UNIVERSITY OF CALIFORNIA, SAN DIEGO

Function Computation with Optimal Number of Queries

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Communication Theory and Systems)

by

Ashkan Jafarpour Koujahi

Committee in charge:

Professor Alon Orlitsky, Chair
Professor Kamalika Chaudhuri
Professor Young-Han Kim
Professor Ramamohan Paturi
Professor Kenneth Zeger

2015

The dissertation of Ashkan Jafarpour Koujahi is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2015

DEDICATION

To my family.

EPIGRAPH

Things come true when you don't expect them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, colleagues, friends and family for their help and support in the making of this dissertation. Their support was a great help for me to complete this work.

My special thanks go to my advisor Alon Orlitsky for recruiting and supporting me before and after joining his group at UCSD. His passion always inspired me and meeting him was encouragement for me to pursue my research. His broad knowledge yet clear insights helped me to find new ideas and pursue them. I thank him and admire his insights to ideas that we discussed.

I am very thankful to my committee members Kamalika Chaudhuri, Young-Han Kim, Ramamohan Paturi, and Kenneth Zeger for taking active interest in my research. I remember attending their graduate courses and want to thank them for teaching me new courses that indeed helped me to better continue my research. I have also enjoyed taking many other courses at UCSD, especially those taught by Larry Milstein and Paul Siegel.

I thank all my kind and wise colleagues Hirakendu Das, Shengjun Pan, Jayadev Acharya, Ananda Theertha Suresh, Venkatadheeraj Pichapati, and Moein Falahatgar for their collaboration and assistance and I am glad for having them. There is a long list of friends at university and outside, which is not possible to name all of them without forgetting part of the list. I am thankful to all of them and appreciate their accompany during the time of my PhD.

I also want to thank my family from the deepest part of my feelings. They are not distance wise close to me now, but they have been always supporting and helping me without any question and I owe them a lot of love and hope to give it back to them soon.

Chapters 1, 2, and 3 are adapted from Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Sorting with adversarial comparators and application to density estimation", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 2014; and Jayadev Acharya, Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Maximum Selection and Sorting with Adversarial Comparators and Application to Density Estima-

tion", In Preparation, 2015.

Chapters 4 and 5 are adapted from Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Shengjun Pan, Ananda Theertha Suresh, "On the query computation and verification of functions", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 2012; and Jayadev Acharya, Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "On the Computation and Verification Query complexity of Symmetric Functions", In preparation, 2015.

The dissertation author is a primary researcher and author of the above papers.

VITA

2009	B. S. in Electrical Engineering, Sharif University of Technology
2009-2015	Graduate Student Researcher, University of California, San Diego
2011	M. S. in Electrical Engineering (Communication Theory and Systems), University of California, San Diego
2015	Ph. D. in Electrical Engineering (Communication Theory and Systems), University of California, San Diego

PUBLICATIONS

Jayadev Acharya, Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Shengjun Pan, "Competitive closeness testing", *Journal of Machine Learning Research - Proceedings Track (COLT)*, 47-68, 2011.

Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, "Expected Query Complexity of Symmetric Boolean Functions", *Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 26-29, 2011.

Jayadev Acharya, Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Shengjun Pan, "Estimating multiple concurrent processes", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 1628-1632, 2012.

Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Shengjun Pan, Ananda Theertha Suresh, "On the query computation and verification of functions", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 2711-2715, 2012.

Jayadev Acharya, Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Shengjun Pan, Ananda Theertha Suresh, "Competitive classification and closeness testing", *Journal of Machine Learning Research - Proceedings Track (COLT)*, 22.1-22.18, 2012.

Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "A competitive test for uniformity of monotone distributions", *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics(AISTATS)*, 57-65, 2013.

Jayadev Acharya, Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Tight bounds for universal compression of large alphabets", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 2875-2879, 2013.

Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Optimal probability estimation with applications to prediction and classification", *Journal of Machine Learning Research - Proceedings Track (COLT)*, 764-796, 2013.

Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Efficient Compression of Monotone and m-Modal Distributions", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 1867-1871, 2014.

Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Poissonization and universal compression of envelope classes", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 1872-1876, 2014.

Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Sublinear algorithms for outlier detection and generalized closeness testing", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 3200-3204, 2014.

Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Sorting with adversarial comparators and application to density estimation", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 1682-1686, 2014.

Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Universal compression of envelope classes: Tight characterization via Poisson sampling", *Submitted to IEEE Transactions on Information Theory*, 2014.

Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Near-Optimal-Sample Estimators for Spherical Gaussian Mixtures", *Proceedings of the Neural Information Processing Systems (NIPS)*, 1395-1403, 2014.

Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Venkatadheeraj Pichapati, Ananda Theertha Suresh, "Universal Compression of Power-Law Distributions", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 2015.

Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Venkatadheeraj Pichapati, Ananda Theertha Suresh, "Faster Algorithms for Testing under Conditional Sampling", *CoRR, abs/1504.04103*, 2015.

Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Venkatadheeraj Pichapati, Ananda Theertha Suresh, "Universal Compression of Power-Law Distributions", *CoRR, abs/1504.08070*, 2015.

Jayadev Acharya, Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Maximum Selection and Sorting with Adversarial Comparators and Application to Density Estimation", In Preparation, 2015.

Jayadev Acharya, Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "On the Computation and Verification Query complexity of Symmetric Functions", In preparation, 2015.

Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Competitive Test for Relatively Few-Modal Distributions", In Preparation, 2015.

ABSTRACT OF THE DISSERTATION

Function Computation with Optimal Number of Queries

by

Ashkan Jafarpour Koujahi

Doctor of Philosophy in Electrical Engineering (Communication Theory and Systems)

University of California, San Diego, 2015

Professor Alon Orlitsky, Chair

Evaluating a multivariate function is a crucial and ubiquitous task whose importance has inspired countless different computational models. These models are made by restricting the function's class, changing the computational elements, moderating the computation error probability, adding assumptions on input probabilities, assuming noisy queries, *etc.*

In most function-computation problems, the major cost of calculating the function is the price of samples or *queries*. The query meaning may change from one model to another but the aspiration to minimize the number of queries is unavoidable in all the models. In this dissertation, we consider and optimize the number of queries for two models in Part I and II.

In part I, we study maximum selection and sorting of n numbers using pairwise comparators that output the larger of their two inputs if the inputs are more than a given threshold apart, and output an adversarially-chosen input otherwise. We consider two adversarial models. A non-adaptive adversary that decides on the outcomes in advance based solely on the inputs, and an adaptive adversary that can decide on the outcome of each query depending on previous queries and outcomes. Against the non-adaptive adversary, we derive a maximum-selection algorithm that uses at most $2n$ comparisons in expectation, and a sorting algorithm that uses at most $2n \ln n$ comparisons in expectation. These numbers are within small constant factors from the best possible. Against the adaptive adversary, we propose a maximum-selection algorithm that uses $\Theta(n \log(1/\epsilon))$ comparisons to output a correct answer with probability at least $1 - \epsilon$. The existence of this algorithm affirmatively resolves an open problem in [5].

Our study was motivated by a density-estimation problem where, given samples from an unknown underlying distribution, we would like to find a distribution in a known class of n candidate distributions that is close to underlying distribution in ℓ_1 distance. Scheffe's algorithm [19] outputs a distribution at an ℓ_1 distance at most 9 times the minimum and runs in time $\Theta(n^2 \log n)$. Using maximum selection, we propose an algorithm with the same approximation guarantee but run time $\Theta(n \log n)$.

In Part II, we consider and propose a new method for function computation. The function's computation query complexity is the lowest expected number of queries required by any query order. Instead of computation, it is often easier to consider verification, where the value of the function is given and the queries aim to verify it. The lowest expected number of queries necessary is the function's verification query complexity. We show that for all symmetric functions of independent binary random variables, the computation and verification complexities coincide. This provides a simple method for finding the query complexity and optimal query order for computing many functions. We also show that after relaxing any of the symmetry, independence, or binary inputs restrictions, there are functions whose verification complexity is strictly lower than their computation complexity.

Part I

Ranking with Application to Density Estimation

Chapter 1

Introduction

1.1 General background and motivation

Maximum selection and sorting based on pairwise comparisons are crucial operations with wide-spread applications in many aspects of computing, sports, marketing, and decision making [35, 18]. Algorithms performing these operations using few comparisons are consequently taught in most algorithm-design courses and textbooks, *e.g.,* [15].

In a typical application, each of several items has a score, and using pairwise comparisons, we would like to find the largest item, or to sort the items, based on their scores. In the simplest model, every pairwise comparison yields the item with the higher score.

However, in many applications the comparison may be imperfect. For example, in sports, we would like to find the strongest team based on pairwise matches that don't always result in the stronger team winning, and chance may play a role. Or when deciding which product is most desirable based on consumers' choices between pairs of products, the responses depend on the individuals providing the answers. Given such imperfect comparisons, we would like to find algorithms that find or approximate the largest element or sort the elements using few comparisons.

Depending on the motivating application, several methods of quantifying the comparison inaccuracy have been proposed. The Bradley-Terry-Luce [12] model assumes that if two teams with scores x and y are compared, then the

team with score x is selected with probability $x/(x+y)$. Algorithms for ranking and estimating weights under this model were proposed, *e.g.,* in [31]. Another model assumes that the output of any comparator gets reversed with probability less than 1/2. Algorithms applying this model for maximum selection were proposed in [4] and for ranking in [23, 13].

We consider a third model were, unlike the previous models, the comparison outcome can be adversarial. If the two team capabilities differ by more than a threshold Δ, namely one team is significantly stronger than the other, the higher-capability team wins. However, if the team capabilities differ by at most Δ, then the winning team is chosen arbitrarily, and possibly even adversarially.

