

Query Complexity of Recursively Composed Functions

by

Bandar Al-Dhalaan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2024

© Bandar Al-Dhalaan 2024

Author's Declaration

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

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

Abstract

In this work, we explore two well-studied notions of randomized query complexity; bounded-error randomized ($R(f)$), and zero-error randomized ($R_0(f)$). These have their natural analogues from the classical model of computation, R corresponding to BPP or “Monte Carlo” algorithms and R_0 to ZPP or “Las Vegas” algorithms.

For a query complexity measure M , one can define the composition limit of M on f by $M^*(f) = \lim_{k \rightarrow \infty} \sqrt[k]{M(f^k)}$. The composition limit is a useful way to understand the asymptotic complexity of a function with respect to a specific measure (e.g. if $M(f) = O(1)M(g)$, then $M^*(f) = M^*(g)$).

We show that under the composition limit, Las Vegas algorithms can be reduced to Monte Carlo algorithms in the query complexity world. Specifically, $R_0^*(f) = \max(C^*(f), R^*(f))$ for all possibly-partial boolean functions f . This has wide-reaching implications for the classical query complexity of boolean functions that are still open. For example, this result implies that any bounded-error algorithm for recursive 3-majority can be converted into a zero-error algorithm with no additional cost (i.e. $R^*(3\text{-MAJ}) = R_0^*(3\text{-MAJ})$).

Furthermore, we explore one possible generalization of the recursive 3-majority problem itself, by analyzing 3-majority as a special case of a combinatorial game we call Denial Nim.

Acknowledgements

I am in great debt to my advisor Shalev Ben-David for his mentorship, his support, and most of all for putting up with me through all my shortcomings.

Dedication

To my parents, for their incredible patience and support.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	viii
1 Introduction	1
1.1 Query Complexity	1
1.2 Randomization	2
1.2.1 Las Vegas and Monte Carlo	3
1.3 Composed Functions	4
1.4 Quantum Query Complexity	4
1.5 Our Results	5
1.5.1 Methods	5
1.6 Organization	7

2	Preliminaries	8
2.1	Notation	8
2.2	Boolean Functions	8
2.3	Deterministic Query Complexity	9
2.4	Randomized Query Complexity	10
2.5	Certificates	11
2.6	Measures and Composition	12
3	Monte Carlo to Las Vegas	14
3.1	Proof of Main Theorem	15
4	3-Majority and Denial Nim	21
4.1	3-MAJ	21
4.1.1	Random Methods	22
4.2	Denial Nim	22
5	Conclusion	27
	References	29

List of Figures

4.1	A height 2 3-majority tree. Green represents a value of 1 and Red a value of zero	23
4.2	An instance of 3-Denial Nim	24
4.3	Game tree for Figure 4.2	24

Chapter 1

Introduction

1.1 Query Complexity

Query complexity is a model of a computation whose object of study is the boolean function. Given an n -ary boolean function f that outputs either 1 or 0, and given an input x in the domain of f , the question is: how many *queries* to the input x are required to determine $f(x)$? In this setting, the bits of the input x are hidden from the algorithm, which must expend a query for each bit it wishes to reveal. A priori, not all the bits of the input are necessary for computing $f(x)$. For example, if f is the OR function, and one of the input bits is 1, we may terminate upon seeing a 1 (in the worst case, however, we must uncover each bit to verify that they are all zeros; this is not always the case, though.). The goal is to compute $f(x)$ while minimizing the number of queries to x . This cost is the “query complexity”.

The query model of computation is very simple, yet can be very effective in studying analogues of classical algorithms, many of which have natural query-complexity versions. Query complexity has been used for example to demonstrate quantum advantage of algorithms like Grover and Shor [4].

In this work, we analyze the relationship between Monte Carlo algorithms (BPP) and Las Vegas algorithms (ZPP) in a query complexity context. Our main result is a conditional equivalence of these two classes in the query complexity world, when the composition limit is taken and when the certificate complexity of the subject function is not too large.

1.2 Randomization

In the world of algorithms, randomization is an often favorable trade-off between determinism and the cost of the algorithm. Significant improvements can often be obtained if small error is allowed. For this reason, randomization in algorithms is ubiquitous, as it is in query complexity.

Consider for instance the partial boolean GAPMAJ_n function, which returns 1 if the Hamming weight of the input is at least $2n/3$, and 0 if it is at most $n/3$. This is an example of a promise problem, where the input is guaranteed to have a certain characteristic. Let's analyze this function in both the deterministic and randomized settings:

Deterministic Complexity: In the deterministic case, $D(\text{GAPMAJ}_n) = 2n/3 + 1$. To see why:

- In the worst case, we might query $n/3$ ones and $n/3$ zeroes before the last bit confirms which value is in the majority.
- Consider an adversarial strategy where the first $2n/3$ bits are evenly split between 0s and 1s.
- After querying these $2n/3$ bits, we still don't know if the input has $n/3$ or $2n/3$ ones.
- We must query one more bit to definitively determine the output.
- Thus, in the worst case, we need to query $2n/3 + 1$ bits.

Randomized Complexity: Surprisingly, $R(\text{GAPMAJ}_n) = 1$. Here's why:

- Consider the algorithm that randomly selects a single bit to query.
- If the input has Hamming weight $\geq 2n/3$, the probability of selecting a 1 is at least $2/3$.
- If the input has Hamming weight $\leq n/3$, the probability of selecting a 0 is at least $2/3$.
- Thus, by querying a single random bit, we can determine the correct answer with probability at least $2/3$.

- This meets the definition of bounded-error randomized query complexity.

This example demonstrates the potential power of randomization in query complexity. With just a single query, a randomized algorithm can solve a problem that requires $\Omega(n)$ queries for any deterministic algorithm.

1.2.1 Las Vegas and Monte Carlo

In classical computing, we distinguish between two main classes of randomized algorithms: zero-error randomized complexity (ZPP) and bounded-error randomized complexity (BPP). Zero-error algorithms (aka Las Vegas) always output the correct answer, though they may fail to make a decision, necessitating a re-run. Their complexity is measured in the average case. Bounded-error algorithms (aka Monte Carlo), however, output the correct answer with constant probability ($\geq 2/3$)¹ in each run. The analogues of these classes in the query complexity model are $R(f)$ for the bounded-error randomized complexity of a boolean function f , and $R_0(f)$ for the zero-error randomized complexity of f .

The relationship between BPP and ZPP has been a subject of considerable interest in theoretical computer science, primarily due to the implications of derandomization and the potential collapse of these classes under certain assumptions. BPP, being a more general class, contains ZPP as a subset, where algorithms are allowed to have a small error probability. However, the main question revolves around whether every problem in BPP can effectively be reduced to ZPP, thereby eliminating the need for error by using randomness more efficiently. Research, such as the work by Impagliazzo and Wigderson, suggests that if certain cryptographic assumptions hold, notably those regarding the hardness of specific problems, then efficient pseudorandom generators can be constructed, which might imply that BPP could indeed be equal to ZPP [17]. This result would signify not only a profound theoretical impact in complexity theory by showing the power of randomness in algorithm design but also practical implications in designing algorithms with optimal error rates and computational efficiency.

