thesis

Nikhil Shagrithaya

Acknowledgements

Firstly, I am extremely grateful to Rajat for not only agreeing to advise me, but also for introducing me to the study of Boolean functions, for patiently solving my doubts, even when I was asking them for the nth time (where $n \to \infty$), and for teaching me various ways to reason and think about problems in the field. I would also like to thank Arkadev, Manaswi, and Sourav for the numerous insightful discussions that we've had. Outside of research, I would like to thank my friends: Bharat, Nikhil (the other Nikhil), Shraddha, Saiteja, and Abhibhav for the conversations and good times I've spent with them. I would especially like to thank Akhilesh for the engaging chats we had during our tea breaks, and for his constant companionship during my time at IITK.

Lastly, I would not be here without the constant love and support of my parents, and for that I'm forever indebted to them.

Contents

Chapter 1

Introduction

The approximate degree of a Boolean function is defined as the least degree of the polynomial which successfully approximates that Boolean function to within constant error, on all inputs. Approximate degree has numerous applications in not just computational complexity, but other areas of theoretical computer science as well. Upper bounds for approximate degree have found applications in learning theory, differential privacy, and graph complexity. Approximate degree lower bounds underlie several lower bounds in communication complexity, circuit complexity, and quantum query complexity. Moreover, the concept of approximate degree is simple to state, and is an interesting object of study, in its own right.

But despite its importance, there are still many basic questions about approximate degree for which we do not have a satisfactory answer. Perhaps the most basic question one can ask is: what characteristics about a Boolean function determine its hardness of approximation? We have a partial answer for this: in the case of symmetric Boolean functions, if the function switches value on inputs having Hamming weight close to $n/2$, the approximate degree of the function is high. But we do not have a clear cut answer for non-symmetric functions.

A method to obtain lower bounds for approximate degree has emerged over the past decade, called the dual method, which entails describing the notion of approximate degree as a linear program, and exhibiting a suitable solution for the dual of the linear program. Several approximate degree lower bounds have been shown via this method. Additionally, via a technique known as the pattern matrix method, dual lower bounds can be 'lifted' to prove communication complexity lower bounds as well.

In this thesis, we show a number of results which improve upon the state of art, with respect to lower bounds obtained for approximate degree via the dual method, and also provide evidence for a tighter relation between the approximate degree and degree of a Boolean function. The organization of the rest of this thesis is as follows: Chapters 2, 3, 4, and 5 present the preliminaries and background required for understanding our results, along with relevant literature overview. Chapters 6, 7, and 8 detail our contributions, and also present further research directions and open problems.

Chapter 2

Preliminaries

2.1 Basics

We will use $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ to refer to the set of natural numbers, integers and reals, respectively. We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. S_n is used to denote the family of all permutations on $[n]$, which are the bijective functions of the form $\sigma : [n] \to [n]$.

2.2 Boolean functions

We refer to functions of the form:

$$\{0,1\}^n \to \{-1,1\}$$

as Boolean functions. It takes in a Boolean string of length n, and outputs a Boolean value. Real-valued Boolean functions are those Boolean functions whose range is $\mathbb{R}$. In most places, we will be referring to Boolean functions of the form:

$$\{-1,1\}^n \to \mathbb{R}$$

and

$$\{-1,1\}^n \to \{-1,1\}$$

This representation does not change the properties of the Boolean function, and moreover, it offers us some slight advantages.

From here onwards, we take all Boolean functions acting on the domain $\{-1,1\}^n$, unless specified otherwise. We define the *Hamming weight* of an input x to be equal to the number of -1s contained in x, and denote it by $|x|$. We denote the ith bit of x by x_i, and will refer to i as *index*, *position*, or *coordinate*.

2.2.1 Examples

The AND function, is defined to be:

$$\mathsf{AND}(x) = \begin{cases} -1, & \text{if } x = -1^n \\ 1, & \text{otherwise} \end{cases}$$

where -1^n denotes the n length input of all -1s. From the definition, it is apparent that -1 stands for TRUE (or 1), and 1 for FALSE (or 0). Similarly, the OR function takes the value 1 if the input is 1^n, and -1 otherwise.

The parity function, PAR, takes on value -1 if the Hamming weight of the input is odd, and 1 if it is even. This function has many interesting properties, as we shall see later.

2.3 Symmetric Boolean functions

Symmetric Boolean functions are those Boolean functions which are invariant under permutations to the input, i.e, for any $\sigma \in S_n$, the following is true:

$$f(x_1, \ldots, x_n) = f(x_{\sigma(1)}, \ldots, x_{\sigma(n)})$$

It is sometimes convenient to represent a symmetric Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ by an associated univariate function $F : [n] \to \{-1,1\}$, where:

$$F(|x|) := f(x)$$

Intuitively, since f is invariant under permutation of the input bits, the output depends solely on the Hamming weight of the input, meaning two inputs having the same Hamming weight will always attain the same value under f. Therefore, it is convenient to have a function which outputs the value of f when we pass it a Hamming weight value.

The AND and the PAR functions, along with the OR function (1 on 1^n, -1 everywhere else) are symmetric functions. The majority (MAJ) function, which takes value 1 when less than half of the inputs are equal to 1, and -1 everywhere else is symmetric too. The t-threshold (THR_t) function is a generalization of the AND, OR, and the MAJ functions. It is defined as:

$$\mathsf{THR}_t(x) = \begin{cases} 1, & \text{if } |x| \leq t \\ -1, & \text{if } |x| > t \end{cases}$$

It is easy to see from the definition that for any $0 \leq t \leq n$, THR_t is symmetric.

2.4 Total and Partial Boolean functions

The functions that we have talked about until now are actually known as *total Boolean functions*, which means they are defined for every element in the set $\{-1,1\}^n$. On the other hand, partial Boolean functions are only defined on some subset $\chi \subseteq \{-1,1\}^n$, and can be undefined on $\{-1,1\}^n \setminus \chi$. Since total Boolean functions are a subset of the set of all partial Boolean functions, any property which applies to the latter, also applies to the former. From here onwards, unless specified otherwise, the term *Boolean functions* will implicitly refer to *total Boolean functions*.

2.5 Polynomial Representation

For any given Boolean function $f : \{-1,1\}^n \to \mathbb{R}$, we can construct a polynomial $p : \mathbb{R}^n \to \mathbb{R}$, which when given inputs of the form $x \in \{-1,1\}^n$, outputs $f(x)$. We then call p to be a *representing polynomial* of f. We can construct such a polynomial through interpolation:

$$p(x) = \sum_{y \in \{-1,1\}^n} f(y) \prod_{i=1}^{n} \left(\frac{y_i + x_i}{2} \right)$$

The product term on the right vanishes if x and y differ, and is equal to 1 if y equals x. If we expand this representation, and add the partial parities together (a partial parity is of the form $\chi_S(x) = \prod_{i \in S} x_i$, where $S \subseteq [n]$), we notice that we get a representation of the form:

$$f(x) = p(x) = \sum_{S \subseteq [n]} \widehat{f}(S) \chi_S(x)$$

$\widehat{f}(S)$ is known as the *Fourier coefficient* of f with respect to S, and the representation is known as the *Fourier representation* of f. Since there are 2^n subsets of $[n]$, there can be at most 2^n non-zero coefficients of f.

The partial parities can be shown to be basis vectors of a 2^n dimensional vector space, and we can show that every n input Boolean function can be formed by a linear combination of these vectors, and that the coefficients of the linear combination are the Fourier coefficients themselves. It follows from this that the Fourier representation of a given Boolean function is unique. This allows us to define the degree of the Boolean function, defined by $deg(f)$, which is equal to the size of the subset corresponding to the largest non-zero Fourier coefficient. It can be easily calculated that the degree of the functions AND, OR, and PAR are all equal to n.

2.6 Boolean hypercube

The input space of an n input Boolean function, $\{-1, 1\}^n$, has a special combinatorial structure. Each input has a one-to-one correspondence with a vertex in a n-regular undirected graph of size 2^n, which can be embedded inside an n-dimensional Euclidean space. This graph is known as the Boolean hypercube, and it has many interesting combinatorial properties. The graph can be constructed inductively: for $n = 1$, we have just two inputs, 1 and -1, and the hypercube consists of two vertices denoting each of the inputs, along with an edge connecting them.

Now suppose we have the Boolean hypercube for $n - 1$. This is a graph containing 2^{n-1} vertices, and we assign all of them some unique label. Then we make a copy of this hypercube, and connect those vertices in the original and copied hypercube which have the same label. We observe that the hypercube for $n = 1$ looks like a straight line, for $n = 2$ looks like a square, and for $n = 3$ looks like a cube. It becomes difficult for us to imagine the appearance of higher dimensional hypercubes, as our minds seem to struggle while imagining objects in more three dimensions.

We can state a number of properties about the hypercube. The vertices corresponding to two different inputs are adjacent to one another if and only if they differ in value at exactly a single index. For example, 1.1^{n-1} is adjacent to -1.1^{n-1}. From this statement, it follows that every vertex is adjacent to exactly n other vertices, and hence it is an n-regular graph. The parity of the vertices adjacent to some input x is $-\mathsf{PAR}(x)$, because the adjacent vertices differ from x in exactly one bit. We can define the distance between two vertices x and y (denoted by $d(x, y)$) to be equal to the number of edges in the shortest path between x and y. Observe that $PAR(x) = PAR(y)$ if and only if $d(x, y)$ is even, and is equal to -1 otherwise.

Because this graph has a natural embedding in the n dimensional Euclidean space, every edge in the hypercube is aligned along exactly one of the n possible dimensions. We note that if we pick any edge which is aligned along the ith dimension and look at the vertices on either side of the edge, we will observe that the corresponding inputs differ by exactly one bit, and moreover, they differ at the ith index only. Therefore, we can have a bijective mapping from the n coordinates, to the n dimensions, to which we will henceforth refer to as *directions*.

2.7 Properties of Boolean functions

2.7.1 Expectation

We have discussed two representations for Boolean functions until now: first, the point representation, which was implicit in the definition of Boolean functions, where the function was defined by specifying its value at every one of the 2^n different inputs; and second, the Fourier representation, where the function was defined by specifying its Fourier coefficients. We will now see that these two representations are closely linked.

By interpreting the partial parities as the basis vectors of a 2^n dimensional vector space, all Boolean functions can be thought of as vectors in that vector space, with the Fourier coefficients determining the function. We can then come up with the notion of an inner product for the vector space:

$$\langle f, g \rangle = \sum_{S \subseteq [n]} \widehat{f}(S)\widehat{g}(S)$$

If we take f and g to be equal, we observe that the inner product is equal to the sum of squares of the Fourier coefficients of f.

Note that partial parities are Boolean functions as well (indeed, they are Boolean-valued functions), defined in the same way as before, i.e, $\chi_S(x) = \prod_{i \in S} x_i$. Naturally, the Fourier coefficient for χ_S with respect to S, $\widehat{\chi}_S(S)$, is 1, and zero for all other subsets. Therefore, when we take the inner product of a Boolean function f with χ_S, we get the Fourier coefficient of f with respect to S.

$$\langle f, \chi_S \rangle = \sum_{S' \subseteq [n]} \widehat{f}(S')\widehat{\chi}_S(S') = \widehat{f}(S)\widehat{\chi}_S(S) = \widehat{f}(S)$$

We can also define the inner product using the point representation. Plancharel's theorem states that:

$$\langle f, g \rangle = \frac{1}{2^n} \sum_x f(x)g(x)$$

The proof is as follows:

$$\begin{aligned}
\frac{1}{2^n} \sum_x f(x)g(x) &= \frac{1}{2^n} \sum_x \left[\sum_{S \subseteq [n]} \widehat{f}(S)\chi_S(x) \right] \left[\sum_{T \subseteq [n]} \widehat{g}(T)\chi_T(x) \right] \\
&= \frac{1}{2^n} \sum_x \left[\sum_S \widehat{f}(S)\widehat{g}(S) + \sum_{S,T:S \neq T} \widehat{f}(S)\widehat{g}(T)\chi_{S \Delta T}(x) \right] \\
&= \widehat{f}(S)\widehat{g}(S)
\end{aligned}$$

In the second last equality, observe that $\sum_x \chi_S(x) = 0$ whenever S is not the empty set, and is equal to 2^n when it is. The latter case is true because $\chi_\phi(x) = 1$, no matter what x is. To see the former case, consider an index $i \in S$. Then:

$$\sum_x \chi_S(x) = \sum_x \prod_{j \in S} x_j = \sum_{x_1} \cdots \sum_{x_n} \left(x_i \prod_{j \in S; j \neq i} x_j \right) = \sum_{x_i} x_i \left(\sum_{x_1} \cdots \sum_{x_{i-1}} \sum_{x_{i+1}} \cdots \sum_{x_n} \prod_{j \in S; j \neq i} x_j \right)$$

In the last equality, for every term inside the brackets, we multiply it first with 1, and then with -1, and then add them, and hence the whole sum vanishes to zero. Sometimes, we also use the following notation to denote the inner product:

$$\mathbb{E}[fg] = \mathbb{E}_{x \sim \{-1,1\}^n}[f(x)g(x)] = \frac{1}{2^n} \sum_x f(x)g(x) = \langle f, g \rangle.$$

This definition is mostly used when we are dealing with probabilities. Note that it corresponds to the expectation of $f(x)g(x)$, when x is picked uniformly.

By observing this new definition of the inner product, if f is a Boolean valued function, we get the following identity (also known as Parseval's equality):

$$\langle f, f \rangle = \frac{1}{2^n} \sum_x f^2(x) = \sum_{S \subseteq [n]} \widehat{f}(S)^2 = 1$$

2.7.2 Sensitivity and Influence

Suppose for a given Boolean function f, and an input x, we want to measure how 'sensitive' f is at input x. One natural measure would be to try and flip a bit, one at a time, and count on how many flips does the function value change. For example, take the OR function, and the all ones input. Now, if you flip the first bit from 1 to -1, the function value changes from 1 to -1. The same holds true for every bit you flip. Therefore, we get the sense that the OR function is extremely sensitive at the all ones input.

