \dots, x_c) = 1$, then wsize$(P, x)$ equals $||\tilde{z}^{1/2}|w\rangle||^2$, minimized over all witness $|w\rangle$ for $x_0 = 1$. Otherwise, wsize(P, x) equals $||\tilde{z}^{1/2}v_I^T|w'\rangle||^2$, minimized over all witness $|w'\rangle$ for $x_0 = 0$.*

The significance of the witness size is shown in the following theorem.

Theorem 1.2.3 ([6], Def 3.3, Thm. 3.7). *Let P be a span program corresponding to f, and define $y_i(E), s_i(E)$ and s_i as above. Then there exist constants c_1 and c_2 such that*

$$|s_0| \leq c_1 + wsize(P, x)(1 + c_2 E_0 \max_{1\leq i\leq c} s_i) \tag{1.5}$$

for each input x and maximum energy $E_0 > 0$ such that $E_0 \max_{1\leq i\leq c} s_i < \epsilon$ for some small ϵ determined in the proof.

Currently the witness size of a gate depends on the values of s_i for inputs. However, we can extract a less informative, but conceptually simpler quantity that measures the intrinsic difficulty of a gate on a given input. It is clear from the definition that witness size is an increasing function on each s_i. Let $W(P, x) = \min\langle w|w\rangle$ or $\min\langle w'|w'\rangle$ depending on whether $f(x) = 1$ or 0 (minimum taken over all witnesses). Then the following immediately comes Definition 1.2.2:

$$\text{wsize}(P, x) \leq (\max_{1\leq i\leq c} s_i) W(P, x). \tag{1.6}$$

Combining equation 1.6 and 1.5, we have the following equation which we will use in the next chapter. Let $s' = \max_{1 \leq i \leq c} s_i$, then

$$|s_0| \leq c_1 + s'W(P,x)(1 + c_2 E_0 s'). \tag{1.7}$$

The constants $W(P,x)$ can be explicitly calculated. For the span program P constructed above for the majority gate, $W(P,x) = 1$ if $x_1 = x_2 = x_3$ and $W(P,x) = 2$ otherwise. In particular the span program above is optimized so that $\max_x W(P,x) = 2$ is the smallest possible. For the NAND gate, one can construct a span program for the AND gate, then append single edges to achieve the effect of a NOT gate. The result is the NAND tree given in the previous section with two single edges inserted between each level. The effect of these two edges cancel out, giving the original NAND tree (in particular single edges do not increase $y(E)$ by much). For one possible span program for the AND gate which we will use later, we have $W(P,x) = 2$ if $x_1 \neq x_2$ and 1 otherwise. It is also possible to construct an AND gate with $\max_x W(P,x) = \sqrt{2}$, which gives the optimal running time for arbitrary inputs.

We will use Equation 1.7 and these values of $W(P,x)$ to give upper bounds on the running time of the quantum algorithm in the next section. However, we will also give explicit calculations to provide more intuition, as well as verifying the values of $W(P,x)$ given here.

From the results summarized above, Reichardt and Špalek gave an $O(2^n)$ algorithm for the 3-MAJ tree, as well as similar algorithms for other gates. Moreover, they showed that for a large class of trees, including trees of uniform height made of a single boolean function on at most three variables, the algorithm given by the span program is optimal. In later papers, Reichardt extended this optimality result to essentially all boolean formulas, see [5] and [4] for details.

Chapter 2

Analysis of Quantum Algorithm

In the previous chapter, we described how to construct quantum algorithms to evaluate a boolean function F that can be written as a tree with function f at each node, given a span program corresponding to f. We noted that the running time required by the algorithm depends on the behavior of the function $y(E)$, in particular the functions $s(E)$ defined in 1.4.

In this chapter we will describe in detail the calculation of $s(E)$. In [6], the focus is on the maximum of $s(E)$ over all inputs. Here we will focus on how it varies with the actual input, and for which inputs will it be small.

In section 2.1, we will use equation 1.7 from last chapter to obtain a general result applying to any span programs. In section 2.2, we give concrete calculations for the NAND tree and 3-MAJ tree, verifying the values of $W(P, x)$ as well as equation 1.7 in these cases. In section 2.3, we will give another way to see how the results for NAND trees can be obtained.

2.1 General Case

The general result is the following.

Theorem 2.1.1 (cf. [6] Thm. 4.16). *Let P be a span program corresponding to function f. Let F be a boolean function that can be written as a tree with f at each node. Let n be the height of the tree. Construct the quantum algorithm as before and* *let s be the value of s_0 at the root. Suppose $W(P,x) \geq 1$ for all x. Then for a given input x to F, we have $s = O(nW_T)$ with $E_0 = O(n^{-2}W_T^{-1})$ where*

$$W_T = \max_{paths\ \chi} \prod_{b \in \chi} W(P, x_{(b)})) \tag{2.1}$$

where the maximum is taken over all paths from the root to the leaves, and $x_{(b)}$ is the input at the node b.

Proof. If we disregard the c_1 and c_2 terms in equation 1.7, then at every node of the tree representing F, the value of s_0 is the maximum over all paths of the product of s_i at the leaf and the difficulties $W(P, x_{(b)})$ at the gates along the path. The value of s_i at the leaves is at most 1. This gives $s \leq W_T$ if $c_1 = c_2 = 0$.

Since we are giving an upper bound we may take $c_1, c_2 \geq 0$. Then s is the maximum over all paths from root to the leaves of a sequence of additions by $c_1 \geq 0$ and multiplications by $W(P,x)(1+c_2E_0s_b) \geq 1$. For any such sequence, we can always put all additions before all multiplications, without lowering the resulting value. The addition part gives $1 + nc_1$ (where 1 comes from the initial s_i at the leaves). The multiplication part for path χ is

$$\prod_{b \in \chi} W(P, x_{(b)})(1 + c_2E_0s_b) \leq \prod_{b \in \chi} W(P, x_{(b)})(1 + c_2E_0s).$$

By taking $E_0 = 1/(c_2ns)$, we can make the product to be at most $e\prod W(P, x_{(b)})$. Taking the maximum over all paths, we find

$$s \leq (1 + nc_1) \cdot (eW_T) = O(nW_T).$$

From this we obtain $E_0 = O(n^{-2}W_T^{-1})$. □

If for some span program P, there are inputs $x = (x_1, \dots, x_c)$ and $y = (y_1, \dots, y_c)$ such that $f(x) \neq f(y)$ and $W(P,x) = W(P,y) = 1$, then there are inputs where $W(P, x_{(b)}) = 1$ for most of the nodes b in the tree. In particular, we can consider trees such that along every path from root to the leaves, the number of nodes b on the path where $W(P, x_{(b)}) \neq 1$ is at most k, where $k \ll n$ is fixed. This motivates the following definition.

Definition 2.1.2. *Let P be a span program corresponding to a function f. We say an input $x = (x_1, \ldots, x_c)$ is trivial if $W(P, x) = 1$, and non-trivial if otherwise. For a given k, we say an input x to the overall function F satisfies the k-faults rule if along any path from the root to the leaves in the tree corresponding to F, there are at most k nodes b where the input $x_{(b)}$ at b is non-trivial. Often we will just say that the tree corresponding to F satisfies the k-faults rule.*

The following is immediate:

Corollary 2.1.3. *Consider P, f, F and s as before. If input x satisfies the k-faults rule, then $s = O(n \cdot W(P)^k)$ with $E_0 = O(n^{-2} \cdot W(P)^{-k})$. So the running time of the quantum algorithm can be set as $O(n^2 W(P)^k)$.*

Proof. The statements for s and E_0 follow immediately from Theorem 2.1.1 and Definition 2.1.2. As observed in the last chapter, the shortest running time for which the quantum algorithm makes errors with small probability is inverse proportional to the size of the range of E where $y(E)$ is far from 1. For $|E| < E_0$, we have $y(E) \leq sE = O(n^{-1})$ if the output is 1, and $y(E) \geq 1/(sE) = O(n)$ otherwise. So $y(E)$ is far from 1 within a range of size E_0. □

If we take $k = a \log n$ for some constant a, then the running time becomes $n^2 W(P)^{a \log n} = n^{2 + a \log W(P)}$, which is polynomial in n. In the next chapter, we will give, for a class of boolean functions f that includes NAND and 3-MAJ, a lower bound of $\Omega((\log n/d_1)^{k/d_2}) = n^{a(\log\log n - \log d_1)/d_2}$ evaluations for classical algorithms, where d_1, d_2 are constants. This gives the superpolynomial separation between classical and quantum algorithms on evaluating F for inputs satisfying the k-faults rule.

Note the conclusion in corollary 2.1.3 does not depend on the condition that the leaves of the original NAND tree are all at the same height. The corollary applies to the evaluation of value at root of any tree where the leaves have value 0 and internal nodes are assigned value by the extended NAND rule - the usual NAND rules together with negating the value at single edges. Since single edges do not increase $y(E)$ by much, the parent of that edge is not considered a fault. We will make use of this observation in chapter 4.

2.2 Two Specific Cases

In this section, we will give detailed computations for the NAND gate and the 3-MAJ gate, verifying the result in the previous section. Since the result is proven in the previous section, we will not be entirely rigorous in this section.

For the NAND gate, we will simply analyze the algorithm described in section 1.1. Recall that we associated the function $y_b(E)$ to each node b (instead of each edge as for span programs). This is defined by $y_b(E) = \langle b|E\rangle / \langle b'|E\rangle$ where b' is the parent of b, and the recursion relation is:

$$y_b(E) = \frac{1}{E - y_{b_1}(E) - y_{b_2}(E)}$$

where b_1, b_2 are child nodes of b, and $y_b(E) = 1/E$ if b is a leaf. It is clear that for the recursion relation, $y(E)$ increases with both $y_{b_1}(E)$ and $y_{b_2}(E)$ on each side of the pole $y_{b_1}(E) + y_{b_2}(E) = E$.