The main focus of our work is to explore the relationship between the standard query complexity analogues of BPP and ZPP. Informally, we find that ZPP query algorithms (Las-Vegas) can be reduced to BPP ones (Monte-Carlo) “in the limit”, and assuming certificate complexity is not large ($R_0^*(f) = \max(R^*(f), C^*(f))$). This establishes a notion of equivalence between BPP and ZPP in the query complexity model.

¹The exact choice of constant is not important as long as it is $> 1/2$ (due to amplification), but $2/3$ is the standard

1.3 Composed Functions

Composition is a central concept in the study of Boolean functions: many functions of interest can be represented as compositions of simpler functions. For Boolean functions $f: \{0, 1\}^n \rightarrow \{0, 1\}$ and $g: \{0, 1\}^m \rightarrow \{0, 1\}$, their composition (sometimes called “block composition”) is the Boolean function $f \circ g: \{0, 1\}^{nm} \rightarrow \{0, 1\}$ defined by applying g to n independent inputs, and plugging the resulting n -bit string into f to get a 1-bit answer.

Our main interest is a specific class of composed functions, called recursively composed functions. That is, using f^k to denote the composition $f \circ f \circ \dots \circ f$ (with k levels of f), we wish to understand the limiting behavior of f^k as $k \rightarrow \infty$. This is called the composition limit. Recursively composing a Boolean function has become a standard tool in query complexity: functions constructed this way are commonly used to provide separations between query complexity measures, among many other uses. For some examples of specific recursively composed functions, see [1, 3, 5, 11, 12, 19, 20, 25, 26, 28, 30]. In Chapter 3, we introduce the recursive 3-majority problem, a prominent example in the literature whose randomized complexity is still not fully understood.

The composition limit of a query-complexity measure, $M(f)$, is defined as

$$\lim_{k \rightarrow \infty} \sqrt[k]{M(f^k)}$$

This value is denoted $M^*(f)$. In the paper, we primarily concern ourselves with the composition limits of the following query complexity measures: $R(f)$ (bounded-error randomized complexity), $R_0(f)$ (zero-error randomized complexity), and $C(f)$ (certificate complexity). Our main result defines a relationship between these.

1.4 Quantum Query Complexity

While our focus is on classical randomized query complexity, it is worth noting the significant role of quantum computing in this area. Quantum query complexity, introduced by Bernstein and Vazirani [8], offers a natural quantum analogue to classical query complexity. In this model, quantum algorithms are allowed to make queries in superposition, potentially offering substantial speedups over classical algorithms. Beals et al. [6] established fundamental relationships between quantum and classical complexity measures.

The relationship between quantum and classical query complexity is an active area of research. For many problems, quantum query complexity can be quadratically smaller

than classical randomized query complexity [14]. In some cases, the separation can be even more dramatic. For instance, Shor’s period-finding algorithm [27], which forms the basis of quantum factoring, demonstrates an exponential separation between quantum and classical query complexities for a partial function.

However, the exact relationship between quantum and classical query complexities remains an open question. Recent work by Ben-David and Kothari [7] has shown that classical lower bounds can sometimes be derived from quantum upper bounds, highlighting the deep connections between these models. Understanding these relationships not only sheds light on the power of quantum computation but also provides new tools for analyzing classical algorithms.

The techniques developed for analyzing quantum query complexity, such as the adversary method [2] and the polynomial method [6], have also found applications in classical complexity theory. This cross-pollination of ideas underscores the importance of studying query complexity in both classical and quantum settings.

1.5 Our Results

Our main result concerns the relationship between R^* and R_0^* , bounded-error and zero-error randomized query complexity respectively.

Theorem 1 (Main). *For all possibly-partial boolean functions f , $R_0^*(f) = \max(R^*(f), C^*(f))$*

In other words, we show that it is possible to reduce “Las-Vegas” algorithms (R_0) to “Monte-Carlo” algorithms (R) with no additional cost unless the certificate complexity is large.

Our main contribution is the upper bound on R_0^* . It’s easy to see that $R_0^*(f) \geq C^*(f)$, as any zero-error random algorithm must produce a certificate. We can see that $R_0^*(f) \geq R^*$ as a bounded-error random algorithm can simulate a zero-error one, and halt after a multiple of $R_0(f)$ steps. Theorem 1 implies, in particular, that bounded error and zero error complexities of the 3-majority problem (3-MAJ) are equal. Even this special case was not previously known, despite significant work on these two measures [13, 18, 21, 22].

1.5.1 Methods

In order to prove Theorem 1, we need to show how one can use a bounded-error randomized algorithm R for f^k in order to find a certificate for f^k (assuming k is large compared to

the size of f). The function f^k can be represented as a tree of depth k , where each node has n children and applies the gate f to their values; at the bottom of the tree is the input of length n^k .

Our approach is to construct a zero-error algorithm is to use a bounded-error algorithm to evaluate each of the n children of the root of the tree; that is, we run a bounded-error algorithm for each of the n copies of f^{k-1} that are fed into the outermost copy of f . We amplify each of these runs to reduce the error. This gives us a string of length n of the best-guess values for the input to the outermost f . Then, assuming this n -bit string is the true input, we choose a certificate C for it which is cheapest in a relevant sense. Finally, we recursively call the zero-error algorithm on each of the copies of f^k that are used by the certificate C ; this algorithm produces a certificate for each of these copies, and together they form a certificate for the outer copy of f , which we return.

There are several subtleties we need to address in this argument. One is that we want to achieve the upper bound $C^*(f)$ (at least if $C^*(f) \geq R^*(f)$). A priori, this could require us to choose the certificate C very carefully for the outermost copy of f . The optimal certificate isn't necessarily the smallest certificate for the input, because it is possible that the smallest certificate uses only (say) 0 bits, which will cause our recursive calls to look for certificates for 0-inputs to f^{k-1} . This is a problem if 0-inputs are harder to certify than 1-inputs: in that case, we would prefer to choose a certificate that uses fewer 0 bits, even if it is larger overall due to the number of 1 bits it uses. However, it ends up being that case that the naive choice of the shortest certificate at every level is sufficient to show the upper bound. Here we use the characterization of $C^*(f)$ provided by [12] in terms of the largest eigenvalue in a certain set of 2×2 matrices associated with the function f .

Another subtlety is that running the bounded-error algorithm for each copy of f^{k-1} loses a factor of n as well as a factor depending on our amplification. We will use a very large amount of amplification to get the probability of error to be extremely small and minimize the chances that the algorithm fails (even considering the recursive calls). However, recall the definition of the composition limit $R_0^*(f) = \lim_{k \rightarrow \infty} R_0(f^k)^{1/k}$. The power $1/k$ turns a multiplicative factor of n into a multiplicative factor of $n^{1/k}$, which approaches 1 as $k \rightarrow \infty$. This allows us to lose any multiplicative factor of the form $2^{o(k)}$, and since error amplification is exponentially efficient, we can get the errors down to $2^{-2^{k^{1-\epsilon}}}$ for any $\epsilon > 0$.

