Quantum Speedups in Query Complexity

by

Shalev Ben-David

B.Math., University of Waterloo (2011)
S.M., Massachusetts Institute of Technology (2014)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

Signature redacted

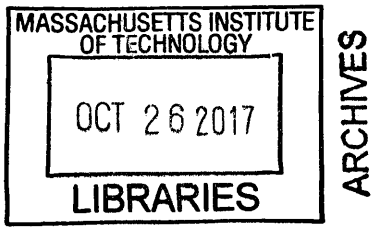
Author .....
Department of Electrical Engineering and Computer Science
August 21, 2017

Signature redacted

Certified by.....
Scott J. Aaronson
Visiting Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Signature redacted

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students



Quantum Speedups in Query Complexity

by

Shalev Ben-David

Submitted to the Department of Electrical Engineering and Computer Science
on August 21, 2017, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In this thesis, we study randomized and quantum algorithms in the query complexity model. We investigate when and by how much quantum algorithms provide a speedup over the best possible classical algorithm in the query complexity setting.

We introduce a total Boolean function that exhibits a power 2.5 quantum speedup compared to the best possible randomized algorithm. In the process, we introduce the "cheat sheet" method for turning partial Boolean functions into total Boolean functions, and examine some of its other applications.

We also study lower bound techniques for randomized algorithms. We introduce a measure called randomized sabotage complexity which lower bounds randomized query complexity and behaves well under compositions. This tool for controlling the randomized query complexity of composed functions combines nicely with the cheat sheet technique, which often features composed functions in its applications. In addition, we study the quantum analogue of this tool, and use it to show a new power 5 relationship between zero-error and bounded-error quantum query complexity.

Finally, we characterize the total Boolean functions that exhibit exponential quantum speedups when their domain is restricted to an arbitrarily chosen set. We show that such a "sculpting" of a quantum speedup is possible if and only if the original total function has many inputs with large certificate complexity. Along the way, we also show that functions defined on very small domains or that are very unbalanced can display at most a quadratic quantum speedup.

Thesis Supervisor: Scott J. Aaronson
Title: Visiting Associate Professor
Department of Electrical Engineering and Computer Science

Acknowledgments

Let me start by thanking my supervisor, Scott Aaronson. Scott's passion for research is contagious, and he would always come up with interesting new problems to work on. Scott has a habit of responding to emails within minutes, no matter the hour or time zone he is in, and sometimes these emails would contain pages of detailed research insights. He showed great dedication as a supervisor, and would always be available to chat and share his ideas and advice.

I would also like to thank my committee members, Aram Harrow and Ryan Williams for reading the thesis and serving on my committee.

Special thanks go to my close collaborator Robin Kothari. I have learned a lot from him, and working together helped produce some of the research I am most proud of. Other collaborators I have learned from include Anurag Anshu, Aleksandrs Belovs, Adam Bouland, Ankit Garg, Pooya Hatami, Mika Göös, Rahul Jain, Troy Lee, Miklos Santha, Or Sattath, and Avishay Tal. In addition, I thank Hardik Bansal, Mark Bun, Iordanis Kerenidis, Frédéric Magniez, Ashwin Nayak, Ansis Rosmanis, John Watrous, and Ronald de Wolf for providing useful feedback or comments on some of the papers included in this thesis.

I thank all my friends at MIT and my former roommates for their support and friendship; they helped make my time at MIT fun. I thank my parents for always being there for me throughout my life. Finally, I thank my wife for her constant love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the central goals of complexity theory is to understand the power of various computational resources. When do resources such as nondeterminism or randomness help? What can be done in the presence of advice or of an untrusted prover that cannot be done otherwise? Which intractable problems turn tractable with the aid of a quantum computer?

Many of the most famous conjectures in all of computer science, such as the P vs. NP problem or the P vs. BPP problem, arise as manifestations of these questions as they apply to Turing machines. In particular, the P vs. NP problem asks whether polynomial-time Turing machines benefit from non-determinism, while the P vs. BPP problem asks if they benefit from access to randomness.

One of the most exciting questions in this direction is the question of BPP versus BQP: can quantum computers solve problems that ordinary computers cannot? This question is of immediate interest because of the possibility that quantum computers will be built in the next few decades. Is building such computers a worthwhile endeavor, or can they be simulated efficiently by computers we have today? The fact that we already know some highly nontrivial quantum algorithms (such as for factoring [78]) only increases the importance of this question: if it turns out that BQP = BPP, this would mean that a polynomial-time classical algorithm for factoring integers exists, letting us break many cryptographic protocols currently in use today.

Unfortunately, proving the non-existence of polynomial time algorithms for various problems has turned out to be an enormously difficult task. Showing that $\mathsf{P} \neq \mathsf{NP}$, as conjectured by most researchers, is beyond our current abilities. Settling the P vs. BPP vs. BQP problem hits similar barriers [49]. The inability to answer even the most basic complexity questions in the Turing machine model limits our ability to conduct more fine-tuned analysis into when and by how much such resources help.

Query complexity attempts to circumvent this difficulty by switching the model: instead of studying the questions for Turing machines whose computational cost is the number of operations used, in the query complexity setting the machine has access to a blackbox, and its only cost is the number of queries made to the blackbox; the number of operations is no longer restricted. In this new setting, a "fast algorithm" means an algorithm that makes few queries to the blackbox, whether or not it takes a lot of post-processing time to return the answer.

Analogues of the classes P, NP, BPP, and BQP can be formalized in the query complexity setting, and settling the pairwise equality questions becomes relatively easy; this opens the door to asking more fine-tuned questions such as exactly by how much and in what contexts

we should expect randomness, non-determinism, or quantum computation to help.

In this thesis, we focus our study primarily on randomized and quantum computation in the query complexity setting. Quantum query complexity is a particularly appealing model to study because quantum query algorithms have a close correspondence to quantum algorithms in the Turing machine model: famous algorithms such as Shor's algorithm and Grover's algorithm can be formalized and studied in the query world, and in some sense this is their natural setting.

1.1 Query Complexity

We now go over the basics of query complexity. More detailed query complexity background can be found in Chapter 2. Alternatively, the reader is encouraged to check out the survey by Buhrman and de Wolf [26][1].

In query complexity, there is a known Boolean function $f : \{0,1\}^n \to \{0,1\}$, and an unknown string $x \in \{0,1\}^n$. The goal is to compute $f(x)$ by looking at as few of the bits of x as possible. That is, we're allowed to make adaptive queries to the bits of x, and the goal is to minimize the number of queries made before the value of $f(x)$ is determined.

The worst-case number of such queries required by the best possible algorithm is called the deterministic query complexity of f, denoted $\mathrm{D}(f)$. We can also let the algorithm to make randomized queries, and allow it to err with some constant (small) probability; the number of queries required in this randomized model is denoted $\mathrm{R}(f)$.

Finally, we can also consider quantum query algorithms. Like randomized algorithms, we allow quantum algorithms to err with some small probability. The difference is that instead of simply making randomized queries to the bits of f, we now allow the algorithm to query in superposition: the algorithm can "query" the superposition $\sum_k \alpha_k |i_k\rangle|b_k\rangle$, where $i_k \in \{1, 2, \ldots, n\}$ and $b_k \in \{0,1\}$, and the query will return the state $\sum_k \alpha_k |i_k\rangle|b_k \oplus x_{i_k}\rangle$.

Interestingly, these measures are all polynomially related to each other for functions $f : \{0,1\}^n \to \{0,1\}$: it was shown in [17] that $\mathrm{Q}(f) \leq \mathrm{R}(f) \leq \mathrm{D}(f) \leq O(\mathrm{Q}(f)^6)$. The best separations between the measures remain open.

In contrast, if we consider functions defined on a *subset* of $\{0,1\}^n$ (and only require the query algorithm to succeed on the subset), we can get exponential separations between these measures. There is a query function f with $\mathrm{R}(f) = O(1)$ and $\mathrm{D}(f) = \Omega(n)$, and another query function with $\mathrm{Q}(f) = O(1)$ and $\mathrm{R}(f) = \tilde{\Omega}(\sqrt{n})$. Achieving the maximum 1 vs. n separation for $\mathrm{R}(f)$ vs. $Q(f)$ is impossible [2], and achieving a $\log n$ vs. n gap is open.

Some of the problems studied in query complexity include finding the largest possible separations for total and partial functions, examining the query complexities of various natural problems, and exploring how the query complexity changes when the function f is modified in various ways.

Examples

Some examples of Boolean functions and their query complexities might be helpful. A simple example is the parity function, $\textsc{Parity} : \{0,1\}^n \to \{0,1\}$, which maps a string of length n to 0 if its Hamming weight is even and to 1 otherwise. Since one must read all bits of a string to determine its parity, we have $\mathrm{D}(\textsc{Parity}) = n$. It turns out that $\mathrm{R}(\textsc{Parity})$ and $\mathrm{Q}(\textsc{Parity})$ are also $\Theta(n)$.

[1] In this reference, the term 'decision tree complexity' is used to refer to query complexity.

An easier Boolean function is the index function, $\text{IND} : \{0,1\}^{k+2^k} \to \{0,1\}$, which uses the first k bits of a string to index a position of the remaining 2^k bits of the string, and outputs this bit. The index function can be computed using only $k+1$ deterministic queries, which is logarithmic in the input size $n = k + 2^k$.

The previous two functions were total functions, defined on all Boolean strings of the right length. We can also analyze the query complexity of partial functions; for instance, let f be the function satisfying $f(x) = 0$ if the Hamming weight of x is at most $n/3$, and $f(x) = 1$ if the Hamming weight of x is at least $2n/3$. If the Hamming weight of x lies between $n/3$ and $2n/3$, we leave f undefined; we simply promise that the input will never have this form. Then f is a partial function, and it is not hard to see that $\text{D}(f) = \Omega(n)$ and $\text{R}(f) = O(1)$.

1.2 Overview of Results

We briefly describe the organization of this thesis.

In Chapter 2, we give detailed background on query complexity; we rigorously define query complexity measures such as $\text{R}(f)$ and $\text{Q}(f)$, discuss some common lower bound techniques, and prove some of the fundamental results, such as the polynomial relationship between query measures of total functions. This chapter is readable on its own as an introduction to the subject.

In Chapter 3, we study the question of the largest possible quantum speedup over classical (randomized) algorithms for total functions. We introduce the cheat sheet method for turning partial function query separations into total function separations, and use it to provide a power 2.5 separation between $\text{R}(f)$ and $\text{Q}(f)$ for a family of total functions. This chapter is based on work that appeared in [19] and [4], the latter of which is joint work with Scott Aaronson and Robin Kothari.

In Chapter 4, we introduce a technique for lower bounding the randomized query complexity. This technique is captured by a measure we call the randomized sabotage complexity, $\text{RS}(f)$. We show how this measure can be used to study the behavior of randomized query complexity of composed functions. We also give an application to communication complexity, showing a lifting theorem reduction. This chapter is based on work that appeared in [20], which is joint work with Robin Kothari.

In Chapter 5, we introduce the quantum version of sabotage complexity. We examine its relationship to the non-negative quantum adversary bound and to the complexity class QSZK, and we use it to show a new power 5 relationship between zero-error and bounded-error quantum algorithms. This chapter is based on unpublished work with Robin Kothari.

In Chapter 6, we study the question of the type of structure necessary for an exponential quantum speedup to be possible. More precisely, we study the total functions $f : \{0,1\}^n \to \{0,1\}$ for which there exists some promise $P \subseteq \{0,1\}^n$ such that $f|_P$ has an exponential quantum speedup over randomized algorithms. We show that such "sculptable" functions are characterized by having a large number of inputs with large certificate complexity, and we define an h-index measure which captures sculptability. We also obtain some interesting side results, including a nearly-quadratic relationship between $\text{Q}(f)$ and $\text{D}(f)$ for partial functions whose domain is small. This chapter is based on work that appeared in [3], which is joint work with Scott Aaronson.

Chapter 2

Query Complexity Preliminaries

In this chapter, we give an introduction to query complexity and a summary of the current state of knowledge. This chapter is meant to be readable on its own, independently of the other chapters, so that it can be used as an introductory reference for the field. For a published survey of query complexity (which is also known as decision tree complexity), see Buhrman and de Wolf [26].

2.1 The Basics

The primary objects of study in query complexity are Boolean functions and algorithms for computing them. A Boolean function is a function $f : \{0,1\}^n \to \{0,1\}$, where n is a positive integer; we are often also interested in partial Boolean functions, which are defined on a subset of $\{0,1\}^n$. That is, partial Boolean functions are functions $f : P \to \{0,1\}$ where $P \subseteq \{0,1\}^n$. We use $\mathrm{Dom}(f)$ to refer to P, the domain of f.

We say that an algorithm A computes a (possibly partial) Boolean function f if it takes as input a string $x \in \mathrm{Dom}(f)$ and returns $f(x)$. The defining feature of query complexity is that we care not about how much *time* A takes to do this computation, but about how many bits of the input x A looks at. In other words, we are interested in algorithms that output $f(x)$ without even reading the entire input, and indeed we wish to minimize the amount of the input that is read.

One common way to think about query complexity is to imagine that the input string x is actually a blackbox function (not to be confused with the function f, which is assumed to be fixed and known in advance). In other words, we can think of x as the function $x : [n] \to \{0,1\}$ defined by $x(i) := x_i$ for all $i \in [n]$ (here $[n]$ denotes the set $\{1,2,\ldots,n\}$). In this framework, the algorithm A receives the blackbox function x as input, and calls it some number of times before returning the output $f(x)$. We wish to minimize the number of calls A makes to the blackbox, but we do not care how much time A takes to return the answer.

We now formalize exactly what we mean by "algorithm". Importantly, we will not need to make any reference to Turing machines in order to do so, demonstrating the relative tractability of the query complexity setting.

2.1.1 Deterministic Query Complexity and Decision Trees

We start with deterministic algorithms. Recall that there is a known (possibly partial) function $f : \{0,1\}^n \to \{0,1\}$ and an unknown input $x \in \mathrm{Dom}(f)$, and our goal is to

compute $f(x)$ using *adaptive queries* to the bits of x.

To represent such an algorithm, we only need to keep track of the sequence of queries it makes to the input. That is, the first query is just a number i in $[n]$; for the second query, there are two possibilities: one for the case where the first queried bit x_i was 0, and one for the case where x_i was 1. The entire algorithm can therefore be represented as a binary tree whose internal nodes are labeled by $[n]$ and whose edges are labeled by $\{0,1\}$. The output of the algorithm will be stored in the leaves, which are also labeled by $\{0,1\}$.

Definition 2.1. *Fix a positive integer n. A* deterministic query algorithm *or* decision tree *on n bits is either*

- *A leaf, which is an element of $\{0,1\}$, or*
- *A triple (i, D_0, D_1) where $i \in [n]$ is the label of the root, D_0 and D_1 are decision trees on n bits, and neither D_0 nor D_1 have any nodes labeled by i.*

In this definition, D_0 and D_1 represent the algorithms to be executed in the case the first query x_i returns 0 or 1, respectively. They are forbidden from including i as a label of an internal node in order to prevent the algorithm from making the same query twice, and also in order to ensure the above definition allows finite trees only. We can now define the evaluation of a deterministic algorithm D on an input $x \in \{0,1\}^n$, as well as the number of queries D makes on x (which we refer to as the height of x in D).

Definition 2.2. *Let D be a decision tree on n bits, and let $x \in \{0,1\}^n$. The value of D when run on x, denoted by $D(x)$, is*

- *D itself if $D \in \{0,1\}$ is a leaf, or*
- *recursively, the value $D_{x_i}(x)$ if $D = (i, D_0, D_1)$.*

The height *of x in D, denoted $h(D,x)$, is the number of times the recursion step above is applied in the evaluation of D on x. The height of the decision tree D, denoted $h(D)$, is the maximum value of $h(D,x)$ over all $x \in \{0,1\}^n$.*

Next, we define what it means for an algorithm D to compute a function f.

Definition 2.3. *Let D be a decision tree on n bits, and let f be a (possibly partial) Boolean function on n bits. We say D computes f if $D(x) = f(x)$ for all $x \in \mathrm{Dom}(f)$.*

Finally, we define the deterministic query complexity of a Boolean function, which is the minimum number of queries required (in the worst case over input x) by the best possible algorithm computing f.

Definition 2.4. *Let f be a (possibly partial) Boolean function on n bits. The* deterministic query complexity *of f, denoted $\mathrm{D}(f)$, is the minimum value of $h(D)$ over all decision trees D that compute f.*

We observe that for all Boolean functions f on n bits, $\mathrm{D}(f)$ is an integer between 0 and n, and it is 0 if and only if f is constant. In the literature, functions with $\mathrm{D}(f) = n$ are sometimes called *evasive*.

Examples

Some common examples of total Boolean functions include PARITY_n and OR_n. It's clear that to determine the parity of n bits, one must look at all n of the bits; this means that $\text{D}(\text{PARITY}_n) = n$. For OR on n bits, we only need to examine the bits until a 1 is found. However, when the input string is $x = 0^n$, there is no way to be sure what $\text{OR}_n(x)$ is without reading all the bits of x; for this reason, $\text{D}(\text{OR}_n) = n$. Hence OR_n and PARITY_n are both evasive.

What are some functions for which $\text{D}(f)$ is smaller than n? One simple example is the dictator function defined by $f(x) = x_1$; for this function, $\text{D}(f) = 1$. A less trivial example is the index function: it is defined on $n = k + 2^k$ bits, and $f(x)$ is the bit $x_{\ell+k+1}$, where ℓ is the number whose binary representation is the string $x_1 x_2 \dots x_k$. That is, the function uses the first k bits to index a position in the remaining 2^k bits, and outputs the bit in that position. It is clear that we have $\text{D}(f) = k + 1 = \Theta(\log n)$ for the index function.

2.1.2 Randomized Query Complexity

In a similar manner to the deterministic query complexity defined above, we can define the notion of randomized query complexity, in which we allow randomized algorithms for computing f. For this, we need a formalization of a randomized query algorithm, or randomized decision tree. One way to do this is to add to the decision tree a "probabilistic layer" before each ordinary layer of nodes. In other words, there will be two types of nodes: one type specifies the index of the input to query, and the other type represents a probability distribution over which node to go to next. The edges coming out of the probability nodes will be labeled by non-negative real numbers that sum up to 1, and these edges will all lead to query nodes. The edges coming out of the query nodes will be the usual pair of edges labeled by 0 and 1, and they will lead to probability nodes.

While the above formulation of randomized decision trees is natural – it gives the decision tree the ability to make random decisions after each query – it is a little complicated. Luckily, it can be simplified: instead of considering such a randomized decision tree, we can consider a probability distribution over deterministic decision trees. It is not hard to see that there is a close correspondence between the two concepts: a probability distribution over deterministic decision trees simply corresponds to making the randomized internal choices "in advance". For this reason, we will simply define a randomized decision tree to be a probability distribution over deterministic decision trees.

Definition 2.5. *Fix a positive integer n. A* randomized decision tree *or* randomized query algorithm *on n bits is a probability distribution over deterministic decision trees on n bits.*

For deterministic query algorithms, we introduced the height of the decision tree, which corresponded to the worst-case number of queries the algorithm could make. For a randomized algorithm R, there are two natural notions of worst-case number of queries, or "height" of the randomized decision tree. The first is the overall worst case, corresponding to the maximum over both the input x and the choice of randomness; that is, getting the worst possible input and also getting extremely unlucky. The second is the expected worst-case, corresponding to the maximum over inputs x of the expected running time of R on x; that is, getting the worst possible input but averaging out the number of queries on that input for different choices of randomness. We denote the former by $h(R)$ and the latter by $\overline{h}(R)$.

Definition 2.6. *Let R be a randomized decision tree on n bits, and let $x \in \{0,1\}^n$. The worst-case height of R on x, denoted $h(R,x)$, is the maximum value of $h(D,x)$ over all decision trees D in the support of R. The expected height of R on x, denoted $\overline{h}(R,x)$, is the expected value of $h(D,x)$ when D is sampled from R. We also define the worst-case and expected height of R, denoted $h(R)$ and $\overline{h}(R)$, as the maximum over $x \in \{0,1\}^n$ of $h(R,x)$ and $\overline{h}(R,x)$, respectively.*

Next, we need to define what it means for a randomized decision tree to compute a Boolean function f. We start by defining the evaluation of a randomized decision tree R on an input x, as follows.

Definition 2.7. *Let R be a randomized decision tree on n bits, and let $x \in \{0,1\}^n$. Then the value of R when run on x, denoted $R(x)$, is the random variable $D(x)$ when D is sampled from R.*

Note that $R(x)$ is a random variable with support $\{0,1\}$; that is, running the algorithm R on x gives a probability distribution over $\{0,1\}$ answers. Therefore, the notion of computing f will depend on the tolerated level of error.

Definition 2.8. *Let R be a randomized decision tree on n bits, and let f be a (possibly partial) Boolean function on n bits. For $\epsilon \in [0,1/2)$, We say that R* computes *f* with error *ϵ if $\Pr[R(x) = f(x)] \geq 1 - \epsilon$ for all $x \in \mathrm{Dom}(f)$.*

Note that the randomized decision tree which is a uniform distribution over $\{0,1\}$ computes every function with error $1/2$, so taking $\epsilon \geq 1/2$ is meaningless. We can now define the randomized query complexity of f with error ϵ.

Definition 2.9. *Let f be a (possibly partial) Boolean function on n bits, and let $\epsilon \in (0,1/2)$. The* randomized query complexity of f with error ϵ, *denoted $\mathrm{R}_\epsilon(f)$, is the minimum value of $h(R)$ over all randomized decision trees R computing f with error ϵ. When $\epsilon = 1/3$, we denote this by $\mathrm{R}(f)$ and refer to the measure as simply the randomized query complexity of f.*

We note that the randomized query complexity of f is sometimes referred to as two-sided randomized query complexity and denoted $\mathrm{R}_2(f)$ in the literature. We also note that $\mathrm{R}_0(f)$ means something different from the above definition with $\epsilon = 0$.

In the following section, we clean up a few loose ends regarding this definition, such as the arbitrariness of the value $1/3$ and the use of $h(R)$ rather than $\overline{h}(R)$ as the cost of a randomized query algorithm.

Examples

The function $\textsc{Parity}_n$ has maximal randomized query complexity, $\mathrm{R}(\textsc{Parity}_n) = n$. On the other hand, for the function OR_n one can save a constant factor by using the fact that the randomized algorithm is allowed to err: we have $\mathrm{R}(\mathrm{OR}_n) \leq \lceil 2n/3 \rceil$, though $\mathrm{R}(\mathrm{OR}_n) = \Omega(n)$.

We can construct functions that separate randomized query complexity from deterministic query complexity. This is easiest for partial functions: consider the function f defined only strings of Hamming weight $n/3$ or $2n/3$, which returns 0 if the Hamming weight is $n/3$ and 1 otherwise (this function is the majority function with the promise that the input has a certain Hamming weight). A randomized query algorithm for f can just pick a bit at

random and output it, so $\mathrm{R}(f) = 1$. On the other hand, a deterministic query algorithm must, in the worst case, query $n/3 + 1$ bits before knowing the value of $f(x)$ with certainty; thus $\mathrm{D}(f) = \Theta(n)$.

Separating randomized and deterministic query complexities for total functions is harder, and we leave it for later.

2.1.3 House Cleaning for Randomized Algorithms

There are a few seemingly arbitrary choices in the definition of $\mathrm{R}(f)$, which we now clarify. First of all, why choose $\epsilon = 1/3$? The following theorem shows that the value of ϵ doesn't matter, so long as it is bounded between 0 and $1/2$ as so long as we don't care about constant factors.

Lemma 2.10 (Amplification). *If f is a Boolean function and R is a randomized algorithm for f with error $\epsilon < 1/2$, repeating R several times and taking the majority vote of the outcomes decreases the error. To reach error $\epsilon' > 0$, it suffices to repeat the algorithm $\frac{2\ln(1/\epsilon')}{(1-2\epsilon)^2}$ times. In particular, if $0 < \epsilon' < \epsilon < 1/2$, we have*

$$\mathrm{R}_\epsilon(f) \leq \mathrm{R}_{\epsilon'}(f) \leq \frac{2\ln(1/\epsilon')}{(1-2\epsilon)^2}\,\mathrm{R}_\epsilon(f).$$

This lemma implies that $\mathrm{R}_\epsilon(f) = \Theta(\mathrm{R}_{\epsilon'}(f))$ for all ϵ and ϵ' that are constants in $(0, 1/2)$; the choice of $1/3$ is therefore just a convention and doesn't matter except up to constant factors. When $\epsilon \geq 1/2$, the query complexity of every function becomes 0. Finally, when $\epsilon = 0$, every deterministic decision tree in the domain of a randomized decision tree computing f also computes f, so the randomized query complexity equals $\mathrm{D}(f)$.

Next, what happens if we define the cost of a randomized algorithm by $\overline{h}(R)$ instead of $h(R)$?

Definition 2.11. *Let f be a Boolean function, and let $\epsilon \in [0, 1/2)$. The* expected randomized query complexity *of f with error ϵ, denoted $\overline{\mathrm{R}}_\epsilon(f)$, is the infimum of $\overline{h}(R)$ over all randomized decision trees R that compute f with error ϵ. When $\epsilon = 1/3$, we omit it and write $\overline{\mathrm{R}}(f)$.*

Observe that while $h(R)$ is always an integer, $\overline{h}(R)$ need only be a real number, explaining the use of infimum instead of minimum. However, note that the set of all randomized decision trees on n bits is simply the set of probability distributions over a finite domain (the deterministic decision trees on n bits), which is compact. Note also that $\overline{h}$ is a continuous function in the probabilities inside R. Since continuous functions attain their minimum over compact sets, the infimum in the definition of $\overline{\mathrm{R}}_\epsilon(f)$ is always attained by a single randomized decision tree.

Lemma 2.10 tells us that $\overline{\mathrm{R}}_\epsilon(f) = \Theta(\overline{\mathrm{R}}_{\epsilon'}(f))$ for all $\epsilon, \epsilon' \in (0, 1/2)$. How does $\overline{\mathrm{R}}(f)$ relate to $\mathrm{R}(f)$? Since $\overline{h}(R) \leq h(R)$ for all randomized decision trees R, we have $\overline{\mathrm{R}}(f) \leq \mathrm{R}(f)$. For the other direction, we use the following lemma.

Lemma 2.12 (Markov's Inequality). *Let R be a randomized decision tree on n bits, and let $\delta \in (0, 1)$. On any input $x \in \{0,1\}^n$, the probability that $h(D, x) \leq \lfloor \overline{h}(R, x)/\delta \rfloor$ when D is sampled from R is at least $1 - \delta$.*

This lemma lets us switch from an expected cost algorithm to a worst case algorithm while losing an additive constant in the error and a multiplicative constant in the query

complexity: all we need to do is to "cut off" the deterministic decision trees in the support of R that are too tall, replacing the paths that are too long with a random guess. Doing this and combining with amplification yields the following result.

Lemma 2.13. *If f is a (possibly partial) Boolean function, then for all $\epsilon \in (0, \frac{1}{2})$, we have $\mathrm{R}_\epsilon(f) \leq 14\frac{\ln(1/\epsilon)}{(1-2\epsilon)^2}\overline{\mathrm{R}}_\epsilon(f)$. When $\epsilon = \frac{1}{3}$, we can improve this to $\mathrm{R}(f) \leq 10\overline{\mathrm{R}}(f)$.*

In other words, all definitions of randomized query complexity we've seen so far are equivalent up to constant factors.

What happens when we set $\epsilon = 0$ for $\overline{\mathrm{R}}_\epsilon(f)$? In this case, we actually get a new measure, and not simply $\mathrm{D}(f)$ like before.

Definition 2.14. *Let f be a (possibly partial) Boolean function. The* zero-error randomized query complexity of f, *denoted $\mathrm{R}_0(f)$, is defined to be $\overline{\mathrm{R}}_0(f)$.*

The zero-error randomized query complexity of f is larger than or equal to $\overline{\mathrm{R}}_\epsilon(f)$ for all $\epsilon \in (0, 1/2)$ and all f, but smaller than or equal to $\mathrm{D}(f)$. We also have $\mathrm{R}(f) \leq (3/2)\,\mathrm{R}_0(f)$, which follows from applying Lemma 2.12 by cutting off the zero-error algorithm after $3/2$ times the expected number of steps and guessing randomly afterwards.

We can also define one-sided randomized query complexity.

Definition 2.15. *Let f be a (possibly partial) Boolean function on n bits, and let R be a randomized decision tree on n bits. Let $\epsilon \in (0, 1)$. We say that R* computes f with one-sided error ϵ on the 1 side *if $\Pr[R(x) = f(x)] = 1$ for all $x \in f^{-1}(1)$ and $\Pr[R(x) = f(x)] \geq 1 - \epsilon$ for all $x \in f^{-1}(0)$. We say R computes f with one-sided error on the 0 side if these probabilities are flipped.*

We define $\mathrm{R}^{(0)}_{1,\epsilon}(f)$ to be the minimum of $h(R)$ over randomized decision trees R that compute f with one-sided error on the 0 side, and we define $\mathrm{R}^{(1)}_{1,\epsilon}(f)$ similarly for the 1 side. We define $\mathrm{R}_{1,\epsilon}(f)$ as the minimum of these two measures. Finally, if $\epsilon = 1/2$, we omit it from these measures, so that $\mathrm{R}_1(f) = \mathrm{R}_{1,1/2}(f)$.

An amplification theorem similar to Lemma 2.10 can be shown for one-sided randomized algorithms, demonstrating that ϵ does not matter up to constant factors when it is inside $(0, 1)$. When $\epsilon = 0$, one-sided error randomized query complexity simply becomes $\mathrm{D}(f)$, and when $\epsilon = 1$ it is always zero. The expectation version $\overline{\mathrm{R}}_1(f)$ can be defined in the natural way, and it will equal $\mathrm{R}_1(f)$ up to constant factors for all $\epsilon \in (0, 1)$. In addition, $\overline{\mathrm{R}}_{1,\epsilon}(f)$ equals $\mathrm{R}_0(f)$ when $\epsilon = 0$.

It is not hard to observe that $\mathrm{R}(f) \leq \mathrm{R}_1(f)$, since we can turn a one-sided error algorithm with error $1/2$ on one side and 0 on the other into a two-sided error algorithm with error $1/3$ on both sides by simply flipping the output with probability $1/3$ on one of the outputs. Using Lemma 2.12, it is not hard to show that $\mathrm{R}_1(f) \leq 2\,\mathrm{R}_0(f)$. We also have $\mathrm{R}_1(f) \leq \mathrm{D}(f)$.

Finally, we note that $\max\{\mathrm{R}^{(0)}_1(f), \mathrm{R}^{(1)}_1(f)\}$ equals $\mathrm{R}_0(f)$ up to constant factors. The direction $\max\{\mathrm{R}^{(0)}_1(f), \mathrm{R}^{(1)}_1(f)\} \leq 2\,\mathrm{R}_0(f)$ follows from Markov's inequality. For the other direction, note that if we have a one-sided error algorithm R_0 that never errs when it outputs 0 and a one-sided error algorithm R_1 that never errs when it outputs 1, we can simply run both, and repeat until one of the two outputs the guaranteed-correct output. If $\epsilon = 1/2$, we only need to repeat these algorithms twice each on expectation, as we show in Appendix A. Thus $\mathrm{R}_0(f) \leq 4\max\{\mathrm{R}^{(0)}_1(f), \mathrm{R}^{(1)}_1(f)\}$.

In sum, we defined three notions of randomized query complexity that are *not* equivalent up to constant factors: $\mathrm{R}_2(f)$, $\mathrm{R}_1(f)$, and $\mathrm{R}_0(f)$. They correspond to BPP algorithms,

RP$\cup$coRP algorithms, and ZPP algorithms, respectively. They satisfy the relations $\mathrm{R}_2(f) \leq \mathrm{R}_1(f) \leq \mathrm{D}(f)$, and $\mathrm{R}_0(f)$ lies between $\mathrm{R}_1(f)$ and $\mathrm{D}(f)$ up to constant factors. The measure $\mathrm{R}_2(f)$ is more commonly denoted $\mathrm{R}(f)$, and is the default definition of randomized query complexity.

2.1.4 Quantum Query Complexity

Next, we define the notion of quantum query complexity, attempting to model quantum algorithms that have query access to the bits of an input string x. Quantum query complexity no longer looks like a decision tree. To motivate the definition, we will start by redefining deterministic and randomized query complexities in a different way.

We can view a deterministic query algorithm as a tape of memory that is manipulated by two players: the algorithm and the oracle. The tape will be split into four registers: the first will be called the general memory register (of arbitrary size), the second will be the query index register (which stores a number in $[n]$), the third will be the query answer register (which stores a bit in $\{0,1\}$), and the fourth will be the final output register (which also stores a bit in $\{0,1\}$). The two players alternate turns, with the algorithm player going first. In his turn, the algorithm player can manipulate all registers in any way (though the manipulation cannot depend on the input x). On the oracle's turn, she manipulates the tape in a very specific way: by reading $i \in [n]$ from the query index register, and writing x_i to the query answer register.

After T alterations between the players, the algorithm player takes one final turn, after which the final output of the algorithm is defined to be the contents of the output register (that is, the algorithm player should write his final answer for $f(x)$ in that designated register). This defines a T-query deterministic algorithm.

It is not hard to see that this definition is equivalent to the decision tree definition of algorithms, in that both allow the same types of computation for the same cost. Indeed, a decision tree can be simulated in this model by storing the whole tree plus the current position inside the tree in the general memory register, and then moving in the tree (and changing the query register to match the label of the current node) in each step.

Similarly, a randomized algorithm can be defined in this register model by allowing the algorithm player to make randomized transformations on the four registers. That way, the state of the tape in each step will be a probability distribution over deterministic tape contents.

We're now ready to define a quantum query algorithm. The idea is to simply take the register definition of query complexity, and allow unitary transformation to be applied to the tape. That is, each possible deterministic content of the tape will constitute a basis vector in a Hilbert space; the contents of the tape at each point of time will be represented by a unit vector in that Hilbert space (that is, a quantum superposition over deterministic tape contents); and the manipulations the players make to the tape will be unitary transformations.

One catch is that a unitary transformation is always reversible, so we must be careful to make sure the action of the oracle player is defined appropriately. Concretely, we assume deterministic contents of the tape always have the form $|A\rangle|i\rangle|b\rangle|o\rangle$, where A (the general memory register) is arbitrarily large, i is an index in $[n]$, and b and o are bits in $\{0,1\}$. We define the oracle unitary U^x (which depends on the input x) as the map $|A\rangle|i\rangle|b\rangle|o\rangle \rightarrow |A\rangle|i\rangle|b \oplus x_i\rangle|o\rangle$ extended linearly. In particular, when $b = 0$, this transformation simply writes x_i in the query answer register, where i is specified by the query index register.

A T-query quantum algorithm will then be a sequence of $T+1$ unitary matrices, $U_0, U_1, \ldots, U_T$, defining the action of the algorithm player. The output of the algorithm will be the result of measuring output register after applying

$$U_T U^x U_{T-1} U^x \ldots U^x U_1 U^x U_0 |A_0\rangle|1\rangle|0\rangle|0\rangle,$$

where A_0 is an arbitrary fixed state (independent of x). Note that the starting state $|A_0\rangle|1\rangle|0\rangle|0\rangle$ can be replaced with any other fixed state independent of x; the choice doesn't matter, because U_0 can map it to another state anyway. The size of the Hilbert space for the first register can be arbitrarily large.

Recall that measurement outcomes are probabilistic, with the probability of an outcome determined the square of the absolute value of its amplitude. In other words, if the final state of the algorithm has the form

$$\sum_{k \in K_0} \alpha_k |A_k\rangle|i_k\rangle|b_k\rangle|0\rangle + \sum_{k \in K_1} \alpha_k |A_k\rangle|i_k\rangle|b_k\rangle|1\rangle,$$

the probability that the algorithm outputs 0 is $\sum_{k \in K_0} |\alpha_k|^2$ and the probability that the algorithm outputs 1 is $\sum_{k \in K_1} |\alpha_k|^2$ (note that since the final state is a unit vector, these probabilities sum to 1). If Q is a quantum query algorithm on n bits and $x \in \{0,1\}^n$, we let $Q(x)$ denote the random variable for the outcome of the algorithm when run on x; this is a random variable with support $\{0,1\}$. We can also define $h(Q)$ to be T, the number of queries made by the quantum algorithm, though this notation is rarely used.

Definition 2.16. *Let Q be a T-query quantum algorithm on n bits, as defined above. Let f be a (possibly partial) Boolean function on n bits, and let $\epsilon \in (0, 1/2)$. We say that Q* computes *f with error ϵ if $\Pr[Q(x) = f(x)] \geq 1 - \epsilon$ for all $x \in \mathrm{Dom}(f)$.*

We define the quantum query complexity of *f with error ϵ, denoted $\mathrm{Q}_\epsilon(f)$, as the minimum integer T such that there exists a T-query quantum algorithm that computes f with error ϵ. When $\epsilon = 1/3$, we omit it and write $\mathrm{Q}(f)$.*

We note that this definition effectively measures the worst-case running time of the quantum algorithm, not the expected running time; in other words, $\mathrm{Q}_\epsilon(f)$ is the quantum analogue of $\mathrm{R}_\epsilon(f)$, not of $\overline{\mathrm{R}}_\epsilon(f)$. This means that when we set $\epsilon = 0$ in this definition, we get an analogue of $\mathrm{D}(f)$; we denote this measure by $\mathrm{Q}_E(f)$ in order to reserve $\mathrm{Q}_0(f)$ for something more analogous to $\mathrm{R}_0(f)$.

Definition 2.17. *Let f be a (possibly partial) Boolean function. The* exact quantum query complexity of *f, denoted $\mathrm{Q}_E(f)$, is defined as in Definition 2.16 with ϵ set to 0.*

To argue that the choice of $\epsilon = 1/3$ does not matter, we need to discuss amplification for quantum algorithms. Lemma 2.10 tells us that we can reduce the error of any algorithm by repeating it and taking a majority vote; it remains to argue that if Q is a quantum query algorithm, there is another quantum query algorithm Q' that repeats the first algorithm k times and outputs the majority vote of the runs. This is not hard to see: Q' can use a larger general memory register than Q, and following each run of Q it can store the final state of the algorithm. Q' will then only need to compute the majority function on the k stored copies of the output register, and store this majority in the real output register. That way, when the real output register is measured, the result will be equivalent to measuring the k stored output registers and taking the majority of them.

The way we defined things, it is easy to see that quantum algorithms can simulate deterministic ones, so that $Q(f) \leq Q_E(f) \leq D(f)$. It is also true that quantum algorithms can simulate randomized algorithms, so we have $Q_\epsilon(f) \leq R_\epsilon(f)$ for all ϵ. The measures $Q_E(f)$ and $R(f)$ are in general incomparable.

We can define $Q_1(f)$ analogously to $R_1(f)$, and observe that amplification for it works the same way as amplification for one-sided randomized algorithms. One possible definition of $Q_0(f)$ will then simply be $\max\{Q_1^{(0)}(f), Q_1^{(1)}(f)\}$, since we saw that the randomized analogue of this measure is within a constant factor of $R_0(f)$.

Another possible way of defining $Q_0(f)$ is by changing the definition of a quantum algorithm to allow a "not yet done" output symbol, which we denote $*$. We say that such a quantum algorithm Q computes f with zero error and success probability $\delta \in (0,1)$ if for all $x \in \mathrm{Dom}(f)$, we have $\Pr[Q(x) = 1 - f(x)] = 0$ and $\Pr[Q(x) = *] \leq 1 - \delta$. In other words, we require Q to never err, but allow it to say "I don't know" with some fixed probability. When $\delta = 1$, this measure becomes $Q_E(f)$; when $\delta = 0$, this measure is 0; but for all values of δ in between, we get a new measure $Q_{0,\delta}(f)$, and we can amplify δ. It is not hard to show that for the randomized analogue of this measure, we have $R_{0,\delta}(f) = \Theta(R_0(f))$. We therefore define $Q_0(f)$ to be $Q_{0,1/2}(f)$.

In conclusion, we defined the quantum measures

$$Q(f) \leq Q_1(f) \leq Q_0(f) \leq Q_E(f)$$

without making reference to any "expected" number of queries made by a quantum algorithm. These measures are all smaller than the corresponding randomized measures, except for $Q_0(f)$, which is smaller than $2\,R_0(f)$ rather than $R_0(f)$ due to the difference in definitions.

2.1.5 Further Remarks

So far, we introduced eight query complexity measures, which satisfy

$$0 \leq R(f) \leq R_1(f) \lesssim R_0(f) \leq D(f) \leq n,$$

$$0 \leq Q(f) \leq Q_1(f) \leq Q_0(f) \leq Q_E(f) \leq n,$$

with the quantum measures all smaller than the corresponding classical measures. We note that comparisons involving $R_0(f)$ are often off by a constant factor, since we defined it using the expected height of a randomized decision tree[1].

Eight measures are a lot to keep track of, and in the next section we'll introduce a lot more. In fact, most of the "complexity zoo" can arguably be defined in query complexity, and the zoo currently has hundreds of defined complexity classes; there are too many to keep track of. In the spirit of simplicity, we note that the main truly important measures defined so far are $D(f)$, $R(f)$, $R_0(f)$, and $Q(f)$; the other four are considered somewhat more esoteric.

We make one final remark, which is that all measures defined so far easily generalize to non-Boolean functions. In particular, the domain of f can be a subset of Σ^n instead of $\{0,1\}^n$ for some alphabet Σ of constant size, and the codomain can similarly be a finite alphabet Σ'

[1]It might be tempting to switch to a different definition to avoid the constant factors, such as one that is analogous to $Q_0(f)$. However, we stick with our current definition, because it behaves better with respect to compositions and direct sum results. One might even argue that it's the other measures that should be changed by a constant factor, and that $R_0(f)$ is "correct".

instead of $\{0,1\}$; the definitions of decision trees and of quantum query algorithms generalize readily to capture this setting. The only thing to remember is that amplification of two-sided error algorithms always requires $\epsilon < 1/2$, despite the fact that for functions with non-Boolean outputs, larger error probabilities still make sense to talk about[2].

2.2 Relationships for Total Functions

One important direction for query complexity is the study of relationships between the measures for total functions. It turns out that all the measures we have defined so far are polynomially related for such functions: we have $\mathrm{D}(f) = O(\mathrm{Q}(f)^6)$ for all total f. In this section, we introduce the tools necessary to prove such relationships. We also discuss the known separations. For most pairs of measures, the known relationships and known separations do not match, so that the largest gap between the measures remains open.

2.2.1 Notation

We start by introducing some useful notation. First, for a bit $b \in \{0,1\}$, we use $\overline{b}$ to denote $1-b$. We now define a few standard terms.

Definition 2.18 (Block). *Fix $n \in \mathbb{N}$. A* block *is a subset of $[n]$. If $B \subseteq [n]$ is a block and $x \in \{0,1\}^n$, we use x^B to denote the string x with the bits in B flipped, that is, $(x^B)_i = x_i$ for $i \notin B$ and $(x^B)_i = \overline{x_i}$ for $i \in B$. We also use x_B to denote the string in $\{0,1\}^{|B|}$ consisting of the bits $x_i : i \in B$.*

A block is simply a set of positions in the input string, and the notation x^B allows for easy manipulations of a Boolean string. We can generalize blocks to non-Boolean alphabets (where the input string is in Σ^n), though it is a little more complicated: we need to specify not only a set of indices of the input string, but also how they are to be transformed. We define a block B to be a subset A of $[n]$ together with $|S|$ permutations $\sigma_1, \sigma_2, \ldots, \sigma_{|S|} \in S_\Sigma$. Then x^B will be defined by $(x^B)_i = x_i$ if $i \notin A$ and $\sigma_j(x_i)$ if i is the j^{th} element of A.

Definition 2.19 (Partial Assignment). *Let $n \in \mathbb{N}$. A* partial assignment *is an element of $\{0,1,*\}^n$. We identify each partial assignment p with the set $\{(i, p_i) : p_i \neq *\}$, allowing us to use notation like $|p|$ and $p \subseteq q$ for partial assignments p and q. Two partial assignments p and q are* consistent *if they agree on their non-$*$ entries. The* support $\mathrm{supp}(p)$ *of a partial assignment p is the block $\{i \in [n] : p_i \neq *\}$.*

A partial assignment represents partial knowledge of a Boolean string $x \in \{0,1\}^n$. Recall that an unknown input string x can be thought of as a blackbox function mapping i to x_i for $i \in [n]$; from this perspective, a partial assignment is simply a partial blackbox function, and its support is simply its domain.

We can also generalize partial assignments to non-Boolean inputs: when the input is in Σ^n for some finite alphabet Σ, a partial assignment is an element of $\Sigma \cup \{*\}$.

Next, we define various notions of a restriction of a Boolean function.

Definition 2.20 (Restriction). *Let f be a (possibly partial) Boolean function on n bits.*

[2]One can even generalize query complexity to *relations*, which have multiple valid outputs. In this setting, a query algorithm is said to compute a relation f if for any given input, the output of the algorithm is one of the valid outputs of the function with probability at least $1-\epsilon$. However, one should no longer assume amplification works in this setting unless further justified.

- *If $P \subseteq \mathrm{Dom}(f)$, the restriction of f to P, denoted by $f|_P$, is the function f with restricted domain.*
- *If $B \subseteq [n]$ is a block, the restriction of f to B, denoted f_B, is the function on $|B|$ bits mapping x_B to $f(x)$ for all $x \in \mathrm{Dom}(f)$. This is only well-defined for blocks B such that x_B determines the value of $f(x)$ for all $x \in \mathrm{Dom}(f)$.*
- *If $p \subseteq \{0,1,*\}^n$ is a partial assignment, the restriction of f to p, denoted f_p, is the function on $n-|p|$ bits that is induced by revealing the non-$*$ bits of p in the input string; that is, this is $(f|_P)_B$ where $P = \{x \in \mathrm{Dom}(f) : p \subseteq x\}$ and $B = \{i \in [n] : p_i = *\}$.*

The first type of restriction represents adding an additional promise on the input string, reducing the number of options for it. It makes the function easier to compute. The second type of restriction represents reducing the number of questions we can ask the oracle. This makes the function harder to compute. Finally, the third type of restriction represents being given some queries for free. This makes the function easier to compute.

To summarize, all of the query complexity measures M introduced so far satisfy the following properties:

- (Range) If f is constant, $M(f) = 0$. If f is not constant and defined on n bits, $M(f) \in [1, n]$.
- (Restriction to a promise) If $P \subseteq \mathrm{Dom}(f)$, we have $M(f|_P) \leq M(f)$.
- (Restriction of queries) If $B \subseteq [n]$ and f_B is well-defined, we have $M(f_B) \geq M(f)$.
- (Restriction to partial assignment) If $p \in \{0,1,*\}^n$ is a partial assignment, we have $M(f_p) \leq M(f)$.
- (Input alphabet change) If $B \subseteq [n]$ is a block and f' is the function defined by $f'(x^B) = f(x)$ for all $x \in \mathrm{Dom}(f)$, we have $M(f') = M(f)$.
- (Bit duplication) If $f^{(i)}$ is the partial function on $n+1$ bits where the þn+1 bit is promised to equal the þi bit, and $f^{(i)}(x)$ evaluates to f on the first n bits, then $M(f^{(i)}) = M(f)$.
- (Padding) If f' is the partial function on $n+1$ bits where the last bit is promised to be constant (either always 0 or always 1), and $f'(x)$ evaluates to f on the first n bits, then $M(f') = M(f)$.

Another property that many measures satisfy is $M(\bar{f}) = M(f)$, where $\bar{f}$ denotes the negation of the function, so $\bar{f}(x) = \overline{f(x)}$. Query complexity properties that satisfy this property are called *two-sided* measures.

2.2.2 Certificates

One of the most important tool in the study of query complexity is the notion of certificates.

Definition 2.21 (Certificate). *Fix a (possibly partial) Boolean function f. We say a partial assignment p is a* certificate *(with respect to f) if $f(x)$ is the same for all strings $x \in \mathrm{Dom}(f)$ satisfying $p \subseteq x$. If $f(x) = 0$ for such strings, we say p is a 0-certificate; otherwise, we say p is a 1-certificate.*

This definition easily extends to functions with non-Boolean inputs or outputs.

This defines certificates with respect to a function f, but we can also talk about certificates for a particular *input* x to f. We say p is a certificate for the string x if p is both (1) a certificate (with respect to f), and (2) consistent with x. Note that x is always a certificate for x. We can define a measure of query complexity based on the cost of certifying.

Definition 2.22 (Certificate Complexity). *Let f be a (possibly partial) Boolean function, and let $x \in \mathrm{Dom}(f)$. Let $\mathrm{C}_x(f)$ denote the size of the smallest certificate for x with respect to f. The* certificate complexity *of f is $\mathrm{C}(f) := \max_{x \in \mathrm{Dom}(f)} \mathrm{C}_x(f)$. We also define the one-sided measures $\mathrm{C}^{(0)}(f) := \max_{x \in f^{-1}(0)} \mathrm{C}_x(f)$ and $\mathrm{C}^{(1)}(f) := \max_{x \in f^{-1}(1)} \mathrm{C}_x(f)$.*

$\mathrm{C}(f)$ measures the worst-case cost of certifying the value of the input. This is the same as the worst-case cost of non-deterministically computing the function; in fact, $\mathrm{C}^{(1)}(f)$ can be thought of as a query analogue of NP.

A deterministic or even zero-error randomized algorithm must find a certificate for the input it is given, as we now show.

Lemma 2.23. *Let f be a (possibly partial) Boolean function. Then any zero-error randomized algorithm computing f can be made to output a certificate for its input, and in particular, $\mathrm{R}_0(f) \geq \mathrm{C}(f)$. Similarly, $\mathrm{R}_1^{(0)}(f) \geq \mathrm{C}^{(0)}(f)$ and $\mathrm{R}_1^{(1)}(f) \geq \mathrm{C}^{(1)}(f)$.*

Proof. Let R be the zero-error randomized algorithm whose worst-case expected running time is $\mathrm{R}_0(f)$. Then R is a distribution over randomized decision trees, all of which compute f. On the input $x \in \mathrm{Dom}(f)$, the algorithm will sample a deterministic decision tree D and apply it; let p be the partial assignment resulting from running D on x.

We claim that p is a certificate for x with respect to f. Indeed, suppose not. Then there would be some input $y \in \mathrm{Dom}(f)$ such that $p \subseteq y$ and $f(y) \neq f(x)$. If we now run D on y, it will follow the same path down the decision tree, since y agrees with x on all the relevant queries. This means y reaches the same leaf of D that x reached, so $D(y) = D(x)$. However, since $f(y) \neq f(x)$, this contradicts the assumption the D computes f. Hence p is a certificate, and the algorithm R can output it. Finally, since R outputs a certificate for x on each input x, in must make at least $\mathrm{C}_x(f)$ queries to x, regardless of the choice of randomness; hence $\overline{h}(R, x) \geq \mathrm{C}_x(f)$. Maximizing over all $x \in \mathrm{Dom}(f)$ gives $\overline{h}(R) \geq \mathrm{C}(f)$, so $\mathrm{R}_0(f) \geq \mathrm{C}(f)$.

The case of one-sided algorithms follows similarly, except that the algorithm need only find certificates on one type of inputs (say, 0-inputs). □

Query complexity and certificate complexity can be related in the other direction when f is a total function, as we now show. The following theorem is due to [42, 23, 84].

Theorem 2.24. *Let $f : \{0,1\}^n \to \{0,1\}$ be a total Boolean function. Then*

$$\mathrm{D}(f) \leq \mathrm{C}^{(0)}(f)\,\mathrm{C}^{(1)}(f) \leq \mathrm{C}(f)^2.$$

Later, we will prove a strengthened version of this theorem (Theorem 2.30). However, we present a stand-alone proof of this theorem here for clarity, since it is a little simpler.

Proof. We describe a deterministic query algorithm D that makes the desired number of queries to the input x. This algorithm keeps track of the partial assignment $p \in \{0, 1, *\}^n$ of queries it made so far; at the beginning, $p = *^n$. It then repeats the following.

1. Check if p is a certificate; if it is, the algorithm must know the value of $f(x)$, so output that value and halt.

2. If p is not a certificate, find some $y \in f^{-1}(0)$ consistent with p. Pick a certificate $c \subseteq y$ such that $|c| \leq \mathrm{C}^{(0)}(f)$.

3. Query the support of c, and update p with the answers.

In each iteration, the algorithm queries at most $\mathrm{C}^{(0)}(f)$ bits of the input x. We now show that the algorithm halts after at most $\mathrm{C}^{(1)}(f)$ iterations.

The key insight is that each 0-certificate and each 1-certificate must conflict, since no string in $\{0,1\}^n$ can be consistent with both. This means their supports must overlap in at least one index, and on that index the 0-certificate will claim a different value than the 1-certificate. Because of this, the query in step 3 of the algorithm must query at least one bit from every 1-certificate consistent with p.

It follows that after $\mathrm{C}^{(1)}(f)$ iterations, every 1-certificate of size at most $\mathrm{C}^{(1)}(f)$ must either be inconsistent with p or else be fully revealed in p. At this point, if p contains a 1-certificate, the algorithm will terminate; but if p does not contain a 1-certificate, it contradicts all 1-certificates of size at most $\mathrm{C}^{(1)}(f)$, and hence all 1-inputs to f; this means p must be a 0-certificate, so the algorithm terminates anyway. We conclude this algorithm makes at most $\mathrm{C}^{(0)}(f)\,\mathrm{C}^{(1)}(f)$ queries. □

Corollary 2.25. *Let f be a total Boolean function. Then*

$$\mathrm{D}(f) \leq \mathrm{R}_0(f)^2 \text{ and } \mathrm{D}(f) \leq \mathrm{R}_0(f)\,\mathrm{R}_1(f).$$

2.2.3 Sensitivity and Block Sensitivity

We now introduce two query complexity measures that are fundamental to the study of query complexity of total functions.

Definition 2.26 (Block sensitivity). *Fix a (possibly partial) Boolean function f and an input $x \in \mathrm{Dom}(f)$. We say a block $B \subseteq [n]$ is sensitive for x (with respect to f) if $x^B \in \mathrm{Dom}(f)$ and $f(x^B) \neq f(x)$.*

The block sensitivity *of x (with respect to f), denoted $\mathrm{bs}_x(f)$, is the maximum size of a set of disjoint sensitive blocks of x. The block sensitivity of the function f, denoted $\mathrm{bs}(f)$, is $\max_{x \in \mathrm{Dom}(f)} \mathrm{bs}_x(f)$. We also define $\mathrm{bs}^{(0)}(f) := \max_{x \in f^{-1}(0)} \mathrm{bs}_x(f)$ and $\mathrm{bs}^{(1)}(f) := \max_{x \in f^{-1}(1)} \mathrm{bs}_x(f)$.*

The block sensitivity of an input is the number of disjoint changes one can make to it to flip its value under the function f. Intuitively, when given an input x, one must make at least one query in each sensitive block B before determining the value of $f(x)$, as otherwise, one cannot know whether the input was actually x^B rather than x. Since the input has $\mathrm{bs}_x(f)$ disjoint sensitive blocks, we must make at least $\mathrm{bs}_x(f)$ queries. Block sensitivity will therefore be useful as a lower bound technique for query complexity in various models.