For energies $|E| \leq E_0$, where E_0 is to be determined later, we will verify that $0 \geq y_b(E) \geq -s_b E$ if the value at node b is 1, and that $y_b(E) \geq 1/(2s_b E)$ otherwise (we will see soon the purpose of the factor of 2). Initially this is true with $s_b \leq 1$. Consider a node b with child nodes b_1 and b_2. Let $s = s_b$ and $s' = \max(s_{b_1}, s_{b_2})$. Then y_{b_i} satisfies the conditions for s' if it does for s_{b_i}. So we may replace each s_{b_i} by s'. There are three cases for the recursion:

Case 1: $v(b_1) = v(b_2) = 0$. From $y_{b_i}(E) \geq 1/(2s'E), i = 1, 2$ we have

$$y_b(E) \geq \frac{1}{E - \frac{1}{2s'E} - \frac{1}{2s'E}} = -\frac{s'E}{1 - s'E^2} = -s'E(1 + O(s'E^2)).$$

We assumed that $y_{b_i}(E) \gg E$ which should be valid for small E. So we may take $s = s'(1+O(s'E^2))$. This verifies the equation analogous to 1.7 with $W(P, (0,0)) = 1$.

Case 2: $v(b_1) = v(b_2) = 1$. From $0 \geq y_{b_i}(E) \geq -s'E$ we have

$$y_b(E) \geq \frac{1}{E + s'E + s'E} = \frac{1}{2(s' + \frac{1}{2})E}.$$

So we may take $s = s' + 1/2$, verifying $W(P, (1,1)) = 1$.

Case 3: $v(b_1) = 0, v(b_2) = 1$. So

$$y_b(E) \geq \frac{1}{E + s'E - \frac{1}{2s'E}} = -\frac{2s'E}{1 - 2s'E^2 - 2s'^2E^2} = -2s'E(1 + O(s'E)).$$

So we may take $s = 2s'(1 + O(s'E))$, verifying $W(P, (0,1)) = W(P, (1,0)) = 2$. Note that we do need a condition on s_{b_1}, since if it is too large, the term $y_{b_1}(E)$ may dominate $y_{b_2}(E)$ for the size of E we are considering here.

Now we consider the 3-MAJ gate. For a given node b in the tree with child nodes $b_i, i = 1, 2, 3$, we may again consider $y_b(E)$ as a function of $y_{b_i}(E)$ for $i = 1, 2, 3$. Now it is more difficult to show that this function is increasing in each $y_{b_i}(E)$. We will assume that this is the case. So if $s' = \max(s_{b_1}, s_{b_2}, s_{b_3})$, then we may replace each $y_{b_i}(E)$ by either $-s'E$ or $1/(s'E)$.

The gadget for the 3-MAJ gate is shown in Figure 1-3. By abuse of notation, we will let a_0 also represent $\langle a_0 | E \rangle$, and similarly for other nodes. Reading from the figure, we see that the equation $H|E\rangle = E|E\rangle$ becomes:

$$\begin{aligned}
Eb_0 &= a_0 + \alpha(a_1 + a_2 + a_3) \\
Eb_0' &= a_1 + \omega a_2 + \omega^3 a_3 \\
Ea_1 &= \alpha b_0 + b_0' + b_1 \\
Ea_2 &= \alpha b_0 + \omega b_0' + b_2 \\
Ea_3 &= \alpha b_0 + \omega^2 b_0' + b_3
\end{aligned}$$

If we set $a_0 = -1$, and substituting in $b_i = a_i y_i(E), 1 \leq i \leq 3$, we get the following system of equations:

$$\begin{pmatrix}1\\0\\0\\0\\0\end{pmatrix} = \begin{pmatrix} -E & 0 & \alpha & \alpha & \alpha \\ 0 & -E & 1 & \omega & \omega^2 \\ \alpha & 1 & -E+y_{b_1}(E) & 0 & 0 \\ \alpha & \omega & 0 & -E+y_{b_2}(E) & 0 \\ \alpha & \omega^2 & 0 & 0 & -E+y_{b_3}(E) \end{pmatrix} \begin{pmatrix} b_0 \\ b_0' \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

It is clear how to generalize this to other span programs. For generic E this has a unique solution. Then $y_b(E) = b_0/a_0 = -b_0$. For the case $x_1 = x_2 = x_3 = 1$, we may substitute in $y_{b_i}(E) = -s'E$. The result is

$$y_b(E) = -b_0 = -\frac{s'E+E}{1-s'E^2-E^2} = -(s'+1)E(1+O(s'E^2)).$$

So we may take $s = (s'+1)(1+O(s'E^2))$, verifying equation 1.7 with $W(P,(1,1,1)) = 1$. Similarly we can compute:

$$\begin{aligned} x_1 = x_2 = x_3 = 0: &\quad y_b(E) = \tfrac{1-s'E^2}{(s'+1)E-s'E^3} = \tfrac{1+O(s'E^2)}{(s'+1)E}. \\ x_1 = 0, x_2 = x_3 = 1: &\quad y_b(E) = -\tfrac{6(s'+1)E+O(s'E)}{3+O(s'E)} = -2(s'+1)E(1+O(s'E)). \\ x_1 = x_2 = 0, x_3 = 1: &\quad y_b(E) = \tfrac{3+O(s'E)}{6s'E+O(E)} = \tfrac{1+O(s'E)}{2s'E}. \end{aligned}$$

which verifies $W(P,(0,0,0)) = 1$ and $W(P(0,1,1)) = W(P(0,0,1)) = 2$.

2.3 Another Viewpoint

In this section we will focus on NAND trees as described in section 1.1. We will give the expansion of $y(E)$ around $E = 0$, and another way to view the result in the previous two sections. While this approach does not yet yield any rigorous proofs, it may give additional intuition about the situation, especially when considering for what additional trees may the value of $y(E)$ be small. Indeed this is the method by which the constraint on the input is first arrived at.

Recall the construction of the quantum system given in section 1.1. Let H be the Hamiltonian of the entire system S, and H' be the Hamiltonian of the system S' derived from a graph consisting of only the tree (including leaves added for the input). Let $|E\rangle$ be an eigenstate of H with energy E, and let $|E'\rangle$ be the vector obtained by restricting $|E\rangle$ to S'. Then $|E'\rangle$ satisfies the equation $H'|E'\rangle = E|E'\rangle$ everywhere except at the root. For the root, an extra term is needed to account for the $|r=0\rangle$ neighbor of the root. These can be summarized as:

$$\langle r=0|E\rangle|\text{root}\rangle + H'|E'\rangle = E|E'\rangle.$$

Now let $|\chi_i\rangle$ be the energy eigenstates of H'. Expanding the equation above in terms of these eigenstates, we have:

$$\sum_i \langle r=0|E\rangle \cdot a_i|\chi_i\rangle + \lambda_i c_i|\chi_i\rangle = \sum_i E c_i|\chi_i\rangle.$$

where λ_i is the eigenvalue of $|\chi_i\rangle$, and a_i, c_i are coefficients of $|\text{root}\rangle$ and $|E'\rangle$ respectively when expanded in terms of the eigenstates $|\chi_i\rangle$. Since $|\chi_i\rangle$ form a basis, the equation holds for each i. Rearranging, we get

$$c_i = \langle r=0|E\rangle \frac{a_i}{E-\lambda_i}.$$

Substitute into $|E'\rangle = \sum_i c_i|\chi_i\rangle$, we get

$$|E'\rangle = \langle r=0|E\rangle \sum_i \frac{a_i}{E-\lambda_i}|\chi_i\rangle.$$

Taking the coefficient at the root:

$$y(E) = \frac{\langle \text{root}|E\rangle}{\langle r=0|E\rangle} = \frac{\langle \text{root}|E'\rangle}{\langle r=0|E\rangle} = \sum_i \frac{a_i}{E-\lambda_i}\langle \text{root}|\chi_i\rangle = \sum_i \frac{|a_i|^2}{E-\lambda_i}, \qquad (2.2)$$

where we used $\langle \text{root}|\chi_i\rangle = a_i^*$. We are most concerned about the behavior of $y(E)$ when expanded around $E=0$. Therefore we make such an expansion:

$$y(E) = \sum_i \frac{|a_i|^2}{E-\lambda_i} = -\sum_i \frac{|a_i|^2/\lambda_i}{1-E/\lambda_i} = -\sum_i \frac{|a_i|^2}{\lambda_i}(1+\frac{E}{\lambda_i}+\frac{E^2}{\lambda_i^2}+\dots).$$

First, we need to explain what happens when $\lambda_i = 0$ for some i. If $|a_i|^2$ for one such i is nonzero, then $y(E)$ has a simple pole at $E = 0$. This corresponds to the case where the root has value 0, and $y(E) \to \pm\infty$. On the other hand, if the root has value 1, then we must have $|a_i|^2 = 0$, or $a_i = 0$, for each i such that $\lambda_i = 0$. We will focus on the case where the root has value 1, so we can ignore the part of the sum where $\lambda_i = 0$.