Now, take an input y which has 1 at every position, except at position i, where it has -1. What is the sensitivity of this input, with respect to the OR function? (Note that it is very important we talk about

the sensitivity of an input with respect to a certain function, because, as we shall soon see when we define the concept of sensitivity formally, the sensitivity of an input greatly varies with respect to the function). $\mathsf{OR}(y) = -1$, and if we flip the bit at position i, y turns into the all ones input, and hence the function value switches. But flipping the bit at any position other than i gives us an input with -1 at two positions, and OR applied to such an input still gives -1. So the input y is not that sensitive to OR.

To summarize the above discussion in a formal manner, we define the *sensitivity* of a function f with respect to input x as:

$$s_x(f) = |\{i : f(x) \neq f(x_{\oplus i}\}|$$

where $x_{\oplus i}$ is equal to x with the ith input bit flipped.

Therefore, according to the above definition, $s_{1^n}(\mathsf{OR}) = n$, and $s_y(\mathsf{OR}) = 1$ when y is an input with Hamming weight 1. Similarly, $s_{-1^n}(\mathsf{AND}) = n$. It can also be easily observed that $s_x(\mathsf{PAR}) = n$, for any input x.

Until now, we have been looking at the behaviour of function switches from the perspective of an input. But we can also look at from the perspective of a certain coordinate. We can fix a Boolean function f, pick a coordinate $i \in [n]$, and ask the following: for how many inputs $x \in \{-1, 1\}^n$, does the following hold: $f(x) \neq f(x_{\oplus i})$?

It turns out that it is much more useful to define this quantity as a fraction of the total number of inputs.

Definition 2.7.1. *The **influence** of coordinate i with respect to a Boolean function f is defined by the following quantity:*

$$\mathsf{Inf}_i(f) := \frac{\left|\{x : f(x) \neq f(x^{\oplus i}\}\right|}{2^n}$$

A particular example of interest here is the dictator function. More specifically, the ith dictator function is defined as:

$$\mathrm{Dict}_i(x) := x_i$$

That is, the function value solely depends on the value of x_i, and all other input bits are ignored. It is easy to see that $\mathsf{Inf}_i(\mathrm{Dict}_i) = 1$, and $\mathsf{Inf}_j(\mathrm{Dict}_i) = 0$ for any $j \neq i$.

Note that the influences for the AND and the OR function is the same for each coordinate, equal to 2^{1-n}.

We can also think of influence in terms of the Boolean hypercube. $\mathsf{Inf}_i(f)$ is equal to the fraction of edges aligned in the direction corresponding to i whose two end vertices have different values in f. It is easy to see this because the two end vertices of the edge differ only at the ith position.

The *total influence* of a function f is simply defined as the sum of influences of all the coordinates:

$$\mathsf{Inf}(f) = \sum_i \mathsf{Inf}_i(f)$$

Since the PAR function has an influence of 1 for every coordinate, $\mathsf{Inf}(\mathsf{PAR})$ is equal to n. This is the maximum total influence value a function can have.

Revisiting the concept of sensitivity, we define the *sensitivity* of a function as the maximum sensitivity across all inputs. That is:

$$s(f) := \max_x s_x(f)$$

It turns out that sensitivity and total influence of a function are closely related:

$$\mathsf{Inf}(f) = \sum_i \mathsf{Inf}_i(f) = \sum_i \frac{\left|\{x : f(x) \neq f(x^{\oplus i}\}\right|}{2^n} = \frac{1}{2^n} \sum_x \sum_i \mathbb{1}[f(x) \neq f(x^{\oplus i})] = \mathbb{E}_x[s_f(x)]$$

Chapter 3

Complexity measures of Boolean functions

In this chapter, we are going to list out the various complexity measures related to Boolean functions. In the next section, we will discuss the relations among these complexity measures.

3.1 Deterministic Query Complexity

It turns out that the query model is a much simpler model to judge the complexity of Boolean functions. There are different variants of query complexity measures, but the simplest one of them all is deterministic query complexity.

As a contrast, we first consider the time complexity of a Boolean function f. We know that there exist infinitely many Turing machines which are capable of:

1. determining the value $f(x)$ for a given input x correctly, for all inputs, and
2. halting thereafter.

For an input size n, every Turing machine will have a 'worst case running time', that is, the maximum running time required across all inputs of length n. We will pick the Turing machine which minimizes the worst case running time, for all sufficiently large n, and its worst case running time is defined to be the time complexity of f. It is clear from the definition that the running time is the constraint here, and we are interesting in minimizing it. Informally speaking, a function is thought to be more complex if the running time of the best possible Turing machine computing f is high.

In the query computation model, we don't have any restrictions on the running time, but the catch is that the input is not fully presented to us from the get go. Instead, the input is stored in an oracle, and we have to 'query' it. The oracle can be thought of as a function $O_x : [n] \to \{-1, 1\}$, and each possible input $i \in [n]$ to it is a query. $O_x(i)$ is equal to the value of the bit x_i. Fixing a particular function f, the goal is to calculate $f(x)$ correctly, on all possible inputs x, using as few queries as possible. Similarly as above, we can consider all Turing machines which are capable of querying the oracle, computing $f(x)$, then halting, and pick the one which minimizes the number of queries on the worst case input, for sufficiently large n. We denote this complexity measure by $\mathrm{D}(f)$. Clearly, $D(f) = 0$ for the two constant functions, while in the case of non-constant functions, $D(f) > 0$, and $D(f) \leq n$.

It is extremely useful to think of the query computational model as a binary tree, which we call as a decision tree. For a particular function f, and a TM with oracle access which is computing f, we can construct a decision tree as follows: The first index queried is independent of the input, clearly, because the TM can only gain knowledge about the input by querying the oracle. We highlight this fact by creating a new binary tree with a single root node, and this root node represents the first bit queried. We have two outcomes now, one where the queried bit was 1, and the other where it was -1. All subsequent bits queried will be conditioned on this fact. Therefore, it is natural to represent the two outcomes with the two branches

of the root node, and to represent the second bit queried under each outcome with the left and right child nodes. By going through all possible query outcomes, we can construct the complete binary tree. Once we reach the leaves, we are certain of the function value, and hence the leaves represent the final function value. Therefore, it can be easily seen that the number of queries made in the worst case is equal to the length of the longest path in the tree.

One nice way to interpret the decision tree structure is: For each bit queried, we are partitioning the input space under consideration into two partitions. The first query divides the entire input space $\{-1, 1\}^n$ into two partitions. Then, the left sub tree works on one partitions, and the right one on the other. We stop going down a query path only when we are sure that every input in the partition we created has the same function value. So in some sense, the goal can be seen as finding a way to partition the input space into 'unambiguous' blocks, while making as few queries as possible. In some cases, it may even be that several of these unambiguous blocks contain just a single input.

As a simple example, the dictator function Dict_i has deterministic query complexity equal to 1, because we just need to query the ith bit. At the other end of the spectrum, we have $D(\mathsf{OR}) = n$. Because of how OR is defined, if the algorithm (we will use the term *algorithm* instead of halting TM from now onwards) fails to query even a single bit, then we have two inputs having different values OR in a single block, the all ones input, and another with a -1 only at the position not queried. Hence, the block is ambiguous.

Broadly speaking, query computational models ask the following question: How much information about the input needs to be sought, in order to determine the function value on that input? The deterministic version requires the algorithm to emit the correct answer each time, but the next complexity measure that we are going to look at relaxes that restriction a bit.

3.2 Randomized Query Complexity

Looking at deterministic query complexity, a natural question one can ask is: what if the algorithm is not required to output the correct answer all the time, but is required to do so with high probability? Formally speaking, consider a Boolean function f, and consider a deterministic algorithm that has access to an oracle containing the input, and additionally, has the ability to sample strings from a set of strings S, according to a probability distribution $\mu(S)$. The deterministic algorithm samples from S once, and depending on the string it gets, it decides which bits to query, and then computes $f(x)$. Now it might turn out that for a particular input and for a particular string, the deterministic algorithm computes the wrong value, but this is fine. The requirement is only that for each input x, the algorithm outputs $f(x)$ with high probability. Put more formally, we require the following for each input x:

$$\sum_{s \in S} Pr_{s \sim \mu(S)}(s) \mathbb{I}[A(s, x) = f(x)] \geq 1 - \epsilon$$

Where A denotes the deterministic algorithm, and ϵ is some constant where $0 < \epsilon < 1/2$. Note that we need epsilon to be strictly lesser than half, otherwise we could have the trivial algorithm which simply guesses the value, and outputs the correct answer with probability $1/2$. Any algorithm (i.e, the deterministic algorithm with access to the oracle and sampling ability acc. to $\mu(S)$), which satisfies the above requirement is called a *randomized algorithm* for f.

How does one calculate the query cost of a randomized algorithm? Even for a fixed input, the algorithm might query a different number of inputs, depending on the string which was sampled. Hence, we take the expected cost over all strings. Taking $C_A(s, x)$ to be the number of queries made by the deterministic algorithm A on input x when string s has been sampled, we get:

$$C(x) := \mathbb{E}_{s \sim \mu(S)}[C_A(s, x)]$$

Finally, we minimize the cost $C(x)$ made in the worst case, over all randomized algorithms for f with error at most ϵ, and this is taken to be the *randomized query complexity* of f, denoted by $R_\epsilon(f)$.

Note that for any two constants ϵ_1 and ϵ_2, $R_{\epsilon_1}(f)$ need not necessarily be equal to $R_{\epsilon_2}(f)$, but we can easily see that if $\epsilon_1 \leq \epsilon_2$, then $R_{\epsilon_1}(f) \geq R_{\epsilon_2}(f)$. Moreover, the exact choice of constant need not matter, since we can always improve the error bound of any randomized algorithm to any arbitrary constant ϵ by repeatedly running it $O(log(1/\epsilon))$ times, and then taking a majority over the outputs. For this reason, we

can prove bounds on the randomized query complexity for a given function with respect to a single constant, and they automatically transfer over to every other constant, to within a constant multiplicative factor. For notational convenience, we take $R(f)$ to be equal to $R_{1/3}(f)$.

3.2.1 Interpretations

We can interpret a randomized algorithm for f as a decision tree too. In fact, there are two possible interpretations, and both of them turn out to be equivalent:

1. **Collection of decision trees:** This is the simpler interpretation of the two, but it is a bit difficult to see how it follows from the definition above. We first construct a bunch of decision trees as described in the previous section, and then we define some probability distribution over these decision trees. Upon receiving an input, the algorithm samples one of the decision trees according to the probability distribution, and then evaluates the decision tree in a deterministic manner.

2. **Representing bits from the string as nodes:** This interpretation follows easily from the definition. Each time the deterministic algorithm reads some particular bit of the string it is provided, there are two possible computation paths from that point, one if the bit read was a 0, the other if it was a 1. Therefore, we also treat every 'bit read' of the provided string as a node, and its two branches as computation paths. Therefore, instead of having multiple decision trees as in the first interpretation, we have one single decision tree, but its nodes can be of two types: one which represents 'bit reads' from the provided string, and one which represents oracle queries.

In the second interpretation, we can attach a probability to each branch of a node representing a 'bit read' from the string. For example, suppose the computation proceeds down the left branch if we read a 0. Then the probability attached to the left branch is equal to the sum of all probabilities of those strings in set S which have a 0 at that position. Therefore, the computation can proceed down either the left sub tree, or the right sub tree, according to the probabilities attached to their corresponding branches. Now suppose that there exist t nodes corresponding to a 'bit read' throughout the decision tree. Then, there are 2^t possible combinations of deterministic decision trees that we could have, where at every 'bit read' node 'fix' the sub tree that we go down. The probability attached to a particular decision tree would turn out to be the product of the probabilities of picking each of the individual sub trees, at the 'bit read' nodes. Therefore, we have a probability distribution over deterministic decision trees, and we have shown a method to go from the second interpretation to the first.

Note that for any given Boolean function f, $D(f) \geq R(f)$. This is because any deterministic query algorithm satisfies all the requirements for a randomized algorithm; indeed, it does not need to be given a string from set S, and it gives the correct answer with full probability. In fact, the deterministic query complexity of a function is an upper bound for every complexity measure that we will define subsequently.

3.3 Sensitivity, Block Sensitivity

We have already defined sensitivity in section 2.7.2. A related measure is block sensitivity. Generalizing the notation we introduced in section 2.7.2, we can define $x_{\oplus S}$, where $S \subseteq [n]$, is equal to x, but the bits at indexes belonging to S are flipped. We say that a Boolean function f is sensitive to S at input x if $f(x) \neq f(x_{\oplus S})$. We define the block sensitivity of f at x, denoted by $bs_x(f)$, to be the maximum number of disjoint subsets to which f is sensitive at x. We then define the block sensitivity of f, denoted by $bs(f)$, as the maximum of $bs_x(f)$ over all inputs x. From the definition, it is clear that $s(f) \leq bs(f)$, for all f, because each sensitive bit can be thought of as a disjoint block.

For example, $bs(\mathsf{OR}) = bs(\mathsf{AND}) = n$, because for the inputs 1^n and -1^n respectively, we can have n disjoint blocks for each function.

3.4 Degree of representing polynomial

We have seen this definition in section 2.5. This measure, denoted by $deg(f)$, is equal to the degree of the unique representing polynomial for f. The degree of a constant function is trivially 0. The lowest

possible degree any non-constant Boolean function having non-zero influence at every index is equal to $log(n) - O(loglogn)$, and this bound is tight upto polylog factors, as witnessed by the *addressing* function.

We can construct the representing polynomial f from any decision tree for f. Each leaf has a value which represents the value of the function, and it also has a unique path leading upto the root. Therefore, the idea is to build a sort of an indicator function for each leaf, such that a single particular function value is returned if, and only if the computation proceeds down the corresponding path. Multiplying the leaf value with its corresponding indicator function, and doing the same for the other leaves; and finally summing them all up gives us the desired polynomial. It is easy to check that the degree of this polynomial is at most the length of the longest path in the tree, hence for all Boolean functions f, $deg(f) \leq D(f)$.

3.5 Approximate degree

Often times, it is not necessary that we need a polynomial outputting the exact value of the function. We will be satisfied if it returns an approximate value as well. To put this formally, we say that a polynomial p approximates a Boolean function with error at most ϵ, if the following condition is satisfied:

$$|p(x) - f(x)| \leq \epsilon$$

Note that we require ϵ to be a constant, and also that its value be lesser than 1, otherwise the constant polynomial 0 would trivially satisfy the condition.