Lemma 2.27. *Let f be a (possibly partial) Boolean function, and let $x \in \mathrm{Dom}(f)$. Then $\mathrm{bs}_x(f) \leq \mathrm{C}_x(f)$. As a consequence, we have $\mathrm{bs}^{(0)}(f) \leq \mathrm{C}^{(0)}(f)$, $\mathrm{bs}^{(1)}(f) \leq \mathrm{C}^{(1)}(f)$, and $\mathrm{bs}(f) \leq \mathrm{C}(f)$.*

Proof. We note that every certificate c consistent with x must contain at least one bit from each sensitive block of x. Indeed, if B is a sensitive block of x, then $f(x^B) \neq f(x)$, which means c is inconsistent with x^B as it is a certificate for x. But since c is consistent with x, its disagreement with x^B must lie inside the block B.

Now, since there are $\mathrm{bs}_x(f)$ disjoint sensitive blocks for x and any certificate must intersect all of them, it follows that any certificate for x has size at least $\mathrm{bs}_x(f)$, so $\mathrm{C}_x(f) \geq \mathrm{bs}_x(f)$. □

We make use of the following property of randomized algorithms, which we state here in its most general form.

Lemma 2.28. *Let f be a (possibly partial) function, let $x \in \mathrm{Dom}(f)$ be an input, and let B be a sensitive block of x with respect to f.*

Then any randomized algorithm that solves f using at most T expected queries and with error at most ϵ must, when run on x, query a bit in B with probability at least $1 - 2\epsilon$.

Note that we do not require f to have Boolean inputs or outputs for this lemma to hold.

Proof. Let R be such a randomized algorithm. Let p be the probability that R queries an entry in B when it is run on x. Then with probability $1-p$, it does not query an entry in B when run on x, which means it also does not query an entry in B when run on x^B. Hence the probability that R does not query an entry in B when run on x^B is also $1 - p$.

Let q be the probability that R does not output $f(x)$ when run on x given that it doesn't query inside B. Let r be the probability that R does not output $f(x^B)$ when run on x given that it doesn't query such a difference. Since one of these events always happens, we have $q + r \geq 1$. Note that R errs with probability at least $(1-p)q$ when run on x and at least $(1-p)r$ when run on x^B. This means that $(1-p)q \leq \epsilon$ and $(1-p)r \leq \epsilon$. Summing these gives $1 - p \leq (1-p)(q+r) \leq 2\epsilon$, so $p \geq 1 - 2\epsilon$, as desired. □

Corollary 2.29. *Let f be a (possibly partial) Boolean function. Then* $\mathrm{R}(f) \geq \overline{R}(f) \geq \mathrm{bs}(f)/3$.

Proof. Let $x \in \mathrm{Dom}(f)$ be such that $\mathrm{bs}_x(f) = \mathrm{bs}(f)$, and let $B_1, B_2, \ldots, B_{\mathrm{bs}(f)}$ be disjoint sensitive blocks of x. Let R be a randomized algorithm for f. By Lemma 2.28, when R is run on x, it queries an entry inside B_i with probability at least $1 - 2(1/3) = 1/3$. Hence the expected number of bits R queries inside these blocks of x is at least $\mathrm{bs}(f)/3$, so $\overline{h}(R, x) \geq \mathrm{bs}(f)/3$. It follows that $\overline{R}(f) \geq \mathrm{bs}(f)/3$. □

We now prove some upper bounds in terms of $\mathrm{bs}(f)$ for total functions. We start with an improvement on Theorem 2.24. This result is implicit in [65], and proven explicitly in [17].

Theorem 2.30. *Let $f : \{0,1\}^n \to \{0,1\}$ be a total Boolean function. Then*

$$\mathrm{D}(f) \leq \mathrm{C}^{(0)}(f)\,\mathrm{bs}^{(1)}(f) \leq \mathrm{C}(f)\,\mathrm{bs}(f).$$

Proof. We use the same deterministic algorithm as in Theorem 2.24, but with a better analysis. In particular, we show that the number of iterations of the algorithm is upper bounded by $\mathrm{bs}^{(1)}(f)$ rather than just by $\mathrm{C}^{(1)}(f)$.

Let x be the input on which the algorithm runs for the largest number of iterations. The idea is to consider the state of the algorithm right before it makes its last batch of queries

on x. At that point, the partial assignment p is still not a certificate, meaning there is some 1-input y that is consistent with p. When the deterministic algorithm runs on y, it makes the same number of iterations as when it runs on x, but this time we know that $f(y) = 1$ (we had no control over $f(x)$).

Now, let each batch of queries made by the algorithm define a block. That is, if the algorithm ran for k iterations, we have k disjoint sets of bits that were queried, $A_1, A_2, \ldots, A_k$, one per iteration. We claim that each A_i contains a sub-block B_i which is sensitive. This is because in the i^{th} iteration of the algorithm, the positions queried were the support of some 0-certificate consistent with all the previous queries; hence, the bits we must change to make y contain that 0-certificate all lie inside A_i. If those bits are $B_i \subseteq A_i$, we have $f(y^{B_i}) = 0$ and $f(y) = 1$, so B_i is sensitive.

We have therefore constructed a set of k disjoint sensitive blocks for y, where k is the number of iterations of the algorithm. The maximum possible number of disjoint sensitive blocks for y is $\mathrm{bs}_y(f) \leq \mathrm{bs}^{(1)}(f)$, so the number of iterations is at most $\mathrm{bs}^{(1)}(f)$. Thus $\mathrm{D}(f) \leq \mathrm{C}^{(0)}(f)\,\mathrm{bs}^{(1)}(f)$. □

Corollary 2.31. *Let f be a total Boolean function. Then $\mathrm{D}(f) \leq 3\,\mathrm{R}_1(f)\,\mathrm{R}(f)$.*

Proof. We have $\mathrm{D}(f) \leq \mathrm{bs}^{(0)}(f)\,\mathrm{C}^{(1)}(f)$ and $\mathrm{D}(f) \leq \mathrm{bs}^{(1)}(f)\,\mathrm{C}^{(0)}(f)$, so we must have $\mathrm{D}(f) \leq \mathrm{bs}(f)\,\mathrm{C}_1(f)$ where $\mathrm{C}_1(f) = \min\{\mathrm{C}^{(0)}(f), \mathrm{C}^{(1)}(f)\}$. Then use Lemma 2.23 and Corollary 2.29. □

Next, we give an upper bound for $\mathrm{C}(f)$ in terms of $\mathrm{bs}(f)$ for total functions. To get the strongest version of the result, we define some new complexity measures.

Definition 2.32. *Let f be a total Boolean function, and let x be an input to f. The* sensitivity *of x (with respect to f), denoted $\mathrm{s}_x(f)$, is the number of bits $i \in [n]$ such that $f(x^i) \neq f(x)$. We also use $\mathrm{or}_x(f)$ to denote the size of the largest set of bits $S \subseteq [n]$ such that $f(x^B) \neq f(x)$ for all $B \subseteq S$ with $B \neq \emptyset$.*

As usual, we define $\mathrm{s}(f) = \max_x \mathrm{s}_x(f)$ and $\mathrm{or}(f) = \max_x \mathrm{or}_x(f)$, with $\mathrm{s}^{(0)}(f)$, $\mathrm{s}^{(1)}(f)$, $\mathrm{or}^{(0)}(f)$, and $\mathrm{or}^{(1)}(f)$ defined in the natural way.

Note that the sensitivity of an input is simply the block sensitivity with all the blocks restricted to be of size 1. Hence $\mathrm{s}_x(f) \leq \mathrm{bs}_x(f)$ for all x. Moreover, $\mathrm{or}_x(f)$ is the largest copy of the OR function we can find in the vicinity of x. The set S in the definition of $\mathrm{or}_x(f)$ must consist of sensitive bits for x, so $\mathrm{or}_x(f) \leq \mathrm{s}_x(f) \leq \mathrm{bs}_x(f)$.

We have defined $\mathrm{s}(f)$ and $\mathrm{or}(f)$ for total functions only, because unlike $\mathrm{bs}(f)$. This is because these measures make less sense for partial functions, and are essentially never used in that context.

We are now ready to prove an upper bound on $\mathrm{C}(f)$ for total functions. The following argument is due to Nisan [65].

Theorem 2.33. *Let $f : \{0,1\}^n \to \{0,1\}$ be a total Boolean function. Then*

$$\mathrm{C}_x(f) \leq \mathrm{bs}_x(f)\,\mathrm{or}^{(\overline{f(x)})}(f) \leq \mathrm{bs}_x(f)\,\mathrm{s}^{(\overline{f(x)})}(f).$$

In particular,

$$\mathrm{C}^{(0)}(f) \leq \mathrm{bs}^{(0)}(f)\,\mathrm{or}^{(1)}(f) \leq \mathrm{bs}^{(0)}(f)\,\mathrm{s}^{(1)}(f) \leq \mathrm{bs}^{(0)}(f)\,\mathrm{bs}^{(1)}(f),$$

so $\mathrm{C}(f) \leq \mathrm{bs}(f)\min\{\mathrm{or}^{(0)}(f), \mathrm{or}^{(1)}(f)\} \leq \mathrm{bs}(f)^2$.

Proof. Let S be a maximal set of disjoint sensitive blocks of x, and assume each block in S is minimal: that is, no proper subset of it is also a sensitive block. Note that $|S| = \mathrm{bs}_x(f)$. The partial assignment consisting of all the bits in all the blocks in S must be a certificate for x. This is because otherwise, there would be some input y with $f(y) \neq f(x)$ consistent with the partial assignment, and this input y would disagree with x on some block disjoint from those in S, which contradicts the maximality of S.

Hence, we described a certificate c for x. By definition, its size will be at least $\mathrm{C}_x(f)$, since $\mathrm{C}_x(f)$ is the size of the smallest possible certificate for x. On the other hand, the size of c is at most $\mathrm{bs}_x(f)$ times the maximum size of a block in S. It remains to upper bound the size of a minimal sensitive block of x.

Let B be a minimal sensitive block of x. Then and $f(x^B) \neq f(x)$. Since B is minimal, A is not a sensitive block for each proper subset A of B. Thus $f(x^A) = f(x)$. Note that $x^A = (x^B)^{B \setminus A}$. Setting $C = B \setminus A$, we see that $f((x^B)^C) = f(x) \neq f(x^B)$ for all non-empty subsets C of B. By definition, we conclude that $\mathrm{or}_{x^B}(f) \geq |B|$. Hence $|B| \leq \mathrm{or}^{(f(x^B))}(f) = \mathrm{or}^{(\overline{f(x)})}(f)$.

We conclude that $\mathrm{or}^{(\overline{f(x)})}(f)$ upper bounds the size of each minimal sensitive block of x, so we have $\mathrm{C}_x(f) \leq |c| \leq \mathrm{bs}_x(f)\,\mathrm{or}^{(\overline{f(x)})}(f)$. □

Corollary 2.34. *If f is a total Boolean function, we have $\mathrm{D}(f) \leq \mathrm{bs}(f)^3 \leq 27\,\mathrm{R}(f)$.*

2.2.4 Polynomial Degree

An interesting class of complexity measures are those corresponding to the degree of a representing polynomial. In other words, we will represent a function $f : \{0,1\}^n \to \{0,1\}$ by a polynomial on n variables, corresponding to the n input bits.

Definition 2.35. *Fix a field $\mathbb{F}$. For a (possibly partial) Boolean function f and a polynomial p on n variables over $\mathbb{F}$, we say that p* represents *f if $f(x) = p(x)$ for all $x \in \mathrm{Dom}(f)$, where we identify the bits 0 and 1 with the zero and one elements of the field $\mathbb{F}$.*

When $\mathbb{F} = \mathbb{R}$, we add the requirement that $p(x) \in [0,1]$ even for input $x \in \{0,1\}^n$ that is outside of $\mathrm{Dom}(f)$.

The additional requirement in the case that $\mathbb{F}$ will come in useful later, when we discuss polynomial degree over the reals as a lower bound for randomized and quantum algorithms.

We will focus on multilinear polynomials (where the degree in each variable is at most 1), since for Boolean input bits, we have $x_i^2 = x_i$ anyway. The following lemma will come in handy.

Lemma 2.36. *Let $\mathbb{F}$ be a field. Then any function $f : \{0,1\}^n \to \mathbb{F}$ can be represented in a unique way as a multilinear polynomial of degree at most n with coefficients in $\mathbb{F}$.*

Proof. Any function $f : \{0,1\}^n \to \mathbb{F}$ can be viewed as a vector in $\mathbb{F}^{2^n}$. For each block $S \subseteq [n]$, the monomial $m_S := \Pi_{i \in S} x_i$ is a function $m_S : \{0,1\}^n \to \mathbb{F}$ that evaluates to 1 on inputs x that are 1 on the bits in S, and evaluates to 0 otherwise. There are 2^n monomials on n variables, and it is not hard to see that they constitute independent vectors in $\mathbb{F}^{2^n}$; hence, they form a basis for this vector space. This means any function $f : \{0,1\}^n \to \mathbb{F}$ can be represented uniquely as a linear combination of monomials, or in other words, as a multilinear polynomial of degree at most n. □

In particular, this means all total Boolean functions can be uniquely represented as a multilinear polynomial of degree at most n over every field, though the actual polynomial can be different for each field. Partial Boolean functions can still be represented in this way, but the representation will no longer be unique. Polynomial representation over a fixed field defines a query complexity measure, as follows.

Definition 2.37. *Let $\mathbb{F}$ be a field, and let f be a (possibly partial) Boolean function. The* degree of f over $\mathbb{F}$, *denoted* $\deg_{\mathbb{F}}(f)$, *is the minimum degree of a polynomial over $\mathbb{F}$ representing f. When $\mathbb{F} = \mathbb{R}$, we omit it and denote the degree by* $\deg(f)$.

The key relevance of polynomials to query complexity comes from the following observation.

Lemma 2.38. *Let f be a (possibly partial) Boolean function. Then $\mathrm{D}(f) \geq \deg_{\mathbb{F}}(f)$ for every field $\mathbb{F}$, and $\mathrm{Q}_E(f) \geq \deg(f)/2$.*

Proof. We prove that for every decision tree of height T, there is a polynomial of degree at most T computing the same function as that tree. We proceed by induction on T. If $T = 0$, the decision tree is a leaf in $\{0, 1\}$, and computes the constant function; hence there is a constant polynomial (whose degree is 0) that computes the same function as the decision tree.

For $T > 0$, let $D = (i, D_0, D_1)$ be a decision tree of height T. Then D_0 and D_1 are decision trees of height at most $T-1$, so by the induction hypothesis they can be represented by polynomials p_0 and p_1 of degree at most $T - 1$. Recall that $D(x)$ evaluates to $D_0(x)$ if $x_i = 0$ and to $D_1(x)$ if $x_i = 1$. Therefore, a polynomial for D is $x_i \cdot p_1(x) + (1 - x_i) \cdot p_0(x)$, which has degree at most T, as desired. Note that the field did not matter.

We now deal with quantum algorithms. Let Q be a T-query quantum algorithm that computes f exactly. Then Q is a sequence of unitary matrices $U_0, U_1, \ldots, U_T$, none of which depend on the input x. The output of the algorithm is the result of measuring the output register of the tape $U_T U^x U_{T-1} U^x \ldots U^x U_0 |init\rangle$. Now, recall that the oracle unitary U^x maps $|A\rangle|i\rangle|b\rangle|o\rangle \to |A\rangle|i\rangle|b \oplus x_i\rangle|o\rangle$. We write

$$|A\rangle|i\rangle|b \oplus x_i\rangle|o\rangle = x_i|A\rangle|i\rangle|b\rangle|o\rangle + (1 - x_i)|A\rangle|i\rangle|b \oplus 1\rangle|o\rangle.$$

It is then clear that if we write the entries of U^x as functions of x, each of these functions is either the constant 0, the function x_i for some $i \in [n]$, or the function $1 - x_i$ for some $i \in [n]$. In other words, the entries of U^x are linear in the bits of x. From this, it follows that the amplitudes of the final state of the algorithm are polynomials (over $\mathbb{C}$) of degree at most T in the variables $x_1, x_2, \ldots, x_T$.

Now, the result of measuring the output register is defined by probabilities that are the sum of squared norms of amplitudes; hence the probabilities of the events $Q(x) = 0$ and $Q(x) = 1$ are polynomials (over $\mathbb{R}$) of degree at most $2T$ in the variables $x_1, x_2, \ldots, x_T$. In particular, the probability of the event $Q(x) = 1$ is a real polynomial p of degree at most $2T$ which satisfies $p(x) = 1$ when $f(x) = 1$ and $p(x) = 0$ when $f(x) \neq 1$. Moreover, on each input $x \in \{0, 1\}^n$ (even outside of $\mathrm{Dom}(f)$), the probability of getting 1 when Q is run on x is between 0 and 1, so $p(x) \in [0, 1]$. Hence p computes f over the reals, and $\deg(f) \leq 2T = 2\,\mathrm{Q}_E(f)$. □

We can also use a version of polynomial degree to lower bound bounded-error query complexity measures such as $\mathrm{R}(f)$ and $\mathrm{Q}(f)$.

Definition 2.39. *Let f be a (possibly partial) Boolean function on n variables. A polynomial p on n variables over $\mathbb{R}$ is said to* approximate *f to error ϵ if $|p(x) - f(x)| \leq \epsilon$ for all $x \in \mathrm{Dom}(f)$ and $p(x) \in [0,1]$ for all $x \in \{0,1\}^n$ (even inputs x outside the domain of f).*

The approximate degree of f to error ϵ, denoted $\widetilde{\deg}_\epsilon(f)$, is the minimum degree of a polynomial approximating f to error ϵ. When $\epsilon = 1/3$, we omit it and write $\widetilde{\deg}(f)$.

Lemma 2.40. *Let f be a (possibly partial) Boolean function. Then $\mathrm{R}_\epsilon(f) \geq \widetilde{\deg}_\epsilon(f)$ and $\mathrm{Q}_\epsilon(f) \geq (1/2)\widetilde{\deg}_\epsilon(f)$.*

Proof. As we saw in the proof of Lemma 2.38, the acceptance probability of a T-query quantum algorithm can be represented as a real polynomial of degree at most $2T$. This acceptance probability is in $[0,1]$ for all Boolean input strings (even those outside the domain of f), and it is at least $1-\epsilon$ for inputs x with $f(x) = 1$ and at most ϵ for inputs with $f(x) = 0$. Therefore, the polynomial we get from a quantum query algorithm computing f is an approximating polynomial for f, so $\mathrm{Q}_\epsilon(f) \geq (1/2)\widetilde{\deg}_\epsilon(f)$.

Next, consider the randomized algorithm computing f with $\mathrm{R}_\epsilon(f)$ queries. This is a probability distribution over deterministic algorithms that use at most $\mathrm{R}_\epsilon(f)$ queries each. Each deterministic algorithm computes some Boolean function (possibly different from f), and by Lemma 2.38, that function is computed by a real polynomial of degree at most $\mathrm{R}_\epsilon(f)$. Consider the real polynomial p we get by taking the linear combination of these polynomials defined by the probability distribution of the randomized algorithm. Since all the polynomials satisfied $p(x) \in [0,1]$ for all $x \in \{0,1\}^n$, their weighted average p also satisfies this. In addition, the correctness of the randomized algorithm implies that p approximates f to error ϵ on $\mathrm{Dom}(f)$, so p is an approximating polynomial for f. Since the degree of p is at most $\mathrm{R}_\epsilon(f)$, we get $\mathrm{R}_\epsilon(f) \geq \widetilde{\deg}_\epsilon(f)$. □

Next, we show that $\widetilde{\deg}(f)$ is lower bounded by $\sqrt{\mathrm{bs}(f)}$, which means block sensitivity can be used as a lower bound for quantum query complexity. We start with the following lemma.

Lemma 2.41. *Let M be a query complexity measure behaving well with input alphabet change, bit duplication, restriction to a promise, padding, and two-sidedness. Then for all (possibly partial) Boolean functions f, we have $M(f) \geq M(\textsc{PromiseOR}_{\mathrm{bs}(f)})$.*

Proof. First, observe that block sensitivity is itself a two-sided measure that behaves well with input alphabet change, bit duplication, restriction to a promise, and padding.

Let $x \in \{0,1\}^n$ be such that $\mathrm{bs}(f) = \mathrm{bs}_x(f)$. If $f(x) = 1$, we prove the lemma for $\bar{f}$ instead, and appeal to two-sidedness to get the result for M. Thus, without loss of generality, we have $f(x) = 0$. If $x \neq 0^n$, we prove the result for $f_{\oplus x}$ instead, and appeal to the alphabet change property to get $M(f_{\oplus x}) = M(f)$, proving the result for f. Thus, without loss of generality, we have $x = 0^n$.

Let $B_1, B_2, \ldots, B_{\mathrm{bs}(f)}$ be disjoint sensitive blocks for 0^n. Let $S \subseteq \mathrm{Dom}(f)$ be the set of strings consisting of 0^n and $(0^n)^{B_i}$ for all $i \in [\mathrm{bs}(f)]$. Then $M(f) \geq M(f|_S)$ (by restriction to a promise) and $\mathrm{bs}(f|_S) = \mathrm{bs}(f)$, so it suffices to prove the result for $g = f|_S$. Next, we appeal to padding to remove any bits in $[n]$ that are not in any of the sensitive blocks (such bits are always 0 inside the domain S of g). We also appeal to bit duplication to remove all but one bit in each of the blocks $B_1, B_2 \ldots, B_{\mathrm{bs}(f)}$. The remaining function is exactly $\textsc{PromiseOR}_{\mathrm{bs}(f)}$, so we've shown that $M(f) \geq M(g) = M(\textsc{PromiseOR}_{\mathrm{bs}(f)})$, as desired. □

Lemma 2.42. *The measures $\deg_{\mathbb{F}}(f)$ and $\widetilde{\deg}_{\epsilon}(f)$, for any field $\mathbb{F}$ and any $\epsilon \in [0, 1/2)$, are two-sided and behave well under alphabet change, bit duplication, restriction to a promise, and padding.*

Proof. Let p be a representing or approximating polynomial for f. Then $1 - p(x)$ is a representing or approximating polynomial for $\bar{f}$, so these measures are two-sided.

For alphabet change, fix $y \in \{0,1\}^n$, and consider $f_{\oplus y}$. Let p' be the polynomial p with $(1 - x_i)$ substituted for the variable x_i whenever $y_i = 1$. Then p' represents (approximates) $f_{\oplus y}$, and the degree of p' is at most that of p. Thus $M(f_{\oplus y}) \leq M(f)$ for any degree measure M. Since $(f_{\oplus y})_{\oplus y} = f$, we get $M(f_{\oplus y}) = M(f)$.

If f' is f with a duplicated bit, then a representing (approximating) polynomial for f also represents (approximates) f'. For the other direction, take a polynomial for f' and substitute the variable for the duplicated bit into the variable of its copy; this gives a polynomial for f (after using $x_i^2 = x_i$ to make the polynomial multilinear). The case of padding follows similarly (just substitute a constant instead). □

To give a lower bound for $\widetilde{\deg}(f)$ in terms of $\mathrm{bs}(f)$, it therefore suffices to give a lower bound for $\widetilde{\deg}(\textsc{PromiseOR}_{\mathrm{bs}(f)})$. To do this, we will use the symmetrization trick, first used by Minsky and Papert [62].

Lemma 2.43. *Let $p : \{0,1\}^n \to \mathbb{R}$ be a real multilinear polynomial. Then the function*

$$p^{\mathrm{sym}} := \frac{1}{n!} \sum_{\sigma \in S_n} p(x_{\sigma(1)} x_{\sigma(2)} \cdots x_{\sigma(n)})$$

can be represented as a single-variate real polynomial q such that $q(|x|) = p^{\mathrm{sym}}(x)$ for all $x \in \{0,1\}^n$ (here $|x|$ denotes the Hamming weight of x). Moreover, the degree of q is at most that of p.

Proof. For each multilinear monomial m of degree d, m^{sym} is the same polynomial J_d, the average of all degree-d multilinear monomials. Writing p as a sum of monomials $p = \sum_{S \subseteq [n]} \alpha_S m_S$, we get

$$p^{\mathrm{sym}} = \sum_{S \subseteq [n]} \alpha_S m_S^{\mathrm{sym}} = \sum_{S \subseteq [n]} \alpha_S J_{|S|} = \beta_0 J_0 + \beta_1 J_1 + \cdots + \beta_{\deg(p)} J_{\deg(p)}$$

for some constants $\beta_0, \beta_1, \ldots, \beta_{\deg(p)} \in \mathbb{R}$. It therefore suffices to show that each J_d can be represented by a single-variate polynomial q_d with $q_d(|x|) = J_d(x)$ for all $x \in \{0,1\}^n$.

We use induction. We have $J_0 = 1$ and $J_1 = x_1 + x_2 + \cdots + x_n = |x|$ for $x \in \{0,1\}^n$. For $d \geq 2$, consider the polynomial $|x|^d = (x_1 + x_2 + \cdots + x_n)^d$. Make this polynomial multilinear by substituting $x_i^2 \to x_i$ repeatedly (this does not change the value of the polynomial over $\{0,1\}^n$). Then we get a symmetric multilinear polynomial s_d of degree at most d which evaluates to $|x|^d$ over $\{0,1\}^n$. Since it is symmetric, it is not hard to see that $s_d^{\mathrm{sym}} = s_d$. Thus s_d is a linear combination of $J_0, J_1, \ldots, J_d$. Moreover, the degree of s_d is d, which means the coefficient of J_d in this linear combination is non-zero. But this means we can write J_d as a linear combination of s_d and $J_0, J_1, \ldots, J_{d-1}$. By the induction hypothesis, each J_i with $i < d$ is represented by a single-variate polynomial q_i of degree i. Then for all $x \in \{0,1\}^n$, $J_d(x)$ is equal to a linear combination of $q_0(|x|), q_1(|x|), \ldots, q_{d-1}(|x|), |x|^d$, so it is a polynomial of degree at most d in the variable $|x|$. This completes the argument. □

Note that if we symmetrize a polynomial approximating f, we get a univariate polynomial q such that $q(k)$ approximates (to within the same error ϵ) the average of $f(x)$ over strings x of Hamming weight k. This means that to lower bound $\widetilde{\deg}(\text{PROMISEOR}_n)$, it suffices to lower bound the degree of a univariate real polynomial q with $q(0) \approx 0$, $q(1) \approx 1$, and $q(k) \in [0,1]$ for $k = 2, 3, \ldots, n$.

Corollary 2.44. *Let f be a (possibly partial) Boolean function. Let $S \subseteq \{0,1,\ldots n\}$ be the set of Hamming weights k such that every $x \in \{0,1\}^n$ of Hamming weight k is in $\mathrm{Dom}(f)$. Let $f^{\mathrm{sym}} : S \to \mathbb{R}$ be the function where $f^{\mathrm{sym}}(k)$ is the average of $f(x)$ over all $x \in \mathrm{Dom}(f)$ with Hamming weight k. We say a single variate polynomial $q : \mathbb{R} \to \mathbb{R}$ approximates f^{sym} to error ϵ if $|q(k) - f^{\mathrm{sym}}(k)| \leq \epsilon$ and $q(k) \in [0,1]$ for all $k = 0, 1, \ldots, n$. We use $\widetilde{\deg}_\epsilon(f^{\mathrm{sym}})$ to denote the minimum degree of a polynomial approximating f^{sym} to error ϵ.*

Then $\widetilde{\deg}_\epsilon(f) \geq \widetilde{\deg}_\epsilon(f^{\mathrm{sym}})$.

Proof. This follows immediately from Lemma 2.43 by noting that the average of approximations for $f(x)$ (where x has Hamming weight k) is an approximation for the average of $f(x)$ (over all x with Hamming weight k). □

We use the following tool for dealing with polynomials that are bounded in an interval.

Theorem 2.45 (Markov Brothers' Inequality). *Let $p : \mathbb{R} \to \mathbb{R}$ be a real polynomial of degree d. If $p(x) \in [b, b']$ for all $x \in [a, a']$, then for all $x \in [a, a']$, we have $|p'(x)| \leq \frac{b'-b}{a'-a} d^2$.*

Echlich and Zeller [33], and independently Rivlin and Cheney [71], extended Markov Brothers' inequality to the case where the polynomial is only bounded on integer points.

Theorem 2.46. *Let $p : \mathbb{R} \to \mathbb{R}$ be a real polynomial of degree d. If $p(i) \in [b, b']$ for all $i = 0, 1, \ldots, n$, then for all $x \in [0, n]$, $|p'(x)| \leq \frac{b'-b}{n-d^2} d^2$.*

Proof. Let $c = \max_{x \in [0,n]} |p'(x)|$. The key idea is that if the derivative of p is bounded, it cannot deviate too far outside of $[b, b']$ on the domain $[0, n]$. This is because for any $x \in [0, n]$ for which $p(x)$ is outside $[b, b']$, we find $i \in \{0, 1, \ldots, n\}$ such that $|x - i| \leq 1/2$, and by the mean value theorem, there is a point z between x and i such that $|p'(z)| \geq |p(x) - p(i)|/|x - i| \geq 2|p(x) - p(i)|$. Since $p(i) \in [b, b']$, this implies $p(x)$ lies inside $[b - c/2, b' + c/2]$ for all $x \in [0, n]$.

Markov Brothers' inequality (Theorem 2.45) now implies $c \leq \frac{c+(b'-b)}{n} d^2$. Rearranging gives $c \leq \frac{b'-b}{n-d^2} d^2$. □

Putting everything together, we now prove that approximate degree is lower bounded by block sensitivity, as shown by Nisan and Szegedy [66].

Theorem 2.47. *Let f be a (possibly partial) Boolean function. Then $\widetilde{\deg}(f) \geq \sqrt{\mathrm{bs}(f)}/2$.*

Proof. By Lemma 2.41 and Lemma 2.42, to lower bound $\widetilde{\deg}_\epsilon(f)$ it suffices to lower bound $\widetilde{\deg}_\epsilon(\text{PROMISEOR}_n)$ for $n = \mathrm{bs}(f)$. By Corollary 2.44, it suffices to lower bound the approximate degree of $\text{PROMISEOR}_n^{\mathrm{sym}} : \{0, 1, \ldots, n\} \to [0, 1]$, the single variate function with $\text{PROMISEOR}_n^{\mathrm{sym}}(0) = 0$, $\text{PROMISEOR}_n^{\mathrm{sym}}(1) = 1$, and $\text{PROMISEOR}_n^{\mathrm{sym}}$ behaving arbitrarily (inside $[0, 1]$) on $2, 3, \ldots, n$.

Any polynomial p approximating this function to error ϵ satisfies $p(i) \in [0, 1]$ for $i = 0, 1, \ldots, n$, as well as $p(0) \leq \epsilon$ and $p(1) \geq 1 - \epsilon$. By the mean value theorem, there is some

$x \in [0,1]$ with $|p'(x)| \geq (p(1) - p(0))/(1-0) \geq 1 - 2\epsilon$. By Theorem 2.46, the degree of p is lower bounded by $\sqrt{n(1-2\epsilon)/(2-2\epsilon)}$. It follows that

$$\widetilde{\deg}_\epsilon(f) \geq \sqrt{\mathrm{bs}(f)(1-2\epsilon)/(2-2\epsilon)},$$

so $\widetilde{\deg}(f) \geq \sqrt{\mathrm{bs}(f)}/2$. □

Corollary 2.48. *Let f be a (possibly partial) Boolean function. Then $\mathrm{Q}(f) \geq \sqrt{bs(f)}/4$.*

Proof. This follows from Theorem 2.47 and Lemma 2.40. □

Corollary 2.49. *Let f be a total function. Then $\mathrm{D}(f) \leq 4096\,\mathrm{Q}(f)^6$.*

Proof. This follows from Corollary 2.48 and Corollary 2.34. □

2.2.5 Further Results for Total Functions

A few further relationships are known for query complexity measures of total Boolean functions. First, Midrijanis [60] showed the following.

Theorem 2.50. *Let f be a total Boolean function. Then $\mathrm{D}(f) \leq \deg(f)\,\mathrm{bs}(f)$.*

Combining with Theorem 2.47 (setting $\epsilon = 0$) gives

Corollary 2.51. *Let f be a total Boolean function. Then $\mathrm{D}(f) \leq 2\deg(f)^3 \leq 16\,\mathrm{Q}_E(f)^3$.*

Another result of Midrijanis [61] relates $\mathrm{R}(f)$ and $\mathrm{R}_0(f)$ for total Boolean functions. This result was improved slightly by Kulkarni and Tal [54], who showed the following.

Theorem 2.52. *Let f be a total Boolean function. Then $\mathrm{R}_0(f) = O(\mathrm{R}(f)^2 \log \mathrm{R}(f))$.*

For this result, Kulkarni and Tal used a measure called randomized certificate complexity (or fractional block sensitivity). This measure, denoted by $\mathrm{RC}(f)$, was first defined by [1]; we do not define it here, but we note that it lies between $\mathrm{bs}(f)$ and $\mathrm{R}(f)$.

This essentially completes the list of known relationships between query complexity measures of total Boolean functions; other known relationships between the measures introduced so far can all be derived from combining the theorems we saw in elementary ways.

We now summarize the best known relations between the complexity measures studied in this chapter. Figure 2-1 depicts all known relations of the type $M_1(f) = O(M_2(f))$ for complexity measures $M_1(f)$ and $M_2(f)$. An upward line from M_1 to M_2 in Figure 2-1 indicates $M_1(f) = O(M_2(f))$ for all (partial or total) Boolean functions f.

All other known relationships between these measures follow by combining the relationships in Figure 2-1 with the following relationships that we showed hold for all total Boolean functions:

- $C(f) \leq \mathrm{bs}(f)^2$ [65]
- $D(f) \leq C(f)\,\mathrm{bs}(f)$ [17]
- $D(f) \leq \deg(f)^3$ [60, 83]
- $\mathrm{RC}(f) = O(\widetilde{\deg}(f)^2)$ [54]
- $R_0(f) = O(R(f)^2 \log R(f))$ [54]

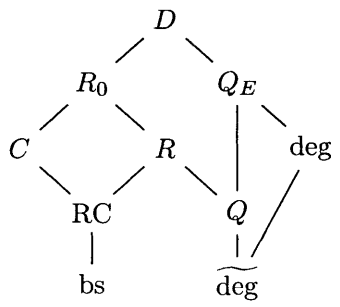


Figure 2-1: Relations between complexity measures.

Separations for total functions are discussed in Chapter 3. In particular, the work of Ambainis, Balodis, Belovs, Lee, Santha, and Smotrovs [9] showed several new separations between pairs of measures, and in Chapter 3 we give a general framework for turning partial function separations into total function separations. Before these works, the largest separations were either simple functions like OR, or recursively composed functions like the NAND-TREE function [73].

As a final remark, we define the composition of Boolean functions, which is a useful for designing new Boolean functions. For any two (possibly partial) Boolean functions $g : G \to \{0,1\}$, where $G \subseteq \{0,1\}^n$, and $h : H \to \{0,1\}$, where $H \subseteq \{0,1\}^m$, we can define the composed function $g \circ h : D \to \{0,1\}$, where $D \subseteq \{0,1\}^{nm}$, as follows. For an input $(z_1, z_2, \ldots, z_{nm}) \in \{0,1\}^{nm}$, we define $g \circ h$ as follows (as depicted in Figure 2-2):

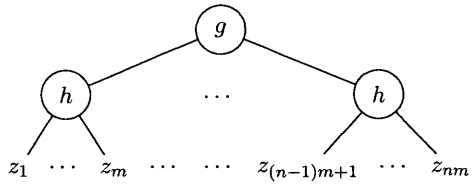


Figure 2-2: Composition $g \circ h$.

$$g \circ h(z) := g(h(z_1, \ldots, z_m), h(z_{m+1}, \ldots, z_{2m}), \ldots, h(z_{(n-1)m+1}, \ldots, z_{nm})), \tag{2.1}$$

where the function $g \circ h$ is only defined on z where the inputs to the h gates lie in the domain H and the n-bit input to g (consisting of $h(z_1, \ldots, z_m)$, $h(z_{m+1}, \ldots, z_{2m})$, $\ldots$, $h(z_{(n-1)m+1}, \ldots, z_{nm})$) lies in G.

Chapter 3

Cheat Sheets

3.1 Introduction

This chapter is based on work that appeared in [19] and [4], which is joint work with Scott Aaronson and Robin Kothari. We introduce the cheat sheet technique for constructing total Boolean functions, and characterize the behavior of cheat sheet functions in various models.

The goal of the cheat sheet technique is to turn a partial function that shows some large separation between two query complexity measures into a total function that preserves at least a bit of the same separation. In particular, we show how the cheat sheet technique can be applied to construct a family of total Boolean functions that satisfy $\mathrm{R}(f) = \tilde{\Omega}(\mathrm{Q}(f)^{2.5})$, beating the previous quadratic separation due to Grover search [41]. (Recall that the relationship $\mathrm{R}(f) = O(\mathrm{Q}(f)^6)$ is known [17], so we cannot hope for super-polynomial separations).

The cheat sheet technique is motivated by some recent query complexity breakthroughs, showing new total function query complexity separations for problems that stood open for decades: Ambainis, Balodis, Belovs, Lee, Santha, and Smotrovs [9] showed new separations for $\mathrm{D}(f)$ vs. $\mathrm{R}(f)$ (quadratic) and $\mathrm{D}(f)$ vs. $\mathrm{Q}(f)$ (quartic), as well as many others. They in turn built upon the techniques of Göös, Pitassi, and Watson [39], who showed a quadratic separation between $\mathrm{D}(f)$ and $\deg(f)$. Before these recent results, the largest separation between deterministic and randomized query complexity was around 1.3 from the NAND-TREE function, due to Saks and Wigderson [73].

3.1.1 Results

Randomized versus quantum query complexity

We present the first super-quadratic separation between quantum and classical query complexity when both models are allowed bounded error. This is a counterexample to a conjecture that had been widely believed about the nature of quantum speed-ups: that if the function is total, so that the inputs are not handpicked to orchestrate a quantum speed-up, then the best quantum advantage possible is the quadratic speed-up achieved by Grover's algorithm.

Theorem 3.1. *There exists a total function f such that $R(f) = \tilde{\Omega}(Q(f)^{2.5})$.*

Theorem 3.1 can be boosted to a cubic separation, i.e., $R(f) = \tilde{\Omega}(Q(f)^3)$, in a completely black-box manner if there exists a partial function (a function defined only on a subset

of $\{0,1\}^n$) with $\tilde{\Omega}(n)$ randomized query complexity but only poly($\log(n)$) quantum query complexity (the best possible separation between these measures up to log factors). It is conjectured that a recently studied partial function called k-fold Forrelation achieves this separation [2].

Quantum query complexity versus polynomial degree

Approximate degree has proved to be a fruitful lower bound technique for quantum query complexity, especially for problems like the collision and element distinctness problems [5], where it is known that the original quantum adversary method of Ambainis, another commonly used lower bound technique, cannot show nontrivial lower bounds.

For any lower bound technique, it is natural to ask whether there exist functions where the technique fails to prove a tight lower bound. Answering this question, Ambainis showed that the approximate degree of a function can be asymptotically smaller than its quantum query complexity by exhibiting a function with $Q(f) = \Omega(\widetilde{\deg}(f)^{1.3219})$ [7]. We dramatically strengthen this separation to obtain nearly a 4^{th} power gap.

Theorem 3.2. *There exists a total function f such that $Q(f) \geq \widetilde{\deg}(f)^{4-o(1)}$.*

Theorem 3.2 is optimal assuming the conjecture that $D(f) = O(\mathrm{bs}(f)^2)$.

The cheat sheet technique

These separations are shown using a new technique for proving separations between query measures, which we call the cheat sheet technique. Our technique is based on a generic transformation that takes any (partial or total) function and transforms it into a "cheat sheet version" of the function that has desirable properties for proving separations.

While the strategy is inspired by the recent breakthrough results [39, 9] (and bears some similarity to older works [57, 10]), it represents a more general approach: it provides a framework for proving separations and allows many separations to be shown in a unified manner. Thus the task of proving separations is reduced to the simpler task of finding the right function to plug into this construction. For example, it can be used to convert a partial function separation between two models into a weaker total function separation.

In this chapter we demonstrate the power of the cheat sheet technique by using it to exhibit several new total function separations in query complexity.

Other separations

On the path to proving Theorem 3.2, we show several new separations. First, we quadratically separate quantum query complexity from certificate complexity, which is essentially optimal since $Q(f) \leq C(f)^2$.

Theorem 3.3. *There exists a total function f such that $Q(f) = \tilde{\Omega}(C(f)^2)$.*

Besides being a stepping stone to proving Theorem 3.2, the question of $Q(f)$ versus $C(f)$ has been studied because it is known that the original adversary method of Ambainis (also known as the positive-weights adversary method) cannot prove a lower bound greater than $C(f)$ [81, 44].

Plugging the function of Theorem 3.3 into the cheat sheet framework directly yields a function whose quantum query complexity is quadratically larger than its (exact) degree, improving the recent result of [39], who exhibited a function with $D(f) = \tilde{\Omega}(\deg(f)^2)$.

Theorem 3.4. *There exists a total function f such that $Q(f) = \tilde{\Omega}(\deg(f)^2)$.*

This theorem works in a very black-box way: any separation between a measure like $Q(f)$ or $R(f)$ and $C(f)$ can be plugged into the cheat sheet technique to obtain the same separation (up to log factors) between that measure and exact degree. For example, since the AND-OR function satisfies $R(f) = \Omega(C(f)^2)$, the cheat sheet version of AND-OR satisfies $R(f) = \tilde{\Omega}(\deg(f)^2)$.

This result also shows limitations on using $\deg(f)$ to lower bound $Q_E(f)$, since $\deg(f)$ can sometimes be quadratically smaller than $Q_E(f)$ and can even be quadratically smaller than $Q(f)$.

Summary of results

We summarize our new results in the table below.

Separation achieved	Known relation	Result
$R(f) = \tilde{\Omega}(Q(f)^{2.5})$	$R(f) = O(Q(f)^6)$	Theorem 3.1
$Q(f) \geq \widetilde{\deg}(f)^{4-o(1)}$	$Q(f) = O(\widetilde{\deg}(f)^6)$	Theorem 3.2
$Q(f) = \tilde{\Omega}(C(f)^2)$	$Q(f) = O(C(f)^2)$	Theorem 3.3
$Q(f) = \tilde{\Omega}(\deg(f)^2)$	$Q(f) = O(\deg(f)^3)$	Theorem 3.4

Table 3.1: New separations shown in this chapter

We are also able to use the cheat sheet technique to reprove many of the query separations of [39, 9, 38]. Some of their results are subsumed by the results in Table 3.1. We also prove $R_0(f) = \tilde{\Omega}(Q(f)^3)$ (Theorem 3.23) and $R(f) = \tilde{\Omega}(Q_E(f)^{3/2})$ (Theorem 3.24) in Section 3.5.4. For more details, see Table 3.2 in which we summarize our results and the best known separations and relations between query measures.

It is also worth pointing out what we are unable to reproduce with our framework. The only separations from these papers that we do not reproduce are those that make essential use of "back pointers". Reproducing these separations in the cheat sheet framework is an interesting direction for future research.

3.1.2 Overview of techniques

Cheat sheet technique and randomized versus quantum query complexity

While we know large partial function separations between randomized and quantum query complexity, such as Simon's problem [79] or the Forrelation problem [2] that satisfies $R(g) = \tilde{\Omega}(\sqrt{n})$ and $Q(g) = 1$, it is unclear how to make these functions total. We could simply define g to be 0 on inputs not in the domain, but then the quantum algorithm would need to be able to decide whether an input is in the domain.

To solve this problem, we compose the Forrelation problem g with a total function $h = \text{AND-OR}$ on n^2 bits, to obtain a partial function $f = g \circ h$ on n^3 bits that inherits the properties of both g and h. Since AND-OR has low certificate complexity, it is easier to certify that an input lies in the domain of $g \circ h$, since we only need to certify the outputs of the n different h gates.

Since $R(\text{AND-OR}_{n^2}) = \Omega(n^2)$, it can be shown that $R(f) = R(g \circ h) = \Omega(n^{2.5})$. However, since $Q(\text{AND-OR}_{n^2}) = O(n)$, we have $Q(f) = O(n)$. Now f is still a partial function, but

Table 3.2: Best known separations between complexity measures

	D	R_0	R	C	RC	bs	Q_E	deg	Q	$\widetilde{\deg}$
D		2, 2 [9]	2*, 3 [9]	2, 2 $\wedge \circ \vee$	2*, 3 $\wedge \circ \vee$	2*, 3 $\wedge \circ \vee$	2, 3 [9]	2, 3 [39]	4*, 6 [9]	4*, 6 [9]
R_0	1, 1 $\oplus$		2, 2 [9]	2, 2 $\wedge \circ \vee$	2*, 3 $\wedge \circ \vee$	2*, 3 $\wedge \circ \vee$	2, 3 [9]	2, 3 [38]	3, 6 [9]	4*, 6 [9]
R	1, 1 $\oplus$	1, 1 $\oplus$		2, 2 $\wedge \circ \vee$	2*, 3 $\wedge \circ \vee$	2*, 3 $\wedge \circ \vee$	1.5, 3 [9]	2, 3 [38]	2.5, 6 Th. 3.1	4*, 6 [9]
C	1, 1 $\oplus$	1, 1 $\oplus$	1, 2 $\oplus$		2, 2 [36]	2, 2 [36]	1.1, 3 [8]	1.6, 3 [67]	2, 4 $\wedge$	2, 4 $\wedge$
RC	1, 1 $\oplus$	1, 1 $\oplus$	1, 1 $\oplus$	1, 1 $\oplus$		1.5, 2 [36]	1.1, 2 [8]	1.6, 2 [67]	2, 2 $\wedge$	2, 2 $\wedge$
bs	1, 1 $\oplus$	1, 1 $\oplus$	1, 1 $\oplus$	1, 1 $\oplus$	1, 1 $\oplus$		1.1, 2 [8]	1.6, 2 [67]	2, 2 $\wedge$	2, 2 $\wedge$
Q_E	1, 1 $\oplus$	1.3, 2 $\overline{\wedge}$-tree	1.3, 3 $\overline{\wedge}$-tree	2, 2 $\wedge \circ \vee$	2*, 3 $\wedge \circ \vee$	2*, 3 $\wedge \circ \vee$		2, 3 Th. 3.4	2, 6 $\wedge$	4*, 6 Th. 3.2
deg	1, 1 $\oplus$	1.3, 2 $\overline{\wedge}$-tree	1.3, 3 $\overline{\wedge}$-tree	2, 2 $\wedge \circ \vee$	2*, 3 $\wedge \circ \vee$	2*, 3 $\wedge \circ \vee$	1, 1 $\oplus$		2, 6 $\wedge$	2, 6 $\wedge$
Q	1, 1 $\oplus$	1, 1 $\oplus$	1, 1 $\oplus$	2, 2 Th. 3.3	2*, 3 Th. 3.3	2*, 3 Th. 3.3	1, 1 $\oplus$	2, 3 Th. 3.4		4*, 6 Th. 3.2
$\widetilde{\deg}$	1, 1 $\oplus$	1, 1 $\oplus$	1, 1 $\oplus$	2, 2 [29]	2, 3 [29]	2, 3 [29]	1, 1 $\oplus$	1, 1 $\oplus$	1, 1 $\oplus$	

An entry a, b in the row M_1 and column M_2 roughly[a] means $M_1(f) = \tilde{O}(M_2(f)^b)$ for all total f and there exists a total f with $M_1(f) = \tilde{\Omega}(M_2(f)^a)$. Each cell contains a citation or a description of a separating function, where $\oplus$ = PARITY, $\wedge$ = AND, $\wedge \circ \vee$ = AND-OR, and $\overline{\wedge}$-tree is the balanced NAND-tree function [73, 74]. Entries followed by a star (e.g., 2*) correspond to separations that are optimal if $D(f) = O(\mathrm{bs}(f)^2)$. Separations colored red are new to this work. Separations colored gray are separations from recent papers that we reprove in this work.

[a]More precisely it means $a \leq \mathrm{crit}(M_1, M_2) \leq b$, where we define $\mathrm{crit}(\cdot, \cdot)$ in Section 3.1.3.

there is a small certificate for the input being in the domain. If we could ensure that only the quantum algorithm had access to this certificate, but the randomized algorithm did not, we would have our power 2.5 separation.

To achieve this, we hide a "cheat sheet" in the input that contains all the information the quantum algorithm needs. We store it in a vast array of size n^{10} of potential cheat sheets, which cannot be searched quickly by brute force by any algorithm, classical or quantum. Thus, we must provide the quantum algorithm the address of the correct cheat sheet. But what information do we have that is known only to the quantum algorithm? The value of f on the input! While one bit does not help much, we can use $10 \log n$ copies of f acting on $10 \log n$ different inputs. The outputs to these $10 \log n$ problems can index into an array of size n^{10}, and can be determined by the quantum algorithm using only $O(n \log n)$ queries. At this index we will store the cheat sheet, which contains certificates for all $10 \log n$ instances of f, convincing us that all the inputs lie in the domain. Our cheat sheet function now evaluates to 1 if and only if all $10 \log n$ instances of f lie in the domain, *and* the cell in the array pointed to by the outputs of these $10 \log n$ instances of f contains a cheat sheet certifying that these inputs lie in the domain.

Quantum query complexity versus certificate complexity

Consider the k-SUM problem, in which we are given n numbers and have to decide if any k of them sum to 0 (mod M) for some M. For large k, the quantum query complexity of k-SUM is nearly linear in n [18]. While it is easy to certify 1-inputs by showing any k elements that sum to 0, the 0-inputs are difficult to certify and hence the certificate complexity is also linear in n.

Building on k-SUM, we define a new function that we call BLOCK k-SUM, whose quantum query complexity and certificate complexity are both linear in n. However, BLOCK k-SUM has a curious property: for both 1-inputs and 0-inputs, the certificates themselves consist almost exclusively of input bits set to 1. This means that if we compose this function with k-SUM, the composed function on n^2 bits has certificates of size $\tilde{O}(n)$ because any certificate of BLOCK k-SUM consists almost entirely of 1s, which correspond to k-SUM instances that output 1, which are easy to certify. On the other hand, the quantum query complexity of the composed function, which we call BKK for BLOCK k-SUM of k-SUM, is the product of the quantum query complexities of individual functions. Thus the certificate complexity of BKK is $\tilde{O}(n)$, but its quantum query complexity is $\tilde{\Omega}(n^2)$.

Quantum query complexity versus polynomial degree

We show that plugging any function that achieves $Q(f) = \tilde{\Omega}(n^2)$ and $C(f) = \tilde{O}(n)$ into the cheat sheet framework yields a function with $Q(f) = \tilde{\Omega}(n^2)$ and $\deg(f) = \tilde{O}(n)$. The quantum lower bound uses the hybrid method [21] and the recent strong direct product theorem for quantum query complexity [59]. The degree upper bound holds because for every potential cheat sheet location, there exists a degree $\tilde{O}(n)$ polynomial that checks if this location is the one pointed to by the given input. And since the input can point to at most one cheat sheet, the sum of these polynomials equals the function f.

To achieve a fourth power separation between $Q(f)$ and $\widetilde{\deg}(f)$, we need to check whether a cheat sheet is valid using a polynomial of degree $\tilde{O}(\sqrt{C(f)})$. Some certificates can be checked by Grover's algorithm in time $\tilde{O}(\sqrt{C(f)})$, which would yield an approximating polynomial of similar degree, but this is not true for all functions f. To remedy this, we

construct a new function based on composing BKK with itself recursively $\log n$ times, to obtain a new function we call RECBKK, and show that certificates for RECBKK can be checked quickly by a quantum algorithm.

3.1.3 Summary of Separations

To conveniently express these results, we use a notion called the *critical exponent*, as defined by [36]. The the critical exponent for a measure M_1 relative to a measure M_2, denoted $\mathrm{crit}(M_1, M_2)$, is the infimum over all r such that the relation $M_1(f) = O(M_2(f)^r)$ holds for all total functions f. For example, because $D(f) \leq C(f)^2$ for all total f, we have $\mathrm{crit}(D, C) \leq 2$. Furthermore, since we also know that the AND-OR function on k^2 bits satisfies $D(f) = k^2$ and $C(f) = k$, we have $\mathrm{crit}(D, C) = 2$. Note that the critical exponent between M_1 and M_2 is 2 even if we have $M_1(f) = O(M_2(f)^2 \log M_2(f))$, or more generally if $M_1(f) \leq M_2(f)^{2+o(1)}$.

Table 3.2 lists the best known separations between these measures. A cell in the table has the form a, b, where $a \leq b$ are the best known lower and upper bounds on the critical exponent for the row measure relative to the column measure. The cell also contains a citation for the function that provides the lower bound. For example, in the cell corresponding to row D and column deg, we have the entry $2, 3$, which means $2 \leq \mathrm{crit}(D, \mathrm{deg}) \leq 3$, and a function achieving the separation appears in [39].

3.2 Randomized versus quantum query complexity

In this section we show a power 2.5 separation between bounded-error randomized query complexity, $R(f)$, and bounded-error quantum query complexity, $Q(f)$. In Section 3.2.1 we motivate the cheat sheet framework and provide a sketch of the separation. In Section 3.2.2 we formally prove the separation.

3.2.1 Intuition

We begin with the best known separation between randomized and quantum query complexity for *partial* functions, which is currently $\tilde{\Omega}(\sqrt{n})$ versus 1 provided by the Forrelation problem [2]. (Note that Simon's problem also provides a similar separation up to log factors [79].) Let g be the Forrelation function on $D \subseteq \{0,1\}^n$ with $R(g) = \tilde{\Omega}(\sqrt{n})$ and $Q(g) = 1$, although any function with these query complexities (up to log factors) would do.

One way to make g total would be to define it to be 0 on the rest of the domain $\{0,1\}^n \backslash D$. But then it is unclear if the new function g has low quantum query complexity, since the quantum algorithm would have to test whether the input lies in the promised set D. In general since partial function separations often require a stringent promise on the input, we may expect that it is necessary to examine most input bits to ascertain that $x \in D$.

Indeed, it is not even clear how to *certify* that $x \in D$ for our function g. On the other hand, the domain certification problem is trivial for total functions. So we can compose g with a total function h to obtain some of the desirable properties of both. We can certify that an input to $g \circ h$ lies in its domain by certifying the outputs of all instances of h. This means we want h to have low certificate complexity, and since we will use $g \circ h$ to separate randomized and quantum query complexity, we would also like h to have $Q(h)$ smaller than $R(h)$. A function that fits the bill perfectly is the AND-OR_{m^2} function that satisfies $R(h) = \Omega(m^2)$, $C(h) = m$, and $Q(h) = O(m)$.