Now we need to interpret the coefficient of E^n in this sum. We will now write H instead of H' for the Hamiltonian corresponding to the tree alone, since this will become our primary concern. Let H^{-1} denote the pseudoinverse of H. That is, the operator satisfying $H^{-1}|\chi_i\rangle = \lambda_i^{-1}|\chi_i\rangle$ if $\lambda_i \neq 0$, and $H^{-1}|\chi_i\rangle = 0$ otherwise. Let H^{-n} denote H^{-1} raised to n'th power. We can compute:

$$\langle\text{root}|H^{-n}|\text{root}\rangle = \sum_{i,j}\langle\text{root}|\chi_i\rangle\langle\chi_i|H^{-n}|\chi_j\rangle\langle\chi_j|\text{root}\rangle = \sum_{i,\lambda_i\neq 0}|a_i|^2\lambda_i^{-n}$$

Substitute in, we obtain:

$$y(E) = -\sum_{n\geq 0}\langle\text{root}|H^{-n-1}|\text{root}\rangle E^n. \tag{2.3}$$

It is easily seen that $H^{-n}|\text{root}\rangle$ is the vector with the smallest norm among all vectors $|A\rangle$ such that $H^n|A\rangle = |\text{root}\rangle$. In our case the situation can be simplified further because our graph is *bipartite*. That is, its nodes can be partitioned into two parts such that each edge connects two nodes from different parts. The vector $|\text{root}\rangle$ contains coefficients only in one of the two parts. If $|A\rangle = H^{-1}|\text{root}\rangle$ contains coefficients in both parts, then we can set its coefficients in the part containing the root to zero. This will not change $H|A\rangle$ since this will just set the coefficients of that vector in the parts not containing the root to zero, which they must be in the first place. Since $|A\rangle$ has the property that norm is minimized, we can conclude that $|A\rangle$ contains coefficients only in the part not containing the root. By induction, we know $|A\rangle = H^{-n}|\text{root}\rangle$ contains coefficients only in the part containing the root if n is even, and only in the other part if n is odd. This shows $\langle\text{root}|H^{-2n-1}|\text{root}\rangle = 0$. Using the hermiticity of H^{-1}, which follows from that of H, the above equation can be simplified to give our main result:

$$y(E) = -\sum_{n\geq 1}\langle\text{root}|H^{-2n}|\text{root}\rangle E^{2n-1} = -\sum_{n\geq 1}|H^{-n}|\text{root}\rangle|^2 E^{2n-1}.$$

From the equation, we see that the rate of growth of $y(E)$ around $E = 0$ can be associated with the norm of the vectors $H^{-n}|\text{root}\rangle$. Another way to see this is to go back to equation (2.3). The location of the pole of $y(E)$ closest to $E = 0$ gives a good indication of when $y(E)$ increases to unity. This is the smallest nonzero eigenvalue of H, which becomes the largest eigenvalue of H^{-1}, which in general will measure how fast the norms of $H^{-n}|\text{root}\rangle$ increase as n increases. The fact that the running time depends on the size of the smallest positive eigenvalue is a reflection of the possible interpretation of the quantum algorithm as a spectral analysis around $E = 0$, see [6] for details.

One way to estimate $y(E)$ around $E = 0$ is to directly estimate the largest eigenvalue of H^{-1}. In this section, however, we will just focus on estimating $a = |H^{-1}|\text{root}\rangle|^2$. In many cases $y(E)$ will increase to order unity around $E = a^{-1}$, but sometimes a can be small and $|H^{-n}|\text{root}\rangle|^2$ are large for some $n > 1$, making $y(E)$ increase faster than can be predicted from the size of a^{-1}.

By the minimal property of $H^{-1}|\text{root}\rangle$, we know $a \leq |A|^2$ for any vector A such that $HA = |\text{root}\rangle$. Therefore, to give an upper bound on a, it suffices to find one such vector A and compute its norm. Finding such a vector is equivalent to assigning weights to each node of the tree, such that the sum of weights at neighbors of the root is 1, and the sum at neighbors of any other node is 0. This can be done according to the following rules:

- All nodes with value 1 have weight zero.

- For each node b with value 1, whose parent is has weight z.

  If b has two child nodes both with value 0, then each child node has weight $-z/2$.

  If there is a fault at b, then every node in the subtree of the child node with value 1 has weight 0, and the child node with value 0 has weight $-z$.

- The value of z for the root is -1, and the assignment of weights proceeds inductively starting from the root.

The nodes attached to the leaves with value 1 of the original NAND tree can be considered a child node of the corresponding leaf, so the new tree is still labeled by the extended NAND rule, with every leaf having value 0. See Figure 2-1 for an example.

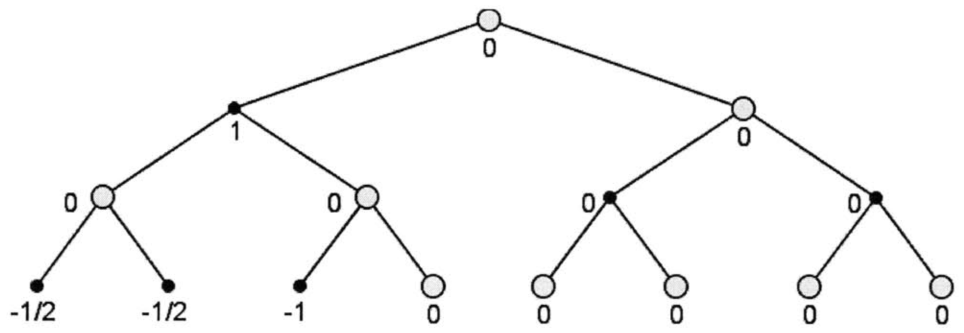

Figure 2-1: Assignment of weights for a sample tree. Empty circles denote nodes with value 1, and solid circles denote nodes with value 0.

It is clear that only nodes at even levels have nonzero weights (when the tree is considered as a bipartite graph, this is exactly the part not containing the root). If there are no faults, the weights halve every two levels down the tree. This means the contribution to $|A|^2$ of each node is multiplied by $1/4$. There are also four times more nodes every two levels down the tree. So these changes cancel out and the contribution of each level is the same. The sum $|A|^2$ is then linear to the height of the tree.

At every fault in the tree, the weight at the child node with value 0 is twice the weight that node would have if there is no fault. A simple upper bound for $|A|^2$ for trees satisfying the k-faults rule is then found as follows: for each node in the tree, there are at most k faults in the path from the root to that node. At each fault the ratio of the actual weights on the path to the weights that would be assigned if there are no faults at all will either double or become zero. This means the contribution to $|A|^2$ from each node cannot be greater than $(2^k)^2 = 4^k$ times what it is before. Combined with the result in the previous paragraph, this shows $|A|^2$ is of order at most $n \cdot 4^k$ (this can be reduced to $n \cdot 2^k$ with more careful arguments).

As we stressed before, the linear term is not the whole story. One way to see this is that we have not used all the conditions in the k-faults rule. Indeed our argument seems to work regardless of what happens at nodes in the side of a fault rooted at the node with value 1, since all these nodes have weight zero. However, when we go to the next power $H^{-2}|\text{root}\rangle$, some of these nodes will be numbered, and locally according to the rules given above. If the subtree descending from the Y-child node contains many faults, the norm of $H^{-2}|\text{root}\rangle$ will be large, therefore $y(E)$ will still grow quickly.

On the other hand, this view also indicates that there are trees that do not satisfy k-faults rule for any small k, but still has slow-growing $y(E)$. One example includes trees that have faults at every node along some path (say the right-most path), and no fault anywhere else. The most difficult case among these trees is when the values of nodes alternate between 0 and 1 along the path, so the weights on the path never becomes identically zero (see Figure 2-2). Considering the assignment of weights on this tree, it is clear that $|A|^2$ is quadratic in n, so the running time can still be polynomial in n. This appears to be the case from numerical computations of $y(E)$.

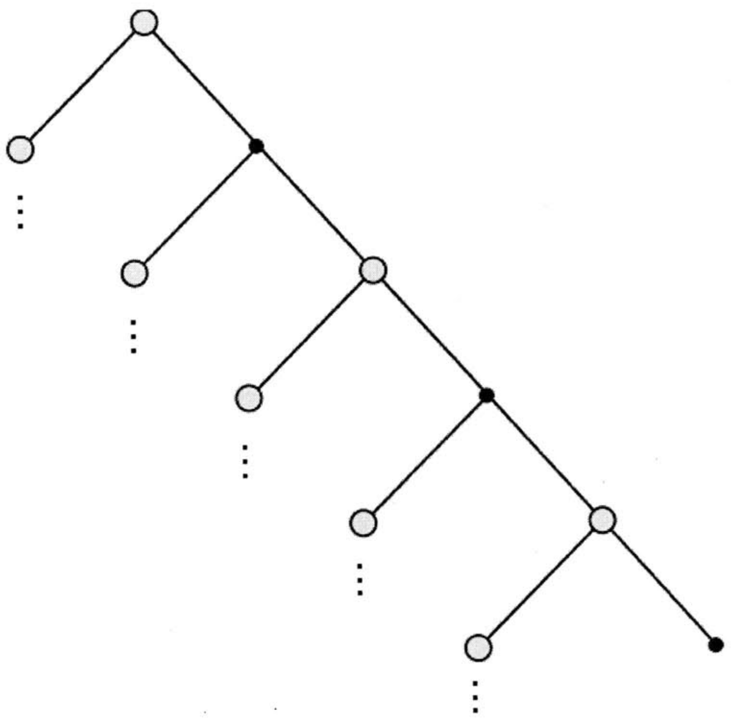

Figure 2-2: Example of a tree that does not satisfy k-faults rule for small k, but still has $|A|^2$ polynomial in n. Empty circles denote nodes with value 1, and solid circles denote nodes with value 0. Ellipsis denotes attaching a subtree without any faults to make the leaves of the resulting tree at the same height.

Chapter 3

Classical Lower Bound

3.1 Statement of the Theorem

In this chapter we will give lower bounds on the cost of classical algorithms for evaluating one class of boolean formulas with restricted inputs. Recall that we are considering boolean formulas that can be represented as a balanced c-nary tree, with each node acting as a gate, evaluating a function $f(x_1, \ldots, x_c)$ of its child nodes. The set of all possible inputs to a gate is partitioned into two types: trivial and non-trivial. We call a node in the tree a *fault* if it is not a leaf and the input given by its child nodes is nontrivial. We restrict the input to the overall formula so that along each path from the root to the leaves, there are at most k-faults for some given k (this will be called the *k-faults* condition). Let n be the height of the tree, as the number of edges on any path from the root to the leaves. We say the root is at level 1 of the tree, so the leaves are at level $n+1$.

The main goal of this chapter is the following theorem.

Theorem 3.1.1. *Given a function $f : [0,1]^c \to [0,1]$ and a partition of the inputs of f into two types, trivial and nontrivial. Suppose the following two conditions hold:*

1. There exists $X = (x_1, \ldots, x_c) \in [0,1]^c$ such that both X and $\bar{X} = (\bar{x_1}, \ldots, \bar{x_c})$ are trivial, and $f(X) \neq f(\bar{X})$.