1.6 Organization

Chapter 2 contains some preliminary definitions on the basics of query complexity. For further reference, see [\[10\]](#), a comprehensive introduction to query complexity. In Chapter 3, we introduce the recursive 3-majority problem, a much studied boolean function whose randomized query complexity remains an open problem. We explore one possible generalization. In particular, we analyze 3-majority as an instance of a combinatorial game we call “denial nim”, and then briefly explore some of its properties. In Chapter 4, we introduce some preliminary definitions and results before proving [Theorem 1](#).

Chapter 2

Preliminaries

2.1 Notation

For all $n \in \mathbb{N}$, $[n]$ by convention will denote the set $\{1, \dots, n\}$. Given a function f , $\text{Dom}(f)$ will refer to the domain of f . Given a pseudobinary string $s \in \{0, 1, *\}^n$, we use $|s|_1$ to denote the number of ones in s , $|s|_0$ to denote the number of zeroes in s , and $|s| = |s|_0 + |s|_1$. Given an $m \times n$ matrix A , the infinity norm $\|A\|_\infty$ is the maximum absolute row sum of the matrix,

$$\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^n |a_{ij}|$$

2.2 Boolean Functions

Boolean functions are fundamental objects of study in query complexity and theoretical computer science in general. They form the basis for understanding more complex computational problems and are central to our study of recursively composed functions.

Definition 2 (Boolean Function). *A Boolean function f is a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, where $n \in \mathbb{N}$ is called the arity of f .*

Boolean functions can be represented in various ways, such as by truth tables, decision trees (see Section 2.3), boolean formulas, and polynomials [24].

The more generalized object of study is the “possibly-partial” n -ary boolean function, whose domain is a subset of $\{0, 1\}^n$.

Definition 3 (Possibly-Partial Boolean Functions). *A possibly-partial n -ary Boolean function f is a function $f : S \rightarrow \{0, 1\}$, where $S \subseteq \{0, 1\}^n$*

Several complexity measures have been defined for Boolean functions, each capturing different aspects of the function’s structure or difficulty of computation. We’ve already encountered certificate complexity in Section 2.4. Two other important measures are sensitivity and block sensitivity.

Definition 4 (Sensitivity). *For $x \in \{0, 1\}^n$, the sensitivity of f on x , denoted $s_f(x)$, is the number of bits in x that, when flipped, change the value of f . The sensitivity of f , denoted $s(f)$, is $\max_{x \in \{0, 1\}^n} s_f(x)$.*

Definition 5 (Block Sensitivity). *For $x \in \{0, 1\}^n$, the block sensitivity of f on x , denoted $bs_f(x)$, is the maximum number of disjoint blocks of bits in x that, when flipped, change the value of f . The block sensitivity of f , denoted $bs(f)$, is $\max_{x \in \{0, 1\}^n} bs_f(x)$.*

The relationship between these measures has been a subject of intense study. It is known that for all total Boolean functions f , $s(f) \leq bs(f) \leq C(f)$ [23]. The long-standing Sensitivity Conjecture, which posited that $bs(f)$ is polynomially bounded by $s(f)$, was recently resolved in the affirmative by Huang [16]. These complexity measures often relate closely to various query complexities. For instance, the certificate complexity $C(f)$ provides a lower bound for the deterministic query complexity $D(f)$, while block sensitivity $bs(f)$ is known to be polynomially related to the randomized query complexity $R(f)$ [23].

2.3 Deterministic Query Complexity

We’ll start by defining some of the basic terms in query complexity. We first define “possibly-partial” Boolean functions, the object of study in query complexity:

As described in the introduction, query algorithms work by querying the bits of some input $x \in \text{Dom}(f)$ in order to determine $f(x)$. In the deterministic case, a querying algorithm can be formalized as a decision tree

Definition 6 (Decision Tree). *A decision tree T on n bits is a root-ordered binary tree, where*

- the internal nodes are labeled by $[n]$, with no internal node having the same label as any ancestor.
- the leaves are labeled by $\{0, 1\}$
- the arcs (branches) are labeled by $\{0, 1\}$

For $n \in \mathbb{N}$, denote by $\mathcal{T}_n$ the set of all decision trees on n -bits.

Given an input $x \in \text{Dom}(f)$, a decision tree T for computing $f(x)$ would begin by querying a bit of the input. This bit is labeled on the root of the tree, $i \in [n]$. Based on the value of $x_i \in \{0, 1\}$, the algorithm takes the branch labeled x_i , and recursively works its way down the tree according to this process. Once it reaches a leaf, the process is over, with the value of the leaf as the output of the execution. This value is denoted $T(x)$. T is said to *compute* f if and only if for all $x \in \text{Dom}(f)$, $T(x) = f(x)$.

On a given input x , we call $\text{height}(T, x)$ the length of the path taken by executing T on x . Notice that the worst-case cost of a decision tree is represented by the maximum-height of the tree, i.e. $\max_{x \in \text{Dom}(f)} \text{height}(T, x) = \text{height}(T)$

Definition 7 ($D(f)$). *The deterministic query complexity of a possibly-partial n -ary boolean function f , denoted $D(f)$, is*

$$\min_{T \in \mathcal{T}_n: T \text{ computes } f} \text{height}(T)$$

2.4 Randomized Query Complexity

Randomized query-algorithms are defined in terms of decision trees. A randomized decision tree R on n -bits is a probability distribution on $\mathcal{T}_n$. We define the “height” of R as the maximum height of all decision trees in the support of R .

In defining the *cost* of a randomized decision tree R , there are two main approaches. The Monte-Carlo notion of random algorithms allows some error $\varepsilon \in (0, 1/2)$, and measures the cost as the height of R . This gives us one definition of randomized query complexity:

Definition 8 ($R_\varepsilon(f)$). *First, we say that a randomized decision tree R on n -bits computes a possibly-partial n -ary boolean function f with error $\varepsilon \in (0, 1/2)$ iff*

$$\Pr_{T \sim R} [T(x) \neq f(x)] < \varepsilon$$

henceforth we write $R(x)$ instead of $T(x)$ and $T \sim R$. Let $\mathcal{R}(f, \varepsilon)$ denote the set of randomized decision trees computing f with error ε .

Then, we define $R_\varepsilon(f)$ as

$$\min_{R \in \mathcal{R}(f, \varepsilon)} \text{height}(R)$$

Furthermore, $R(f) := R_{1/3}(f)$

In the alternative definition of randomness (“Las-Vegas”), we insist on zero-error, but relax the cost measure to *average* query complexity. In this way we measure the cost of a randomized decision tree by its *expected* height.

Definition 9 ($R_0(f)$). Where f is a possibly-partial n -ary boolean function, and $\mathcal{R}(f)$ denotes the set of randomized decision trees that compute f to zero-error, we define $R_0(f)$ as

$$\min_{R \in \mathcal{R}(f)} \mathbb{E}_{T \sim R}(\text{height } T)$$

It is known that $D(f) \leq R_0(f)^2$. [9, 15, 29]. Furthermore, for all $\varepsilon \in (0, 1/2)$, $D(f) \in O(R_\varepsilon(f)^3)$ (when f is total) [23].

2.5 Certificates

Another fundamental query-complexity measure is certificate complexity, which measures the minimum number of bits required to certify the value of a boolean function f on a given input $x \in \text{Dom}(f)$. For example, when $f = \text{OR}_n$, and $x \neq 0^n$, $C_x(f) = 1$. However, $C_{0^n}(f) = n$, as any verification scheme is forced to verify that each input bit is 0. When we write $C(f)$ without an input subscript, we are referring to the worst case certificate complexity.