We can now compute the various complexities of $f := g \circ h$. First we have $Q(f) = \tilde{O}(m)$, by composing algorithms for g and h. There also exists a certificate of size $\tilde{O}(nm)$ that proves that the input satisfies the promise, since we can certify the outputs of the n different h gates using certificates of size $C(h) = m$. Also, given query access to this certificate of size $\tilde{O}(nm)$, a quantum algorithm can check the certificate's validity using Grover search in $\tilde{O}(\sqrt{nm})$ queries. Hence a quantum algorithm with access to a certificate can solve the problem with $\tilde{O}(m + \sqrt{nm})$ queries. On the other hand, it can be shown that $R(f) = R(g \circ h) = \Omega(R(g)R(h)) = \tilde{\Omega}(\sqrt{n}m^2)$.

Now if we set $m = n$, then we see that $Q(f) = \tilde{O}(n)$ and $R(f) = \tilde{\Omega}(n^{2.5})$, but f is still a partial function. However, f has the desirable property that a quantum algorithm given query access to a certificate can decide whether the input satisfies the promise using $\tilde{O}(n)$ queries. But we cannot simply append the certificate to the input as we do not want the randomized algorithm to be able to use it. It is, however, acceptable if the randomized algorithm finds the certificate after it spends $R(f)$ queries, since that is the lower bound we want to prove.

What we would like is to hide the certificate somewhere in the input where only the quantum algorithm can find it. What information do we have that is only known to the quantum algorithm and remains unknown to the randomized algorithm unless it spends $R(f)$ queries? Clearly the value of f on the input has this property.

While one bit does not help much, we can obtain additional bits of this kind by using (say) $10 \log n$ copies of f. Now the answer string to these problems can address an array of size n^{10}, much larger than can be brute-force searched by a randomized algorithm in a reasonable amount of time. Furthermore, this address can be found by a quantum algorithm using only $\tilde{O}(Q(f))$ queries. At this location we will store a cheat sheet: a collection of certificates for all $10 \log n$ instances of f.

Now the new function, which we call f_{CS} (shown in Figure 3-1), evaluates to 1 if and only if the $10 \log n$ inputs are in the domain of f, and the outputs of the $10 \log n$ copies of f point to a valid cheat sheet, i.e., a set of valid certificates for the $10 \log n$ copies of f. This construction ensures that the quantum algorithm can find the cheat sheet using $\tilde{O}(Q(f)) = \tilde{O}(n)$ queries, and then verify it using $\tilde{O}(n)$ queries, which gives $Q(f_{\mathrm{CS}}) = \tilde{O}(n)$. On the other hand, we intuitively expect that the randomized algorithm cannot even find the cheat sheet unless it computes f by spending at least $R(f) = \tilde{\Omega}(n^{2.5})$ queries, which gives us the desired separation.

3.2.2 Implementation

In this section we prove Theorem 3.1, restated for convenience:

Theorem 3.1. *There exists a total function f such that $R(f) = \tilde{\Omega}(Q(f)^{2.5})$.*

Let g be the Forrelation function on $D \subseteq \{0,1\}^n$ with $R(g) = \tilde{\Omega}(\sqrt{n})$ and $Q(g) = O(1)$. Let $h : \{0,1\}^{m^2} \to \{0,1\}$ be the AND-OR$_{m^2}$ function, defined as $\mathrm{AND}_m \circ \mathrm{OR}_m$. This function satisfies $R(h) = \Omega(m^2)$, which can be shown using a variety of methods: it follows from the partition bound [46, Theorem 4] and also from our more general Theorem 3.5. We also have $C(h) = m$, since one 1 from each OR gate is a valid 1-certificate and an OR gate with all zero inputs is a valid 0-certificate. Lastly, $Q(h) = \tilde{O}(m)$ follows from simply composing quantum algorithms for AND and OR, and indeed $Q(h) = O(m)$ [45].

Let $f = g \circ h$ be a partial function on nm^2 bits. We have $Q(f) = O(m)$ by composition, since quantum algorithms can be composed in general without losing a log factor due to

error reduction [70, 58]. There also exists a certificate of size $\tilde{O}(nm)$ that proves that the input satisfies the promise, since we can certify the outputs of the n different h gates using $C(h) = m$ pointers to the relevant input bits. Since each pointer uses $O(\log n)$ bits, the certificate is of size $\tilde{O}(nm)$. Also note that a quantum algorithm with *query access* to this certificate can check its validity using Grover search in $\tilde{O}(\sqrt{nm})$ queries.

Lastly, we claim that $R(f) = R(g \circ h) = \Omega(R(g)R(h)) = \tilde{\Omega}(\sqrt{n}m^2)$. While a general composition theorem for bounded-error or zero-error randomized query complexity is unknown, we can show such a result when the inner function is the OR function.

Theorem 3.5. *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq \{0,1\}^n$. Then $R(f \circ \mathrm{OR}_m) = \Omega(mR(f))$ and $R_0(f \circ \mathrm{OR}_m) = \Omega(mR_0(f))$. Therefore $R(f \circ \mathrm{AND}_m \circ \mathrm{OR}_m) = \Omega(m^2R(f))$ and $R_0(f \circ \mathrm{AND}_m \circ \mathrm{OR}_m) = \Omega(m^2R_0(f))$.*

Proof. We prove this for bounded-error algorithms; the argument for zero-error algorithms will be almost identical. Let A be the best randomized algorithm for $f \circ \mathrm{OR}_m$. We convert A into an algorithm B for f, as follows.

Given input x to f, B generates n random inputs to OR_m, denoted $Y_1, Y_2, \ldots, Y_n$, in the following way. First, each Y_i is set to 0^m, the all-zero string. Next, a random entry of Y_i is chosen and replaced with a $*$ symbol. Finally, for each index i, the $*$ in Y_i will be replaced by the bit x_i. This causes $\mathrm{OR}_m(Y_i) = x_i$ to be true for all i.

To evaluate $f(x)$, the algorithm B can simply run A on the string $Y_1Y_2 \ldots Y_n$. Since

$$f \circ \mathrm{OR}_m(Y_1Y_2 \ldots Y_n) = f(\mathrm{OR}_m(Y_1)\mathrm{OR}_m(Y_2) \ldots \mathrm{OR}_m(Y_n)) = f(x_1x_2 \ldots x_n) = f(x), \quad (3.1)$$

this algorithm evaluates $f(x)$ with the same error as A. This process uses $R(f \circ \mathrm{OR}_m)$ queries to the input Y. However, not all of these queries require B to query x. In fact, B only needs to query x when algorithm A queries a bit that was formerly a $*$ in one of the Y_i. The expected number of queries B makes is therefore the expected number of $*$ entries found by the algorithm A. Using Markov's inequality, we can turn B into an $R(f)$ algorithm, which uses a fixed number of queries to calculate $f(x)$ with bounded error; it follows that the expected number of $*$ entries found by A is at least $\Omega(R(f))$.

We can now view A as a randomized algorithm that finds $\Omega(R(f))$ star bits (in expectation) given input in $Y_1Y_2 \ldots Y_n$. Set $Y = Y_i$ for a randomly chosen i. The number of queries A makes to Y must be exactly $R(f \circ \mathrm{OR}_m)/n$ in expectation. Let the probability that A finds k stars be p_k, for $k = 0, 1, \ldots, n$. Then

$$\sum_{k=0}^{n} kp_k = \Omega(R(f)). \quad (3.2)$$

For each k, the probability that A finds a star in Y given it found k stars in total is k/n. The overall probability that A finds a star in Y is therefore

$$\sum_{k=0}^{n} p_k(k/n) = \frac{1}{n}\sum_{k=0}^{n} kp_k = \Omega(R(f)/n). \quad (3.3)$$

In other words, A makes $R(f \circ \mathrm{OR}_m)/n$ expected queries to Y, and finds the $*$ in Y with probability $\Omega(R(f)/n)$. Now, finding the single $*$ in an otherwise all-zero string is an unordered search problem; the chance of solving this problem after T queries is at most T/m. In other words, any decision tree of height T has probability only T/m of solving

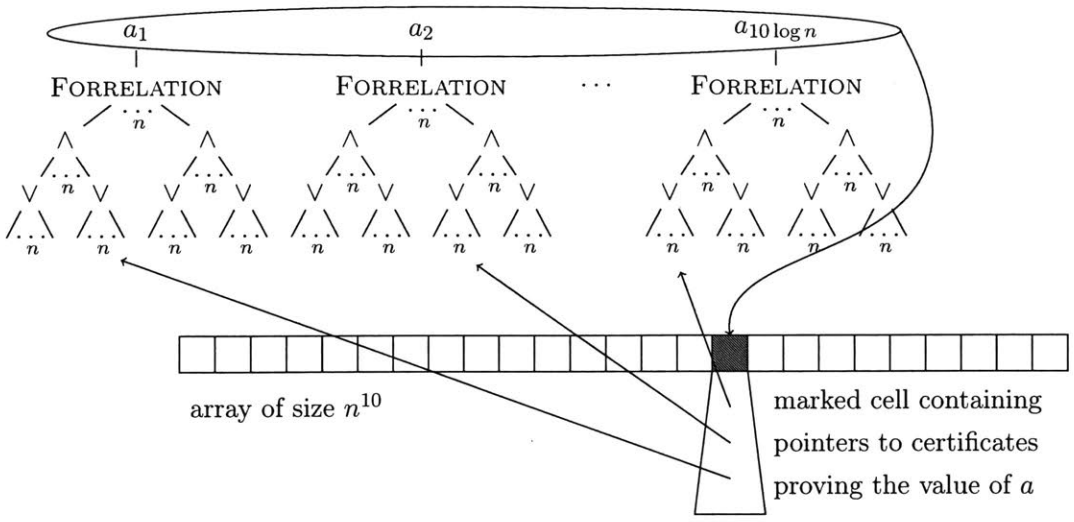


Figure 3-1: The function f that achieves a superquadratic separation between $Q(f) = \tilde{O}(n)$ and $R(f) = \tilde{\Omega}(n^{2.5})$.

the problem. Since A's querying strategy on Y can be written as a probability distribution over decision trees with expected height $R(f \circ \mathrm{OR}_m)/n$, it follows that the probability of A finding the $*$ is at most $R(f \circ \mathrm{OR}_m)/nm$. Thus we have $R(f \circ \mathrm{OR}_m)/nm = \Omega(R(f)/n)$, or $R(f \circ \mathrm{OR}_m) = \Omega(mR(f))$.

If A is a zero-error randomized algorithm instead of a bounded-error algorithm, the same argument follows verbatim, except that there's no longer a need to use Markov's inequality to conclude that the expected number of stars found by A is at least $\Omega(R_0(f))$.

The last part follows since we can show the same result for AND_m, using the fact that the query complexity of a function $f(x_1, \ldots, x_n)$ equals that of $f(\overline{x_1}, \ldots, \overline{x_n})$ and the query complexity of f and $\bar{f}$ is the same. □

We now define the cheat sheet version of f, which we call f_{CS}, depicted in Figure 3-1. Let the input to f_{CS} consist of $10 \log n$ inputs to f, each of size nm^2, followed by n^{10} blocks of bits of size $\tilde{O}(mn)$ each, which we refer to as the array of cells or array of cheat sheets. Each cell is large enough to hold certificates for all $10 \log n$ inputs to f that certify for each input the output of f evaluated on that input and that the promise holds.

Let us denote the input to f_{CS} as $z = (x^1, x^2, \ldots, x^{10 \log n}, Y_1, Y_2, \ldots, Y_{n^{10}})$, where x^i is an input to f, and the Y_i are the aforementioned cells of size $\tilde{O}(mn)$. We define the value of $f_{\mathrm{CS}}(z)$ to be 1 if and only if the following conditions hold:

1. For all i, x^i is in the domain of f. If this condition is satisfied, let ℓ be the positive integer corresponding to the binary string $(f(x^1), f(x^2), \ldots, f(x^{10 \log n}))$.
2. Y_ℓ certifies that all x^i are in the domain of f and that ℓ equals the binary string formed by their output values, $(f(x^1), f(x^2), \ldots, f(x^{10 \log n}))$.

We now upper bound the quantum query complexity of f_{CS}. The quantum algorithm starts by assuming that the first condition of f_{CS} holds and simply computes f on all $10 \log n$ inputs, which uses $\tilde{O}(Q(f)) = \tilde{O}(m)$ queries. The answers to these inputs points to the cheat sheet Y_ℓ, where ℓ is the integer corresponding to the binary string $(f(x^1), f(x^2), \ldots, f(x^{10 \log n}))$. As discussed, verifying the cheat sheet of size $\tilde{O}(mn)$ requires

only $\tilde{O}(\sqrt{mn})$ queries by a recursive application of Grover search. The algorithm outputs 1 if and only if the verification of Y_ℓ succeeded. If the certificate is accepted by the algorithm, then both conditions of f_{CS} are satisfied and hence it is easy to see the algorithm is correct. Hence $Q(f_{\mathrm{CS}}) = \tilde{O}(m + \sqrt{mn})$.

We also know that $R(f) = \tilde{\Omega}(\sqrt{n}m^2)$. We now prove that this implies $R(f_{\mathrm{CS}}) = \tilde{\Omega}(R(f)) = \tilde{\Omega}(\sqrt{n}m^2)$. Proving this completes the proof of Theorem 3.1, since setting $m = n$ immediately yields $Q(f_{\mathrm{CS}}) = \tilde{O}(n)$ and $R(f_{\mathrm{CS}}) = \tilde{\Omega}(n^{2.5})$.

To complete the proof, we show in general that $R(f_{\mathrm{CS}}) = \tilde{\Omega}(R(f))$.

Lemma 3.6. *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq [M]^n$, and let f_{CS} be the cheat sheet version of f with $c = 10 \log n$ copies of f. Then $R(f_{\mathrm{CS}}) = \Omega(R(f)/c^2) = \tilde{\Omega}(R(f))$.*

Proof. Let A be any bounded-error randomized algorithm for evaluating f_{CS}. We will prove that A makes $\Omega(R(f)/c^2)$ queries.

We start by proving the following claim: Let $x^1, x^2, \ldots, x^c \in \mathrm{Dom}(f)$ denote c valid inputs to f and let z be an input to f_{CS} consisting of $x^1 x^2 \cdots x^c$ with blank array—that is, it has 0 in all the entries of all cheat sheets. Let us assume that the all zero string is not a valid cheat sheet for any input. This can be enforced, for example, by requiring that all cheat sheets begin with the first bit equal to 1. Let ℓ be the binary number $f(x^1)f(x^2)\cdots f(x^c)$, and let h_ℓ be the ℓ^{th} cheat sheet. Then we claim that A must query a bit of h_ℓ when run on z with probability at least $1/3$.

This claim is easy to prove. Since the all zero string is an invalid cheat sheet, it follows that $f_{\mathrm{CS}}(z) = 0$, so A outputs 0 when run on z with probability at least $2/3$. Now if we modify h_ℓ in z to be the correct cheat sheet for the given input $x^1, x^2, \ldots, x^c$, then we obtain a new input z' with $f_{\mathrm{CS}}(z') = 1$. This means A outputs 1 when run on z' with probability at least $2/3$. However, since these inputs only differ on h_ℓ, if A does not query h_ℓ with high probability, it cannot distinguish input z from input z'. Since A is a valid algorithm for f_{CS}, the claim follows.

We now use the hybrid argument to prove the lemma. For any input z with blank array, let $p_z \in [0,1]^{2^c}$ be the vector such that for each $i = [2^c]$, $(p_z)_i$ is the probability that A makes a query to the i^{th} cheat sheet when given input z. Then the 1-norm of p_z is at most the running time of A, which is at most $R(f)$. By the claim, if ℓ is the relevant cheat sheet for z, we have $(p_z)_\ell \geq 1/3$. On the other hand, since p_z has $2^c \geq n^{10} \geq R(f)^{10}$ entries that sum to at most $R(f)$, almost all of them have value less than, say, $R(f)^{-5}$.

Next, consider hard distributions $\mathcal{D}^0$ and $\mathcal{D}^1$ over the 0- and 1-inputs to f, respectively. We pick these distributions such that distinguishing between them with probability at least $1/2 + 1/12c$ takes at least $\Omega(R(f)/c^2)$ queries for any randomized algorithm.

For each $i \in [2^c]$, let q_i be the expectation over p_z when z is made of c inputs to f generated from $\mathcal{D}^i = \mathcal{D}^{i_1} \times \mathcal{D}^{i_2} \times \cdots \times \mathcal{D}^{i_c}$, together with a blank array (here i_j means the j^{th} bit of i when written in binary). Then for all $i \in [2^c]$, we have $(q_i)_i \geq 1/3$, and the 1-norm of q_i is at most $R(f)$, so most entries of q_i are less than $R(f)^{-5}$. The entries of q_i can be interpreted as the probabilities of each cheat sheet being queried by the algorithm on an input sampled from $\mathcal{D}^i$.

Let $k \in [2^c]$ be such that $(q_0)_k < 1/6$. Let $k_0, k_1, \ldots, k_c$ be such that $k_0 = 0$, $k_c = k$, and consecutive k's differ by at most one bit (when represented as binary vectors of length c). Since $(q_{k_0})_k < 1/6$ and $(q_{k_c})_k \geq 1/3$, there must be some $j \in [c]$ such that $(q_{k_{j+1}})_k - (q_{k_j})_k > 1/6c$ and $(q_{k_j})_k < 1/3$. Let $a = (q_{k_{j+1}})_k$ and let $b = (q_{k_j})_k$.

We can now use this to distinguish the product distribution $\mathcal{D}^{k_{j+1}}$ from $\mathcal{D}^{k_j}$, with probability of error at most $1/2 - 1/12c$: simply run the algorithm A, output 1 if it queried an entry of the k^{th} cheat sheet, and otherwise output 1 with probability $\max(0, \frac{1-a-b}{2-a-b})$. If this maximum is 0, it means we have $b > 2/3$ and $a < 1/3$, so this algorithm has error at most $1/3$. Otherwise, it is not hard to check that the algorithm will determine if the input is from $\mathcal{D}^{k_{j+1}}$ with probability at most $1/2 - (b-a)/2 < 1/2 - 1/12c$.

Finally, note that since k_{j+1} and k_j differ in only one bit, distinguishing $\mathcal{D}^{k_{j+1}}$ from $\mathcal{D}^{k_j}$ allows us to distinguish between $\mathcal{D}^0$ and $\mathcal{D}^1$. This is because given any input from $\mathcal{D}^0$ or $\mathcal{D}^1$, the algorithm can sample other inputs to construct an input from $\mathcal{D}^{k_{j+1}}$ or $\mathcal{D}^{k_j}$, and then run the distinguishing algorithm. It follows that the running time of A is at least $\Omega(R(f)/c^2)$, by the choice of the distributions $\mathcal{D}^0$ and $\mathcal{D}^1$. □

3.3 Quantum Query Complexity versus Certificate Complexity and Degree

We now show a nearly quadratic separation between $Q(f)$ and $C(f)$, which yields a similar separation between $Q(f)$ and $\deg(f)$. We will also use the functions introduced in this section as building blocks to obtain the nearly 4^{th} power separation between $Q(f)$ and $\widetilde{\deg}(f)$ proved in Section 3.4.

3.3.1 Quadratic gap with certificate complexity

In this section we establish Theorem 3.3, restated for convenience:

Theorem 3.3. *There exists a total function f such that $Q(f) = \tilde{\Omega}(C(f)^2)$.*

Consider the k-SUM problem, k-SUM : $[M]^n \to \{0,1\}$, which asks if there are k elements in the input string $x_1, x_2, \ldots, x_n \in [M]$ that sum to 0 (mod M). Belovs and Špalek [18] showed that $Q(k\text{-SUM}) = \Omega(n^{k/(k+1)})$ when the alphabet M has size n^k and k is constant. In Appendix B, we show that their proof implies a bound of $Q(k\text{-SUM}) = \Omega(n^{k/(k+1)}/\sqrt{k})$ for super-constant k.

Now the 1-certificate complexity of the k-SUM problem is k (assuming it costs one query to get an element of M), since it suffices to provide the k elements that sum to 0 (mod M). If we take $k = \log n$, we get $Q(k\text{-SUM}) = \Omega(n/\sqrt{\log n})$ and $C_1(k\text{-SUM}) = O(\log n)$. Although this function is not Boolean, turning it into a Boolean function will only incur an additional polylogarithmic loss.

While k-SUM does not separate quantum query complexity from certificate complexity, its 1-certificate complexity is much smaller than its quantum query complexity. Composing a function with small 0-certificates, such as the AND_n with k-SUM already gives a function whose quantum query complexity is larger than its certificate complexity: in this case, we have certificate complexity $\tilde{O}(n)$ and quantum query complexity $\tilde{\Omega}(n^{3/2})$, which follows from the following general composition theorem [44, 70, 58, 51]:

Theorem 3.7 (Composition theorem for quantum query complexity). *Let $f : D \to \{0,1\}$ and $g : E \to \{0,1\}$ be partial functions where $D \subseteq \{0,1\}^n$ and $E \subseteq \{0,1\}^m$. Then $Q(f \circ g) = \Theta(Q(f)Q(g))$.*

To get an almost quadratic gap between $Q(f)$ and $C(f)$, we use a variant of k-SUM itself as the outer function instead of the AND function. From k-SUM, we define a new

Boolean function that we call BLOCK k-SUM, whose quantum query complexity is $\widetilde{\Theta}(n)$ and certificate complexity is also $\widetilde{\Theta}(n)$. However, although its certificate complexity is linear, the certificates consist almost exclusively of input bits set to 1 and only $\tilde{O}(1)$ input bits set to 0. This means if we compose this function with k-SUM, the composed function has certificates of size $\tilde{O}(n)$, since the certificates of BLOCK k-SUM are essentially composed of 1s, which are easy to certify for k-SUM. We denote this composed function $\mathrm{BKK} : \{0,1\}^{n^2} \to \{0,1\}$. It satisfies $C(\mathrm{BKK}) = \tilde{O}(n)$ and $Q(\mathrm{BKK}) = \tilde{\Omega}(n^2)$, which yields the desired quadratic separation.

We now define the BLOCK k-SUM problem.

Definition 3.8. *Let* BLOCK k-SUM *be a total Boolean function on n bits defined as follows. We split the input into blocks of size $10k \log n$ each and say a block is* balanced *if it has an equal number of 0s and 1s. Let the balanced blocks represent numbers in an alphabet M of size $\Omega(n^k)$. The value of the function is 1 if and only if there are k balanced blocks whose corresponding numbers sum to 0 (mod M) and all other blocks have at least as many 1s as 0s.*

We then compose this function with k-SUM to get the function BKK.

Definition 3.9. *Let* $\mathrm{BKK}_{n^2,k} : \{0,1\}^{n^2} \to \{0,1\}$ *be the function* BLOCK k-SUM *on n bits composed with a Boolean version of* k-SUM *on n bits, and define* BKK_{n^2} *to be* $\mathrm{BKK}_{n^2,k}$ *with* $k = \log n$.

We are now ready to establish the various complexities of BKK.

Theorem 3.10. *For the total Boolean function* $\mathrm{BKK}_{n^2,k} : \{0,1\}^{n^2} \to \{0,1\}$, *we have*

$$C(\mathrm{BKK}_{n^2,k}) = O(k^2 n \log n) \quad \textit{and} \quad Q(\mathrm{BKK}_{n^2,k}) = \Omega\Big(\frac{n^{2-2/(k+1)}}{k^3 \log^2 n}\Big). \tag{3.4}$$

Proof. We start by analyzing the certificates of BLOCK k-SUM. The key property we need is that every input of BLOCK k-SUM has a certificate that uses very few 0s, but can use a large number of 1s. To see this, note that we can certify a 1-input by showing the k balanced blocks that sum to 0 which requires $O(k^2 \log n)$ 0s, and all the 1s in every other block. There are two kinds of 0 inputs to certify: A 0-input that has a block with more 0s than 1s can be certified by providing that block, which only uses $O(k \log n)$ 0s. A 0-input in which all blocks have at least as many 1s as 0s can be certified by providing all the 1s: this provides the number represented by each block if it were balanced (though it does not prove the block is actually balanced), which is enough to check that no k of them sum to zero. In conclusion, BLOCK k-SUM can always be certified by providing $O(n)$ 1s and $O(k^2 \log n)$ 0s.

We now analyze the certificate complexity of $\mathrm{BKK}_{n^2,k}$. For each input, the outer BLOCK k-SUM has a certificate using $O(k^2 \log n)$ 0s and $O(n)$ 1s. The inner function, k-SUM, has 1-certificates of size $O(k^2 \log n)$ since there are k numbers to exhibit and each uses $k \log n$ bits when represented in binary, and has 0-certificates of size $O(n)$. Therefore, the composed function always has a certificate of size $O(k^2 n \log n)$. Hence $C(\mathrm{BKK}_{n^2,k}) = O(k^2 n \log n)$.

The quantum query complexity of BLOCK k-SUM is $\Omega(Q(k\text{-SUM})/k \log n)$ by a reduction from k-SUM. Using the result in Appendix B, this is $\Omega(n^{1-1/(k+1)}/k^{3/2} \log n)$. Invoking the composition theorem for quantum query complexity (Theorem 3.7), we get $Q(\mathrm{BKK}_{n^2,k}) = \Omega\left(\frac{n^{2-2/(k+1)}}{k^3 \log^2 n}\right)$. □

Thus for the function BKK : $\{0,1\}^{n^2} \to \{0,1\}$, defined as $\mathrm{BKK} = \mathrm{BKK}_{n^2,\log n}$, we have

$$C(\mathrm{BKK}) = O(n \log^3 n) = \tilde{O}(n) \quad \text{and} \quad Q(\mathrm{BKK}) = \Omega\Big(\frac{n^2}{\log^5 n}\Big) = \tilde{\Omega}(n^2). \tag{3.5}$$

This establishes Theorem 3.3, since BKK is a total Boolean function.

3.3.2 Quadratic gap with (exact) degree

We now show how to obtain a total function that nearly quadratically separates $Q(f)$ from $\deg(f)$ using any total function that achieves a similar separation between $Q(f)$ and $C(f)$. This proves Theorem 3.4:

Theorem 3.4. *There exists a total function f such that $Q(f) = \tilde{\Omega}(\deg(f)^2)$.*

Let $f : \{0,1\}^n \to \{0,1\}$ be a total function with $Q(f) = \tilde{\Omega}(n)$ and $C(f) = \tilde{O}(\sqrt{n})$, such as the BKK function introduced in the previous section. Let f_{CS} denote the cheat sheet version of f (as described in Section 3.2), created using $10 \log n$ copies of f that point to a cheat sheet among n^{10} potential cheat sheets, where a valid cheat sheet contains certificates of size $\tilde{O}(\sqrt{n})$ for each of the $10 \log n$ inputs to f and the binary string corresponding to their outputs equals the location of the cheat sheet. In this case f is a total function so the cheat sheets do not certify that the input satisfies the promise, but only the value of f evaluated on the input. We claim that the cheat sheet version of f satisfies $Q(f_{\mathrm{CS}}) = \tilde{\Omega}(n)$ and $\deg(f_{\mathrm{CS}}) = \tilde{O}(\sqrt{n})$.

Let us start with the degree upper bound, $\deg(f_{\mathrm{CS}}) = \tilde{O}(\sqrt{n})$. Let $\ell \in [n^{10}]$ be a potential location of the cheat sheet. For any ℓ, consider the problem of outputting 1 if and only if $f_{\mathrm{CS}}(z) = 1$ and ℓ is the location of the cheat sheet for the input z. Since ℓ is known, this can be solved by a deterministic algorithm $\mathcal{A}_\ell$ that makes $\tilde{O}(\sqrt{n})$ queries, since it can simply check if the certificate stored at cell ℓ in the array is valid: it can check all the $10 \log n$ certificates of size $\tilde{O}(\sqrt{n})$ for each of the $10 \log n$ instances of f and then check if the outputs of f evaluate to the location ℓ. Since polynomial degree is at most deterministic query complexity, we can construct a representing polynomial for $\mathcal{A}_\ell$ for any location ℓ. This is a polynomial p_ℓ of degree $\tilde{O}(\sqrt{n})$ such that $p_\ell(z) = 1$ if and only if $f_{\mathrm{CS}}(z) = 1$ and ℓ is the position of the cheat sheet on input z. Now we can simply add all the polynomials p_ℓ together to obtain a new polynomial q of the same degree. We claim $q(z) = f_{\mathrm{CS}}(z)$ since if $f_{\mathrm{CS}}(z) = 0$ then certainly all the polynomials $p_\ell(z) = 0$ (since none of the cheat sheets is valid) and if $f_{\mathrm{CS}}(z) = 1$ then $q(z) = 1$ because exactly one of many $p_\ell(z)$ will evaluate to 1, the one corresponding to the location of the cheat sheet for the input z. Note that the property used here is that in a 1-input to f_{CS}, exactly one location serves as the correct cheat sheet, i.e., the location of the cheat sheet for a 1-input is unique.

The claim that $Q(f_{\mathrm{CS}}) = \tilde{\Omega}(n)$ seems intuitive since $Q(f) = \tilde{\Omega}(n)$ and the cheat sheet version of f cannot be easier than f itself. This intuitive claim is true and we show below that $Q(f_{\mathrm{CS}}) = \Omega(Q(f))$ in general, which completes the proof of Theorem 3.4.

To prove this general result for quantum query complexity, we will need the following strong direct product theorem due to Lee and Roland [59].

Theorem 3.11. *Let f be a (partial) function with $Q_{1/4}(f) \geq T$. Then any T-query quantum algorithm that outputs the value of f evaluated on c independent instances has success probability at most $O((3/4)^{c/2})$.*

We now prove the lower bound on the quantum query complexity of cheat sheet functions.

Lemma 3.12. *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq [M]^n$, and let f_{CS} be a cheat sheet version of f with $c = 10 \log n$ copies of f. Then $Q(f_{\mathrm{CS}}) = \Omega(Q(f))$.*

Proof. By Theorem 3.11, we know that given $c = 10 \log n$ instances of f, any quantum algorithm that makes fewer than $T = Q_{1/4}(f)$ queries will correctly decide all the instances with probability at most $O((3/4)^{c/2})$.

Now suppose by way of contradiction that $Q_{1/4}(f_{\mathrm{CS}}) \leq T$. Then we will show how to decide c copies of f with probability $\Omega\left(1/T^2\right)$ by making T quantum queries. Since

$$\frac{1}{T^2} \geq \frac{1}{n^2} \geq \frac{1}{2^{c/5}} \gg \left(\frac{3}{4}\right)^{c/2}, \tag{3.6}$$

this is enough to contradict Theorem 3.11.

Let Q be a quantum algorithm that decides f_{CS} using at most T queries. Then consider running Q on an input $z = (x^1, \ldots, x^c, Y_1, Y_2, \ldots, Y_{2^c})$, where the x^i are in the domain of f and whose cheat-sheet array $(Y_1, Y_2, \ldots, Y_{2^c})$ has been completely zeroed out. We assume again that the all-zero cheat sheet is invalid for any input. This can always be enforced by requiring a valid cheat sheet to have the first bit set to 1.

For all $y \in [2^c]$ and $t \in [T]$, define $m_{y,t}$, or *the query magnitude on y at query t*, to be the probability that Q would be found querying some bit in cell Y_y, were we to measure in the standard basis immediately before the t^{th} query.

For an input z where we have zeroed out the cheat sheet array, clearly $f_{\mathrm{CS}}(z) = 0$ and hence Q outputs 0 on this input with high probability. On the other hand, if we let $\ell = f\left(x^1\right) \cdots f\left(x^c\right) \in [2^c]$, by modifying only the ℓ^{th} cell in the array, Y_ℓ, we could produce an input z' such that $f_{\mathrm{CS}}\left(z'\right) = 1$. From these facts, together with the standard BBBV hybrid argument [21], it follows that

$$\sqrt{m_{\ell,1}} + \cdots + \sqrt{m_{\ell,T}} = \Omega\left(1\right). \tag{3.7}$$

So by the Cauchy–Schwarz inequality, we have

$$m_{\ell,1} + \cdots + m_{\ell,T} = \Omega\left(\frac{1}{T}\right). \tag{3.8}$$

This implies that, if we simply choose a $t \in [T]$ uniformly at random, run Q until the t^{th} query with the cheat-sheet array zeroed out, and then measure in the standard basis, we will observe $\ell = f\left(x^1\right) \cdots f\left(x^c\right)$ with probability $\Omega\left(1/T^2\right)$. This completes the proof. □

3.4 Quantum query complexity versus approximate degree

We now show how to obtain a nearly quartic separation between quantum query complexity and approximate degree from a function that quadratically separates $Q(f)$ from $C(f)$. As in Section 3.2, we first motivate the construction in Section 3.4.1 and then formally give the separation in Section 3.4.2. The main result of this section is Theorem 3.2:

Theorem 3.2. *There exists a total function f such that $Q(f) \geq \widetilde{\deg}(f)^{4-o(1)}$.*

3.4.1 Intuition

To obtain the quartic separation with approximate degree, we could try the same approach as in the previous section for exact degree. Using notation from Section 3.3.2, consider the algorithm $\mathcal{A}_\ell$ that makes $\tilde{O}(\sqrt{n})$ queries. Instead of using a polynomial to exactly represent $\mathcal{A}_\ell$, we could try to construct more efficient approximating polynomials. If there were a quantum algorithm that checked the certificate quadratically faster than the deterministic $\mathcal{A}_\ell$, then we would be done. This is because such a quantum algorithm would yield a polynomial of degree $\tilde{O}(n^{1/4})$ that approximates $\mathcal{A}_\ell$. Then we could sum up these polynomials as before, with some error reduction to ensure that the error in each polynomial is small enough not to affect the sum of the polynomials. This error reduction only adds an overhead of $O(\log n)$, which gives us a polynomial of degree $\tilde{O}(n^{1/4})$ that approximates f_{CS}.

However, it is unclear if there is a quantum algorithm to check a certificate of size $\tilde{O}(\sqrt{n})$ quickly. Reading the certificate itself could require $\tilde{\Omega}(\sqrt{n})$ queries. All we know is that the certificate can be checked using $\tilde{O}(\sqrt{n})$ queries classically, but this does not imply a quadratic quantum speedup. To obtain a quadratic speedup the certificates need some structure that can, for example, be exploited with Grover's algorithm. But the certificates in this case are simply 0-inputs of k-SUM of size $\tilde{O}(\sqrt{n})$, and it is unclear how to quantumly check if an input is a 0-input (even with query access to any certificate) any quicker than querying the entire input. What we would like is for the certificate to be checkable using $\tilde{O}(n^{1/4})$ queries to the certificate and the input. So we construct a new function that has this property by modifying the BKK function used in Section 3.3.2.

Let f be BKK : $\{0,1\}^{n^2} \to \{0,1\}$ for some large, but constant n. We choose n to be large enough that $\text{Adv}(f) \geq n^{1.99}$ and $C(f) \leq n^{1.01}$. Such an n exists because asymptotically $C(\text{BKK}) = \tilde{O}(n)$ and $Q(\text{BKK}) = \tilde{\Omega}(n^2)$ (and $Q(\text{BKK}) = \Theta(\text{Adv}(\text{BKK}))$ [58]). Composing this function with itself d times gives us a new function $g : \{0,1\}^N \to \{0,1\}$, where $N = n^{2d}$. This function has $C(g) \leq C(f)^d \leq n^{1.01d} \leq N^{0.51}$. Since the general adversary bound satisfies a perfect composition theorem [58], we have $Q(g) = \Omega(\text{Adv}(g)) = \Omega(\text{Adv}(f)^d) = \Omega(n^{1.99d}) = \Omega(N^{0.99})$.

If we view the function g as a tree of depth d and fanin n^2, it has N leaves but only $N^{0.51}$ of these leaves are part of a certificate. Consider the subtree of g that consists of all nodes that are ancestors of these $N^{0.51}$ certificate leaves. This subtree has size $O(N^{0.51})$ and the set of values of all nodes in this subtree is a certificate for g. Furthermore, this certificate can be checked by a quantum algorithm in only $O(N^{0.26})$ queries, since each level of the tree consists of at most $N^{0.51}$ instances of f acting on a constant number of bits. Checking a constant-sized f costs only $O(1)$ queries, and searching over all $N^{0.51}$ instances for an invalid certificate costs $O(N^{0.26})$ queries by Grover's algorithm. This can be done for each level, resulting in a quantum algorithm with query complexity $\tilde{O}(N^{0.26})$.

Thus we have constructed a function g for which $Q(g)$ is close to N, but the complexity of quantumly checking a certificate (given query access to it) is close to $N^{1/4}$. Plugging this into the cheat sheet construction yields a nearly 4^{th} power separation between quantum query complexity and approximate degree.

3.4.2 Implementation

Recall the function BKK : $\{0,1\}^{n^2} \to \{0,1\}$ introduced in Theorem 3.10. We introduce a function we call RECBKK which consists of recursively composing BKK with itself d times; that is, we replace each bit in BKK with a new copy of BKK, then replace each bit in the

new copies with yet more copies of BKK, and so on. The resulting function will look like a tree of height d where each vertex has n^2 children and will have total input size $N = n^{2d}$. We will choose $d = (4/25)\log n/\log\log n$ to optimize our construction. The resulting function RECBKK has the following properties.

Theorem 3.13. *There exists a total function* RECBKK : $\{0,1\}^N \to \{0,1\}$ *such that given query access to a certificate of size* $N^{1/2+o(1)}$*, a quantum algorithm can check the validity of the certificate using at most* $N^{1/4+o(1)}$ *queries. Furthermore,* $Q(\text{RECBKK}) = N^{1-o(1)}$.

Proof. Our function RECBKK is defined as the d-fold composition of the function BKK : $\{0,1\}^{n^2} \to \{0,1\}$ with itself. This yields a function on $N = n^{2d}$ bits and we set $d = (4/25)\log n/\log\log n$.

With this choice of d, we can show more precisely that the function RECBKK has a certificate that can be checked by a quantum algorithm using $O(N^{1/4}L_N[1/2,1]) = N^{1/4+o(1)}$ queries, where $L_N[a,c] = \exp((c+o(1))\log^a N \log\log^{1-a} N)$, and also that $Q(\text{RECBKK}) = \Omega(N/L_N[1/2,1]) = N^{1-o(1)}$.

With this choice of d, the parameters are now related as follows: Since $N = n^{2d}$, we have $\log N = 2d\log n = (8/25)\log^2 n/\log\log n$. This gives $\log n = (5/4+o(1))\sqrt{\log N \log\log N}$, so we get $n = L_N[1/2,5/4]$. Additionally, since $(\log n)^d = \exp(d\log\log n) = n^{4/25}$, it follows that $(\log n)^d = L_N[1/2,1/5]$.

We start with the quantum lower bound for $Q(\text{RECBKK})$. By Theorem 3.10 and the optimality of the general adversary bound [58], we have

$$\text{Adv}(\text{BKK}) = \Omega(Q(\text{BKK})) = \Omega\left(\frac{n^2}{\log^5 n}\right) = \frac{n^2}{(\log n)^{5+o(1)}}. \tag{3.9}$$

By using the fact that $\text{Adv}(f^d) = \text{Adv}(f)^d$ for Boolean functions f [58], we get

$$Q(\text{RECBKK}) = \Omega(\text{Adv}(\text{BKK})^d) = \frac{n^{2d}}{(\log n)^{(5+o(1))d}} = \frac{N}{L_N[1/2,1/5]^{5+o(1)}}$$
$$= \Omega(N/L_N[1/2,1]).$$

Now we show the upper bound on quantumly checking a certificate. First note that every non-leaf node in the RECBKK tree corresponds to a BKK instance. For each such node, there is therefore a set of $C(\text{BKK})$ children that constitute a certificate for that BKK instance. We can therefore certify the RECBKK instance by starting from the top of the tree, listing out $C(\text{BKK})$ children that constitute a certificate, then listing out $C(\text{BKK})$ children for each of those, and so on. In each layer i of the tree, we thus have $C(\text{BKK})^i$ nodes that belong to a certificate.

We require our quantumly checkable certificate to provide, for each non-leaf node that belongs to a certificate, pointers to $C(\text{BKK})$ of the node's children that constitute a certificate for that BKK instance, starting with the root of the tree. A quantum algorithm can then use Grover search to search for a bad certificate. More precisely, the algorithm checks the certificate for the root to see if the $C(\text{BKK})$ children of the root pointed to and their claimed values do indeed form a certificate. It then checks if all the claimed values in the first level are correct assuming the claimed values of nodes in the second level and so on. The total number of certificates to check is

$$1 + C(\text{BKK}) + C(\text{BKK})^2 + \cdots + C(\text{BKK})^{d-1} \leq 2C(\text{BKK})^{d-1}, \tag{3.10}$$

where each certificate has size $C(\mathrm{BKK})$. Therefore, the quantum algorithm will make

$$\tilde{O}(C(\mathrm{BKK})^{(d-1)/2}C(\mathrm{BKK})) = \tilde{O}(C(\mathrm{BKK})^{(d+1)/2}) \tag{3.11}$$

queries, where we have logarithmic factors due to the pointers being encoded in binary and error reduction. From Theorem 3.10, we have $C(\mathrm{BKK}) = O(n\log^3 n) = n(\log n)^{3+o(1)}$, so the search takes

$$\tilde{O}(n^{(d+1)/2}(\log n)^{(3+o(1))(d+1)/2}) = \tilde{O}(N^{1/4}L_N[1/2, 37/40])$$

quantum queries, which is is $O(N^{1/4}L_N[1/2,1])$. □

Using this function RecBKK we can now establish Theorem 3.2. This proof is very similar to the separation between quantum query complexity and exact degree in Section 3.3.2.

Let f be the total function RecBKK : $\{0,1\}^n \to \{0,1\}$ with $Q(f) = n^{1-o(1)}$ and $C(f) = n^{1/2+o(1)}$, and more importantly given query access to this certificate, a quantum algorithm can check its validity using $n^{1/4+o(1)}$ queries. Let f_{CS} denote the cheat sheet version of f created using $10\log n$ copies of f as before. From Lemma 3.12 we know that $Q(f_{\mathrm{CS}}) = \Omega(Q(f))$ and hence $Q(f_{\mathrm{CS}}) = n^{1-o(1)}$. We now show $\widetilde{\deg}(f_{\mathrm{CS}}) = n^{1/4+o(1)}$.

Let $\ell \in [n^{10}]$ be a potential location of the cheat sheet. For any ℓ, consider the problem of outputting 1 if and only if $f_{\mathrm{CS}}(z) = 1$ and ℓ is the location of the cheat sheet for the input z. For any fixed ℓ, this can be solved by a quantum algorithm $\mathcal{Q}_\ell$ that makes $n^{1/4+o(1)}$ queries as shown in Theorem 3.13, since it can simply check if the certificate stored at cell ℓ in the array is valid: it can check all the $10\log n$ certificates for each of the $10\log n$ instances of f and then check if the outputs of f evaluate to the location ℓ.

Since approximate polynomial degree is at most quantum query complexity, we can construct a representing polynomial for $\mathcal{Q}_\ell$ for any location ℓ. This is a polynomial p_ℓ of degree $n^{1/4+o(1)}$ such that $p_\ell(z) = 1$ if and only if $f_{\mathrm{CS}}(z) = 1$ and ℓ is the position of the cheat sheet on input z. Now we can simply add all the polynomials p_ℓ together to obtain a new polynomial q of the same degree, except that we first reduce the error in each polynomial to below $1/n^{10}$ so that the total error is bounded. (This can be done, for example, using the amplification polynomial construction of [28, Lemma 1].) Now $q(z) = f_{\mathrm{CS}}(z)$ as argued in Section 3.3.2.

3.5 Cheat sheet functions

In this section we define the cheat sheet framework more generally. Once the framework is set up, proofs based on the framework are short and conveniently separate out facts about the framework from results about the separation under consideration. To demonstrate this, we reprove some known separations in Section 3.5.4.

3.5.1 Certifying functions and cheat sheets

Intuitively, a certifying function for a function $f : D \to \{0,1\}$, where $D \subseteq [M]^n$, is a function ϕ that takes in an input $x \in [M]^n$, a claimed value y_1 for $f(x)$, and a proof $(y_2, y_3, \ldots, y_k)$ that we indeed have $x \in D$ and $f(x) = y_1$. The value of the certifying function will be 1 only when these conditions hold. We would also like the certifying function to depend nontrivially on the certificate y, i.e., for every $x \in D$ there should be some y that makes the function output 1, and some y that makes it output 0. For convenience of analysis, we

enforce that the certificate $y = 0^k$ is invalid for all x. Any nontrivial certifying function can be made to have this property by requiring that (say) the second bit of y, y_2, is 1 for all valid certificates.

Definition 3.14 (Certifying function). *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq [M]^n$ for some integer $M \geq 2$. We say a total function $\phi : [M]^n \times \{0,1\}^k \to \{0,1\}$, where k is any positive integer, is a* certifying function *for f if the following conditions are satisfied:*

1. $\forall x \notin D$, $\phi(x,y) = 0$ *(invalid inputs should not have certificates)*
2. $\forall x \in [M]^n$, $\forall y \in \{0,1\}^k$, *if* $y_1 \neq f(x)$ *then* $\phi(x,y) = 0$ *(certificate asserts $y_1 = f(x)$)*
3. $\forall x \in D$, $\exists y \in \{0,1\}^k$ *such that* $\phi(x,y) = 1$ *(valid inputs should have certificates)*
4. $\forall x \in [M]^n$, $\phi(x,0^k) = 0$ *(nontriviality condition)*

Typically, a certifying function will be defined so that its value is only 1 if y includes pointers to a certificate in x and if f is a partial function, we will also want y to include a proof that x satisfies the promise of f. Thus the query complexity of ϕ may be smaller than that of f.

We now define the cheat sheet version of a function f with certifying function ϕ. In the separations in the previous sections we used $10 \log n$ copies of the function f to create the address of the cheat sheet. For generality we now use $c = \lceil 10 \log D(f) \rceil$ instead so that the construction only adds logarithmic factors in $D(f)$, as opposed to logarithmic factors in n, which may be larger than $D(f)$. This does not make a difference in our applications, however.

Definition 3.15 (Cheat sheet version of f). *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq [M]^n$ for some integer $M \geq 2$, and let $c = \lceil 10 \log D(f) \rceil$. Let $\phi : [M]^n \times \{0,1\}^k \to \{0,1\}$, for some positive integer k, be a certifying function for f. Then* the cheat sheet version of f with respect to ϕ, *denoted $f_{\mathrm{CS}(\phi)}$, is a total function*

$$f_{\mathrm{CS}(\phi)} : ([M]^n)^c \times (((\{0,1\}^k)^c)^{2^c} \to \{0,1\} \tag{3.12}$$

acting on an input $(X, Y_1, Y_2, \ldots, Y_{2^c})$, where $X \in ([M]^n)^c$ and $Y_i \in (\{0,1\}^k)^c$. Also, let $X \in ([M]^n)^c$ be written as $X = (x^1, x^2, \ldots, x^c)$, where $x^i \in [M]^n$. Then we define the value of $f_{\mathrm{CS}(\phi)}(X, Y_1, Y_2, \ldots Y_{2^c})$ to be 1 if and only if conditions (1) and (2) hold.

(1) For all $i \in [c]$, $x^i \in D$.

If condition (1) is satisfied, let $\ell \in [2^c]$ be the positive integer corresponding to the binary string $(f(x^1), f(x^2), \ldots, f(x^c))$ and let $Y_\ell = (y^1, y^2, \ldots y^c)$ where each $y^i \in \{0,1\}^k$.

(2) For all $i \in [c]$, $x^i \in D$ and $\phi(x^i, y^i) = 1$.

Intuitively, the input to $f_{\mathrm{CS}(\phi)}$ is interpreted as c inputs to f, which we denote by $(x^1, x^2, \ldots, x^c)$, followed by 2^c strings $(Y_1, Y_2, \ldots Y_{2^c})$ of length ck called cheat sheets. The value of the function will be 0 if any of the c inputs to f do not satisfy the promise of f. If they do satisfy the promise, let $\ell \in [2^c]$ be the number encoded by the binary string $f(x^1)f(x^2)\ldots f(x^c)$, where 0^c and 1^c encode the numbers 1 and 2^c respectively. If the ℓ^{th} cheat sheet in the input, Y_ℓ, is written as $Y_\ell = (y^1, y^2, \ldots, y^c)$, where $y^i \in \{0,1\}^k$ for all i, then the value of $f_{\mathrm{CS}(\phi)}$ is 1 if $\phi(x^i, y^i) = 1$ for all $i \in [c]$, and 0 otherwise.

In other words, the value of $f_{\mathrm{CS}(\phi)}$ is 1 if the first part of its input consists of c valid inputs to f that together encode a location of a cheat sheet in the second part of the input, and this cheat sheet in turn contains pointers that certify the values of the c inputs to f. The idea is that the only way to find the cheat sheet is to solve the c copies of f, so that $f_{\mathrm{CS}(\phi)}$ has query complexity at least that of f, but once the cheat sheet has been found one can verify the value of $f_{\mathrm{CS}(\phi)}$ using only the query complexity of ϕ, which may be much smaller.

We now characterize the query complexity of $f_{\mathrm{CS}(\phi)}$ in terms of the query complexities of f and ϕ in various models. The following result summarizes our results for the query complexity of cheat sheet functions.

Theorem 3.16 (Query complexity of cheat sheet functions). *Let $f : D \to \{0,1\}$, where $D \subseteq \{0,1\}^n$, be a partial function, and let ϕ be a certifying function for f. Then the following relations hold.*

- $D(f_{\mathrm{CS}(\phi)}) = \widetilde{\Theta}(D(f) + D(\phi))$
- $R(f_{\mathrm{CS}(\phi)}) = \widetilde{\Theta}(R(f) + R(\phi))$
- $Q(f_{\mathrm{CS}(\phi)}) = \widetilde{\Theta}(Q(f) + Q(\phi))$
- $\deg(f_{\mathrm{CS}(\phi)}) = \widetilde{\Theta}(\deg(\phi))$
- $\widetilde{\deg}(f_{\mathrm{CS}(\phi)}) = \widetilde{\Theta}(\widetilde{\deg}(\phi))$
- $R_0(f_{\mathrm{CS}(\phi)}) = \tilde{\Omega}(R_0(f) + R_0(\phi))$ *(but the corresponding upper bound relation is false)*
- $Q_E(f_{\mathrm{CS}(\phi)}) = \tilde{O}(Q_E(f) + Q_E(\phi))$ *(we conjecture that the corresponding lower bound holds)*

The algorithmic measures, D, R, and Q behave as expected since Theorem 3.16 asserts that the straightforward way to compute $f_{\mathrm{CS}(\phi)}$ in these models, which is to first compute all c copies of f and then check if they point to a valid cheat sheet, is essentially optimal. We conjecture that Q_E, the quantum analogue of D, should behave similarly to D or Q, but we can only prove the upper bound (see Lemma 3.17). In fact, if the lower bound for Q_E can be proved, then we would get a near cubic separation between $Q(f)$ and $Q_E(f)$.

For the zero-error class R_0, while we can show that the lower bound holds, the upper bound relation is false (see Theorem C.3 for a counterexample). We believe the zero-error class Q_0 behaves similarly: we conjecture the analogous lower bound holds for Q_0, but are unable to prove it. For Q_0, the corresponding upper bound is false, for reasons similar to those for R_0.

Notably, one-sided error measures do not behave well for cheat sheet functions at all. In Theorem C.2, we demonstrate f and ϕ such that $R_1(f_{\mathrm{CS}(\phi)}) \ll R_1(f)$, showing that the lower bound fails. The upper bound need not hold either.

Lastly, note that deg and $\widetilde{\deg}$ behave fundamentally differently on cheat sheet functions than the algorithmic measures. The (approximate) degree of the function f does not even play a role in determining the (approximate) degree of $f_{\mathrm{CS}(\phi)}$.

Theorem 3.16 is proved in the next two sections. In Appendix C we present some examples where the naïvely expected relations do not hold.

3.5.2 Upper bounds on the complexity of cheat sheet functions

We start by showing that the usual algorithmic models can compute $f_{\mathrm{CS}(\phi)}$ in roughly the number of queries they require for f and ϕ.

Lemma 3.17. *Let $f : D \to \{0,1\}$, where $D \subseteq [M]^n$, be a partial function, and ϕ be a certifying function for f. Then the following upper bounds hold.*

- $D(f_{\mathrm{CS}(\phi)}) = \tilde{O}(D(f) + D(\phi))$
- $R(f_{\mathrm{CS}(\phi)}) = \tilde{O}(R(f) + R(\phi))$
- $Q(f_{\mathrm{CS}(\phi)}) = \tilde{O}(Q(f) + Q(\phi))$
- $Q_E(f_{\mathrm{CS}(\phi)}) = \tilde{O}(Q_E(f) + Q_E(\phi))$

Proof. The algorithm to evaluate $f_{\mathrm{CS}(\phi)}$ is the following: First, evaluate f on the c inputs to f contained in z, assuming they satisfy the promise of f. If one of these inputs is discovered to contradict the promise of f, then output 0 and halt. This takes $cD(f)$ queries for a deterministic algorithm (and $cQ_E(f)$ queries for an exact quantum algorithm), $O(R(f)c\log c)$ for a bounded-error randomized algorithm, and $O(Q(f)c\log c)$ for a bounded-error quantum algorithm, where the factor of $O(\log c)$ comes from reducing the error enough to ensure that all c functions are computed correctly with high probability. From the c outputs we get a binary number ℓ. We can then go to the appropriate cheat sheet, and evaluate ϕ on all the (x^i, y^i) pairs and output 1 if they are all 1. This takes $cD(\phi)$ queries for a deterministic algorithm (and $cQ_E(\phi)$ queries for an exact quantum algorithm), $O(R(\phi)c\log c)$ queries for a randomized algorithm, and $O(Q(\phi)c\log c)$ queries for a quantum algorithm. Since $\phi(x,y) = 0$ when x does not satisfy the promise of f, this algorithm is correct. □

Before proving the upper bounds for the degree measures, we prove a technical lemma that shows that $\log(D(f))$ and $\log(C(\phi))$ or $\log(\widetilde{\deg}(\phi))$ are the same up to constants. We prove this to show that the log factors that appear in our construction really can be neglected compared to the measures we consider.

Lemma 3.18. *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq [M]^n$, and let ϕ be a certifying function for f. Then $D(f) \leq C^{(1)}(\phi)^2$ and $C(f) \leq C^{(1)}(\phi)$. As a consequence, we have $D(f) \leq D(\phi)^2$ and $C(f) \leq C(\phi)$.*

Proof. Note that every 1-certificate for $\phi(x,y)$ proves that $f(x) = y_1$. In particular, such a 1-certificate must reveal a certificate for f in x. This already shows that $C(f) \leq C^{(1)}(\phi)$.