2. There exists k_0 and n_0 such that for each $r \in [0,1]$, there is a distribution $\mathcal{G}_r$ of *trees with height n_0, root evaluating to r, and satisfying the k_0-faults rule, such that the probability of any leaf evaluating to zero is exactly $1/2$ in this distribution.*

Then for large n and k polynomial in $\log n$, no probabilistic algorithm can obtain the value of the root of every tree with height kn and satisfying the (kk_0)-faults rule with confidence at least $2/3$ before evaluating β^k leaves, where $\beta = \lfloor(\log \tilde{n})/5\rfloor$, and $\tilde{n} = n - n_0$.

Both the NAND gate and the 3-MAJ gate satisfy the conditions of the theorem. For the first condition, we can take $x_i = 0$ in both cases. For the second condition, one distribution for the NAND gate is shown in Figure 3-1, plus a suitable permutation of inputs. One distribution for the 3-MAJ gate is shown in Figure 3-2, again plus a

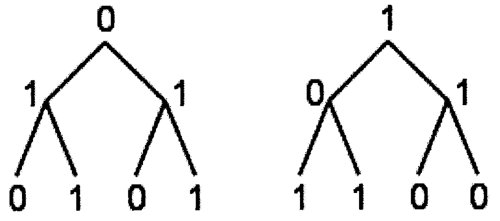

Figure 3-1: A sample distribution for the NAND gate

suitable permutation of inputs. Here the case for $r = 1$ is symmetric.

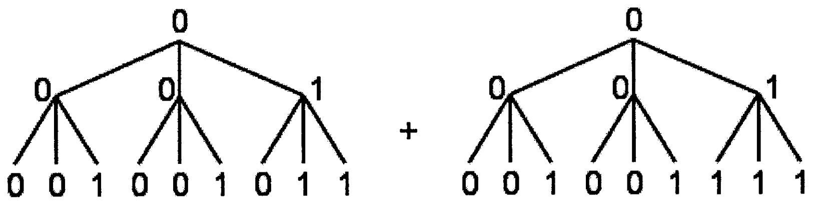

Figure 3-2: A sample distribution for the 3-MAJ gate

The first condition allows the following construction: given a root value $r \in [0, 1]$ and height n, there is a unique tree of height n, such that for any non-leaf node a in the tree, the values on the child nodes of a are either X or $\bar{X}$. Moreover the tree with root 0 and the tree with root 1 differ at every node of the tree.

3.2 Preparation

We choose X, n_0, k_0 and $\mathcal{G}_r$ according to the conditions of the theorem. Fix n sufficiently large (in particular $n \gg n_0$). To prove the theorem, we construct for each $k \geq 1$ a distribution $\mathcal{T}_k$ of trees with height kn and satisfying the (kk_0)-faults rule. We show that no deterministic algorithm can obtain the root value of a randomly chosen tree from $\mathcal{T}_k$ with probability at least $2/3$, before evaluating β^k leaves. It is clear that this implies Theorem 3.1.1.

The distribution $\mathcal{T}_k$ is constructed by induction on k. For $k = 1$ the construction is as follows: first the root value $r \in [0, 1]$ and an integer $i \in \{1, 2, \ldots, \tilde{n}\}$ are chosen uniformly. Then the first i levels of the tree are specified by requiring that for each node a in the first $i - 1$ levels, the child nodes of a have values either X or $\bar{X}$ as given in condition 1. By the discussion above this choice is unique. For each node b at level i, we choose the next n_0 levels of the tree rooted at b from the distribution $\mathcal{G}_{v(b)}$, where $v(b)$ denotes the value of the tree at node b. Doing this independently for each node at level i, we obtain the next n_0 levels of the tree. Finally, we complete the tree by requiring that each non-leaf node at or below level $i + n_0$ has child nodes with value either X or $\bar{X}$. This completes the construction for $k = 1$.

For any $k > 1$, we construct the first $n + 1$ levels of the tree according to the distribution $\mathcal{T}_1$. Then for each node b at level $n + 1$, we choose the subtree rooted at b independently from the distribution $\mathcal{T}_{k-1}$. This gives a distribution $\mathcal{T}_k$. It is clear that any tree in $\mathcal{T}_k$ will have height kn and satisfy the (kk_0)-faults rule. For any k, we say a tree in $\mathcal{T}_k$ has category i if $i \in \{1, 2, \ldots, \tilde{n}\}$ is chosen during the construction of the first $n + 1$ levels of the tree.

Now we will follow a deterministic algorithm trying to determine the value at the root of a random tree from these distributions. Note that at any point during the computation, we know exactly which trees in the distribution are not excluded by the known values of the leaves. Let p_0, p_1 be the probability (in the distribution) that a random tree remaining has root value 0 and 1, respectively. Then the algorithm should guess 1 if $p_1 > 1/2$ and 0 if $p_1 < 1/2$. Moreover the probability that the algorithm will be correct is

$$\max(p_0, p_1) = \frac{1}{2} + \frac{|p_0 - p_1|}{2}.$$

Naturally we will try to bound $|p_0 - p_1|$, which we call the *confidence level.*

Let $p_r(i), r \in [0,1], 1 \leq i \leq \tilde{n}$ be the relative probability that a random tree remaining has root value r and category i. Write $S = \sum_i p_0(i) + \sum_i p_1(i)$. Note that we have not specified $S = 1$ or any other normalization. With any normalization, we have

$$|p_0 - p_1| = \frac{|\sum_i p_0(i) - \sum_i p_1(i)|}{S} \leq \frac{\sum_{i=1}^{\tilde{n}} |p_0(i) - p_1(i)|}{S}.$$

Write $D = \sum_{i=1}^{\tilde{n}} |p_0(i) - p_1(i)|$ (for difference). Our general approach is to give, up to some small probability, an upper bound on D and a lower bound on S.

For the lower bound on S, we need the following lemma:

Lemma 3.2.1. *Fix $A_0 > 0$. For each integer $t > 0$ we choose a number $0 \leq p_t \leq 1$ (knowing the value of A_{t-1}). Then $A_t = p_t A_{t-1}$ with probability p_t, and $A_t = (1 - p_t)A_{t-1}$ otherwise. For any $m \in \mathbb{Z}^+$ and $0 < C < 1$, the probability that $A_m < CA_0$ is less than $2^m C$, regardless of the choices of p_t.*

Proof. First let $A_0 = 1$. We can model the process as trying to bound a real number x uniformly distributed between 0 and 1, where at each step t we choose a number y_t and is told whether $x < y_t$. Here A_t corresponds to the size of the interval that is not excluded after t steps, and p_t corresponds to the relative position of y_t within the non-excluded interval ($p_t = 0$ if y_t is to the left of the non-excluded interval, and $p_t = 1$ if y_t is to the right).

After m steps, each of the 2^m possible strings of answers we receive corresponds to some subinterval of $[0,1]$. These subintervals cover $[0,1]$ without overlaps (if some string of answers is impossible, let it correspond to the empty interval). The condition $A_m < C$ corresponds to x falling in an interval of size less than C. Since there are just 2^m intervals, the combined length of intervals with size less than C is less than $2^m C$. So the probability that x will fall into one of these intervals is less than $2^m C$.

It is clear that for general $A_0 > 0$, the same conclusion holds with $A_m < C$ replaced by $A_m < CA_0$. $\square$

In this chapter we will use $A_0 = 2\tilde{n}$, $m = \beta = \lfloor(\log \tilde{n}/5)\rfloor$, and $C = \tilde{n}^{-2/5}$. So $A_m < 2\tilde{n}^{3/5}$ with probability less than $p = 2^m C \leq \tilde{n}^{-1/5}$. This particular choice is used to ensure that $\tilde{n}/A_m^2 = o(1)$ with high probability.

3.3 Case $k = 1$

Now we begin the proof for the $k = 1$ case. We fix the normalization of $p_r(i)$ so it equals the probability that a random tree with root value r and category i from the distribution $\mathcal{T}_1$ is not excxluded by the known values of leaves. So at the beginning of the computation, we have $p_0(i) = p_1(i) = 1$ for all $1 \leq i \leq \tilde{n}$, and $S = 2\tilde{n}$. Then Lemma 3.2.1 can be applied as follows: suppose the algorithm chooses to evaluate leaf b at step t, then p_t corresponds to the probability that a random tree remaining after step $t-1$ has value 0 at b, and A_t corresponds to the value of S after t steps. By the lemma we know that after $m = \beta$ steps, the probability that $S < 2\tilde{n}^{3/5}$ is less than $\tilde{n}^{-1/5}$.

For the upper bound on D, we need to examine how the values $p_0(i)$ and $p_1(i)$ are updated after evaluating each leaf. The main property of the construction is the following: for any node b at level i, where i is the category of the tree, if a is the first leaf descending from b that is evaluated, then the probability that a evaluates to 0 is exactly $1/2$, even conditioning on the value at b. For any $r \in [0, 1]$ and i, it follows that $p_r(i) = 1/2$ after the first step. Now suppose leaf a is evaluated at step $t > 1$. Let h be the level of the first node where the path from root to a breaks off the tree formed by the already evaluated leaves. More precisely, let E be the set of leaves already evaluated, and represent both a and each $b_j \in E$ by coordinates in $[c]^n$ coding the path taken to reach the leaf. Then h is the number of initial coordinates that agree between a and b_j, maximized over $b_j \in E$. Let $E' \in E$ be the subset of already evaluated leaves that agree with a to height h. We will examine how each $p_r(i)$ is updated upon learning the value of a. There are three cases:

Case 1, $i+n_0 \leq h$: Here we are conditioning on the faults occuring before the last split. In particular the leaves a and $b_j \in E'$ are contained in a subtree formed entirely of inputs X and $\bar{X}$. Recall that in such a subtree the value of one leaf determines the value of all others. If the values of E' are inconsistent with each other, then $p_r(i) = 0$ for all $i + n_0 \leq h$ to begin with. Otherwise, the value of a can be predicted from the values of $b_j \in E'$. If the actual value of a is the same, then $p_r(i)$ remains the same. Otherwise $p_r(i) = 0$ after evaluating a.