The approximating polynomial for f whose degree is a minimum, is known as the *ϵ-approximating polynomial for f with the least degree*, and its degree is known as the *ϵ-approximate degree* of f, denoted by $\widetilde{deg}_\epsilon(f)$. As in the case of randomized query complexity, the exact value ϵ is of little importance, as we can in principle extend the bounds to any arbitrary constant, with only a constant factor increase in degree (the method is fully described in [Tal14]). For convenience, we take $\widetilde{deg}(f) := \widetilde{deg}_{1/3}(f)$.

It was shown by Paturi [Pat92] that the approximate degree of all symmetric functions is fully characterized by the distance of the closest 'jump' from the midpoint, when looking at the corresponding univariate version of the function. The lesser the distance, the harder it is to approximate the function. For PAR and the majority function, this is equal to 0 ± 1, for OR and AND it is equal to $n/2$.

Approximate degree is an interesting measure to study in its own right, but nevertheless, it still has many important applications in theoretical computer science. For example, approximate degree lower bounds have been used to exhibit tight quantum query complexity lower bounds [BBC+01], [AS04], [KSdW07], various lower bounds on protocols in communication complexity [CA08], [She16], [She08], along with fundamental circuit complexity lower bounds [Bei93], [She07].

3.6 Sign Degree

This notion is similar to approximate degree, but we relax the restrictions here. We allow a polynomial to be even unbounded, as long as the polynomial agrees in sign with the Boolean function, on every input. More precisely, the sign degree of a Boolean function f, is the degree of the polynomial having the least degree among polynomials satisfying:

$$sgn(p(x)) = f(x)$$

for all inputs $x \in \{-1, 1\}^n$. It is denoted by $deg_\pm(f)$. From the definition, it is clear that for every Boolean function f, $\widetilde{deg}(f) \geq deg_\pm(f)$.

We know of large separations between approximate degree and sign degree, especially for symmetric functions. All threshold functions (functions which take value 1 for all x where $|x| \leq t$, and -1 otherwise, where t is some integer such that $0 \leq t \leq n$) exhibit a separation. For example, $\widetilde{deg}(\mathsf{OR}) = \Omega(\sqrt{n})$, and $deg_\pm(\mathsf{OR}) = O(1)$. The largest possible separation is acquired by the MAJ function, for which $\widetilde{deg}(\mathsf{MAJ}) = \Omega(n)$, and $deg_\pm(\mathsf{MAJ}) = O(1)$.

3.7 Relations and separations between complexity measures

From the definitions of these complexity measures, it is apparent that there exist a few relations between these measures, which hold for all Boolean functions. One that we already stated above is of the form:

$$\forall f \quad \widetilde{deg}(f) \leq deg(f)$$

The complexity measures we defined in the previous section are all related to one another in a stronger sense: They are polynomially related to one another. This means that for any two complexity measures A and B defined above, there exist constants α and β, $\alpha < \beta$ such that the following holds for all Boolean functions f:

$$A(f)^{\alpha} \leq B(f) \leq A(f)^{\beta}$$

Intuitively, this means that for any given function f, none of these measures can drastically exceed the other, and they are 'within range' of one another.

Example: For $deg(f)$ and $D(f)$, the best known constants α and β are 1 and 3, respectively. That is:

$$\forall f \quad deg(f) \leq D(f) \leq deg(f)^3$$

Separations are the other side of the coin. For every pair of polynomially related complexity measures, there exists some boolean function for which the 'difference' between the two complexity measures is the largest, among all Boolean functions. For example, in the case of approximate degree and degree and degree, we have the following polynomial relation:

$$\forall f \quad \widetilde{deg}(f) \leq deg(f) \leq \widetilde{deg}(f)^4$$

the best separation we have, as of now, for the approximate degree and degree is given by the OR function. We have:

$$\widetilde{deg}(\mathsf{OR}) = O(\sqrt{n})$$

and:

$$deg(\mathsf{OR}) = \Omega(n)$$

This implies:

$$\exists f : \quad deg(f) \leq \widetilde{deg}(f)^2$$

It is believed that this is the best possible separation between degree and approximate degree. All relations and separations stated above — and the ones which will be stated from this point onwards — are given in the form of a table in [ABDKT20].

Note that if two complexity measures are polynomially related, then there exists a single constant γ, which describes both the optimal relation and the optimal separation. To state this more formally, suppose we have two complexity measures A and B that are polynomially related. Then, in particular, for some constant β, we know that:

$$\forall f \quad B(f) \leq A(f)^{\beta}$$

But it might be the case that this bound is not tight. For example, it might be true that for all f, the following statement holds as well: $B(f) \leq A(f)^{\beta-\epsilon}$, for some constant $\epsilon > 0$.

We say that a relation is tight when we have a constant γ such that:

$$\forall f \quad B(f) \leq A(f)^{\gamma+o(1)}$$

and, also:

$$\exists g \quad B(f) \geq A(f)^{\gamma-o(1)}$$

The $o(1)$ exists so that we can ignore polylog factors. The function g is a 'witness' to the separation between the two complexity measures.

Among all the measures listed above, all of them were known to be polynomially related to each other, except for sensitivity, until Hao Huang published a breakthrough proof [Hua19] which showed that sensitivity is polynomially related to the degree of the function. Since then, his technique has been used in [ABDKT20] to show the optimal relation between deterministic and quantum query complexities. We will discuss quantum query complexity in the next chapter.

Chapter 4

The Quantum Query model

4.1 Introduction

The classical computers that we see around us today operate based on the principles of classical physics. But since the beginning of the 20th century, we know that our world is inherently quantum mechanical in nature, which often not only goes against our intuition of how the world works, but also sometimes contradicts the laws of classical physics. For example, a quantum system can exist in a superposition of multiple states, something which is not possible in a classical system. Realizing the disconnect between the classical and quantum theories, the field of quantum computing seeks to answer the following question: "Is it possible to build systems taking advantage of quantum mechanical principles, which are significantly more powerful than classical computers?"

In 1982, Richard Feynman mentioned that scientists looking to simulate quantum systems on classical computers were running into scaling problems, that is, as the size of the system increased linearly, the requirements of simulating it would grow exponentially. He suggested building computers based on the principles of quantum mechanics to overcome this issue. Nothing much happened after that, until Peter Shor's breakthrough findings in 1994 [Sho94], which established that the problems of integer factoring and discrete logarithm can be solved efficiently by quantum computers. This was remarkable because both problems are generally thought to have no efficient classical algorithm. Soon after, in 1995, Lov Grover gave a quantum algorithm [Gro96] for unstructured search, which shows a quadratic speedup over best possible classical algorithms. Bernstein and Vazirani gave a formal, mathematical description of a Quantum Turing machine in [BV97], where they also prove that every problem which is efficiently solvable by a Turing machine can also be efficiently solved by a Quantum Turing machine.

The Church-Turing thesis states that the Turing model of computation is as powerful as any other model of computation. That is, if a problem is efficiently solvable in some computational model, then that problem can also be efficiently solved in the Turing model. Here, by efficiently solvable, we mean that the number of steps taken to solve the problem is at most polynomial in the input size.

We know that Shor's algorithm for integer factoring runs in polynomial time on a quantum computer (in other words, requires a polynomial number of quantum gates), while the best known classical algorithms require near-exponential time. If it is discovered that integer factoring cannot be solved in polynomial time on a classical machine, then this would imply that the Church Turing hypothesis is false.

However, for various reasons, it is difficult to estimate the number of quantum gates required to solve any given problem, and hence, we run into issues while trying to determine the time complexity of any given problem on a Quantum Turing machine. On the other hand, it has been noticed that by using another related model known as the *query model*, we are able to capture the power of quantum computers over classical computers extremely well. We will first go over the basics of quantum computing, and then introduce the quantum query model, and finally, give examples of problems where quantum algorithms outperform classical ones, within the query model.

4.2 Basics

One of the basic components of quantum computing is the principle of *superposition*. A classical bit can be either 0 or 1 at any given point of time. But a qubit (quantum bit) can exist as both 0 and 1 at any given time. Formally, taking $|0\rangle$ and $|1\rangle$ to denote the 0, 1 classical states respectively, we can mathematically describe the state of a qubit as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where ψ can be thought of as a label with which we refer to the qubit, and α and β are complex numbers. α and β are known as the amplitudes of $|0\rangle$ and $|1\rangle$, respectively. The reason why the amplitudes are complex has to do with the postulates of quantum mechanics, but we need not go into it here. There is a further condition imposed on α and β:

$$|\alpha|^2 + |\beta|^2 = 1$$

When a qubit is initially prepared, we say that it exists in a superposition of $|0\rangle$ and $|1\rangle$, meaning that it exists as both $|0\rangle$ and $|1\rangle$ at the same time, each state having some amplitude. When we measure the qubit, we say that it 'collapses' to one of the classical states. It collapses with probability $|\alpha|^2$ to state $|0\rangle$, and with probability $|\beta|^2$ to state $|1\rangle$. Note that because of the above conditions on them, both $|\alpha|^2$ and $|\beta|^2$ are real non-negative numbers which sum up to 1, and hence the qualify as probabilities of all possible outcomes.

Once a qubit has been measured and has collapsed to either of the classical (also known as ground) states, it is impossible to recover the original quantum state. Hence, it is impossible determine the values of α or β by measuring just a single qubit. However, we can get arbitrarily close approximations if we have the ability to produce multiple qubits in the same state. Once created, we measure all the qubits, and the fraction of qubits collapsing to $|0\rangle$ will give an approximation for $|\alpha|^2$, and the fraction of qubits collapsing to $|1\rangle$ will give an approximation for $|\beta|^2$.

Instead of working with a single qubit, we can work with multiple qubits at a time. Given a register of n classical bits, they can exist in one of 2^n classical states at any given point of time. However, a register of n qubits can exist in a superposition of all these states:

$$|\psi\rangle = \sum_{x=0}^{2^n-1} \alpha_x |x\rangle$$

where we have represented the 2^n ground states as $|0\rangle, |1\rangle, |2\rangle, \ldots |2^n - 1\rangle$. By generalizing the above restriction on amplitudes, we require that:

$$\sum_{x=0}^{2^n-1} |\alpha_x|^2 = 1$$

Apart from measurement, we can also choose to manipulate the amplitudes of the quantum system, without collapsing it. There are certain restrictions on the manipulations that one is allowed to perform. The square of the new amplitudes must still sum up to one, as they must still represent probabilities. One nice way of describing these manipulations (which we call operations) is to first represent the quantum state as a vector, with the amplitudes being the entries present in the vector, and representing the operation as a unitary matrix. The product of this unitary matrix with the vector gives us a vector corresponding to the transformed state.

Formally, any n-state quantum system can be completely described by a 2^n-dimensional vector $|\psi\rangle \in \mathbb{C}^{2^n}$:

$$|\psi\rangle = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{2^n} \end{bmatrix}$$

where the ith entry denotes the amplitude of the ith state. So, for example, the state $\alpha|0\rangle + \beta|1\rangle$ is represented by:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

and the ground state $|0\rangle$ is represented by:

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

By observing the restriction on the amplitudes, we notice that the vector representing a quantum state must have an L_2 norm equal to 1. Conversely, any vector $v \in \mathbb{C}^{2^n}$ with L_2 norm equal to 1 represents an n-qubit quantum state.

Focusing on operations, we notice that any operation performed on a quantum state must result in a quantum state whose corresponding vector representation still has an L_2 norm equal to 1. In other words, the operation is not allowed to increase the magnitude of the vector, but only rotate it. It turns out that this is the only restriction which we need to impose on the operations, and we already know linear algebra that a unitary matrix has the capability of rotating any given vector, without changing its magnitude. Thus, we can completely represent quantum operations by unitary matrices. A trivial example is the identity matrix, which leaves the quantum state unchanged. Here is another simple example, which switches the amplitudes associated with the ground states:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

Another important example is the Hadamard matrix. For the single qubit case, it defined as:

$$H_1 := \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

For the n-qubit case, we define it in an inductive manner:

$$H_n := \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix}$$

The operation associated with the Hadamard matrix sends an n-qubit register present in the ground state of $|0\rangle$ to the equal superposition state, which is that quantum state in which each and every ground state has the same amplitude (which for the n-qubit case, is equal to $1/\sqrt{2^n}$).

$$H_n|0\rangle = \sum_{x=0}^{2^n-1} \frac{1}{2^{n/2}}|x\rangle$$

One last requirement that a quantum operation needs to have is that it must be reversible. This seems odd, as such a requirement does not exist in the classical world. Indeed, there is no way of knowing what the input to an AND operation was, from its output. But in the quantum realm, such a system cannot exist. The information contained within a system cannot decrease, and each operation must be reversible. In the vector representation, this means that for any state $\psi\rangle$, and for any operation denoted by the unitary matrix U, there must exist a U' such that:

$$U'(U|\psi\rangle) = |\psi\rangle$$

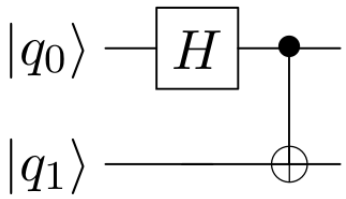
But we already know that such a U' exists, and it is equal to $U^\dagger$, since $U^\dagger U = I$, and therefore:

$$U^\dagger(U|\psi\rangle) = (U^\dagger U)|\psi\rangle = I|\psi\rangle = |\psi\rangle$$

(Note that $U^\dagger$ denotes the adjoint of U).

An interesting example of an inverse is the Hadamard matrix, which is its own inverse. Hence, by applying the Hadamard matrix to the equal superposition state, one can get back to the ground state of $|0\rangle$.

Finally, we can fully describe a quantum algorithm in a diagrammatic fashion, in the form of circuits, with quantum gates implementing the operations described by unitary matrices. Here is an example of a quantum circuit which takes in two qubits, $q0$ and $q1$ as inputs, and first applies the Hadamard gate on $q0$, and then the CNOT (controlled-NOT) gate on both the qubits.