This model is motivated by several physical observations. The popular Weber-Fechner law [21] stipulates that if two physical stimuli are within a threshold Δ from each other, a human will not be able to distinguish between them, while if the stimuli are farther than Δ apart, then the observer would easily differentiate between them. In sports, a judge or a home-team advantage may, even adversarially, sway the outcome of a game between two similarly-capable teams, but a significantly more capable team will always beats its opponent.

1.2 Density estimation via pairwise comparisons

Our motivation for studying maximum selection derives from a density-estimation and distribution-learning problem. In a typical PAC-learning setup [36, 24], we are given a known distribution class $\mathcal{P}$ and samples from an unknown distribution $p_0 \in \mathcal{P}$, and would like to find, with high probability, a distribution $\hat{p} \in \mathcal{P}$ such that $\|\hat{p} - p_0\|_1 < \delta$.

One approach to solve this problem proceeds in two steps [19]. First, offline, construct a *δ-cover* of $\mathcal{P}$, a finite collection $\mathcal{P}_\delta \subseteq \mathcal{P}$ of distributions such that every distribution in $\mathcal{P}$ is within ℓ_1 distance $\leq \delta$ from some distribution in $\mathcal{P}_\delta$. Then, given samples from p_0, find with high probability a distribution in $\mathcal{P}_\delta$ whose ℓ_1 distance to p_0 is close to the least possible. These two steps result in a distribution whose ℓ_1 distance from p_0 is *close* to δ. Surprisingly, for several common distribution

classes, such as Gaussian mixtures, the number of samples required by this generic approach matches the information theoretically optimal sample complexity, up to logarithmic factors [17, 3].

A popular method for implementing the second step, namely to find a distribution in $\mathcal{P}_\delta$ with a small ℓ_1 distance from p_0, is via binary comparisons. The Scheffe Algorithm [34, 19] takes every pair of distributions in $\mathcal{P}_\delta$ and uses samples from p_0 to decide which of them is closer to p_0. It then declares the distribution that "wins" the most pairwise closeness comparisons to be closest to p_0. As shown in [19], with high probability, the Scheffe algorithm yields a distribution that is at most 9 times further from p_0 than the distributions with the lowest distance in $\mathcal{P}_\delta$ plus a diminishing additive term.

Since the Scheffe algorithms compares every pair of distributions in $\mathcal{P}_\delta$, its running time grows quadratically in $|\mathcal{P}_\delta|$. In Section 2.2, we use maximum-selection results to describe an alternative algorithm with the same error guarantees but running time that is linear in $|\mathcal{P}_\delta|$.

1.3 Definitions

Let $\mathcal{X} \stackrel{\text{def}}{=} \{x_1, \ldots, x_n\}$ be a multiset of n real numbers, possibly with repetitions. We would like to estimate the largest element $x^* \stackrel{\text{def}}{=} \max\{x_1, \ldots, x_n\}$ using noisy pairwise comparisons. The comparators we consider take two real inputs x and y, and if $|x - y| > \Delta$, output $\max\{x, y\}$, while if $|x - y| \le \Delta$, they output either x or y, possibly adversarially. Without loss of generality, we assume that $\Delta = 1$, hence[1],

$$\mathcal{C}(x, y) = \begin{cases} \max\{x, y\} & \text{if} \quad |x - y| > 1, \\ x \text{ or } y \text{ (adversarially)} & \text{if} \quad |x - y| \le 1. \end{cases} \tag{1.1}$$

[1]Note that in most applications, we compare objects, such as teams or probability distributions, that have scores, and the output of the comparator is not one of the two scores, but rather one of the two objects. For our algorithms to apply to these problems, we allow the comparator to determine the output based not only on the scores but also on the objects themselves, as long as the output is consistent with (1.1). For example, if two teams A and B have the same score, the comparator can consistently output one of the teams, say consistently team B.

We consider two types of adversarial comparators, *non-adaptive* and *adaptive*. A non-adaptive adversary that has complete prior knowledge of $\mathcal{X}$ and the algorithm, but must decide on the comparator output for all input pairs before the algorithm starts running. It cannot decide on a comparison result for an unseen pair based on previous queries and outcomes. Note that in the adversarial model we will never compare the same pair of inputs more than once. A natural way to represent such a comparator is via a complete directed graph with n nodes representing the inputs and if $\mathcal{C}(x, y) = x$, then there is a directed edge from x to y.

Let $\mathcal{C}_{\text{non}}(\mathcal{X})$ or simply $\mathcal{C}_{\text{non}}$ denote the set of all possible adversarial comparators under this model.

A stronger adaptive adversary not only has access to the algorithm and the inputs, but it is also allowed to adaptively decide the outcomes of the queries taking into account all the previous queries made by the algorithm. The adversaries under this model are denoted by $\mathcal{C}_{\text{adpt}}$.

For consistency with existing literature, we refer to each comparison as a *query*. The number of queries algorithm $\mathcal{A}$ makes for $\mathcal{X} = \{x_1, \ldots, x_n\}$ or the *query complexity* of $\mathcal{A}$ is denoted by $Q_n^{\mathcal{A}}$. Note that $Q_n^{\mathcal{A}}$ is a random variable since our algorithms are randomized. The *expected query complexity* of $\mathcal{A}$ for $\mathcal{X}$ is

$$q_n^{\mathcal{A}} \stackrel{\text{def}}{=} \mathbb{E}[Q_n^{\mathcal{A}}],$$

where the expectation is over the randomness of the algorithm.

The *maximum expected query complexity* of $\mathcal{A}$ against a non-adaptive adversary is

$$q_n^{\mathcal{A},\text{non}} \stackrel{\text{def}}{=} \max_{\mathcal{C} \in \mathcal{C}_{\text{non}}} \max_{\mathcal{X}} q_n^{\mathcal{A}}, \tag{1.2}$$

Similar to the non-adaptive adversarial model, the maximum expected query complexity of $\mathcal{A}$ for an adaptive adversary is

$$q_n^{\mathcal{A},\text{adpt}} \stackrel{\text{def}}{=} \max_{\mathcal{C} \in \mathcal{C}_{\text{adpt}}} \max_{\mathcal{X}} q_n^{\mathcal{A}}.$$

We evaluate an algorithm by measuring how close its output is to the maximum. A number x is a *t-approximation* of x^* if $x \geq x^* - t$. The *t-approximation*

error of an algorithm $\mathcal{A}$ for n inputs is

$$\mathcal{E}_n^{\mathcal{A}}(t) \stackrel{\text{def}}{=} \Pr(Y_{\mathcal{A}}(\mathcal{X}) < x^* - t),$$

the probability that $\mathcal{A}$'s output $Y_{\mathcal{A}}(\mathcal{X})$ is not a t-approximation of x^*. For an Algorithm $\mathcal{A}$, the maximum t-approximation error for the worst non-adaptive adversary is

$$\mathcal{E}_n^{\mathcal{A},\text{non}}(t) \stackrel{\text{def}}{=} \max_{\mathcal{C}\in\mathcal{C}_{\text{non}}} \max_{\mathcal{X}} \mathcal{E}_n^{\mathcal{A}}(t),$$

and similarly for the adaptive adversary,

$$\mathcal{E}_n^{\mathcal{A},\text{adpt}}(t) \stackrel{\text{def}}{=} \max_{\mathcal{C}\in\mathcal{C}_{\text{adpt}}} \max_{\mathcal{X}} \mathcal{E}_n^{\mathcal{A}}(t).$$

For the non-adaptive adversary, the minimum t-approximation error of any algorithm is

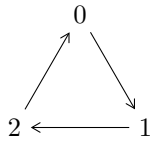
$$\mathcal{E}_n^{\text{non}}(t) \stackrel{\text{def}}{=} \min_{\mathcal{A}} \mathcal{E}_n^{\mathcal{A},\text{non}}(t),$$

and similarly for the adaptive adversary,

$$\mathcal{E}_n^{\text{adpt}}(t) \stackrel{\text{def}}{=} \min_{\mathcal{A}} \mathcal{E}_n^{\mathcal{A},\text{adpt}}(t).$$

Clearly, $\mathcal{E}_n^{\text{adpt}}(t) \geq \mathcal{E}_n^{\text{non}}(t)$ for all t. The following simple example shows that $\mathcal{E}_3^{\text{non}}(t) \geq \frac{1}{3}$ for all $t < 2$.

Example 1. $\mathcal{E}_3^{\text{non}}(t) \geq \frac{1}{3}$ *for all* $t < 2$. *Consider* $\mathcal{X} = \{0, 1, 2\}$ *and the following comparators.*



By symmetry, no algorithm can differentiate between the three inputs, hence any algorithm will output 0 with probability 1/3.

By replicating each value for $\frac{n}{3}$ times, we can generalize this example to larger n. In Section 1.5.1 we show a stronger result, namely for all $t < 2$, we cannot obtain a t-approximation with probability greater than $1/2 - 1/(2n)$.

We are interested in designing algorithms that have small query complexity, and small t-approximation error for small values of t.

1.4 Previous and new results

In Section 1.5.1 we lower bound $\mathcal{E}_n^{\text{non}}(t)$ as a function of t. We show that for all $t < 1$ and odd n, $\mathcal{E}_n^{\text{non}}(t) \geq 1 - 1/n$, namely for some $\mathcal{X}$, approximating the maximum to within less than one is equivalent to guessing a random x_i as the maximum. In Lemma 3, we modify Example 1 and show that for all $t < 2$ and odd n, any algorithm has t-approximation error close to $1/2$ for some input.

We propose a number of algorithms to approximate the maximum. These algorithms have different performance guarantees in terms of the probability of error, approximation factor, and query complexity.

We first consider two simple algorithms, the complete tournament, COMPL, and the sequential, SEQ. COMPL compares all pairs of inputs, and outputs the one with the maximum number of wins. We show the simple result that COMPL achieves a 2-approximation of x^* with zero error probability. We then consider the sequential algorithm SEQ that compares a pair of inputs and discards the loser and compares the winner with a new input. We show that even under randomization, with high probability, this algorithm cannot provide a constant approximation.