Case 2, $i > h$: Here a is the first leaf we evaluated that descends from some node b at level i. The probability that a has value 1 is exactly 1/2, even if we know something about the value of b. So $p_r(i)$ is reduced by a half for all $i > h$.

Case 3, $i \leq h < i+n_0$: We cannot say anything about these cases without knowing the details of the function f. However, the increase of $|p_0(i) - p_1(i)|$ due to these cases is at most n_0.

Neither case 1 nor case 3 involves any increase in D. So the increase is by at most n_0 for each step. Since $D = 0$ at the beginning, we know $D \leq \beta n_0$ after β steps. Combining with the upper bound on S, we have

$$|p_0 - p_1| \leq \frac{D}{S} \leq \frac{n_0 \beta}{2\tilde{n}^{3/5}}$$

with probability at least $1 - \tilde{n}^{-1/5}$.

3.4 General k

We will prove by induction statements of the following form: for a given $1 \leq l \leq k$, if less than β^l leaves are evaluated on a random tree from the distribution $\mathcal{T}_k$, then with probability $1 - p_{k,l}$, the confidence level $|p_0 - p_1|$ is at most $c_{k,l}$. For $k = 1$, we proved this for

$$l = 1, p_{1,1} = \tilde{n}^{-1/5}, \text{ and } c_{1,1} = \frac{n_0 \beta}{2\tilde{n}^{3/5}}.$$

In general, we will show the following:

$$p_{k,k} \lesssim \tilde{n}^{-1/5},\ p_{k,l} \lesssim 4\beta^2\tilde{n}^{-(k-l+1)/5} \text{ for } l < k, \tag{3.1}$$

$$\text{and } c_{k,l} \lesssim (8n_0)^{k-l+1}\tilde{n}^{-3(k-l+1)/5}\beta^{3+k-l}. \tag{3.2}$$

where $a \lesssim b$ means $a \leq b(1 + C\beta^{\alpha}n^{\gamma})$, for fixed constants $\alpha > 0$ and $\gamma < 0$, and C grows polynomially in k and l. The $l = k$ part of the above statements easily imply the theorem.

The proof naturally has three parts: bounds on S, D and $p_{k,l}$ for each k, l. But first, we shall lay down a framework describing when is the algorithm 'lucky' during the computation.

3.4.1 Exceptions

The concept of exception is defined inductively as follows. For $k = l = 1$, an exception occured if the algorithm is lucky in the division process specified in Lemma 3.2.1 (so the probability is bounded by $\tilde{n}^{-1/5}$, from now on we will just say the division process is lucky). For $k > 1$, let L be the set of nodes at height $n + 1$. We will imagine the algorithm as trying to evaluate leaves descending from different $b \in L$, then piece together information about nodes in L to guess the value at the root. It doesn't matter whether the algorithm is following this type of strategy, since any sequence of evaluations is allowed in this view, and in principle we can calculate exactly which root value the algorithm should return, along with the confidence level. For $l < k$, an exception occured if exceptions occured for at least two of the subtrees (with roots in L, same below). If this does not happen, then we will show that the algorithm has high confidence in the value of at most one $b \in L$. Since the first sure value of a node in L simply sets each $p_r(i)$ to $1/2$, the division process does not properly begin. If $l = k > 1$, an exception occured if exceptions occured for at least two of the subtrees, or if the division process is lucky. In this case, we may evaluate β^{k-1} or more nodes on a subtree (which is in the distribution $\mathcal{T}_{k-1}$). For these subtrees we have no bound on the confidence level. The number of such subtrees is at most $\beta - 1$. If at most one exception occured on subtrees, then the algorithm has high confidence on at most $\beta - 1 + 1 = \beta$ nodes in L, which is just the maximum number of steps in the division process.

In the next two subsections, we will give bounds on S and D given that exceptions did not occur. By induction, we may use the value of $c_{k-1,i}$ for any $1 \leq i \leq k-1$ when considering the case (k, l). Also, to simplify the argument, we give the algorithm the correct value of a node $b \in L$ after β^{k-1} leaves descending from that node are evaluated, or after an exception occured at b. Equivalently we give the algorithm the values of all leaves descending from b. Clearly the algorithm cannot do worse with this help. In the last subsection, we will bound the probability of an exception, which becomes $p_{k,l}$.

3.4.2 Bounds on S

First, we will describe how $p_r(i)$ is calculated in the case $k > 1$. For each $b \in L$ and $r \in [0, 1]$, let $p(b, r)$ be the probability that a randomly chosen tree with root value r in $\mathcal{T}_{k-1}$ is not excluded by the known values of leaves descending from b. For example, at the start of the computation $p(b, 0) = p(b, 1) = 1$. If enough leaves are evaluated so that we are sure from the leaves descending from b alone that b has value 0, then $p(b, 1) = 0$.

For each tree $t \in \mathcal{T}_1$ and $b \in L$, let $v(t, b)$ be the value of t at b. Then

$$p(t) = \prod_{b \in L} p(b, v(t, b)).$$

gives the probability that a randomly chosen tree from $\mathcal{T}_k$ with first $n+1$ levels equal to t is not excluded. So we can let $p_r(i)$ be the weighted average of $p(t)$ for all $t \in \mathcal{T}_1$ such that t has root value r and category i.

The values of $p(b, r)$ and $p(t)$ calculated this way may be extremely small. To make them easier to handle, we will change the normalization of $p(b, r)$ as follows: if we give the algorithm the correct value of b, as specified at the end of last subsection, so $p(b, r) = 0$ for some $r \in [0, 1]$. Then we reset $p(b, \bar{r}) = 1$. Otherwise, by induction we know the confidence level for the value of b is low, so the relative difference between $p(b, 0)$ and $p(b, 1)$ is very small. We multiply both $p(b, 0)$ and $p(b, 1)$ by the same constant so that $p(b, 0) + p(b, 1) = 2$, so both $p(b, 0)$ and $p(b, 1)$ are close to 1. Then $p(t)$ and $p_r(i)$ are calculated from $p(b, r)$ as above. This is simply a change of normalization, so $p_r(i)$ are still relative probabilities.

Now we will make precise the claim that $p(b, 0)$ and $p(b, 1)$ are close to 1 before either an exception or β^{k-1} evaluations on b. If no exception occured at b and the number of evaluations on b is less than β^j, then $|p_0 - p_1| \leq c_{k-1,j}$ for the value at b. This is

$$|p_0 - p_1| = \frac{|p(b,0) - p(b,1)|}{p(b,0) + p(b,1)} = \frac{|p(b,0) - p(b,1)|}{2} \leq c_{k-1,j},$$

so

$$|p(b, r) - 1| \leq c_{k-1,j} \text{ for } r \in [0, 1].$$

In particular, suppose $p(b, r)'$ is the value of $p(b, r)$ after evaluating a leaf descending from b, without obtaining the correct value of b. Then $p(b, r)' = p(b, r)A$ where

$$A \in \left[\frac{1 - c_{k-1,j}}{1 + c_{k-1,j}}, \frac{1 + c_{k-1,j}}{1 - c_{k-1,j}}\right] \implies |\log A| \lesssim 2c_{k-1,j}, \tag{3.3}$$

assuming by induction that $c_{k-1,j}$ is small.

Now we begin to estimate S, assuming that exception did not happen overall. First the case $l = k$. We have at most β sure values for nodes $b \in L$, including $\beta - 1$ from evaluating more than β^{k-1} leaves on nodes in L, and 1 from a possible exception at some $b \in L$. When a sure value is given, we update the value of $p(b, r)$ (and therefore $p_r(i)$) by first setting one of $p(b, r)$ to zero and then change the other to one. Therefore for at most β times during the computation, some $p(b, r)$ is set to zero. At all other times $p(b, r)$ simply fluctuates around one.

We claim that the decrease in S due to setting some of $p(b, r)$ to zero is modeled by the division process in Lemma 3.2.1. First consider the case where we give the value of $b \in L$ to the algorithm after evaluating β^{k-1} nodes descending from b. In this case evaluating the β^{k-1}'th node descending from b can be considered as a query for the value at b. Let p_i (in the lemma) be the probability that a randomly chosen tree from those remaining has 0 at b (this is not necessarily $p(b,0)/2$). Then the probability that b indeed has value 0 is exactly p_i. Also, if b turns out to have value 0, then setting $p(b,1) = 0$ without changing $p(b,0)$ will change S to $p_i S$. This gives a correspondence to the situation in the lemma. Now consider the case where we give the value of $b \in L$ due to an exception at b. Note whether an exception occured depends only on the leaves evaluated and their values. So whenever the algorithm evaluates a leaf a descending from b, it knows which values of a will produce an exception at b. If both values of a will, then the situation is the same as the previous case. If one of the values of a, say 0, will produce an exception, then let p_i be the probability that a randomly chosen tree from those remaining that has 0 at a will have 0 at b (same probability as above, except conditioning on $v(a) = 0$). In the event of an exception at b, then b has value 0 with probability p_i, and $S \to p_i S$ in this case. If no exception occured, then we do not consider this step as part of the division process.

Given this correspondence, also assuming that the division process is not lucky, we conclude that the decrease in S due to sure values of nodes in L is by a factor of at most $\tilde{n}^{-2/5}$. That is, the decrease in $\log S$, written as $\Delta \log S$, is at most $(2 \log \tilde{n})/5$.

Now we estimate the decrease in $\log S$ due to other nodes. For each $b \in L$, let $M(b)$ be the number of leaves descending from b that have been evaluated. For each $j, 1 \leq j \leq l$, let $A_j = \{b \in L | \beta^{j-1} \leq M(b) < \beta^j\}$. Note the sets A_j depends on where we are during the computation. If there is an exception at $b \in L$, we will exclude b from any A_j. We have the obvious bound that $|A_j| \leq \beta^{l-j+1} - 1$. Nodes in A_k will produce sure values which we considered above. For A_j with $j < k$, we will repeatly make use of Equation 3.3.