The CNOT gate here acts on two qubits: If the first bit is 0, it does nothing. If the first bit is 1, it flips the second bit. Here is the unitary matrix corresponding to CNOT:

$$\text{CNOT}_2 := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

As you can see, the amplitudes of states $|00\rangle$ and $|01\rangle$ (ones which correspond to the first bit being 1) are not changed, but the ones belonging to states $|10\rangle$ and $|11\rangle$ are exchanged with one another.

4.3 Quantum queries and Quantum Query complexity

A quantum query algorithm can query the input bits in superposition, and this offers a certain advantage over its classical counterpart.

There are two types of quantum queries, and both are equivalent, in the sense that a quantum algorithm making use of one type cannot make significantly less queries than it can while using the other. We shall concern ourselves with only the first type here.

A quantum query algorithm which queries an N-bit input has the following qubit register form:

$$|i, b, z\rangle$$

where i is at represents the index that we want to query, and it ranges from 0 to at most $\lceil logN \rceil$. b can be thought of the location where the ith input bit is stored, and z represents the workspace, that is, those qubits which is used by the algorithm to perform computations.

An oracle O acts on a state $|i, b, z\rangle$ in the following manner:

$$|i, b, z\rangle \xrightarrow{O} |i, b \oplus x_i, z\rangle$$

That is, the bit represented by b is flipped if and only if x_i equals 1. Taking $n := \lceil logN \rceil$, and assuming an m-qubit workspace, the algorithm acts on a total of $n + 1 + m$ qubits, and therefore, the state can be described as a linear combination of 2^{n+1+m} basis states.

A quantum algorithm making T queries can be represented as a sequence of unitary transformations such as this:

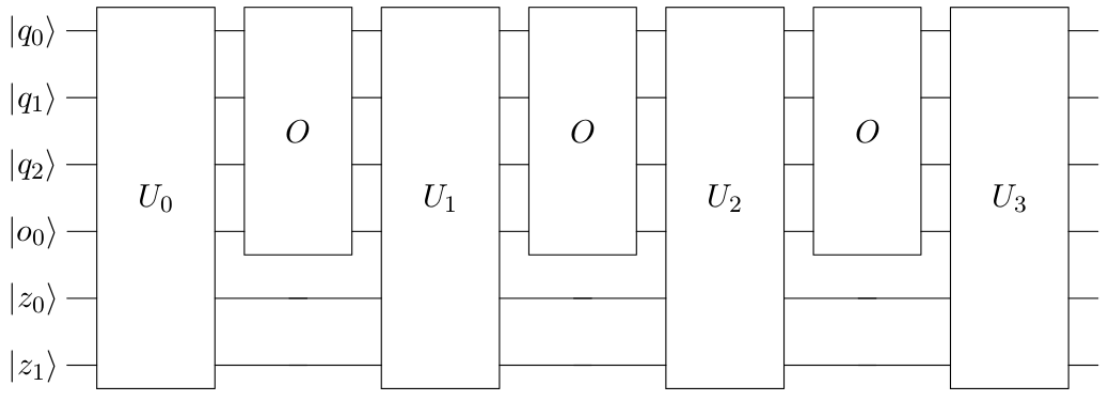
$$U_T O U_{T-1} O \dots U_1 O U_0 |0\rangle$$

We generally start with the entire qubit register initialized to state $|0\rangle$. The oracle operations are unitary operations as well. Note that we have represented the intermediate computations between the oracle queries with just a single unitary, because the product of multiple unitary matrices is a unitary matrix as well.

We generally use a single qubit from the workspace portion of the qubit register to denote the output. If after the computations, we measure the qubit register and the observed value is taken to be the value of the function, as calculated by the algorithm. Naturally, the goal of a quantum algorithm is to calculate the function value for all inputs with high enough probability. We would like that the probability to be some constant strictly greater than half, so that the algorithm is non-trivial. Generally, we take the probability

to be 2/3. In a manner analogous to randomized algorithms, we minimize over the worst-case number of queries made by all quantum algorithms which calculate a Boolean function f with probability 2/3, and call this quantity the quantum query complexity of f, and denote it by $Q(f)$. As in the case of randomized algorithms, the success probability 2/3 is arbitrary, and any given probability can be achieved with an increase of at most a constant multiplicative factor.

A 3-query quantum circuit operating on two query qubits ($|q_0\rangle, |q_1\rangle$), one output qubit ($|o_0\rangle$), and two workspace qubits ($|z_0\rangle, |z_1\rangle$) is shown below:



In 1996, Grover showed a quantum query algorithm [Gro96] for the OR function which used $\Theta(\sqrt{N})$ queries. This was the first total Boolean function for which a separation between the randomized and quantum query complexities was shown (the randomized query complexity of OR being $\Theta(n)$). However, in 1998, Beals, Buhrman, Cleve, Mosca, and de Wolf [BBC+01] showed that a polynomial separation between randomized and quantum query complexities was the best possible one, when one considered total Boolean functions. Specifically, they showed:

Theorem 4.3.1. *[BBC+01] $D(f) = O(Q(f)^6)$ for all total Boolean functions f.*

However, there exist superpolynomial separations for partial Boolean functions. For example, Simon [Sim94] showed a quantum algorithm for a problem which made exponentially fewer queries than the best possible classical algorithm. The problem is as follows: given a multi-output function $f : \{0,1\}^n \to \{0,1\}^n$, with the promise that $f(x) = f(y)$ if and only if $y = x \oplus s$, for some fixed, nonzero $s \in \{0,1\}^n$, the task is to find s. Simon was able to show that every classical algorithm which correctly computes the values of s requires $\Omega(2^{n/2})$ queries, while a quantum algorithm can compute s with only $O(n)$ queries.

4.4 Lower Bounds via Polynomial method

In 1998, Beals, Buhrman, Cleve, Mosca, and de Wolf [BBC+01] gave a technique to prove lower bounds on the quantum query complexity of a Boolean function, via a technique known as the *polynomial method*. Basically, they were able to show that the approximate degree of a Boolean function forms a lower bound for the quantum query complexity. More precisely:

Theorem 4.4.1. *[BBC+01] Given a Boolean function f (total or partial) whose quantum query complexity is $Q(f) = t$, it is possible to construct a real-valued polynomial whose variables are the inputs provided to the function, whose degree is at most $2t$, and is an approximating polynomial for f. Therefore, $Q(f) \geq \widetilde{deg}(f)/2$ for every Boolean function f.*

Proof. The proof is constructive, in the sense that given a quantum algorithm which queries the input bits and computes f, we can construct an approximating polynomial for f. The proof relies on the following lemma:

Lemma 4.4.2. *The amplitudes of the ground states in every t-query quantum algorithm can be represented by a polynomial having complex coefficients, whose variables are the function inputs, and whose degree is at most t.*

From the above lemma, we see that the probability of a certain ground state being measured can be written as $|C_i(x)|^2$, where x is the function input, and C_i is the polynomial of degree at most t, which describes the amplitude of the ith ground state. Notice that $|C_i(x)|^2$ is a polynomial having real coefficients, and of degree at most $2t$. If $S \subseteq [2^{n+1+m}]$ is the subset containing all the accepting states, then the polynomial:

$$p(x) := \sum_{i \in S} |C_i(x)|^2$$

represents the sum of probabilities of all accepting states. This is a polynomial of degree $2t$ as well, and since we know that for any input x which is accepted by the algorithm, the sum of the probabilities of the accepting states must be at least $2/3$, we know that $p(x)$ must therefore, be at least $2/3$. By a similar argument, $p(x)$ is atmost $1/3$ for any input x which is rejected by the quantum algorithm. Now all that is left is to prove the intermediate lemma. □

Proof. of Lemma 4.4.2: We can prove the lemma by an inductive argument. The ground state amplitudes when the oracle has not been queried yet can be represented by constant polynomials, since they will not depend upon the input in any form. Therefore, the theorem holds true when $t = 0$. Now assume it is true for all integers until $t-1$. The ground state amplitudes after the action of the $t-1$th oracle query can then be written as polynomials of degree at most $t-1$. Note that the operations described by unitary matrices simply result in new amplitudes which are linear combinations of the older ones, and hence the unitary operations do not increase the degree.

Recall that an oracle maps a state $|i, b, z\rangle$ to $|i, b \oplus x_i, z\rangle$. If the amplitude of states $|i, 0, z\rangle$ and $|i, 1, z\rangle$ were α and β before the tth query (both polynomials of degree at most $t-1$), then the amplitude of state $|i, 0, z\rangle$ after the tth query is $(1 - x_i)\alpha + x_i\beta$, and the amplitude of state $|i, 1, z\rangle$ becomes $x_i\alpha + (1 - x_i)\beta$. Therefore, we see that the new amplitudes are described by polynomials of degree at most t. Since this holds for all i and z, this completes the proof. □

In [AS04], Aaronson and Shi showed lower bounds for the quantum query complexities of collision and element distinctness by using the polynomial method. In [Amb03], Ambainis exhibited a function showing a separation between quantum query complexity and approximate degree. However, this function was quite unnatural. In 2018, Sherstov showed [She18] that the Surjectivity function exhibits a separation between the two measures as well. In [ABP19], Arunachalam, Briët, and Palazuelos provided a modified formulation of approximate degree which is equivalent to the quantum query complexity, for all (total and partial) Boolean functions.

Chapter 5

The Dual method

5.1 Introduction

Besides establishing lower bounds on approximate degree in order to show lower bounds on quantum query complexities of Boolean functions, the characterization of approximate degree has other important applications in theoretical computer science, as detailed in section 3.5. Despite this fact, the approximate degree of several important functions is still unknown, and its behaviour with respect to composed functions is still uncertain.

Minsky and Papert's symmetrization technique in [MP69] gave a method to show lower bounds for approximate degree, but the technique is not lossless, as information is potentially lost in the symmetrization process, and hence the resulting lower bound may not be tight. The technique involves converting a multivariate, multilinear polynomial of degree d into a univariate polynomial having degree at most d, and then proving lower bounds on this univariate polynomial.

Paturi's work [Pat92] fully characterized the approximate degree of all symmetric functions, but the chracterization of non-symmetric functions is still open to this day. In 2008, a new technique was outlined in [Spa08], known as the dual technique, which formulates the notion of approximate degree as a linear program. A feasible solution for the dual of this linear program is a proof of a lower bound on the approximate degree, and we call the feasible solution a *witness*. This technique is lossless, that is, it is able to exhibit tight lower bounds, and moreover, it works for any Boolean function, be it total or partial.

5.2 Literature overview

The idea of formulating the notion of ϵ-approximate degree as a linear program was first introduced by Špalek in [Spa08]. He also derives the dual of the linear program, and then gives a dual which witnesses a tight lower bound of $\Omega(\sqrt{n})$ for the OR function. This was a univariate dual, and it exhibited the property of polynomial decay, that is, its absolute value was at most $O(1/k^2)$, if k was the input given to the dual.

In [BT13], Bun and Thaler outlined a method to produce a dual witnessing a tight lower bound for every symmetric function. It was inspired by Paturi's result [Pat92], which characterized the approximate degree of all symmetric functions.

Regarding composition of functions, Sherstov [She09] and Troy Lee [Lee09] independently gave a proof for the following theorem:

Theorem 5.2.1. *For all total Boolean functions f and g:*

$$\widetilde{deg}(f(g(x_1),\ldots,g(x_n))) \geq \Omega(\widetilde{deg}(f)\deg_{\pm}(g))$$

To prove this result, they came up with a method for composing the duals of f and g. While this method does not give us the conjectured tight lower bound of $\Omega(\widetilde{deg}(f)\widetilde{deg}(g))$, it has found applications in other places. For example, it was used by Bun and Thaler in [BT13] to give a tight lower bound for $\mathsf{AND} \circ \mathsf{OR}$. As a by product of his work on direct product theorems, Sherstov was able to show a tight lower bound for

$\mathsf{PAR} \circ f$, for any total Boolean function f in [She11]. This work introduced the notion of multiplying the composed dual with another polynomial which does not affect the pure high degree much, but eliminates the errors caused by the inner dual.

In the realm of partial functions, Bun, Kothari and Thaler's paper [BKT18] proves lower bounds for a range of composed partial functions. A dual for a particular promise variant of the $\mathsf{AND} \circ \mathsf{OR}$ function was used to show tight lower bounds for functions like *Surjectivity*, *k-distinctness*, and *Image Testing*.

Bogdanov et al. showed a multivariate dual for AND in [BMTW19], as a special case of their result for a tight lower bound on the weighted approximate degree.

5.3 The Dual Method

In this section, we show the linear program which captures the notion of approximate degree of a Boolean function, and show how the dual acts as a proof for the lower bound. In this section, we only consider total Boolean functions. Partial Boolean functions can have two notions of approximate degree, and we will go over both of them in the next section.

Fix a total Boolean function f. Our aim now is to create a linear program whose constraints are imposed by the requirements of any ϵ-approximating polynomial of degree at most d: that it should be ϵ-close to f for all inputs, and all non-zero Fourier coefficients' corresponding monomial sizes should be at most d. Moreover, any feasible solution to this linear program must be an ϵ-approximating polynomial of degree at most d, and conversely, any such polynomial should be a feasible solution. One can easily see that the linear program described below captures all such required notions:

$$\begin{aligned} &\min \epsilon \\ \sum_{S \subseteq [n]} \hat{p}(S)\chi_S(x) + \epsilon &\geq f(x) \quad \forall x \in \{-1,1\}^n \\ \sum_{S \subseteq [n]} \hat{p}(S)\chi_S(x) - \epsilon &\leq f(x) \quad \forall x \in \{-1,1\}^n \\ \hat{p}(S) &= 0 \quad \forall S \text{ s.t. } |S| > d \end{aligned}$$

The variables in this linear program are the Fourier coefficients $\hat{p}(S)$, and ϵ. Note that fixing an order for all $S \subseteq [n]$, and negating the second constraint on both sides allows us to write the linear program constraints in the following standard form:

$$\begin{aligned} &\min c^T x \\ &Ax \geq b \end{aligned}$$

where the variable vector x has the Fourier coefficients $\hat{p}(S)$ arranged according to the aforementioned order, and the ϵ appended at the end. The vector c consists of all zeroes, except for a 1 at the very end. The constraint matrix A has two rows corresponding to the two constraints which are imposed for every input, and an additional $2(n-d)$ rows for the degree constraints. The b column vector simply contains the RHS from all the constraint inequalities.