We then modify knock-out to derive a new algorithm KO-MOD that achieves a 3-approximation with error probability at most ϵ against the adaptive adversary. We note that [5] proposed a different algorithm with similar performance guarantees.

Motivated by quick-sort, we propose a quick-select algorithm Q-SELECT that outputs a 2-approximation with zero error probability and has the expected query complexity of at most $2n$ against the non-adaptive adversary. However, as we see in Example 14, the algorithm may require $\binom{n}{2}$ queries against the adaptive adversary.

This leaves the question of finding a 2-approximation of x^* with $\mathcal{O}(n)$ queries against the adaptive adversary. In fact, [5] asked whether there exists an $\mathcal{O}(n)$-query randomized algorithm that outputs a 2-approximation of the maximum. We solve this problem by designing an algorithm COMB that combines quick-select and knock-out. We prove that COMB outputs a 2-approximation with probability of error at most ϵ, using $\mathcal{O}(n \log \frac{1}{\epsilon})$ queries. We summarize the results

in Table 1.1.

Table 1.1: Maximum selection algorithms

algorithm	notation	approximation	$q_n^{\mathcal{A},\text{non}}$	$q_n^{\mathcal{A},\text{adpt}}$
complete tournament	COMPL	$\mathcal{E}_n^{\text{COMPL,adpt}}(2) = 0$	$\binom{n}{2}$	
deterministic upper bound [5]	-	$\mathcal{E}_n^{\mathcal{A},\text{adpt}}(2) = 0$	$\Theta(n^{\frac{3}{2}})$	
deterministic lower bound [5]	-	$\mathcal{E}_n^{\mathcal{A},\text{adpt}}(2) = 0$	-	$\Omega(n^{\frac{4}{3}})$
sequential	SEQ	$\mathcal{E}_n^{\text{SEQ,non}}\left(\frac{\log n}{\log\log n} - 1\right) \to 1$	$n-1$	
modified knock-out	KO-MOD	$\mathcal{E}_n^{\text{KO-MOD,adpt}}(3) < \epsilon$	$n + \frac{1}{2}\log^4 n \left\lceil \frac{1}{\epsilon}\ln\frac{1}{\epsilon}\right\rceil^2$	
quick-select	Q-SELECT	$\mathcal{E}_n^{\text{Q-SELECT,adpt}}(2) = 0$	$2n$	$\binom{n}{2}$
Knock-out and quick-select	COMB	$\mathcal{E}_n^{\text{COMB,adpt}}(2) < \epsilon$	$\mathcal{O}\left(n\log\frac{1}{\epsilon}\right)$	

We note that while we focus on randomized algorithms, [5] also studied the best possible trade-offs for deterministic algorithms. They designed a deterministic algorithm for 2-approximation of the maximum using only $\mathcal{O}(n^{3/2})$ queries. Moreover, they prove that no deterministic algorithm with fewer than $\Omega(n^{4/3})$ queries can output a 2-approximation of x^* for the adaptive adversarial model.

1.5 Simple results

In Lemmas 2 and 3 we prove lower bounds on the error probability of any algorithms that provide a t-approximation of x^* for $t < 1$ and $t < 2$ respectively. We then consider two straight-forward algorithms for finding the maximum. One is the complete tournament, where each pair of inputs is compared and the other sequentially compares inputs, discarding the loser at each stage.

1.5.1 Lower bounds

We show the following two lower bounds.

- For all $0 \le t < 1$ and odd n, $\mathcal{E}_n^{\text{non}}(t) \ge 1 - \frac{1}{n}$.
- For all $1 \le t < 2$ and odd n, $\mathcal{E}_n^{\text{non}}(t) \ge \frac{1}{2} - \frac{1}{2n}$.

These lower bounds can be applied to even values of n by adding an extra input which is smaller than all others and loses to everyone. Note that Example 1 showed that $\forall t < 2$, $\mathcal{E}_3^{\text{non}}(t) \ge \frac{1}{3}$. Lemma 3 improves the error probability to about 1/2.

Lemma 2. *For all $0 \le t < 1$ and odd n, $\mathcal{E}_n^{\text{non}}(t) \ge 1 - \frac{1}{n}$.*

Proof. Consider $0 \le c_1 < c_2 < \ldots < c_{n-1} < 1 - t$. Let $(x_1, x_2, \ldots, x_n)$ be an unknown permutation of $(1, c_1, c_2, \ldots, c_{n-1})$. Suppose we consider an adversary that ensures each input wins exactly $(n-1)/2$ times. An example is shown in Figure 1.1 for $n = 5$.

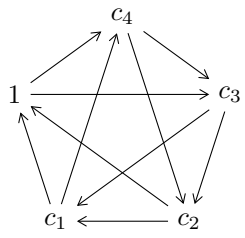


Figure 1.1: Tournament for Lemma 2 when $n = 5$

We want a lower bound on the performance of any randomized algorithm. By Yao's principle, we consider only deterministic algorithms over a uniformly chosen permutation of the inputs, namely only one of the coordinates is 1, and remaining are less than $1 - t$. In this case, if we fix any comparison graph (as in the figure above), and permute the inputs, the algorithm cannot distinguish between 1 and c's, and outputs one of the c's with probability $1 - 1/n$. □

Lemma 3. *For all $1 \le t < 2$ and odd n, $\mathcal{E}_n^{\text{non}}(t) \ge \frac{1}{2} - \frac{1}{2n}$.*

Proof. Consider $0 \leq c_1 < c_2 < \ldots < c_{n-1} < 2 - t$. Let m be $(n-1)/2$. Let $(x_1, x_2, \ldots, x_n)$ be an unknown permutation of $(2, 1 + c_1, 1 + c_2, \ldots, 1 + c_m, c_{m+1}, c_{m+2}, \ldots, c_{2m})$. Suppose the adversary ensures that 2 loses against all the inputs above 1 and indeed all inputs have exactly $(n-1)/2$ wins. An example is shown in Figure 1.2.

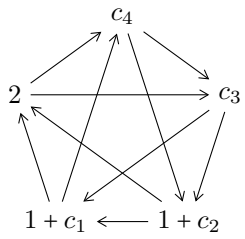


Figure 1.2: Tournament for Lemma 3 when $n = 5$

Similar to Lemma 2, the inputs are all identical to the algorithm and therefore, the algorithm outputs one of the c's with probability $\frac{m}{n} = \frac{1}{2} - \frac{1}{2n}$. □

1.5.2 Two elementary algorithms

In this section, we analyze two familiar maximum selection algorithms, the complete tournament and sequential algorithms. We discuss their strength and weakness and show that there is a trade-off between query complexity and approximation guarantee of these two algorithms. Another well-known algorithm for maximum selection is knock-out algorithm and we discuss a variation of it in Section 2.1.1.

Complete tournament (round-robin)

As its name evinces, a complete tournament involves a match between every pair of teams. Using this metaphor to competitions, we compare all the $\binom{n}{2}$ input pairs, and the input winning most times is declared as the output. If two or more inputs end up with the highest number of wins, any of them can be declared as the output. This algorithm is formally stated in COMPL and was formerly studied in the context of density estimation in [19].

input: $\mathcal{X}$

compare all input pairs in $\mathcal{X}$

count the number of times each input wins

output: an input with the maximum number of wins

Algorithm 1: COMPL - Complete tournament

We bound the performance of the algorithm as follows.

Lemma 4. $q_n^{\text{COMPL,adpt}} = \binom{n}{2}$ *and* $\mathcal{E}_n^{\text{COMPL,adpt}}(2) = 0$.

Proof. The number of queries is clearly $\binom{n}{2}$. To show $\mathcal{E}_n^{\text{COMPL,adpt}}(2) = 0$, note that if $y < x^* - 2$ then if y beats some z, then $z \leq y + 1 < x^* - 1$, hence x^* also beats z. Moreover, x^* also beats y. Therefore, x^* beats more inputs than y, so y cannot be the output of the algorithm. It follows that the input with the maximum wins is a 2-approximation of x^*. □

COMPL is deterministic and after $\binom{n}{2}$ queries outputs a 2-approximation of x^*. If the comparators are noiseless, we can simply compare the inputs sequentially, discarding the loser at each step, thus requiring only $n - 1$ comparisons. This evokes the hope of finding a deterministic algorithm that requires a linear number of comparisons and outputs a 2-approximation of x^*. As mentioned earlier, [5] showed it is not achievable. They proved that any deterministic 2-approximation algorithm requires $\Omega(n^{4/3})$ queries. They also showed a strictly superlinear lower bound on any deterministic constant-approximation algorithm. On the other hand, they designed a deterministic 2-approximation algorithm using $\mathcal{O}(n^{3/2})$ queries.

Therefore, we consider only randomized algorithms with the hope of finding a 2-approximation algorithm with $\mathcal{O}(n)$ queries. We next consider the sequential algorithm in the adversarial setup and show that it cannot achieve a constant-approximation of x^*.

Sequential selection

Sequential selection first compares a random pair of inputs, and at each successive step, compares the winner of the last comparison with a randomly chosen *new* input. It outputs the final remaining input.

input: $\mathcal{X}$
  choose a random $y \in \mathcal{X}$ and remove it from $\mathcal{X}$
  while $\mathcal{X}$ is not empty
    choose a random $x \in \mathcal{X}$ and remove it from $\mathcal{X}$
    $y \leftarrow \mathcal{C}(x, y)$
  end while
output: y

Algorithm 2: SEQ - Sequential selection

This algorithm uses $n-1$ queries. We give an example of inputs where this algorithm, even against the non-adaptive adversary, with high probability cannot output a constant-approximation of x^*.

Lemma 5. *Let* $s = \frac{\log n}{\log\log n}$. *For all* $t < s$, $\mathcal{E}_n^{\mathrm{SEQ,non}}(t) \geq 1 - \frac{1}{\log\log n}$.