There are at most β times when some $p(b,r)$ is set to zero. Consider the (at most) $\beta + 1$ intervals in between. Since we will no longer use information about what the algorithm knows during the computation (only Lemma 3.2.1 makes use of this information), we can rearrange the order of evaluations within each interval, so that evaluations descending from the same $b \in L$ are placed together. For each interval, consider the sets A_j just before the end of the interval. For each $b \in A_j, j < k$, the total change in $\log p(b,r)$ during the interval is bounded by $2c_{k-1,j}$ (here we also count the change of $p(b,r)$ to 1 immediately after b gives an exception). This is also the maximum possible change in $p_r(i)$ and therefore S. Summing these together, we have

$$\Delta \log S \lesssim \frac{2\log \tilde{n}}{5} + 2(\beta+1)(c_{k-1,k-1}\beta^2 + c_{k-1,k-2}\beta^3 + \dots).$$

Using values of $c_{k-1,j}$, we have $\Delta \log S \lesssim (2\log \tilde{n})/5$. So $S \gtrsim 2\tilde{n}^{3/5}$.

Now consider the case $l < k$. We will not use Lemma 3.2.1 at all so we may rearrange the evaluations to put those for the same $b \in L$ together, also putting evaluations for the only exception at the front (if there is any). The possible exception will simply reduce each $p_r(i)$, and therefore S, by a half. Adding to this reduction of S due to other nodes, we have

$$\Delta \log S \lesssim \log 2 + 2(c_{k-1,l}\beta + c_{k-1,l-1}\beta^2 + \dots) \lesssim \log 2.$$

So

$$S \gtrsim \frac{2\tilde{n}}{2} = \tilde{n}.$$

3.4.3 Bounds on D

Again, we rearrange the evaluations to put those for the same $b \in L$ together, and putting evaluations for those $b \in L$ for which we have sure values at the front. So in the first phase, the evolution of $p_r(i)$ is just the same as the $k = 1$ case.

At the end of this phase, each $p(b,r)$ is either 1 or 0. For each $b \in L$, let $p_r(i,b)$ be the probability that a remaining tree with root r and category i will have 0 at b (clearly we only need to consider those i, r for which $p_r(i) \neq 0$). By the same reasoning as in the case $k = 1$, there is an integer h such that $p_r(i,b) = 1/2$ for $i > h$, and either 0 or 1 for $i + n_0 \leq h$. If $p_r(i,b) = 0$ or 1 it will not change again. Suppose leaves descending from b are evaluated during the second phase, and consider all nodes $b' \in L$ visited before b. For each such $b' \in A_j$ (where A_j are constructed at the end of the computation), the change in $\log p(t)$ for each tree $t \in \mathcal{T}_1$ due to b' is at most $2c_{k-1,j}$ as before. This shows that for root value r and category i, the change in $\log p_r(i,b)$ is at most $4c_{k-1,j}$. Summing over $b' \in L$ visited before b, we have

$$\Delta \log p_r(i,b) \lesssim 4(c_{k-1,l}\beta + c_{k-1,l-1}\beta^2 + \dots),$$

where the first term is omitted for the case $l = k$ (and same below). This means for $i > h$, we have

$$|p_r(i,b) - \frac{1}{2}| \lesssim 4(c_{k-1,l}\beta + c_{k-1,l-1}\beta^2 + \dots), \tag{3.4}$$

just before leaves descending from b are evaluated.

Now we begin to estimate D. Suppose $b \in L$ is visited in the second phase, with $b \in A_j$. Since b is not visited in the first phase, we have $p(b,0) = p(b,1) = 1$ beforehand. Now let $p(b,0)$ and $p(b,1)$ denote their values after visiting b. We have $|p(b,r) - 1| \lesssim c_{k-1,j}$. Let $p_r(i)_0$ be the value of $p_r(i)$ before visiting b, and let $p_r(i)_1$ be the value after. Then we have

$$p_r(i)_1 = p_r(i)_0[p_r(i,b)p(b,0) + (1 - p_r(i,b))p(b,1)].$$

In particular

$$p_r(i) \lesssim 1 + 2(c_{k-1,l}\beta + c_{k-1,l-1}\beta^2) + \dots \lesssim 1,$$

throughout the computation.

There are three cases for i:

Case 1, $i + n_0 \leq h$: we know $p_0(i,b)$ and $p_1(i,b)$ are either both 0 or both 1. So $|p_0(i) - p_1(i)|$ are multiplied by the same factor less than $2c_{k-1,j}$. This has the effect of at most multiplying D by $1 + 2c_{k-1,j}$.

Case 2, $i > h$: in addition to the same effect above, there is another addition to D caused by the difference between $p_0(i,b)$ and $p_1(i,b)$. This difference is bounded by Equation 3.4. The addition to D from this source is at most

$$(\max_r p_r(i)_0)|(p_0(i,b) - p_1(i,b))(p(b,0) - p(b,1))| \lesssim 8(c_{k-1,l}\beta + \dots) \cdot 2c_{k-1,j}.$$

Case 3, $i \leq h < i + n_0$: we have no assumption on $p_r(i, b)$. But $|p(b, 0) - p(b, 1)|$ is still bounded. So the addition to D from this part is at most

$$(\max_{r,i} p_r(i)_0) n_0 |p(b, 0) - p(b, 1)| \lesssim n_0 \cdot 2c_{k-1,j}.$$

Summing these together for all b, putting multiplications from case 1 and 2 after the additions from case 2 and 3, we have

$$D \lesssim [D_0 + 16\tilde{n}(c_{k-1,l}\beta + \dots)^2 + 2n_0(c_{k-1,l}\beta + \dots)](1 + 2c_{k-1,l}\beta + \dots),$$

where D_0 is the value of D after the first phase, which by the argument in the case $k = 1$, is at most $2n_0\beta$ if $l = k$ and 0 if $l < k$. The point is that any $c_{k,l}$ contains a factor of $n^{-\alpha}$ with $\alpha \geq 3/5$, so any term other than D_0 in the expression above is at most of order $n^{-1/5}$. If $l = k$, then D_0 dominates the sum, so for this case $D \lesssim 2n_0\beta$. If $l < k$, from the values of $c_{k-1,j}$ we see that the terms $16\tilde{n}(c_{k-1,l}\beta)^2$ and $2n_0 c_{k-1,l}\beta$ dominate.

Combining the bound on D and S, we have for $l = k$,

$$c_{k,k} \leq \frac{D}{S} \lesssim \frac{2n_0\beta}{\tilde{n}^{3/5}},$$

and for $l < k$, it is a straightforward calculation to verify that

$$c_{k,l} \lesssim (8n_0)^{k-l+1}\tilde{n}^{-3(k-l+1)/5}\beta^{3+k-l},$$

satisfies the recursion relation.

3.4.4 Bounds on $p_{k,l}$

Now we begin the proof of the bound on $p_{k,l}$. By induction we can use the values $p_{k-1,j}$, which give the probability of getting exceptions at subtrees (rooted at nodes in L, same below). We say an overall exception is of type I if it is caused by at least two exceptions on subtrees, and type II if it is a result of luckiness in the division process.

We make two observations to prepare for the proof. First, note that the condition for getting an exception at the root is the same among all l less than k. Moreover this condition is the same as the condition for getting a type I exception for the case $l = k$. For $l = k$, if less than β^{k-1} leaves are evaluated, then the division process has not began, so the probability of getting a type II exception conditioning on the values known so far is still bounded by $\tilde{n}^{-1/5}$.

Second, note that for large n, the number of nodes in L is far greater than the number of evaluations we have. This means we can combine evaluations of different subtrees into the same subtree. Let's make this more precise since it will be used repeatedly below. Suppose an algorithm evaluated leaves from subtrees rooted at $b_1, \ldots, b_m$. The algorithm may interleave evaluations on these subtrees. We may simulate this process on a single subtree rooted at b as follows: for each $1 \leq j \leq m$, we set aside a sufficiently large set B_i of subtrees of b (that is, rooted at level $2n + 2$ from the original root). Each time the algorithm evaluates a new subtree b'_{ij} of b_i, we choose a new subtree $\bar{b}'_{ij}$ in B_i and keep track of the correspondence so we can simulate any further evaluations on b'_{ij} as evaluations on $\bar{b}'_{ij}$. The key difference produced by this simulation is that the total number of evaluations on b is usually larger than the number of evaluations on each b_i. The number of steps in the division process in $\bar{b}$ may also be larger. However, the number of evaluations on a subtree of b_i is the same as the number of evaluations on the corresponding subtree of b.