Now, it is clear that the objective for the dual is to maximize $b^T y$, subject to the constraints $A^T y = c$, and $y \geq 0$, where y is the vector containing the dual variables. We take the variables corresponding to the two constraints for each input to be $\psi_1(x)$ and $\psi_2(x)$, respectively. The variables corresponding to the degree constraints are taken as $\delta_1(S)$ and $\delta_2(S)$. From the nature of the inequalities in the primal, we obtain the following constraints on the dual:

$$\begin{aligned} \psi_1(x), \psi_2(x) &\geq 0 \quad \forall x \\ \delta_1(S), \delta_2(S) &\geq 0 \quad \forall S \subseteq [n] \text{ s.t. } |S| > d \end{aligned}$$

The constraint obtained from the last column of A and the last entry of c (which was a 1) is:

$$\sum_x (\psi_1(x) + \psi_2(x)) = 1$$

The constraints from the foremost 2^n columns are:

$$\sum_x (\psi_1(x) - \psi_2(x))\chi_S(x) = 0 \quad \forall S \text{ s.t. } |S| \leq d$$

$$\sum_x (\psi_1(x) - \psi_2(x))\chi_S(x) + \delta_1(S) - \delta_2(S) = 0 \quad \forall S \text{ s.t. } |S| > d$$

Weak duality implies that $b^T y \leq c^T x$ must always hold true, for every feasible dual solution y and primal solution x. This implies that if for some ϵ, a feasible solution for the dual is found such that the corresponding dual objective is strictly greater than ϵ, then there does not exist any primal solution which achieves a primal objective of ϵ. In other words, the dual solution is a proof of non-existence of an ϵ-approximating polynomial for f, of degree at most d. Conversely, if an ϵ-approximating polynomial of degree at most d exists, then there does not exist any dual solution whose objective value exceeds ϵ. Because of these properties, the dual method is *lossless*, in that a dual solution will always exist for the tightest possible lower bound.

We say that a polynomial *witnesses* a lower bound of d on the ϵ-approximate degree of f if it is a solution to the dual whose objective value exceeds ϵ, and the polynomial is known as the *dual* for the ϵ-approximate degree of f. Although the technique is lossless, duals witnessing tight lower bounds are hard to obtain.

Therefore, in summary, a dual must exhibit the following properties:

$$\sum_x (\psi_1(x) - \psi_2(x))f(x) > \epsilon$$

$$\sum_x (\psi_1(x) - \psi_2(x))\chi_S(x) = 0 \quad \forall S \text{ s.t. } |S| \leq d$$

$$\sum_x (\psi_1(x) - \psi_2(x))\chi_S(x) + \delta_1(S) - \delta_2(S) = 0 \quad \forall S \text{ s.t. } |S| > d$$

$$\sum_x (\psi_1(x) + \psi_2(x)) = 1$$

$$\psi_1(x), \psi_2(x) \geq 0 \quad \forall x$$

$$\delta_1(S), \delta_2(S) \geq 0 \quad \forall S \subseteq [n] \text{ s.t. } |S| \leq d$$

We can frame the dual solution in a much more succinct way, though. Since the functions δ_1 and δ_2 appear only in one constraint (apart from the constraint that they should be non-negative), we can set their values as dictated by ψ_1 and ψ_2, and forget about them entirely, while focusing on the latter pair of functions. We can also assume $\phi_1(x)\phi_2(x) = 0$, for all inputs x, because if not, we can always subtract $\min(\phi_1(x), \phi_2(x))$ from both quantities, and ϕ_1 and ϕ_2 would still satisfy all the constraints. Then, taking $\phi(x) := \phi_1(x) - \phi_2(x)$, we get $\psi_1(x) + \psi_2(x) = |\psi(x)|$. This allows us to rewrite the constraints as:

$$\sum_x |\psi(x)| = 1$$

$$\sum_x \psi(x)f(x) > \epsilon$$

$$\sum_x \psi(x)\chi_S(x) = 0 \quad \forall S \subseteq [n] \text{ s.t. } |S| \leq d$$

Thinking of ψ and f as vectors in $\mathbb{R}^{2^n}$, the first condition can be thought of as the L_1 norm of vector ψ being equal to 1, and the second condition can be though of as the inner product, or correlation between the vectors ψ and f being greater than ϵ. The third condition implies that no monomials of degree lesser than or d can have non-zero Fourier coefficients in the Fourier expansion of ψ. Intuitively, the dual ψ represents that part of f's Fourier spectrum which is hard for any low degree polynomial to approximate correctly. We say that a function $\mathbb{R} \to \{-1, 1\}$ has pure high degree d if it does not have any non-zero coefficients for monomials of size lesser than d. Therefore, the third condition requires that the pure high degree of the dual is at least d.

5.4 The Dual Method for partial functions

There exist two notions of approximate degree for partial functions in literature: approximate degree, and unbounded approximate degree.

Definition 5.4.1. *For a partial Boolean function f with domain $X \subseteq \{-1,1\}^n$, the ϵ-approximate degree of f is equal to the degree of the polynomial having least degree, among all polynomials which successfully approximate f to within ϵ for all inputs within the domain (i.e, $|p(x) - f(x)| \leq \epsilon$ for all $x \in X$), and satisfy $|p(x)| \leq 1 + \epsilon$ for all inputs outside of f's domain. We denote it by $\widetilde{deg}(f)$.*

We can easily modify the linear program above to satisfy the additional properties required by this definition:

$$\begin{aligned}
&\min \epsilon \\
\sum_{S \subseteq [n]} \hat{p}(S)\chi_S(x) + \epsilon &\geq f(x) \quad \forall x \in X \\
\sum_{S \subseteq [n]} \hat{p}(S)\chi_S(x) - \epsilon &\leq f(x) \quad \forall x \in X \\
\sum_{S \subseteq [n]} \hat{p}(S)\chi_S(x) &\leq 1 + \epsilon \quad \forall x \notin X \\
\sum_{S \subseteq [n]} \hat{p}(S)\chi_S(x) &\leq -(1 + \epsilon) \quad \forall x \notin X \\
\hat{p}(S) &= 0 \quad \forall S \text{ s.t. } |S| > d
\end{aligned}$$

Going through the process of taking the dual and simplifying it again, we get:

$$\begin{aligned}
\sum_x |\psi(x)| &= 1 \\
\sum_{x \in X} \psi(x) f(x) - \sum_{x \notin X} |\psi(x)| &> \epsilon \\
\sum_x \psi(x)\chi_S(x) &= 0 \quad \forall S \subseteq [n] \text{ s.t. } |S| \leq d
\end{aligned}$$

Definition 5.4.2. *For a partial Boolean function f with domain $X \subseteq \{-1,1\}^n$, the ϵ-unbounded approximate degree of f is equal to the degree of the polynomial having least degree, among all polynomials which successfully approximate f to within ϵ for all inputs within the domain (i.e, $|p(x) - f(x)| \leq \epsilon$ for all $x \in X$). We denote it by $\widetilde{ubdeg}(f)$.*

This definition eases up on the restrictions, as compared to the earlier definition, as it allows the polynomial to be unbounded on inputs outside the domain. Notice that these definitions are the same when applied to Boolean functions. It is easy to see that $\widetilde{ubdeg}(f) \leq \widetilde{deg}(f)$, for all Boolean functions.

The linear program for this definition is the same as above, except for the constraints on inputs outside the domain. The dual for this linear program is:

$$\begin{aligned}
\sum_x |\psi(x)| &= 1 \\
\sum_{x \in X} \psi(x) f(x) &> \epsilon \\
\sum_{x \notin X} \psi(x) &= 0 \quad \forall x \in X \\
\sum_x \psi(x)\chi_S(x) &= 0 \quad \forall S \subseteq [n] \text{ s.t. } |S| \leq d
\end{aligned}$$

Chapter 6

Large Error Degree Composition

One of the tasks of characterizing approximate degree is to obtain the approximate degree of composed Boolean functions in terms of the approximate degrees of the component Boolean functions. In [She12], Sherstov solved one half of the equation, where he showed that for all Boolean functions f and g,

$$\widetilde{deg}(f(g(x_1),\ldots,g(x_n))) = O(\widetilde{deg}(f)\widetilde{deg}(g)).$$

Note that the naive method of composing the approximating polynomials of f and g won't necessarily work here, since the approximating polynomial for f will be fed inputs from $([-1-\epsilon,-1+\epsilon]\cup[1-\epsilon,1+\epsilon])^n$, and not from $\{-1,1\}^n$, and we don't know how the polynomial will behave on inputs from outside the Boolean hypercube.

This upper bound is obtained by showing that for every approximating polynomial of degree d, a polynomial can be constructed which will approximate f even when given 'noisy' inputs, coming from the lower approximating polynomial. Additionally, this new polynomial, which is known as a robust polynomial, only has an additive $log(1/\delta)$ increase in the degree, where δ is the maximum error. Prior to this work, the existence of a robust polynomial for parity was shown in [BNRdW05], which proved it via an intermediate result stating that any quantum query algorithm making t queries to evaluate a function implies the existence of a robust approximating polynomial for the function, of degree at most $2t$. Sherstov uses a different approach, and gives an explicit robust polynomial for parity by using an analytical argument, and is able to generalize it to give a robust polynomial for all polynomials which approximate Boolean functions.

This upper bound in conjectured to be tight, that is, it is believed that for all Boolean functions f and g:

$$\widetilde{deg}(f(g(x_1),\ldots,g(x_n))) = \Omega(\widetilde{deg}(f)\widetilde{deg}(g)).$$

Sherstov and Lee independently were able to prove a weaker version of this bound in [She09] and [Lee09], respectively. They proved that the following holds for all Boolean functions f and g:

$$\widetilde{deg}(f(g(x_1),\ldots,g(x_n))) = \Omega(\widetilde{deg}(f)deg_{\pm}(g)).$$

They showed this by giving a black box method for composing the duals of f and g. We state and prove the method here:

Theorem 6.0.1. *[She09], [Lee09] Take two Boolean functions, $f:\{-1,1\}^n\to\{-1,1\}$, and $g:\{-1,1\}^m\to\{-1,1\}$. Take the dual witnessing a tight lower bound of $\widetilde{deg}(f)$ for the approximate degree of f to be ψ_f, and the dual witnessing a lower bound of $deg_{\pm}(g)$ for the approximate degree of g to be ϕ_g. Then:*

$$(\psi_f\circ\phi_g)(x_1,\ldots,x_n) := 2^n\psi_f(\ldots,sgn(\phi_g(x_i)),\ldots)\prod_{i=1}^{n}|\phi_g(x_i)|$$

witnesses a lower bound of $\widetilde{deg}(f)deg_{\pm}(g)$ for the approximate degree of the function $f(g(x_1),\ldots,g(x_n))$.

Proof. Firstly, in this proof, we will differentiate between an *approximate degree dual* and a *sign degree dual* for a Boolean function. We have already defined an approximate degree dual in the previous chapter. The notion of sign degree can be formulated in a linear program as well, and the sign degree dual for a Boolean function g obtained from it can be expressed in terms of a probability distribution $\mu(x)$ which satisfies the following property:

$$\sum_x \mu(x)g(x)p(x) = 0 \tag{6.1}$$

for all polynomials p of degree lesser than $deg_{\pm}(g)$.

We now construct an approximate degree dual witnessing a (not necessarily tight) lower bound of $deg_{\pm}(g)$ for the approximate degree of g. Note that this dual is not necessarily implied to exist in 5.3, where we only proved the existence of a dual witnessing a tight lower bound of $\widetilde{deg}(g)$. In fact the approximate degree dual can simply be constructed as follows:

$$\phi_g(x) := \mu(x)g(x).$$

We can easily see that the L_1 norm of ϕ_g is equal to 1, as μ is a probability distribution. The pure high degree requirement is satisfied because of equation 6.1, and the correlation is:

$$\sum_x \phi_g(x)g(x) = \sum_x \mu(x)g(x)g(x) = \sum_x \mu(x) = 1$$

Even though the lower bound witnessed by this approximate degree dual is not necessarily tight, it is special in the sense that its sign agrees with g on all inputs. We will exploit this property in the proof.