Proof. Let s, $\log n$, and $\log\log n$ be integers and let

$$x_i = \begin{cases} s & \text{for } i = 1, \\ s-1 & \text{for } i = 2, \ldots, \log n, \\ s-2 & \text{for } i = \log n + 1, \ldots, \log^2 n, \\ \vdots & \\ m & \text{for } i = \log^{s-m-1} n + 1, \ldots, \log^{s-m} n, \\ \vdots & \\ 0 & \text{for } i = \frac{n}{\log n} + 1, \ldots, n. \end{cases}$$

Consider the following non-adaptive adversarial comparator,

$$\mathcal{C}(x_i, x_j) = \begin{cases} \max\{x_i, x_j\} & \text{if } |x_i - x_j| > 1, \\ \min\{x_i, x_j\} & \text{if } |x_i - x_j| \leq 1. \end{cases}$$

Observe that SEQ sequentially compares a random permutation of inputs. Let L_j be the random location in the permutation where input with value j appears last. The following two observations help to prove the lemma.

Observation 1. *Input* j *appears at least* $(\log n - 1)$ *times more than input* $j+1$.

Observation 2. *For the defined adversarial comparator, if* $L_0 > L_1 > \ldots > L_s$ *then* SEQ *outputs* 0.

As a consequence of Observation 1, in the random permutation of inputs, $L_j > L_{j+1}$ with probability at least $1 - \frac{1}{\log n}$. By the union bound, $L_0 > L_1 > \ldots > L_s$ with probability at least,

$$1 - \frac{s}{\log n} = 1 - \frac{1}{\log \log n}.$$

By applying Observation 2, SEQ outputs 0 with probability at least $1 - \frac{1}{\log \log n}$. □

Acknowledgements

Chapter 1 is adapted from Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Sorting with adversarial comparators and application to density estimation", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 2014; and Jayadev Acharya, Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Maximum Selection and Sorting with Adversarial Comparators and Application to Density Estimation", In Preparation, 2015; The dissertation author was the primary researcher and author of these papers.

Chapter 2

Maximum Selection with Adversarial Noise

2.1 Algorithms

In the previous chapter we saw that the complete tournament, COMPL, always outputs a 2-approximation, but has quadratic query complexity, while the sequential algorithm, SEQ, has linear query complexity but a poor approximation guarantee. A natural question to ask is whether the benefits of these two algorithms can be combined to derive bounded error with linear query complexity. In this section, we propose algorithms with linear query complexity and approximation guarantees that compete with the best possible, *i.e.,* 2-approximation of x^*.

We now propose three algorithms, with varying performance guarantees:

- **Modified knock-out**, described in Section 2.1.1, has linear query complexity and with high probability outputs a 3-approximation of x^* against both the adaptive and non-adaptive adversaries.
- **Quick-select**, described in Section 2.1.2, has linear expected query complexity and always outputs a 2-approximation of x^* against the non-adaptive adversary.
- **Knock-out and quick-select combination**, described in Section 2.1.3, has

linear query complexity, and with high probability outputs a 2-approximation of x^* against the adaptive adversary.

We now go over these in detail.

2.1.1 Modified knock-out

The knock-out algorithm derives its name from knock-out competitions where the tournament is divided into $\log n$ successive rounds. In round i, there are $\frac{n}{2^{i-1}}$ inputs. These inputs are paired at random, and the winners advance to the next round. The winner after $\log n$ rounds is declared as the maximum.

At each round of knock-out algorithm, the largest remaining element decreases by at most one. Therefore, knock-out algorithm finds at least $\log n$-approximation of x^*. Analyzing the precise approximation error of knock-out algorithm appears difficult. However, simulations suggest that for any large n, the set consisting of $0.2 \cdot n$ 0's, $\alpha \cdot n$ 1's, $(0.7 - \alpha) \cdot n$ 2's, $0.1 \cdot n$ 3's, and a single 4, where $0 < \alpha < 0.7$ is an appropriately chosen parameter, the knock-out algorithm is not able to find 4-approximation of x^* with positive constant probability.

The problem with knock-out algorithm is that if at any of the $\log n$ rounds, many elements are within 1 from the largest element in that round, there is a fair chance that the largest element will be eliminated. If this elimination happens in several rounds, we will end up with a number measurably smaller than x^*. To avoid this problem, we select a specified number of elements at each round and save them for the very end, thereby ensuring that in every round, if the largest element is eliminated, then an element within 1 from this element has been saved. We then perform a complete tournament on the saved elements.

More precisely, to ensure that a number close to x^* remains till the end, at each round, we would like to keep an input within 1-approximation of x^* with high probability. Our proposed algorithm to achieve this goal is modification of knock-out algorithm KO-MOD.

We show that KO-MOD has 3-approximation error less than ϵ. Let $n_1 \stackrel{\text{def}}{=} \left\lceil \frac{1}{\epsilon} \ln \frac{1}{\epsilon} \log n \right\rceil$. At each round, KO-MOD retains n_1 inputs in an additional multiset $\mathcal{Y}$

input: $\mathcal{X}$

pair the inputs of $\mathcal{X}$ randomly, let $\mathcal{X}'$ be the winners

output: $\mathcal{X}'$

Algorithm 3: KO-SUB - Subroutine for KO-MOD and COMB

input: $\mathcal{X}, \epsilon$

$\mathcal{Y} = \varnothing$, $n_1 = \left\lceil \frac{1}{\epsilon} \ln \frac{1}{\epsilon} \log n \right\rceil$

while $|\mathcal{X}| > n_1$

copy n_1 of random inputs from $\mathcal{X}$ to $\mathcal{Y}$

$\mathcal{X} \leftarrow$ KO-SUB$(\mathcal{X})$

end while

output: COMPL$(\mathcal{X} \bigcup \mathcal{Y})$

Algorithm 4: KO-MOD - Modified knock-out algorithm

and runs the knock-out subroutine KO-SUB on the multiset $\mathcal{X}$. When the number of inputs in $\mathcal{X}$ is $\leq n_1$, then the algorithm runs the complete tournament on $\mathcal{X} \bigcup \mathcal{Y}$.

Since $|\mathcal{X} \bigcup \mathcal{Y}|$ is poly-logarithmic in n, the extra query complexity of the complete tournament will be poly-logarithmic in n. For the approximation guarantee, we show that at each round of the algorithm, an input within 1-approximation of x^* remains in the set $\mathcal{X} \bigcup \mathcal{Y}$ with probability greater than $1 - \epsilon / \log n$. Since there are at most $\log n$ rounds, by union bound, with probability greater than $1 - \epsilon$, an input within 1-approximation of x^* remains in $\mathcal{X} \bigcup \mathcal{Y}$. Observe that the complete tournament outputs a 2-approximation of its maximum input. As a result, KO-MOD outputs a 3-approximation of x^* with probability greater than $1 - \epsilon$.

Theorem 6. *For* $n_1 \geq 4$, $q_n^{\text{KO-MOD,adpt}} < n + \frac{1}{2} \log^4 n \left\lceil \frac{1}{\epsilon} \ln \frac{1}{\epsilon} \right\rceil^2$ *and* $\mathcal{E}_n^{\text{KO-MOD,adpt}}(3) < \epsilon$.

Proof. The number of comparisons made by KO-SUB is at most $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots < n$. Observe that KO-SUB is called $m \stackrel{\text{def}}{=} \left\lceil \log \frac{n}{n_1} \right\rceil$ times. Let $\mathcal{X}_i$ be the multiset $\mathcal{X}$ at the start of ith call to KO-SUB. Let $\mathcal{X}_{m+1}$ and $\mathcal{Y}_{m+1}$ be the multisets $\mathcal{X}$ and $\mathcal{Y}$ right before the call to COMPL.

The number of inputs in $\mathcal{X}_{m+1} \bigcup \mathcal{Y}_{m+1}$ is

$$\begin{aligned}
|\mathcal{X}_{m+1} \bigcup \mathcal{Y}_{m+1}| &\le |\mathcal{X}_{m+1}| + |\mathcal{Y}_{m+1}| \\
&\le n_1 + \sum_{i=1}^{m} (|\mathcal{Y}_{i+1}| - |\mathcal{Y}_i|) \\
&\le n_1 + m n_1 \\
&= \left\lceil \log \frac{n}{n_1} + 1 \right\rceil \left\lceil \frac{1}{\epsilon} \ln \frac{1}{\epsilon} \log n \right\rceil \\
&\le \log^2 n \left\lceil \frac{1}{\epsilon} \ln \frac{1}{\epsilon} \right\rceil,
\end{aligned}$$

where the final inequality follows as $n_1 \ge 4$. As a result, the complete tournament uses less than $\frac{1}{2} \log^4 n \left\lceil \frac{1}{\epsilon} \ln \frac{1}{\epsilon} \right\rceil^2$ queries. Therefore, the total number of queries is less than $n + \frac{1}{2} \log^4 n \left\lceil \frac{1}{\epsilon} \ln \frac{1}{\epsilon} \right\rceil^2$.

We now bound the error of KO-MOD. Let

$$\mathcal{X}^* \stackrel{\text{def}}{=} \{x \in \mathcal{X} : x \ge x^* - 1\},$$

the multiset of inputs that are at least $x^* - 1$. For $i = 1, \ldots, m+1$, let $\mathcal{X}_i^* = \mathcal{X}_i \bigcap \mathcal{X}^*$ and $\mathcal{Y}_{m+1}^* = \mathcal{Y}_{m+1} \bigcap \mathcal{X}^*$.

Let $\alpha_i \stackrel{\text{def}}{=} \frac{|\mathcal{X}_i^*|}{|\mathcal{X}_i|}$ and $\alpha = \max\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$. We show that with high probability, $|\mathcal{X}_{m+1}^* \bigcup \mathcal{Y}_{m+1}^*| \ge 1$, *i.e.,* some input in $\mathcal{X}_{m+1} \bigcup \mathcal{Y}_{m+1}$ belongs to $\mathcal{X}^*$. In particular, we show that for large α, at least one input from $\mathcal{X}^*$ will be in $\mathcal{Y}_{m+1}$, and if α is small then x^* remains in $\mathcal{X}_{m+1}$.