Moreover, every 1-certificate of $\phi(x,y)$ where $y_1 = 0$ must conflict with every 1-certificate of $\phi(x,y)$ where $y_1 = 1$ on some bit in x. This is because otherwise, there would be some x in the promise of f that can be certified to be both a 0- and a 1-input for f.

We can then use the following strategy to compute $f(x)$. On input $x \in \mathrm{Dom}(f)$, pick a 1-certificate of some $\phi(x,y)$ with $y_1 = 0$, and query all of its entries in x. This reveals at least one bit of every 1-certificate of ϕ that has $y_1 = 1$. We then find another 1-certificate of some $\phi(x,y)$ where $y_1 = 0$, consistent with the inputs revealed in x so far. We once again reveal all of its entries in x. After $C^{(1)}(\phi)$ iterations of this process, we have either eliminated all 1-certificates of ϕ with $y_1 = 0$, or else all 1-certificates of ϕ with $y_1 = 1$. This tells us the value of $f(x)$, because since $x \in \mathrm{Dom}(f)$, there must be some y such that $\phi(x,y) = 1$, and hence there must be some 1-certificate of ϕ consistent with x. The total number of queries used was at most $C^{(1)}(\phi)^2$. □

Next, we show the upper bounds on the degree measures of $f_{\mathrm{CS}(\phi)}$ and show that they only depend on the degree of ϕ and not on any measure of the function f.

Lemma 3.19. *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq \{0,1\}^n$, and ϕ be a certifying function for f. Then $\deg(f_{\mathrm{CS}(\phi)}) = \tilde{O}(\deg(\phi))$ and $\widetilde{\deg}(f_{\mathrm{CS}(\phi)}) = \tilde{O}(\widetilde{\deg}(\phi))$.*

Proof. We start by showing this for $\deg(f_{\mathrm{CS}(\phi)})$. For each $\ell \in [2^c]$, we construct a polynomial which is 1 if $f_{\mathrm{CS}(\phi)}(z) = 1$ and the cheat sheet of z is the ℓ^{th} cheat sheet, and 0 otherwise. We can do this by simply multiplying together the polynomials of $\phi(x^i, y^i)$ for all $i \in [c]$, where x^i are the inputs to f in z and y^i are the entries found in the ℓ^{th} cheat sheet of z.

The resulting polynomial, p_ℓ, has degree $c\deg(\phi)$, is 1 on input z if z is a 1-input for $f_{\mathrm{CS}(\phi)}$ and ℓ is the location of the cheat sheet for input z, and is 0 on all other inputs. To get a polynomial for $f_{\mathrm{CS}(\phi)}$, we simply sum up the polynomials p_ℓ for all $\ell \in [2^c]$. The resulting polynomial is equal to $f_{\mathrm{CS}(\phi)}(z)$ on all inputs z, and has degree $c\deg(\phi)$.

The same trick works for $\widetilde{\deg}$, except in that case we cannot simply construct the AND of c polynomials by multiplying them together: we must first perform error reduction on the polynomials and map the output range $[0, 1/3] \cup [2/3, 1]$ to $[0, 1/3c] \cup [1 - 1/3c, 1]$. It is possible to do this amplification while increasing the degree of the polynomials by at most $O(\log c)$ using, for example, the amplification polynomial construction of [28, Lemma 1]. Therefore we have $\widetilde{\deg}(f_{\mathrm{CS}(\phi)}) = O(c\,\widetilde{\deg}(\phi)\log c) = \tilde{O}(\widetilde{\deg}(\phi))$.

Finally, note that c is at most logarithmic in $\widetilde{\deg}(\phi)$, since we have $c = O(\log D(f)) = O(\log C^{(1)}(\phi)) = O(\log C(\phi)) = O(\log \widetilde{\deg}(\phi))$, where the last equality follows from [26] and from the fact that ϕ is a total function. □

3.5.3 Lower bounds on the complexity of cheat sheet functions

Lemma 3.20. *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq \{0,1\}^n$, and let ϕ be a certifying function for f. Then the complexity of $f_{\mathrm{CS}(\phi)}$ is at least that of ϕ with respect to any complexity measure that does not increase upon learning input bits, which includes all reasonable complexity models such as Q, D, R, R_0, R_1, Q_1, Q_0, Q_E, C, RC, bs, $\deg$, and $\widetilde{\deg}$.*

Proof. Consider restricting $f_{\mathrm{CS}(\phi)}$ to a promise that guarantees that all cheat sheets of the input are identical. An input z to this promise problem evaluates to 1 if and only if the ϕ inputs from the first cheat sheet all evaluate to 1. We add the additional promise that this is true for all but the last input to ϕ. The value of an input z to this function then becomes exactly ϕ evaluated on a certain portion of the input. The complexity measures mentioned do not increase when the function is restricted to a promise. Thus the complexity of $f_{\mathrm{CS}(\phi)}$ under these measures is at least that of ϕ. □

Lemma 3.21. *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq [M]^n$, and let ϕ be a certifying function for f. Then $D(f_{\mathrm{CS}(\phi)}) = \Omega(D(f))$.*

Proof. We describe an adversary strategy that can be used to force a deterministic algorithm to make $\Omega(D(f))$ queries. Let $x^1, x^2, \ldots, x^c$ represent the c inputs to f. Whenever the algorithm queries a bit not in $x^1, x^2, \ldots, x^c$, the adversary returns 0. Whenever the algorithm queries a bit in x^i, the adversary uses the adversarial strategy for f on that input; that is, it returns answers that are consistent with the promise of f, but that force any deterministic algorithm to use $D(f)$ queries to discern $f(x^i)$.

Now if a deterministic algorithm uses fewer than $D(f)$ queries on an input to $f_{\mathrm{CS}(\phi)}$, then there must be many cheat sheets in the input that it did not query at all. In addition, such an algorithm cannot determine the value of $f(x^i)$ for any i. It follows that this algorithm could not have discerned the value of $f_{\mathrm{CS}(\phi)}$ on the given input, and thus $D(f_{\mathrm{CS}(\phi)}) \geq D(f)$. □

Lemma 3.22. *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq [M]^n$, and let ϕ be a certifying function for f. Then $R_0(f_{CS(\phi)}) = \Omega(R_0(f))$.*

Proof. Let A be a zero-error randomized algorithm for $f_{CS(\phi)}$. Then A must always find a certificate for $f_{CS(\phi)}$. Consider running A on an input z consisting of c 0-inputs to f, together with an all-zeros array. Certifying such an input requires presenting 0-certificates for some of the c inputs to f, and presenting at least one bit from each cheat sheet whose index has not been disproven by the 0-certificates of the f inputs.

Now, since there are at least $D(f)^{10}$ cheat sheets, only a small fraction can be provided in a certificate of size at most $O(R_0(f))$. It follows that the algorithm A must find certificates to at least half of the c 0-inputs to f with high probability.

We can now turn A into a new algorithm B that finds a certificate in a given 0-input to f sampled from the hard distribution $\mathcal{D}^0$ over the 0-inputs of f. Given such an input x, the algorithm B will generate $c-1$ additional inputs from $\mathcal{D}^0$, shuffle them together, and add an all-zero array. B will then feed this input to A. We know that with high probability, A will find a certificate for at least half the inputs to f, so it must find a certificate for x with probability close to $1/2$ (say, probability at least $1/3$). If it does not return a certificate, B can repeat this process. The expected running time of B is at most 3 times that of A.

Similarly, if we feed A an input consisting of c 1-inputs to f together with an all-zero array, we can use a similar argument to construct an algorithm C for finding 1-certificates in inputs sampled from $\mathcal{D}^1$ (the hard distribution over 1-inputs for f). We can then alternate running steps of B and steps of C to get an algorithm that finds a certificate in an input sampled from either $\mathcal{D}^0$ or $\mathcal{D}^1$, which uses an expected number of queries that is at most 6 times that of A.

This last algorithm evaluates f on inputs sampled from the hard distribution for f, and therefore must have expected running time at least $R_0(f)$. It follows that the expected running time of A is at least $R_0(f)/6$. □

The lower bounds for bounded-error randomized query complexity and bounded-error quantum query complexity are exactly as in Lemma 3.6 and Lemma 3.12.

3.5.4 Proofs of known results using cheat sheets

Theorem 3.23 ([9]). *There exists a total function f such that $R_0(f) = \tilde{\Omega}(Q(f)^3)$.*

Proof. This proof is similar to that of Theorem 3.1, except with a different function g. Let $g : D \to \{0,1\}$, where $D \in \{0,1\}^n$, be a partial function that satisfies $Q(g) = 1$ and $R_0(g) = \Omega(n)$, such as the Deutsch–Jozsa problem [31, 30]. $R_0(g) = \Omega(n)$ follows from the fact that even certifying that the input is all zeros or all ones requires a certificate of linear size.

Let $h : \{0,1\}^m \to \{0,1\}$ be the AND-OR function on m bits. Let $f = g \circ h$ and let ϕ be a certifying function that requires the obvious certificate for f (as used in the proof of Theorem 3.1). Then we have $Q(\phi) = O(n + \sqrt{n}m^{1/4})$ and $Q(f) = Q(g \circ h) = O(\sqrt{m})$. From Theorem 3.5 we have that $R_0(f) = R_0(g \circ h) = \tilde{\Omega}(nm)$. Using Theorem 3.16, we have $Q(f_{CS(\phi)}) = \tilde{O}(Q(f) + Q(\phi)) = \tilde{O}(\sqrt{m} + n + \sqrt{n}m^{1/4})$ and $R(f_{CS(\phi)}) = \tilde{\Omega}(R_0(f)) = \tilde{\Omega}(nm)$. Plugging in $m = n^2$, we get $Q(f_{CS(\phi)}) = \tilde{O}(n)$ and $R(f_{CS(\phi)}) = \tilde{\Omega}(n^3)$. □

Theorem 3.24 ([9]). *There exists a total function f such that $R(f) = \tilde{\Omega}(Q_E(f)^{3/2})$.*

Proof. Let $g : D \to \{0,1\}$, where $D = \{x \in \{0,1\}^n : |x| = 0 \text{ or } |x| = 1\}$, be the partial function that evaluates to 0 if the input is the all-zeros string and evaluates to 1 if the input string has Hamming weight 1. This function satisfies $R(g) = \Omega(n)$, since the block sensitivity of the 0-input is linear and $Q_E(g) = O(\sqrt{n})$, since Grover's algorithm can be made exact in this case [25].

Let $h : \{0,1\}^m \to \{0,1\}$ be the AND function on m bits, which satisfies $R(h) = \Omega(m)$ and $Q_E(h) = O(m)$. Let $f = g \circ h$, and let ϕ be a certifying function for f. The certificate complexity of this function is $O(n)$ in the no case (by showing one 0-input to each of the n AND gates), and is $O(m)$ in the yes case (by showing all the 1 inputs to an AND gate). We choose the certifying function ϕ that requires this certificate for f and hence $Q_E(\phi) \leq D(\phi) = \tilde{O}(n+m)$.

Now by composing quantum algorithms for g and h we have $Q_E(f) = O(m\sqrt{n})$, and by Theorem 3.5 we have $R(f) = \Omega(mn)$. Hence using Theorem 3.16, we have $Q_E(f_{\mathrm{CS}(\phi)}) = \tilde{O}(Q_E(f) + Q_E(\phi)) = \tilde{O}(n + m + m\sqrt{n})$, and $R(f_{\mathrm{CS}(\phi)}) = \tilde{\Omega}(mn)$. Choosing $m = \sqrt{n}$ gives $Q_E(f_{\mathrm{CS}(\phi)}) = \tilde{O}(n)$ and $R(f_{\mathrm{CS}(\phi)}) = \tilde{\Omega}(n^{3/2})$. □

Chapter 4

Randomized Sabotage Complexity

4.1 Introduction

This chapter is based on work that appeared in [20], which is joint work with Robin Kothari.. We introduce a lower bound technique for randomized query algorithms that we call randomized sabotage complexity, and use it to prove composition theorems for randomized query complexity.

4.1.1 Composition theorems

A basic structural question that can be asked in any model of computation is whether there can be resource savings when computing the same function on several independent inputs. We say a direct sum theorem holds in a model of computation if solving a problem on n independent inputs requires roughly n times the resources needed to solve one instance of the problem. Direct sum theorems hold for deterministic and randomized query complexity [47], fail for circuit size [68], and remain open for communication complexity [50, 14, 34].

More generally, instead of merely outputting the n answers, we could compute another function of these n answers. If f is an n-bit Boolean function and g is an m-bit Boolean function, we define the composed function $f \circ g$ to be an nm-bit Boolean function such that $f \circ g(x_1, \ldots, x_n) = f(g(x_1), \ldots, g(x_n))$, where each x_i is an m-bit string. The composition question now asks if there can be significant savings in computing $f \circ g$ compared to simply running the best algorithm for f and using the best algorithm for g to evaluate the input bits needed to compute f. If we let f be the identity function on n bits that just outputs all its inputs, we recover the direct sum problem.

Composition theorems are harder to prove and are known for only a handful of models, such as deterministic [83, 63] and quantum query complexity [70, 58, 51]. More precisely, let $\mathrm{D}(f)$, $\mathrm{R}(f)$, and $\mathrm{Q}(f)$ denote the deterministic, randomized, and quantum query complexities of f. Then for all (possibly partial) Boolean[1] functions f and g, we have

$$\mathrm{D}(f \circ g) = \mathrm{D}(f)\,\mathrm{D}(g) \qquad \text{and} \qquad \mathrm{Q}(f \circ g) = \Theta(\mathrm{Q}(f)\,\mathrm{Q}(g)). \tag{4.1}$$

In contrast, in the randomized setting we only have the upper bound direction, $\mathrm{R}(f \circ g) = O(\mathrm{R}(f)\,\mathrm{R}(g)\log \mathrm{R}(f))$. Proving a composition theorem for randomized query complexity remains a major open problem.

[1] Composition theorems usually fail for trivial reasons for non-Boolean functions. Hence we restrict our attention to Boolean functions, which have domain $\{0,1\}^n$ (or a subset of $\{0,1\}^n$) and range $\{0,1\}$.

Open Problem 1. Does it hold that $\mathrm{R}(f \circ g) = \Omega(\mathrm{R}(f)\,\mathrm{R}(g))$ for all Boolean functions f and g?

In this chapter we prove something close to a composition theorem for randomized query complexity. While we cannot rule out the possibility of synergistic savings in computing $f \circ g$, we show that a composition theorem does hold if we insert a small gadget in between f and g to obfuscate the output of g. Our gadget is "small" in the sense that its randomized (and even deterministic) query complexity is $\Theta(\log \mathrm{R}(g))$. Specifically we choose the index function, which on an input of size $k+2^k$ interprets the first k bits as an address into the next 2^k bits and outputs the bit stored at that address. The index function's query complexity is $k+1$ and we choose $k = \Theta(\log \mathrm{R}(g))$.

Theorem 4.1. *Let f and g be (partial) Boolean functions and let* IND *be the index function with* $\mathrm{R}(\textsc{Ind}) = \Theta(\log \mathrm{R}(g))$. *Then*

$$\mathrm{R}(f \circ \textsc{Ind} \circ g) = \Omega(\mathrm{R}(f)\,\mathrm{R}(\textsc{Ind})\,\mathrm{R}(g)) = \Omega(\mathrm{R}(f)\,\mathrm{R}(g)\log \mathrm{R}(g)).$$

Theorem 4.1 can be used instead of a true composition theorem in many applications. For example, in Chapter 3 a composition theorem for randomized query complexity was needed in the special case when g is the AND function. Our composition theorem would suffice for this application, since the separation shown there only changes by a logarithmic factor if an index gadget is inserted between f and g.

We prove Theorem 4.1 by introducing a new lower bound technique for randomized query complexity. This is not surprising since the composition theorems for deterministic and quantum query complexities are also proved using powerful lower bound techniques for these models, namely the adversary argument and the negative-weights adversary bound [44] respectively.

4.1.2 Sabotage complexity

To describe the new lower bound technique, consider the problem of computing a Boolean function f on an input $x \in \{0,1\}^n$ in the query model. In this model we have access to an oracle, which when queried with an index $i \in [n]$ responds with $x_i \in \{0,1\}$.

Imagine that a hypothetical saboteur damages the oracle and makes some of the input bits unreadable. For these input bits the oracle simply responds with a $*$. We can now view the oracle as storing a string $p \in \{0,1,*\}^n$ as opposed to a string $x \in \{0,1\}^n$. Although it is not possible to determine the true input x from the oracle string p, it may still be possible to compute $f(x)$ if all input strings consistent with p evaluate to the same f value. On the other hand, it is not possible to compute $f(x)$ if p is consistent with a 0-input and a 1-input to f. We call such a string $p \in \{0,1,*\}^n$ a *sabotaged input.* For example, let f be the OR function that computes the logical OR of its bits. Then $p = 00{*}0$ is a sabotaged input since it is consistent with the 0-input 0000 and the 1-input 0010. However, $p = 01{*}0$ is not a sabotaged input since it is only consistent with 1-inputs to f.

Now consider a new problem in which the input is promised to be sabotaged (with respect to a function f) and our job is to find the location of a $*$. Intuitively, any algorithm that solves the original problem f when run on a sabotaged input must discover at least one $*$, since otherwise it would answer the same on 0- and 1-inputs consistent with the sabotaged input. Thus the problem of finding a $*$ in a sabotaged input is no harder than the problem of computing f, and hence naturally yields a lower bound on the complexity of computing f. As we show later, this intuition can be formalized in several models of computation.

As it stands the problem of finding a $*$ in a sabotaged input has multiple valid outputs, as the location of any star in the input is a valid output. For convenience we define a decision version of the problem as follows: Imagine there are two saboteurs and one of them has sabotaged our input. The first saboteur, Asterix, replaces input bits with an asterisk ($*$) and the second, Obelix, uses an obelisk ($\dagger$). Promised that the input has been sabotaged exclusively by one of Asterix or Obelix, our job is to identify the saboteur. This is now a decision problem since there are only two valid outputs. We call this decision problem f_{sab}, the *sabotage problem* associated with f.

We now define lower bound measures for various models using f_{sab}. For example, we can define the *deterministic sabotage complexity of* f as $\text{DS}(f) := \text{D}(f_{\text{sab}})$ and in fact, it turns out that for all f, $\text{DS}(f)$ equals $\text{D}(f)$ (Theorem 4.29).

We could define the *randomized sabotage complexity of* f as $\text{R}(f_{\text{sab}})$, but instead we define it as $\text{RS}(f) := \text{R}_0(f_{\text{sab}})$, where R_0 denotes zero-error randomized query complexity, since $\text{R}(f_{\text{sab}})$ and $\text{R}_0(f_{\text{sab}})$ are equal up to constant factors (Theorem 4.7). $\text{RS}(f)$ has the following desirable properties.

1. (Lower bound for R)	For all f, $\text{R}(f) = \Omega(\text{RS}(f))$	(Theorem 4.8)
2. (Perfect composition)	For all f and g, $\text{RS}(f \circ g) \geq \text{RS}(f)\,\text{RS}(g)$	(Theorem 4.15)
3. (Composition with R)	For all f and g, $\text{R}(f \circ g) = \Omega(\text{R}(f)\,\text{RS}(g))$	(Theorem 4.17)
4. (Separated from prt(f))	$\exists$ total f with $\text{RS}(f) \geq \text{prt}(f)^{2-o(1)}$	(Theorem 4.26)
5. (Separated from Q(f))	$\exists$ total f with $\text{RS}(f) = \tilde{\Omega}(\text{Q}(f)^{2.5})$	(Theorem 4.26)
6. (Quadratically tight)	For all total f, $\text{R}(f) = O(\text{RS}(f)^2 \log \text{RS}(f))$	(Theorem 4.28)

Here $\text{prt}(f)$ denotes the partition bound [46, 48], which subsumes most other lower bound techniques such as approximate polynomial degree, randomized certificate complexity, block sensitivity, etc. The only general lower bound technique not subsumed by $\text{prt}(f)$ is quantum query complexity, $\text{Q}(f)$, which can also be considerably smaller than $\text{RS}(f)$ for some functions. In fact, we are unaware of any total function f for which $\text{RS}(f) = o(\text{R}(f))$, leaving open the intriguing possibility that this lower bound technique captures randomized query complexity for total functions.

4.1.3 Lifting theorems

Using randomized sabotage complexity we are also able to show a relationship between lifting theorems in communication complexity. A lifting theorem relates the query complexity of a function f with the communication complexity of a related function created from f. Recently, Göös, Pitassi, and Watson [39] showed that there is a communication problem G_{IND}, also known as the two-party index gadget, with communication complexity $\Theta(\log n)$ such that for any function f on n bits, $\text{D}^{\text{cc}}(f \circ G_{\text{IND}}) = \Omega(\text{D}(f) \log n)$, where $\text{D}^{\text{cc}}(F)$ denotes the deterministic communication complexity of a communication problem F.

Analogous lifting theorems are known for some complexity measures, but no such theorem is known for either zero-error randomized or bounded-error randomized query complexity. Our second result shows that a lifting theorem for zero-error randomized query complexity implies one for bounded-error randomized query. We use $\text{R}_0^{\text{cc}}(F)$ and $\text{R}^{\text{cc}}(F)$ to denote the zero-error and bounded-error communication complexities of F respectively.

Theorem 4.2. *Let $G : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ be a communication problem with the property that $\min\{|\mathcal{X}|, |\mathcal{Y}|\} = O(\log n)$. If it holds that for all n-bit partial functions f,*

$$\mathrm{R}_0^{\mathrm{cc}}(f \circ G) = \Omega(\mathrm{R}_0(f)/\operatorname{polylog} n), \tag{4.2}$$

then for all n-bit partial functions f,

$$\mathrm{R}^{\mathrm{cc}}(f \circ G_{\mathrm{IND}}) = \Omega(\mathrm{R}(f)/\operatorname{polylog} n), \tag{4.3}$$

where $G_{\mathrm{IND}} : \{0,1\}^b \times \{0,1\}^{2^b} \to \{0,1\}$ is the index gadget (Definition 4.24) with $b = \Theta(\log n)$.

Proving a lifting theorem for bounded-error randomized query complexity was an important open problem in communication complexity. Such a theorem allows the recent separations in communication complexity shown by Anshu et al. [12] to be proved simply by establishing their query complexity analogues, which was done in [4] and [11]. Our result shows that it is sufficient to prove a lifting theorem for zero-error randomized protocols instead.

Interestingly, a lifting theorem for both zero-error and bounded-error randomized algorithms was recently shown [40, 13]. This makes our lifting theorem reduction less interesting. However, the lifting theorems shown use logarithmic sized gadgets, so it is conceivable that our reduction will come in useful for proving a randomized lifting theorem with a constant sized gadget.

4.1.4 Open problems

The main open problem is to determine whether $\mathrm{R}(f) = \widetilde{\Theta}(\mathrm{RS}(f))$ for all total functions f. This is known to be false for partial functions, however. Any partial function where all inputs in $\mathrm{Dom}(f)$ are far apart in Hamming distance necessarily has low sabotage complexity. For example, any sabotaged input to the collision problem[2] has at least half the bits sabotaged making $\mathrm{RS}(f) = O(1)$, but $\mathrm{R}(f) = \Omega(\sqrt{n})$.

It would also be interesting to extend the sabotage idea to other models of computation and see if it yields useful lower bound measures. For example, we can define quantum sabotage complexity as $\mathrm{QS}(f) := \mathrm{Q}(f_{\mathrm{sab}})$, but we were unable to show that it lower bounds $\mathrm{Q}(f)$.

4.2 Basic Properties of Randomized Algorithms

We will assume familiarity with the following basic properties of randomized algorithms, some of which were already mentioned in Chapter 2. For completeness, we prove these properties in Appendix A. We restate them here for convenience.

First, we have Markov's inequality, which allows us to convert an algorithm with a guarantee on the expected number of queries into an algorithm with a guarantee on the maximum number of queries with a constant factor loss in the query bound and a constant factor increase in the error. This can be used, for example, to convert zero-error randomized algorithms into bounded-error randomized algorithms.

[2] In the collision problem, we are given an input $x \in [m]^n$, and we have to decide if x viewed as a function from $[n]$ to $[m]$ is 1-to-1 or 2-to-1 promised that one of these holds.

Lemma 2.12 (Markov's Inequality). *Let R be a randomized decision tree on n bits, and let $\delta \in (0,1)$. On any input $x \in \{0,1\}^n$, the probability that $h(D,x) \leq \lfloor \overline{h}(R,x)/\delta \rfloor$ when D is sampled from R is at least $1-\delta$.*

The next property allows us to amplify the success probability of an ϵ-error randomized algorithm.

Lemma 2.10 (Amplification). *If f is a Boolean function and R is a randomized algorithm for f with error $\epsilon < 1/2$, repeating R several times and taking the majority vote of the outcomes decreases the error. To reach error $\epsilon' > 0$, it suffices to repeat the algorithm $\frac{2\ln(1/\epsilon')}{(1-2\epsilon)^2}$ times. In particular, if $0 < \epsilon' < \epsilon < 1/2$, we have*

$$\mathrm{R}_\epsilon(f) \leq \mathrm{R}_{\epsilon'}(f) \leq \frac{2\ln(1/\epsilon')}{(1-2\epsilon)^2}\,\mathrm{R}_\epsilon(f).$$

Recall that we defined $\overline{\mathrm{R}}_\epsilon(f)$ to be the minimum expected number of queries made by a randomized algorithm that computes f with error probability at most ϵ. Clearly, we have $\overline{\mathrm{R}}_\epsilon(f) \leq \mathrm{R}_\epsilon(f)$, since the expected number of queries made by an algorithm is at most the maximum number of queries made by the algorithm. Using Lemma 2.12, we can now relate them in the other direction.

Lemma 4.3. *Let f be a partial function, $\delta > 0$, and $\epsilon \in [0, 1/2)$. Then we have $\mathrm{R}_{\epsilon+\delta}(f) \leq \frac{1-2\epsilon}{2\delta}\overline{\mathrm{R}}_\epsilon(f) \leq \frac{1}{2\delta}\overline{\mathrm{R}}_\epsilon(f)$.*

The next lemma shows how to relate these measures with the same error ϵ on both sides of the inequality. This also shows that $\overline{\mathrm{R}}_\epsilon(f)$ is only a constant factor away from $\mathrm{R}_\epsilon(f)$ for constant ϵ.

Lemma 2.13. *If f is a (possibly partial) Boolean function, then for all $\epsilon \in (0, \frac{1}{2})$, we have $\mathrm{R}_\epsilon(f) \leq 14\frac{\ln(1/\epsilon)}{(1-2\epsilon)^2}\overline{\mathrm{R}}_\epsilon(f)$. When $\epsilon = \frac{1}{3}$, we can improve this to $\mathrm{R}(f) \leq 10\overline{\mathrm{R}}(f)$.*

Although these measures are closely related for constant error, $\overline{\mathrm{R}}_\epsilon(f)$ is more convenient than $\mathrm{R}_\epsilon(f)$ for discussing composition and direct sum theorems.

We can also convert randomized algorithms that find certificates with bounded error into zero-error randomized algorithms.

Lemma 4.4. *Let R be a randomized algorithm that uses T queries on expectation and finds a certificate with probability $1-\epsilon$. Then repeating A when it fails to find a certificate turns it into an algorithm that always finds a certificate (i.e., a zero-error algorithm) that uses at most $T/(1-\epsilon)$ queries in expectation.*

Finally, the following lemma is useful for proving lower bounds on randomized algorithms.

Lemma 2.28. *Let f be a (possibly partial) function, let $x \in \mathrm{Dom}(f)$ be an input, and let B be a sensitive block of x with respect to f.*

Then any randomized algorithm that solves f using at most T expected queries and with error at most ϵ must, when run on x, query a bit in B with probability at least $1-2\epsilon$.

4.3 Sabotage complexity

We now formally define sabotage complexity. Given a (partial or total) n-bit Boolean function f, let $P_f \subseteq \{0,1,*\}^n$ be the set of all partial assignments of f that are consistent with both a 0-input and a 1-input. That is, for each $p \in P_f$, there exist $x, y \in \mathrm{Dom}(f)$ such that $f(x) \neq f(y)$ and $x_i = y_i = p_i$ whenever $p_i \neq *$. Let $P_f^\dagger \subseteq \{0,1,\dagger\}^n$ be the same as P_f, except using the symbol $\dagger$ instead of $*$. Observe that P_f and $P_f^\dagger$ are disjoint. Let $Q_f = P_f \cup P_f^\dagger \subseteq \{0,1,*,\dagger\}^n$. We then define f_{sab} as follows.

Definition 4.5. *Let f be an n-bit partial function. We define $f_{\mathrm{sab}} : Q_f \to \{0,1\}$ as $f_{\mathrm{sab}}(q) = 0$ if $q \in P_f$ and $f_{\mathrm{sab}}(q) = 1$ if $q \in P_f^\dagger$.*

Note that even when f is a total function, f_{sab} is always a partial function. See Section 4.1.2 for more discussion and motivation for this definition. Now that we have defined f_{sab}, we can define deterministic and randomized sabotage complexity.

Definition 4.6. *Let f be a partial function. Then* $\mathrm{DS}(f) := \mathrm{D}(f_{\mathrm{sab}})$ *and* $\mathrm{RS}(f) := \mathrm{R}_0(f_{\mathrm{sab}})$.

We will primarily focus on $\mathrm{RS}(f)$ in this work and only discuss $\mathrm{DS}(f)$ in Section 4.8. To justify defining $\mathrm{RS}(f)$ as $\mathrm{R}_0(f_{\mathrm{sab}})$ instead of $\mathrm{R}(f_{\mathrm{sab}})$ (or $\overline{\mathrm{R}}(f_{\mathrm{sab}})$), we now show these definitions are equivalent up to constant factors.

Theorem 4.7. *Let f be a partial function. Then* $\mathrm{R}_0(f_{\mathrm{sab}}) \geq \overline{\mathrm{R}}_\epsilon(f_{\mathrm{sab}}) \geq (1-2\epsilon)\,\mathrm{R}_0(f_{\mathrm{sab}})$.

Proof. The first inequality follows trivially. For the second, let $x \in Q_f$ be any valid input to f_{sab}. Let x' be the input x with asterisks replaced with obelisks and vice versa. Then since $f_{\mathrm{sab}}(x) \neq f_{\mathrm{sab}}(x')$, by Lemma 2.28 any ϵ-error randomized algorithm that solves f_{sab} must find a position on which x and x' differ with probability at least $1-2\epsilon$. The positions at which they differ are either asterisks or obelisks. Since x was an arbitrary input, the algorithm must always find an asterisk or obelisk with probability at least $1-2\epsilon$. Since finding an asterisk or obelisk is a certificate for f_{sab}, by Lemma 4.4, we get a zero-error algorithm for f_{sab} that uses $\overline{\mathrm{R}}_\epsilon(f_{\mathrm{sab}})/(1-2\epsilon)$ expected queries. Thus $\mathrm{R}_0(f_{\mathrm{sab}}) \leq \overline{\mathrm{R}}_\epsilon(f_{\mathrm{sab}})/(1-2\epsilon)$, as desired. □

Finally, we prove that $\mathrm{RS}(f)$ is indeed a lower bound on $\mathrm{R}(f)$, i.e., $\mathrm{R}(f) = \Omega(\mathrm{RS}(f))$.

Theorem 4.8. *Let f be a partial function. Then* $\mathrm{R}_\epsilon(f) \geq \overline{\mathrm{R}}_\epsilon(f) \geq (1-2\epsilon)\,\mathrm{RS}(f)$.

Proof. Let A be a randomized algorithm for f that uses $\overline{\mathrm{R}}_\epsilon(f)$ randomized queries and outputs the correct answer on every input in $\mathrm{Dom}(f)$ with probability at least $1-\epsilon$. Now fix a sabotaged input x, and let p be the probability that A finds a $*$ or $\dagger$ when run on x. Let q be the probability that A outputs 0 if it doesn't find a $*$ or $\dagger$ when run on x. Let x_0 and x_1 be inputs consistent with x such that $f(x_0) = 0$ and $f(x_1) = 1$. Then A outputs 0 on x_1 with probability at least $q(1-p)$, and A outputs 1 on x_0 with probability at least $(1-q)(1-p)$. These are both errors, so we have $q(1-p) \leq \epsilon$ and $(1-q)(1-p) \leq \epsilon$. Summing them gives $1-p \leq 2\epsilon$, or $p \geq 1-2\epsilon$.

This means A finds a $*$ entry within $\overline{\mathrm{R}}_\epsilon(f)$ expected queries with probability at least $1-2\epsilon$. By Lemma 4.4, we get $\frac{1}{1-2\epsilon}\overline{\mathrm{R}}_\epsilon(f) \geq \mathrm{RS}(f)$, or $\overline{\mathrm{R}}_\epsilon(f) \geq (1-2\epsilon)\,\mathrm{RS}(f)$. □

We also define a variant of RS where the number of asterisks (or obelisks) is limited to one. Specifically, let $U \subseteq \{0,1,*,\dagger\}^n$ be the set of all partial assignments with exactly one $*$ or $\dagger$. Formally, $U := \{x \in \{0,1,*,\dagger\}^n : |\{i \in [n] : x_i \notin \{0,1\}\}| = 1\}$.

Definition 4.9. *Let f be an n-bit partial function. We define f_{usab} as the restriction of f_{sab} to U, the set of strings with only one asterisk or obelisk. That is, f_{usab} has domain $Q_f \cap U$, but is equal to f_{sab} on its domain. We then define $\mathrm{RS_u}(f) := \mathrm{R_0}(f_{\text{usab}})$. If $Q_f \cap U$ is empty, we define $\mathrm{RS_u}(f) := 0$.*

The measure $\mathrm{RS_u}$ will play a key role in our lifting result in Section 4.6. Since f_{usab} is a restriction of f_{sab} to a promise, it is clear that its zero-error randomized query complexity cannot increase, and so $\mathrm{RS_u}(f) \leq \mathrm{RS}(f)$. Interestingly, when f is total, $\mathrm{RS_u}(f)$ equals $\mathrm{RS}(f)$. In other words, when f is total, we may assume without loss of generality that its sabotaged version has only one asterisk or obelisk.

Theorem 4.10. *If f is a total function, then $\mathrm{RS_u}(f) = \mathrm{RS}(f)$.*

Proof. We already argued that $\mathrm{RS}(f) \geq \mathrm{RS_u}(f)$. To show $\mathrm{RS_u}(f) \geq \mathrm{RS}(f)$, we argue that any zero-error algorithm A for f_{usab} also solves f_{sab}. The main observation we need is that any input to f_{sab} can be completed to an input to f_{usab} by replacing some asterisks or obelisks with 0s and 1s. To see this, let x be an input to f_{sab}. Without loss of generality, $x \in P_f$. Then there are two strings $y, z \in \mathrm{Dom}(f)$ that are consistent with x, satisfying $f(y) = 0$ and $f(z) = 1$.

The strings y and z disagree on some set of bits B, and x has a $*$ or $\dagger$ on all of B. Consider starting with y and flipping the bits of B one by one, until we reach the string z. At the beginning, we have $f(y) = 0$, and at the end, we reach $f(z) = 1$. This means that at some point in the middle, we must have flipped a bit that flipped the string from a 0-input to a 1-input. Let w_0 and w_1 be the inputs where this happens. They differ in only one bit. If we replace that bit with $*$ or $\dagger$, we get a partial assignment w consistent with both, so $w \in P_f$. Moreover, w is consistent with x. This means we have completed an arbitrary input to f_{sab} to an input to f_{usab}, as claimed.

The algorithm A, which correctly solves f_{usab}, when run on w (a valid input to f_{usab}) must find an asterisk or obelisk in w. Now consider running A on the input x to f_{sab} and compare its execution to when it is run on w. If A ever queries a position that is different in x and w, then it has found an asterisk or obelisk and the algorithm can now halt. If not, then it must find the single asterisk or obelisk present in w, which is also present in x. This shows that the slightly modified version of A that halts if it queries an asterisk or obelisk solves f_{sab} and hence $\mathrm{RS}(f) = \mathrm{R_0}(f_{\text{sab}}) \leq \mathrm{R_0}(f_{\text{usab}}) = \mathrm{RS_u}(f)$. □

4.4 Direct sum and composition theorems

In this section, we establish the main composition theorems for randomized sabotage complexity. To do so, we first need to establish direct sum theorems for the problem f_{sab}. In fact, our direct sum theorems hold more generally for zero-error randomized query complexity of partial functions (and even relations). To prove this, we will require Yao's minimax theorem [89].

Theorem 4.11 (Minimax). *Let f be an n-bit partial function. There is a distribution μ over inputs in $\mathrm{Dom}(f)$ such that all zero-error algorithms for f use at least $\mathrm{R_0}(f)$ expected queries on μ.*

We call any distribution μ that satisfies the assertion in Yao's theorem a *hard distribution* for f.

4.4.1 Direct sum theorems

We start by defining the m-fold direct sum of a function f, which is simply the function that accepts m inputs to f and outputs f evaluated on all of them.

Definition 4.12. *Let $f : \mathrm{Dom}(f) \to \mathcal{Z}$, where $\mathrm{Dom}(f) \subseteq \mathcal{X}^n$, be a partial function with input and output alphabets $\mathcal{X}$ and $\mathcal{Z}$. The m-fold direct sum of f is the partial function $f^{\oplus m} : \mathrm{Dom}(f)^m \to \mathcal{Z}^m$ such that for any $(x_1, x_2, \ldots, x_m) \in \mathrm{Dom}(f)^m$, we have*

$$f^{\oplus m}(x_1, x_2, \ldots, x_m) = (f(x_1), f(x_2), \ldots, f(x_m)). \tag{4.4}$$

We can now prove a direct sum theorem for zero-error randomized and more generally ϵ-error expected randomized query complexity, although we only require the result about zero-error algorithms. We prove these results for partial functions, but they also hold for arbitrary relations.

Theorem 4.13 (Direct sum)**.** *For any n-bit partial function f and any positive integer m, we have $\mathrm{R}_0(f^{\oplus m}) = m\,\mathrm{R}_0(f)$. Moreover, if μ is a hard distribution for f given by Theorem 4.11, then $\mu^{\otimes m}$ is a hard distribution for $f^{\oplus m}$. Similarly, for ϵ-error randomized algorithms we get $\overline{\mathrm{R}}_\epsilon(f^{\oplus m}) \geq m\overline{\mathrm{R}}_\epsilon(f)$.*

Proof. The upper bound follows from running the $\mathrm{R}_0(f)$ algorithm on each of the m inputs to f. By linearity of expectation, this algorithm solves all m inputs after $m\,\mathrm{R}_0(f)$ expected queries.

We now prove the lower bound. Let A be a zero-error randomized algorithm for $f^{\oplus m}$ that uses T expected queries when run on inputs from $\mu^{\otimes m}$. We convert A into an algorithm B for f that uses T/m expected queries when run on inputs from μ.

Given an input $x \sim \mu$, the algorithm B generates $m-1$ additional "fake" inputs from μ. B then shuffles these together with x, and runs A on the result. The input to A is then distributed according to $\mu^{\otimes m}$, so A uses T queries (in expectation) to solve all m inputs. B then reads the solution to the true input x.

Note that most of the queries A makes are to fake inputs, so they don't count as real queries. The only real queries B has to make happen when A queries x. But since x is shuffled with the other (indistinguishable) inputs, the expected number of queries A makes to x is the same as the expected number of queries A makes to each fake input; this must equal T/m. Thus B makes T/m queries to x (in expectation) before solving it.

Since B is a zero-error randomized algorithm for f that uses T/m expected queries on inputs from μ, we must have $T/m \geq \mathrm{R}_0(f)$ by Theorem 4.11. Thus $T \geq m\,\mathrm{R}_0(f)$, as desired.

The same lower bound proof carries through for ϵ-error expected query complexity, $\overline{\mathrm{R}}_\epsilon(f)$, as long as we use a version of Yao's theorem for this model. For completeness, we prove this version of Yao's theorem in Appendix A. □

Theorem 4.13 is essentially [47, Theorem 2], but our theorem statement looks different since we deal with expected query complexity instead of worst-case query complexity. From Theorem 4.13, we can also prove a direct sum theorem for worst-case randomized query complexity since for $\epsilon \in (0, 1/2)$,

$$\mathrm{R}_\epsilon(f^{\oplus m}) \geq \overline{\mathrm{R}}_\epsilon(f^{\oplus m}) \geq m\overline{\mathrm{R}}_\epsilon(f) \geq 2\delta m\,\mathrm{R}_{\epsilon+\delta}(f), \tag{4.5}$$

for any $\delta > 0$, where the last inequality used Lemma 4.3.

For our applications, however, we will need a strengthened version of this theorem, which we call a threshold direct sum theorem.

Theorem 4.14 (Threshold direct sum). *Given an input to $f^{\oplus m}$ sampled from $\mu^{\otimes m}$, we consider solving only some of the m inputs to f. We say an input x to f is solved if a z-certificate was queried that proves $f(x) = z$. Then any randomized algorithm that takes an expected T queries and solves an expected k of the m inputs when run on inputs from $\mu^{\otimes m}$ must satisfy $T \geq k \, \mathrm{R}_0(f)$.*

Proof. We prove this by a reduction to Theorem 4.13. Let A be a randomized algorithm that, when run on an input from $\mu^{\otimes m}$, solves an expected k of the m instances, and halts after an expected T queries. We note that these expectations average over both the distribution $\mu^{\otimes m}$ and the internal randomness of A.

We now define a randomized algorithm B that solves the m-fold direct sum $f^{\oplus m}$ with zero error. B works as follows: given an input to $f^{\oplus m}$, B first runs A on that input. Then B checks which of the m instances of f were solved by A (by seeing if a certificate proving the value of f was found for a given instance of f). B then runs the optimal zero-error algorithm for f, which makes $\mathrm{R}_0(f)$ expected queries, on the instances of f that were not solved by A.

Let us examine the expected number of queries used by B on an input from $\mu^{\otimes m}$. Recall that a randomized algorithm is a probability distribution over deterministic algorithms; we can therefore think of A as a distribution. For a deterministic algorithm $D \sim A$ and an input x to $f^{\oplus m}$, we use $D(x)$ to denote the number of queries used by D on x, and $S(D, x) \subseteq [m]$ to denote the set of inputs to f the algorithm D solves when run on x. Then by assumption

$$T = \mathop{\mathbb{E}}_{x \sim \mu^{\otimes m}} \mathop{\mathbb{E}}_{D \sim A} D(x) \qquad \text{and} \qquad k = \mathop{\mathbb{E}}_{x \sim \mu^{\otimes m}} \mathop{\mathbb{E}}_{D \sim A} |S(D, x)|. \tag{4.6}$$

Next, let R be the randomized algorithm that uses $\mathrm{R}_0(f)$ expected queries and solves f on any input. For an input x to $f^{\oplus m}$, we write $x = x_1 x_2 \ldots x_m$ with $x_i \in \mathrm{Dom}(f)$. Then the expected number of queries used by B on input from $\mu^{\otimes m}$ can be written as

$$\mathop{\mathbb{E}}_{x \sim \mu^{\otimes m}} \mathop{\mathbb{E}}_{D \sim A} \left(D(x) + \mathop{\mathbb{E}}_{D_1 \sim R} \mathop{\mathbb{E}}_{D_2 \sim R} \cdots \mathop{\mathbb{E}}_{D_m \sim R} \sum_{i \in [m] \setminus S(D,x)} D_i(x_i) \right) \tag{4.7}$$

$$= \mathop{\mathbb{E}}_{x \sim \mu^{\otimes m}} \mathop{\mathbb{E}}_{D \sim A} \left(D(x) + \sum_{i \in [m] \setminus S(D,x)} \mathop{\mathbb{E}}_{D_i \sim R} D_i(x_i) \right) \tag{4.8}$$

$$\leq \mathop{\mathbb{E}}_{x \sim \mu^{\otimes m}} \mathop{\mathbb{E}}_{D \sim A} \left(D(x) + \sum_{i \in [m] \setminus S(D,x)} \mathrm{R}_0(f) \right) \tag{4.9}$$

$$= \mathop{\mathbb{E}}_{x \sim \mu^{\otimes m}} \mathop{\mathbb{E}}_{D \sim A} \left(D(x) + (m - |S(D, x)|) \, \mathrm{R}_0(f) \right) \tag{4.10}$$

$$= T + (m - k) \, \mathrm{R}_0(f). \tag{4.11}$$

Since B solves the direct sum problem on $\mu^{\otimes m}$, the expected number of queries it uses is at least $m \, \mathrm{R}_0(f)$ by Theorem 4.13. Hence $T + (m - k) \, \mathrm{R}_0(f) \geq m \, \mathrm{R}_0(f)$, so $T \geq k \, \mathrm{R}_0(f)$. □

4.4.2 Composition theorems

Using the direct sum and threshold direct sum theorems we have established, we can now prove composition theorems for randomized sabotage complexity. We start with the behavior of RS itself under composition.

Theorem 4.15. *Let f and g be partial functions. Then* $\mathrm{RS}(f \circ g) \geq \mathrm{RS}(f)\,\mathrm{RS}(g)$.

Proof. Let A be any zero-error algorithm for $(f \circ g)_{\mathrm{sab}}$, and let T be the expected query complexity of A (maximized over all inputs). We turn A into a zero-error algorithm B for f_{sab}.

B takes a sabotaged input x for f. It then runs A on a sabotaged input to $f \circ g$ constructed as follows. Each 0 bit of x is replaced with a 0-input to g, each 1 bit of x is replaced with a 1-input to g, and each $*$ or † of x is replaced with a sabotaged input to g. The sabotaged inputs are generated from μ, the hard distribution for g_{sab} obtained from Theorem 4.11. The 0-inputs are generated by first generating a sabotaged input, and then selecting a 0-input consistent with that sabotaged input. The 1-inputs are generated analogously.

This is implemented in the following way. On input x, the algorithm B generates n sabotaged inputs from μ (the hard distribution for g_{sab}), where n is the length of the string x. Call these inputs $y_1, y_2, \ldots, y_n$. B then runs the algorithm A on this collection of n strings, pretending that it is an input to $f \circ g$, with the following caveat: whenever A tries to query a $*$ or † in an input y_i, B instead queries x_i. If x_i is 0, B selects an input from $f^{-1}(0)$ consistent with y_i, and replaces y_i with this input. It then returns to A an answer consistent with the new y_i. If x_i is 1, B selects a consistent input from $f^{-1}(1)$ instead. If x_i is a $*$ or †, B returns a $*$ or † respectively.

Now B only makes queries to x when it finds a $*$ or † in an input to g_{sab}. But this solves that instance of g_{sab}, which was drawn from the hard distribution for g_{sab}. Thus the query complexity of B is upper bounded by the number of instances of g_{sab} that can be solved by a T-query algorithm with access to n instances of g_{sab}. We know from Theorem 4.14 that if A makes T expected queries, the expected number of $*$ or † entries it finds among $y_1, y_2, \ldots, y_n$ is at most $T/\mathrm{RS}(g)$. Hence the expected number of queries B makes to x is at most $T/\mathrm{RS}(g)$. Thus we have $\mathrm{RS}(f) \leq T/\mathrm{RS}(g)$, which gives $T \geq \mathrm{RS}(f)\,\mathrm{RS}(g)$. □

Using this we can lower bound the randomized query complexity of composed functions. In the following, f^n denotes the function f composed with itself n times, i.e., $f^1 = f$ and $f^{i+1} = f \circ f^i$.

Corollary 4.16. *Let f be a partial function. Then* $\mathrm{R}(f^n) \geq \mathrm{RS}(f)^n/3$.

This follows straightforwardly from observing that $\mathrm{R}(f^n) = \mathrm{R}_{1/3}(f^n) \geq (1-2/3)\,\mathrm{RS}(f^n)$ (using Theorem 4.8) and $\mathrm{RS}(f^n) \geq \mathrm{RS}(f)^n$ (using Theorem 4.15).

We can also prove a composition theorem for zero-error and bounded-error randomized query complexity in terms of randomized sabotage complexity. In particular this yields a composition theorem for $\mathrm{R}(f \circ g)$ when $\mathrm{R}(g) = \Theta(\mathrm{RS}(g))$.

Theorem 4.17. *Let f and g be partial functions. Then* $\overline{\mathrm{R}}_\epsilon(f \circ g) \geq \overline{\mathrm{R}}_\epsilon(f)\,\mathrm{RS}(g)$.

Proof. The proof follows a similar argument to the proof of Theorem 4.15. Let A be a randomized algorithm for $f \circ g$ that uses T expected queries and makes error ϵ. We turn A into an algorithm B for f by having B generate inputs from μ, the hard distribution for

g_{sab}, and feeding them to A, as before. The only difference is that this time, the input x to B is not a sabotaged input. This means it has no $*$ or $\dagger$ entries, so all the sabotaged inputs that B generates turn into 0- or 1-inputs if A tries to query a $*$ or $\dagger$ in them.

Since A uses T queries, by Theorem 4.14, it finds at most $T/\operatorname{RS}(g)$ asterisks or obelisks (in expectation). Therefore, B makes at most $T/\operatorname{RS}(g)$ expected queries to x. Since B is correct whenever A is correct, its error probability is at most ϵ. Thus $\overline{\mathrm{R}}_\epsilon(f) \leq T/\operatorname{RS}(g)$, and thus $T \geq \overline{\mathrm{R}}_\epsilon(f)\operatorname{RS}(g)$. □

Setting ϵ to 0 yields the following corollary.

Corollary 4.18. *Let f and g be partial functions. Then* $\mathrm{R}_0(f \circ g) \geq \mathrm{R}_0(f)\operatorname{RS}(g)$.

For the more commonly used $\mathrm{R}(f \circ g)$, we obtain the following composition result.

Corollary 4.19. *Let f and g be partial functions. Then* $\mathrm{R}(f \circ g) \geq \mathrm{R}(f)\operatorname{RS}(g)/10$.

This follows from Lemma 2.13, which gives $\overline{\mathrm{R}}_{1/3}(f) \geq \mathrm{R}(f)/10$, and Theorem 4.17, since $\mathrm{R}(f \circ g) \geq \overline{\mathrm{R}}_{1/3}(f \circ g) \geq \overline{\mathrm{R}}_{1/3}(f)\operatorname{RS}(g) \geq \mathrm{R}(f)\operatorname{RS}(g)/10$.

Finally, we can also show an upper bound composition result for randomized sabotage complexity.

Theorem 4.20. *Let f and g be partial functions. Then* $\operatorname{RS}(f \circ g) \leq \operatorname{RS}(f)\mathrm{R}_0(g)$. *We also have* $\operatorname{RS}(f \circ g) = O(\operatorname{RS}(f)\mathrm{R}(g)\log\operatorname{RS}(f))$.

Proof. We describe a simple algorithm for finding a $*$ or $\dagger$ in an input to $f \circ g$. Start by running the optimal algorithm for the sabotage problem of f. This algorithm uses $\operatorname{RS}(f)$ expected queries. Then whenever this algorithm tries to query a bit, run the optimal zero-error algorithm for g in the corresponding input to g.

Now, since the input to $f \circ g$ that we are given is a sabotaged input, it must be consistent with both a 0-input and a 1-input of $f \circ g$. It follows that some of the g inputs are sabotaged, and moreover, if we represent a sabotaged g-input by $*$ or $\dagger$, a 0-input to g by 0, and a 1-input to g by 1, we get a sabotaged input to f. In other words, from the inputs to g we can derive a sabotaged input for f.

This means that the outer algorithm runs uses an expected $\operatorname{RS}(f)$ calls to the inner algorithm, and ends up calling the inner algorithm on a sabotaged input to g. Meanwhile, each call to the inner algorithm uses an expected $\mathrm{R}_0(g)$ queries, and will necessarily find a $*$ or $\dagger$ if the input it is run on is sabotaged. Therefore, the described algorithm will always find a $*$ or $\dagger$, and its expected running time is $\operatorname{RS}(f)\mathrm{R}_0(g)$ by linearity of expectation and by the independence of the internal randomness of the two algorithms.

Instead of using a zero-error randomized algorithm for g, we can use a bounded-error randomized algorithm for g as long as its error probability is small. Since we make $O(\operatorname{RS}(f))$ calls to the inner algorithm, if we boost the bounded-error algorithm's success probability to make the error much smaller than $1/\operatorname{RS}(f)$ (costing an additional $\log\operatorname{RS}(f)$ factor), we will get a bounded-error algorithm for $(f \circ g)_{\text{sab}}$. Since $\mathrm{R}((f \circ g)_{\text{sab}})$ is the same as $\operatorname{RS}(f \circ g)$ up to a constant factor (Theorem 4.7),

$$\operatorname{RS}(f \circ g) = O(\operatorname{RS}(f)\mathrm{R}(g)\log\operatorname{RS}(f)), \tag{4.12}$$

as desired. □

4.5 Composition with the index function

We now prove our main result (Theorem 4.1) restated more precisely as follows.

Theorem 4.1 (Precise version). *Let f and g be (partial) functions, and let $m = \Omega(\mathrm{R}(g)^{1.1})$. Then $\mathrm{R}(f \circ \textsc{Ind}_m \circ g) = \Omega(\mathrm{R}(f)\,\mathrm{R}(g)\log m) = \Omega(\mathrm{R}(f)\,\mathrm{R}(\textsc{Ind}_m)\,\mathrm{R}(g))$.*

Before proving this, we formally define the index function.

Definition 4.21 (Index function). *The index function on m bits, denoted $\textsc{Ind}_m : \{0,1\}^m \to \{0,1\}$, is defined as follows. Let c be the largest integer such that $c + 2^c \le m$. For any input $x \in \{0,1\}^m$, let y be the first c bits of x and let $z = z_0 z_1 \cdots z_{2^c-1}$ be the next 2^c bits of x. If we interpret y as the binary representation of an integer between 0 and $2^c - 1$, then the output of $\textsc{Ind}_m(x)$ equals z_y.*

To prove Theorem 4.1, we also require the strong direct product theorem for randomized query complexity that was established by Drucker [32].

Theorem 4.22 (Strong direct product). *Let f be a partial Boolean function, and let k be a positive integer. Then any randomized algorithm for $f^{\oplus k}$ that uses at most $\gamma^3 k\,\mathrm{R}(f)/11$ queries has success probability at most $(1/2 + \gamma)^k$, for any $\gamma \in (0, 1/4)$.*

The first step to proving $\mathrm{R}(f \circ \textsc{Ind} \circ g) = \Omega(\mathrm{R}(f)\,\mathrm{R}(\textsc{Ind})\,\mathrm{R}(g))$ is to establish that $\mathrm{R}(\textsc{Ind} \circ g)$ is essentially the same as $\mathrm{RS}(\textsc{Ind} \circ g)$ if the index gadget is large enough.