Recall that in order to prove that $(\psi_f \circ \phi_g)$ is an approximate degree dual witnessing a lower bound of $\widetilde{deg}(f)deg_{\pm}(g)$ for the approximate degree of $f(g(x_1), \ldots, g(x_n))$, we need to show that the construction satisfies three properties: L_1 norm, correlation, and pure high degree.

i. L_1 **norm:** Firstly, observe that $\sum_x \phi_g(x) = 0$, as the we can assume the pure high degree of ϕ_g to be greater than zero (it will only be equal to zero if g is a constant function, in which case the result is trivially true). This observation, along with the fact that the L_1 norm of ϕ_g is 1, gives us:

$$\sum_{x:\phi_g(x)>0} |\phi_g(x)| = \sum_{x:\phi_g(x)<0} |\phi_g(x)| = \frac{1}{2} \tag{6.2}$$

This fact turns out to be extremely useful for us, for it allows us to think of $|\phi_g|$ as two conditional probability distributions. Analyzing the composed dual, we get:

$$\begin{aligned}
||(\psi_f \circ \phi_g)||_1 &= \sum_x 2^n |\psi_f(\ldots, sgn(\phi_g(x_i)), \ldots)| \prod_{i=1}^n |\phi_g(x_i)| \\
&= \sum_{z\in\{-1,1\}^n} |\psi_f(z)| \prod_{i=1}^n \sum_{x:sgn(\phi_g(x))=z_i} 2|\phi_g(x_i)| \\
&= \sum_{z\in\{-1,1\}^n} |\psi_f(z)| = 1
\end{aligned}$$

Since the weight of ϕ_g is equal to half for both $z_i = 1$ and $z_i = -1$ (from equation 6.2), we get a product probability distribution in $\sum_{x:sgn(\phi_g(x))=z_i} 2|\phi_g(x_i)|$.

ii. Correlation: Here, we will use the fact that approximate degree dual ϕ_g agrees in sign with g on all

inputs (we denote $f(g(x_1),\ldots,g(x_n))$ as $(f\circ g)(x_1,\ldots,x_n)$:

$$\begin{aligned}\sum_x (f\circ g)(x_1,\ldots,x_n)(\psi_f\circ\phi_g)(x) &= \sum_x (f\circ g)(x_1,\ldots,x_n)2^n\psi_f(\ldots,sgn(\phi_g(x_i)),\ldots)\prod_{i=1}^n|\phi_g(x_i)|\\ &= \sum_x (f\circ g)(x_1,\ldots,x_n)\psi_f(\ldots,g(x_i),\ldots)\prod_{i=1}^n 2|\phi_g(x_i)|\\ &= \sum_z f(z)\psi_f(z)\sum_{x:sgn(\phi_g(x))=z_i} 2|\phi_g(x_i)|\\ &= \sum_z f(z)\psi_f(z)\end{aligned}$$

Since we already know that the correlation of f and ψ_f is at least $1/3$ (by definition), we get that the correlation of $(f\circ g)$ and $(\psi_f\circ\phi_g)$ is at least $1/3$ as well. In fact, showing a lower bound of ϵ for the correlation, where ϵ is some constant greater than zero is sufficient too. This is because the dual will then witness a ϵ-approximate degree lower bound for the function, and we know already know that the ϵ'-approximate degree, for some constant $\epsilon'>0$, does not differ by more than a constant multiplicative factor from the former quantity.

iii. Pure High Degree: This property is slightly easier to prove than the other two. Upon expanding $(\psi_f\circ\phi_g)$, we get:

$$(\psi_f\circ\phi_g)(x_1,\ldots,x_n) = 2^n\sum_{S\subseteq[n]:|S|\geq\widetilde{deg}(f)}\widehat{\psi_f}(S)\prod_{i\in S}\phi_g(x_i)\prod_{i\notin S}|\phi_g(x_i)|.$$

Each term in the summation contains a product of at least $\widetilde{deg}(f)$ copies of ϕ_g. Because these inner duals are independent from one another, there is no chance of them 'interfering' with one another and reducing the pure high degree. Therefore, the degree of each monomial is indeed $\widetilde{deg}(f)deg_{\pm}(g)$ (recall that the pure high degree of the approximate degree dual ϕ_g was $deg_{\pm}(g)$). □

This method of dual composition has been adapted in various ways, and has been used to show lower bounds for the composed function $AND\circ OR$ in [BT13], and for Surjectivity, k-distinctness, and Image Size Testing in [BKT18]. In [She11], Sherstov showed a tight lower bound for the function $PAR\circ f$, where f is a general Boolean function, by making use of an additional polynomial which did not affect the pure high degree by much, but which zeroed out a subset of the 'error' inputs. Another method of dual composition was shown by Thaler in [Tha16] for $\mathsf{OMB}\circ f$, where OMB is the ODD-MAX-BIT function. This method was shown for variants of the approximate degree, like the one-sided approximate degree, where the polynomial is required to approximate the function on inputs which evaluate to either 1 or -1, and is only required to match the sign of the function for the other subset of inputs.

6.1 Large Error Degree

We define the large error approximate degree of a Boolean function f to be the degree of the polynomial having the least degree, among all polynomials which approximate f to error at most $1-1/n$, on all inputs. We denote this quantity by $\widetilde{deg}_{1-1/n}(f)$. From the definition, it is clear that for every Boolean function f, $\widetilde{deg}(f)\geq\widetilde{deg}_{1-1/n}(f)\geq deg_{\pm}(f)$.

In this section, we prove the following theorem:

Theorem 6.1.1. *Given a constant $0<\epsilon<1$, and assuming $n\leq cm$ for some suitably small constant c depending on ϵ, we can say the following about any two Boolean functions $f:\{-1,1\}^n\to\{-1,1\}$ and $g:\{-1,1\}^m\to\{-1,1\}$:*

$$\widetilde{deg}_{\epsilon}(f\circ g)\geq\Omega(\widetilde{deg}_{\epsilon}(f).\widetilde{deg}_{1-1/m}(g))$$

Sherstov proved a much general version of this theorem in [She11] (Thm. 6.4), where the inner functions could be different from one another. However, the proof was quite complicated, and depended upon a number of intermediate results. We give two proofs for this theorem, one following from a linear operator based argument, and the other involving composition of duals.

Proof. of 6.1.1 (linear operator argument) To simplify things, define $d := \widetilde{deg}_{\epsilon}(f)$, and $d^* := \widetilde{deg}_{1-1/m}(g)$. We give a proof by contradiction. We assume we have a polynomial p of degree much lesser than $d.d^*$, which successfully approximates $f \circ g$. Then we construct a linear operator $\Gamma_{g'}$ such that $\Gamma_{g'}p$ approximates the function f at all points with constant error, and also has degree much lesser than d, which is a contradiction. Therefore, p has to have degree at least $\Omega(d.d^*)$.

To begin with, we know through strong duality, that there exists a dual ϕ_g witnessing the lower bound of $d*$ on the large error degree of g. We also define $g' : \{-1,1\}^m \to \{-1,1\}$ as $g'(x) = sgn(\phi_g(x))$. We will use this to define $\Gamma_{g'}$, which works as follows: it maps functions of the form $h : \{-1,1\}^{n \times m} \to \mathbb{R}$ to $\Gamma_{g'}h : \{-1,1\}^n \to \mathbb{R}$. It is defined as:

$$\Gamma_{g'}h(z) = \mathbb{E}_{\forall i : x_i \sim \nu_{z_i}}[f(g(x_1), \ldots, g(x_n))]$$

Here, $z \in \{-1,1\}^n$, $x \in \{-1,1\}^m$, and ν_{z_i} is a probability distribution on x_i. The absolute value of the dual ϕ_g defines a natural probability distribution for all x_i, and ν_{z_i} is the probability of sampling x_i conditioned on the probability that $g'(x_i) = z_i$. It follows from the properties of the dual that $g'(x_i)$ is uniformly distributed, and hence we get:

$$\nu_{z_i}(x_i) = 2|\phi_g(x_i)|$$

Now, as outlined in the first paragraph, we assume the existence of a polynomial $p : \{-1,1\}^{n \times m} \to \mathbb{R}$ whose degree $deg(p) < o(d.d^*)$, and which successfully approximates $f \circ g$ to error ϵ. Then we have:

$$||p - (f \circ g)||_\infty < \epsilon$$

Through linearity, we also have:

$$||\Gamma_{g'}p - \Gamma_{g'}(f \circ g)||_\infty < \epsilon$$

Now, if we are able to prove the following statement:

$$||\Gamma_{g'}(f \circ g) - f||_\infty < \epsilon \tag{6.3}$$

Then through the triangle inequality, we get:

$$||\Gamma_{g'}p - f||_\infty < 2\epsilon$$

To prove 6.3, we observe the behaviour of $\Gamma_{g'}(f \circ g)$ at some general point $z \in \{-1,1\}^n$. By following the definition of $\Gamma_{g'}$, we have:

$$\begin{aligned}
\Gamma_{g'}(f \circ g)(z) &= \sum_{\forall i : g'(x_i) = z_i} f(g(x_1), \ldots, g(x_n)) \prod_{i=1}^{n} 2|\phi_g(x_i)| \\
&= \sum_{\forall i : g'(x_i) = z_i = g(x_i)} f(g(x_1), \ldots, g(x_n)) \prod_{i=1}^{n} 2|\phi_g(x_i)| + \\
&\quad \sum_{\exists i : g'(x_i) = z_i \neq g(x_i)} f(g(x_1), \ldots, g(x_n)) \prod_{i=1}^{n} 2|\phi_g(x_i)|
\end{aligned}$$

The absolute value of the second addition term in the last equality is upper bounded by $2n/m$, which we get by observing that for each i, $\sum_{x_i : g'(x_i) = z_i = g(x_i)} 2|\phi_g(x_i)|$ is at most $2/m$, and then taking the union bound. From the hypothesis we have that n is suitably smaller than m, and hence we can assume that the whole thing is at most ϵ.

As for the first addition term, it is equal to:

$$f(z_1,\ldots,z_n)\prod_{i=1}^{n}\sum_{x_i:g'(x_i)=z_i=g(x_i)} 2|\phi_g(x_i)| \geq f(z_1,\ldots,z_n)\left(2.\left(\frac{1}{2}-\frac{1}{m}\right)\right)^n$$

Because for each i, we have:

$$\sum_{g(x_i)=g'(x_i)} |\phi_g(x_i)| \geq 1-1/m$$

$$\implies \sum_{g(x_i)=g'(x_i)=-1} |\phi_g(x_i)| + \sum_{g(x_i)=g'(x_i)=1} |\phi_g(x_i)| \geq 1-1/m$$

Since each term in the LHS is upper bounded by $1/2$, we get a lower bound of $\frac{1}{2}-\frac{1}{m}$ for both terms.

Therefore, we now have the lower bound:

$$\begin{aligned} f(z_1,\ldots,z_n)\prod_{i=1}^{n}\sum_{x_i:g'(x_i)=z_i=g(x_i)} 2|\phi_g(x_i)| &\geq f(z_1,\ldots,z_n)\left(1-\frac{2}{m}\right)^n \\ &\geq f(z_1,\ldots,z_n)\left(1-\frac{2n}{m}\right) \\ &= f(z_1,\ldots,z_n)\pm\frac{2n}{m} \end{aligned}$$

Again, since n is suitably lesser than m, this term is ϵ close to f. By combining this with the analysis of the right term, we get that $\Gamma_{g'}(f\circ g)(z)$ is 2ϵ close to f, for general z. Hence we have proved 6.3.

Thus we have shown that $\Gamma_{g'}p$ successfully approximates f. Now in order to arrive at the contradiction, we need to prove that $deg(\Gamma_{g'}p)\leq o(d)$. We show that if $|S|\geq\Omega(d)$ for any $S\subseteq[n]$, then $\widehat{\Gamma_{g'}p}(S)=0$.

$$\begin{aligned} \widehat{\Gamma_{g'}p}(S) &= \frac{1}{2^n}\sum_z \Gamma_{g'}p(z)\chi_S(z) \\ &= \frac{1}{2^n}\sum_z \sum_{(x_1,\ldots,x_n):\forall i:g'(x_i)=z_i} p(x_1,\ldots,x_n)\chi_S(g'(x_1),\ldots,g'(x_n))\prod_{i=1}^{n}2|\phi_g(x_i)| \\ &= \sum_{(x_1,\ldots,x_n)} p(x_1,\ldots,x_n)\prod_{i\in S}\phi_g(x_i)\prod_{i\notin S}|\phi_g(x_i)| \end{aligned}$$

Now, observe that since:

1. ϕ_g has pure high degree d^*,
2. the x_i's are independent of one another,
3. and $|S|\geq\Omega(d)$,

we get that $\prod_{i\in S}\phi_g(x_i)$ has pure high degree $\Omega(d.d^*)$. In fact, because of independence of variables, we can further claim that $\prod_{i\in S}\phi_g(x_i)\prod_{i\notin S}|\phi_g(x_i)|$ has pure high degree $\Omega(d.d^*)$. By our hypothesis, $p(x_1,\ldots,x_n)$ has degree $deg(p)\leq o(d.d^*)$. Therefore, the whole summation term reduces to zero, and we have proved that $\widehat{\Gamma_{g'}p}(S)=0$ for all $S\subseteq[n]$ such that $|S|\geq\Omega(d)$. □

The dual composition based argument makes use of the dual composition technique defined in Thm. 6.0.1:

Proof. of 6.1.1 (dual composition argument) As before, we know about the existence of duals ψ_f and ϕ_g which witness lower bounds of $\widetilde{deg}_\epsilon(f)$ and $\widetilde{deg}_{1-1/m}(g)$, respectively. Notice that the L_1 norm and pure

high degree proof is similar to the one given in 6.0.1. We give the proof for correlation only. Observe that:

$$\sum_{(x_1,\ldots,x_n)} (\psi_f \circ \phi_g)(x_1,\ldots,x_n) f(g(x_i),\ldots,f(x_n))$$

$$= 2^n \sum_{(x_1,\ldots,x_n)} \psi_f(\ldots, sgn(\phi_g(x_i)),\ldots) \prod_{i=1}^{n} |\phi_g(x_i)| f(\ldots,g(x_i),\ldots)$$

$$= \sum_{z\in\{-1,1\}^n} \psi_f(z) \sum_{x:\forall i, sgn(\phi_g(x_i))=z_i} \prod_{i=1}^{n} 2|\phi_g(x_i)| f(\ldots,g(x_i),\ldots)$$

$$= \sum_{z\in\{-1,1\}^n} \psi_f(z) E_{\mu^n}[f(\ldots,g(x_i),\ldots)|(\ldots,sgn(\phi_g(x_i)),\ldots)=z]$$

where μ^n is the product probability distribution on $(\ldots,x_i,\ldots)$, defined as $\mu^n(\ldots,x_i,\ldots) = \prod_{i=1}^{n} |\phi_g(x_i)|$. Now, for any fixed z, notice that the following two random variables are randomly distributed:

- The string $(\ldots,g(x_i),\ldots)$ when $(\ldots,x_i,\ldots)$ is sampled from μ^n, conditioned on the fact that $(\ldots,sgn(\phi_g(x_i)),\ldots) = z$
- The string $(\ldots,y_iz_i,\ldots)$, where $y_i = -1$ with probability $2\sum_{x_i\in E_{z_i}} |\psi(x_i)|$, where $E_{z_i} = \{x_i : sgn(\phi_g(x_i)) = z_i \neq g(x_i)\}$

This is because:

$$Pr_{x_i\sim\mu}[g(x_i) \neq sgn(\phi_g(x_i))|sgn(\phi_g(x_i)) = z_i] = 2\sum_{x\in E_{z_i}} |\phi_g(x)|$$

Therefore, we can write the last expression as:

$$\sum_{z\in\{-1,1\}^n} \psi_f(z) E_y[f(\ldots,y_iz_i,\ldots)]$$

where $y := (\ldots,y_i,\ldots)$, and for each i, $y_i = -1$ with probability $2\sum_{x_i\in E_{z_i}} |\phi_g(x_i)|$. We now show that the quantity $E_y[f(\ldots,y_iz_i,\ldots)]$ has value which is quite close to $f(\ldots,z_i,\ldots)$, no matter what z is. Hence, the last expression which we rewrote above is quite close to the actual correlation value between ψ_f and f.