Definition 10 (Certificate and Certificate Complexity). For possibly-partial n -ary boolean function f , and an input $x \in \text{Dom}(f)$, a certificate $c \in \{0, 1, *\}^n$ for x with respect to f is a string satisfying:

- $c_i \neq * \implies c_i = x_i$
- $\exists b \in \{0, 1\}$ such that for any $y \in \{s \in \{0, 1\}^n \mid c_i \neq * \implies s_i = c_i\}$, $f(y) = b$

The “length” of c is defined as $|c| := |c|_0 + |c|_1$. The certificate complexity $C_x(f)$ is the minimum length certificate of x with respect to f . Finally,

$$C(f) := \max_{x \in \text{Dom}(f)} C_x(f)$$

Certificate complexity is usually small compared to the query complexities of algorithms meant to determine the value of a function. When the output must be certain (as in deterministic or zero-error randomized), certificate complexity will not exceed the query complexity of the most optimal algorithms. However, for bounded error-randomized algorithms (which don’t need to produce a certificate), there are examples for which $C(f)$ is greater than $R(f)$. We saw this for the gap majority function in the introduction, whereby $C(\text{GAPMAJ}_n) = n/3 > 1 = R(\text{GAPMAJ}_n)$. Because of examples like this, Theorem 1 is “tight” in the sense that $R^*(f) \neq C^*(f)$ in general.

We use the following notation for *certificate space*:

Definition 11. Let $f \in \mathcal{F}$ and let $x \in \text{Dom}(f)$. Denote

$$\text{Cert}_f(x) := \{p \in \{0, 1, *\}^n \mid p \text{ is a certificate for } x \text{ with respect to } f\}$$

Furthermore, for $c \in \text{Cert}_f(x)$, we define

$$\begin{aligned} |c|_0 &:= |\{i \in [n] \mid c_i = 0\}| \\ |c|_1 &:= |\{i \in [n] \mid c_i = 1\}| \\ |c| &:= |c|_0 + |c|_1 \end{aligned}$$

2.6 Measures and Composition

$D(f)$, $R(f)$, and $C(f)$ are all examples of measures. In general, a measure is a positive-real valued function on boolean functions.

Definition 12. For $n \in \mathbb{N}$, let $\mathcal{F}_n$ denote the set of all possibly partial Boolean functions on n bits. Let $\mathcal{F} := \bigcup_{n=1}^{\infty} \mathcal{F}_n$ denote the set of all (possibly partial) Boolean functions.

Definition 13 (Measure). *A measure M on a subset $\mathcal{A} \subseteq \mathcal{F}$ is a function $M: \mathcal{A} \rightarrow [0, \infty)$.*

Composition plays an important role in query complexity and in our work. Boolean function composition is defined in a special way (sometimes called “block composition”) which is different from how usual function composition works. Given a boolean function $f: \{0, 1\}^m \rightarrow \{0, 1\}$ and $g: \{0, 1\}^n \rightarrow \{0, 1\}$, $f \circ g$ is the function on $\{0, 1\}^{mn}$ that applies f to m copies of g .

Definition 14 (Composition). *Given a boolean function $f: \{0, 1\}^m \rightarrow \{0, 1\}$ and $g: \{0, 1\}^n \rightarrow \{0, 1\}$, $f \circ g$ is defined as the boolean function $f \circ g: \{0, 1\}^{mn} \rightarrow \{0, 1\}$, where for $x_1, \dots, x_m \in \{0, 1\}^n$ and $x := x_1 \cdots x_m$,*

$$(f \circ g)(x) = f(g(x_1) \cdots g(x_m))$$

In the case of f or g being partial boolean functions, $f \circ g$ is restricted to the domain where each $x_i \in \text{Dom}(g)$, and the string $g(x_1) \cdots g(x_m) \in \text{Dom}(f)$.

When we compose a function with itself, e.g. $f \circ f$, we use superscript notation, e.g. f^2 . For k self-compositions of f , we write f^k . Now we define the composition *limit* of a query complexity measure M .

Definition 15 (Composition Limit). *Where $\mathcal{F}$ is the set of all possibly-partial boolean functions, and $M: \mathcal{A} \subseteq \mathcal{F} \rightarrow [0, \infty)$ is a measure, the composition limit of $M(f)$ for $f \in \mathcal{A}$ is*

$$\lim_{k \rightarrow \infty} \sqrt[k]{M(f^k)}$$

Chapter 3

Monte Carlo to Las Vegas

In this chapter we prove [Theorem 1](#), showing that $R_0^*(f) = \max(R^*(f), C^*(f))$ for all possibly-partial Boolean functions f . This establishes a reduction from Las Vegas algorithms to Monte Carlo algorithms for recursively composed functions.

Before we begin the proof, we establish some preliminary results about measures:

Lemma 16. *For any measures $M, N : \mathcal{A} \subseteq \mathcal{F} \rightarrow [0, \infty)$, $f \in \mathcal{A}$, and $k \in \mathbb{N}$, the following composition-limit properties hold:*

1. $M^*(f^k) = M^*(f)^k$
2. $M^{**}(f) = M^*(f)$
3. $M(f) \leq N(f) \implies M^*(f) \leq N^*(f)$
4. $(M(f) \log M(f))^* = M^*(f)$
5. $(M(f) + N(f))^* = \max(M^*(f), N^*(f))$

Proof. 1. $M^*(f^k) = \lim_{l \rightarrow \infty} \sqrt[l]{M(f^{lk})} = (\lim_{l \rightarrow \infty} \sqrt[lk]{M(f^{lk})})^k = M^*(f)^k$

2. $M^{**}(f) = \lim_{k \rightarrow \infty} \sqrt[k]{M^*(f^k)} = \lim_{k \rightarrow \infty} \sqrt[k]{M^*(f)^k} = M^*(f)$

3. by the fact that $\sqrt[k]{\cdot}$ is nondecreasing

$$4. (M(f) \log M(f))^* = \lim_{k \rightarrow \infty} \sqrt[k]{M(f^k) \log M(f^k)}$$

Consider $\log \sqrt[k]{M(f^k) \log M(f^k)} = (1/k)(\log M(f^k) + \log \log M(f^k))$. Then $\log M(f^k) = k \ln M^*(f) + o(k)$, and $\log \log M(f^k) = \log k + \log \log M^*(f) + o(1)$. Thus $\frac{\log \log M(f^k)}{k} \rightarrow 0$. Exponentiating both sides of the limit now yields the desired result.

$$= M^*(f) \lim_{k \rightarrow \infty} \sqrt[k]{k \log M^*(f)} = M^*(f)$$

$$5. (M(f) + N(f))^* = \lim_{k \rightarrow \infty} \sqrt[k]{M(f^k) + N(f^k)} \leq \lim_{k \rightarrow \infty} \sqrt[k]{2 \max(M(f^k), N(f^k))} = \max(M^*(f), N^*(f))$$

□

To prove the upper-bound, we will craft a zero-error random algorithm which simulates a bounded-error random algorithm to make use of high-probability guesses.