Observe that,

$$\begin{aligned}
\Pr(\mathcal{X}_{m+1}^* = \varnothing) &\le \Pr(x^* \notin \mathcal{X}_{m+1}^*) \\
&= \sum_{i=1}^{m} \Pr(x^* \notin \mathcal{X}_{i+1}^* | x^* \in \mathcal{X}_i) \cdot \Pr(x^* \in \mathcal{X}_i) \\
&\le \sum_{i=1}^{m} \Pr(x^* \notin \mathcal{X}_{i+1}^* | x^* \in \mathcal{X}_i) \\
&\stackrel{(a)}{\le} \sum_{i=1}^{m} \frac{|\mathcal{X}_i^*| - 1}{|\mathcal{X}_i| - 1} \\
&\le \sum_{i=1}^{m} \alpha_i \\
&\le \alpha m,
\end{aligned}$$

where (a) follows from the fact that KO-SUB randomly pairs the inputs and only inputs inside $\mathcal{X}_i^* \backslash \{x^*\}$ may eliminate x^* at round i.

At round i, the probability that an input in $\mathcal{X}^*$ is not picked up in $\mathcal{Y}$ is

$$\left(1 - \frac{|\mathcal{X}_i^*|}{|\mathcal{X}_i|}\right)^{n_1} = (1-\alpha_i)^{n_1}.$$

Therefore,

$$\Pr(\mathcal{Y}_{m+1}^* = \varnothing) \le \min_i (1-\alpha_i)^{n_1} \le (1-\alpha)^{n_1}.$$

As a result,

$$\begin{aligned}
\Pr(\mathcal{X}_{m+1}^* \bigcup \mathcal{Y}_{m+1}^* = \varnothing) &= \Pr(\mathcal{X}_{m+1}^* = \varnothing \ \wedge \ \mathcal{Y}_{m+1}^* = \varnothing) \\
&\le \max_\alpha \min\{\Pr(\mathcal{X}_{m+1}^* = \varnothing), \Pr(\mathcal{Y}_{m+1}^* = \varnothing)\} \\
&\le \max_\alpha \min\{\alpha m, (1-\alpha)^{n_1}\} \\
&\overset{(a)}{\le} \max\{\alpha m, (1-\alpha)^{n_1}\}\big|_{\alpha=\frac{\epsilon}{\log n}} \\
&= \max\left\{\frac{\epsilon m}{\log n}, \left(1 - \frac{\epsilon}{\log n}\right)^{n_1}\right\} \\
&\overset{(b)}{<} \epsilon,
\end{aligned}$$

where (a) follows since the first term increases and the second term decreases with α, and (b) follows since $m \le \log n$ and $n_1 = \left\lceil \frac{1}{\epsilon} \ln \frac{1}{\epsilon} \log n \right\rceil$.

So far we have shown that with probability $1-\epsilon$, there exists an input inside $\mathcal{X}_{m+1} \bigcup \mathcal{Y}_{m+1}$ that is 1-approximation of x^*. Lemma 4 showed that COMPL outputs a 2-approximation of its maximum input. Consequently, with probability $1-\epsilon$, KO-MOD outputs a 3-approximation of x^*. □

The next example shows that the modified knock-out algorithm cannot achieve better than 3-approximation of x^* with constant probability.

Example 7. *Suppose $n-2$ is multiple of 3 and n is a large number. Let $\mathcal{X}$ be a random permutation of*

$$\{3, \underbrace{2, 2, \ldots, 2}_{\frac{n-2}{3}}, \underbrace{1, 1, \ldots, 1}_{\frac{n-2}{3}}, \underbrace{0, 0, \ldots, 0}_{\frac{n-2}{3}}, 0^*\}.$$

The multiset of inputs consists of an input with value zero but specified with 0^**, since this input is going to behave differently from other* 0*s. Let the adversarial comparator be such that all* 0*s, except* 0^**, and all* 2*s will lose to all* 1*s, and* 3 *loses to all* 2*s. By the properties of comparator it is obvious that any* 2 *will defeat all zeros, including* 0^**. In order to prove our main claim, we make some arguments and show that each of them happens with high probability.*

- *Pr(input with value* 3 *is not present in the final multiset)*$> \frac{3}{10}$.
- *Pr(input* 0^* *is present in the final multiset)*$> \frac{1}{3}$.
- *With high probability, the fraction of* 1*s in the final multiset is high and the fraction of* 0*s and* 2*s are very small.*

Before proving each argument we show why all the above statements are sufficient to prove our claim. Consider the final multiset; with high probability it mainly consists of 1*s and there are small number of* 0*s and* 2*s. Moreover, with probability greater than* $\frac{1}{3} \times \frac{3}{10}$*, input with value* 3 *has been removed before reaching the final multiset and* 0^* *has survived to reach the final multiset. Therefore, if we run Algorithm* COMPL *on the final multiset, the input* 0^* *will have the most number of wins and declared as the output of the algorithm. Hence for all* $t < 3$*, we have* $\mathcal{E}_n^{\text{KO-MOD,non}}(t) >$ *constant. Note that we did not try to optimize this constant.*

Now we show why each of the arguments above are true. Note that all the discussions made below is in expected value. However, the concentration bounds for all these claims are straightforward but not discussed because it is out of the scope of this study. Also we assume that n *is sufficiently large.*

Lemma 8. *With high probability, the fraction of* 1*s in the final multiset is close to 1 and the fraction of* 0*s and* 2*s are very small.*

Proof. We calculate the expected number of 0s, 1s, and 2s at each step. Let $f_i(j)$ be the fraction of j's at the end of step i. After each step, we lose an input with value 1 if and only if they are paired with each other. So we have the following recursion,

$$f_{i+1}(1) = 2 \cdot f_i(1) \left(\tfrac{f_i(1)}{2} + 1 - f_i(1) \right),$$

where the factor 2 on the RHS of the recursion above is due to the fact that at each step we are reducing the number of inputs to half. Starting with $f_0(1) = 1/3$, we get the set of values $\{1/3, 5/9, 65/81, 6305/6561 \sim 0.96, ...\}$ for $f_i(1)$s. We can see that the ratio is approaching 1 very fast. More precisely, the fraction of 0s is decreasing quadratically since their only chance of survival is to get paired among themselves. As a result, after a couple of steps, the fraction of zeros is extremely small, and henceforth the only chance of survival for 2s becomes getting paired among themselves and also their fraction is going to decrease quadratically afterwards. As a result, more samples of 1s will be in the final $\mathcal{Y}$ with high probability. □

Lemma 9. *Pr(input with value* 3 *is not present in the final multiset)*$> \frac{3}{10}$.

Proof. The input with value 3 is going to be removed when it is compared against one of the 2s. There is a slight chance of surviving for it if it is chosen randomly for being in the output. Thus the probability of input 3 being removed from the multiset in the first round is

$$\Pr(\text{input 3 is being removed in the first round}) = \frac{n-2}{3n}\left(1 - \frac{n_1}{n}\right) > \frac{3}{10},$$

where $n_1 = \left\lceil \frac{1}{\epsilon} \ln \frac{1}{\epsilon} \log n \right\rceil$. □

Lemma 10. *Pr(input* 0^* *is present in the final multiset)*$> \frac{1}{3}$.

Proof. Similar to the argument made in the proof of Lemma 8, we have the following recursion for $f_i(2)$.

$$f_{i+1}(2) = 2 \cdot f_i(2)\left(\tfrac{f_i(2)}{2} + 1 - f_i(2) - f_i(1)\right)$$

Thus we have $f_0(2) = 1/3$, $f_1(2) = 1/3$, $f_2(2) = 5/27$, $f_3(2) = 85/2187$. As we stated in the proof of Lemma 8, the expected fraction of 2s is decreasing quadratically and

$$\Pr(0^*\text{surviving}) = (1 - \tfrac{1}{3})(1 - \tfrac{1}{3})(1 - \tfrac{5}{27})(1 - \tfrac{85}{2187})\cdots > \frac{1}{3},$$

proving the lemma. □

2.1.2 Quick-select

Motivated by quick-sort, we propose a quick-select algorithm Q-SELECT that at each stage compares all the inputs with a random pivot to provide stronger performance guarantees against the non-adaptive adversary.

input: $\mathcal{X}$
  pick a pivot $x_p \in \mathcal{X}$ at random
  compare x_p with all other inputs in $\mathcal{X}$
  let $\mathcal{Y} \subset \mathcal{X} \backslash \{x_p\}$ be the multiset of inputs that beat x_p
output: if $\mathcal{Y} \neq \varnothing$ output $\mathcal{Y}$ otherwise output $\{x_p\}$

Algorithm 5: QS-SUB - Subroutine for Q-SELECT and COMB

input: $\mathcal{X}$
  while $|\mathcal{X}| > 1$
    $\mathcal{X} \leftarrow$ QS-SUB$(\mathcal{X})$
  end while
output: the unique input in $\mathcal{X}$

Algorithm 6: Q-SELECT - Quick-select

We first show that Q-SELECT provides a 2-approximation with no error against both the adaptive and non-adaptive adversary. To show this result, observe that x^* will only be eliminated if a 1-approximation of x^* is chosen as pivot and therefore, only inputs that are 2-approximation of x^* will survive.

Lemma 11. $\mathcal{E}_n^{\text{Q-SELECT,adpt}}(2) = 0$.