Lemma 4.23. *Let f be a partial Boolean function and let $m = \Omega(\mathrm{R}(f)^{1.1})$. Then*

$$\mathrm{RS}(\textsc{Ind}_m \circ f) = \Omega(\mathrm{R}(f)\log m) = \Omega(\mathrm{R}(\textsc{Ind}_m)\,\mathrm{R}(f)). \tag{4.13}$$

Moreover, if $f_{\mathrm{ind}}^{\oplus c}$ is the defined as the index function on $c + 2^c$ bits composed with f in only the first c bits, we have $\mathrm{RS}(f_{\mathrm{ind}}^{\oplus c}) \ge \mathrm{RS}_{\mathrm{u}}(f_{\mathrm{ind}}^{\oplus c}) = \Omega(c\,\mathrm{R}(f))$ when $c \ge 1.1 \log \mathrm{R}(f)$.

Before proving Lemma 4.23, let us complete the proof of Theorem 4.1 assuming it.

Proof of Theorem 4.1. By Corollary 4.19, we have $\mathrm{R}(f \circ \textsc{Ind}_m \circ g) \ge \mathrm{R}(f)\,\mathrm{RS}(\textsc{Ind}_m \circ g)/10$. Combining this with Lemma 4.23 gives $\mathrm{R}(f \circ \textsc{Ind}_m \circ g) = \Omega(\mathrm{R}(f)\,\mathrm{R}(g)\log m)$, as desired. □

We can now complete the argument by proving Lemma 4.23.

Proof of Lemma 4.23. To understand what the inputs to $(\textsc{Ind}_m \circ f)_{\mathrm{sab}}$ look like, let us first analyze the function $\textsc{Ind}_m$. We can split an input to $\textsc{Ind}_m$ into a small index section and a large array section. To sabotage an input to $\textsc{Ind}_m$, it suffices to sabotage the array element that the index points to (using only a single $*$ or $\dagger$). It follows that to sabotage an input to $\textsc{Ind}_m \circ f$, it suffices to sabotage the input to f at the array element that the index points to. In other words, we consider sabotaged inputs where the only stars in the input are in one array cell whose index is the output of the first c copies of f, where c is the largest integer such that $c + 2^c \le m$. Note that $c = \log m - \Theta(1)$.

We now convert any $\mathrm{RS}(\textsc{Ind}_m \circ f)$ algorithm into a randomized algorithm for $f^{\oplus c}$. First, using Lemma 2.12, we get a $2\,\mathrm{RS}(\textsc{Ind}_m \circ f)$ query randomized algorithm that finds a $*$ or $\dagger$ with probability $1/2$ if the input is sabotaged. Next, consider running this algorithm on a non-sabotaged input. It makes $2\,\mathrm{RS}(\textsc{Ind}_m \circ f)$ queries. With probability $1/2$, one of these queries will be in the array cell whose index is the true answer to $f^{\oplus c}$ evaluated on the

first cn bits. We can then consider a new algorithm A that runs the above algorithm for $2\,\mathrm{RS}(\mathrm{IND}_m \circ f)$ queries, then picks one of the $2\,\mathrm{RS}(\mathrm{IND}_m \circ f)$ queries at random, and if that query is in an array cell, it outputs the index of that cell. Then A uses $2\,\mathrm{RS}(\mathrm{IND}_m \circ f)$ queries and evaluates $f^{\oplus c}$ with probability at least $\mathrm{RS}(\mathrm{IND}_m \circ f)^{-1}/4$.

Next, Theorem 4.22 implies that for any $\gamma \in (0, 1/4)$, either A's success probability is smaller than $(1/2+\gamma)^c$, or else A uses at least $\gamma^3 c\,\mathrm{R}(f)/11$ queries. This means either

$$\mathrm{RS}(\mathrm{IND}_m \circ f)^{-1}/4 \leq (1/2+\gamma)^c \qquad \text{or} \qquad 2\,\mathrm{RS}(\mathrm{IND}_m \circ f) \geq \gamma^3 c\,\mathrm{R}(f)/11. \tag{4.14}$$

Now if we choose $\gamma = 0.01$, it is clear that the second inequality in (4.14) yields $\mathrm{RS}(\mathrm{IND}_m \circ f) = \Omega(c\,\mathrm{R}(f)) = \Omega(\mathrm{R}(f)\log m)$ no matter what m (and hence c) is chosen to be.

To complete the argument, we show that the first inequality in (4.14) also yields the same. Observe that the first inequality is equivalent to

$$\mathrm{RS}(\mathrm{IND}_m \circ f) = \Omega\left(\left(\frac{2}{1+2\gamma}\right)^c\right) = \Omega\left(\left(\frac{2}{1+2\gamma}\right)^{\log m - \Theta(1)}\right)$$

$$= \Omega(m^{\log_2(2/1.02)}) = \Omega(m^{0.97}).$$

We now have $m^{0.97} = \Omega(m^{0.96}\log m) = \Omega(\mathrm{R}(f)^{1.1\times 0.96}\log m) = \Omega(\mathrm{R}(f)\log m)$, as desired.

The lower bound on $\mathrm{RS_u}(f_{\mathrm{ind}}^{\oplus c})$ follows similarly once we makes two observations. First, this argument works equally well for $f_{\mathrm{ind}}^{\oplus c}$ instead of $\mathrm{IND}_m \circ f$. Second, sabotaging the array cell indexed by the outputs to the c copies of f in $f_{\mathrm{ind}}^{\oplus c}$ introduces only one asterisk or obelisk, so the argument above lower bounds $\mathrm{RS_u}(f_{\mathrm{ind}}^{\oplus c})$ and not only $\mathrm{RS}(f_{\mathrm{ind}}^{\oplus c})$. □

4.6 Relating Lifting Theorems

In this section we establish Theorem 4.2, which proves that a lifting theorem for zero-error randomized communication complexity implies one for bounded-error randomized communication complexity.

To begin, we introduce the two-party index gadget (also used in [39]).

Definition 4.24 (Two-party index gadget). *For any integer $b > 0$, and finite set $\mathcal{Y}$, we define the index function $G_{\mathrm{IND}} : \{0,1\}^b \times \mathcal{Y}^{2^b} \to \mathcal{Y}$ as follows. Let $(x, y) \in \{0,1\}^b \times \mathcal{Y}^{2^b}$ be an input to G_{IND}. Then if we interpret x as the binary representation of an integer between 0 and $2^b - 1$, the function $G_{\mathrm{IND}}(x, y)$ evaluates to y_x, the x^{th} letter of y. We also let G_b be the index function with $\mathcal{Y} = \{0,1\}$ and let G'_b be the index function with $\mathcal{Y} = \{0,1,*,\dagger\}$.*

The index gadget is particularly useful in communication complexity because it is "complete" for functions with a given value of $\min\{|\mathcal{X}|, |\mathcal{Y}|\}$. More precisely, any problem $F : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ can be reduced to G_b for $b = \lceil \log \min\{|\mathcal{X}|, |\mathcal{Y}|\} \rceil$. To see this, say $|\mathcal{X}| \leq |\mathcal{Y}|$ and let $|\mathcal{X}| = 2^b$. We now map every input $(x, y) \in \mathcal{X} \times \mathcal{Y}$ to an input (x', y') for G_b. Since $\mathcal{X}$ has size 2^b, we can view x as a string in $\{0,1\}^b$ and set $x' = x$. The string $y' = y'_0 y'_1 \cdots y'_{2^b-1} \in \{0,1\}^{2^b}$ is defined as $y'_x = F(x, y)$. Hence we can assume without loss of generality that a supposed lifting theorem for zero-error protocols is proved using the two-party index gadget of some size.

Our first step is to lower bound the bounded-error randomized communication complexity of a function in terms of the zero-error randomized communication complexity of a related function.

Lemma 4.25. *Let f be an n-bit (partial) Boolean function and let $G_b : \{0,1\}^b \times \{0,1\}^{2^b} \to \{0,1\}$ be the index gadget with $b = O(\log n)$. Then*

$$\mathrm{R}^{\mathrm{cc}}(f \circ G_b) = \Omega\left(\frac{\mathrm{R}_0^{\mathrm{cc}}(f_{\mathrm{usab}} \circ G'_b)}{\log n \log\log n}\right), \tag{4.15}$$

where G'_b is the index gadget mapping $\{0,1\}^b \times \{0,1,*,\dagger\}^{2^b}$ to $\{0,1,*,\dagger\}$.

Proof. We will use a randomized protocol A for $f \circ G_b$ to construct a zero-error protocol B for $f_{\mathrm{usab}} \circ G'_b$. Note the given input to $f_{\mathrm{usab}} \circ G'_b$ must have a unique copy of G'_b that evaluates to $*$ or $\dagger$, with all other copies evaluating to 0 or 1. The goal of B is to find this copy and determine if it evaluates to $*$ or $\dagger$. This will evaluate $f_{\mathrm{usab}} \circ G'_b$ with zero error.

Note that if we replace all $*$ and $\dagger$ symbols in Bob's input with 0 or 1, we would get a valid input to to $f \circ G_b$, which we can evaluate using A. Moreover, there is a single special $*$ or $\dagger$ in Bob's input that governs the value of this input to $f \circ G_b$ no matter how we fix the rest of the $*$ and $\dagger$ symbols. Without loss of generality, we assume that if the special symbol is replaced by 0, the function $f \circ G_b$ evaluates to 0, and if it is replaced by 1, it evaluates to 1.

We can now binary search to find this special symbol. There are at most $n2^b$ asterisks and obelisks in Bob's input. We can set the left half to 0 and the right half to 1, and evaluate the resulting input using A. If the answer is 0, the special symbol is on the left half; otherwise, it is on the right half. We can proceed to binary search in this way, until we have zoomed in on one gadget that must contain the special symbol. This requires narrowing down the search space from n possible gadgets to 1, which requires $\log n$ rounds. Each round requires a call to A, times a $O(\log\log n)$ factor for error reduction. We can therefore find the right gadget with bounded error, using $O(\mathrm{R}^{\mathrm{cc}}(f \circ G_b) \log n \log\log n)$ bits of communication.

Once we have found the right gadget, we can certify its validity by having Alice send the right index to Bob, using b bits of communication, and Bob can check that it points to an asterisk or obelisk. Since we found a certificate with constant probability, we can use Lemma 4.4 to turn this into a zero-error algorithm. Thus

$$\mathrm{R}_0^{\mathrm{cc}}(f_{\mathrm{usab}} \circ G'_b) = O(b + \mathrm{R}^{\mathrm{cc}}(f \circ G_b) \log n \log\log n). \tag{4.16}$$

Since $b = O(\log n)$, we obtain $\mathrm{R}_0^{\mathrm{cc}}(f_{\mathrm{usab}} \circ G'_b) = O(\mathrm{R}^{\mathrm{cc}}(f \circ G_b) \log n \log\log n)$. □

Equipped with this lemma we can prove the connection between lifting theorems (Theorem 4.2), stated more precisely as follows.

Theorem 4.2 (Precise version)**.** *Suppose that for all partial Boolean functions f on n bits, we have*

$$\mathrm{R}_0^{\mathrm{cc}}(f \circ G_b) = \Omega(\mathrm{R}_0(f)/\operatorname{polylog} n) \tag{4.17}$$

with $b = O(\log n)$. Then for all partial functions Boolean functions, we also have

$$\mathrm{R}^{\mathrm{cc}}(f \circ G_{2b}) = \Omega(\mathrm{R}(f)/\operatorname{polylog} n). \tag{4.18}$$

The $\operatorname{polylog} n$ *loss in the* R^{cc} *result is only* $\log n \log\log^2 n$ *worse than the loss in the* $\mathrm{R}_0^{\mathrm{cc}}$ *hypothesis.*

Proof. First we show that for any function f and positive integer c,

$$\mathrm{R}^{\mathrm{cc}}(f \circ G_{2b}) = \Omega\left(\frac{\mathrm{R}^{\mathrm{cc}}(f_{\mathrm{ind}}^{\oplus c} \circ G_{2b})}{c \log c}\right). \tag{4.19}$$

To see this, note that we can solve $f_{\mathrm{ind}}^{\oplus c} \circ G_{2b}$ by solving the c copies of $f \circ G_{2b}$ and then examining the appropriate cell of the array. This uses $c\,\mathrm{R}^{\mathrm{cc}}(f \circ G_{2b})$ bits of communication, times $O(\log c)$ since we must amplify the randomized protocol to an error of $O(1/c)$.

Next, using (4.19) and Lemma 4.25 on $\mathrm{R}^{\mathrm{cc}}(f_{\mathrm{ind}}^{\oplus c} \circ G_{2b})$, we get

$$\mathrm{R}^{\mathrm{cc}}(f \circ G_{2b}) = \Omega\left(\frac{\mathrm{R}^{\mathrm{cc}}(f_{\mathrm{ind}}^{\oplus c} \circ G_{2b})}{c \log c}\right) = \Omega\left(\frac{\mathrm{R}_0^{\mathrm{cc}}((f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}} \circ G'_{2b})}{c \log c \log n \log\log n}\right). \tag{4.20}$$

From here we want to use the assumed lifting theorem for R_0. However, there is a technicality: the gadget G'_{2b} is not the standard index gadget, and the function $(f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}}$ does not have Boolean alphabet. To remedy this, we use two bits to represent each of the symbols $\{0, 1, *, \dagger\}$. Using this representation, we define a new function $(f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}}^{\mathrm{bin}}$ on twice as many bits.

We now compare $(f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}}^{\mathrm{bin}} \circ G_b$ to $(f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}} \circ G'_{2b}$. Note that the former uses two pointers of size b to index two bits, while the latter uses one pointer of size $2b$ to index one symbol in $\{0, 1, *, \dagger\}$ (which is equivalent to two bits). It's not hard to see that the former function is equivalent to the latter function restricted to a promise. This means the communication complexity of the former is smaller, and hence

$$\mathrm{R}_0^{\mathrm{cc}}((f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}} \circ G'_{2b}) = \Omega(\mathrm{R}_0^{\mathrm{cc}}((f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}}^{\mathrm{bin}} \circ G_b)). \tag{4.21}$$

We are now ready to use the assumed lifting theorem for R_0. To be more precise, let's suppose a lifting result that states $\mathrm{R}_0^{\mathrm{cc}}(f \circ G_b) = \Omega(\mathrm{R}_0(f)/\ell(n))$ for some function $\ell(n)$. Thus

$$\mathrm{R}_0^{\mathrm{cc}}((f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}}^{\mathrm{bin}} \circ G_b) = \Omega(\mathrm{R}_0((f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}}^{\mathrm{bin}})/\ell(n)). \tag{4.22}$$

We note that

$$\mathrm{R}_0((f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}}^{\mathrm{bin}}) = \Omega(\mathrm{R}_0((f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}})) = \Omega(\mathrm{RS}_{\mathrm{u}}(f_{\mathrm{ind}}^{\oplus c})). \tag{4.23}$$

Setting $c = 1.1 \log \mathrm{R}(f)$, we have $\mathrm{RS}_{\mathrm{u}}(f_{\mathrm{ind}}^{\oplus c}) = \Omega(c\,\mathrm{R}(f))$ by Lemma 4.23. Combining this with (4.21), (4.22), and (4.23), we get

$$\mathrm{R}_0^{\mathrm{cc}}((f_{\mathrm{ind}}^{\oplus c})_{\mathrm{usab}} \circ G'_{2b}) = \Omega(c\,\mathrm{R}(f)/\ell(n)). \tag{4.24}$$

Combining this with (4.20) yields

$$\mathrm{R}^{\mathrm{cc}}(f \circ G_{2b}) = \Omega\left(\frac{c\,\mathrm{R}(f)}{\ell(n) c \log c \log n \log\log n}\right) = \Omega\left(\frac{\mathrm{R}(f)}{\ell(n) \log n \log\log^2 n}\right). \tag{4.25}$$

This gives the desired lifting theorem for bounded-error randomized communication with polylog n loss that is at most $\log n \log\log^2 n$ worse than the loss in the assumed $\mathrm{R}_0^{\mathrm{cc}}$ lifting theorem. □

4.7 Comparison with Other Lower Bound Methods

In this section we compare $\mathrm{RS}(f)$ with other lower bound techniques for bounded-error randomized query complexity. Figure 4-1 shows the two most powerful lower bound techniques for $\mathrm{R}(f)$, the partition bound ($\mathrm{prt}(f)$) and quantum query complexity ($\mathrm{Q}(f)$), which subsume all other general lower bound techniques. The partition bound and quantum query complexity are incomparable, since there are functions for which the partition bound is larger, e.g., the OR function, and functions for which quantum query complexity is larger [11]. Another common lower bound measure, approximate polynomial degree ($\widetilde{\deg}$) is smaller than both.

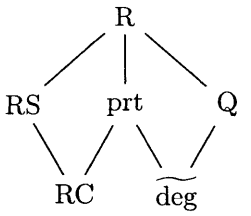


Figure 4-1: Lower bounds on $\mathrm{R}(f)$.

Randomized sabotage complexity (RS) can be much larger than the partition bound and quantum query complexity as we now show. We also show that randomized sabotage complexity is always as large as randomized certificate complexity (RC), which itself is larger than block sensitivity, another common lower bound technique. Lastly, we also show that $\mathrm{R}_0(f) = O(\mathrm{RS}(f)^2 \log \mathrm{RS}(f))$, showing that RS is a quadratically tight lower bound, even for zero-error randomized query complexity.

4.7.1 Partition Bound and Quantum Query Complexity

We start by showing the superiority of randomized sabotage complexity against the two best lower bounds for $\mathrm{R}(f)$. Informally, what we show is that any separation between $\mathrm{R}(f)$ and a lower bound measure like $\mathrm{Q}(f)$, $\mathrm{prt}(f)$, or $\widetilde{\deg}(f)$ readily gives a similar separation between $\mathrm{RS}(f)$ and the same measure.

Theorem 4.26. *There exist total functions f and g such that* $\mathrm{RS}(f) \geq \mathrm{prt}(f)^{2-o(1)}$ *and* $\mathrm{RS}(g) = \tilde{\Omega}(\mathrm{Q}(g)^{2.5})$. *There is also a total function h with* $\mathrm{RS}(h) \geq \widetilde{\deg}(h)^{4-o(1)}$.

Proof. These separations were shown with $\mathrm{R}(f)$ in place of $\mathrm{RS}(f)$ in [4] and [11]. To get a lower bound on RS, we can simply compose IND with these functions and apply Lemma 4.23. This increases RS to be the same as R (up to logarithmic factors), but it does not increase prt, $\widetilde{\deg}$, or Q more than logarithmically, so the desired separations follow. □

As it turns out, we didn't even need to compose IND with these functions. It suffices to observe that they all use the cheat sheet construction, and that an argument similar to the proof of Lemma 4.23 implies that $\mathrm{RS}(f_{\mathrm{CS}}) = \tilde{\Omega}(\mathrm{R}(f))$ for all f (where f_{CS} denotes the cheat sheet version of f, as defined in [4]). In particular, cheat sheets can never be used to separate RS from R (by more than logarithmic factors).

4.7.2 Randomized Certificate Complexity

Finally, we also show that randomized sabotage complexity upper bounds randomized certificate complexity. To show this, we first define randomized certificate complexity.

Given a string x, a block is a set of bits of x (that is, a subset of $\{1, 2, \ldots, n\}$). If B is a block and x is a string, we denote by x^B the string given by flipping the bits specified by B in the string x. If x and x^B are both in the domain of a (possibly partial) function $f : \{0,1\}^n \to \{0,1\}$ and $f(x) \neq f(x^B)$, we say that B is a sensitive block for x with respect to f.

For a string x in the domain f, the maximum number of disjoint sensitive blocks of x is called the block sensitivity of x, denoted by $\mathrm{bs}_x(f)$. The maximum of $\mathrm{bs}_x(f)$ over all x in the domain of f is the block sensitivity of f, denoted by $\mathrm{bs}(f)$.

A fractionally disjoint set of sensitive blocks of x is an assignment of non-negative weights to the sensitive blocks of x such that for all $i \in \{1, 2, \ldots, n\}$, the sum of the weights of blocks containing i is at most 1. The maximum total weight of any fractionally disjoint set of sensitive blocks is called the fractional block sensitivity of x. This is also sometimes called the randomized certificate complexity of x, and is denoted by $\mathrm{RC}_x(f)$ [1, 83, 36]. The maximum of this over all x in the domain of f is $\mathrm{RC}(f)$ the randomized certificate complexity of f.

Aaronson [1] observed that $\mathrm{bs}_x(f) \leq \mathrm{RC}_x(f) \leq \mathrm{C}_x(f)$. We therefore have

$$\mathrm{bs}(f) \leq \mathrm{RC}(f) \leq \mathrm{C}(f) \leq \mathrm{R}_0(f) \leq \mathrm{D}(f). \tag{4.26}$$

The measure $\mathrm{RC}(f)$ is also a lower bound for $\mathrm{R}(f)$; indeed, from arguments in [1] it follows that $\mathrm{R}_\epsilon(f) \geq \mathrm{RC}(f)/(1-2\epsilon)$, so $\mathrm{R}(f) \geq \mathrm{RC}(f)/3$.

Theorem 4.27. *Let $f : \{0,1\}^n \to \{0,1\}$ be a partial function. Then $\mathrm{RS}(f) \geq \mathrm{RC}(f)/4$.*

Proof. Let x be the input that maximizes $\mathrm{RC}_x(f)$. Let $B_1, B_2, \ldots B_m$ be all the (not necessarily disjoint) sensitive blocks of x. For each $i \in \{1, 2, \ldots, m\}$, let y_i be the sabotaged input formed by replacing block B_i in x with $*$ entries. Finding a $*$ in an input chosen from $Y = \{y_1, y_2, \ldots, y_m\}$ is a special case of the sabotage problem for f, so it can be done in $\mathrm{RS}(f)$ expected queries.

We now use an argument from [1] to turn this adaptive algorithm into a non-adaptive algorithm. By Lemma 2.12, after $\lfloor 2\,\mathrm{RS}(f) \rfloor$ queries, we find a $*$ with probability at least $1/2$. For each t between 1 and $T = \lfloor 2\,\mathrm{RS}(f) \rfloor$, let p_t be the probability that the adaptive algorithm finds a $*$ on query t, conditioned on the previous queries not finding a $*$. Then we have

$$p_1 + p_2 + \cdots + p_T \geq \frac{1}{2}. \tag{4.27}$$

If we pick $t \in \{1, 2, \ldots, T\}$ uniformly and simulate query t of the adaptive algorithm (which is possible since we know x and are assuming the previous $t-1$ queries did not find a $*$), we must find a $*$ with probability at least $1/(2T) \geq 1/(4\,\mathrm{RS}(f))$. This is a non-adaptive algorithm for finding a $*$, so it is also a non-adaptive algorithm for finding a difference from x.

Let the probability distribution over inputs bits obtained from this non-adaptive algorithm be $(q_1, q_2, \ldots, q_n)$, so that the algorithm queries bit i with probability q_i. We have $\sum_{i=1}^n q_i = 1$ and for each sensitive block B_j, we have $\sum_{i \in B_j} q_i \geq 1/(4\,\mathrm{RS}(f))$.

For each sensitive block B_j, let w_j be the weight of B_j under the maximum fractional set of disjoint blocks. Then $\sum_{j=1}^m w_j = \mathrm{RC}(f)$ and for each bit i, we have $\sum_{j : i \in B_j} w_j \leq 1$. We then have

$$\frac{\mathrm{RC}(f)}{4\,\mathrm{RS}(f)} = \sum_{j=1}^m w_j \cdot \frac{1}{4\,\mathrm{RS}(f)} \leq \sum_{j=1}^m w_j \sum_{i \in B_j} q_i = \sum_{i=1}^n q_i \sum_{j : i \in B_j} w_j \leq \sum_{i=1}^n q_i \cdot 1 \leq 1.$$

Hence $\mathrm{RS}(f) \geq \mathrm{RC}(f)/4$. □

4.7.3 Zero-Error Randomized Query Complexity

Theorem 4.28. *Let $f : \{0,1\}^n \to \{0,1\}$ be a total function. Then we have $\mathrm{R}_0(f) = O(\mathrm{RS}(f)^2 \log \mathrm{RS}(f))$, or alternately, $\mathrm{RS}(f) = \Omega(\sqrt{\mathrm{R}_0(f)/\log \mathrm{R}_0(f)})$.*

Proof. Let A be the $\mathrm{RS}(f)$ algorithm. The idea will be to run A on an input to x for long enough that we can ensure it queries a bit in every sensitive block of x; this will mean A found a certificate for x. That will allow us to turn the algorithm into a zero-error algorithm for f.

Let x be any input, and let b be a sensitive block of x. If we replace the bits of x specified by b with stars, then we can find a $*$ with probability $1/2$ by running A for $2\,\mathrm{RS}(f)$ queries by Lemma 2.12. This means that if we run A on x for $2\,\mathrm{RS}(f)$ queries, it has at least $1/2$ probability of querying a bit in any given sensitive block of x. If we repeat this k times, we get a $2k\,\mathrm{RS}(f)$ query algorithm that queries a bit in any given sensitive block of x with probability at least $1 - 2^{-k}$.

Now, by [54], the number of minimal sensitive blocks in x is at most $\mathrm{RC}(f)^{\mathrm{bs}(f)}$ for a total function f. Our probability of querying a bit in all of these sensitive blocks is at least $1 - 2^{-k}\,\mathrm{RC}(f)^{\mathrm{bs}(f)}$ by the union bound. When $k \geq 1 + \mathrm{bs}(f)\log_2 \mathrm{RC}(f)$, this is at least $1/2$. Since a bit from every sensitive block is a certificate, by Lemma 4.4, we can turn this into a zero-error randomized algorithm with expected query complexity at most $4(1 + \mathrm{bs}(f)\log_2 \mathrm{RC}(f))\,\mathrm{RS}(f)$, which gives $\mathrm{R}_0(f) = O(\mathrm{RS}(f)\,\mathrm{bs}(f)\log \mathrm{RC}(f))$. Since $\mathrm{bs}(f) \leq \mathrm{RC}(f) = O(\mathrm{RS}(f))$ by Theorem 4.27, we have $\mathrm{R}_0(f) = O(\mathrm{RS}(f)^2 \log \mathrm{RS}(f))$, or $\mathrm{RS}(f) = \Omega(\sqrt{\mathrm{R}_0(f)/\log \mathrm{R}_0(f)})$. □

4.8 Deterministic Sabotage Complexity

Finally we look at the deterministic analogue of randomized sabotage complexity. It turns out that deterministic sabotage complexity (as defined in Definition 4.6) is exactly the same as deterministic query complexity for all (partial) functions. Since we already know perfect composition and direct sum results for deterministic query complexity, it is unclear if deterministic sabotage complexity has any applications.

Theorem 4.29. *Let $f : \{0,1\}^n \to \{0,1\}$ be a partial function. Then $\mathrm{DS}(f) = \mathrm{D}(f)$.*

Proof. For any function $\mathrm{DS}(f) \leq D(f)$ since a deterministic algorithm that correctly computes f must find a $*$ or † when run on a sabotaged input, otherwise its output is independent of how the sabotaged bits are filled in.

To show the other direction, let $\mathrm{D}(f) = k$. This means for every $k-1$ query algorithm, there are two inputs x and y with $f(x) \neq f(y)$, such that they have the same answers to the queries made by the algorithm. If this is not the case then this algorithm computes $f(x)$, contradicting the fact that $\mathrm{D}(f) = k$. Thus if there is a deterministic algorithm for f_{sab} that makes $k-1$ queries, there exist two inputs x and y with $f(x) \neq f(y)$ that have the same answers to the queries made by the algorithm. If we fill in the rest of the inputs bits with either asterisks or obelisks, it is clear that this is a sabotaged input (since it can be completed to either x or y), but the purported algorithm for f_{sab} cannot distinguish them. Hence $\mathrm{D}(f_{\mathrm{sab}}) \geq k$, which means $\mathrm{DS}(f) \geq \mathrm{D}(f)$. □

Chapter 5

Quantum Sabotage Complexity

5.1 Introduction

This chapter is based on unpublished work with Robin Kothari. In Chapter 4, we introduced a new lower bound method for randomized query complexity called randomized sabotage complexity. This measure satisfies several useful composition properties, can be superior to all other lower bound methods, and has applications to transferring lower bounds from query complexity to communication complexity (also known as "lifting theorems").

5.1.1 Quantum sabotage complexity

In this chapter we introduce a quantum analogue of randomized sabotage complexity that we call quantum sabotage complexity. The *quantum sabotage complexity* of a function $f : D \to \{0,1\}$ (where $D \subseteq \{0,1\}^n$), denoted $\mathrm{QS}(f)$, is the minimum number of queries needed to the input $x \in D$ to produce an output state $|\psi_x\rangle$, such that the output states corresponding to 0-inputs and 1-inputs are nearly orthogonal (or far apart in trace distance). In Section 5.2, we formally define quantum sabotage complexity and show that randomized sabotage complexity is the classical analogue of this measure.

Note that the usual bounded-error quantum query complexity of a function f, denoted $\mathrm{Q}(f)$, is defined similarly with the additional requirement that there should exist a 2-outcome measurement that (with high probability) accepts states corresponding to 1-inputs and rejects states corresponding to 0-inputs. Since measurements can only distinguish nearly orthogonal states, every quantum algorithm for computing f satisfies the definition of quantum sabotage complexity. Hence for all functions f, we have $\mathrm{QS}(f) \leq \mathrm{Q}(f)$.

This is a natural relaxation of bounded-error quantum query complexity and has been mentioned in passing in several prior works. Indeed, Barnum, Saks, and Szegedy call this measure $\mathrm{DQA}(f)$ in an early technical report [15, Remark 1]. This measure often comes up in discussions about the (positive-weights) adversary bound,[1] a lower bound for quantum query complexity introduced by Ambainis [6]. The (positive-weights) adversary bound, which we denote by $\mathrm{Adv}(f)$, has several variants [6, 7, 16, 56, 91], which are all essentially the same [81]. It was noted in several works [16, 44] that the proof that the adversary bound lower bounds quantum query complexity only uses the fact that the outputs corresponding to 0-inputs and 1-inputs are nearly orthogonal, and hence for all functions $\mathrm{QS}(f) = \Omega(\mathrm{Adv}(f))$.

[1] Not to be confused with the stronger negative-weights adversary bound (also known as the general adversary bound), which essentially equals quantum query complexity [44, 58].

(We also exhibit functions separating these measures in Section 5.5.)

5.1.2 New query relations

Our first result establishes a new relation between query measures for total functions. A total function is a function of the form $f : \{0,1\}^n \to \{0,1\}$, as opposed to a partial function, which is a function of the form $f : D \to \{0,1\}$, where $D \subseteq \{0,1\}^n$. We show a new upper bound on the zero-error quantum query complexity of f, denoted $\mathrm{Q}_0(f)$, in terms of its quantum sabotage complexity, and hence its quantum query complexity. The zero-error quantum query complexity of f is the minimum number of queries needed by a quantum algorithm that either outputs the correct answer $f(x)$ on input x, or outputs ? indicating that it does not know, but does this with probability at most $1/2$ on any input x. In Section 5.3 we prove the following.

Theorem 5.1. *For all total functions $f : \{0,1\}^n \to \{0,1\}$, we have*

$$\mathrm{Q}_0(f) = O(\mathrm{QS}(f)^5 \log \mathrm{QS}(f)) = O(\mathrm{Q}(f)^5 \log \mathrm{Q}(f)). \tag{5.1}$$

Additionally, the algorithm also outputs a certificate for $f(x)$ when it outputs $f(x)$.

This is an improvement over the previous best relationship between zero-error and bounded-error quantum query complexity, $\mathrm{Q}_0(f) = O(\mathrm{Q}(f)^6)$ [17], which follows from $\mathrm{D}(f) = O(\mathrm{Q}(f)^6)$, where $\mathrm{D}(f)$ is deterministic query complexity. In fact, our result is the first upper bound on zero-error quantum query complexity that does not follow from an upper bound on zero-error randomized query complexity. Our proof is based on a similar classical result, $\mathrm{R}_0(f) = O(\mathrm{R}(f)^2 \log \mathrm{R}(f))$ [61, 54], which is essentially optimal due to a nearly matching separation by Ambainis et al. [9].

5.1.3 Quantum statistical zero knowledge

Next we show that, surprisingly, quantum sabotage complexity lower bounds a more powerful model of computation than quantum query complexity: the query complexity of computing a function using a quantum statistical zero-knowledge (QSZK) proof system. A QSZK proof system is an interactive protocol between a quantum verifier and a computationally unbounded, but untrusted prover in which the verifier learns the value of $f(x)$ but learns essentially no more. QSZK can also be characterized in terms of its complete problem Quantum State Distinguishability [86, 87].

In Section 5.4, we discuss the history of quantum statistical zero-knowledge proofs and define an associated query measure $\mathrm{QSZK}(f)$ based on the complete problem Quantum State Distinguishability. We establish some basic properties of our definition, such as $\mathrm{QSZK}(f) \leq \mathrm{Q}(f)$, which corresponds to the complexity class containment $\mathsf{BQP} \subseteq \mathsf{QSZK}$, and $\mathrm{QSZK}(f) = \Theta(\mathrm{QSZK}(\bar{f}))$, which corresponds to the statement $\mathsf{QSZK} = \mathsf{coQSZK}$. We then show that quantum sabotage complexity lower bounds QSZK complexity.

Theorem 5.2. *For all (partial) Boolean functions f, $\mathrm{QS}(f) \leq \mathrm{QSZK}(f)$.*

As a corollary of Theorem 5.2 and $\mathrm{QS}(f) = \Omega(\mathrm{Adv}(f))$, we have for all (partial) functions f,

$$\mathrm{Q}(f) \geq \mathrm{QSZK}(f) \geq \mathrm{QS}(f) = \Omega(\mathrm{Adv}(f)). \tag{5.2}$$

This sheds some light on why the adversary bound sometimes proves poor lower bounds: it lower bounds a more powerful model of computation! For example, it is well known that the adversary bound cannot prove a super-constant lower bound for the collision problem [5]. It is also easy to see that the collision problem has a constant-query QSZK (and even classical SZK) protocol.

On the bright side, this gives us a new way to prove lower bounds on QSZK query complexity and prove oracle separations against the complexity class QSZK. For example, since we know the OR function on n-bits has $\mathrm{Adv}(\mathrm{OR}) = \Omega(\sqrt{n})$, this yields an oracle A such that $\mathsf{NP}^A \not\subseteq \mathsf{QSZK}^A$, since the OR function has small certificates.

5.1.4 Comparison with other lower bounds

We compare quantum sabotage complexity to the two main lower bound techniques for quantum query complexity: the adversary bound and the polynomial method.

As noted earlier, the adversary bound is weaker than quantum sabotage complexity since for all (partial) functions f, $\mathrm{QS}(f) = \Omega(\mathrm{Adv}(f))$. This implies that $\mathrm{QS}(f)$ coincides with $\mathrm{Q}(f)$ for most functions studied in the literature, since most quantum lower bounds are proved using the adversary method. Moreover, not only is quantum sabotage complexity always larger than the adversary bound, it can be exponentially larger for partial functions and quadratically larger for total functions as we show in Theorem 5.3.

Another popular lower bound technique is the polynomial method [17], which uses the fact that the approximate degree of a function lower bounds $\mathrm{Q}(f)$. The approximate degree of a Boolean function f, denoted $\widetilde{\deg}(f)$, is the minimum degree of a real polynomial $p(x)$ over the input variables such that for all inputs x we have $|f(x) - p(x)| \leq 1/3$.

We do not know an exponential separation between quantum sabotage complexity and approximate degree (for a partial function), since it is not even known if quantum query complexity can be exponentially larger than approximate degree for a partial functions. We do, however, show in Theorem 5.3 that quantum sabotage complexity can be polynomially larger than approximate degree for total functions.

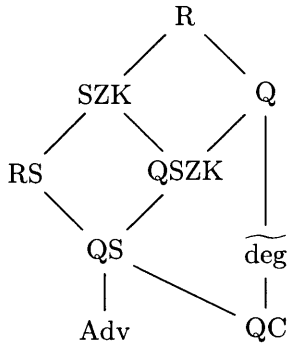


Figure 5-1: Relationships between measures. An upward line denotes that a measure is asymptotically upper bounded by the other measure. E.g.,we have for all (partial) functions f, $\mathrm{Q}(f) = O(\mathrm{R}(f))$.

Theorem 5.3. *There exist total functions f and g with*

$$\mathrm{QS}(f) = \tilde{\Omega}(\mathrm{Adv}(f)^2) \text{ and } \mathrm{QS}(g) \geq \widetilde{\deg}(g)^{4-o(1)}. \tag{5.3}$$

There also exists an n-bit partial function h with

$$\mathrm{QS}(h) = \tilde{\Omega}(n^{1/3}) \text{ and } \mathrm{Adv}(h) = O(\log n). \tag{5.4}$$

This theorem is proved in Section 5.5. Figure 5-1 shows the known relationships between all the measures discussed in this chapter.

5.1.5 Preliminaries

For any matrix A, we define the spectral norm of A, denoted $\|A\|$ as the largest singular value of A. The 1-norm of A, denoted $\|A\|_1$, is defined as $\mathrm{Tr}\left(\sqrt{A^\dagger A}\right)$, which is also equal to the sum of the singular values of A.

We define the trace distance between two quantum states ρ and σ as $\|\rho - \sigma\|_{\mathrm{tr}} = \frac{1}{2}\|\rho - \sigma\|_1$. The factor of $1/2$ makes this distance measure lie between 0 and 1 for density matrices. Trace distance is a useful distance measure since it exactly captures distinguishability of states and is non-increasing under quantum operations [64, Th. 9.2]. For pure states $|\psi\rangle$ and $|\phi\rangle$, trace distance is related to their inner product as follows [88, eq. 1.180].

$$\||\psi\rangle\langle\psi| - |\phi\rangle\langle\phi|\|_{\mathrm{tr}} = \sqrt{1 - |\langle\psi|\phi\rangle|^2}. \tag{5.5}$$

5.2 Quantum sabotage complexity

5.2.1 Definition

We now define quantum sabotage complexity more formally. As explained in the introduction, instead of requiring that the quantum algorithm output the value of the function $f(x)$, as in standard quantum query complexity, we only want the quantum algorithm's outputs to be distinguishable (or nearly orthogonal) for 0-inputs and 1-inputs.

As an example of how these definitions differ, consider the collision problem. In this problem, we are given an input $x \in [n]^n$ and we are promised that if we view x as a function from $[n] \to [n]$, the function is either 1-to-1 or 2-to-1. The goal is to distinguish these two cases under the assumption that the input satisfies this promise. In this problem, since every 0-input and 1-input differ in exactly half the positions $i \in [n]$, our quantum algorithm can simply create the state $|\psi_x\rangle = \frac{1}{\sqrt{n}}\sum_i |i, x_i\rangle$ and the states corresponding to 0-inputs and 1-inputs will have trace distance $\Omega(1)$. Thus this problem has quantum sabotage complexity $O(1)$, but its quantum query complexity is $\Theta(n^{1/3})$ [5].

Definition 5.4 (Quantum Sabotage complexity). *Let $f : D \to \{0,1\}$, where $D \subseteq \{0,1\}^n$, be an n-bit partial function.* $\mathrm{QS}(f)$ *is defined as the smallest integer k such that there exists a k-query quantum algorithm that on input $x \in D$ outputs a quantum state ρ_x such that*

$$\forall x, y \in D \text{ with } f(x) \neq f(y), \quad \|\rho_x - \rho_y\|_{\mathrm{tr}} \geq 1/6.$$

Note that the definition is robust to minor changes. First, we allow outputting mixed states, although this does not offer any additional power over only outputting pure states. The reason is that we can always assume that the quantum algorithm is pure until the final step where some subset of qubits is traced out. But if two states are far apart in trace distance after a partial trace, then they were far apart to begin with since trace distance is non-increasing under partial trace.

The constant $1/6$ in the definition is also arbitrary and any other constant in $(0, 1)$ would not change the measure by more than a multiplicative constant. This is because we can increase the trace distance between the states by outputting multiple copies of the states.

5.2.2 Properties

We can now establish some basic properties of quantum sabotage complexity. First, let us formally show that quantum sabotage complexity lower bounds quantum query complexity.

Proposition 5.5. *For all (partial) Boolean functions f, $\mathrm{QS}(f) \leq \mathrm{Q}(f)$.*

Proof. Let $\mathrm{Q}(f) = k$ and consider the k-query algorithm that witnesses this fact. Let p_x be the probability that this k-query algorithm, when run on input x, outputs 1 upon measuring the first qubit. Since the algorithm computes f with bounded error, we know that for all 1-inputs x, $p_x \geq 2/3$, and for all 0-inputs y, $p_y \leq 1/3$.

Now consider the single-qubit state ρ_x, which is obtained by taking the final state of this algorithm, tracing out all the qubits except the first one, and then applying a completely dephasing channel to it. This state is $\rho_x = \left(\begin{smallmatrix} 1-p_x & 0 \\ 0 & p_x \end{smallmatrix}\right)$. Thus $\|\rho_x - \rho_y\|_{\mathrm{tr}} = |p_x - p_y| \geq 1/3$. □

As noted in the introduction, quantum sabotage complexity is also lower bounded by the adversary bound, i.e.,

$$\mathrm{QS}(f) = \Omega(\mathrm{Adv}(f)). \tag{5.6}$$

We do not prove this since this follows from the arguments that establish that the adversary bound is a lower bound on quantum query complexity [6, 7, 16, 56, 91, 81], since all these proofs only use the fact that the states output on 0-inputs and 1-inputs are nearly orthogonal.

Quantum sabotage complexity is also superior to quantum certificate complexity $\mathrm{QC}(f)$, as we show in Proposition 5.7. Quantum certificate complexity is a lower bound on quantum query complexity defined by Aaronson [1]. It was later shown that quantum certificate complexity also lower bounds polynomial degree [54].

Before proving Proposition 5.7, we first define certificate complexity, randomized certificate complexity, and quantum certificate complexity.

Definition 5.6 (Certificate complexity)**.** *For any (partial) function f and any input $x \in \mathrm{Dom}(f)$, consider the partial function f^x defined on the domain $\{x\} \cup \{y \in \mathrm{Dom}(f) : f(y) \neq f(x)\}$ that satisfies $f^x(x) = 1$ and $f^x(y) = 0$ for all $y \in \mathrm{Dom}(f)$ with $f(y) \neq f(x)$.*

We define the certificate complexity of f, denoted $\mathrm{C}(f)$, the randomized certificate complexity of f, denoted $\mathrm{RC}(f)$, and the quantum certificate complexity of f, denoted $\mathrm{QC}(f)$, as follows:

$$\mathrm{C}(f) = \max_{x \in \mathrm{Dom}(f)} \mathrm{D}(f^x), \quad \mathrm{RC}(f) = \max_{x \in \mathrm{Dom}(f)} \mathrm{R}(f^x), \quad \text{and} \quad \mathrm{QC}(f) = \max_{x \in \mathrm{Dom}(f)} \mathrm{Q}(f^x).$$

The problem f^x is clearly no harder than computing f itself in any model of computation, and hence these are lower bounds on their respective measures, i.e., $\mathrm{C}(f) \leq \mathrm{D}(f)$, $\mathrm{RC}(f) \leq \mathrm{R}(f)$, and $\mathrm{QC}(f) \leq \mathrm{Q}(f)$. We can now prove that $\mathrm{QS}(f)$ is a better lower bound on $\mathrm{Q}(f)$ than $\mathrm{QC}(f)$.

Proposition 5.7. *For all (partial) Boolean functions f, $\mathrm{QS}(f) = \Omega(\mathrm{QC}(f))$.*

Proof. Let $\mathrm{QS}(f) = k$ and consider the k-query quantum algorithm that witnesses this fact. We can use this algorithm to solve f^x for any $x \in \mathrm{Dom}(f)$. Consider the output of the algorithm on input x before the partial trace operation and call this $|\psi_x\rangle$. The trace

distance between $|\psi_x\rangle$ and $|\psi_y\rangle$ for $y \in \mathrm{Dom}(f)$ with $f(y) \neq f(x)$ is at least $1/6$ since trace distance is non-increasing under partial trace [64, Th. 9.2].

Now we construct an algorithm for f^x from this algorithm to show that $\mathrm{Q}(f^x) = O(\mathrm{QS}(f))$. To do so, we run the supposed algorithm and measure whether the output state is $|\psi_x\rangle$ or not and accept only when the measurement accepts. This yields an algorithm that outputs 1 on x with probability 1 and accepts inputs y with $f(x) \neq f(x)$ with some constant probability strictly less than 1. More precisely, the acceptance probability is $|\langle\psi_x|\psi_y\rangle|^2 \leq 1 - (1/6)^2$ due to the relationship between inner product and trace distance for pure states. Repeating this algorithm a constant number of times yields a bounded-error quantum algorithm for f^x. □

5.2.3 Relation with randomized sabotage complexity

We start by reviewing the definition of randomized sabotage complexity, as presented in [20]. Fix a (possibly partial) Boolean function $f : D \to \{0,1\}$ with $D \in \{0,1\}^n$. For any pair $x, y \in \mathrm{Dom}(f)$ such that $f(x) \neq f(y)$, let $p \in \{0,1,*\}^n$ be the partial assignment of all bits where x and y agree (with the symbol $*$ used for the bits where x and y disagree). We call p a "sabotaged input", imagining that a saboteur replaced bits of x with $*$ symbols until it was no longer possible to determine $f(x)$.

Let $S_* \subseteq \{0,1,*\}^n$ be the set of all sabotaged inputs to f, that is, the set of all partial assignments that are consistent with both a 0-input and a 1-input to f. Let $S_\dagger \in \{0,1,\dagger\}^n$ be the same as S_*, except that the $\dagger$ symbol is used instead of the $*$ symbol. Finally, let $f_{\mathrm{sab}} : S_* \cup S_\dagger \to \{0,1\}$ be the function that takes a sabotaged input and identifies whether it has $*$ symbols or $\dagger$ symbols, promised that it contains only one type of symbol. Intuitively, f_{sab} is a decision problem that forces an algorithm computing it to find a $*$ or $\dagger$. We then define $\mathrm{RS}(f) := \mathrm{R}_0(f_{\mathrm{sab}})$, the expected running time of a zero-error randomized algorithm computing f_{sab}.

To show that $\mathrm{RS}(f)$ is larger than $\mathrm{QS}(f)$ for all f, we will start by defining a classical measure analogous to $\mathrm{QS}(f)$. We will then show this measure is equivalent to $\mathrm{RS}(f)$.

Definition 5.8 (Alternate definition of randomized sabotage complexity)**.** *Let $f : D \to \{0,1\}$, where $D \subseteq \{0,1\}^n$, be an n-bit partial function. $\mathrm{RS}'(f)$ is defined as the smallest integer k such that there exists a k-query randomized algorithm that on input $x \in D$ outputs a sample from a probability distribution d_x such that*

$$\forall x, y \in D \text{ with } f(x) \neq f(y), \quad \|d_x - d_y\|_{\mathrm{tr}} \geq 1/6.$$

Here by the trace distance between two probability distributions, we mean to consider these distributions as diagonal density matrices which is the same as the total variation distance between the probability distributions.

Since quantum algorithms can simulate classical algorithms, we immediately get that $\mathrm{QS}(f) \leq \mathrm{RS}'(f)$. Next, we will show that $\mathrm{RS}'(f) = \Theta(\mathrm{RS}(f))$, completing the argument that $\mathrm{QS}(f) = O(\mathrm{RS}(f))$ and justifying the name "quantum sabotage complexity".

Theorem 5.9. *Let f be a partial function. Then* $\mathrm{RS}(f)/12 \leq \mathrm{RS}'(f) \leq (12/11)\,\mathrm{RS}(f)$.

Proof. First, we show that $\mathrm{RS}(f) \leq 12\,\mathrm{RS}'(f)$. Let A be an optimal randomized algorithm for $\mathrm{RS}'(f)$, that on input x outputs a sample from the distribution d_x. Let $z \in \mathrm{Dom}(f_{\mathrm{sab}})$ be a sabotaged input, and consider running A on z. Since z is sabotaged, there are inputs

x and y with $f(x) \neq f(y)$ that are both consistent with the non-$*$, non-† bits of z. The variation distance between d_x and d_y is at least $1/6$.

A randomized algorithm can be viewed as a probability distribution over deterministic algorithms. Split the support of the distribution for A into two parts: a set S consisting of deterministic algorithms that, when run on z, query a $*$ or †, and a set T consisting of deterministic algorithms that don't query a $*$ or † when run on z. Note that algorithms in T behave the same on x and y. If A samples an algorithm from T with probability p, the total variation distance between the run of A on x and the run of A on y must therefore be at most $2(1-p)$. Since this is at least $1/6$, we have $p \leq 11/12$. Hence when A is run on z, it queries a $*$ or † with probability at least $1/12$.

If we repeat A whenever it does not query a $*$ or †, we get an algorithm that always finds such an entry and uses at most $12\,\mathrm{RS}'(f)$ queries on expectation. This is a zero-error randomized algorithm for f_{sab}, so $\mathrm{RS}(f) \leq 12\,\mathrm{RS}'(f)$.

We now handle the other direction, showing $\mathrm{RS}'(f) \leq (12/11)\,\mathrm{RS}(f)$. Let A be an optimal zero-error randomized algorithm for f_{sab}. It makes $\mathrm{RS}(f)$ queries on expectation, and always finds a $*$ or † in any sabotaged input. Consider the algorithm B that, on input $x \in \mathrm{Dom}(f)$, runs A for at most $2\,\mathrm{RS}(f)$ queries and outputs the partial assignment it queried (that is, it outputs all the pairs (i, x_i) that were queried by the algorithm A).

Let x and y be inputs to f with $f(x) \neq f(y)$. Let z be the sabotaged input defined by x and y, that is, $z_i = *$ if $x_i \neq y_i$ and $z_i = x_i = y_i$ otherwise. By Markov's inequality, after $(12/11)\,\mathrm{RS}(f)$ queries, A finds a $*$ with probability at least $1/12$ when it is run on z. This means that when A is run on x, it queries an index i for which $x_i \neq y_i$ with probability at least $1/12$. When this happens, the output of $B(x)$ is not in the support of d_y. This means d_x puts weight at least $1/12$ on symbols not in the support of d_y. Conversely, d_y puts weight at least $1/12$ on symbols not in the support of d_x. The total variation distance between the two distributions is therefore at least $1/6$, meaning B is a valid $\mathrm{RS}'(f)$ algorithm. We conclude that $\mathrm{RS}'(f) \leq (12/11)\,\mathrm{RS}(f)$. □

Combined with $\mathrm{QS}(f) \leq \mathrm{RS}'(f)$, this theorem gives us the following corollary.

Corollary 5.10. *For all (partial) Boolean functions f, $\mathrm{QS}(f) = O(\mathrm{RS}(f))$.*

5.3 New query relations

In this section we prove a new relationship between zero-error quantum query complexity and quantum sabotage complexity and bounded-error quantum query complexity, restated below.

Theorem 5.1. *For all total functions $f : \{0,1\}^n \to \{0,1\}$, we have*

$$\mathrm{Q}_0(f) = O(\mathrm{QS}(f)^5 \log \mathrm{QS}(f)) = O(\mathrm{Q}(f)^5 \log \mathrm{Q}(f)). \tag{5.1}$$

Additionally, the algorithm also outputs a certificate for $f(x)$ when it outputs $f(x)$.

Our proof uses ideas from an analogous classical result [61, 54] and the main quantum ingredient used is the hybrid argument of Bennett, Bernstein, Brassard, and Vazirani. [21].

5.3.1 Hybrid argument

Before describing the hybrid argument, we need to define the notion of a *sensitive block*. For a string $x \in \{0,1\}^n$ and a subset of input bits $B \subseteq [n]$, which we call a block, we use x^B to denote the input with all bits in B flipped. In other words, x^B agrees with x on all positions outside B and disagrees on B. For a function f and an input $x \in \mathrm{Dom}(f)$, we say a block B is a sensitive block if $f(x) \neq f(x^B)$.

Intuitively this means that any algorithm that distinguishes x from x^B, where B is a sensitive block, must "look" at B. For classical algorithms, this simply means the algorithm has to query a bit from B with high probability. The analogous statement for quantum algorithms is not so clear, since quantum algorithms can query all input bits in superposition. Nevertheless, the hybrid argument still allows us to assert something about queries made to a sensitive block [21]:

Lemma 5.11 (Hybrid Argument). *Let $x \in \{0,1\}^n$ be an input, and let $B \subseteq [n]$ be a block. Let Q be a T-query quantum algorithm that accepts x and rejects x^B with high probability, or more generally produces output states that are a constant distance apart in trace distance for x and x^B.*

Let m_i^t be the probability that, when Q is run on x for t queries and then subsequently measured, it is found to be querying position i of x. Then

$$\sum_{t=1}^{T} \sum_{i \in B} m_i^t = \Omega\left(\frac{1}{T}\right).$$

This lemma was implicitly proven in [21]. We reproduce the proof here for the reader's convenience.

Proof. We start by fixing some notation. A quantum query algorithm will have a work tape consisting of four registers,

$$|A\rangle|i\rangle|b\rangle|o\rangle,$$

where A is a general work register of arbitrarily large dimension, $i \in [n]$ specifies the position to be queried, $b \in \{0,1\}$ is a bit for the result of the query, and o is the output register for the entire algorithm. The oracle unitary for input x is the unitary U^x that acts on the tape by mapping

$$|A\rangle|i\rangle|b\rangle|o\rangle \to |A\rangle|i\rangle|b \oplus x_i\rangle|o\rangle.$$

A T-query quantum query algorithm is a sequence of unitaries $U_0, U_1, \ldots, U_T$ acting on the work tape, and the output of the algorithm is the result of applying

$$U_T U^x U_{T-1} U^x \ldots U^x U_0 |init\rangle$$

(for some fixed initial state $|init\rangle$) and examining the output register. In the case of a quantum algorithm computing a Boolean function, the output register $o \in \{0,1\}$ is simply measured. In the case of a quantum sabotage algorithm, the registers other than the output register are traced out, and the resulting mixed state is the output (the dimension of the Hilbert space of $|o\rangle$ can be arbitrarily large for a quantum sabotage algorithm).

Note that if an algorithm computes $f(x)$ with probability $1-\epsilon$, it can output a mixed state ρ_x such that $\|\rho_x - \rho_y\|_{\mathrm{tr}} \geq 1 - 2\epsilon$ whenever $x \neq y$, as we saw in Proposition 5.5. We therefore restrict ourselves to the case where the algorithm outputs a mixed state such that

the mixed state it outputs on input x is $1-2\epsilon$ far in trace distance from the mixed state it outputs on input x^B.