3.1 Proof of Main Theorem

We begin with the lower-bound direction. It is straightforward, but we prove it for completion:

Lemma 17. *For all (possibly-partial) Boolean functions f , $R_0^*(f) \geq \max(R^*(f), C^*(f))$.*

Proof. By Lemma 16, it suffices to show that $R_0(f) = \Omega(C(f))$ and $R_0(f) = \Omega(R(f))$ for all (possibly partial) Boolean functions f . These are both classic results, but we go over the proofs for completeness.

If a zero-error algorithm does not query every bit of some certificate for f on a given input x , then by definition there would be some input y consistent with the queried bits which has $f(y) \neq f(x)$. This means the behavior of the algorithm on y can be the same as the behavior on x with non-zero probability, which implies that the algorithm is not zero-error, a contradiction. This establishes $R_0(f) \geq C(f)$.

As for $R(f)$: let Z be a zero-error random algorithm with average-case query complexity $R_0(f)$. Consider the following bounded error-algorithm Y : simulate Z for at most $3 R_0(f)$ steps. If Z terminates, return its output. If not, return 0. By Markov's inequality, this is a bounded-error random algorithm with query complexity $\leq 3 R_0(f)$. □

Upper Bound

It remains to show that $R_0^*(f)$ is upper bounded by $\max\{R^*(f), C^*(f)\}$. To do this, we will design a zero-error algorithm for computing f^k . Then we will demonstrate that this algorithm witnesses $R_0^*(f) \leq R(f) \log R(f) + C(f)$. Then, we will use the properties of measures (Lemma 16) to recover the desired upper bound.

We define a Las Vegas algorithm $A_f(x, k)$ to compute $f^k(x)$. See Algorithm 1 and the helper subroutine Algorithm 2. To informally describe the algorithm:

1. For the base case of a single-bit input, return that bit
2. Otherwise, let T be a Monte-Carlo algorithm for f^{k-1} , and simulate T on the children of the root of the composition tree represented by x . Repeat this simulation a large number of times for each of the n children in order to reduce the error to a very small amount. Get a tentative value for each of the n subtrees of the root, denoted by the n -bit string $\tilde{t}$.
3. Choose from the string of tentative values a minimal-length certificate $c(t)$. Then, for each bit used by that certificate, recurse the algorithm on the subtree corresponding to that bit.
4. Piece together the certificates of the children, and output the full certificate for $f^k(x)$.

Most of the work of $A_f(x, k)$ will be done by a helper function $A'_f(x, k)$ defined in Algorithm 2. Since A' may err in its tentative evaluations, A' has an upper-limit on the number of queries it makes, and will return NULL if it exceeds it. This allows a fresh evaluation by the main function A_f .

Algorithm 1: $A_f(x, k)$

Input: $k \in \mathbb{N}, x \in \text{Dom}(f^k)$

Output: $c \in \text{Cert}(f, x)$

- 1 $c \leftarrow A'_f(x, k)$
 - 2 **if** $c = \text{NULL}$ **then**
 - 3 return $A_f(x, k)$
 - 4 return c
-

Algorithm 2: $A'_f(x, k)$

Input: $k \in \mathbb{N}, x \in D(f^k)$ **Output:** $c \in \text{Cert}(f, x)$

```
1 if  $k = 0$  then
2   this is the case of the identity function on a single bit; simply query the bit and
   return its value.
3 else
4    $m_f \leftarrow C(f) + R(f) \log R(f)$ 
5    $M_f(k) \leftarrow m_f^{k^{1.5}}$ 
6   Track the number of queries the algorithm makes; if it ever exceeds  $M_f(k)$ ,
   terminate and return NULL.
7   Let  $T^{k-1}$  be an optimal bounded-error algorithm for  $f^{k-1}$ .
8   Run  $T$  on each of the children  $x_1, \dots, x_n$  of the root of the composition tree
    $200 \log(m_f) k^2$  times each to get their tentative values  $\bar{t} = (t(x_1), \dots, t(x_n))$ .
9   Select a minimal-length certificate  $c'$  from these tentative values.
10  foreach  $i$  such that  $c'_i \neq *$  do
11     $c(i) \leftarrow A'_f(k-1, x_i)$ .
12    While  $f^{k-1}(c(i)) \neq t(x_i)$ , repeat  $c(i) \leftarrow A'_f(k-1, x_i)$ 
13    If the output is NULL, return NULL.
14  end
15  Return as a certificate the concatenation of the certificates  $c(i)$ 
16 end
```

Let $q^f(x, k)$ denote the expected query complexity of $A'_f(x, k)$, and likewise $Q^f(x, k)$ for $A_f(x, k)$. Let $q^f(k) := \max_{x \in \text{Dom}(f^k)} q^f(x, k)$, and likewise $Q^f(k)$ for $Q^f(x, k)$. Let $S_{A'_f}(x, k)$ represent the number of queries A'_f makes on input x, k .

Lemma 18. *For all possibly-partial boolean functions f , $\lim_{k \rightarrow \infty} \sqrt[k]{q^f(k)} \leq R(f) \log R(f) + C(f)$, and $\lim_{k \rightarrow \infty} \sqrt[k]{\Pr[S_{A'_f}(x, k) > M_f(k)]} = 0$*

Proof. We will begin by analyzing $q^f(x, k)$. Use the notation from [Algorithm 2](#).

Analysis

Error. We first bound the error incurred by the subroutine T^{k-1} . The probability of a given tentative evaluation $t(x_i)$ being incorrect is the probability that the majority-output of T^{k-1} after $200 \log(m_f)k^2$ runs is incorrect. T^{k-1} has error $\leq 1/3$ per-run. By standard amplification bounds, the probability of t_i being incorrect is at most $m_f^{-k^2}$. Union-bounding over all n tentative evaluations, the probability that any tentative evaluation is incorrect is at most $nm_f^{-k^2}$. Let $p_{\text{err}}(k)$ denote this value.

Let $r(k) := 200n \log(m_f)k^2$ denote the number of times T^{k-1} is run by $A'_f(x, k)$. Then,

$$q^f(x, 0) = 1$$

$$q^f(x, k) \leq p_{\text{err}}(k)M_f(k) + (1 - p_{\text{err}}(k))(r(k)R(f^{k-1}) + C(f)q^f(x, k-1))$$

In the case of error (with probability $\leq p_{\text{err}}(k)$), we assume the worst-case, that the algorithm reaches its maximum allowed $M_f(k)$ before it terminates itself and returns NULL. Otherwise, there are $r(k)$ runs of T^{k-1} , each with worst-case complexity $R(f^{k-1})$. Since all of the values computed by T^{k-1} are correct, we find a certificate of length at most $C(f)$ from our string $\bar{t}$, and we recurse over those children.