Proof. If the output is x^*, the lemma holds. Otherwise, x^* is discarded when it was chosen as a pivot or compared with a pivot. Let x_p be the pivot when x^* is discarded, hence $x_p \geq x^* - 1$. By the algorithm's definition, all the survived elements are $\geq x_p - 1 \geq x^* - 2$. □

We now show that the expected query complexity of quick-select against a non-adaptive adversary is at most $2n$. This result is similar to the case when the

comparisons are noiseless. To obtain high probability results, we prove a super-exponential decay of the number of queries. We finally consider an example for which Q-SELECT requires $\binom{n}{2}$ queries against the adaptive adversary.

Lemma 12. $q_n^{\text{Q-SELECT,non}} < 2n$.

Proof. Recall that the non-adaptive adversary can be modeled as a complete directed graph where each node is an input and there is an edge from x to y if $\mathcal{C}(x, y) = x$. Let $\text{in}(x)$ be the in-degree of x.

At round i the algorithm chooses an x_p at random and compares it to all remaining inputs. By keeping the winners, $\max\{\text{in}(x_p), 1\}$ inputs will remain for the next round. As a result, we have the following recursion for non-adaptive adversaries,

$$\begin{aligned} q_n^{\text{Q-SELECT}} &= \mathbb{E}\left[Q_n^{\text{Q-SELECT}}\right] \\ &= n - 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\left[Q_{\text{in}(x_i)}^{\text{Q-SELECT}}\right] \\ &= n - 1 + \frac{1}{n}\sum_{i=1}^{n} q_{\text{in}(x_i)}^{\text{Q-SELECT}}. \end{aligned}$$

By (1.2),

$$\begin{aligned} q_n^{\text{Q-SELECT,non}} &= \max_{\mathcal{C}\in\mathcal{C}_{\text{non}}} \max_{\mathcal{X}} q_n^{\text{Q-SELECT}} \\ &= \max_{\mathcal{C}\in\mathcal{C}_{\text{non}}} \max_{\mathcal{X}} \left[n - 1 + \frac{1}{n}\sum_{i=1}^{n} q_{\text{in}(x_i)}^{\text{Q-SELECT}} \right] \\ &\le n - 1 + \frac{1}{n}\sum_{i=1}^{n} \max_{\mathcal{C}\in\mathcal{C}_{\text{non}}} \max_{\mathcal{X}} q_{\text{in}(x_i)}^{\text{Q-SELECT}} \\ &= n - 1 + \frac{1}{n}\sum_{i=1}^{n} q_{\text{in}(x_i)}^{\text{Q-SELECT,non}}, \end{aligned} \tag{2.1}$$

where the inequality follows as maximum of sums is at most sum of maximums. We prove by induction that $q_n^{\text{Q-SELECT,non}} \le 2(n-1)$. It holds for $n = 1$. Suppose it

holds for all $n' < n$, then,

$$\begin{aligned} q_n^{\text{Q-SELECT,non}} &\le n - 1 + \frac{1}{n}\sum_{i=1}^{n} q_{\text{in}(x_i)}^{\text{Q-SELECT,non}} \\ &\le n - 1 + \frac{1}{n}\sum_{i=1}^{n} 2 \cdot \text{in}(x_i) \\ &= n - 1 + \frac{n(n-1)}{n} \\ &\le 2(n-1), \end{aligned}$$

where the equality follows since the in-degrees sum to $\frac{n(n-1)}{2}$. □

Lemma 12 shows that $q_n^{\text{Q-SELECT,non}} < 2n$. Next, we show a naive concentration bound for the query complexity of Q-SELECT. By Markov's inequality, for a non-adaptive adversary,

$$\Pr(Q_n^{\text{Q-SELECT}} > 4n) \le \frac{1}{2}.$$

Let k be an integer multiple of 4. Now suppose we run Q-SELECT, allowing kn queries. At each $4n$ queries, the Q-SELECT ends with probability $\ge \frac{1}{2}$. Therefore,

$$\Pr(Q_n^{\text{Q-SELECT}} > kn) \le 2^{-\frac{k}{4}}.$$

This naive bound is exponential in k. In the following lemma, for the non-adaptive adversary, we establish a tighter super-exponential concentration bound.

Lemma 13. *Let $k' = \max\{e, k/2\}$. For the non-adaptive adversary, $Pr(Q_n^{\text{Q-SELECT}} > kn) \le e^{-(k-k')\ln k'}$.*

Proof. Abbreviate $Q_n^{\text{Q-SELECT}}$ by Q_n. As in the Chernoff bound proof, for all $\lambda > 0$,

$$\Pr(Q_n > kn) \le \frac{\mathbb{E}[e^{\lambda Q_n}]}{e^{k\lambda n}}. \tag{2.2}$$

Let $\lambda = \frac{1}{n}\ln k'$ and $\Phi(i) \stackrel{\text{def}}{=} \mathbb{E}[e^{\lambda Q_i}]$. We prove by induction that $\Phi(i) \le e^{k'\lambda i}$. The induction holds for $i = 0$. Similar to (2.1), we have the following recursion for $\Phi(n)$,

$$\begin{aligned} \Phi(n) &\le \frac{e^{\lambda(n-1)}}{n}\sum_{j=1}^{n}\Phi(\text{in}(x_j)) \\ &\le \frac{e^{\lambda n}}{n}\sum_{j=1}^{n}\Phi(\text{in}(x_j)). \end{aligned}$$

Since $\text{in}(x_j) < n$, using induction,

$$\frac{e^{\lambda n}}{n}\sum_{j=1}^{n}\Phi(\text{in}(x_j)) \le \frac{e^{\lambda n}}{n}\sum_{j=1}^{n}e^{k'\lambda \text{in}(x_j)}. \tag{2.3}$$

Observe that $e^{k'\lambda \text{in}(x_j)}$ is a convex function of $\text{in}(x_j)$ and $\sum_{j=1}^{n}\text{in}(x_j) = \frac{n(n-1)}{2}$. As a result, the RHS of (2.3) is maximized when the in-degrees take their extreme values, *i.e.,* any permutation of $(0, 1, \ldots, n-1)$. Therefore,

$$\begin{aligned}\frac{e^{\lambda n}}{n}\sum_{j=1}^{n}e^{k'\lambda \text{in}(x_j)} &\le \frac{e^{\lambda n}}{n}\sum_{j=0}^{n-1}e^{k'\lambda j}\\ &= \frac{e^{\lambda n}}{n}\frac{e^{k'\lambda n}-1}{e^{k'\lambda}-1}.\end{aligned}$$

Combining the above equations,

$$\Phi(n) \le \frac{e^{\lambda n}}{n}\frac{e^{k'\lambda n}-1}{e^{k'\lambda}-1}.$$

Similarly by induction on $1 \le i < n$,

$$\Phi(i) \le \frac{e^{\lambda i}}{i}\frac{e^{k'\lambda i}-1}{e^{k'\lambda}-1}.$$

Next, we show that for $1 \le i \le n$,

$$\frac{e^{\lambda i}}{i}\frac{e^{k'\lambda i}-1}{e^{k'\lambda}-1} \le e^{k'\lambda i}.$$

For a positive real number t, let

$$f(t) \stackrel{\text{def}}{=} \frac{e^{\lambda t}}{t}\frac{1-e^{-k'\lambda t}}{e^{k'\lambda}-1}.$$

We show that $f(t) \le 1$ for $0 < t \le n$. Observe that,

$$\lim_{t\to 0} f(t) = \frac{k'\lambda}{e^{k'\lambda}-1} \le 1.$$

On the other hand,

$$\begin{aligned}f(n) &= \frac{e^{\lambda n}}{n}\frac{1-e^{-k'\lambda n}}{e^{k'\lambda}-1}\\ &\le \frac{k'}{n}\frac{1}{e^{k'\ln k'/n}-1}\\ &\le \frac{k'}{n}\frac{n}{k'\ln k'}\\ &\le 1.\end{aligned}$$

Next we show that $f(t)$ is convex. One can show that,

$$\ln \frac{1-e^{-u}}{u},$$

is a convex function of u. As a result,

$$\ln \frac{1-e^{-k'\lambda t}}{t},$$

is a convex function of t. Observe that $\ln e^{\lambda t}$ is also convex. Therefore,

$$\ln \frac{1-e^{-k'\lambda t}}{t} + \ln e^{\lambda t},$$

is convex. As a result, logarithm of $f(t)$ is convex and therefore, $f(t)$ is convex. So far we showed that $f(t \to 0) \le 1$, and $f(n) \le 1$, and $f(t)$ is convex. Therefore, for all $0 < t \le n$, $f(t) \le 1$. As a result,

$$\frac{e^{\lambda i}}{i}\frac{e^{k'\lambda i}-1}{e^{k'\lambda}-1} \le e^{k'\lambda i}.$$

By induction, we showed that $\Phi(i) \le e^{k'\lambda i}$ for $1 \le i \le n$ and in particular, $\Phi(n) \le e^{k'\lambda n}$. Substituting $\mathbb{E}[e^{\lambda Q_n}] = \Phi(n)$ in (2.2),

$$\begin{aligned}\Pr(Q_n > kn) &\le \frac{e^{k'\lambda n}}{e^{k\lambda n}} \\ &= \frac{e^{k'\ln k'}}{e^{k\ln k'}} \\ &= e^{-(k-k')\ln k'}.\end{aligned}$$

Hence the lemma is proved. □

While Q-SELECT has linear expected query complexity under the non-adaptive adversarial model, the following example suggested to us by Jelani Nelson [32] shows that it has a quadratic query complexity against an adaptive adversary.