Define $|\psi_0^x\rangle := U_0|init\rangle$ and $|\psi_t^x\rangle := U_t U^x \psi_{t-1}^x$ for $t = 1, 2, \ldots, T$, representing the state of the tape after t queries when the algorithm is run on input x. The final state of the algorithm is $|\psi_T^x\rangle$. Since the trace distance is non-increasing under partial trace [64, Th. 9.2], we have $\left\| |\psi_T^x\rangle\langle\psi_T^x| - |\psi_T^{x^B}\rangle\langle\psi_T^{x^B}| \right\|_{\mathrm{tr}} \geq 1-2\epsilon$, which by (5.5) gives

$$|\langle\psi_T^x|\psi_T^{x^B}\rangle| \leq \sqrt{1-(1-2\epsilon)^2} \leq 1-(1/2)(1-2\epsilon)^2.$$

Then

$$\||\psi_T^x\rangle - |\psi_T^{x^B}\rangle\|^2 = 2 - \langle\psi_T^{x^B}|\psi_T^x\rangle - \langle\psi_T^x|\psi_T^{x^B}\rangle = 2 - 2\,\mathrm{Re}(\langle\psi_T^{x^B}|\psi_T^x\rangle) \geq 2 - 2|\langle\psi_T^{x^B}|\psi_T^x\rangle|$$

which is at least $(1-2\epsilon)^2$, so $\||\psi_T^x\rangle - |\psi_T^{x^B}\rangle\| \geq 1-2\epsilon$.

Hence the final states of the algorithm are far in apart on input x and x^B. We also know that the initial states $|\psi_0^x\rangle$ and $|\psi_0^{x^B}\rangle$ are identical. We keep track of how much this distance $d_t := \||\psi_t^x\rangle - |\psi_t^{x^B}\rangle\|$ changes for $t = 0, 1, \ldots, T$. For each t, we have

$$d_{t+1} = \||\psi_{t+1}^x\rangle - |\psi_{t+1}^{x^B}\rangle\| = \|U_{t+1}U^x|\psi_t^x\rangle - U_{t+1}U^{x^B}|\psi_t^{x^B}\rangle\| = \|U^x|\psi_t^x\rangle - U^{x^B}|\psi_t^{x^B}\rangle\|,$$

since U_{t+1} is a unitary and preserves norms. This equals

$$\|U^{x^B}|\psi_t^x\rangle - U^{x^B}|\psi_t^{x^B}\rangle + (U^x - U^{x^B})|\psi_t^x\rangle\| \leq \|U^{x^B}|\psi_t^x\rangle - U^{x^B}|\psi_t^{x^B}\rangle\| + \|(U^x - U^{x^B})|\psi_t^x\rangle\|$$

$$= d_t + \|(U^x - U^{x^B})|\psi_t^x\rangle\|.$$

Next, decompose $|\psi_t^x\rangle$ by the value of the query register. On basis vectors when the query register is not in B, the unitaries U^x and U^{x^B} behave the same; such vectors therefore get mapped to zero. If $|\psi_t^{x,B}\rangle$ denotes the component of $|\psi_t^x\rangle$ whose query register is in B, we get

$$\|(U^x - U^{x^B})|\psi_t^x\rangle\| = \|(U^x - U^{x^B})|\psi_t^{x,B}\rangle\| \leq \|U^x|\psi_t^{x,B}\rangle\| + \|U^{x^B}|\psi_t^{x,B}\rangle\| = 2\||\psi_t^{x,B}\rangle\|$$

$$= 2 \cdot \sqrt{\sum_{i\in B} m_i^{t+1}},$$

where the last equality follows from the definition of the m_i^{t+1} (which are defined to be the probability that the algorithm is found to be querying position i at query $t+1$, after t queries happened). The increase from d_t to d_{t+1} is therefore upper bounded by $2\sqrt{\sum_{i\in B} m_i^{t+1}}$, so we have

$$2\sum_{t=1}^{T} \sqrt{\sum_{i\in B} m_i^t} \geq d_T - d_0 \geq 1-2\epsilon.$$

Using Cauchy-Schwartz on the outer sum gives

$$2\sqrt{T}\sqrt{\sum_{t=1}^{T}\sum_{i\in B} m_i^t} \geq 1-2\epsilon,$$

or

$$\sum_{t=1}^{T} \sum_{i \in B} m_i^t \geq \frac{(1-2\epsilon)^2}{4T},$$

as desired[2]. □

5.3.2 New upper bound

To prove our result we also need to upper bound the number of minimal sensitive blocks of a function. It is not too hard to show that any minimal sensitive block has size at most the sensitivity of f, $s(f)$, which is the maximum number of sensitive blocks of size 1 over all inputs x. Since there are at most $\binom{n}{\mathrm{s}(f)} = O(n^{\mathrm{s}(f)})$ different subsets of n positions of size $\mathrm{s}(f)$, we know that the number of minimal sensitive blocks is at most this quantity. Kulkarni and Tal [54] improve this simple upper bound replacing n with randomized certificate complexity $\mathrm{RC}(f)$ (Definition 2.21).

Lemma 5.12. *For any total function $f : \{0,1\}^n \to \{0,1\}$ and any input $x \in \{0,1\}^n$, the number of minimal sensitive blocks of x with respect to f is at most $O(\mathrm{RC}(f)^{\mathrm{s}(f)})$.*

We are now ready to prove Theorem 5.1.

Proof of Theorem 5.1. Let Q be the optimal quantum sabotage algorithm for f, that uses $T = \mathrm{QS}(f)$ queries. Consider running the following quantum algorithm P on oracle input $x \in \{0,1\}^n$:

1. Pick $t \in [T]$ uniformly at random.
2. Run Q on x for t queries and measure the query register.
3. Write down (on a classical tape) the position i where Q is found to be querying, as well as the query output x_i.

The algorithm P uses $t \leq T$ quantum queries. Now that the probability P wrote down the index i is $(1/T)\sum_{t=1}^{T} m_i^t$. For any block $B \subseteq [n]$, the probability that P wrote down some index in B is

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{i \in B} m_i^t.$$

If B is a sensitive block for the input x, then the hybrid argument (Lemma 5.11) implies the probability that our new algorithm P outputs an index in B is $\Omega(1/T^2)$.

Next, we repeat the algorithm P several times. We claim that after $O(T^2\,\mathrm{s}(f)\log \mathrm{RC}(f))$ repetitions, the outputs of P constitute a certificate for x with constant probability.

To see this, note that for any minimal sensitive block B of the input x, the probability that some run of P (out of the $O(T^2\,\mathrm{s}(f)\log \mathrm{RC}(f))$ many runs) queries in the block B is $1 - O(\mathrm{RC}(f)^{-\mathrm{s}(f)})$. This is because T^2 repetitions boost the probability of querying in a minimal sensitive block from $\Omega(1/T^2)$ to $\Omega(1)$, and then $\mathrm{s}(f)\log \mathrm{RC}(f)$ repetitions of this boosted algorithm further boost the probability to the claimed bound. Hence, by Lemma 5.12 and the union bound, there is a constant probability that these runs of P query a bit in every minimal sensitive block of the input x. But a set of bits that intersects every

[2]This can be slightly improved to $(1 - 2\sqrt{\epsilon(1-\epsilon)})/2T$.

sensitive block of x is a certificate for x. Thus these runs of P output a certificate for the input x with constant probability.

Any algorithm that finds a certificate with constant probability can be turned into a zero-error algorithm by repeating whenever a certificate is not found. We therefore get a zero-error algorithm that works simply by repeating P a sufficient number of times. Note that P uses $O(T)$ quantum queries and must be repeated $O(T^2\,\mathrm{s}(f)\log \mathrm{RC}(f))$ times. Recalling that $T = \mathrm{QS}(f)$, we get

$$\mathrm{Q}_0(f) = O(\mathrm{QS}(f)^3\,\mathrm{s}(f)\log \mathrm{RC}(f)).$$

We can simplify this to $\mathrm{Q}_0(f) = O(\mathrm{QS}(f)^5 \log \mathrm{QS}(f))$, since $\mathrm{s}(f) = O(\mathrm{RC}(f)) = O(\mathrm{QC}(f)^2)$ by [1] and $\mathrm{QC}(f) = O(\mathrm{QS}(f))$ (Proposition 5.7). □

5.4 Quantum statistical zero knowledge

5.4.1 History

The subject of statistical zero-knowledge proof systems has a rich history in the classical setting, and the interested reader is referred to the paper of Sahai and Vadhan [72]. Informally, the complexity class SZK contains problems that can be solved by a probabilistic polynomial-time verifier interacting with a computationally unbounded prover (like the class IP) with the additional restriction that the verifier not learn anything from the prover (statistically) other than the answer to the problem. From this it is clear that BPP $\subseteq$ SZK, since the verifier can simply not interact with the prover, and SZK $\subseteq$ IP, since IP is simply SZK without the zero-knowledge constraint.

More surprisingly, it is also known that SZK = coSZK, and that we can assume without loss of generality that the interaction is only one round and uses public randomness, which means SZK $\subseteq$ AM$\cap$coAM. Another interesting subtlety is that SZK can be defined assuming an honest verifier, one who does not deviate from the protocol to learn more, or a cheating verifier, who may deviate from the protocol. It turns out that these definitions lead to the same complexity class [37]. The class SZK also has a much simpler characterization in terms of a complete problem called *statistical difference*, as shown by Sahai and Vadhan [72], which yields easier proofs of some of these facts. Informally, in the statistical difference problem we are given two circuits that sample from probability distributions, and the task is determine whether the distributions are far or close in total variation distance.

On the quantum side, (honest-verifier) QSZK was first defined by Watrous [86], and like the classical case, it satisfies BQP $\subseteq$ QSZK $\subseteq$ QIP. The same paper strengthened these obvious containments by showing that QSZK is closed under complement (i.e., QSZK = coQSZK) and that the protocol can be assumed to be one round, which gives QSZK $\subseteq$ QIP(2). Watrous also showed that QSZK has a complete problem, called *quantum state distinguishability*, which is a quantum generalization of the statistical difference problem of Sahai and Vadhan. In this problem, we are given two quantum circuits outputting mixed states and have to decide if the states are far apart or close in trace distance. Later, Watrous [87] also showed that honest-verifier QSZK and cheating-verifier QSZK are the same, as in the classical case.

5.4.2 Definition

We now define a query analogue of quantum statistical zero-knowledge. Instead of defining $\mathrm{QSZK}(f)$ in terms of an interactive zero-knowledge protocol for f, we use the complete

problem characterization by Watrous. This yields a considerably simpler definition of QSZK in the query setting.[3]

Definition 5.13 (QSZK). *Let $f : D \to \{0,1\}$, where $D \subseteq \{0,1\}^n$, be an n-bit partial function.* $\mathrm{QSZK}(f)$ *is defined as the smallest integer k such that there exists a k-query quantum algorithm that on input $x \in D$ outputs two unentangled quantum states ρ_x and σ_x of the same size such that*

- $\forall x \in D$ *with* $f(x) = 1$, $\|\rho_x - \sigma_x\|_{\mathrm{tr}} \geq 2/3$,
- $\forall x \in D$ *with* $f(x) = 0$, $\|\rho_x - \sigma_x\|_{\mathrm{tr}} \leq 1/3$.

In this definition, it is only important that the constants $2/3$ and $1/3$ satisfy the relationship $(2/3)^2 > 1/3$. Hence an alternate definition with 0.999 instead of $2/3$ and 0.001 instead of $1/3$ leads to the same complexity measure up to multiplicative constants (see [86, Theorem 5] for more details).

5.4.3 Properties

As a sanity check, let us prove the query analog of the obvious containment $\mathsf{BQP} \subseteq \mathsf{QSZK}$.

Proposition 5.14. *For all (partial) Boolean functions f,* $\mathrm{QSZK}(f) \leq \mathrm{Q}(f)$.

Proof. Let $\mathrm{Q}(f) = k$ and consider the k-query algorithm that witnesses this fact. Let p_x be the probability that this k-query algorithm when run on input x outputs 1 upon measuring the first qubit. Since the algorithm computes f with bounded error, we know that $p_x \geq 2/3$ for 1-inputs and $p_x \leq 1/3$ for 0-inputs.

Now consider the single-qubit state ρ_x, which is obtained by taking the final state of this algorithm, tracing out all the qubits except the first one, and then applying a completely dephasing channel to it. This is equivalent to measuring the first qubit in the standard basis and outputting $|b\rangle$ when the result is b. This state is $\rho_x = \left(\begin{smallmatrix} 1-p_x & 0 \\ 0 & p_x \end{smallmatrix}\right)$. Let us also define σ_x as $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$ for all x.

Now let us check that the conditions of Definition 5.13 are satisfied by these states. For all inputs x, we have $\|\rho_x - \sigma_x\|_{\mathrm{tr}} = |p_x|$. And we know that $p_x \geq 2/3$ for 1-inputs and $0 \leq p_x \leq 1/3$ for 0-inputs, which completes the proof. □

5.4.4 Relation with adversary bound

We have already showed that $\mathrm{QS}(f) \leq \mathrm{Q}(f)$ (Proposition 5.5) and $\mathrm{QSZK}(f) \leq \mathrm{Q}(f)$ (Proposition 5.14). We now show that $\mathrm{QS}(f)$ is actually smaller than $\mathrm{QSZK}(f)$.

Theorem 5.2. *For all (partial) Boolean functions f,* $\mathrm{QS}(f) \leq \mathrm{QSZK}(f)$.

Proof. Let $\mathrm{QSZK}(f) = k$ and consider the k-query quantum algorithm that witnesses this fact. We claim that the output of this algorithm already satisfies the conditions in Definition 5.4 and hence proves $\mathrm{QS}(f) \leq k$.

To see this, observe that the algorithm outputs the state $\rho_x \otimes \sigma_x$ on input x, which satisfies the conditions of Definition 5.13. More precisely, this means for any x and y such

[3] The complete problem is often used to define SZK (and its variants, like NISZK) in query complexity and communication complexity (for example, see [24, 82]). It is not obvious whether the interactive-proof definition and complete-problem definition coincide exactly as the problem is complete under polynomial-time reductions, which may add polynomial overhead.

that $f(x) = 1$ and $f(y) = 0$, we know that $\|\rho_x - \sigma_x\|_{\mathrm{tr}} \geq 2/3$ and $\|\rho_y - \sigma_y\|_{\mathrm{tr}} \leq 1/3$. We want to show that

$$\|\rho_x \otimes \sigma_x - \rho_y \otimes \sigma_y\|_{\mathrm{tr}} \geq 1/6.$$

Since the trace distance is non-increasing under partial trace, we have the inequalities $\|\rho_x \otimes \sigma_x - \rho_y \otimes \sigma_y\|_{\mathrm{tr}} \geq \|\rho_x - \rho_y\|_{\mathrm{tr}}$ and $\|\rho_x \otimes \sigma_x - \rho_y \otimes \sigma_y\|_{\mathrm{tr}} \geq \|\sigma_x - \sigma_y\|_{\mathrm{tr}}$, which imply

$$\|\rho_x \otimes \sigma_x - \rho_y \otimes \sigma_y\|_{\mathrm{tr}} \geq \max\left\{\|\rho_x - \rho_y\|_{\mathrm{tr}}, \|\sigma_x - \sigma_y\|_{\mathrm{tr}}\right\}.$$

Now if we can show the right-hand side is at least $1/6$, then we are done. To show this, toward a contradiction assume that $\max\left\{\|\rho_x - \rho_y\|_{\mathrm{tr}}, \|\sigma_x - \sigma_y\|_{\mathrm{tr}}\right\} < 1/6$. Then we have

$$\begin{aligned}\|\rho_x - \sigma_x\|_{\mathrm{tr}} &= \|\rho_x - \rho_y + \rho_y - \sigma_y + \sigma_y - \sigma_x\|_{\mathrm{tr}} \\ &\leq \|\rho_x - \rho_y\|_{\mathrm{tr}} + \|\rho_y - \sigma_y\|_{\mathrm{tr}} + \|\sigma_y - \sigma_x\|_{\mathrm{tr}} \\ &< 1/6 + 1/3 + 1/6 = 2/3,\end{aligned}$$

which contradicts $\|\rho_x - \sigma_x\|_{\mathrm{tr}} \geq 2/3$. □

As noted, as a corollary of this theorem and $\mathrm{QS}(f) = \Omega(\mathrm{Adv}(f))$, we have for all (partial) functions f,

$$\mathrm{QSZK}(f) = \Omega(\mathrm{Adv}(f)).$$

This can be used to prove lower bounds on QSZK protocols for functions. For example, consider the OR function and let us try to compute it with an interactive protocol without the zero-knowledge requirement. It is easy to see that when $\mathrm{OR}(x) = 1$, a computationally unbounded prover can simply send over the location of a bit i such that $x_i = 1$, which can be checked using only 1 query. Of course, this protocol leaks information and in particular lets the verifier know the location of a 1. But is it necessary that an efficient protocol for OR must leak information? Our lower bound says this must be the case, because $\mathrm{Adv}(\mathrm{OR}) = \Omega(\sqrt{n})$ and hence any zero-knowledge protocol for the function must make $\Omega(\sqrt{n})$ queries.

5.5 Comparison with other lower bounds

In this section, we establish the separations between quantum sabotage complexity and the adversary bound (and the polynomial method) claimed in Theorem 5.3.

To prove this, we will compose known functions with the index function and establish the behavior of quantum sabotage complexity under composition with the index function.

5.5.1 Index functions

Let $\mathrm{IND}_k : \{0,1\}^{k+2^k} \to \{0,1\}$ denote the index function, the function that on input (x, y) with $x \in \{0,1\}^k$ and $y \in \{0,1\}^{2^k}$, outputs the bit of y indexed by the string x. We wish to study the composition of the index function with an arbitrary Boolean function f, but composed only on the first k bits of the index function. We'll denote this composition by $\mathrm{IND}_k \circ_k f$. More precisely, if f is an n-bit function, $\mathrm{IND}_k \circ_k f$ is a function on $nk + 2^k$ bits that evaluates f on the first k n-bit strings to obtain a binary string x of length k, and then uses x to index into the next 2^k bits of the input and outputs the bit indexed by x.

In addition to the index function, which is total, we will also study a function we call the "unambiguous index function," UIND_k. This is a partial function defined similarly to the index function, except that the location of the array y pointed to by the first part of the input is "marked," and we are promised that no other bits of the array are "marked." More explicitly, the function is defined on $k + 2 \cdot 2^k$ bits, with the first k bits indexing a pair of adjacent bits in the remainder of the input. So if the first part of the input represents the integer x, that means it points to the cells $2x$ and $2x + 1$ in the second part of the input. The output of UIND_k is the first bit of the pair pointed to, i.e., it will be the bit stored at array location $2x$. Moreover, we are promised that the second bit of this pair (the bit at array location $2x+1$) will always be 1, and also that the second bit in every *other* pair (i.e., other than the pair $2x$, $2x + 1$) will always be 0.

Intuitively, there is only one strategy to solve IND_k, which is to read the first k bits and find the cell pointed to. But to solve UIND_k, there are two good strategies: either read the first k bits (and determine x), or search the remainder of the input for the unique position where the second bit of a pair is 1, which marks the cell pointed to by x.

5.5.2 Index function composition

We now examine the behavior of quantum sabotage complexity under composition with the Index and Unambiguous Index functions. To prove our result, we need the following strong direct product theorem for quantum query complexity due to Lee and Roland [59]:

Theorem 5.15 (Strong direct product)**.** *Let f be a partial Boolean function with $\mathrm{Dom}(f) \subseteq \{0,1\}^n$, and let $f^{(k)} : \mathrm{Dom}(f)^k \to \{0,1\}^k$ be the task of solving k independent inputs to f simultaneously. Then any quantum algorithm that solves $f^{(k)}$ with success probability at least $(5/6)^k$ uses $\Omega(k\,\mathrm{Q}(f))$ queries.*

We can now prove our composition theorems.

Theorem 5.16. *There is a constant c such that for any partial function f, if $k \geq c \log \mathrm{Q}(f)$, then*

$$\mathrm{QS}(\mathrm{IND}_k \circ_k f) = \Theta(\mathrm{Q}(\mathrm{IND}_k \circ_k f)) = \Theta(k\,\mathrm{Q}(f))$$

$$\mathrm{QS}(\mathrm{UIND}_k \circ_k f) = \Theta(\mathrm{Q}(\mathrm{UIND}_k \circ_k f)) = \Theta(k\,\mathrm{Q}(f)).$$

In other words, composing a function with a large enough index gadget turns QS *into* Q.

Proof. Recall that quantum query complexity composes perfectly [58], so $\mathrm{Q}(\mathrm{IND}_k \circ f) = \Theta(\mathrm{Q}(\mathrm{IND}_k)\,\mathrm{Q}(f)) = O(k\,\mathrm{Q}(f))$. We argue that $\mathrm{Q}(\mathrm{IND}_k \circ_k f)$ is smaller than $\mathrm{Q}(\mathrm{IND}_k \circ f)$. This is because we can convert any algorithm for $\mathrm{Q}(\mathrm{IND}_k \circ f)$ into an algorithm for $\mathrm{Q}(\mathrm{IND}_k \circ_k f)$: fix a 0-input x^0 and a 1-input x^1 for f; then, given an input to $\mathrm{Q}(\mathrm{IND}_k \circ_k f)$, pretend that each 0 bit in the second half of the input is actually x^0, and that each 1 bit is actually x^1 (the algorithm can do this by applying the appropriate unitary). This converts the input into an input for $\mathrm{Q}(\mathrm{IND}_k \circ f)$, completing the reduction.

Thus $\mathrm{Q}(\mathrm{IND}_k \circ_k f) = O(k\,\mathrm{Q}(f))$. Similarly, $\mathrm{Q}(\mathrm{UIND}_k \circ_k f) = O(k\,\mathrm{Q}(f))$. Since QS is smaller than Q, it remains only to show that $\mathrm{QS}(\mathrm{IND}_k \circ_k f) = \Omega(k\,\mathrm{Q}(f))$ and $\mathrm{QS}(\mathrm{UIND}_k \circ_k f) = \Omega(k\,\mathrm{Q}(f))$. We complete the argument for UIND; the argument for IND is similar.

Let Q be an optimal quantum sabotage algorithm for $\mathrm{UIND}_k \circ_k f$. We turn Q into a quantum algorithm Q' that uses the same number of queries, and solves all k copies of f with non-negligible probability; we then apply the direct product theorem (Theorem 5.15) to lower bound the number of queries required by Q', and hence by Q.

Given k inputs to f, the first thing the algorithm Q' does is append an all-0 array to turn it into an input to $\text{UIND}_k \circ_k f$. (Since the array is all zeros, the new input does not satisfy the promise of $\text{UIND}_k \circ_k f$, but we will still be able to run Q on it.) Then Q' picks a random number t between 1 and T uniformly, where $T = \text{QS}(\text{UIND}_k \circ_k f)$ is the number of queries used by Q, and simulates Q for t queries. The algorithm Q' then measures the state of Q to determine the position at which Q was going to query. If this position is in the array part of the input and is inside a pair that has index $i \in \{0,1\}^k$, the algorithm Q' will then output the string i.

Consider the correct pair in the array (the one really pointed to by the k copies of f). Flipping the pair from 00 to 01 causes the input to satisfy the promise of $\text{UIND}_k \circ_k f$, and causes the output to become a 0-input. On the other hand, flipping the pair from 00 to 11 causes the input to become a 1-input. Let $|\psi\rangle$ be the final state of Q when run on the original, illegal input. Let $|\psi_0\rangle$ be the final state of Q when run on the flipped 0-input, and let $|\psi_1\rangle$ be the final state of Q when run on the 1-input. We know that $|\psi_0\rangle$ and $|\psi_1\rangle$ are far in trace distance. Hence $|\psi\rangle$ must be a far in trace distance from at least one on them.

Thus by Lemma 5.11, the probability that Q' finds Q querying inside the correct pair of the array is $\Omega(1/T^2)$. This means that Q' outputs the correct string of answers to the k inputs to f is with probability at least $\Omega(1/T^2)$. Since Q' uses only T queries, by Theorem 5.15 we must have either $T = \Omega(k\,\text{Q}(f))$ or $1/T^2 = O((5/6)^k)$. The latter implies $T = \Omega((6/5)^{k/2}) = \Omega((6/5)^{k/4} \cdot (6/5)^{k/4}) = 2^{\Omega(k)} \cdot 2^{\Omega(k)}$. When $k \geq c \log \text{Q}(f)$ for a large enough constant c, this gives $T \geq 2^{\Omega(k)}\,\text{Q}(f) = \Omega(k\,\text{Q}(f))$. Recalling that $T = \text{QS}(\text{UIND}_k \circ_k f)$, we get $\text{QS}(\text{IND}_k \circ_k f) = \Omega(k\,\text{Q}(f))$, as desired. □

5.5.3 Separations

Using this theorem we can now establish Theorem 5.3, restated for convenience:

Theorem 5.3. *There exist total functions f and g with*

$$\text{QS}(f) = \tilde{\Omega}(\text{Adv}(f)^2) \text{ and } \text{QS}(g) \geq \widetilde{\deg}(g)^{4-o(1)}. \tag{5.3}$$

There also exists an n-bit partial function h with

$$\text{QS}(h) = \tilde{\Omega}(n^{1/3}) \text{ and } \text{Adv}(h) = O(\log n). \tag{5.4}$$

Proof. There exists an n-bit total function f' with a quadratic separation between quantum query complexity and the adversary bound, i.e., $\text{Q}(f') = \tilde{\Omega}(\text{Adv}(f')^2)$. The function is k-sum with $k \approx \log n$ (see [18, 4] for more details). Now consider the function $f = \text{IND}_k \circ f'$, where $k = \Omega(\log \text{Q}(f))$. By Theorem 5.16, the QS of these functions increases to Q. However, since the adversary bound satisfies a composition theorem [44], its value only increases by a factor of k. Thus $\text{QS}(f) = \tilde{\Omega}(\text{Adv}(f)^2)$.

Similarly, if we start with the collision problem which has $\text{Q}(h') = \Theta(n^{1/3})$ [5], but $\text{Adv}(h') = O(1)$, and define $h = \text{IND}_k \circ h'$ for $k = \Theta(\log n)$, then $\text{QS}(h) = \tilde{\Omega}(n^{1/3})$ but $\text{Adv}(f) = O(\log n)$.

There also exist total functions with $\text{Q}(g') \geq \widetilde{\deg}(g')^{4-o(1)}$ [4]. Composing this function with IND_k on the first k bits with $k = \Omega(\log \text{Q}(f))$ yields a function g with the desired separation, since approximate polynomial degree also composes in the upper bound direction [77]. □

Chapter 6

Sculpting

6.1 Introduction

This chapter is based on work that appeared in [3], which is joint work with Scott Aaronson. We introduce the concept of "sculptable" functions, which are total Boolean functions that can be restricted to a promise on which an exponential quantum speedup is possible. We characterize the sculptable Boolean functions in terms of a new query complexity measure $\mathrm{H}(\mathrm{C}_f)$, which is the H-index of the certificate complexity of the Boolean function f. We also show some related results, such as a nearly-quadratic relationship between randomized and quantum query complexity for "unbalanced" functions, as well as a nearly-quadratic relationship between deterministic and quantum query complexity for functions defined on a small domain.

More concretely, the sculpting question can be phrased as follows: given a total function $f : \{0,1\}^N \to \{0,1\}$ for which $\mathrm{R}(f)$ and $\mathrm{Q}(f)$ are both large (say, $N^{\Omega(1)}$), is there a promise $P \subseteq \{0,1\}^N$ such that $f|_P$, the restriction of f to P, has $\mathrm{Q}(f|_P) = O(\mathrm{polylog}\, N)$ and $\mathrm{R}(f|_P) = N^{\Omega(1)}$?

For example, if f is the OR function, such sculpting is not possible, as follows from [1]. As another example, if f is defined to be 1 when Simon's condition is satisfied and 0 otherwise, then sculpting is possible: the promise will simply restrict to inputs that either satisfy Simon's condition or are far from satisfying it; this promise suffices for an exponential quantum speedup [27].

We fully characterize the functions f for which such a promise exists. In particular, we show that sufficiently "rich" functions, such as PARITY or MAJORITY, are sculptable.

The sculpting problem has been previously studied in [90] for the case where f a recursive function such as the NAND-tree. They constructed a promise on which this function gives a small super-polynomial speedup ($\mathrm{polylog}(n)$ vs. $(\log n)^{\Omega(\log\log\log n)}$). Our results also apply to such recursive functions, and we improve the speedup to $\mathrm{polylog}\, n$ vs. $n^{\Omega(1)}$.

Our sculpting construction uses communication complexity in a novel way. In the other direction, to prove non-sculptability, we prove new query complexity relationships. As a corollary, we get nearly quadratic relationships between classical and quantum query complexities for a wider class of functions than previously known.

Results

H-indices

We introduce a new query complexity measure, $\mathrm{H}(\mathrm{C}_f)$, defined as the maximum number h for which there are 2^h inputs to f with certificate complexity at least h. We call this the H-index of certificate complexity (motivated by the citation H-index sometimes used to measure research productivity [43]). This quantity measures the number of inputs there are to a function f that have large certificate complexity. We prove various properties of $\mathrm{H}(\mathrm{C}_f)$; most notably, we show that for total functions, it is nearly quadratically related to $\mathrm{H}(\mathrm{bs}_f)$, the H-index of block sensitivity. This is analogous to the quadratic relationship between C and bs.

Sculpting in Query Complexity

Our main result is the following theorem, which neatly characterizes sculptability in the query complexity model in terms of the H-index of certificate complexity.

Theorem 6.1. *Let $f : \{0,1\}^N \to \{0,1\}$ be a total function. Then there is a promise $P \subseteq \{0,1\}^N$ such that $\mathrm{R}(f|_P) = N^{\Omega(1)}$ and $\mathrm{Q}(f|_P) = N^{o(1)}$, if and only if $\mathrm{H}(\mathrm{C}_f) = N^{\Omega(1)}$. Furthermore, in this case we also have $\mathrm{Q}(f|_P) = O(\log^2 N)$.*

This theorem follows as an immediate corollary of the following more general characterization theorem.

Theorem 6.2. *For all total functions $f : \{0,1\}^N \to \{0,1\}$ and all promises $P \subseteq \{0,1\}^N$, we have*

$$\mathrm{R}(f|_P) = O(\mathrm{Q}(f|_P)^2 \,\mathrm{H}(\mathrm{C}_f)^2).$$

Conversely, for all total functions $f : \{0,1\}^N \to \{0,1\}$, there is a promise $P \subseteq \{0,1\}^N$ such that

$$\mathrm{R}(f|_P) = \Omega\left(\frac{\mathrm{H}(\mathrm{C}_f)^{1/6}}{\log^{11/6} N}\right) \qquad \text{and} \qquad \mathrm{Q}(f|_P) = O(\log^2 \mathrm{H}(\mathrm{C}_f)).$$

We also prove an analogous theorem for D vs. R_0, showing that the same $\mathrm{H}(\mathrm{C}_f) = N^{\Omega(1)}$ condition also characterizes sculpting $\mathrm{D}(f)$ vs. $\mathrm{R}_0(f)$. On the other hand, we show that sculpting $\mathrm{R}_0(f)$ vs. $\mathrm{R}(f)$ is *always* possible: for every total function f with $\mathrm{R}_0(f) = N^{\Omega(1)}$, there is a promise P such that $\mathrm{R}_0(f|_P) = N^{\Omega(1)}$ and $\mathrm{R}(f|_P) = O(1)$.

Query Complexity on Small Promises

On the way to proving Theorem 6.2, we prove the following theorem, providing a quadratic relationship between $\mathrm{Q}(f)$ and $\mathrm{D}(f)$ when the domain of f is small. This provides a ironic twist to the query complexity story: for a long time, it was believed that $\mathrm{D}(f)$ and $\mathrm{Q}(f)$ are quadratically related when the domain of f is very large (in particular, for total functions). This conjecture was recently disproven by [9] (who showed a $\mathrm{D}(f) \sim \mathrm{Q}(f)^4$ separation) and by Chapter 3 (where we showed an f such that $\mathrm{R}(f) \sim \mathrm{Q}(f)^{2.5}$). Instead, we now show that the quadratic relationship holds when the domain of f is very *small*.

Theorem 6.3. *Let $f : \{0,1\}^N \to \{0,1\}$ be a partial function, and let $\mathrm{Dom}(f)$ denote the domain of f. Then*

$$\mathrm{Q}(f) = \Omega\left(\frac{\sqrt{\mathrm{D}(f)}}{\log|\mathrm{Dom}(f)|}\right).$$

Query Complexity for Unbalanced Functions

We show two relationships similar to Theorem 6.3 that hold for functions whose domain is large, but which are unbalanced: they contain very few 0-inputs compared to 1-inputs, or vice versa.

Theorem 6.4. *Let* $f : \{0,1\}^N \to \{0,1\}$ *be a partial function. Define the measure* $\mathrm{Bal}(f) \in [0, N]$ *as* $\mathrm{Bal}(f) := 1 + \min\{\log |f^{-1}(0)|, \log |f^{-1}(1)|\}$ *(or* 0 *if* f *is constant). Then*

$$\mathrm{R}(f) = O(\mathrm{Q}(f)^2 \mathrm{Bal}(f))$$

$$\mathrm{D}(f) = O(\mathrm{R}_0(f) \mathrm{Bal}(f)).$$

A similar polynomial relationship between $\mathrm{R}_0(f)$ *and* $\mathrm{R}(f)$ *does not hold in general.*

New Relationship for Total Functions

We prove the following new query complexity relationship for total functions, generalizing the known relationship $\mathrm{D}(f) = O(\mathrm{Q}(f)^2 \mathrm{C}(f))$.

Theorem 6.5. *Let* $f : \{0,1\}^N \to \{0,1\}$ *be a total function. Then*

$$\mathrm{D}(f) = O(\mathrm{Q}(f)^2 \mathrm{H}(\sqrt{\mathrm{C}_f})^2).$$

Here $\mathrm{H}(\sqrt{\mathrm{C}_f})$ denotes the H-index of the square root of certificate complexity; this is the maximum number h such that there are at least 2^h inputs to f for which $\sqrt{\mathrm{C}(f)}$ is at least h. We note that $\mathrm{H}(\sqrt{\mathrm{C}_f})^2 \leq \mathrm{C}(f)$ for all total functions, so this is an improvement over the relationship $\mathrm{D}(f) = O(\mathrm{Q}(f)^2 \mathrm{C}(f))$. Moreover, when $f = \mathrm{OR}$, we have $\mathrm{H}(\sqrt{\mathrm{C}_f})^2 = 1$ and $\mathrm{C}(f) = N$, so this improvement is strict.

We remark that this result could let us improve the relationship $\mathrm{D}(f) = O(\mathrm{Q}(f)^6)$ if we could show $\mathrm{H}(\sqrt{\mathrm{C}_f})^2 = o(\mathrm{Q}(f)^4)$. Theorem 6.5 therefore provides a new approach for this long-standing open problem.

Sculpting in the Turing Machine Model

In Section 6.8, we examine sculpting in the Turing machine model. We say that a language L is *sculptable* if there is a promise set P such that the promise problem of deciding if an input from P is in L is in PromiseBQP but not in PromiseBPP. We prove two sculptability theorems, both of them providing evidence that most or all languages outside of BPP are sculptable.

Theorem 6.6. *Assume* PromiseBQP *is hard on average for* P/poly. *Then every paddable language outside of* BPP *is sculptable.*

Theorem 6.7. *Assume there exists a* BPP*-bi-immune language in* BQP. *Then every language outside of* BPP *is sculptable.*

For the definitions of paddability and bi-immunity, see Section 6.8. These theorems assume very little about BQP and BPP, and analogous statements hold for other pairs of complexity classes.

6.2 Notation and Preliminaries

Instead of denoting the certificate complexity of an input $x \in \mathrm{Dom}(f)$ by $\mathrm{C}_x(f)$, in this chapter we use the notation $\mathrm{C}_f(x)$. This way, we can treat C_f as a set of certificate complexities for f, which will allow us to use the notation $\mathrm{H}(\mathrm{C}_f)$ to denote the H-index of this set of numbers. We use similar notation for RC_f and bs_f.

6.2.1 Balance and H Indices

We will use $\mathrm{Dom}(f)$ to denote the domain of a partial function f. We define $\mathrm{Bal}(f)$ to be 0 if f is constant, and otherwise, to be the minimum of $1+\log|f^{-1}(0)|$ and $1+\log|f^{-1}(1)|$ (we use log to denote logarithm base 2). Note that since $|f^{-1}(0)| + |f^{-1}(1)| = |\mathrm{Dom}(f)| \leq 2^N$, we have $\mathrm{Bal}(f) \leq N$. Thus $\mathrm{Bal}(f) \in [0, N]$.

We will use a new set of query complexity measures called H-indices (the name is motivated by the H-index measure of citations, a common metric for research output). For a given function $g : \{0,1\}^N \to [0,\infty)$, we will define the H-index of g, denoted by $\mathrm{H}(g)$, as the maximum number h such that there are at least 2^h inputs with $g(x) \geq h$. Alternatively, the H-index of g can be defined as the minimum number h such that there are at most 2^h inputs with $g(x) > h$. It is not obvious that these definitions are equivalent (or even that the minimum and maximum are attained); we prove this in Appendix D.

Note that $\mathrm{H}(g) \in [0, N]$, and $\mathrm{H}(g) \leq \max_x g(x)$. Also, if $g(x) \geq g'(x)$ for all $x \in \{0,1\}^N$, we have $\mathrm{H}(g) \geq \mathrm{H}(g')$.

We'll primarily be interested in measures like $\mathrm{H}(\mathrm{C}_f)$, $\mathrm{H}(\mathrm{RC}_f)$, and $\mathrm{H}(\mathrm{bs}_f)$. We have $\mathrm{H}(\mathrm{C}_f) \leq \mathrm{C}(f)$, $\mathrm{H}(\mathrm{RC}_f) \leq \mathrm{RC}(f)$, and $\mathrm{H}(\mathrm{bs}_f) \leq \mathrm{bs}(f)$. We also have

$$\mathrm{H}(\mathrm{bs}_f) \leq \mathrm{H}(\mathrm{RC}_f) \leq \mathrm{H}(C_f).$$

The H-index of certificate complexity can be much smaller than the certificate complexity itself. For example, the OR function has only one certificate of size greater than 1, so $\mathrm{H}(\mathrm{C}_{\mathrm{OR}}) = 1$, even though $\mathrm{C}(\mathrm{OR}) = n$.

In Appendix D we show that if $\alpha : [0,\infty) \to [0,\infty)$ is an increasing function, then

$$\mathrm{H}(\alpha \circ g) \leq \max\{\mathrm{H}(g), \alpha(\mathrm{H}(g))\}.$$

In particular, this will imply $\mathrm{H}(\mathrm{C}_f^2) \leq \mathrm{H}(\mathrm{C}_f)^2$.

6.2.2 Shattering and the Sauer-Shelah Lemma

For a set of indices $A \subseteq \{1, 2, \ldots, N\}$, let $S|_A \subseteq \{0,1\}^{|A|}$ be the set of restrictions of each string in S to the indices in A. We say A is shattered by S if $S|_A = \{0,1\}^{|A|}$. In other words, A is shattered by S if S has all possible behaviors on A. The Sauer-Shelah lemma [75, 76] is a classic result that upper-bounds the size of S in terms of the size of A. We will use the following corollary of it.

Lemma 6.8. *Let $S \subseteq \{0,1\}^N$ be a collection of strings. Then there is a shattered set of indices of size at least*

$$\frac{\log|S|}{\log(N+1)}.$$

Lemma 6.8 follows straightforwardly from the Sauer-Shelah lemma [75, 76]. We will often use the weaker bound $\frac{\log|S|}{2\log N}$ instead, which holds for $N \geq 2$. This will sometimes lead to simpler formulas.

6.3 Non-Sculptability Theorems

In this section, we prove the non-sculptability direction of Theorem 6.2. The proof has two parts: in Section 6.3.1, we prove a relationship between randomized and quantum query complexities for "unbalanced" functions, and in Section 6.3.2, we use this to prove a sculpting lower bound in terms of the H-index of certificate complexity.

6.3.1 Query Complexity for Unbalanced Functions

We wish to show a nearly-quadratic relationship between randomized and quantum query complexities for functions f for which $\mathrm{Bal}(f)$ is small. Note that this is a generalization of the relation $\mathrm{RC}_f(x) = O(\mathrm{Q}_f(x)^2)$ from [1]. That is, [1] showed that for the task of distinguishing one input from a (possibly large) set of alternatives, randomized and quantum algorithms are quadratically related. We want a similar relationship for the task of distinguishing a *small set* of inputs from a (possibly large) set of alternatives.

We start with the following lemma.

Lemma 6.9. *Let $f : \{0,1\}^N \to \{0,1\}$ be a partial function. For $a \notin f^{-1}(0)$, let $f_{a,0}$ be the problem of distinguishing a from $f^{-1}(0)$. That is, $f_{a,0}$ is the function $f_{a,0} : \{a\} \cup f^{-1}(0) \to \{0,1\}$ with $f(x) = 1$ iff $x = a$. For $a \notin f^{-1}(1)$, define $f_{a,1}$ analogously. Then for all $a \in \{0,1\}^N$, we have either $\mathrm{R}(f_{a,0}) = O(\mathrm{Q}(f)^2)$ or $\mathrm{R}(f_{a,1}) = O(\mathrm{Q}(f)^2)$.*

Note that this holds even when a is not in the promise of f. The constant in the big-O notation is a universal constant independent of a, f, and N.

Proof. Let Q be the quantum algorithm that achieves $\mathrm{Q}(f)$ quantum query complexity in determining the value of f on a given input. When run on any $a \in f^{-1}(0)$, Q will output 0 with probability at least $2/3$, and when run on $a \in f^{-1}(1)$, it will output 1 with probability at least $2/3$.

Consider running Q on an input $a \notin \mathrm{Dom}(f)$. Then Q will output 0 with some probability p and output 1 with probability $1-p$. If $p \geq 1/2$, then Q distinguishes a from $f^{-1}(1)$ with constant probability. If $p \leq 1/2$, then Q distinguishes a from $f^{-1}(0)$ with constant probability. Thus for all $a \in \{0,1\}^N$, we have either $\mathrm{Q}(f_{a,0}) = O(\mathrm{Q}(f))$ or $\mathrm{Q}(f_{a,1}) = O(\mathrm{Q}(f))$. From [1], we have $\mathrm{RC}(g) = O(\mathrm{QC}(g)^2) = O(\mathrm{Q}(g)^2)$ for all functions g, so we conclude that either $\mathrm{RC}(f_{a,0}) = O(\mathrm{Q}(f)^2)$ or $\mathrm{RC}(f_{a,1}) = O(\mathrm{Q}(f)^2)$.

Finally, note that for a problem of distinguishing one input from the rest, randomized query complexity equals randomized certificate complexity. Thus we get that for all $a \in \{0,1\}^N$, either $\mathrm{R}(f_{a,0}) = O(\mathrm{Q}(f)^2)$ and or $\mathrm{R}(f_{a,1}) = O(\mathrm{Q}(f)^2)$. □

We're now ready to prove the desired relationship between R and Q.

Theorem 6.10. *Let $f : \{0,1\}^N \to \{0,1\}$ be a partial function. Then*

$$\mathrm{R}(f) = O(\mathrm{Q}(f)^2 \mathrm{Bal}(f)).$$

Proof. Without loss of generality, assume $|f^{-1}(0)| \leq |f^{-1}(1)|$. We use Lemma 6.9 to construct a randomized algorithm for determining $f(x)$ given oracle access to x, assuming that

$f^{-1}(0)$ is small. The idea is to keep track of the subset $Z \subseteq f^{-1}(0)$ of strings that the input x might feasibly be (consistent with the queries seen so far). We then construct a string a from a majority vote of the elements of Z; that is, for each index $i \in [n]$, a_i will be the majority of y_i over all $y \in Z$ (with ties broken arbitrarily).

This string a need not be in $\mathrm{Dom}(f)$. The important property of it is that if we query an index i of the input x and discover that $x_i \neq a_i$, we can eliminate at least half of the strings from Z, since they are no longer feasible possibilities for x.

We then get the following randomized algorithm for evaluating $f(x)$:

- Initialize $Z = f^{-1}(0)$.
- While $Z \neq \emptyset$:
  1. Calculate a from the entry-wise majority vote of Z.
  2. Pick $b \in \{0,1\}$ such that $\mathrm{R}(f_{a,b}) = O(\mathrm{Q}(f)^2)$ (this exists by Lemma 6.9).
  3. Run the randomized algorithm evaluating $f_{a,b}$ on x with some amplification (to be specified later).
  4. If its output is 1 (i.e. the algorithm thinks $x = a$ rather than $x \in f^{-1}(b)$), output $1 - b$ and halt.
  5. If its output is 0, a bit i was queried to reveal $x_i \neq a_i$, so update Z (removing at least half its elements).
- If $Z = \emptyset$, output 1.

We note a few things about this algorithm. First, in step 3, notice that x need not be in the domain of $f_{a,b}$. However, we may still run the randomized algorithm that evaluates $f_{a,b}$, and use the fact that if x does happen to be in the domain (in particular, if $x \in f^{-1}(b)$), then the algorithm will work correctly. This is exactly what we use in step 4: if the algorithm that distinguishes a from $f^{-1}(b)$ says that x is equal to a, it need not mean that x is in fact equal to a, but it does mean that $x \notin f^{-1}(b)$.

Secondly, step 5 assumes that the randomized algorithm for evaluating $f_{a,b}$ will only conclude that an input x is not equal to a if it finds a disagreement with a. This is a safe assumption, as argued in Lemma 5 of [1].

Finally, we determine the number of queries this algorithm uses. The outer loop happens at most $\lfloor \log |f^{-1}(0)| \rfloor + 1 \leq \mathrm{Bal}(f)$ times. Step 3 in the loop is the only one which queries the input string. Since the loop repeats at most $\mathrm{Bal}(f)$ times, we can safely amplify the algorithm in step 3 $O(\log \mathrm{Bal}(f))$ times. This gives a query complexity of $O(Q(f)^2 \log \mathrm{Bal}(f))$ for step 3, so the overall number of queries is $O(Q(f)^2 \mathrm{Bal}(f) \log \mathrm{Bal}(f))$.

We can get rid of the log factor by being more careful with the amplification. Note that if we ever find a disagreement with a when running the algorithm, we may immediately stop amplifying and proceed to step 5. We keep a count c_0 of how many times we had to amplify in step 3 for functions of the form $f_{a,0}$, and a count c_1 for functions of the form $f_{a,1}$.

If c_0 ever reaches $2\,\mathrm{Bal}(f)$, we output 1 and halt. Similarly, if c_1 ever reaches $2\,\mathrm{Bal}(f)$, we output 0 and halt. This ensures the total amplification is $O(\mathrm{Bal}(f))$, so the total query complexity of the algorithm is $O(Q(f)^2 \mathrm{Bal}(f))$.

Note that if $f(x) = 0$ and the output of the algorithm was 1, it means that we ran the algorithm evaluating $f_{a,0}$ (for varying values of a) $2\,\mathrm{Bal}(f)$ times, and at most $\mathrm{Bal}(f)$ of those times the algorithm said that $x \in f^{-1}(0)$. For each individual run, the probability is

at least $2/3$ that the algorithm would say that $x \in f^{-1}(0)$. An application of the Chernoff bound shows that the probability of this happening is exponentially small. Similarly, the probability of the algorithm giving 0 when in actuality $f(x) = 1$ is also exponentially small.

We conclude that $R(f) = O(Q(f)^2 \operatorname{Bal}(f))$, as desired. □

6.3.2 Application to Non-Sculptability

Theorem 6.10 immediately gives the following non-sculptability result, which says that unbalanced functions cannot be sculpted.

Corollary 6.11. *Let $f : \{0,1\}^N \to \{0,1\}$ be a total function. For any promise $P \subseteq \{0,1\}^N$, we have*

$$\mathrm{R}(f|_P) = O(\mathrm{Q}(f|_P)^2 \operatorname{Bal}(f)).$$

Proof. Note that $\operatorname{Bal}(f|_P) \leq \operatorname{Bal}(f)$ for any f and P. Then, by Theorem 6.10, we have

$$\mathrm{R}(f|_P) = O(\mathrm{Q}(f|_P)^2 \operatorname{Bal}(f|_P)) = O(\mathrm{Q}(f|_P)^2 \operatorname{Bal}(f)).$$

□

We extend this result by showing that any function with a small number of large certificates also cannot be sculpted. This gives us a non-sculptability result in terms of the H-index of certificate complexity.

Theorem 6.12. *Let $f : \{0,1\}^N \to \{0,1\}$ be a total function. For any promise $P \subseteq \{0,1\}^N$, we have*

$$\mathrm{R}(f|_P) = O(\mathrm{Q}(f|_P)^2 \mathrm{H}(\mathrm{C}_f^2)).$$

Proof. We design a deterministic algorithm that reduces the set of possibilities for the input to an unbalanced set. Specifically, the algorithm will reduce the possibilities for the input to a set $S \subseteq \{0,1\}^N$ such that $\operatorname{Bal}(f|_S) \leq \mathrm{H}(\mathrm{C}_f^2) + 1$. We then use Theorem 6.10 to get the desired non-sculptability result.

Note that every 1-certificate of f must conflict with every 0-certificate of f in at least one bit. Therefore, by querying all non-$*$ entries of a 0-certificate, we reveal at least one entry of each 1-certificate.

We design a deterministic algorithm for computing f on an input from P. The algorithm proceeds as follows: it repeatedly picks a 0-certificate p for f of size at most $\sqrt{\mathrm{H}(\mathrm{C}_f^2)}$ that is consistent with all the entries of the input that were revealed so far. It then queries all the non-$*$ entries of p. This is repeated $\sqrt{\mathrm{H}(\mathrm{C}_f^2)}$ times, or until there are no 0-certificates of size at most $\sqrt{\mathrm{H}(\mathrm{C}_f^2)}$ (whichever happens first). Finally, the algorithm returns the set S of strings that are consistent with the revealed entries of the input.

This algorithm uses at most $\mathrm{H}(\mathrm{C}_f^2)$ queries. We check its correctness by examining the set S. Clearly, the input is in S. Furthermore, if any certificate of f was revealed, then f is constant on S, so S contains either no 0-inputs or no 1-inputs.

There are at most $2^{\mathrm{H}(\mathrm{C}_f^2)}$ inputs with certificate complexity larger than $\sqrt{\mathrm{H}(\mathrm{C}_f^2)}$.

If the algorithm terminated because there were no consistent 0-certificates, then the only 0-inputs in S have certificates of size larger than $\sqrt{\mathrm{H}(\mathrm{C}_f^2)}$. There are at most $2^{\mathrm{H}(\mathrm{C}_f^2)}$ of them, so S has at most $2^{\mathrm{H}(\mathrm{C}_f^2)}$ 0-inputs to f. Conversely, if the algorithm went through $\sqrt{\mathrm{H}(\mathrm{C}_f^2)}$

iterations of querying consistent 0-certificates, then it must have revealed $\sqrt{\mathrm{H}(\mathrm{C}_f^2)}$ entries of each 1-certificate to f. If no 1-certificate was discovered, it means the revealed entries contradicted all 1-certificates of size at most $\sqrt{\mathrm{H}(\mathrm{C}_f^2)}$. Thus the only 1-inputs in S have certificate size greater than $\sqrt{\mathrm{H}(\mathrm{C}_f^2)}$, from which it follows that there are less than $2^{\mathrm{H}(\mathrm{C}_f^2)}$ of them.

We conclude that S contains either at most $2^{\mathrm{H}(\mathrm{C}_f^2)}$ 0-inputs to f or at most $2^{\mathrm{H}(\mathrm{C}_f^2)}$ 1-inputs to f. This gives $\mathrm{Bal}(f|_S) \leq \mathrm{H}(\mathrm{C}_f^2) + 1$.

We design a randomized algorithm for $f|_P$ as follows. First, we run the above deterministic algorithm to reduce the problem of computing $f|_P$ to the problem of computing $f|_{S \cap P}$. This costs $\mathrm{H}(\mathrm{C}_f^2)$ queries. By Theorem 6.10, there is a randomized algorithm that uses

$$O(\mathrm{Q}(f|_{S\cap P})^2 \,\mathrm{Bal}(f|_{S\cap P})) = O(\mathrm{Q}(f|_P)^2 \,\mathrm{Bal}(f|_S)) = O(\mathrm{Q}(f|_P)^2 \,\mathrm{H}(\mathrm{C}_f^2))$$

queries to compute $f|_{S\cap P}$. Running this algorithm allows us to compute $f|_P$. The total number of queries used was

$$O(\mathrm{Q}(f|_P)^2 \,\mathrm{H}(\mathrm{C}_f^2) + \mathrm{H}(\mathrm{C}_f^2)) = O(\mathrm{Q}(f|_P)^2 \,\mathrm{H}(\mathrm{C}_f^2)).$$

□

Note that Theorem 6.12 completes the first part of the proof of Theorem 6.2, since $\mathrm{H}(\mathrm{C}_f^2) \leq \mathrm{H}(\mathrm{C}_f)^2$. It is natural to wonder whether Theorem 6.12 is always at least as strong as Corollary 6.11. In Theorem 6.22, we will show that it is, up to a quadratic factor and a $\log N$ factor.

6.4 Sculpting from Communication Complexity

In this section, we show that if a function f has many inputs with large randomized certificate complexity then it *can* be sculpted: there is a promise P so that $f|_P$ exhibits a large quantum speedup. This means that if $\mathrm{H}(\mathrm{RC}_f)$ is large, the function f can be sculpted. In Section 6.5, we will relate $\mathrm{H}(\mathrm{RC}_f)$ to $\mathrm{H}(\mathrm{C}_f)$, thereby completing the proof of Theorem 6.2.

Our sculptability proof will rely on the solution to a problem we call the "extended queries problem," which might be of independent interest. The solution to this problem will in turn use results from communication complexity.

6.4.1 The Extended Queries Problem

We usually let an algorithm for computing a (possibly partial) function $f : \{0,1\}^N \to \{0,1\}$ query the bits of the input x. But what happens if we let the algorithm make other types of queries? For example, if x is a Boolean string, we can let the algorithm query the parity of x. How does this extra power affect the query complexity of f? In particular, is there some special set of additional queries such that if a randomized algorithm is allowed to make the special queries, it can simulate *any* quantum algorithm? If so, how many special queries suffice for this property to hold?

To formalize this question, we need a few definitions.

Definition 6.13. *An* extension function with extension G *is an injective total function* $\phi : \{0,1\}^N \to \{0,1\}^G$ *(in particular, we need* $G \geq N$*).*

An extension function specifies, for each input $x \in \{0,1\}^N$, the types of queries an algorithm is allowed to make on x. In other words, we will let algorithms query from $\phi(x)$ instead of from x. Note that the extension function may provide easy access to information about x that is hard to obtain otherwise (such as its parity).

Definition 6.14. *Let $f : \{0,1\}^N \to \{0,1\}$ be a partial function, and let ϕ be an extension function. The* extended version of f with respect to ϕ *is the partial function $f^\phi : \phi(\mathrm{Dom}(f)) \to \{0,1\}$ defined by $f^\phi(x) = f(\phi^{-1}(x))$.*

Note that f^ϕ is a partial function from $\{0,1\}^G$ to $\{0,1\}$. We can consider $\mathrm{D}(f^\phi)$, $\mathrm{R}(f^\phi)$, $\mathrm{Q}(f^\phi)$, and so on. To pose the extended queries problem, we will need a notion of the complexity of a set of functions, defined as the maximum complexity of any function in that set.