Notice our expression is independent of the input x . Let $E(k) = 2p_{\text{err}}(k)M_f(k)$. We can re-write this expression as:

$$q^f(k) \leq E(k) + r(k)R(f^k) + C(f)q^f(k-1)$$

By repeated substitution:

$$q^f(k) \leq \sum_{i=0}^k C(f)^i (r(k-i)R(f^{k-i}) + E(k-i))$$

Now we use the fact that $R(f \circ g) \leq O(R(f)R(g) \log R(f))$ for all composable, possibly-partial boolean functions f, g .

$$q^f(k) \leq \sum_{i=0}^k C(f)^i (r(k-i)R(f)^{k-i} \log^{k-i}(R(f)) + E(k-i))$$

We use two facts to simplify this expression:

- $r(i)$ is monotonically increasing
- $E(i) \leq 2m_f^{i^{1.5}-i^2}$ is ≤ 2 for all $i \in \mathbb{Z}$

Then,

$$q^f(k) \leq 3r(k) \sum_{i=0}^k C(f)^i R(f)^{k-i} \log^{k-i} R(f)$$

$$q^f(k) \leq 3r(k)(C(f) + R(f) \log R(f))^k$$

So

$$\sqrt[k]{q^f(k)} \leq \sqrt[k]{3r(k)(C(f) + R(f) \log R(f))^k}$$

Now we analyze this expression as $k \rightarrow \infty$. $\sqrt[k]{3r(k)} = \sqrt{k}[600 \log(m_f)k^2]$, so $\sqrt[k]{3r(k)} \rightarrow 1$ as $k \rightarrow \infty$.

$$\lim_{k \rightarrow \infty} \sqrt[k]{q^f(k)} \leq C(f) + R(f) \log R(f)$$

as desired.

This concludes the proof of the first part of the lemma. Now we prove the second part of the lemma, upper-bounding the probability that $A'_f(x, k)$ exceeds the maximum $M_f(k)$ queries and thus returns NULL. By Markov's inequality,

$$\lim_{k \rightarrow \infty} \Pr[S_{A'_f}(x, k) > M_f(k)]^{1/k} \leq \lim_{k \rightarrow \infty} \frac{\sqrt[k]{q^f(x, k)}}{\sqrt[k]{M_f(k)}} \rightarrow 0$$

We've just shown that the numerator is upperbounded by $R(f) \log R(f) + C(f) = m_f$ in the limit. The denominator is $m_f^{k^{1.5}}$, which goes to infinity. We've thus shown $\lim_{k \rightarrow \infty} \Pr[S_{A'_f}(x, k) > M_f(k)]^{1/k} = 0$ as desired.

□

Now we use this lemma to prove our upperbound for $Q^f(x, k)$.

Lemma 19. *For all possibly partial boolean functions f , $\lim_{k \rightarrow \infty} Q^f(k) \leq R(f) \log R(f) + C(f)$*

Proof. Let $p_{\text{NULL}}(x, k)$ denote the probability that $A'_f(x, k)$ exceeds its $M_f(k)$ query-limit.

$$\begin{aligned} Q^f(x, k) &\leq q^f(x, k) + p_{\text{NULL}} Q^f(x, k) \\ Q^f(k) &\leq \frac{1}{1 - p_{\text{NULL}}} q^f(k) \\ \lim_{k \rightarrow \infty} Q^f(k)^{1/k} &\leq \sqrt[k]{\frac{1}{1 - p_{\text{NULL}}}} q^f(k)^{1/k} \end{aligned}$$

By Lemma 18, $\lim_{k \rightarrow \infty} \sqrt[k]{p_{\text{NULL}}} = 0$, so $\sqrt[k]{\frac{1}{1 - p_{\text{NULL}}}} \rightarrow 1$, and $q^f(k)^{1/k} \rightarrow R(f) \log R(f) + C(f)$. This concludes the proof of Lemma 19

□

Proof of Main Theorem:

Since $Q^f(x, k)$ is the expected query complexity of a zero-error algorithm computing $f^k(x)$, $R_0(f^k) \leq Q^f(k)$. Thus by the above lemma, $R_0^*(f) \leq \lim_{k \rightarrow \infty} \sqrt[k]{Q^f(k)} \leq R(f) \log R(f) + C(f)$. Now we use the properties from Lemma 16 to get the final bound:

$$\begin{aligned} R_0^{**}(f) &\leq (R(f) \log R(f) + C(f))^* \\ R_0^*(f) &\leq \max((R(f) \log R(f))^*, C^*(f)) \\ R_0^*(f) &\leq \max(R^*(f), C^*(f)) \end{aligned}$$

as required ■

As mentioned in the introduction, one corollary of this result is that bounded-error random algorithms for the 3-majority problem will be no cheaper than zero-error:

Corollary 20. $R_0^*(3\text{-MAJ}) = R^*(3\text{-MAJ})$

Chapter 4

3-Majority and Denial Nim

Before demonstrating our main results, we will explore the recursive 3-majority problem to develop an intuition about randomized query complexity, as well as discuss one possible generalization of this important problem.

4.1 3-MAJ

The recursive 3-majority problem involves constructing a recursive majority function on a ternary tree, where each node computes the majority of its three children. The randomized query complexity of recursive 3-majority is an open problem. Foundational work was done by Saks and Wigderson [26] on read-once formulae, a type of problem for which 3-majority is a member. They were among the first to present a function for which random methods outperformed deterministic ones. Magniez et al. improved their upper and lower bounds on recursive 3-majority in [22],

We start by defining the 3-MAJ_k function. Let MAJ : {0, 1}³ → {0, 1} be the majority function on three bits. Then 3-MAJ_k : {0, 1}^{3k} → {0, 1} is defined recursively as follows. When $k = 0$, 3-MAJ_k is the identity function. When $k > 0$, and $x_1, x_2, x_3 \in \{0, 1\}^k$, 3-MAJ_k(x_1, x_2, x_3) = MAJ(3-MAJ_{k-1}(x_1), 3-MAJ_{k-1}(x_2), 3-MAJ_{k-1}(x_3)).

It's straightforward to see that the deterministic complexity of 3-MAJ_k is maximal, i.e. $D(3\text{-MAJ}_k) = 3^k$. This can be seen by induction; when $k = 0$, 3-MAJ_k is the identity and takes $3^0 = 1$ queries. For the $k = 1$ scenario, 3-MAJ₁ is simply the majority function, and an adversarial assignment of 1, 0, 1 (or 1, 0, 0) to the sequence of queries would force all 3 bits to be queried. When $k > 1$, and using the tree representation of 3-MAJ_k, computing

the root requires computing the majority of its three children. Again, an adversarial assignment can be made to the subtrees in a recursive way so that evaluation of all three children is required.

4.1.1 Random Methods

However, even a naive random approach provides a significant improvement. Consider the following algorithm for 3-MAJ_k:

- Select two of the three subtrees of the root uniformly at random, and recursively evaluate them
- If they are not equal, recursively evaluate the last subtree

Notice this a zero-error algorithm. Let us denote by C_k the average query cost of this algorithm for 3-MAJ_k. In the case of $k = 0$, $C_0 = 1$. In the case of $k > 0$: in the worst case, the 3 subtree values are two ones and one zero, or the opposite (two zeroes and one one). The likelihood of selecting the two majority subtrees for evaluation is the probability of excluding the minority bit, $1/3$. In this case we perform two recursive evaluations. Otherwise, we are forced to perform all three. This gives us a recurrence relation:

$$C_k = (1/3)(2C_{k-1}) + (2/3)(3C_{k-1}) = (8/3)C_{k-1}$$

Thus we can conclude $C_k = (8/3)^k = 2.666\dots^k$.