Example 14. *Let $\mathcal{X} = \{0, 0, \ldots, 0\}$. At each round, the adversary declares the pivot to be smaller than all other inputs. Consequently, only the pivot is eliminated and the query complexity is $\binom{n}{2}$.*

2.1.3 Knock-out and quick-select combination

Our final algorithm COMB provides a 2-approximation of x^* even against the adaptive adversarial comparator, and has linear query complexity. This resolves an open question of [5]. We begin with a few simple lemmas.

input: $\mathcal{X}, \epsilon$
  $\beta_1 = 9$, $\beta_2 = 25$, $i = 0$
  while $|\mathcal{X}| > 1$
    $i = i + 1$ (i is the round)
    $n_i = |\mathcal{X}|$
    run $\mathcal{X} \leftarrow$ QS-SUB$(\mathcal{X})$ for $\left\lfloor \beta_1 \log \frac{1}{\epsilon} \right\rfloor$ times
    $\mathcal{X}_i = \mathcal{X}$
    if $|\mathcal{X}_i| > \frac{2}{3} n_i$
      run KO-SUB on *fixed* $\mathcal{X}$ for $\left\lfloor \beta_2 \left(\frac{4}{3}\right)^i \log \frac{1}{\epsilon} \right\rfloor$ times
      if there exists an input with $> \frac{3}{4} \left\lfloor \beta_2 \left(\frac{4}{3}\right)^i \log \frac{1}{\epsilon} \right\rfloor$ wins
        let $\mathcal{X}$ be inputs with $> \frac{3}{4} \left\lfloor \beta_2 \left(\frac{4}{3}\right)^i \log \frac{1}{\epsilon} \right\rfloor$ wins
      else
        let $\mathcal{X}$ be an input with highest number of wins
  end while
output: $\mathcal{X}$

Algorithm 7: COMB - Knock-out and quick-select combination

Lemma 15. *At each round $|\mathcal{X}|$ reduces at least by a third,* i.e., $n_{i+1} \le \frac{2}{3} n_i$.

Proof. If at any round $|\mathcal{X}_i| \le \frac{2}{3} n_i$, then the lemma holds and the algorithm does not call KO-SUB. On the other hand, if KO-SUB is called, then by Markov's inequality at most $\frac{2}{3}$rd of the inputs win more than $\frac{3}{4}$th fraction of queries. As a result, at round i at least $\frac{1}{3}$rd of the inputs in $\mathcal{X}$ will be eliminated. □

Recall that

$$\mathcal{X}^* = \{x \in \mathcal{X} : x \ge x^* - 1\}.$$

The next lemma shows that choosing inputs inside $\mathcal{X}^*$ as a pivot, guarantees a 2-approximation of x^*.

Lemma 16. *If $x^* \in \mathcal{X}$, at a call to* QS-SUB *either x^* survives or a pivot from $\mathcal{X}^*$ is chosen where in the later case, only inputs that are 2-approximation of x^* will survive.*

Proof. Similar to the proof of Lemma 11. □

We showed that at each round, COMB reduces $|\mathcal{X}|$ by at least a third. As a result, the number of inputs decreases exponentially and the total number of queries is linear in n. We also show that if x^* is eliminated at some round, then in that round, an input from $\mathcal{X}^*$ has been chosen as a pivot with high probability. Using Lemma 16, this implies that COMB outputs a 2-approximation of x^* with high probability. We now analyze the performance of COMB.

Theorem 17. $q_n^{\text{COMB,adpt}} = \mathcal{O}\left(n \log \frac{1}{\epsilon}\right)$ *and* $\mathcal{E}_n^{\text{COMB,adpt}}(2) < \epsilon$.

Proof. We start by analyzing the query complexity of COMB. By Lemma 15,

$$n_i \le n \cdot \left(\tfrac{2}{3}\right)^{i-1}.$$

Therefore, the total number of queries at round i is at most

$$n\left(\tfrac{2}{3}\right)^{i-1} \beta_1 \log \tfrac{1}{\epsilon} + \tfrac{n}{2}\left(\tfrac{2}{3}\right)^{i-1} \beta_2 \left(\tfrac{4}{3}\right)^{i} \log \tfrac{1}{\epsilon},$$

where the first term is from calls to QS-SUB and the second one is from calls to KO-SUB. Adding the query complexity of all the rounds,

$$\begin{aligned} q_n^{\text{COMB,adpt}} &\le n \log \tfrac{1}{\epsilon} \sum_{i=1}^{\infty} \left(\beta_1 \left(\tfrac{2}{3}\right)^{i-1} + \tfrac{2}{3}\beta_2 \left(\tfrac{8}{9}\right)^{i-1}\right) \\ &\le n(3\beta_1 + 6\beta_2) \log \tfrac{1}{\epsilon} \\ &= \mathcal{O}\left(n \log \tfrac{1}{\epsilon}\right). \end{aligned}$$

We now analyze the approximation guarantee of COMB. We show that at least one of the following events happens with probability greater than $1 - \epsilon$.

- COMB outputs x^*.
- An input inside $\mathcal{X}^*$ is chosen as a pivot at some round.

Let $\mathcal{X}_i^* \stackrel{\text{def}}{=} \mathcal{X}_i \bigcap \mathcal{X}^*$ and $\alpha_i \stackrel{\text{def}}{=} \frac{|\mathcal{X}_i^*|}{|\mathcal{X}_i|}$. We consider the following two cases separately.

- **Case 1** There exists an i such that $|\mathcal{X}_i| > \frac{2}{3}n_i$ and $\alpha_i > \frac{1}{8}$.
- **Case 2** For all i, either $|\mathcal{X}_i| \leq \frac{2}{3}n_i$ or $\alpha_i \leq \frac{1}{8}$.

First we consider the case 1. We show that in this case a pivot from $\mathcal{X}^*$ is chosen with probability $> 1-\epsilon$. Observe that at round i, $|\mathcal{X}|$ is at most n_i which is less than $\frac{3}{2}|\mathcal{X}_i|$. On the other hand, in all the $\left\lfloor \beta_1 \log \frac{1}{\epsilon} \right\rfloor$ calls to QS-SUB, $|\mathcal{X} \bigcap \mathcal{X}^*|$ is at least $|\mathcal{X}_i^*|$ which is $\alpha_i|\mathcal{X}_i|$. Therefore, in all the calls to QS-SUB at round i,

$$\frac{|\mathcal{X} \bigcap \mathcal{X}^*|}{|\mathcal{X}|} \geq \frac{\alpha_i|\mathcal{X}_i|}{\frac{3}{2}|\mathcal{X}_i|} = \frac{2}{3}\alpha_i.$$

Let E be event of not choosing a pivot from $\mathcal{X}^*$ at round i. As a result,

$$\begin{aligned}
\Pr(E) &\leq \left(1 - \tfrac{2}{3}\alpha_i\right)^{\left\lfloor \beta_1 \log \frac{1}{\epsilon} \right\rfloor} \\
&\leq \left(\tfrac{11}{12}\right)^{8 \log \frac{1}{\epsilon}} \\
&< \epsilon.
\end{aligned}$$

Therefore, in case 1, with probability more than $1-\epsilon$, a pivot from $\mathcal{X}^*$ is chosen.

We now consider the case 2. By Lemma 16 during the calls to QS-SUB, either x^* survives or an input from $\mathcal{X}^*$ is chosen as a pivot. Therefore, the only way to lose x^* without choosing a pivot from $\mathcal{X}^*$ is if at some round i, $|\mathcal{X}_i| > \frac{2}{3}n_i$ and x^* wins less than $\frac{3}{4}$th of its queries during the calls to KO-SUB.

Recall that in case 2, if $|\mathcal{X}_i| > \frac{2}{3}n_i$ then $\alpha_i \leq \frac{1}{8}$. Observe that x^* wins against a random input in $\mathcal{X}_i$ with probability greater than $> 1 - \alpha_i$ which is at least $\frac{7}{8}$. Let E_i' be the event that x^* wins fewer than $\frac{3}{4}$th of its queries at round i. By the Chernoff bound,

$$\begin{aligned}
\Pr(E_i') &\leq \exp\left(-D\left(\tfrac{3}{4}\|\tfrac{7}{8}\right)\left\lfloor \beta_2 \left(\tfrac{4}{3}\right)^i \log \tfrac{1}{\epsilon} \right\rfloor\right) \\
&\leq \epsilon^{2\left(\frac{4}{3}\right)^i},
\end{aligned}$$

where $D(p\|q) \stackrel{\text{def}}{=} p \ln \frac{p}{q} + (1-p) \ln \frac{1-p}{1-q}$ is the Kullback-Leibler distance between Bernoulli distributed random variables with parameters p and q respectively. Assuming $\epsilon < \frac{1}{2}$, the total probability of missing x^* without choosing a pivot form $\mathcal{X}^*$

is at most

$$\sum_{i=1}^{\infty} \Pr(E_i') \leq \sum_{i=1}^{\infty} \epsilon^{2\left(\frac{4}{3}\right)^i}$$
$$< \epsilon.$$

So far we showed that with probability $> 1-\epsilon$, either x^* survives or an input inside $\mathcal{X}^*$ is chosen as a pivot. The theorem follows from Lemma 16. □

2.2 Application to density estimation

Our study of maximum selection with adversarial comparators was motivated by the following density estimation problem.