To state it precisely, we break $E_y[f(\ldots,y_iz_i,\ldots)]$ into two terms:

$$\begin{aligned} E_y[f(\ldots,y_iz_i,\ldots)] &\geq Pr_y[\forall i : y_i = 1] f(\ldots,y_iz_i,\ldots) - Pr_y[\exists i : y_i = -1] \\ &\geq \left(1-\frac{1}{m}\right)^n f(\ldots,z_i,\ldots) - \sum_i 2 \sum_{x\in E_{z_i}} |\phi_g(x)| \\ &\geq \left(1-\frac{n}{m}\right) f(\ldots,z_i,\ldots) - n.\frac{2}{m} \end{aligned}$$

In the second to last inequality, we get the last term by taking a union bound. Since we had taken n to be sufficiently smaller than m, we can assume the whole that the lower bound is equal to:

$$f(\ldots,z_i,\ldots) - \delta$$

The precise value of δ will be defined below:

$$\begin{aligned} \sum_{z\in\{-1,1\}^n} \psi_f(z) E_y[f(\ldots,y_iz_i,\ldots)] &\geq \sum_{z\in\{-1,1\}^n} \psi_f(z)[f(z)-\delta] \\ &= \left[\sum_{z\in\{-1,1\}^n} \psi_f(z) f(z)\right] - \delta \end{aligned}$$

We want the last quantity to be some constant greater than zero, therefore we choose m and n such that δ turns out to be sufficiently smaller than the correlation of ψ_f and f. □

Chapter 7

Evidence for Tight Relation between Approximate Degree and Degree

In [NS92], Nisan and Szegedy showed that for all Boolean functions, $deg(f) \leq O(\widetilde{deg}(f)^8)$. Since then, through subsequent works such as [Aar03], [KT16], and more recently, [Hua19], the bound was tightened up to $\widetilde{O}(\widetilde{deg}(f)^4)$. Regarding separations, the best separation we have is for the OR function, where $deg(\mathsf{OR}) = n$, and $\widetilde{deg}(\mathsf{OR}) = \sqrt{n}$. We believe that this separation is the best possible one. In other words, for all Boolean functions f, we believe that $deg(f) \leq \widetilde{O}(\widetilde{deg}(f)^2)$. In this chapter, we provide evidence which supports this case.

In [LNS20], Laplante, Naserasr, and Sunny proved the following (a proof sketch for this theorem at the end of this chapter):

Theorem 7.0.1. *[LNS20] If the degree of a Boolean function is n, then its sensitivity graph has a tree having at least $n+1$ vertices in it, and of depth at most 2.*

The *sensitivity graph* of a function is the subgraph induced by the function on the Boolean hypercube, which is as follows: if a vertex x is a neighbour of y in the Boolean hypercube, and $f(x) \neq f(y)$, then (and only then) the edge (x, y) is a part of the sensitivity graph. A couple of observations: the sensitivity graph of a Boolean function is always bipartite, and the vertex having the greatest degree is the vertex having the greatest sensitivity in f.

We conjecture that every function whose sensitivity graph contains a tree as described above has approximate degree at least $\sqrt{n}$. Then it is easy to see that this implies $deg(f) \leq \widetilde{deg}(f)^2$ for all f, because in the case that the degree of the function is not equal to n, we can look at the representing polynomial of the function, and zero out all variables which do not appear in its largest monomial. This then gives us a new function whose degree is equal to its number of inputs, and one whose sensitivity at each vertex is at most that of the original function. We then work with this new function, and because each of the remaining vertices' degree in the new sensitivity graph is at most its old degree, there is no danger of having the tree described above in the new graph, when it was not present in the old one.

As evidence for the conjecture stated above, we prove it for the sub case where the tree has depth at most 1. That is, the case where the sensitivity graph contains a vertex, as well as each of its n neighbours.

Theorem 7.0.2. *If the sensitivity graph of a Boolean function f contains a tree of depth at most 1, then $\widetilde{deg}(f) \leq \Omega(\sqrt{n})$.*

Proof. We will prove this via the dual method, by showing an analysis for the multivariate dual for AND exhibited in [BMTW19], which shows a lower bound of $\Omega(\sqrt{n})$. We shall first prove it for the case when the tree has its root at the zero Hamming weight input (that is, the tree includes the zero Hamming weight input, as well as all n neighbours, each of which has a Hamming weight equal to one.) The dual shown in [BMTW19] is of the following form:

$$\phi(x) = \frac{1}{Z}\chi_{[n]}(x)\left(\mathbb{E}_{S\sim H}[\chi_S(x)]\right)^2$$

Where H is defined as the uniform distribution over the subsets $\{S \subseteq [n] : |S| \leq (n-d)/2\}$, and Z is a normalization constant equal to:

$$Z := \sum_x \mathbb{E}_{S\sim H}[\chi_S(x)]^2$$

We take d to be equal to $\alpha\sqrt{n}$, where α is some constant to be defined later.

The pure high degree of this dual is at least $\alpha\sqrt{n}$. This is because the polynomial $\mathbb{E}_{S\sim H}[\chi_S(x)]^2$ has degree at most $n - \alpha\sqrt{n}$, and when this polynomial is multiplied by the full parity outside it, this will result in every monomial having degree at least $n - (n - \alpha\sqrt{n})$, which is equal to $\alpha\sqrt{n}$. The L_1 norm is equal to 1 because:

$$\sum_x \left| \frac{1}{Z}\chi_{[n]}(x) \left(\mathbb{E}_{S\sim H}[\chi_S(x)]\right)^2 \right| = \sum_x \frac{1}{Z} \left(\mathbb{E}_{S\sim H}[\chi_S(x)]\right)^2 = 1$$

For the correlation, the idea is to show that the inputs with Hamming weight 0 and 1 collectively possess more than half of the total L_1 weight, and couple it with the observation that the sign of the dual matches the value of the function at these inputs as well. We begin by noting the following observation (recall that 1^n denotes the input with Hamming weight zero):

$$\phi(1^n) = |\phi(1^n)| = \frac{1}{Z}$$

Additionally, we obtain an upper bound on $|H|$ which we will use later. The first inequality below is from [Gal11]:

$$\begin{aligned} |H| = \sum_{i=0}^{(n-d)/2} \binom{n}{i} &< 2^{n-1} \frac{\binom{n}{(n-d)/2+1}}{\binom{n}{n/2}} \\ &< 2^{n-1} \frac{1.1 \times \binom{n}{\frac{n-d}{2}}}{\binom{n}{n/2}} \end{aligned} \tag{7.1}$$

Now we need to obtain a lower bound for $|\phi(z)|$ when z is an input with Hamming weight 1. Consider an input $z \in \{-1, 1\}^n$ having the value -1 at index i, and 1 everywhere else. For every subset $S \subseteq [n] - \{i\}$ of size at most $(n-d)/2 - 1$, we have $\chi_S(z) = 1$. Moreover, $\chi_{S\cup\{i\}}(z) = -1$. This implies that inside the expectation term, partial parities of the form $\chi_S(z)$ and $\chi_{S\cup\{i\}}(z)$ will cancel each other out, leaving out only partial parities corresponding to subsets of $[n] - i$ having size exactly $(n-d)/2$. These partial parities will all have value 1, since the subsets are taken from $[n] - i$, and from the definition of z, we only have ones at these places. Therefore:

$$\begin{aligned} |\phi(z)| &= \frac{1}{Z}\left(\frac{1}{|H|}\binom{n-1}{\frac{n-d}{2}}\right)^2 \\ &= \frac{1}{Z}\left(\frac{1}{|H|}\frac{n+d}{2n}\binom{n}{\frac{n-d}{2}}\right)^2 \\ &\geq \frac{1}{4Z}\left(\frac{1}{|H|}\binom{n}{\frac{n-d}{2}}\right)^2 \\ &\geq \frac{1}{4Z}\left(\frac{\binom{n}{n/2}}{1.1 \times 2^{n-1}}\right)^2 \end{aligned} \tag{7.2}$$

Where the last inequality is from 7.1. We now use the following fact (valid for sufficiently large n):

$$\binom{n}{n/2} \geq \frac{2^{n-1}}{\sqrt{n}}$$

Plugging this into last inequality of 7.2 we get:

$$|\phi(z)| \geq \frac{1}{4Z}\frac{1}{1.1^2 n} \geq \frac{|\phi(1^n)|}{4n}$$

Since we did not assume anything about the position i, the lower bound is valid for any input having Hamming weight 1. By combining this result with the value of $|\phi(1^n)|$, we get:

$$|\phi(1^n)| + n|\phi(z)| \geq \frac{5}{4}|\phi(1^n)| = \frac{5}{4}\frac{|H|}{2^n}$$

Recall that we had taken d to be equal to $\alpha\sqrt{n}$. By choosing an appropriate value for α, we can have $|H|$ arbitrarily close to $2^n/2$. Therefore, we choose an alpha such that $|H| = 0.9 \times 2^{n-1}$. By plugging this value into the above inequality, we get a lower bound which is $1/2+\epsilon$ for some constant ϵ. As the values of the dual for inputs of Hamming weight zero and agree in sign with with f, the positive correlation is at least $1/2+\epsilon$, and since the L_1 norm is 1, the negative correlation is at most $1/2-\epsilon$. Therefore, the total correlation turns out to be at least:

$$\sum_x \phi(x)f(x) \geq \frac{1}{2} + \epsilon - (1/2 - \epsilon) = 2\epsilon$$

Now, notice that for the case when the root vertex is not the zero Hamming weight input, and is some other input $z \in \{-1, 1\}^n$, we can simply replace each input bit x_i in the dual with the bit $x_i z_i$. The proof given above generalizes easily. □

7.1 Approaches for the general case

The most difficult case which seems to resist all dual based approaches seems to be the one where the root vertex has $\sqrt{n}$ neighbouring children, and each of those children has $(n-\sqrt{n})/\sqrt{n}$ children. The last quantity is almost $\sqrt{n}$ for sufficiently large n, so intuitively, this case is one where the tree is not 'concentrated' at one node, but spread out more or less equally among all of them. Using current dual based methods, we only get a lower bound of $n^{1/4}$ for this case. Any approach which attempts to tackle the might need to make use of additional properties of the sensitivity graph when the function being considered is of degree n.

7.2 Short proof of Thm. 7.0.1

Proof. The theorem is basically a corollary of two facts. The first fact is that a special, signed adjacency matrix of the Boolean hypercube has an eigenspace of dimension 2^{n-1}, and if the eigenvectors corresponding to an induced subgraph's vertices form a linear dependence in this eigenspace, then the subgraph has a tree as described in the theorem statement. The second fact is that a function having degree n has a sensitivity graph that contains at least $2^{n-1}+1$ vertices, and hence their corresponding eigenvectors automatically have a linear dependence in the eigenspace mentioned above. □

Chapter 8

Approximate degree of derivatives of Boolean functions

Because characterizing the approximate degree is hard for some Boolean functions, it might be worthwhile to look at related notions of approximate degree for the derivatives of Boolean functions. If it turns out that these notions for the derivatives provide good bounds on the approximate degree of the original functions, then it might. In this chapter, we define approximate degree for both symmetric and non-symmetric functions, and show that for the case of symmetric functions, the approximate degree of the derivative is equal to the approximate degree of the original Boolean function. For non-symmetric functions, we show that we are able to construct a approximating polynomial for f, if we are given the approximating polynomials for the n directional derivatives of f, with an increase of a log factor, thus proving that $\widetilde{deg}(f) \leq \widetilde{O}(\max_i f'_i)$, where f'_i denotes the ith directional derivative. Note that $\widetilde{O}$ is similar to the big-O notation, except it ignores all polylog factors.

8.1 Characterization for symmetric functions

Definition 8.1.1. *Let $F : [n] \to \{0, 1\}$ be the associated univariate function to a symmetric Boolean function f (recall that we had defined the associated univariate function in section 2.3). We define $F' : [n-1] \to \{-1, 0, 1\}$, as:*

$$F'(i) = \frac{F(i+1) - F(i)}{2}.$$

We then define the associated multivariate function F', which we denote by f', to be the discrete derivative of the symmetric Boolean function f.

Next, we prove the following theorem:

Theorem 8.1.2. *For all symmetric Boolean functions f, $\widetilde{deg}(f') = \Theta(\widetilde{deg}(f))$.*

We present the proof as part of two lemmas, which together imply the above theorem.

Lemma 8.1.3. *For any symmetric Boolean function f, and for any $0 < \epsilon < 1/2$, $deg_\epsilon(f') \leq (deg_{\epsilon/2}(f))$.*

Proof. For simplicity, we show the proof when $\epsilon = 1/3$. The proof can be easily generalized for other ϵ. We construct:

$$p(i) = \frac{q(i+1) - q(i)}{2}$$

where $q(i)$ is the symmetric version of the approximating polynomial for f, having degree at most $deg_{1/6}(f)$. The polynomial q can be obtained via symmetrization. Notice that p's degree is equal to that of q, and moreover, it satisfies $|p(i) - f'(i)| \leq 1/3$ for all i. The last part can be easily seen by taking all 4 possible values of the tuple $(f(i+1), f(i))$, and taking the allowed values of $(q(i+1), q(i))$ for that tuple which maximize $|q(i+1) - q(i)|$. The associated multivariate counterpart of p is the desired approximating polynomial for f'. □

Next, we show the lemma which proves the lower bound:

Lemma 8.1.4. $deg_\epsilon(f') = \Omega(deg_\epsilon(f))$

Proof. In order to prove $deg_\epsilon(f') = \Omega(deg_\epsilon(f))$, we essentially generalize Paturi's proof in [Pat92], which lower bounds the approximate degree for symmetric functions. His proof exploits the fact that polynomials having high derivative near the origin have high degree. Note that this fact is applicable to approximating polynomials for f as well as f', since the 'jump' closest to the origin (which causes the derivative to be high) is at the same location for both functions. We shall make this statement more concrete below.