Magniez et al. have the tightest current upper bound, showing $R_0(3\text{-MAJ}_k) \leq (1.007) \cdot 2.64944^k$. Göös & Jayram have the tightest lower bound, showing that $R_0(3\text{-MAJ}_k) \geq \Omega(2.59^k)$ [13].

4.2 Denial Nim

One way to gain more insight about the 3-majority problem is to explore generalizations. There are several possible generalizations, the most obvious of which is recursive k -majority on a k -ary tree. However, we will explore a generalization that comes from viewing the 3-majority problem as a game-tree for a combinatorial game with the following basic properties:

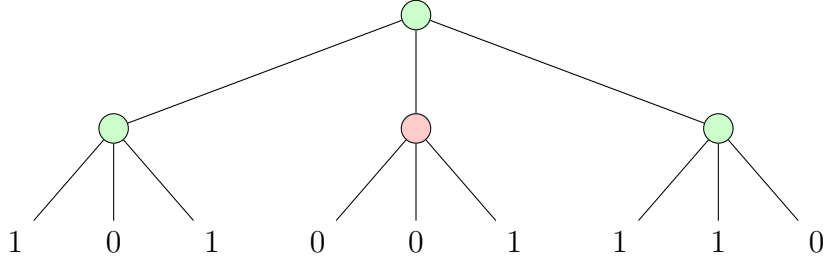


Figure 4.1: A height 2 3-majority tree. Green represents a value of 1 and Red a value of zero

- There are always k possible moves on each turn
- On a player's turn, their opponent can deny one of the possible moves

Though a number of combinatorial games can be used to formalize these properties, we use the following, which we call “denial nim”. k -Nim is a game played on k piles, each containing a certain number of objects. On a players turn, they must select a pile and remove at least one object from it. The player to remove the last object wins. For our purposes, we define a variant of Nim where (i) *exactly* one object must be removed per turn, and (ii) the first player to deplete a pile wins. Call this variant Nim_1 . Now we define k -Denial Nim as a version of Nim_1 , defined as follows

Definition 21. k -Denial Nim is a Nim_1 game on k piles, where

- the previous player must deny one pile for the next player's move
- the current player must choose one pile (from the $k-1$ available), and remove exactly one object from it

the first player to reduce a pile to zero wins.

Denial forces the current player to play the “second best” possible move. We now induce from this definition of k -Denial Nim_1 a boolean function.

Let the bit 1 represent N positions (configurations where the next player to move is winning), and 0 represent P positions (previous player win). For the current player to win, they must have two reachable P positions, as one will be denied. The boolean function this translates to is a function that returns 1 if and only if at least two of its input bits are zero. We'll call this k -negated 2-threshold, or N2T^k . We now define k -Denial Nim_1 (or DN^k) as a boolean function recursively

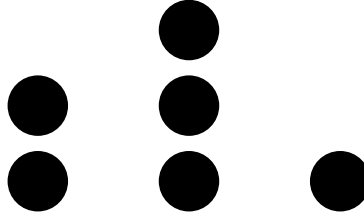


Figure 4.2: An instance of 3-Denial Nim

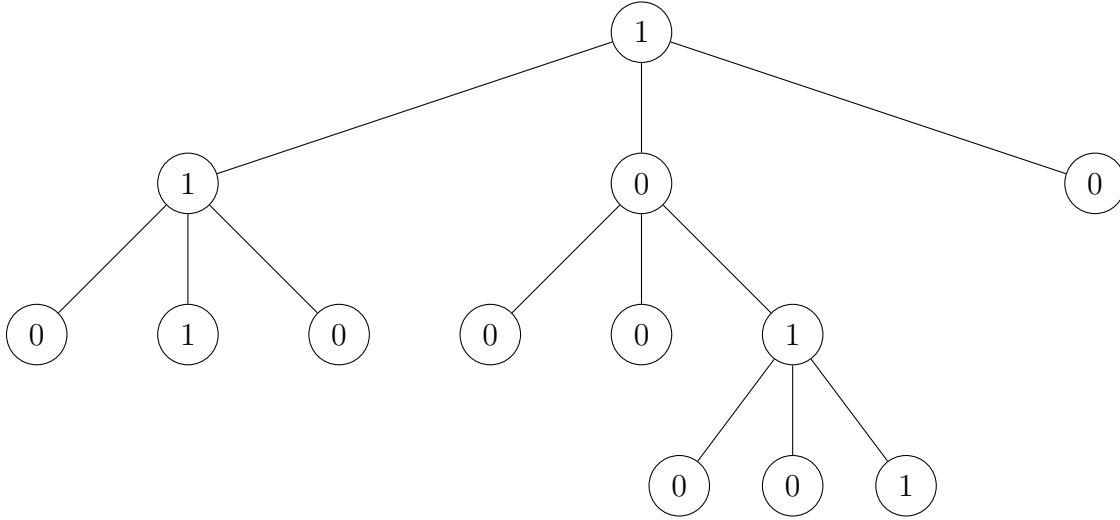


Figure 4.3: Game tree for Figure 4.2

Definition 22. When $h = 0$, DN_h^k is the identity on one bit. When $h > 0$, and $\bar{x} = (x_1, \dots, x_k) \in \{0, 1\}^{k^h}$, then $\text{DN}_h(x)^k = \text{N2T}^k(\text{DN}_{h-1}^k(x_1), \dots, \text{DN}_{h-1}^k(x_k))$, where each $x_i \in \{0, 1\}^{k^{h-1}}$

Note at this point that 3-MAJ_h is a special case of DN_h^k , namely $3\text{-MAJ}_h = (h \bmod 2) \oplus \text{DN}_h^3$. In other words, the output of DN_h^3 is the negated output of 3-MAJ_h when the height h is odd. We can show this by induction on h . When $h = 0$, both 3-MAJ_h and DN_h^3 are the identity, and the output is not negated. When $h > 0$, and h is odd,

$$\begin{aligned} \text{DN}_h(x)^3 &= \text{N2T}^k(\text{DN}_{h-1}^3(x_1), \dots, \text{DN}_{h-1}^3(x_3)) \\ &= \text{N2T}^k(3\text{-MAJ}_{h-1}(x_1), \dots, 3\text{-MAJ}_{h-1}(x_3)) \\ &= 1 - 3\text{-MAJ}_h \end{aligned}$$

as the negated two-threshold of three bits is simply the majority. We apply the inductive hypothesis likewise when h is even:

$$\begin{aligned} \text{DN}_h(x)^3 &= \text{N2T}^k(\text{DN}_{h-1}^3(x_1), \dots, \text{DN}_{h-1}^3(x_3)) \\ &= \text{N2T}^k(1 - 3\text{-MAJ}_{h-1}(x_1), \dots, 1 - 3\text{-MAJ}_{h-1}(x_3)) \\ &= 1 - (1 - 3\text{-MAJ}_h) = 3\text{-MAJ}_h \end{aligned}$$

as required.