Definition 6.15. *For any set of functions S, we define $\mathrm{D}(S) := \max_{f \in S} \mathrm{D}(f)$. We define $\mathrm{R}(S)$, $\mathrm{Q}(S)$, etc. similarly. Further, we define $\mathrm{D}^G(S)$, the* extended query complexity of S with extension G, *to be the minimum, over all extension functions ϕ with extension G, of $\max_{f \in S} \mathrm{D}(f^\phi)$. We define $\mathrm{R}^G(S)$, $\mathrm{Q}^G(S)$, etc. similarly.*

In other words, for any set of functions, the extended query complexity of the set with G extension is the number of queries required to compute all functions in the set given the best possible extension. We observe that if $G \geq |S|$, the extended query complexity $\mathrm{D}^G(S)$ is 1, since the extension $\phi(x)$ for a given input x could simply specify the values of all the functions in S on x. We also observe that for all $G \geq N$, we have $\mathrm{D}^G(S) \leq \mathrm{D}(S)$, since the identity function is always a valid extension function. Moreover, the extended query complexity of a set is decreasing in G. We now ask the following question.

The Extended Queries Problem. Is there a set of functions S for which $\mathrm{Q}(S)$ is small but $\mathrm{R}^G(S)$ is large, even when the extension G is exponentially large in the input size N? We can also ask this question for other complexity measures, such as $\mathrm{R}(S)$ vs. $\mathrm{R}_0^G(S)$ or $\mathrm{R}_0(S)$ vs. $\mathrm{D}^G(S)$.

It turns out that a positive solution to the extended queries problem implies a sculptability result in terms of $\mathrm{H}(\mathrm{RC}_f)$, as the following theorem shows.

Theorem 6.16. *Let $f : \{0,1\}^N \to \{0,1\}$ be a total function. Let $A = \frac{\mathrm{H}(\mathrm{RC}_f)}{4 \log N}$, and let S be any set of partial functions from $\{0,1\}^A$ to $\{0,1\}$. Then there is a promise $P \subseteq \{0,1\}^N$ such that*

$$\mathrm{Q}(f|_P) = O(\mathrm{Q}(S)), \qquad \mathrm{R}(f|_P) = \Omega(\mathrm{R}^N(S)).$$

Analogous statements hold for other pairs of complexity measures, such as D and R_0 or R_0 and R.

We delay the proof of Theorem 6.16 to Section 6.4.3. First, we settle the extended queries problem for R vs. Q: Theorem 6.18 will provide an exponential lower bound on G by reducing the extended queries problem to a problem in communication complexity.

6.4.2 Reducing Extension to Communication Complexity

For a partial function $f : \{0,1\}^{N_1} \times \{0,1\}^{N_2} \to \{0,1\}$, we will denote the communication complexities of f by $\mathrm{D}^{\mathrm{CC}}(f)$, $\mathrm{R}^{\mathrm{CC}}(f)$, $\mathrm{Q}^{\mathrm{CC}}(f)$, and $\mathrm{R}_0^{\mathrm{CC}}(f)$. We will use the following definition.

Definition 6.17. *Let $f : \{0,1\}^{N_1} \times \{0,1\}^{N_2} \to \{0,1\}$ be a partial function. For any $x \in \mathrm{Dom}(f)$, we write $x = x_A x_B$, with $x_A \in \{0,1\}^{N_1}$ and $x_B \in \{0,1\}^{N_2}$. Let $\mathrm{Dom}_A(f) = \{x_A : x \in \mathrm{Dom}(f)\}$ and $\mathrm{Dom}_B(f) = \{x_B : x \in \mathrm{Dom}(f)\}$. For any $a \in \mathrm{Dom}_A(f)$, we define the marginal of f with respect to a to be the partial function $f_a : \{0,1\}^{N_2} \to \{0,1\}$ defined by $f_a(b) := f(a,b)$ for all $b \in \{0,1\}^{N_2}$ such that $(a,b) \in \mathrm{Dom}(f)$. We define $\mathrm{Mar}(f) = \{f_a : a \in \mathrm{Dom}_A(f)\}$ to be the set of all marginal functions for f.*

We now connect communication complexity to the extended queries problem.

Theorem 6.18. *Let $f : \{0,1\}^{N_1} \times \{0,1\}^{N_2} \to \{0,1\}$ be a partial function. Then for all $G \geq N$,*

$$\mathrm{R}^G(\mathrm{Mar}(f)) = \Omega\left(\frac{\mathrm{R}^{\mathrm{CC}}(f)}{\log G}\right).$$

Similarly, we have $\mathrm{D}^G(\mathrm{Mar}(f)) = \Omega(\mathrm{D}^{\mathrm{CC}}(f)/\log G)$, $\mathrm{R}_0^G(\mathrm{Mar}(f)) = \Omega(\mathrm{R}_0^{\mathrm{CC}}(f)/\log G)$, and $\mathrm{Q}^G(\mathrm{Mar}(f)) = \Omega(\mathrm{Q}^{\mathrm{CC}}(f)/\log G)$.

Proof. We prove the theorem for R. The statements for D, R_0, and Q will follow analogously. Let $\phi : \{0,1\}^{N_2} \to \{0,1\}^G$ be the best possible extension function, so that $\mathrm{R}^G(\mathrm{Mar}(f)) = \max_{g \in \mathrm{Mar}(f)} \mathrm{R}(g^\phi)$.

We now describe a randomized communication protocol for computing f. Alice receives a string a, and must compute $f(a,b)$, where b is Bob's string. This is equivalent to computing $f_a(b)$. Since Alice knows f_a, she also knows f_a^ϕ. Let R a randomized algorithm that computes f_a^ϕ using at most $\mathrm{R}^G(\mathrm{Mar}(f))$ queries. Alice will run this algorithm, and for each query, she will send the index of that query to Bob (as a number between 1 and G). Bob will reply with the corresponding bit of $\phi(y)$ (as a bit in {0,1}). This allows Alice to compute $f_a(b) = f(a,b)$.

The total communication in this protocol is at most $(\lceil \log G \rceil + 1)\,\mathrm{R}^G(\mathrm{Mar}(f))$. Since this upper-bounds the randomized communication complexity of f (using private coins), the desired result follows. □

Theorem 6.18 allows us to use communication complexity as a tool for lower-bounding the extended query complexity of certain sets of functions. To use it to solve the extended queries problem, we need a function f that has large randomized communication complexity but for which $\mathrm{Q}(\mathrm{Mar}(f))$ is small. To construct such a function, we start from a simple function that was recently shown to separate randomized from quantum communication complexity, called the Vector in Subspace problem.

The Vector in Subspace Problem. In this problem, Bob gets a unit vector $v \in \mathbb{R}^n$, and Alice gets a subspace H of $\mathbb{R}^n$ of dimension $n/2$. It is promised that either $v \in H$ or $v \in H^\perp$; the task is to determine which is the case. We assume for simplicity that n is a power of 2.

This problem was first studied in [53] and was also described in [69]. Klartag and Regev [52] showed that this problem has randomized communication complexity $\Omega(n^{1/3})$. In addition, it is easy to see that the one-way quantum communication complexity of the problem is at most $\log n$: Bob can send a superposition over $\log n$ bits with amplitudes determined by v; Alice can then apply the projective measurement given by $(H, H^\perp)$.

To apply this function to the extended queries problem, we need a few modifications. First, we need a discrete version of the problem. [52] showed that a lower bound of $\Omega(n^{1/3})$ for randomized communication complexity applies to a discrete version of the problem in which each real number is described using $O(\log n)$ bits; that is, Alice's subspace is given

using $n/2$ vectors of length n, whose entries are specified using $O(\log n)$ bits, and Bob's vector is specified using n real numbers of $O(\log n)$ bits each.

$\mathrm{Mar}(f)$ is the set of functions where we know Alice's subspace H, and are allowed to query from Bob's input vector. However, phrased this way, it is not clear how to use a quantum algorithm to compute such functions using few queries. To solve this problem, we modify the way Bob's input is specified. Instead of specifying only the entries to the vector, Bob's input string also lists some "partial sums" of the vector entries.

The idea is for Bob's vector to allow Alice to use the following algorithm to construct the state with amplitudes specified by v. We interpret v as specifying a superposition over strings of length $\log n$. Alice starts by querying the probability p that the first bit of this string is 0 when this state is measured. Alice will now place a $\sqrt{p}$ amplitude on querying the probability that the second bit is 0 conditioned on the first bit being 0, and a $\sqrt{1-p}$ amplitude on querying the probability that the second bit is 0 conditioned on the first bit being 1. Alice keeps going in this way, until she gets to the final bit of the string of length $\log n$, at which point she queries the phase. This allows her to construct the state determined by the amplitudes in v.

Of course, for this to work, Bob's input must provide all of these conditional probabilities. There is one such probability to specify for the first bit, two for the second, four for the third, and so on. Since there are $\log n$ bits, Bob's input needs to specify only $O(n)$ probabilities. Each can be specified with $O(\log n)$ precision, so Bob's total input size is $O(n \log n)$. Moreover, Alice constructs the desired state after $O(\log n)$ queries to the probabilities, or $O(\log^2 n)$ queries to the bits of Bob's input.

We thus get the following theorem.

Theorem 6.19. *For all $A \in \mathbb{N}$, there is a set S of partial functions from $\{0,1\}^A$ to $\{0,1\}$ such that for all $G \geq A$,*

$$\mathrm{Q}(S) = O(\log^2 A), \quad \mathrm{R}^G(S) = \Omega\left(\frac{A^{1/3}}{\log^{1/3} A \cdot \log G}\right).$$

Proof. Let f be the function described above with $n = A/\log A$, and let $S = \mathrm{Mar}(f)$. Then $\mathrm{Q}(S) = O(\log^2 n) = O(\log^2 A)$ and $\mathrm{R}^{\mathrm{CC}}(f) = \Omega(n^{1/3}) = \Omega(A^{1/3}/\log^{1/3} A)$. By Theorem 6.18, we get $\mathrm{R}^G(S) = \Omega(A^{1/3}/(\log^{1/3} A \cdot \log G))$. □

Together with Theorem 6.16, this implies that any function with large $\mathrm{H}(\mathrm{RC}_f)$ can be sculpted, simply by plugging S from Theorem 6.19 into Theorem 6.16 and setting $G = N$.

6.4.3 Reducing Sculpting to Extended Query Complexity

We now prove Theorem 6.16, restated here for convenience.

Theorem 6.16. *Let $f : \{0,1\}^N \to \{0,1\}$ be a total function. Let $A = \frac{\mathrm{H}(\mathrm{RC}_f)}{4 \log N}$, and let S be any set of partial functions from $\{0,1\}^A$ to $\{0,1\}$. Then there is a promise $P \subseteq \{0,1\}^N$ such that*

$$\mathrm{Q}(f|_P) = O(\mathrm{Q}(S)), \qquad \mathrm{R}(f|_P) = \Omega(\mathrm{R}^N(S)).$$

Analogous statements hold for other pairs of complexity measures, such as D *and* R_0 *or* R_0 *and* R.

Proof. There are at least $2^{\mathrm{H}(\mathrm{RC}_f \cdot 2\log N)}$ inputs $x \in \{0,1\}^N$ that satisfy the relation $\mathrm{RC}_f(x) \geq \mathrm{H}(\mathrm{RC}_f \cdot 2\log N)/(2\log N)$. Let the set of such inputs be C. By Lemma 6.8, if $N \geq 2$, there is a set B of

$$\frac{\mathrm{H}(\mathrm{RC}_f \cdot 2\log N)}{2\log N} \geq \frac{\mathrm{H}(\mathrm{RC}_f)}{2\log N}$$

indices in $\{1,2,\ldots,N\}$ that is shattered by the inputs in C. We'll restrict B to have size at most $\mathrm{H}(\mathrm{RC}_f)/(4\log N)$, so $|B| = A$. Let $\phi : \{0,1\}^A \to \{0,1\}^N$ be defined by mapping each string $x \in \{0,1\}^A$ to a string z in C such that restricting z to A gives x. This is an injective mapping, so ϕ is an extension function with extension size N.

Next, consider the set S of partial Boolean functions from $\{0,1\}^A$ to $\{0,1\}$. Let $S^\phi = \{g^\phi : g \in S\}$. Then $\mathrm{R}(S^\phi) \geq \mathrm{R}^N(S)$. It follows that there is some function $g^\phi \in S^\phi$ such that $\mathrm{R}(g^\phi) \geq \mathrm{R}^N(S)$.

We will use the function g^ϕ to define the desired promise P. The domain of g^ϕ is contained in C. Let x be in this domain, so $\mathrm{RC}_f(x) \geq \mathrm{H}(\mathrm{RC}_f \cdot 2\log N)/(2\log N) \geq 2A$. Let μ_x be a distribution over inputs y such that $f(x) \neq f(y)$, with the property that for any bit i, $\Pr[y_i \neq x_i] \leq 1/\mathrm{RC}_f(x) \leq 1/(2A)$. Then for all $x \in C$, a randomized algorithm has a hard time distinguishing between x and μ_x. For each such x, let μ'_x be the distribution μ_x conditioned on the sampled input agreeing with x on the bits in B. Since the probability of an input sampled from μ_x disagreeing with x on B is at most $|B| \cdot 1/(2A) \leq 1/2$, the distribution μ'_x is not too far from μ_x. In particular, any randomized algorithm that finds a disagreement with x on an input sampled from μ'_x with probability p will also find a disagreement with x on an input sampled from μ_x with probability at least $p/2$. It follows that a randomized algorithm must use $\Omega(A)$ queries to distinguish x from μ'_x.

We construct the promise P as follows. Start with $P = \emptyset$. For each $x \in \mathrm{Dom}(g^\phi)$, we add x to P if $f(x) = g^\phi(x)$; otherwise, we add the support of μ'_x to P.

It remains to lower-bound $\mathrm{R}(f|_P)$ and to upper-bound $\mathrm{Q}(f|_P)$. We start with the upper bound. Let $y \in P$, and consider the value of y on B. If x is an input of the domain of g^ϕ that caused y to be added, then x and y agree on B. Further, the values of x on B are simply $\phi^{-1}(x) \in \{0,1\}^{|B|}$, and $g(\phi^{-1}(x)) = g^\phi(x) = f(y)$. This means $g(y|_B) = f(y)$. We now have the quantum algorithm work only with the bits of $y|_B$, ignoring the rest. The algorithm need only compute $g(y|_B)$. Since $g \in S$, we get $\mathrm{Q}(f|_P) \leq \mathrm{Q}(g) \leq \mathrm{Q}(S)$, as desired. A similar argument would upper-bound other complexity measures, such as R, R_0, or D.

For the lower bound, consider the hard distribution μ on inputs to g^ϕ obtained from Yao's minimax principle [89]. This distribution has the property that any randomized algorithm for g^ϕ that succeeds with probability at least $2/3$ on inputs sampled from μ must use $\mathrm{R}(g^\phi)$ queries. We construct a new distribution μ' over P by generating an element $x \in \mathrm{Dom}(g^\phi)$ according to μ, and then outputting either x or a sample from μ'_x, depending on which of them was added to P. We lower-bound the number of queries a randomized algorithm requires to compute f on an input sampled from μ' by a reduction from either computing g^ϕ on inputs sampled from μ, or else distinguishing x from μ'_x.

Let R be a randomized algorithm for $f|_P$. Let $x \sim \mu$. We wish to compute $g^\phi(x)$. Although x may not be in P, consider running R on x anyway. The algorithm will correctly output $g^\phi(x)$ with some probability p, depending on both the internal randomness of R and on μ. If $p \geq 3/5$, we could amplify R a constant number of times to turn it into an algorithm for g that works on inputs sampled from the hard distribution μ, which means R must use $\Omega(\mathrm{R}(g^\phi)) = \Omega(\mathrm{R}^N(S))$ queries. So suppose that $p \leq 3/5$.

Next, given $x \sim \mu$, we let y_x be either x or a sample from μ'_x, as μ' dictates. Then

running R on y_x gives $f(y_x) = g^\phi(x)$ with probability at least $2/3$. On the other hand, running R on x gives output $g^\phi(x)$ with probability at most $3/5$. That is, we have

$$\Pr_{R,x\sim\mu}[R(x) = g^\phi(x)] = \mathop{\mathbb{E}}_{x\sim\mu}\left[\Pr_R[R(x) = g^\phi(x)]\right] \leq 3/5$$

$$\Pr_{R,x\sim\mu,y_x}[R(y_x) = g^\phi(x)] = \mathop{\mathbb{E}}_{x\sim\mu}\left[\Pr_{R,y_x}[R(y_x) = g^\phi(x)]\right] \geq 2/3$$

From which it follows that

$$\mathop{\mathbb{E}}_{x\sim\mu}\left[\Pr_{R,y_x}[R(y_x) = g^\phi(x)] - \Pr_R[R(x) = g^\phi(x)]\right] \geq 1/15.$$

This means there must be some specific input $\hat{x}$ such that the probability of R outputting $g^\phi(\hat{x})$ when run on $y_{\hat{x}}$ is at least $1/15$ more than the probability of R outputting $g^\phi(\hat{x})$ when run on $\hat{x}$. In particular, we must have $y_{\hat{x}} \neq \hat{x}$, so $y_{\hat{x}}$ is a sample from $\mu'_{\hat{x}}$. Therefore, R distinguishes $\hat{x}$ from $\mu'_{\hat{x}}$ with constant probability, so it uses at least $\Omega(A)$ queries.

We conclude that $\mathrm{R}(f|_P) = \Omega(\min\{\mathrm{R}^N(S), A\})$. Since the domain of the functions in S is $\{0,1\}^A$, their query complexity is at most A. Thus $\mathrm{R}(f|_P) = \Omega(\mathrm{R}^N(S))$, as desired. A similar argument lower-bounds other complexity measures, such as R_0 or D. □

This proof uses the fact that RC lower-bounds R, so it would not work on complexity measures that are not lower-bounded by RC (for example, $\mathrm{C}^{(1)}$). For Q, it might be possible to use a similar argument and suffer a quadratic loss, since Q is lower-bounded by $\sqrt{\mathrm{RC}}$. However, this might be trickier to prove (we will not need it).

We can use the previous theorems to get a sculptability result for R vs. Q in terms of the H-index of randomized certificate complexity.

Corollary 6.20. *Let $f : \{0,1\}^N \to \{0,1\}$ be a total function. Then there is a promise $P \subseteq \{0,1\}^N$ such that*

$$\mathrm{Q}(f|_P) = O(\log^2 \mathrm{H}(\mathrm{RC}_f)), \qquad \mathrm{R}(f|_P) = \Omega\left(\frac{\mathrm{H}(\mathrm{RC}_f)^{1/3}}{\log^{5/3} N}\right).$$

Proof. This follows from Theorem 6.16 together with Theorem 6.19. □

To complete the proof of Theorem 6.2, all that remains is relating $\mathrm{H}(\mathrm{RC}_f)$ to $\mathrm{H}(\mathrm{C}_f)$.

6.5 Relating $\mathrm{H}(\mathrm{C}_f)$, $\mathrm{H}(\mathrm{RC}_f)$, and $\mathrm{H}(\mathrm{bs}_f)$

In this section, we relate $\mathrm{H}(\mathrm{C}_f)$ to $\mathrm{H}(\mathrm{RC}_f)$, completing the characterization of sculpting. Actually, we will prove a relationship between $\mathrm{H}(\mathrm{C}_f)$ and $\mathrm{H}(\mathrm{bs}_f)$, which implies the desired relationship since $\mathrm{H}(\mathrm{bs}_f) \leq \mathrm{H}(\mathrm{RC}_f)$. The proof is somewhat involved, but splits naturally into three parts. In Lemma 6.21, we show a relationship between $\mathrm{C}_f(x)$ and $\mathrm{RC}_f(x)$ in terms of the number of 0- and 1-inputs of f. In Theorem 6.22, we show that $\mathrm{H}(\mathrm{C}_f) = O(\mathrm{Bal}(f)\log N)$. Finally, Theorem 6.23 gives the desired relationship between $\mathrm{H}(\mathrm{C}_f)$ and $\mathrm{H}(\mathrm{bs}_f)$.

Lemma 6.21. *Let $f : \{0,1\}^N \to \{0,1\}$ be a partial function, and let $x \in \mathrm{Dom}(f)$. If $f(x) = 0$, then*

$$\mathrm{C}_f(x) \leq \mathrm{RC}_f(x)(1 + \log |f^{-1}(1)|)$$

and if $f(x) = 1$, then

$$\mathrm{C}_f(x) \leq \mathrm{RC}_f(x)(1 + \log |f^{-1}(0)|).$$

Proof. For $x \in f^{-1}(1)$, we wish to upper-bound $\mathrm{C}_f(x)$ in terms of $\mathrm{RC}_f(x)$, assuming $f^{-1}(0)$ is small. A certificate for x consists of a partial assignment of x that contradicts all the elements of $f^{-1}(0)$.

Consider the greedy strategy for certifying x, which works by repeatedly choosing the bit of x that contradicts as many of the 0-inputs as possible, and adding it to the certificate. By definition, this strategy produces a certificate for x of size at least $\mathrm{C}_f(x)$.

Let p_i be the fraction of the remaining inputs which are contradicted by the i-th bit of the greedy algorithm. The number of remaining inputs during the run of the greedy algorithm is then

$$|f^{-1}(0)|,\ |f^{-1}(0)|(1-p_1),\ |f^{-1}(0)|(1-p_1)(1-p_2), \ldots$$

The number of remaining inputs in the greedy algorithm will be upper-bounded by a geometric sequence that starts at $|f^{-1}(0)|$ and has ratio $1 - \min_i p_i$. Such a sequence decreases to 1 after at most

$$\frac{-1}{\log(1 - \min_i p_i)}(1 + \log |f^{-1}(0)|) \leq \frac{1 + \log |f^{-1}(0)|}{\min_i p_i}$$

steps. It follows that

$$\mathrm{C}_f(x) \leq \frac{1 + \log |f^{-1}(0)|}{\min_i p_i}.$$

It remains to show that $\mathrm{RC}_f(x) = \Omega(1/\min_i p_i)$. Let j be the step of the greedy algorithm that achieves this minimum, i.e. $p_j = \min_i p_i$. Then before the j^{th} step of the algorithm, there is a non-empty set S of 0-inputs for f such that for any bit of the input, at most a p_j fraction of the elements of S disagree with x on that bit. In other words, x is entry-wise very close to the "average" of the elements of S. If we give each element of S weight $1/(p_j|S|)$, we would get a feasible set of fractional blocks with total weight $1/p_j$. Thus $\mathrm{RC}_f(x) \geq 1/p_j$, so $\mathrm{C}_f(x) \leq \mathrm{RC}_f(x)(\log |f^{-1}(0)| + 1)$. An analogous argument works when x is a 0-input to f. □

Theorem 6.22. *Let $N \geq 2$, and let $f : \{0,1\}^N \to \{0,1\}$ be a total function. Then*

$$\mathrm{H}(\mathrm{C}_f) \leq 10\,\mathrm{Bal}(f) \log N.$$

Proof. Without loss of generality, suppose $|f^{-1}(0)| \leq |f^{-1}(1)|$. The number of 0-inputs with large certificates is at most $|f^{-1}(0)| \leq 2^{\mathrm{Bal}(f)}$. Let S be the set of 1-inputs with certificates of size greater than $5\,\mathrm{Bal}(f)$. We wish to show that S is small. Lemma 6.8 implies there is a set $B = \{i_1, i_2, \ldots, i_{|B|}\}$ of indices of the input of size at least $\log |S|/(2 \log N)$ that is shattered by S. Therefore, to show that S is small, it suffices to show that B is small.

From Lemma 6.21, we have $\mathrm{C}_f(x) \leq \mathrm{RC}_f(x)\,\mathrm{Bal}(f)$ for any 1-input x, so for all $x \in S$, we have $\mathrm{RC}_f(x) \geq \mathrm{C}_f(x)/\,\mathrm{Bal}(f) > 5$. This means for all $x \in S$, there is a distribution μ_x

over 0-inputs such that for each i, the probability that $y_i \neq x_i$ when y is sampled from μ_x is less than $1/5$.

Let μ_B be the uniform distribution over B. Let $\delta(b,c) = 1$ if $b \neq c$ and 0 otherwise. We then write

$$\frac{1}{5} > \mathop{\mathbb{E}}_{i\sim\mu_B}\left(\mathop{\mathbb{E}}_{y\sim\mu_x}\delta(x_i,y_i)\right) = \mathop{\mathbb{E}}_{y\sim\mu_x}\left(\mathop{\mathbb{E}}_{i\sim\mu_B}\delta(x_i,y_i)\right).$$

We can conclude that for any $x \in S$, there exists a 0-input y_x that differs from x in less than one fifth of the bits of B. In other words, the distance between $x|_B$ and $y_x|_B$ is less than $|B|/5$. The idea is now to upper-bound $|B|$ by using the fact that for every string in $\{0,1\}^{|B|}$ there is a 0 input y such that $y|_B$ is close to that string, and there are few 0-inputs overall. Indeed, the number of strings in $\{0,1\}^{|B|}$ is $2^{|B|}$, and each 0-input can only be of distance less than $|B|/5$ from $2^{H(1/5)|B|}$ of them (where $H(1/5)$ denotes the entropy of $1/5$). Therefore, to cover all the strings in $\{0,1\}^{|B|}$, there must be more than $2^{(1-H(1/5))|B|}$ 0-inputs. Then

$$\mathrm{Bal}(f) \geq \log|f^{-1}(0)| > (1-H(1/5))|B| \geq (1-H(1/5))\frac{\log|S|}{2\log N},$$

so

$$\log|S| < \frac{2\,\mathrm{Bal}(f)\log N}{1-H(1/5)} \leq 8\,\mathrm{Bal}(f)\log N.$$

This means there are less than $2^{8\,\mathrm{Bal}(f)\log N}$ 1-inputs with certificate size at least $5\,\mathrm{Bal}(f)$. There are also at most $2^{\mathrm{Bal}(f)}$ 0-inputs with certificate size at least $5\,\mathrm{Bal}(f)$ (because there are at most that many 0-inputs in total). Thus the log of the total number of inputs with certificates larger than $5\,\mathrm{Bal}(f)$ is at most $10\,\mathrm{Bal}(f)\log N$. It follows that $\mathrm{H}(\mathrm{C}_f) \leq 10\,\mathrm{Bal}(f)\log N$. □

Theorem 6.23. *Let $f : \{0,1\}^N \to \{0,1\}$ be a total function. Then*

$$\mathrm{H}(\mathrm{C}_f) = O(\mathrm{H}(\mathrm{bs}_f)^2 \log N).$$

Proof. Let A be the set of inputs that have certificate size more than $\mathrm{H}(\mathrm{C}_f)$. Let A_0 be the set of 0-inputs in A, and let A_1 be the set of 1-inputs in A. Let B be the set of inputs that have block sensitivity more than b, with $b = \sqrt{\mathrm{H}(\mathrm{C}_f)/2}$. Let B_0 be the set of 0-inputs in B, and let B_1 be the set of 1-inputs in B. Without loss of generality, assume $|A_0| \geq |A_1|$. Since $|A| \geq 2^{\mathrm{H}(\mathrm{C}_f)}$, we have $|A_0| \geq 2^{\mathrm{H}(\mathrm{C}_f)-1}$.

Now, let $g : \{0,1\}^N \to \{0,1\}$ be the total function defined by $g(x) = 1$ if and only if $x \in B_1$. Suppose x is an element of $A_0 \backslash B_0$. Consider certifying that x is a 0-input for g; let p be the smallest such certificate. Then p is consistent with x but inconsistent with all the strings in B_1. We claim that this certificate must be large: its size must be greater than $\mathrm{H}(\mathrm{C}_f) - b^2 = \mathrm{H}(\mathrm{C}_f)/2$. To show this, we show that we can turn p into a certificate for x with respect to f (instead of with respect to g) by adding only b^2 bits to it.

Let q be a minimal sensitive block of x (with respect to f) that is disjoint from p. Since x is a 0-input for f, x^q is a 1-input for f. Since q is disjoint from p, x^q is consistent with p, so $x^q \notin B_1$. Thus the block sensitivity of x^q is at most b. However, since q was a minimal sensitive block, the sensitivity of x^q is at least $|q|$; thus $|q| \leq b$. It follows that all minimal sensitive blocks of x that are disjoint from p must have size at most b.

In addition, since $x \in A_0 \backslash B_0$, the block sensitivity of x is at most b. We can now construct a certificate for x by taking a maximal set of minimal disjoint sensitive blocks for

x, all of which are disjoint from p. There will be at most b such blocks, and each will have size at most b. Therefore, this certificate for x has size at most $|p| + b^2$. Since $x \in A_0$, we must have $|p| + b^2 > \mathrm{H}(\mathrm{C}_f)$, or $|p| > \mathrm{H}(\mathrm{C}_f) - b^2 = \mathrm{H}(\mathrm{C}_f)/2$. We have shown that the elements of $A_0 \backslash B_0$ all have certificate size greater than $\mathrm{H}(\mathrm{C}_f)/2$ even with respect to g.

Now, by Theorem 6.22, the number of inputs x that have certificate size more than $10(1 + \log |B_1|) \log N$ with respect to g is at most $2^{10(1+\log |B_1|) \log N}$. It follows that either $\mathrm{H}(\mathrm{C}_f)/2 \leq 10(1 + \log |B_1|) \log N$ (so that the theorem doesn't apply), or else $|A_0 \backslash B_0| \leq 2^{10(1+\log |B_1|) \log N}$.

In the former case, we have

$$\log |B| \geq \log |B_1| \geq \frac{\mathrm{H}(\mathrm{C}_f)}{20 \log N} - 1.$$

In the latter case, we have

$$2^{\mathrm{H}(\mathrm{C}_f)-1} \leq |A_0| \leq |B_0| + 2^{10(1+\log |B_1|) \log N} = |B_0| + (2|B_1|)^{10 \log N} \leq (2|B|)^{10 \log N},$$

so in that case,

$$\log |B| \geq \frac{\mathrm{H}(\mathrm{C}_f) - 1}{10 \log N} - 1.$$

Thus, in both cases,

$$\log |B| \geq \frac{\mathrm{H}(\mathrm{C}_f) - 1}{20 \log N} - 1 = \Omega\left(\frac{\mathrm{H}(\mathrm{C}_f)}{\log N}\right).$$

Hence there are $2^{\Omega(\mathrm{H}(\mathrm{C}_f)/\log N)}$ inputs with block sensitivity more than $\sqrt{\mathrm{H}(\mathrm{C}_f)/2}$. We thus have

$$\mathrm{H}(\mathrm{bs}_f) = \Omega\left(\min\left\{\frac{\mathrm{H}(\mathrm{C}_f)}{\log N}, \sqrt{\mathrm{H}(\mathrm{C}_f)}\right\}\right) = \Omega\left(\sqrt{\frac{\mathrm{H}(\mathrm{C}_f)}{\log N}}\right). \qquad \square$$

Theorem 6.2 now follows from Theorem 6.10 (the non-sculptability theorem in terms of $\mathrm{H}(\mathrm{C}_f^2)$), Corollary 6.20 (the sculptability result in terms of $\mathrm{H}(\mathrm{RC}_f)$), and Theorem 6.23 (relating $\mathrm{H}(\mathrm{bs}_f)$ to $\mathrm{H}(\mathrm{C}_f)$), together with the properties that $\mathrm{H}(\mathrm{C}_f^2) \leq \mathrm{H}(\mathrm{C}_f)^2$ and that $\mathrm{H}(\mathrm{bs}_f) \leq \mathrm{H}(\mathrm{RC}_f)$. We restate Theorem 6.2 here for convenience.

Theorem 6.2. *For all total functions $f : \{0,1\}^N \to \{0,1\}$ and all promises $P \subseteq \{0,1\}^N$, we have*

$$\mathrm{R}(f|_P) = O(\mathrm{Q}(f|_P)^2 \mathrm{H}(\mathrm{C}_f)^2).$$

Conversely, for all total functions $f : \{0,1\}^N \to \{0,1\}$, there is a promise $P \subseteq \{0,1\}^N$ such that

$$\mathrm{R}(f|_P) = \Omega\left(\frac{\mathrm{H}(\mathrm{C}_f)^{1/6}}{\log^{11/6} N}\right) \qquad \text{and} \qquad \mathrm{Q}(f|_P) = O(\log^2 \mathrm{H}(\mathrm{C}_f)).$$

Theorem 6.1 follows as a corollary. This completes the proof of our main result.

6.6 Sculpting Randomized Speedups

Now that we've characterized sculpting quantum query complexity, we turn our attention to sculpting other measures. Recall that

$$\mathrm{Q}(f) \leq \mathrm{R}(f) \leq \mathrm{R}_0(f) \leq \mathrm{D}(f).$$

We showed that sculpting $\mathrm{R}(f)$ vs. $\mathrm{Q}(f)$ is possible if and only if f has a large number of large certificates. We now show that the exact same condition characterizes sculpting $\mathrm{D}(f)$ vs. $\mathrm{R}_0(f)$. On the other hand, we show that $\mathrm{R}_0(f)$ vs. $\mathrm{R}(f)$ behaves differently: it's *always* possible to sculpt a function f to a promise P such that $\mathrm{R}(f|_P)$ is constant and $\mathrm{R}_0(f|_P)$ is almost as large as $\mathrm{R}_0(f)$.

We start by characterizing sculpting for D vs. R_0.

6.6.1 Sculpting D vs. R_0

The proof of this characterization will follow that of Theorem 6.2. For the non-sculptability direction, we need an analogue of Theorem 6.10, relating deterministic and zero-error randomized query complexities in terms of $\mathrm{Bal}(f)$. We prove the following theorem.

Theorem 6.24. *Let $f : \{0,1\}^N \to \{0,1\}$ be a partial function. Then*

$$D(f) \leq 2R_0(f)\,\mathrm{Bal}(f).$$

Proof. Consider the zero-error randomized algorithm which takes $\mathrm{R}_0(f)$ expected queries to evaluate f. By Markov's inequality, if we let this algorithm make $2\,\mathrm{R}_0(f)$ queries on input x, it will succeed in computing $f(x)$ (and provide a certificate for x) with probability at least $1/2$. This gives us a probability distribution μ over deterministic algorithms, each of which makes $2\,\mathrm{R}_0(f)$ queries, such that for each input x the probability that an algorithm sampled from μ finds a certificate when run on x is at least $1/2$.

For a deterministic algorithm D and an input x, let $c(D,x) = 1$ if D finds a certificate for x, and $c(D,x) = 0$ otherwise. Let $Z \subseteq \{0,1\}^N$. Then

$$\mathop{\mathbb{E}}_{D\sim\mu}\left[\sum_{x\in Z} c(D,x)\right] = \sum_{x\in Z} \mathop{\mathbb{E}}_{D\sim\mu}[c(D,x)] \geq \sum_{x\in Z}(1/2) = \frac{|Z|}{2}.$$

It follows that there is a deterministic algorithm D_Z that makes $2\,\mathrm{R}_0(f)$ queries and finds a certificate when run on half the inputs in Z.

Suppose without loss of generality that $|f^{-1}(0)| \leq |f^{-1}(1)|$. Now, on input x, set $Z = f^{-1}(0)$, and run D_Z on x. If it fails to find a certificate, then we have eliminated half of Z as possibilities for the input. Repeating this $\lfloor \log |f^{-1}(0)| \rfloor + 1 \leq \mathrm{Bal}(f)$ times suffices to eliminate all of $f^{-1}(0)$ as possibilities for x, and hence to determine the value of $f(x)$. The total number of queries used is at most $2\,\mathrm{R}_0(f)\,\mathrm{Bal}(f)$. □

Note that Theorem 6.24 and Theorem 6.10 together complete the proof of Theorem 6.4.

Next, we turn Theorem 6.24 into a non-sculptability theorem in terms of $\mathrm{H}(\mathrm{C}_f)$. The argument in Theorem 6.12 follows verbatim, and we get the following sculpting lower bound.

Corollary 6.25. *Let $f : \{0,1\}^N \to \{0,1\}$ be a total function. For any promise $P \subseteq \{0,1\}^N$, we have*

$$\mathrm{D}(f|_P) = O(\mathrm{R}_0(f|_P)\,\mathrm{H}(\mathrm{C}_f)^2).$$

We now prove the other direction: we show that sculpting is possible when $\mathrm{H}(\mathrm{RC}_f)$ is large. Using the arguments from Section 6.4, it suffices to solve the extended queries problem for D vs. R_0. We do this using the reduction to communication complexity in Theorem 6.18.

Theorem 6.26. *For all $N \in \mathbb{N}$, there is a set of partial functions S from $\{0,1\}^N$ to $\{0,1\}$ such that for all $G \geq N$,*

$$\mathrm{R}_0(S) = O(1), \qquad \mathrm{D}^G(S) = \Omega\left(\frac{N}{\log G}\right).$$

Proof. We start with EQUALITY, in which Alice and Bob are each given an n-bit string and wish to know if their strings are equal. This problem has deterministic query complexity $\Omega(n)$, but small randomized query complexity. To make the zero-error randomized query complexity small as well, we give Alice and Bob two strings each, with the promise that either their first strings are equal and the second strings are not, or vice versa. The goal will be to determine which is the case. It is not hard to see that the deterministic communication complexity of this problem is still $\Omega(n)$.

We need to get the zero-error randomized query complexity of the marginal functions to be small. To do this, we introduce another modification: we encode each of Bob's strings using a fixed random code of length $3n$. This code will have the property that the distance between any pair of codewords is $\Omega(n)$. To compute a marginal function f_{a_1,a_2} indexed by Alice's strings, we can simply randomly sample from each of Bob's strings; after $O(1)$ samples, we will discover which of his strings do not match the codeword corresponding to a_1 and a_2.

This construction gives us a function $f : \{0,1\}^{2n} \times \{0,1\}^{6n} \to \{0,1\}$ such that $\mathrm{D}^{\mathrm{CC}}(f) = \Omega(n)$ and $\mathrm{R}_0(\mathrm{Mar}(f)) = O(1)$. Setting $N = 6n$ and using Theorem 6.18 finishes the proof. □

Putting this together, we get the following sculpting theorem which, together with Corollary 6.25, is analogous to Theorem 6.2.

Theorem 6.27. *For all total functions $f : \{0,1\}^N \to \{0,1\}$, there is a promise $P \subseteq \{0,1\}^N$ such that*

$$\mathrm{D}(f|_P) = \Omega\left(\frac{\sqrt{\mathrm{H}(\mathrm{C}_f)}}{\log^{5/2} N}\right) \qquad \text{and} \qquad \mathrm{R}_0(f|_P) = O(1).$$

Proof. This follows from Theorem 6.26, Theorem 6.16 and Theorem 6.23. □

We also get the following corollary, analogous to Theorem 6.1.

Corollary 6.28. *Let $f : \{0,1\}^N \to \{0,1\}$ be a total function. Then there is a promise $P \subseteq \{0,1\}^N$ such that $\mathrm{D}(f|_P) = N^{\Omega(1)}$ and $\mathrm{R}_0(f|_P) = N^{o(1)}$, if and only if $\mathrm{H}(\mathrm{C}_f) = N^{\Omega(1)}$. Futhermore, in this case we also have $\mathrm{R}_0(f|_P) = O(1)$.*

6.6.2 Sculpting R_0 vs. R

While it is possible to use the above argument to get a sculptability result for R_0 vs. R, we can get a stronger result by a direct argument. In fact, unlike R vs. Q or D vs. R_0, sculpting R_0 vs. R is *always* possible (there is no dependence on any H-index).

Theorem 6.29. *Let $f : \{0,1\}^N \to \{0,1\}$ be a non-constant total function. Then there is a promise $P \subseteq \{0,1\}^N$ such that*

$$\mathrm{R}(f|_P) = 1, \qquad \mathrm{R}_0(f|_P) \geq \frac{\mathrm{R}_0(f)^{1/3}}{6}.$$

Proof. We actually prove a stronger result, finding a promise P such that $\mathrm{R}(f|_P) = 1$ and $\mathrm{R}_0(f|_P) \geq \mathrm{bs}(f)/6$. We then use the known relationship $\mathrm{R}_0(f) \leq \mathrm{bs}(f)^3$ for total functions to get the desired result. Note that finding P with $\mathrm{R}(f|_P) = 1$ and $\mathrm{R}_0(f|_P) \geq \mathrm{bs}(f)/6$ is trivial when $\mathrm{bs}(f) \leq 6$; thus we assume $\mathrm{bs}(f) > 6$.

Let $x \in \{0,1\}^N$ be such that $\mathrm{bs}_f(x) = \mathrm{bs}(f)$. Assume without loss of generality that $f(x) = 0$. Let S_1 be the set of all 1-inputs with Hamming distance at least $(2/3)N$ from x. For any partial assignment p consistent with x, let $S_1^p \subseteq S_1$ be the set of all inputs y in S_1 that are consistent with p.

There are two cases. If S_1^p is non-empty for all partial assignments p consistent with x of size less than $\mathrm{bs}_f(x)/6$, then we can pick the promise to be $P = \{x\} \cup S_1$. It then follows that certifying that $f|_P$ is 0 on input x takes at least $\mathrm{bs}_f(x)/6$ queries, whence $\mathrm{R}_0(f|_P) \geq \mathrm{bs}_f(x)/6$. On the other hand, a randomized algorithm can make 1 query to check if the input differs from x. Thus $\mathrm{R}(f|_P) = 1$.

The other case is that there is some partial assignment p of size less than $\mathrm{bs}_f(x)/6$ such that S_1^p is empty. We restrict our attention to inputs consistent with p. Since x has $\mathrm{bs}_f(x)$ disjoint sensitive blocks, it has at least $(5/6)\,\mathrm{bs}_f(x)$ disjoint sensitive blocks that do not overlap with p. We exclude blocks of size larger than $N/3$. Since there are at most 2 such blocks, this leaves at least $(5/6)\,\mathrm{bs}_f(x) - 2$. Let B be the set of inputs we get by flipping one of these blocks of x. Then B contains only 1-inputs to f that are consistent with p, all of which have Hamming distance at most $N/3$ from x. Since $\mathrm{bs}_f(x) = \mathrm{bs}(f) > 6$, we have $B \neq \emptyset$.

Let S be the set of inputs consistent with p that have Hamming distance at least $(2/3)N$ from x. Since S_1^p is empty, S contains only 0-inputs to f. Let $P = B \cup S$. Now, consider certifying that an input y to $f|_P$ is a 1-input. Since all inputs of Hamming distance at least $(2/3)N$ from x that are consistent with p are 0-inputs, this requires showing at least $N/3 - |p|$ bits of y. Since $|p| < \mathrm{bs}_f(x)/6 \leq N/6$, this is at least $N/6$. Thus $\mathrm{R}_0(f|_P) \geq N/6 \geq \mathrm{bs}(f)/6$.

On the other hand, a bounded-error randomized algorithm can simply query a bit of the input at random, and check for agreement with x. If the bit agrees, the algorithm can output 1, and if the bit disagrees, the algorithm can output 0. This works because 0-inputs have distance at least $(2/3)N$ from x, while all 1-inputs have distance at most $N/3$ from x (since the sensitive blocks used to construct B were of size at most $N/3$). Thus $\mathrm{R}(f|_P) = 1$. □

6.7 Other Query Complexity Results

We can use some of the tools introduced in the previous sections to prove some new relations in query complexity. In Section 6.7.1, we prove a quadratic relationship between $\mathrm{D}(f)$ and $\mathrm{Q}(f)$ for partial functions f that have small domain. In Section 6.7.2, we prove a quadratic relationship between $\mathrm{D}(f)$ and $\mathrm{Q}(f)$ for total functions f for which $\mathrm{H}(\mathrm{C}_f)$ is small.

6.7.1 Query Complexity on Small Promises

We prove Theorem 6.3, which we restate for convenience.

Theorem 6.3. *Let $f : \{0,1\}^N \to \{0,1\}$ be a partial function, and let $\mathrm{Dom}(f)$ denote the domain of f. Then*

$$\mathrm{Q}(f) = \Omega\left(\frac{\sqrt{\mathrm{D}(f)}}{\log|\mathrm{Dom}(f)|}\right).$$

Proof. We follow the proof of Theorem 6.10. The randomized algorithm used in that proof relies only on the existence of a randomized algorithm distinguishing a string $a \in \{0,1\}^N$ from either $f^{-1}(0)$ or $f^{-1}(1)$, which is in turn guaranteed by Lemma 6.9. To make that algorithm deterministic, we only need to turn this distinguishing algorithm into a deterministic one. By Lemma 6.21, we have $\mathrm{C}_f(x) = O(\mathrm{RC}_f(x)\log|\mathrm{Dom}(f)|)$. On the task of distinguishing a single input from a set of inputs, certificate complexity equals deterministic query complexity. Using this observation, we can modify the proof of Theorem 6.10 to get the result

$$\mathrm{D}(f) = O(\mathrm{Q}(f)^2\,\mathrm{Bal}(f)\log|\mathrm{Dom}(f)|) = O(\mathrm{Q}(f)^2\log^2|\mathrm{Dom}(f)|),$$

from which the desired result follows. □

6.7.2 Relationship for Total Functions

We can use H-indices to improve some of the relationships between complexity measures on total functions, proving Theorem 6.5. Recall that for total functions, we have $\mathrm{D}(f) \leq \mathrm{C}(f)\,\mathrm{bs}(f)$ and $\mathrm{bs}(f) = O(\mathrm{Q}(f)^2)$, from which we have $\mathrm{D}(f) = O(\mathrm{Q}(f)^2\,\mathrm{C}(f))$. We strengthen this result to $\mathrm{D}(f) = O(\mathrm{Q}(f)^2\,\mathrm{H}(\sqrt{\mathrm{C}_f})^2)$ for total Boolean functions. Since $\mathrm{H}(\sqrt{\mathrm{C}_f}) \leq \sqrt{\mathrm{C}(f)}$, this result is always stronger. In addition, since $\mathrm{C}(\mathrm{OR}) = n$ and $\mathrm{H}(\mathrm{C}_{\mathrm{OR}}) = 1$, this improvement is sometimes very strong.

We restate Theorem 6.5 for convenience.

Theorem 6.5. *Let $f : \{0,1\}^N \to \{0,1\}$ be a total function. Then*

$$\mathrm{D}(f) = O(\mathrm{Q}(f)^2\,\mathrm{H}(\sqrt{\mathrm{C}_f})^2).$$

Proof. The proof follows the proof that $\mathrm{D}(f) \leq \mathrm{C}(f)\,\mathrm{bs}(f)$ [17]. We start by reviewing this proof. The deterministic algorithm repeatedly picks possible 0-certificates that are consistent with the input observed so far, and queries the entries of these certificates. If the queried entries match the 0-certificate, the algorithm is done (the value of $f(x)$ is known to be 0). If ever there are no additional 0-certificates consistent with the observed part of the input, the value of the function is known to be 1.

The key insight is that if this process repeats k times, then the block sensitivity of the function is at least k. Indeed, let p be the partial assignment revealed after k iterations. Pick a 1-input y for f that is consistent with p. Let B_i be the set of entries queried in the i-th iteration of the algorithm. Then for each i, there is a way to change only the variables in B_i to form a 0-certificate for f. It follows that each B_i contains a sensitive block for y. Since the B_i sets are disjoint, we get $\mathrm{bs}_f(y) \geq k$, so $\mathrm{bs}(f) \geq k$.

We modify the algorithm as follows. In each step, we only allow the algorithm to pick 0-certificates that are of size at most $\mathrm{H}(\sqrt{\mathrm{C}_f})^2$. Thus the algorithm uses at most $\mathrm{bs}(f)\,\mathrm{H}(\sqrt{\mathrm{C}_f})^2$ queries before it gets stuck. When it gets stuck, either the value of f on the input is determined, or else there are no more 0-certificates that are small enough.

Next, we repeat the same process with 1-certificates instead of 0-certificates. If the value of f is not yet determined, it means that the input is not consistent with any small enough

certificates, so the certificate complexity of the input x is greater than $\mathrm{H}(\sqrt{\mathrm{C}_f})^2$. This gives $\sqrt{\mathrm{C}_f(x)} > \mathrm{H}(\sqrt{\mathrm{C}_f})$.

By definition of the H-index, there are now at most $2^{\mathrm{H}(\sqrt{\mathrm{C}_f})}$ possibilities for the input. We've therefore restricted f to a small domain P. We now use Theorem 6.3 to evaluate f using

$$O(\mathrm{Q}(f)^2 \log^2 |\mathrm{Dom}(f|_P)|) = O(\mathrm{Q}(f)^2 \mathrm{H}(\sqrt{\mathrm{C}_f})^2)$$

deterministic queries. This is added to the $\mathrm{bs}(f)\,\mathrm{H}(\sqrt{\mathrm{C}_f})^2$ queries from before. Using $\mathrm{bs}(f) = O(\mathrm{Q}(f)^2)$, we get

$$\mathrm{D}(f) = O(\mathrm{Q}(f)^2 \mathrm{H}(\sqrt{\mathrm{C}_f})^2).$$

□

6.8 Sculpting Computational Complexity

In this section, we examine sculpting in the computational complexity model. We start with some notation. Given a language $L \subseteq \{0,1\}^*$, we let $L(x) \in \{0,1\}$ be its characteristic function. Also, given a language L together with a promise $P \subseteq \{0,1\}^*$, we let $L|_P$ be the promise problem of distinguishing the set $P \cap L$ from the set $P \setminus L$.

Now, we call the language L *sculptable* if there exists a promise P, such that the promise problem $L|_P$ is in PromiseBQP but not in PromiseBPP. We will use the following definition.

Definition 6.30 ([22]). *A language L is called* paddable, *if there exists a polynomial-time function $f(x,y)$ such that*

(1) f is polynomial-time invertible, and

(2) for all x, y, we have $x \in L \iff f(x,y) \in L$.

In other words, L is paddable if it is possible to "pad out"any input x with irrelevant information y, in an invertible way, without affecting membership in L.

The paddable languages were introduced by Berman and Hartmanis [22], as part of their exploration of whether all NP-complete languages are polynomial-time isomorphic: they showed that the answer was 'yes' for all *paddable* NP-complete languages. Under strong cryptographic assumptions, we now know that there exist NP-complete languages that are neither paddable nor isomorphic to each other [55]. Nevertheless, it remains the case that almost all the languages that "naturally arise in complexity theory" are paddable.

Next, let us say that PromiseBQP *is hard on average for* P/poly if there exists a promise problem $H|_S \in$ PromiseBQP, as well as a family of distributions $\{\mathcal{D}_n\}_n$ with support on the promise set S, such that

(1) $\mathcal{D}_n$ is samplable in classical poly(n) time, and

(2) there is no family of classical circuits $\{C_n\}_n$, of size poly(n), such that for all n,

$$\Pr_{y \sim \mathcal{D}_n} [C_n(y) = H(y)] \geq \frac{3}{4}.$$

So for example, because of Shor's algorithm [78], combined with the worst-case/average-case equivalence of the discrete log problem, we can say that *if discrete log is not in* P/poly, *then* PromiseBQP *is hard on average for* P/poly.

We now prove Theorem 6.6, which we restate here for convenience.

Theorem 6.6. *Assume* PromiseBQP *is hard on average for* P/poly. *Then every paddable language outside of* BPP *is sculptable.*

Proof. Let L be a paddable language, and let f be the padding function for L. Also, let $H|_S$ be any problem in PromiseBQP that is hard on average for P/poly, and let $\{\mathcal{D}_n\}_n$ be the associated family of hard distributions. Then we need to construct a promise, $P \subseteq \{0,1\}^*$, such that the promise problem $L|_P$ is in PromiseBQP but not in PromiseBPP.

Our promise P will simply consist of all inputs of the form $f(x,y,a)$ such that $y \in S$ and

$$L(x) \equiv H(y) + a \,(\mathrm{mod}\, 2).$$

Here $a \in \{0,1\}$ is a single bit, which we think of as concatenated onto the end of y.

Clearly, $L|_P$ is in PromiseBQP: just invert f to extract the "comment"(y,a), then compute $H(y) + a \,(\mathrm{mod}\, 2)$.

We need to show that $L|_P$ is not in PromiseBPP. Suppose by contradiction that it was, and let $\mathcal{A}$ be the algorithm such that $\mathcal{A}(x) = L(x)$ for all $x \in P$. Then we'll show how to either

(1) decide L in BPP (with no promise), or

(2) decide H in P/poly, with high probability over $\mathcal{D}_n$.

Given an arbitrary input $x \in \{0,1\}^n$, imagine we do the following: first sample $y \sim \mathcal{D}_n$, then run $\mathcal{A}$ on the inputs $f(x,y,0)$ and $f(x,y,1)$. There are two cases: first suppose

$$\mathcal{A}\left(f(x,y,0)\right) = \mathcal{A}\left(f(x,y,1)\right).$$

Now, *one* of the two inputs $f(x,y,0)$ and $f(x,y,1)$ must belong to P. If $f(x,y,0) \in P$, then $\mathcal{A}\left(f(x,y,0)\right) = L(x)$, while if $f(x,y,1) \in P$, then $\mathcal{A}\left(f(x,y,1)\right) = L(x)$. Either way, then, we have learned whether $x \in L$, and we know we have learned this.

Second, suppose

$$\mathcal{A}\left(f(x,y,0)\right) \neq \mathcal{A}\left(f(x,y,1)\right).$$

Then assuming $y \in S$:

$$\begin{aligned}
x \in L, y \in H &\Longrightarrow \mathcal{A}\left(f(x,y,0)\right) = 1 \Longrightarrow \mathcal{A}\left(f(x,y,1)\right) = 0,\\
x \in L, y \notin H &\Longrightarrow \mathcal{A}\left(f(x,y,1)\right) = 1 \Longrightarrow \mathcal{A}\left(f(x,y,0)\right) = 0,\\
x \notin L, y \in H &\Longrightarrow \mathcal{A}\left(f(x,y,1)\right) = 0 \Longrightarrow \mathcal{A}\left(f(x,y,0)\right) = 1,\\
x \notin L, y \notin H &\Longrightarrow \mathcal{A}\left(f(x,y,0)\right) = 0 \Longrightarrow \mathcal{A}\left(f(x,y,1)\right) = 1.
\end{aligned}$$

Thus, regardless of whether $x \in L$, we have learned whether $y \in H$, and again we know we have learned this.

Now suppose there were an input $x \in \{0,1\}^n$, such that running $\mathcal{A}$ as above told us whether $y \in H$ with probability more than (say) $1/2$ over the choice of $y \sim \mathcal{D}_n$. Then let C_n be a polynomial-size circuit that hardwires x, and that given an input $y \in S$:

- Simulates both $\mathcal{A}\left(f(x,y,0)\right)$ and $\mathcal{A}\left(f(x,y,1)\right)$.
- Outputs $H(y)$ whenever it successfully learns the value of $H(y)$.

- Guesses a hardwired value for $H(y)$ (whichever of $\{0,1\}$ is more probable) whenever it does not.

Then

$$\Pr_{y \sim \mathcal{D}_n} [C_n(y) = H(y)] \geq \frac{3}{4},$$

violating the assumption that no such circuit exists.