In our proof, we will be making use of the Markov Bernstein inequality here, which we state below:

Fact 8.1.5. ***Markov-Bernstein inequality:*** *Let $p : \mathbb{R} \to \mathbb{R}$ be a degree-d polynomial with real coefficients. Then, taking $||p|| = sup\{|p(x)| : x \in [-1, 1]\}$, we have for $x \in [-1, 1]$:*

$$\left| \frac{d\sqrt{1-x^2}+1}{d^2} p'(x) \right| \leq 2||p||$$

For this part of the proof, we redefine the derivative as $f' : [n-1] \to \{0, 1/2, 1\}$ (mapping from new definition to old is $0 \to -1, 1/2 \to 0, 1 \to 1$), and prove lower bounds on the degree of the set of all polynomials approximating this f' in the new definition. The degree lower bounds holds for the original definition as well, since the polynomials are identical, upto shifting and scaling.

Additionally, we also modify the domain of the function. Specifically, we work with a scaled version of the approximating polynomial so that it takes inputs in the interval $[-1, +1]$, which allows us to utilize the Markov-Bernstein inequalities. Again, there is no change in the degree. The approximating polynomial p whose degree we denote by d, therefore, has the following property:

$$\left| p\left(\frac{2i}{n} - 1\right) - f'(i) \right| \leq \epsilon \quad \forall 0 \leq i \leq n$$

We can see from the above condition that after the scaling, 0 corresponds to -1, $n/2$ corresponds to 0, and n corresponds to 1. The distance between two points where the condition applied was 1 previously, and is now $2(i+1)/n - 2i/n = 2/n$.

Without loss of generality, we can assume that n is even, this is because if n was odd, we can consider a new function $g' : [n+1] \to \{0, 1/2, 1\}$ which is identical to f' on inputs $i \in [n]$, and has $g'(n+1)$ could be any arbitrary value. Then consider a polynomial which approximates g' only on inputs $i \in [n]$.

Note that $p(0)$ can lie within one of the following three intervals: $[-\epsilon, \epsilon]$, $[\frac{1}{2} - \epsilon, \frac{1}{2} + \epsilon]$, $[1 - \epsilon, 1 + \epsilon]$. We are interested in the point at which p crosses over to a different interval, while going away from the origin. For example, suppose $p(0) \in [\frac{1}{2} - \epsilon, \frac{1}{2} + \epsilon]$. Additionally, assume we have a point z such that $p(z) = 1 - \epsilon$, and for every point in $x \in [-z, z)$, $\epsilon < p(x) < 1 - \epsilon$. Then z is the point we are interested in.

More formally, we define $z := inf\{|x| : x \in [-1, 1]$ and $p(x) \leq 1 - \epsilon$ or $p(x) \geq \epsilon\}$. Note that the above definition will change accordingly if $p(0)$ lies in a different interval. For simplicity, for the rest of this proof, we will assume the values we took in the example, i.e, $p(0) \in [\frac{1}{2} - \epsilon, \frac{1}{2} + \epsilon]$, and $p(z) = 1 - \epsilon$, and also assume w.l.o.g that z is positive (because if z was negative, we can just work with $p(-x)$). The proof is similar for the other cases.

Now, let k be such that $2k/n < z \leq 2(k+1)/n$. Also, recall the definition of t from above. By its definition, we note the following observations:

1. $2k \leq t$.
2. $p(2k/n) \leq \frac{1}{2} + \epsilon$ and $p(z) \geq 1 - \epsilon$.

We now divide the problem into two cases, depending on whether the first 'jump' from the centre occurs close to it, or farther away. Specifically, choose a constant $c = 1/100$. Then:

Case 1: $\frac{2k}{n} - 1 \leq 1 - c$ This is the case where $|k - n|$ is bound by a constant. As z is at most 1, for any $x \in [-1, 1]$, we have the following fact:

$$|p(x + z)| \leq 2^{d+1} e^{\sqrt{3}d}$$

We won't prove this fact here, but instead state that it comes from upper bounds derived on Chebyshev polynomials, which themselves upper bound $|p(x+z)|$.

We now obtain q by $q(x) := p(x+z)(1-x^2)^{c_1 d}$, where c_1 is a constant depending on c. q contains $p(x+z)$ within it because we wanted to shift the 'jump' location closer to the origin, and we multiplied it with $(1-x^2)^{c_1 d}$ in order to decrease the magnitude as we go farther away from the origin. We show that q has high derivative near the origin. We can easily see that $|q(0)| = |p(z)| \geq 1-\epsilon$, and $|q(2k/n - z)| < |p(2k/n)| \leq 1/2 + \epsilon$. We already knew that $|2k/n - z| < 2/n$, from the definition of k, and hence we have, for some $x \in [2k/n, 0]$:

$$q'(x) \geq n\left(\frac{1}{4} - \epsilon\right)$$

Note that for any $x \in [-1, c] \cup [c, 1]$, we have:

$$|q(x)| = |p(x+z)||(1-x^2)^{c_1 d}| \leq |p(x+z)|e^{-c_1 d} \leq 1$$

Therefore, $|q|$ is low on all points which are at a distance of c or more from the origin. If it also turns out that $|q|$ is low within $[-c, c]$, then we can use the lower bound on $q'(x)$ we obtained above, and plugging both $||q||$ and q' into the Markov Bernstein inequality, we get a lower bound of $\Omega(n)$ on the degree of q.

Now consider the case that $|q|$ is high at some point which lies in $[-c, c]$ (i.e, $|q| > 2(1+\epsilon)$. If it is lesser, the above case holds). But since $|q(2i/n - z)| < p(2i/n) \leq 1 + \epsilon$, for all integers $0 \leq i \leq n$, if it rises to a high value, then it must come down very quickly quickly as well. Therefore, using the mean value theorem:

$$q'(x) \geq \frac{||q|| - (1+\epsilon)}{\frac{2}{n} \times \frac{1}{2}} \geq \frac{n||q||}{2}$$

As before, by plugging this inequality into the Markov Bernstein inequality, we get a lower bound of $\Omega(n)$ on the degree of q. By the definition of q, this translates to a lower bound on the degree of p as well.

Case 2: $\frac{2k}{n} - 1 > 1 - c$ This proof for this case proceeds in a similar manner, except we multiply p by a trigonometric polynomial, and achieve a lower bound of:

$$\Omega(\sqrt{n^2 - (2(k+1))^2}) \geq \Omega(\sqrt{n^2 - t^2}) \geq \Omega(\sqrt{n(n-t)})$$

□

8.2 Non symmetric functions

Firstly, we define the ith directional derivative as follows:

Definition 8.2.1. *For a Boolean function f, we define the ith directional derivative as:*

$$D_i f(x_1, \ldots, x_i, \ldots, x_n) = \frac{f(x_1, \ldots, 1, \ldots, x_n) - f(x_1, \ldots, -1, \ldots, x_n)}{2}.$$

Observe that $D_i f$'s range is the set $\{-1, 0, 1\}$.

We also note the following simple observation:

Observation 8.2.2. *Any Boolean function f can be written as:*

$$f(x_1, \ldots, x_n) = f(1, \ldots, 1) + (x_1 - 1)D_1 f(x_1, 1, \ldots, 1) + (x_2 - 1)D_2 f(x_1, x_2, 1, \ldots, 1) + \cdots + (x_n - 1)D_n f(x_1, \ldots, x_n)$$

Proof. The term $(x_i - 1)$ disappears when x_i is equal to one, and equals -2 when $x_i = -1$. By combining this observation with the following fact:

$$D_i f(x_1, .., x_i, 1, .., 1) = \frac{f(x_1, ..x_{i-1}, 1, .., 1) - f(x_1, ..x_{i-1}, -1, ..1)}{2}$$

, we can rewrite the RHS as:

$$f(1, .., 1) + \sum_{i: x_i = -1} -f(x_1, ..x_{i-1}, 1, .., 1) + f(x_1, ..x_{i-1}, -1, .., 1)$$

Now, fix a particular input x, and consider two indices j and k such that $j < k$, $x_j = x_k = -1$, and there's no -1 between the positions j and k. Now, by looking at the second part of the term corresponding to j, and the first part of the term corresponding to k, we get:

$$f(x_1, ..x_{j-1}, \underset{j}{\underline{-1}}, .., \underset{k}{\underline{1}}, .., 1) - f(x_1, x_{j-1}, \underset{j}{\underline{x_j}}, .., \underset{k}{\underline{1}}, .., 1)$$

This term equals zero, because $x_j = -1$ holds (by assumption), and all positions between j and k contain ones. The whole summation then becomes a telescopic sum, with the first term equalling $-f(1, .., 1)$, and the last equalling $f(x_1, x_2, .., x_n)$, therefore the whole term is equal to the LHS. □

From the lemma, one can see that if we have ϵ-approximating polynomials for every $D_i f$, where ϵ is a constant, then we can convert them into $1/n$-approximating polynomials, with at most a lgn factor increase in degree. Then, the following polynomial will be an ϵ-approximating polynomial for f (ϵ is constant here as well):

$$f(1, \ldots, 1) + (x_1 - 1)p_1 f(x_1, 1, \ldots, 1) + (x_2 - 1)p_2 f(x_1, x_2, 1, \ldots, 1) + \cdots + (x_n - 1)p_n f(x_1, \ldots, x_n)$$

where $p_i f$ is the $1/n$-approximating polynomial for $D_i f$. Since $f(1, \ldots, 1)$ is a constant, the degree of this polynomial will be at most $\widetilde{O}(\max_i f'_i)$. Therefore, we get $\widetilde{deg}(f) \leq \widetilde{O}(\max_i f'_i)$, for all f.

Bibliography

[Aar03] Scott Aaronson. Quantum certificate complexity. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 171–178. IEEE Computer Society, 2003.

[ABDKT20] Scott Aaronson, Shalev Ben-David, Robin Kothari, and Avishay Tal. Quantum implications of huang's sensitivity theorem. *Electronic Colloquium on Computational Complexity (ECCC)*, 27:66, 2020.

[ABP19] Srinivasan Arunachalam, Jop Briët, and Carlos Palazuelos. Quantum query algorithms are completely bounded forms. *SIAM Journal on Computing*, 48(3):903–925, Jan 2019.

[Amb03] Andris Ambainis. Polynomial degree vs. quantum query complexity. In *Annual Symposium on Foundations of Computer Science Proceedings*, pages 230–239, 2003.

[AS04] Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM (JACM)*, 51(4):595–605, 2004.

[BBC+01] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald De Wolf. Quantum lower bounds by polynomials. *Journal of the ACM (JACM)*, 48(4):778–797, 2001.

[Bei93] Richard Beigel. The polynomial method in circuit complexity. *Proceedings of the Eighth Annual Structure in Complexity Theory Conference*, pages 82–95, 1993.

[BKT18] Mark Bun, Robin Kothari, and Justin Thaler. The polynomial method strikes back: tight quantum query bounds via dual polynomials. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 297–310. ACM, 2018.

[BMTW19] Andrej Bogdanov, Nikhil S. Mande, Justin Thaler, and Christopher Williamson. Approximate degree, secret sharing, and concentration phenomena. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, volume 145 of *LIPIcs*, pages 71:1–71:21, 2019.

[BNRdW05] Harry Buhrman, Ilan Newman, Hein Röhrig, and Ronald de Wolf. Robust polynomials and quantum algorithms. In *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings*, volume 3404 of *Lecture Notes in Computer Science*, pages 593–604, 2005.

[BT13] Mark Bun and Justin Thaler. Dual lower bounds for approximate degree and markov-bernstein inequalities. In *40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965, pages 303–314, 2013.

[BV97] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM J. Comput.*, 26(5):1411–1473, October 1997.

[CA08] Arkadev Chattopadhyay and Anil Ada. Multiparty communication complexity of disjointness. *Electronic Colloquium on Computational Complexity (ECCC)*, 02 2008.

[Gal11] Jean Gallier. *Discrete Mathematics*. Springer-Verlag New York, 2011.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, 1996.

[Hua19] Hao Huang. Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *Annals of Mathematics*, 190(3):949, 2019.

[KSdW07] Hartmut Klauck, Robert Spalek, and Ronald de Wolf. Quantum and classical strong direct product theorems and optimal time-space tradeoffs. *SIAM Journal on Computing*, 36(5):1472–1493, 2007.

[KT16] Raghav Kulkarni and Avishay Tal. On fractional block sensitivity. *Chic. J. Theor. Comput. Sci.*, 2016, 2016.

[Lee09] Troy Lee. A note on the sign degree of formulas. *CoRR*, abs/0909.4607, 2009.

[LNS20] Sophie Laplante, Reza Naserasr, and Anupa Sunny. Sensitivity lower bounds from linear dependencies. *Electron. Colloquium Comput. Complex.*, 27:2, 2020.

[MP69] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 1969.

[NS92] Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 462–467. ACM, 1992.

[Pat92] Ramamohan Paturi. On the degree of polynomials that approximate symmetric boolean functions (preliminary version). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 468–474. Association for Computing Machinery, 1992.

[She07] Alexander A. Sherstov. Separating ac 0 from depth-2 majority circuits. In *In Proc. of the 39th Symposium on Theory of Computing (STOC*, pages 294–301, 2007.

[She08] Alexander A. Sherstov. The pattern matrix method for lower bounds on quantum communication. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 85–94. Association for Computing Machinery, 2008.

[She09] Alexander A. Sherstov. The intersection of two halfspaces has high threshold degree. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 343–362. IEEE Computer Society, 2009.

[She11] Alexander A. Sherstov. Strong direct product theorems for quantum communication and query complexity. *Proceedings of the 43rd annual ACM symposium on Theory of computing - STOC '11*, 2011.

[She12] Alexander A. Sherstov. Making polynomials robust to noise. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 747–758, 2012.

[She16] Alexander A. Sherstov. The multiparty communication complexity of set disjointness. *SIAM Journal on Computing*, 45(4):1450–1489, 2016.

[She18] Alexander A. Sherstov. Algorithmic polynomials. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 311–324, 2018.

[Sho94] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, page 124–134. IEEE Computer Society, 1994.

[Sim94] D. R. Simon. On the power of quantum computation. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, page 116–123. IEEE Computer Society, 1994.

[Spa08] R. Spalek. A dual polynomial for or. *ArXiv*, abs/0803.4516, 2008.

[Tal14] Avishay Tal. Shrinkage of de morgan formulae by spectral techniques. *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 551–560, 12 2014.

[Tha16] Justin Thaler. Lower bounds for the approximate degree of block-composed functions. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 17:1–17:15, 2016.