We begin with the case $l \leq k-2$. Here only type I exceptions are relevant. Assume that there exists an algorithm $\mathcal{A}$ that has probability P of producing exceptions at more than one subtree. This algorithm may evaluate on as many as β^l subtrees. We will produce a new algorithm $\mathcal{B}$ simulating the old one using only β subtrees. Each time $\mathcal{A}$ begins evaluations on a new subtree b, the algorithm $\mathcal{B}$ randomly chooses one of the β subtrees, which we call $\bar{b}$, and sets apart enough nodes at level $n + 1$ of $\bar{b}$ to simulate future evaluations of $\mathcal{A}$ on b. Now the probability of $\mathcal{A}$ producing exceptions at more than one subtree is P. If this happens, consider the first two subtrees b_1, b_2 at which an exception is produced. The probability that $\bar{b}_1$ and $\bar{b}_2$ are the same subtree is exactly β^{-1}. If they are not the same, algorithm $\mathcal{B}$ also produced at least two exceptions. This is because the condition for an exception at a subtree stays the same before β^{k-2} evaluations are taken, so the extra evaluations on $\bar{b}$ compared to b does not matter. Also β^{k-2} evaluations are not enough to begin the division process at any b or $\bar{b}$. This shows the probability of exception for algorithm $\mathcal{B}$ is at least $P(1-\beta^{-1})$. On the other hand, we may consider the algorithm $\mathcal{B}$ as choosing β subtrees and evaluating them up to β^l. So the probability of producing two exceptions is at most $p_{k-1,l}^2\beta(\beta-1)/2$ (adding together probabilities of exceptions at each pair of subtrees). So

$$P(1-\frac{1}{\beta}) \le p_{k-1,l}^2\frac{\beta(\beta-1)}{2} \implies P < (p_{k-1,l}\beta)^2.$$

Now consider the case $l = k-1$. Again we simulate on only β subtrees. The new feature is that now it is possible to produce exceptions of type II on a subtree. We need β^{k-2} evaluations on a subtree to begin the division process, so there are at most β attempts, for a probability of at most $\beta\tilde{n}^{-1/5}$ to get one exception of type II. Now suppose $\mathcal{A}$ produced an exception of type I at subtree b, then there is an exception of type I on $\bar{b}$, even though the number of evaluations on $\bar{b}$ may be as many as β^{k-1}. So the probability of one exception of type I is at most $p_{k-1,k-1}\beta$ and by the same argument as in case $l \le k-2$, the probability of two exceptions of type I is at most $(p_{k-1,k-1}\beta)^2$. Adding together all ways to produce two exceptions, we get

$$p_{k,k-1} \le (\beta\tilde{n}^{-1/5} + p_{k-1,k-1}\beta)^2 \lesssim 4\beta^2\tilde{n}^{-2/5}.$$

Finally we consider the case $l = k$, again simulating on β nodes. We may consider exceptions of type I and type II separately. Probability of exceptions of type II is just $\tilde{n}^{-1/5}$. Now focus on exceptions of type I. Suppose algorithm $\mathcal{A}$ has an exception on a subtree b. There are again two cases: first, the exception is of type II. Since β^{k-2} evaluations at b are required to initiate the division process, there are at most β^2 tries. So the probability of getting one such exception is at most $\beta^2\tilde{n}^{-1/5}$. Now suppose the exception at b is of type I. This means there are exceptions on two subtrees of b.

Then there are also exceptions on two subtrees of $\bar{b}$. The problem is that the number of evaluations on $\bar{b}$ may exceed β^{k-1}, so the available values of $p_{k-1,i}$ cannot help directly.

This suggests that we establish the following statement, used with $k \to k+1$ and $\bar{b}$ being a tree from $\mathcal{T}_k$.

Lemma 3.4.1. *Suppose an algorithm made less than β^{k+1} evaluations on a tree from $\mathcal{T}_k$, then the probability of getting one exception on subtrees is $\lesssim (\beta^3 + \beta^2)\tilde{n}^{-1/5}$. The probability of getting two exceptions on subtrees is $\lesssim (\beta^3 + \beta^2)^2\tilde{n}^{-2/5}$.*

Proof. We will prove this by induction on k. For case $k = 1$, there is no such thing as exceptions on subtrees, so the probability is zero.

Given algorithm $\mathcal{A}$, we produce a new algorithm $\mathcal{B}$ by simulating on β^2 subtrees, again randomly assigning any subtree b evaluated by $\mathcal{A}$ to one of the subtrees $\bar{b}$ in the simulation. If $\mathcal{A}$ evaluates more than β^{k-1} leaves on a subtree b, then it will no longer produce an exception at b, so we may as well stop simulating that subtree. If $\mathcal{A}$ produces a type II exception on a subtree b, then it must have evaluated more than β^{k-2} leaves on b, so there are at most β^3 tries, and the probability of getting one exception this way is at most $\beta^3\tilde{n}^{-1/5}$. If $\mathcal{A}$ produces a type I exception on b, then we may use the case $k-1$ of this lemma as long as the number of operations made on $\bar{b}$ does not exceed β^k. We will bound the probability that the number of evaluations will exceed β^k on any of the β^2 subtrees used in the simulation. In the simulation the number of evaluations corresponding to one subtree in $\mathcal{A}$ is at most β^{k-1}. Moreover the total number of evaluations is at most β^{k+1}. Lemma 3.4.2 below shows the probability that the number of evaluations on a given one of the β^2 subtrees exceeding β^k is at most of order $1/\beta!$. The value $\beta!$ is superpolynomial in $\tilde{n}$, so this probability may be ignored, even after multiplying by β^2. So up to a negligible probability, a type I exception on b implies at least two exceptions on the subtrees of $\bar{x}$, which has probability $\lesssim (\beta^3+\beta^2)^2\tilde{n}^{-2/5} < \tilde{n}^{-1/5}$ by the $k-1$ case of the lemma. The probability of collision is now β^{-2}. So the probability of getting one type I exception is at most $\tilde{n}^{-1/5}\beta^2$ and the probability of getting two type I exceptions is at most

$$\tilde{n}^{-2/5}\frac{\beta^2(\beta^2-1)}{2}(1-\frac{1}{\beta^2})^{-1} < \tilde{n}^{-2/5}\beta^4,$$

by a similar argument as before. Adding together the probabilities we get the lemma for case k. $\square$

Lemma 3.4.2. *Given a finite sequence of real numbers $\{a_i\}$. Suppose each $a_i \leq \beta^{k-1}$ and their sum is at most β^{k+1}. Consider a subset S of $\{a_i\}$ constructed by placing each a_i into S with probability β^{-2}. Then the probability that $\sum_{a_i \in S} a_i \leq \beta^k$ is of order $1/n!$.*

Proof. Normalize by dividing everything by β^{k-1}. The sum of elements in S is a random variable X that is the sum of random variables X_i, with each mean value $\bar{X}_i \leq \beta^2$, and $\sum \bar{X}_i \leq 1$. Each X_i takes 0 with probability $1-\beta^{-2}$ and a nonzero value with probability β^{-2}. Then the distribution of X is very closely poisson with parameter at most 1. So the probability that $X > \beta$ is at most of order $1/\beta!$. $\square$

To finish the proof, we just need to sum together all possibilities for the case $l = k$. This gives

$$p_{k,k} \lesssim \tilde{n}^{-1/5} + (\beta^2\tilde{n}^{-1/5} + (\beta^3+\beta^2)^2\tilde{n}^{-2/5})^2 \lesssim \tilde{n}^{-1/5}.$$

One can check that the bounds of $p_{k,l}$ given in Eq. 3.1 are satisfied.

3.5 An 'Easy' Boolean Function

In this section we give a boolean function that does not satisfy the conditions of the theorem, and for which the classical algorithm can evaluate faster than implied by the theorem. Consider the AND gate with $(0,0)$ and $(1,1)$ being the trivial inputs. Condition 1 of the theorem is satisfied but condition 2 is not, since any tree with root value 1 must have value 1 at all leaves. Consider any tree with height n and satisfying the k-faults rule. If the root value is 0, then some thought shows that at least a proportion 2^{-k} of the leaves must have value 0. If the root has value 1 then all leaves have value 1. So one can obtain the value of the root by sampling $O(2^k)$

leaves, and return 0 if a leaf with value 0 is found. This gives high probability of success when the number of evaluations is a large but constant multiple of 2^k.

Chapter 4

Quantum Walk Through Decision Trees

In this chapter we will apply some previous results to quantum walks on decision trees, as formulated by Farhi and Gutmann in [3]. In the first section, we will briefly summarize the construction and main results in [3]. In the second section, we will give a class of decision trees that is quantum penetrable in polynomial time.

4.1 Summary of Previous Work

Abstractly, a decision tree is just a binary tree. However, here we are considering the binary tree as encoding some decision problem that involves finding a solution satisfying certain constraints, among a large number of possible solutions. For example, in 3-SAT problem on n variables, we are asked to find an n-bit string $(x_1, \ldots, x_n)$ satisfying all of the given clauses. The example given in [3] is the "exact cover" problem. In this problem, we are given a set S and a collection $\mathcal{A}$ of subsets of S. We want to find a subcollection $\mathcal{A}' \subset \mathcal{A}$ such that the union of sets in $\mathcal{A}'$ is S, and that the sets in $\mathcal{A}'$ are mutually disjoint. Again we can consider a possible solution as an n-bit string, where n is the size of $\mathcal{A}$. The strings encode which sets in $\mathcal{A}$ are chosen.

Each of the above two problems can be considered as a binary tree, with each node in the tree representing a partial solution. One way of doing this, which we will consider here, is to associate each node at level i as a choice of the first i bits, and the two child nodes of each node b correspond to the two ways of extending the partial solution of b (in this chapter we will let root be at level 0, so the leaves are at level n). Starting from the complete tree with height n, we remove all leaves that do not satisfy the constraint. We may also remove nodes that represent partial solutions that cannot be extended to a valid solution, for example those already violating some of the constraints. The original problem is then converted to the problem of finding a node at level n, starting from the root.

One way of finding a node at level n is to perform a random walk through the tree, starting from the root. At each time step, we choose a neighbor of the current node with uniform probability and move to that neighbor. We say a class of decision trees is classically penetrable if we can arrive at a node at level n within polynomial time. That is, there exists constants A, B such that for large height n, the probability that we arrive at a node in level n before n^A steps is greater than n^{-B}.

Now we construct a quantum system from this binary tree, by taking the Hamiltonian to be the adjacency matrix. In the original paper, the Hamiltonian is defined as $\langle a|H|b\rangle = -1$ if a, b are joined by an edge, and $\langle a|H|a\rangle = d(a)$, where $d(a)$ is the number of neighbors of a. With this Hamiltonian the equations of motion resembles that of the classical random walk. However, we will simply use the adjacency matrix for a closer correspondence with previous sections. The difference is mostly a shift in overall energy and a change of sign, so it should not have a large effect on the analysis.

One way to define quantum penetrability for a class of decision trees is as follows. Start with the initial state $|\text{root}\rangle$ localized at the root of the tree. If there are constants A, B and C such that after evolving the system for time Cn^A followed by a measurement with the nodes as the basis, then the probability that the state is on a node in level n is at least n^{-B}. In other words, there is A, B and C such that

$$\sum_a |\langle a|e^{-iHt}|\text{root}\rangle|^2 > n^{-B},$$

where $t = Cn^A$, and the sum is taken over all nodes a in level n. This problem is found to be more difficult to analyze. Therefore we will make two simplifications: first, we suppose there is only one node at level n, and second, we attach long lines of nodes to both the root and the unique node at level n. Then we can consider the tree as a runway with several branches coming out of it. See Figure 4-1 for an example.