So we conclude that for every $x \in \{0,1\}^n$, we must instead learn whether $x \in L$ with probability at least $1/2$ over the choice of $y \sim \mathcal{D}_n$. This, in turn, means that by simply generating y's randomly until we succeed, we can decide L in PromiseBPP. □

Next, given a language $H \subseteq \{0,1\}^*$, we say H is BPP-*bi-immune* if neither H nor its complement $\overline{H}$ has any infinite subset in BPP. The notion of immunity was introduced by [35]. Here is a useful alternative characterization:

Lemma 6.31. *A language H is* BPP*-bi-immune if and only if there is no infinite set $S \in$* BPP*, such that the promise problem $H|_S$ is solvable in* PromiseBPP.

Proof. First, suppose H is not BPP-bi-immune, so that either H or $\overline{H}$ has an infinite subset $S \in$ BPP. Then clearly, S itself is an infinite set in BPP such that the promise problem $H|_S$ is trivial (the answer is either always 0 or always 1).

Conversely, suppose there exists an infinite set $S \in$ BPP such that $H|_S$ is solvable in polynomial time. Then clearly $S \cap H$ and $S \cap \overline{H}$ are both in BPP, and at least one of the two must be infinite. So H is not BPP-bi-immune. □

We now suggest what, as far as we know, is a new conjecture in quantum complexity theory.

Conjecture 6.32. *There exists a* BPP*-bi-immune language in* BQP.

Conjecture 6.32 is extremely strong. Note, in particular, that none of the "standard"BQP languages, such as languages based on factoring or discrete log, will be BPP-bi-immune, because they all have infinite special cases that are classically recognizable and easy (for example, the powers of 2, in the case of factoring). Nevertheless, we believe Conjecture 6.32 is plausible. As a concrete candidate for a BPP-bi-immune language in BQP, let $g : \{0,1\}^* \rightarrow \{0,1\}^*$ be some strong pseudorandom generator. Then consider the language

$$L = \{x : g(x), \text{ as a positive integer, has an odd number of distinct prime factors}\}.$$

We now prove Theorem 6.7, restated here for convenience.

Theorem 6.7. *Assume Conjecture 6.32. Then every language outside of* BPP *is sculptable.*

Proof. Assume by way of contradiction that $L \notin$ BPP is non-sculptable. Also, let H be a BPP-bi-immune language in BQP. Then consider the set

$$S := \{x : L(x) = H(x)\}.$$

By our assumption, S is a promise on which no superpolynomial quantum speedup is possible for L, and $\overline{S}$ is another such promise. Hence, there must be a BPP algorithm, call it $\mathcal{A}_S$, that solves the promise problem $H|_S$, which (by the definition of S) is equivalent to solving

$L|_S$. And there must be another polynomial-time classical algorithm, call it $\mathcal{A}_{\overline{S}}$, that solves $H|_{\overline{S}}$, which (again by the definition of S) is equivalent to solving $\overline{L}|_{\overline{S}}$.

Now, given an input x, suppose we run both $\mathcal{A}_S$ and $\mathcal{A}_{\overline{S}}$. Then as in the proof of Theorem 6.6, there are two possibilities. If $\mathcal{A}_S(x) = \mathcal{A}_{\overline{S}}(x)$, then $x \in S$ implies $H(x) = \mathcal{A}_S(x)$ while $x \notin S$ implies $H(x) = \mathcal{A}_{\overline{S}}(x)$, so either way we have learned $H(x)$ (and we know that we have learned it). On the other hand, if $\mathcal{A}_S(x) \neq \mathcal{A}_{\overline{S}}(x)$, then $x \in S$ implies $L(x) = \mathcal{A}_S(x)$ while $x \notin S$ implies $L(x) = 1 - \mathcal{A}_{\overline{S}}(x)$. So, merely by seeing that $\mathcal{A}_S(x)$ and $\mathcal{A}_{\overline{S}}(x)$ are different, we have learned $L(x)$ (and we know that we have learned it).

In summary, there is a BPP algorithm $\mathcal{B}$ that, for every input $x \in \{0,1\}^*$, correctly outputs either $H(x)$ or $L(x)$, and that moreover tells us which one it output.

Now let Q be the set of all x such that $\mathcal{B}(x)$ outputs $H(x)$. Then there are two possibilities: if Q is finite, then $\mathcal{B}$ decides L on all but finitely many inputs. Hence $L \in$ BPP, contrary to assumption. If, on the other hand, Q is infinite, then $H|_Q$ is an infinite promise problem in PromiseBPP. So H was not BPP-bi-immune, again contrary to assumption. □

In Theorem 6.6 and Theorem 6.7, there is almost nothing specific to the complexity classes BQP and BPP, apart from some simple closure properties. Thus, one can prove analogous sculpting theorems for many other pairs of complexity classes. In some cases, we do not even need an unproved conjecture. For example, we have:

Theorem 6.33. *For every language $L \notin$ P, there exists a promise S such that $L|_S$ is solvable in exponential time, but is not solvable in polynomial time.*

Proof. The proof of Theorem 6.7 follows through for P and EXP instead of BPP and BQP. In addition, it is known that there is a P-bi-immune language in EXP [22]. The desired result follows. □

6.9 Concluding Remarks and Open Problems

In this work, we gave a full characterization of the class of Boolean functions f that can be sculpted into a promise problem with an exponential quantum speedup in query complexity. We similarly characterized sculptability for R_0 vs. R and D vs. R_0. Along the way, we showed that Q is polynomially related (indeed, *quadratically* related) to D and R for a much wider set of promise problems than was previously known. Finally, we studied sculpting in *computational* complexity, giving a strong conjecture under which every language outside BPP is sculptable into a superpolynomial quantum speedup, and a weaker conjecture under which every *paddable* language outside BPP is sculptable.

One might object that many of our sculpted promise problems are somewhat artificial. This is particularly clear in the case of paddable languages, where (in essence) one uses the paddability to append to each instance x, as a "comment," an instance of a hard BQP problem (such as factoring) that is promised to have the same answer as x. Even in the query complexity setting, however, one can observe by direct analogy that *the property of being sculptable is not closed under the removal of dummy variables.* So for example, we saw before that the N-bit OR function is not sculptable. By contrast, observe that the function

$$f(x_1, \ldots, x_{2N}) := \mathrm{OR}(x_1, \ldots, x_N)$$

is sculptable. This follows as an immediate consequence of Theorem 6.1: just by adding dummy variables to the OR function, we have vastly increased the number of inputs x that

have large certificate complexity, from 1 to 2^N. However, an even simpler way to see why f is sculptable, is that we can embed (say) Simon's problem into the variables $x_{N+1}, \ldots, x_{2N}$, and then impose the promise that

$$\mathrm{OR}(x_1, \ldots, x_N) = \textsc{Simon}(x_{N+1}, \ldots, x_{2N})$$

(in addition to the Simon promise itself).

Of course, most Boolean functions do *not* contain such dummy variables, so the problems of sculpting them, and deciding whether they are sculptable at all, are much more complicated, as we saw in this chapter.

Now, it might feel like "cheating" to sculpt a promise problem with a large quantum/classical gap by using dummy variables to encode a different, unrelated problem. If so, however, that points to an interesting direction for future research: namely, can we somehow formalize what we mean by a "natural" special case of a problem, and can we then understand which problems are "naturally" sculptable?

Here are some more specific open problems.

- Some of our inequalities could be off by polynomial factors; it would be nice to tighten them (or prove separations). For example, it may be possible to improve Theorem 6.3 to $\mathrm{Q}(f) = \Omega(\sqrt{\mathrm{D}(f)/\log|\mathrm{Dom}(f)|})$, quadratically improving the $\log|\mathrm{Dom}(f)|$ factor.
- Can our results – and specifically, Theorem 6.5 – be used to improve the relation $\mathrm{D}(f) = O(\mathrm{Q}(f)^6)$ due to Beals et al. [17]?
- Can we give a characterization of the sculptable Boolean functions in *communication* complexity – analogous to this chapter's characterization of sculptability in query complexity?
- Is there any natural pair of complexity classes $\mathcal{C} \subseteq \mathcal{D}$, for which $\mathcal{C}$ is known or believed to be strictly contained in $\mathcal{D}$, and yet it is plausible that no languages in $\mathcal{D}$ are $\mathcal{C}$-bi-immune, and (related to that) there exist languages $L \notin \mathcal{C}$ that *cannot* be sculpted into a promise problem in $\mathcal{D} \setminus \mathcal{C}$?
- One can, of course, consider sculpting for many other pairs of computational models, besides R vs. Q or R_0 vs. R or D vs. R_0. One interesting case is sculpting versus certificate complexity – for example, D vs. C. What is the correct characterization there?

We make some observations on the last problem. It's easy to see that $\mathrm{D}(\mathrm{OR}|_P) = \mathrm{C}(\mathrm{OR}|_P)$ for any promise P, so sculpting D vs. C is not always possible. On the other hand, sculpting D vs. C is sometimes possible even when $\mathrm{H}(\mathrm{C}_f)$ is small. To see this, consider the function f with $f(x) = 1$ if and only if the Hamming weight of x is 1, and the single '1' bit occurs on the left half of the input string. This function can be sculpted to $\mathrm{D}(f|_P) = N/2$ and $\mathrm{C}(f|_P) = 1$ by setting P to the set of inputs with Hamming weight 1. However, $\mathrm{H}(\mathrm{C}_f) = O(\log N)$ for this function, since all inputs with Hamming weight at least 2 have small certificates (just display two '1' bits).

This means something qualitatively different happens with D vs. C than what was found in this chapter.

Appendix A

Properties of randomized algorithms

We now provide proofs of the properties of randomized algorithms described in Chapter 2, which we restate for convenience.

Lemma 2.12 (Markov's Inequality). *Let R be a randomized decision tree on n bits, and let $\delta \in (0,1)$. On any input $x \in \{0,1\}^n$, the probability that $h(D,x) \leq \lfloor \overline{h}(R,x)/\delta \rfloor$ when D is sampled from R is at least $1-\delta$.*

Proof. If A does not terminate within $\lfloor T/\delta \rfloor$ queries, it must use at least $\lfloor T/\delta \rfloor + 1$ queries. Let's say this happens with probability p. Then the expected number of queries used by A is at least $p(\lfloor T/\delta \rfloor + 1)$ (using the fact that the number of queries used is always non-negative). We then get $T \geq p(\lfloor T/\delta \rfloor + 1) > pT/\delta$, or $p < \delta$. Thus A terminates within T/δ queries with probability greater than $1-\delta$. □

Lemma 2.10 (Amplification). *If f is a Boolean function and R is a randomized algorithm for f with error $\epsilon < 1/2$, repeating R several times and taking the majority vote of the outcomes decreases the error. To reach error $\epsilon' > 0$, it suffices to repeat the algorithm $\frac{2\ln(1/\epsilon')}{(1-2\epsilon)^2}$ times. In particular, if $0 < \epsilon' < \epsilon < 1/2$, we have*

$$\mathrm{R}_\epsilon(f) \leq \mathrm{R}_{\epsilon'}(f) \leq \frac{2\ln(1/\epsilon')}{(1-2\epsilon)^2}\,\mathrm{R}_\epsilon(f).$$

Proof. Let's repeat A an odd number of times, say $2k+1$. The error probability of A', the algorithm that takes the majority vote of these runs, is

$$\sum_{i=0}^{k} \binom{2k+1}{i} \epsilon^{2k+1-i}(1-\epsilon)^i \leq \epsilon^{2k+1-k}(1-\epsilon)^k \sum_{i=0}^{k} \binom{2k+1}{i}, \tag{A.1}$$

which is at most

$$\epsilon^{k+1}(1-\epsilon)^k(2^{2k+1}/2) = \epsilon^{k+1}(1-\epsilon)^k 4^k = \epsilon(4\epsilon(1-\epsilon))^k = \epsilon(1-(1-2\epsilon)^2)^k. \tag{A.2}$$

It suffices to choose k large enough so that $\epsilon(1-(1-2\epsilon)^2)^k \leq \epsilon'$, or $\ln\epsilon + k\ln(1-(1-2\epsilon)^2) \leq \ln\epsilon'$. Using the inequality $\ln(1-x) \leq -x$, it suffices to choose k so that $k(1-2\epsilon)^2 \geq \ln(1/\epsilon') - \ln(1/\epsilon)$, or

$$k \geq \frac{\ln(1/\epsilon') - \ln(1/\epsilon)}{(1-2\epsilon)^2}. \tag{A.3}$$

In particular, we can choose

$$k = \left\lceil \frac{\ln(1/\epsilon') - \ln(1/\epsilon)}{(1-2\epsilon)^2} \right\rceil \leq \frac{\ln(1/\epsilon')}{(1-2\epsilon)^2} + 1 - \frac{\ln(1/\epsilon)}{(1-2\epsilon)^2}. \tag{A.4}$$

It is not hard to check that $3(1-2\epsilon)^2 \leq 2\ln(1/\epsilon)$ for all $\epsilon \in (0, 1/2)$, so we can choose k to be at most $\frac{\ln(1/\epsilon')}{(1-2\epsilon)^2} - 1/2$. This means $2k+1$ is at most $\frac{2\ln(1/\epsilon')}{(1-2\epsilon)^2}$, as desired. □

Lemma 4.3. *Let f be a partial function, $\delta > 0$, and $\epsilon \in [0, 1/2)$. Then we have $\mathrm{R}_{\epsilon+\delta}(f) \leq \frac{1-2\epsilon}{2\delta}\overline{\mathrm{R}}_\epsilon(f) \leq \frac{1}{2\delta}\overline{\mathrm{R}}_\epsilon(f)$.*

Proof. Let A be the $\overline{\mathrm{R}}_\epsilon(f)$ algorithm. Let B be the algorithm that runs A for $\lfloor \overline{\mathrm{R}}_\epsilon(f)/\alpha \rfloor$ queries, and if A doesn't terminate, outputs 0 with probability 1/2 and 1 with probability 1/2. Then by Lemma 2.12, the error probability of B is at most $\alpha/2 + (1-\alpha)\epsilon$. If we let $\alpha = 2\delta/(1-2\epsilon)$, then the error probability of B is at most

$$\frac{\delta}{1-2\epsilon} + \frac{(1-2\epsilon-2\delta)\epsilon}{1-2\epsilon} = \epsilon + \delta, \tag{A.5}$$

as desired. The number of queries made by B is at most $\lfloor \overline{\mathrm{R}}_\epsilon(f)/\alpha \rfloor \leq \frac{1-2\epsilon}{2\delta}\overline{\mathrm{R}}_\epsilon(f) \leq \frac{1}{2\delta}\overline{\mathrm{R}}_\epsilon(f)$. □

Lemma 2.13. *If f is a (possibly partial) Boolean function, then for all $\epsilon \in (0, \frac{1}{2})$, we have $\mathrm{R}_\epsilon(f) \leq 14\frac{\ln(1/\epsilon)}{(1-2\epsilon)^2}\overline{\mathrm{R}}_\epsilon(f)$. When $\epsilon = \frac{1}{3}$, we can improve this to $\mathrm{R}(f) \leq 10\overline{\mathrm{R}}(f)$.*

Proof. Repeating an algorithm with error 1/3 three times decreases its error to 7/27, so in particular $\overline{\mathrm{R}}_{7/27}(f) \leq 3\overline{\mathrm{R}}(f)$. Then using Lemma 4.3 with $\epsilon + \delta = 1/3$ and $\epsilon = 7/27$, we get

$$\mathrm{R}(f) \leq \frac{1-14/27}{2(1/3-7/27)}\overline{\mathrm{R}}_{7/27}(f) = \frac{13}{4}\overline{\mathrm{R}}_{7/27}(f) \leq \frac{39}{4}\overline{\mathrm{R}}(f) \leq 10\overline{\mathrm{R}}(f). \tag{A.6}$$

To deal with arbitrary ϵ, we need to use Lemma 2.10. It gives $\mathrm{R}_{\epsilon'}(f) \leq \frac{2\ln(1/\epsilon')}{(1-2\epsilon)^2}\mathrm{R}_\epsilon(f)$. When combined with Lemma 4.3, this gives

$$\mathrm{R}_{\epsilon'}(f) \leq \frac{1-2\epsilon'}{\epsilon-\epsilon'}\frac{\ln(1/\epsilon')}{(1-2\epsilon)^2}\overline{\mathrm{R}}_{\epsilon'}(f). \tag{A.7}$$

Setting $\epsilon = (1+4\epsilon')/6$ (which is greater than ϵ' if $\epsilon' < 1/2$) gives

$$\mathrm{R}_{\epsilon'}(f) \leq \frac{27\ln(1/\epsilon')}{2(1-2\epsilon')^2}\overline{\mathrm{R}}_{\epsilon'}(f) \leq 14\frac{\ln(1/\epsilon')}{(1-2\epsilon')^2}\overline{\mathrm{R}}_{\epsilon'}(f), \tag{A.8}$$

which gives the desired result (after exchanging ϵ and ϵ'). □

Lemma 4.4. *Let R be a randomized algorithm that uses T queries on expectation and finds a certificate with probability $1-\epsilon$. Then repeating A when it fails to find a certificate turns it into an algorithm that always finds a certificate (i.e., a zero-error algorithm) that uses at most $T/(1-\epsilon)$ queries in expectation.*

Proof. Let A' be the algorithm that runs A, checks if it found a certificate, and repeats if it didn't. Let N_1 be the random variable for the number of queries used by A'. We know

that the maximum number of queries A' ever uses is the input size; it follows that $\mathbb{E}(N_1)$ converges and is at most the input size.

Let M_1 be the random variable for the number of queries used by A in the first iteration. Let S_1 be the Bernoulli random variable for the event that A fails to find a certificate. Then $\mathbb{E}(M_1) = T$ and $\mathbb{E}(S_1) = \epsilon$. Let N_2 be the random variable for the number of queries used by A' starting from the second iteration (conditional on the first iteration failing). Then

$$N_1 = M_1 + S_1 N_2, \tag{A.9}$$

so by linearity of expectation and independence,

$$\mathbb{E}(N_1) = \mathbb{E}(M_1) + \mathbb{E}(S_1)\mathbb{E}(N_2) = T + \epsilon\mathbb{E}(N_2) \leq T + \epsilon\mathbb{E}(N_1). \tag{A.10}$$

This implies

$$\mathbb{E}(N_1) \leq T/(1-\epsilon), \tag{A.11}$$

as desired. □

Lemma 2.28. *Let f be a (possibly partial) function, let $x \in \mathrm{Dom}(f)$ be an input, and let B be a sensitive block of x with respect to f.*

Then any randomized algorithm that solves f using at most T expected queries and with error at most ϵ must, when run on x, query a bit in B with probability at least $1 - 2\epsilon$.

Proof. Let p be the probability that A queries an entry on which x differs from y when it is run on x. Let q be the probability that A outputs an invalid output for x given that it doesn't query a difference from y. Let r be the probability that A outputs an invalid output for y given that it doesn't query such a difference. Since one of these events always happens, we have $q + r \geq 1$. Note that A errs with probability at least $(1-p)q$ when run on x and at least $(1-p)r$ when run on y. This means that $(1-p)q \leq \epsilon$ and $(1-p)r \leq \epsilon$. Summing these gives $1 - p \leq (1-p)(q+r) \leq 2\epsilon$, so $p \geq 1 - 2\epsilon$, as desired. □

Minimax theorem for bounded-error algorithms

We need the following version of Yao's minimax theorem for $\overline{\mathrm{R}}_\epsilon(f)$. The proof is similar to other minimax theorems in the literature, but we include it for completeness.

Theorem A.1. *Let f be a partial function and $\epsilon \geq 0$. Then there exists a distribution μ over $\mathrm{Dom}(f)$ such that any randomized algorithm A that computes f with error at most ϵ on all $x \in \mathrm{Dom}(f)$ satisfies $\mathbb{E}_{x\sim\mu}A(x) \geq \overline{\mathrm{R}}_\epsilon(f)$, where $A(x)$ is the expected number of queries made by A on x.*

We note that Theorem A.1 talks only about algorithms that successfully compute f (with error ϵ) on all inputs, not just those sampled from μ. An alternative minimax theorem where the error is with respect to the distribution μ can be found in [85], although it loses constant factors.

Proof. We think of a randomized algorithm as a probability vector over deterministic algorithms; thus randomized algorithms lie in $\mathbb{R}^N$, where N is the number of deterministic decision trees. In fact, the set S of randomized algorithms forms a simplex, which is a closed and bounded set.

Let $\mathrm{err}_{f,x}(A) := \Pr[A(x) \neq f(x)]$ be the probability of error of the randomized A when run on x. Then it is not hard to see that $\mathrm{err}_{f,x}(A)$ is a continuous function of A. Define $\mathrm{err}_f(A) := \max_x \mathrm{err}_{f,x}(A)$. Then $\mathrm{err}_f(A)$ is also the maximum of a finite number of continuous functions, so it is continuous.

Next consider the set of algorithms $S_\epsilon := \{A \in S : \mathrm{err}_f(A) \leq \epsilon\}$. Since $\mathrm{err}_f(A)$ is a continuous function and S is closed and bounded, it follows that S_ϵ is closed and bounded, and hence compact. It is also easy to check that S_ϵ is convex. Let P be the set of probability distributions over $\mathrm{Dom}(f)$. Then P is also compact and convex. Finally, consider the function $\alpha(A, \mu) := \mathbb{E}_{x \sim \mu} \mathbb{E}_{D \sim A} D(x)$ that accepts a randomized algorithm and a distribution as input, and returns the expected number of queries the algorithm makes on that distribution. It is not hard to see that α is a continuous function in both variables. In fact, α is linear in both variables by the linearity of expectation.

Since S_ϵ and P are compact and convex subsets of the finite-dimensional spaces $\mathbb{R}^N$ and $\mathbb{R}^{\mathrm{Dom}(f)}$ respectively, and the objective function $\alpha(\cdot, \cdot)$ is linear, we can apply Sion's minimax theorem (see [80] or [88, Theorem 1.12]) to get

$$\max_{\mu \in P} \min_{A \in S_\epsilon} \alpha(A, \mu) = \min_{A \in S_\epsilon} \max_{\mu \in P} \alpha(A, \mu). \tag{A.12}$$

The right hand side is simply the worst-case expected query complexity of any algorithm computing f with error at most ϵ, which is $\overline{\mathrm{R}}_\epsilon(f)$ by definition. The left hand side gives us a distribution μ such that for any algorithm A that makes error at most ϵ on all $x \in \mathrm{Dom}(f)$, the expected number of queries A makes on μ is at least $\overline{\mathrm{R}}_\epsilon(f)$. □

Appendix B

Quantum query complexity of k-SUM

Belovs and Špalek [18] proved a lower bound of $\Omega(n^{k/(k+1)})$ on the quantum query complexity of k-SUM for constant k, as long as the alphabet is large enough. However, in their result, Belovs and Špalek do not keep track of the dependence on k. We trace their proof to determine this dependence, ultimately proving the following theorem.

Theorem B.1. *For the function k-SUM : $[q]^n \to \{0,1\}$, if $|q| \geq 2\binom{n}{k}$, we have*

$$Q(k\text{-SUM}) = \Omega(n^{k/(k+1)}/\sqrt{k}), \tag{B.1}$$

where the constant in the Ω is independent of k.

Proof. We proceed by tracing Belovs and Špalek's proof, which can be found in section 3 of [18], to extract an unsimplified lower bound. We then tune some parameters to determine the best dependence on k.

The proof in [18] proceeds by the generalized adversary method. Belovs and Špalek construct an adversary matrix Γ, which depends on parameters α_m for $m = 0, 1, \ldots, n-k$. This construction can be found in Section 3.1 of their paper. They then show a lower bound on $\|\Gamma\|$ and an upper bound on $\|\Gamma \circ \Delta_1\|$ in terms of the parameters α_m (the upper bounds on $\|\Gamma \circ \Delta_i\|$ for $i \neq 1$ follow by symmetry). Finally, they pick the values for the parameters that give the best lower bound.

A lower bound on $\|\Gamma\|$ without Ω notation, which therefore does not ignore factors of k, can be found in Section 3.6 of their paper. In that section, they show

$$\|\Gamma\| \geq \alpha_0 \sqrt{\binom{n}{k}\left(1 - \frac{1}{q}\binom{n}{k}\right)}, \tag{B.2}$$

where q is the alphabet size. When $q \geq \frac{4}{3}\binom{n}{k}$, this gives

$$\|\Gamma\| \geq \frac{\alpha_0}{2}\sqrt{\binom{n}{k}}. \tag{B.3}$$

For the upper bound on $\|\Gamma \circ \Delta_1\|$, it turns out to be more convenient to switch to a different matrix, $\tilde{\Gamma}$, which satisfies $\|\Gamma \circ \Delta_1\| \leq \|\tilde{\Gamma} \circ \Delta_1\|$. This is explained in the beginning of Section 3 of their paper and in Section 3.1. To upper bound $\|\tilde{\Gamma} \circ \Delta_1\|$, in Section 3.3 the authors switch to yet another matrix, $\tilde{\Gamma}_1$, with the property that $\tilde{\Gamma}_1 \circ \Delta_1 = \tilde{\Gamma} \circ \Delta_1$ and that $\|\tilde{\Gamma}_1 \circ \Delta_1\| \leq 2\|\tilde{\Gamma}_1\|$. This gives $\|\Gamma \circ \Delta_1\| \leq 2\|\tilde{\Gamma}_1\|$, so it suffices to upper bound $\|\tilde{\Gamma}_1\|$.

In Section 3.4, the authors express $\|\tilde{\Gamma}_1\|^2$ as the norm of a sum of matrices, with the sum ranging over a set $\mathcal{S}$. They split $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$; by the triangle inequality, it then suffices to separately upper bound the norm of the sum over $\mathcal{S}_1$ and the norm of the sum over $\mathcal{S}_2$. The former is upper bounded by $\max_m \alpha_m^2 \binom{m+k-1}{k-1}$, and the latter by $k\binom{n-1}{k} \max_m (\alpha_m - \alpha_{m+1})^2$. We therefore conclude that

$$\|\Gamma \circ \Delta_1\|^2 \leq 4 \max_m \alpha_m^2 \binom{m+k-1}{k-1} + 4k \binom{n-1}{k} \max_m (\alpha_m - \alpha_{m+1})^2. \tag{B.4}$$

It remains to pick values for α_m to optimize the resulting adversary bound. There are no restrictions on the α_m parameters. To maximize the adversary bound, we want α_0 to be large, consecutive α's to be close (to minimize $\max_m (\alpha_m - \alpha_{m+1})^2$), and α_m for large m to be small (to minimize $\max_m \alpha_m^2 \binom{m+k-1}{k-1}$). We choose to make the α_m parameters decrease to 0 in an arithmetic sequence, with $\alpha_m - \alpha_{m+1} = c > 0$ for all $m \leq \alpha_0/c$, and $\alpha_m = 0$ for all $m \geq \alpha_0/c$. Let $\beta = \alpha_0/c$. Then $\alpha_m = \alpha_0 - cm$ for $m \leq \beta$.

We can write the final adversary bound as follows:

$$\text{Adv}(k\text{-SUM})^2 = \Omega\left(\frac{\alpha_0^2 \binom{n}{k}}{\max_{m \leq \beta} (\alpha_0 - cm)^2 \binom{m+k-1}{k-1} + k\binom{n-1}{k} c^2}\right) \tag{B.5}$$

$$= \Omega\left(\min\left\{\frac{\binom{n}{k}}{\max_{m \leq \beta} (1 - m/\beta)^2 \binom{m+k-1}{k-1}}, \frac{\binom{n}{k}}{k\binom{n-1}{k}} \beta^2\right\}\right) \tag{B.6}$$

The second term in the minimization simplifies to $\frac{n}{k(n-k)}\beta^2$, which is $\Omega(\beta^2/k)$. To deal with the first term, we use the well-known inequalities $(n/k)^k \leq \binom{n}{k} \leq (en/k)^k$, which hold for all n and k. This gives

$$\text{Adv}(k\text{-SUM})^2 = \Omega\left(\min\left\{\frac{n^k}{(k-1)\max_{m \leq \beta} (1 - m/\beta)^2 (e(m+k-1))^{k-1}} \left(\frac{k-1}{k}\right)^k, \frac{\beta^2}{k}\right\}\right)$$

$$= \Omega\left(\min\left\{\frac{n^k}{k \max_{m \leq \beta} (1 - m/\beta)^2 (e(m+k))^{k-1}}, \frac{\beta^2}{k}\right\}\right).$$

We can solve the maximization in the denominator using calculus. The unique maximum occurs at $m = \beta - \left(\frac{2}{k+1}\right)(\beta - k)$. Substituting this in and simplifying, we get

$$\text{Adv}(k\text{-SUM})^2 = \Omega\left(\min\left\{\frac{k\beta}{(1-k/\beta)^2}\left(\frac{n}{e(\beta+3)}\right)^k, \frac{\beta^2}{k}\right\}\right) = \Omega\left(\min\left\{k\beta\left(\frac{n}{3\beta}\right)^k, \frac{\beta^2}{k}\right\}\right)$$

where for the last equality we assumed $\beta \geq 2k$ and $\beta \geq 3e/(3-e)$.

Finally, we set $\beta = (1/3)n^{k/(k+1)}$. This gives

$$\text{Adv}(k\text{-SUM})^2 = \Omega\left(\min\left\{kn^{k/(k+1)}\left(\frac{n}{n^{k/(k+1)}}\right)^k, \frac{n^{2k/(k+1)}}{k}\right\}\right) = \Omega(n^{2k/(k+1)}/k), \tag{B.7}$$

so

$$Q(k\text{-SUM}) = \Omega(\text{Adv}(k\text{-SUM})) = \Omega\left(n^{k/(k+1)}/\sqrt{k}\right). \tag{B.8}$$

Note that we assumed $\beta \geq 2k$. Since $\beta = n^{k/(k+1)}/3$, we need $n^{k/(k+1)} \geq 6k$, or

$n \geq (6k)^{1+1/k}$. Since $(6k)^{1/k} = \exp(\ln(6k)/k) = 1 + \Theta(\log(k)/k)$, it suffices to have $n \geq 6k + \omega(\log k)$. In particular, this bound works as long as we have $k \leq n/10$. When $k \geq n/10$, we can directly prove a lower bound of $\Omega(\sqrt{n})$ by a reduction from Grover search; this means there are no restrictions on the size of n and k in this result. □

Appendix C

Measures that behave curiously with cheat sheets

In this appendix we show that R_1 and Q_1 can behave strangely on cheat sheet functions, potentially decreasing from $R_1(f)$ to $R(f)$ and from $Q_1(f)$ to $Q(f)$.

Theorem C.1. *Let $f : D \to \{0,1\}$ be a partial function, where $D \subseteq [M]^n$, and let ϕ be a certifying function for f. Then*

$$R_1(f_{\mathrm{CS}(\phi)}) = \tilde{O}(R(f) + R_0(\phi)) \qquad \text{and} \qquad Q_1(f_{\mathrm{CS}(\phi)}) = \tilde{O}(Q(f) + Q_0(\phi)). \tag{C.1}$$

Proof. We show a randomized (and quantum) algorithm for $f_{\mathrm{CS}(\phi)}$ that uses the required number of queries and finds a 1-certificate with constant probability. Such an algorithm could be made to have one-sided error by outputting 1 only if it finds a 1-certificate.

The algorithm works as follows. First, use the randomized (resp. quantum) algorithm on the c inputs to f (with some amplification), to determine the number ℓ of the correct cheat sheet with high probability (assuming the promises of f all hold). Next, go to the ℓ^{th} cheat sheet, and use a zero-error algorithm to evaluate $\phi(x_i, y_i)$ for $i = 1, 2, \ldots, c$, where x_i are the inputs to f and y_i are the cheat sheet strings. This finds certificates for each input to ϕ.

Now, if the value of the function is 1 on the given input, then with constant probability, all the c certificates found should be 1-certificates for ϕ. These 1-certificates certify the value of the inputs to f. Therefore, taken together, they constitute a valid 1-certificate for $f_{\mathrm{CS}(\phi)}$. It follows that this algorithm uses $\tilde{O}(R(f) + R_0(\phi))$ randomized queries and finds a 1-certificate with constant probability. The result for quantum algorithms follows similarly. □

We now show that it is possible for $R_1(f_{\mathrm{CS}(\phi)})$ to be much smaller than $R_1(f)$, unlike some of the other measures studied. Intuitively, this is because 1-inputs of $f_{\mathrm{CS}(\phi)}$ contains a cheat sheet with a certificate and hence even if the algorithm is not sure of its answer before finding the cheat sheet, the cheat sheet may convince the algorithm of its answer.

Theorem C.2. *There is a total function $f : \{0,1\}^n \to \{0,1\}$ and a certifying function ϕ for f such that $R_1(f_{\mathrm{CS}(\phi)}) = \tilde{O}(\sqrt{R_1(f)})$.*

Proof. We can use Theorem C.1 to construct an explicit function f and a certifying function ϕ for f such that $R_1(f_{\mathrm{CS}(\phi)})$ is smaller than $R_1(f)$. The construction is as follows. In [9], a

Boolean function was constructed that has $R_0 = \tilde{\Omega}(n^2)$ and $R_1 = \tilde{O}(n)$. This function has certificate complexity roughly equal to n. Now, by taking the XOR of this function with an additional bit, we get a function f for which R_1 is roughly n^2 but R and C are still roughly n.

We define ϕ to be a certifying function that simply checks if y provides pointers to a certificate for f. Then $R_0(\phi) = \tilde{O}(n)$. It follows that $R_1(f_{\mathrm{CS}(\phi)}) = \tilde{O}(n)$, while $R_1(f) = \tilde{\Omega}(n^2)$. □

Lastly we show that even zero-error randomized query complexity behaves curiously with cheat sheets. We might have expected that the obvious upper bound $R_0(f_{\mathrm{CS}(\phi)}) = \tilde{O}(R_0(f) + R_0(\phi))$ holds, but in fact it does not. The reason is subtle and relates to the behavior of the zero-error algorithm on inputs outside of the domain D. For a partial function $f : D \to \{0,1\}$, a zero-error algorithm's behavior is not constrained on inputs outside the domain D. The algorithm could, for example, run forever on such inputs. The obvious algorithm, which simply runs the zero-error algorithm for f on all c inputs to f now fails to output any answer on 0-inputs to $f_{\mathrm{CS}(\phi)}$ in which the inputs to f do not lie in D. We exploit this observation to prove the following counterexample.

Theorem C.3. *There is a partial function $f : D \to \{0,1\}$, where $D \subseteq \{0,1\}^n$, and a certifying function ϕ for f such that $R_0(f_{\mathrm{CS}(\phi)}) = \tilde{\Omega}(R_0(f)^{3/2} + R_0(\phi)^{3/2})$.*

Proof. Let $g : \{0,1\}^{4m} \to \{0,1\}$ be the partial function defined as follows. On input (x,y) with $x, y \in \{0,1\}^{2m}$, define $g(x,y) = 0$ if the Hamming weight of x is m and the Hamming weight of y is 0, and define $g(x,y) = 1$ if the Hamming weight of y is m and the Hamming weight of x is 0. The promise of g is that one of these two conditions holds for every input (x,y).

It is easy to see that $R_0(g) = O(1)$. Let f be the composition of g with an m by m AND-OR, and let $n = 4m^3$. Then $R_0(f) = O(m^2) = O(n^{2/3})$. Let ϕ be a certifying function for f that takes an input to f and an additional string, and asserts that the latter provides pointers to a certificate for each AND-OR instance in the input to f. This assertion can be verified (or refuted) in $\tilde{O}(m^2)$ deterministic queries, so $R_0(\phi) \leq D(\phi) = \tilde{O}(m^2)$.

We now show that $R_0(f_{\mathrm{CS}(\phi)}) = \tilde{\Omega}(m^3)$. Consider an input consisting of c inputs to f, each of which has all its AND-OR instances evaluate to 0. Note that these inputs to f do not satisfy the promise of f. We attach a blank cheat sheet array after these c inputs. By definition of $f_{\mathrm{CS}(\phi)}$, the resulting input is therefore a 0-input to $f_{\mathrm{CS}(\phi)}$. However, any 0-certificate of it (of reasonable size) must "partially certify" at least half of the c inputs to f; that is, for at least half the inputs to f, it must either prove the input is not a 0-input or prove it is not a 1-input. The only way to do this is to display $(c/2)m$ 0-certificates for AND-OR instances.

This means we have an algorithm that takes in $4cm$ zero-inputs to AND-OR, and finds a certificate for at least $cm/8$ of them using $R_0(f_{\mathrm{CS}(\phi)})$ expected queries. It follows that given one input from the hard distribution over 0-inputs to f, we can find a certificate for it by generating $4cm - 1$ additional inputs from this distribution, mixing them together, and running the $R_0(f_{\mathrm{CS}(\phi)})$ algorithm. This finds a certificate with probability at least $1/8$, and uses $R_0(f_{\mathrm{CS}(\phi)})/cm$ expected queries. By repeating the algorithm if necessary, we find a certificate using $8R_0(f_{\mathrm{CS}(\phi)})/cm$ expected queries.

We construct a function f' is the same as f except with NOT-AND-OR instead of AND-OR. Repeating the argument provides a randomized algorithm that finds a 1-certificate

for AND-OR using $8R_0(f'_{CS(\phi')}/cm$ expected queries. By running both algorithms in parallel, we get

$$R_0(\text{AND-OR}) \leq 8R_0(f_{\text{CS}(\phi)})/cm + 8R_0(f'_{\text{CS}(\phi')})/cm. \tag{C.2}$$

Since $R_0(\text{AND-OR}) = \Omega(m^2)$, it follows that either $R_0(f_{\text{CS}(\phi)}) = \Omega(m^3)$ or $R_0(f'_{\text{CS}(\phi')}) = \Omega(m^3)$. The desired result follows. □

Appendix D

Properties of H Indices

Lemma D.1. *Let* $g : \{0,1\}^n \to [0,\infty)$. *Define*

$$\mathrm{H}(g) := \inf\left\{h \in [0,\infty) : |\{x \in \{0,1\}^n : g(x) > h\}| \leq 2^h\right\}.$$

Then

1. $\mathrm{H}(g) \in [0,n]$
2. $\mathrm{H}(g) \leq \max_x g(x)$
3. *The number of* $x \in \{0,1\}^n$ *for which* $g(x) > \mathrm{H}(g)$ *is at most* $2^{\mathrm{H}(g)}$ *(equivalently, the infimum in the definition of* $\mathrm{H}(g)$ *is actually a minimum)*
4. *If* $g' : \{0,1\}^n \to [0,\infty)$ *is such that* $g(x) \leq g'(x)$ *for all* $x \in \{0,1\}^n$, *then* $\mathrm{H}(g) \leq \mathrm{H}(g')$
5. *If* $\alpha : [0,\infty) \to [0,\infty)$ *is an increasing function, then* $\mathrm{H}(\alpha \circ g) \leq \max\{\mathrm{H}(g), \alpha(\mathrm{H}(g))\}$.
6. *There are at least* $2^{\mathrm{H}(g)}$ *inputs* $x \in \{0,1\}^n$ *with* $g(x) \geq \mathrm{H}(g)$.

Proof. Let $S_g(h) = \{x \in \{0,1\}^n : g(x) > h\}$ and let $H_g = \{h \in [0,\infty) : |S_g(h)| \leq 2^h\}$. Then $\mathrm{H}(g) = \inf H_g$. Part 1 follows from noticing that for all h, $S_g(h) \subseteq \{0,1\}^n$, so $|S_g(h)| \leq 2^n$, whence $n \in H_g$. Part 2 follows from noticing that $S_g(\max_x g(x))$ is empty, so $\max_x g(x) \in H_g$.

To show 3, we show that H_g contains its infimum. Consider an infinite decreasing sequence $h_1, h_2, \ldots \in H_g$ that converges to $\mathrm{H}(g)$. Then the sequence $|S_g(h_1)|, |S_g(h_2)|, \ldots$ is a non-decreasing sequence of integers which is bounded above by 2^n. In addition, $S_g(h_i) \subseteq S_g(h_{i+1})$ for all i. It follows that there is some ℓ such that $S_g(h_i) = S_g(h_\ell)$ for all $i \geq \ell$. For each $x \in S_g(h_\ell)$, we have $g(x) > h_\ell > \mathrm{H}(g)$, and for each $x \notin S_g(h_\ell)$, we have $g(x) \leq h_i$ for all i. It follows that $g(x) \leq \mathrm{H}(g)$ for each $x \notin S_g(h_\ell)$, so $S_g(\mathrm{H}(g)) = \{x \in \{0,1\}^n : g(x) > \mathrm{H}(g)\} = S_g(h_i)$ for all $i \geq \ell$. Finally, since $h_i \in H_g$ for all i, we have $|S_g(\mathrm{H}(g))| = |S_g(h_i)| \leq 2^{h_i}$ for all $i \geq \ell$. From this it follows that $|S_g(\mathrm{H}(g))| \leq \lim_{i\to\infty} 2^{h_i} = 2^{\mathrm{H}(g)}$, so $\mathrm{H}(g) \in H_g$.

We now show 4. If g' is point-wise greater or equal to g, then $S_g(\mathrm{H}(g')) \subseteq S_{g'}(\mathrm{H}(g'))$. Since $\mathrm{H}(g') \subseteq H_g$, we have $|S_{g'}(\mathrm{H}(g'))| \leq 2^{\mathrm{H}(g')}$, so $|S_g(\mathrm{H}(g'))| \leq 2^{\mathrm{H}(g')}$. Thus $\mathrm{H}(g') \in H_g$, so $\mathrm{H}(g) = \inf H_g \leq \mathrm{H}(g')$.

We prove 5. Let α be an increasing function. We have

$$\begin{aligned} S_g(\mathrm{H}(g)) &= \{x \in \{0,1\}^n : g(x) > \mathrm{H}(g)\} \\ &= \{x \in \{0,1\}^n : \alpha \circ g(x) > \alpha(\mathrm{H}(g))\} \\ &= S_{\alpha \circ g}(\alpha(\mathrm{H}(g)). \end{aligned}$$

Thus

$$|S_{\alpha \circ g}(\max\{\mathrm{H}(g), \alpha(\mathrm{H}(g))\})| \leq |S_{\alpha \circ g}(\alpha(\mathrm{H}(g))| = |S_g(\mathrm{H}(g))| \leq 2^{\mathrm{H}(g)} \leq 2^{\max\{\mathrm{H}(g), \alpha(\mathrm{H}(g))\}}$$

so $\max\{\mathrm{H}(g), \alpha(\mathrm{H}(g))\} \in H_{\alpha \circ g}$. Hence $\mathrm{H}(\alpha \circ g) \leq \max\{\mathrm{H}(g), \alpha(\mathrm{H}(g))\}$.

Finally, we show 6. If it was false, there would be less than $2^{\mathrm{H}(g)}$ inputs with $g(x) \geq \mathrm{H}(g)$. Thus there is some $\epsilon > 0$ such that there are less than $2^{\mathrm{H}(g)-\epsilon}$ inputs with $g(x) \geq \mathrm{H}(g) > \mathrm{H}(g) - \epsilon$. But this implies $\mathrm{H}(g) - \epsilon \geq \mathrm{H}(g)$, a contradiction. □

Bibliography

[1] Scott Aaronson. Quantum certificate complexity. *Journal of Computer and System Sciences*, 74(3):313–322, 2008. Computational Complexity 2003.

[2] Scott Aaronson and Andris Ambainis. Forrelation: A problem that optimally separates quantum from classical computing. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 307–316. ACM, 2015.

[3] Scott Aaronson and Shalev Ben-David. Sculpting quantum speedups. In *31st Conference on Computational Complexity (CCC)*, pages 26:1–26:28, 2016.

[4] Scott Aaronson, Shalev Ben-David, and Robin Kothari. Separations in query complexity using cheat sheets. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 863–876, 2016.

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

[6] Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, June 2002.

[7] Andris Ambainis. Polynomial degree vs. quantum query complexity. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (FOCS 2003)*, pages 230–239, 2003.

[8] Andris Ambainis. Superlinear advantage for exact quantum algorithms. In *Proceedings of the 45th ACM Symposium on Theory of Computing (STOC 2013)*, pages 891–900, 2013.

[9] Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, and Juris Smotrovs. Separations in query complexity based on pointer functions. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 800–813, 2016.

[10] Andris Ambainis and Ronald de Wolf. How low can approximate degree and quantum query complexity be for total boolean functions? *Comput. Complex.*, 23(2):305–322, June 2014.

[11] Andris Ambainis, Martins Kokainis, and Robin Kothari. Nearly optimal separations between communication (or query) complexity and partitions. In *31st Conference on Computational Complexity (CCC)*, volume 50 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:14, 2016.

[12] Anurag Anshu, Aleksandrs Belovs, Shalev Ben-David, Mika Göös, Rahul Jain, Robin Kothari, Troy Lee, and Miklos Santha. Separations in communication complexity using cheat sheets and information complexity. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2016.

[13] Anurag Anshu, Naresh B Goud, Rahul Jain, Srijita Kundu, and Priyanka Mukhopadhyay. Lifting randomized query complexity to randomized communication complexity. *arXiv preprint arXiv:1703.07521*, 2017.

[14] Boaz Barak, Mark Braverman, Xi Chen, and Anup Rao. How to compress interactive communication. *SIAM Journal on Computing*, 42(3):1327–1363, 2013.

[15] Howard Barnum, Michael Saks, and Mario Szegedy. Quantum decision trees and semidefinite programming. Technical report, Los Alamos National Laboratory, 2001.

[16] Howard Barnum, Michael Saks, and Mario Szegedy. Quantum query complexity and semi-definite programming. In *18th Conference on Computational Complexity (CCC 2003)*, pages 179–193, 2003.

[17] 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.

[18] Aleksandrs Belovs and Robert Špalek. Adversary lower bound for the k-sum problem. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 323–328, 2013.

[19] Shalev Ben-David. A super-grover separation between randomized and quantum query complexities. *arXiv preprint*, 2015.

[20] Shalev Ben-David and Robin Kothari. Randomized query complexity of sabotaged and composed functions. In *43rd International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 60:1–60:14, 2016.

[21] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing (special issue on quantum computing)*, 26:1510–1523, 1997.

[22] Leonard Berman and Juris Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322, 1977.

[23] Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 118–126. IEEE, 1987.

[24] Adam Bouland, Lijie Chen, Dhiraj Holden, Justin Thaler, and Prashant Nalini Vasudevan. On the power of statistical zero knowledge. *To appear in Symposium on Foundations of Computer Science (FOCS 2017)*, 2017.

[25] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum computation and information*, volume 305 of *Contemporary Mathematics*, pages 53–74. AMS, 2002.

[26] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[27] Harry Buhrman, Lance Fortnow, Ilan Newman, and Hein Röhrig. Quantum property testing. *SIAM Journal on Computing*, 37(5):1387–1400, 2008.

[28] Harry Buhrman, Ilan Newman, Hein Rohrig, and Ronald de Wolf. Robust polynomials and quantum algorithms. *Theory of Computing Systems*, 40(4):379–395, 2007.

[29] Mark Bun and Justin Thaler. A nearly optimal lower bound on the approximate degree of AC^0. *arXiv preprint arXiv:1703.05784*, 2017.

[30] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, 1998.

[31] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 439(1907):553–558, 1992.

[32] Andrew Drucker. Improved direct product theorems for randomized query complexity. *Computational Complexity*, 21(2):197–244, 2012.

[33] Hartmut Ehlich and Karl Zeller. Schwankung von polynomen zwischen gitterpunkten. *Mathematische Zeitschrift*, 86(1):41–44, 1964.

[34] Tomás Feder, Eyal Kushilevitz, Moni Naor, and Noam Nisan. Amortized communication complexity. *SIAM Journal on Computing*, 24(4):736–750, 1995.

[35] Philippe Flajolet and Jean-Marc Steyaert. On sets having only hard subsets. In *Automata, Languages and Programming*, pages 446–457. Springer, 1974.

[36] Justin Gilmer, Michael Saks, and Sudarshan Srinivasan. Composition limits and separating examples for some boolean function complexity measures. *Combinatorica*, pages 1–47, 2016. CCC 2013.

[37] Oded Goldreich, Amit Sahai, and Salil Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In *Proceedings of the 30th Symposium on Theory of Computing*, STOC '98, pages 399–408, 1998.

[38] Mika Göös, T.S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication vs. partition number. *Electronic Colloquium on Computational Complexity (ECCC)* TR15-169, 2015.

[39] Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. In *Foundations of Computer Science (FOCS 2015)*, pages 1077–1088. IEEE, 2015. TR15-050.

[40] Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for BPP. *arXiv preprint arXiv:1703.07666*, 2017.

[41] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[42] Juris Hartmanis and Lane A Hemachandra. One-way functions, robustness, and the non-isomorphism of NP-complete sets. Technical report, Cornell University, 1986.

[43] Jorge E Hirsch. An index to quantify an individual's scientific research output. *Proceedings of the National academy of Sciences of the United States of America*, 102(46):16569–16572, 2005.

[44] Peter Høyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC 2007)*, pages 526–535, 2007.

[45] Peter Høyer, Michele Mosca, and Ronald de Wolf. Quantum search on bounded-error inputs. In *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 291–299. Springer Berlin Heidelberg, 2003.

[46] Rahul Jain and Hartmut Klauck. The partition bound for classical communication complexity and query complexity. In *Proceedings of the 2010 IEEE 25th Annual Conference on Computational Complexity*, CCC '10, pages 247–258, 2010.

[47] Rahul Jain, Hartmut Klauck, and Miklos Santha. Optimal direct sum results for deterministic and randomized decision tree complexity. *Information Processing Letters*, 110(20):893 – 897, 2010.

[48] Rahul Jain, Troy Lee, and Nisheeth K. Vishnoi. A quadratically tight partition bound for classical communication complexity and query complexity. *arXiv preprint* `arXiv:1401.4512`, 2014.

[49] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 355–364. ACM, 2003.

[50] 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.

[51] Shelby Kimmel. Quantum adversary (upper) bound. In *Automata, Languages, and Programming*, volume 7391 of *Lecture Notes in Computer Science*, pages 557–568, 2012.

[52] Boáz Klartag and Oded Regev. Quantum one-way communication can be exponentially stronger than classical communication. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 31–40. ACM, 2011.

[53] Ilan Kremer. *Quantum communication.* PhD thesis, Citeseer, 1995.

[54] Raghav Kulkarni and Avishay Tal. On fractional block sensitivity. *Electronic Colloquium on Computational Complexity (ECCC)* `TR13-168`, 2013.

[55] Stuart A Kurtz, Stephen R Mahaney, and James S Royer. The isomorphism conjecture fails relative to a random oracle. *Journal of the ACM (JACM)*, 42(2):401–420, 1995.

[56] Sophie Laplante and Frédéric Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. In *Proceedings of the 19th Conference on Computational Complexity*, pages 294–304, June 2004.

[57] François Le Gall. Exponential separation of quantum and classical online space complexity. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, pages 67–73, New York, NY, USA, 2006. ACM.

[58] Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Špalek, and Mario Szegedy. Quantum query complexity of state conversion. In *Foundations of Computer Science (FOCS 2011)*, pages 344–353, 2011.

[59] Troy Lee and Jérémie Roland. A strong direct product theorem for quantum query complexity. *computational complexity*, 22(2):429–462, 2013.

[60] Gatis Midrijanis. Exact quantum query complexity for total boolean functions. *arXiv preprint* `arXiv:quant-ph/0403168`, 2004.

[61] Gatis Midrijanis. On randomized and quantum query complexities. *arXiv preprint quant-ph/0501142*, 2005.

[62] Marvin Minsky and Seymour Papert. *Perceptrons*. MIT press, 1969.

[63] Ashley Montanaro. A composition theorem for decision tree complexity. *Chicago Journal of Theoretical Computer Science*, 2014(6), July 2014.

[64] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge Series on Information and the Natural Sciences. Cambridge University Press, 2000.

[65] Noam Nisan. Crew prams and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991.

[66] Noam Nisan and Mario Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity*, 15(4):557–565, 1995.

[67] Noam Nisan and Avi Wigderson. On rank vs. communication complexity. *Combinatorica*, 15(4):557–565, 1995.

[68] Denis Pankratov. Direct sum questions in classical communication complexity. Master's thesis, University of Chicago, 2012.

[69] Ran Raz. Exponential separation of quantum and classical communication complexity. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 358–367. ACM, 1999.

[70] Ben W Reichardt. Reflections for quantum query algorithms. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms (SODA 2011)*, pages 560–569. SIAM, 2011.

[71] Theodore J Rivlin and Elliott Ward Cheney. A comparison of uniform approximations on an interval and a finite subset thereof. *SIAM Journal on numerical Analysis*, 3(2):311–320, 1966.

[72] Amit Sahai and Salil Vadhan. A complete problem for statistical zero knowledge. *Journal of the ACM*, 50(2):196–249, March 2003.

[73] Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 29–38. IEEE, 1986.

[74] Miklos Santha. On the monte carlo boolean decision tree complexity of read-once formulae. *Random Structures & Algorithms*, 6(1):75–87, 1995.

[75] Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

[76] Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

[77] Alexander A. Sherstov. Making polynomials robust to noise. In *Proceedings of the 44th Symposium on Theory of Computing*, STOC '12, pages 747–758, 2012.

[78] 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.

[79] Daniel R Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[80] Maurice Sion. On general minimax theorems. *Pacific Journal of Mathematics*, 8(1):171–176, 1958.

[81] Robert Špalek and Mario Szegedy. All quantum adversary methods are equivalent. *Theory of Computing*, 2(1):1–18, 2006.

[82] List of Open Problems in Sublinear Algorithms. Problem 77: Frontiers in Structural Communication Complexity. `https://sublinear.info/index.php?title=Open_Problems:77`, 2017.

[83] Avishay Tal. Properties and applications of boolean function composition. In *Innovations in Theoretical Computer Science (ITCS 2013)*, pages 441–454, 2013. `TR12-163`.

[84] 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, Dec. 1989.

[85] Nikolai K. Vereshchagin. Randomized boolean decision trees: Several remarks. *Theoretical Computer Science*, 207(2):329 – 342, 1998.

[86] John Watrous. Limits on the power of quantum statistical zero-knowledge. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, FOCS '02, pages 459–, 2002.

[87] John Watrous. Zero-knowledge against quantum attacks. *SIAM Journal on Computing*, 39(1):25–58, 2009.

[88] John Watrous. *Theory of Quantum Information*. Unpublished, January 2016. Available at `https://cs.uwaterloo.ca/~watrous/TQI/`.

[89] A. Yao. Probabilistic computations: Toward a unified measure of complexity. *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.

[90] Bohua Zhan, Shelby Kimmel, and Avinatan Hassidim. Super-polynomial quantum speed-ups for boolean evaluation trees with hidden structure. In *Proceedings of the 3rd Innovations in Theoretical Computer Science conference*, pages 249–265. ACM, 2012.

[91] Shengyu Zhang. On the power of Ambainis lower bounds. *Theoretical Computer Science*, 339(2):241–256, 2005.