Now, we will do a randomized query complexity analysis of the naive algorithm computing DN_h^n , similar to the naive random algorithm for 3-MAJ. In the case of a leaf node, we simply query it and return. Otherwise, we randomly select subtrees to evaluate until the function is computed. Let $f(h|1)$ denote the expected number of queries of the naive algorithm given that $\text{DN}_h^n(x)$ evaluates to 1, and $f(h|0)$ denote the expected number of queries given 0. Then, on worst case inputs, $f(h|0) = (n-1)f(h-1|1) + \frac{n-1}{n}f(h-1|0)$, and $f(h|1) = \sum_{m=2}^n p(m)(2f(h|0) + (m-2)f(h|1))$, where $p(m)$ is the probability we find the two 0s (and therefore can stop) in our first m random choices, i.e.

$$p(m) = 2 \binom{n-2}{m-2} \binom{n}{m-1}^{-1} \frac{1}{n - (m-1)}$$

We can rewrite $f(h|1)$ as

$$f(h|1) = C_1 f(h|0) + C_2 f(h|1)$$

Where $C_1 = 4 \sum_{m=2}^n p(m)$ and $C_2 = 2 \sum_{m=2}^n p(m)(m-2)$. C_1 simplifies to 2 and $C_2 = (2/3)(n-2)$. So we have the recurrence relations

$$\begin{aligned} f(h|0) &= (n-1)f(h-1|1) + \frac{n-1}{n}f(h-1|0) \\ f(h|1) &= \frac{2}{3}(n-2)f(h-1|1) + 2f(h-1|0) \end{aligned}$$

We can express this in a matrix: let $x(h) = \begin{pmatrix} f(h|0) \\ f(h|1) \end{pmatrix}$, then $x(h) = Ax(h-1)$, where

$$A = \begin{bmatrix} \frac{n-1}{n} & n-1 \\ 2 & \frac{2}{3}(n-2) \end{bmatrix}$$

Through calculation we find the principal eigenvalue:

$$\frac{2n^2 + \sqrt{4n^4 + 44n^3 - 11n^2 - 42n + 9} + 7n - 3}{6n}$$

This gives us an asymptotic upperbound on the expected runtime: $R_0(\text{DN}_h^n) = O((2/3)^n)$

Chapter 5

Conclusion

This thesis has explored the relationship between zero-error (Las Vegas) and bounded-error (Monte Carlo) randomized query complexity for recursively composed functions. Our main result, Theorem 1, establishes that $R_0^*(f) = \max(R^*(f), C^*(f))$ for all possibly-partial boolean functions f . This equivalence in the composition limit bridges a significant gap in our understanding of randomized algorithms in the query model.

The implications of this result are far-reaching. For instance, it settles the long-standing question about the relationship between zero-error and bounded-error randomized query complexity for the recursive 3-majority function, showing that $R^*(3\text{-MAJ}) = R_0^*(3\text{-MAJ})$. This is particularly noteworthy given the extensive study this problem has received [13, 22, 26].

Our proof technique, which simulates a bounded-error algorithm with enough amplification that the error terms disappear in the limit, provides a general framework for reducing zero-error algorithms to bounded-error ones in the query complexity setting for recursively composed functions.

The exploration of Denial Nim as a generalization of 3-majority offers a new perspective on recursively composed functions. While preliminary, this approach may yield further insights into the structure of problems with separations between different query complexity measures.

Several questions remain open for future investigation:

- Are there other complexity measures that exhibit interesting relationships in the composition limit?

- Can our techniques be extended to quantum query complexity?
- What are the implications of our results for other areas of theoretical computer science, such as communication complexity or circuit complexity?

In conclusion, this work advances our understanding of randomized query complexity, particularly for recursively composed functions. By establishing a fundamental relationship between zero-error and bounded-error complexity in the limit, we have taken a step towards a more unified theory of randomized computation in the query model.

References

- [1] Scott Aaronson. Quantum certificate complexity. *Journal of Computer and System Sciences*, 74(3):313–322, 2008. Previous version in CCC 2003.
- [2] Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002-06. Previous version in STOC 2000.
- [3] Andris Ambainis. Superlinear advantage for exact quantum algorithms. *SIAM Journal on Computing*, 2016. Previous version in STOC 2013.
- [4] Andris Ambainis. Understanding quantum algorithms via query complexity. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3265–3285. World Scientific, 2018.
- [5] Kaspars Balodis, Shalev Ben-David, Mika Göös, Siddhartha Jain, and Robin Kothari. Unambiguous DNFs and alon-saks-seymour. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2021-02-16.
- [6] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald De Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. Previous version in FOCS 1998.
- [7] Shalev Ben-David, Adam Bouland, Ankit Garg, and Robin Kothari. Classical lower bounds from quantum upper bounds. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2018-10.
- [8] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997-10. Previous version in STOC 1993.
- [9] Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 118–126. IEEE, 1987.

- [10] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.
- [11] 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)*. ACM Press, 2018.
- [12] Justin Gilmer, Michael Saks, and Sudarshan Srinivasan. Composition limits and separating examples for some boolean function complexity measures. *Combinatorica*, 2016. Previous version in CCC 2013.
- [13] Mika Göös and T. S. Jayram. A composition theorem for conical juntas. In *Proceedings of the 31st Conference on Computational Complexity (CCC)*, 2016.
- [14] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 212–219. ACM, 1996.
- [15] Juris Hartmanis and Lane A. Hemachandra. One-way functions and the nonisomorphism of NP-complete sets. *Theoretical Computer Science*, 1991. Previous version in CCC 1987.
- [16] Hao Huang. Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *Annals of Mathematics*, 190(3):949, 2019.
- [17] Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the xor lemma. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 220–229, 1997.
- [18] T. S. Jayram, Ravi Kumar, and D. Sivakumar. Two applications of information complexity. In *Proceedings of the 35th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, STOC03. ACM, 2003-06.
- [19] Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1:26, 2017-08.
- [20] Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Computational Complexity*, 5(3-4):191–204, 1995.

- [21] Nikos Leonardos. An improved lower bound for the randomized decision tree complexity of recursive majority,. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 696–708. Springer Berlin Heidelberg, 2013.
- [22] Frédéric Magniez, Ashwin Nayak, Miklos Santha, Jonah Sherman, Gábor Tardos, and David Xiao. Improved bounds for the randomized decision tree complexity of recursive majority. *Random Structures and Algorithms*, 48(3):612–638, 2015-07. Previous version in ICALP 2011.
- [23] Noam Nisan. Crew prams and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991. Previous version in STOC 1989.
- [24] Noam Nisan and Mario Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity*, 15(4):557–565, 1994. Previous version in STOC 1992.
- [25] Noam Nisan and Avi Wigderson. On rank vs. communication complexity. *Combinatorica*, 15(4):557–565, 1995. Previous version in FOCS 1994.
- [26] Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 29–38. IEEE, 1986.
- [27] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. Previous version in FOCS 1994.
- [28] Avishay Tal. Properties and applications of boolean function composition. In *Proceedings of the 4th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 441–454, 2013.
- [29] Gabor Tardos. Query complexity, or why is it difficult to separate $NP^A \cap coNP^A$ from P^A by random oracles A ? *Combinatorica*, 9(4):385–392, 1989-12-01.
- [30] Ingo Wegener and Laszlo Zádori. A note on the relations between critical and sensitive complexity, 1988.