The initial state is a wave packet on the left part of the runway travelling toward the right, with a peak around $E = 0$ in the energy spectrum. As the system evolves, the packet will travel through the trees coming out the runway, and one part of it will clear all of the trees and continue toward the right. For a given energy E, we may define the transmission coefficient $T(E)$ as usual. Then we say the tree is quantum-penetrable if there exist constants A, B and C such that $|T(E)| > n^{-B}$ for all $|E| < Cn^{-A}$. Moreover the phase of $T(E)$ should not fluctuate with frequency more than polynomial in n. This condition implies that a wave packet containing energies E with $|E| < Cn^{-A}$ will mostly transmit through the trees, so the time of evolution and length of runway required for the packet to pass through are both polynomial in n.

As in the previous chapters, we can define a function $y_m(E)$ for each tree coming out of the node m. If $|m\rangle$ is the m'th node on the runway and $|m'\rangle$ is the node just above m in Figure 4-1, then we define $y_m(E) = \langle m'|E\rangle / \langle m|E\rangle$, where $|E\rangle$ is the eigenstate of the system with energy E (for this definition we again imagine the runways to be infinite. See section 1.1).

The transmission coefficients $T(E)$ are determined from $y_m(E)$. See [3] for details. Here we will just give the result. Let

$$M_m = \begin{pmatrix} E - y_m(E) & -1 \\ 1 & 0 \end{pmatrix},$$

and

$$M = M_{n-1}M_{n-2}\dots M_0 = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

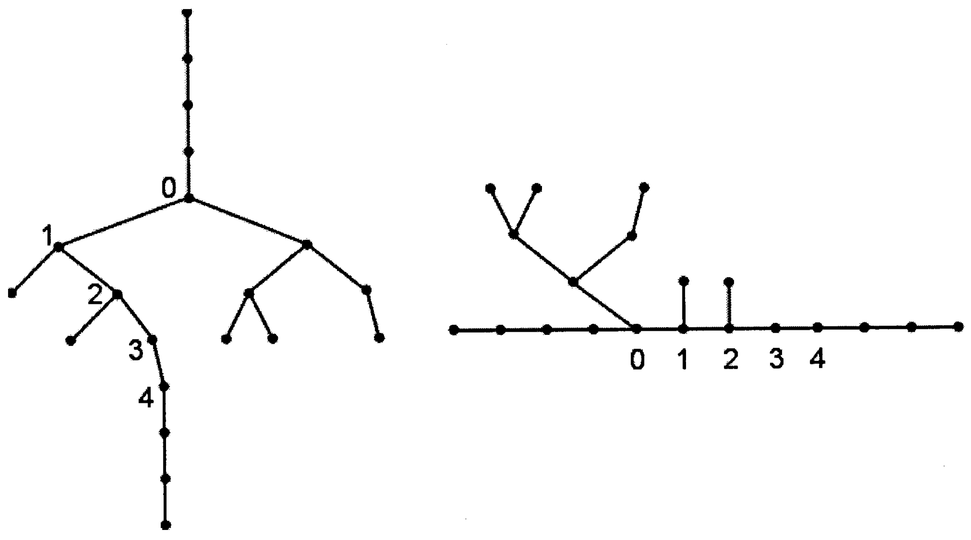

Figure 4-1: An example of a decision tree with two lines attached to the root and the unique node at level 4.

Then

$$T(E) = e^{-in\theta} \frac{2i \sin\theta}{c - b + (d-a)\cos\theta + (d+a)\sin\theta}, \tag{4.1}$$

where

$$E = 2\cos\theta.$$

Note that the expression for M_m and E are slightly different due to the use of a different Hamiltonian.

Farhi and Gutmann gave in [3] a family of trees that is quantum penetrable but not classically penetrable. However, it is possible to design a classical algorithm that, knowing only the depth of nodes in the tree (that is, which way is up and which way is down), can find the node at level n in polynomial time for this family of trees. Now we will give a wider class of trees that is quantum penetrable, but for which there may not be such classical algorithms.

4.2 Main Result

First, we establish the following intuitive result: if for some energy E near zero, each $y_m(E)$ is small (that is, each tree coming out of the runway has transmission coefficient near 1), then the overall amount of transmission $|T(E)|$ is near 1. Moreover the phase of $|T(E)|$ does not fluctuate quickly.

We will prove this by visualizing the entries in M as a weighted sum over paths from left to right in Figure 4-2, where each path is given the weight that is the product of weights on edges in the path.

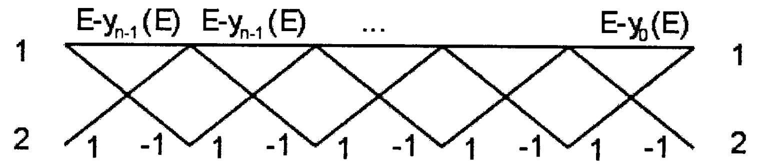

Figure 4-2: Visualization of entries in M. Each M_{ij} corresponds to a weighted sum of paths going from i on the left to j on the right.

This is a result of writing

$$M_{ij} = \sum_{s_1, s_2, \ldots, s_{n-1}=1,2} (M_{n-1})_{i,s_{n-1}} (M_{n-2})_{s_{n-1},s_{n-2}} \cdots (M_1)_{s_2,s_1} (M_0)_{s_1,j}.$$

If E is small and each $y_m(E)$ are small, then going horizontally will sharply decrease the weight of a path. So the paths with the highest weights by magnitude are those that never travelled along the horizontal line. If n is even, there is such a path for each $i = j$, whose weights are 1. If n is odd, then there is such a path for each $i \neq j$, with weights ± 1. There are at most n^a paths that travel along the horizontal line for a steps. Let $\epsilon = \max_m(E - y_m(E))$. Then the total contribution from paths that travel along the horizontal line is at most $\sum_{a \geq 1} \epsilon^a n^a = O(\epsilon n)$ for ϵn small. When $E \approx 0$, we have $\cos\theta = O(E)$ and $1 - \sin\theta = O(E^2)$. From equation 4.1, we see that $1 - |T(E)| = O(\epsilon n) + O(E^2)$. Moreover $T(E)$ is always close to one of $\pm i e^{-in\theta}$ so the phase also does not fluctuate much.

In particular, if there is E_0 and s_m such that for each m and $|E| < E_0$ there is $|y_m(E)| \leq s_m|E|$, then the transmission coefficient $T(E)$ satisfies $1 - |T(E)| = O(\max_m n s_m |E|) + O(n|E|)$. Combining with the results in section 2.1, we have the following result.

Theorem 4.2.1. *For a given decision tree, consider each tree T_m coming out of the runway as a NAND tree with each leaf given value 0. If each T_m evaluates to 1 and satisfies k-faults rule for a fixed k, then $1-|T(E)| = O(n^2 2^k |E|)$ for $|E| = O(n^{-2}2^{-k})$. Moreover the phase of $T(E)$ does not fluctuate much in this range of E. In particular, a class of decision trees is quantum penetrable if each tree satisfies this condition with $k \leq a \log n$ for a fixed constant a.*

Proof. Follows from previous discussion with $s_m = O(n \cdot 2^k)$. □

For instance, the example of even-height bushes given in [3] contains trees with no faults. The condition on the parity of the tree corresponds to the condition that these trees evaluate to 1 at the root.

Chapter 5

Conclusion

In this paper we investigated the running time of quantum walk algorithms for evaluating boolean functions on a special set of inputs. We showed that on these inputs, there is a super-polynomial separation in cost between the quantum algorithm and classical algorithms. We also applied this result to give a class of decision trees that is quantum penetrable, but may be difficult for a classical algorithm to search through.

There are still many directions in which one can extend these results. First, one can try to formulate a better condition for when a NAND tree can be evaluated quickly by the quantum walk algorithm (that is, for which trees is the growth rate of $y(E)$ polynomial in n). We showed at the end of section 2.3 that there are trees which do not satisfy k-faults rule for any small k, but nevertheness have $y(E)$ growing at rate polynomial in n. We would like to find a succinct condition that includes these trees and many other such examples.

This question can also be asked for general span programs. Since the process of assigning weights to nodes described in section 2.3 is used to motivate the above example, a related question is how to generalize this process to other span programs, and how to construct rigorous proofs based on this technique.

The same situation applies to classifying quantum penetrable decision trees. It is clear that the previous question is a necessary first step in this classification. One can also consider what can be said in the related but more realistic situation where there are many valid solutions and there are no lines attached to those solutions.

In the realm of classical computing, we have not yet proved that the class of decision trees given in section 4.2 is not efficiently searchable by a classical algorithm that knows depth in the tree. Finally, we can ask whether it is possible to construct more realistic classical problems (for example, non-oracular problems), that generate NAND trees, general span programs, or decision trees with restriction to inputs as in the paper.

Bibliography

[1] Andris Ambainis. A nearly optimal discrete query quantum algorithm for evaluating NAND formulas. arXiv:0704.3628v1 [quant-ph].

[2] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum algorithm for the Hamiltonian NAND tree. quant-ph/0702144v2.

[3] Edward Farhi and Sam Gutmann. Quantum computation and decision trees. quant-ph/9706062.

[4] Ben W. Reichardt. Span-program-based quantum algorithm for evaluating unbalanced formulas. arXiv:0907.1622v1 [quant-ph].

[5] Ben W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. arXiv:0904.2759v1 [quant-ph].

[6] Ben W. Reichardt and Robert Špalek. Span-program-based quantum algorithm for evaluating formulas. arXiv:0710.2630v3 [quant-ph].

[7] Michael Saks and Avi Widgerson. Probabilistic boolean decision trees and the complexity of evaluating game trees. *Proc. 27th FOCS*, pages 29–38, 1986.