Given a *known* set $\mathcal{P}_\delta = \{p_1, \ldots, p_n\}$ of n distributions and k samples from an *unknown* distribution p_0, output a j such that for a small constant $C > 1$ with high probability,

$$\|p_0 - p_j\|_1 \leq C \cdot \min_i \|p_0 - p_i\|_1 + o_k(1).$$

This problem was studied in [19] who showed that for $n = 2$, the SCHEFFE-TEST takes k samples and with probability $1-\varepsilon$ outputs a distribution $\hat{p} \in \mathcal{P}_\delta$ such that

$$\|\hat{p} - p_0\|_1 \leq 3 \cdot \min_{p \in \mathcal{P}_\delta} \|p - p_0\|_1 + \sqrt{\frac{10 \log \frac{1}{\varepsilon}}{k}}. \tag{2.4}$$

input: distributions p_1 and p_2, k *i.i.d.* samples of unknown distribution p_0

let $\mathcal{S} = \{x : p_1(x) > p_2(x)\}$

let $p_1(\mathcal{S})$ be the probability mass that p_1 assigns to $\mathcal{S}$

let $p_2(\mathcal{S})$ be the probability mass that p_2 assigns to $\mathcal{S}$

let $\mu_{\mathcal{S}}$ be the frequency of samples in $\mathcal{S}$

output: if $|p_1(\mathcal{S}) - \mu_{\mathcal{S}}| \leq |p_2(\mathcal{S}) - \mu_{\mathcal{S}}|$ output p_1, otherwise output p_2

Algorithm 8: SCHEFFE-TEST- Scheffe test for two distributions

SCHEFFE-TEST provides a factor 3-approximation with high probability. The algorithm, as stated in its pseudocode, requires computing $p_i(\mathcal{S})$ which can

be hard since the distributions are not restricted. However, as noted in [3], the algorithm can be made to run in time linear in k. [19] also extend this result for $n > 2$. Their proposed algorithm for $n > 2$, runs SCHEFFE-TEST for each pair of distributions in $\mathcal{P}_\delta$ and outputs the distribution with maximum number of wins, where a distribution is a winner if it is the output of SCHEFFE-TEST. This algorithm is referred to as the Scheffe tournament or the complete tournament. They showed that this algorithm finds a distribution $\hat{p} \in \mathcal{P}_\delta$ such that

$$\|\hat{p} - p_0\|_1 \leq 9 \min_{p \in \mathcal{P}_\delta} \|p - p_0\|_1 + o_k(1),$$

and the running time is clearly $\Theta(n^2 k)$, quadratic in the number of distributions.

[27] showed that the optimal coefficients for the Scheffe algorithms is indeed 3 and 9 for $n = 2$ and $n > 2$ respectively. They also proposed an algorithm with an improved factor 3-approximation for $n > 2$, however still running in time $\Theta(n^2)$. They also proposed a linear-time algorithm, but it requires a preprocessing step that runs in time exponential in n.

Scheffe's method has been used recently to obtain nearly sample optimal algorithms for learning Poisson Binomial distributions [16], and Gaussian mixtures [17, 3].

We now describe how our noisy comparison model can be applied to this problem to yield a linear-time algorithm with the same estimation guarantee as Scheffe tournament. Our algorithm uses Scheffe test as a subroutine. Given a sufficient number of samples, $k = \Theta(\log n)$, the small term in the RHS of (2.4) vanishes and the Scheffe test's outputs

$$\begin{cases} p_i & \text{if } \|p_i - p_0\|_1 < \frac{1}{3} \|p_j - p_0\|_1, \\ p_j & \text{if } \|p_j - p_0\|_1 < \frac{1}{3} \|p_i - p_0\|_1, \\ \text{unknown} & \text{otherwise.} \end{cases}$$

Let $x_i = -\log_3 \|p_i - p_0\|_1$, then analogously to the maximum selection with adversarial noise in (1.1), the Scheffe test outputs

$$\begin{cases} \max\{x_i, x_j\} & \text{if } |x_i - x_j| > 1, \\ \text{unknown} & \text{otherwise.} \end{cases}$$

Given a fixed multiset of samples the tournament results are fixed, hence this setup is identical to the non-adaptive adversarial comparators. In particular, with probability $1 - \varepsilon$, our quick-select algorithm can find $\hat{p} \in \mathcal{P}_\delta$ such that

$$\|\hat{p} - p_0\|_1 \leq 9 \cdot \min_{p \in \mathcal{P}_\delta} \|p - p_0\|_1 + o_k(1),$$

with running time $\Theta(nk)$.

Acknowledgements

Chapter 2 is adapted from Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Sorting with adversarial comparators and application to density estimation", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 2014; and Jayadev Acharya, Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Maximum Selection and Sorting with Adversarial Comparators and Application to Density Estimation", In Preparation, 2015; The dissertation author was the primary researcher and author of these papers.

Chapter 3

Noisy sorting

3.1 Problem statement

We now consider sorting with noisy comparators. The comparator model is as before but the goal is to approximately sort the inputs in decreasing order with noisy comparisons.

Consider an Algorithm $\mathcal{A}$ for sorting the inputs. The output of $\mathcal{A}$ is denoted by $\mathbf{Y}_{\mathcal{A}}(\mathcal{X}) \stackrel{\text{def}}{=} (Y_1, Y_2, \ldots, Y_n)$ which is a particular ordering of the inputs. Note that since algorithms are randomized, $\mathbf{Y}_{\mathcal{A}}$ has randomness. Similar to maximum selection problem, a t-approximation error is

$$\mathcal{E}_n^{\mathcal{A}}(t) \stackrel{\text{def}}{=} \Pr\left(\max_{i,j:i>j}(Y_i - Y_j) > t\right),$$

i.e., the probability of Y_i appearing after Y_j in $\mathbf{Y}_{\mathcal{A}}$ while $Y_i - Y_j > t$. The definitions of $\mathcal{E}_n^{\mathcal{A},\text{non}}(t)$ and $\mathcal{E}_n^{\mathcal{A},\text{adpt}}(t)$ is same as before.

In the following, we first revisit complete tournament with small modification for the sake of sorting problem and we show that under adaptive adversarial model, it has zero 2-approximation error and query complexity of $\binom{n}{2}$. Then we discuss quick-sort algorithm Q-SORT and show that it has zero 2-approximation error but with improved query complexity for the non-adaptive adversary. We apply the known bounds for running time of general quick-sort algorithm with n distinct inputs to find the query complexity of Q-SORT.

3.2 Complete tournament

The algorithm is similar to COMPL in Section 1.5.2 and we refer to it as COMPL-SORT. The only difference is in the output of the algorithm.

input: $\mathcal{X}$

compare all input pairs in $\mathcal{X}$

count the number of times each input wins

output: output the inputs in the order of their number of wins, breaking the ties randomly

Algorithm 9: COMPL-SORT - Complete tournament

The following lemma and its proof is similar to Lemma 4 and therefore we skip the proof.

Lemma 18. $q_n^{\text{COMPL-SORT,adpt}} = \binom{n}{2}$ *and* $\mathcal{E}_n^{\text{COMPL-SORT,adpt}}(2) = 0$.

Next we discuss an algorithm with improved query complexity.

3.3 Quick-sort

Quick-sort is a well known algorithm and here is denoted by Q-SORT. The expected query complexity of quick-sort with noiseless comparisons and distinct inputs is

$$f(n) \stackrel{\text{def}}{=} 2n\ln n - (4-2\gamma)n + 2\ln n + \mathcal{O}(1), \tag{3.1}$$

where γ is Euler's constant [28]. Note that $f(n)$ is a convex function of n.

In the rest of this section we study the error guarantee of quick-sort and its query complexity in the presence of noise. In Lemma 19 we show that the error guarantee of quick-sort for our noise model is same as complete tournament, *i.e.,* it can sort the inputs with zero 2-approximation error. Next in Lemma 20 we show that the expected query complexity of quick-sort with non-adaptive adversarial noise is smaller than its expected query complexity in noiseless model.

Lemma 19. $\mathcal{E}_n^{\text{Q-SORT,adpt}}(2) = 0$.

Proof. The proof is by contradiction. Suppose $x_i > x_j + 2$ but x_j appears before x_i in the output of quick-sort algorithm. Then there must have been a pivot x_p such that $\mathcal{C}(x_i, x_p) = x_p$ while $\mathcal{C}(x_j, x_p) = x_j$. Since $x_i > x_j + 2$ no such a pivot exists. □

Quick-sort algorithm chooses a pivot randomly to divide the set of inputs into smaller-size sets. The optimal pivot for noiseless quick-sort is known to be the median of the inputs to balance the size of remained sets. In fact, it is easy to show that if we choose the median of the inputs as pivot, the query complexity of quick-sort reduces to $\approx n \log n$. Observe that in a non-adaptive adversarial model, the probability of having balanced sets after choosing pivot increases. As a result, in the next lemma we show that the expected query complexity of quick-sort in the presence of noise is upper bounded by $f(n)$.

Lemma 20. $q_n^{\text{Q-SORT,non}} = f(n)$ *and is achieved when queries are noiseless and inputs are distinct.*

Proof. Let $\text{in}(x)$ and $\text{out}(x)$ be the in-degree and out-degree of node x in the complete tournament respectively. For the noiseless comparator with distinct inputs, the in-degrees and out-degrees of inputs are permutation of $(0, 1, \ldots, n-1)$. We show that

$$\operatorname*{argmax}_{\mathcal{C} \in \mathcal{C}_{\text{non}}} \max_{\mathcal{X}} q_n^{\text{Q-SORT}},$$

is a comparator where its complete tournament in-degrees and out-degrees are permutation of $(0, 1, \ldots, n-1)$. For the simplicity of notation let $q_n = q_n^{\text{Q-SORT,non}}$. We have the following recursion for quick-sort similar to (2.1).

$$q_n \le n - 1 + \frac{1}{n} \sum_{i=1}^{n} q_{\text{out}(x_i)} + q_{\text{in}(x_i)} \tag{3.2}$$

By induction, we show that the solution to (3.2) is bounded above by $f(n)$, a convex function of n. The induction is true for $n = 0, 1$, and 2. Now suppose the induction is true for all $i < n$. Since $f(n)$ is a convex function of n and $\sum_i \text{in}(x_i) = \sum_i \text{out}(x_i) = \frac{n(n-1)}{2}$, the right hand side of (3.2) is maximized when the in-degrees and out-degrees take their extreme values, *i.e.,* when they are permutation of

$(0, 1, \ldots, n-1)$. Plugging in these values, (3.2) is equivalent to,

$$\begin{aligned} q_n &\leq n - 1 + \frac{1}{n}\sum_{i=1}^{n} f(\text{in}(x_i)) + f(\text{out}(x_i)) \\ &\leq n - 1 + \frac{1}{n}\sum_{i=1}^{n} f(i-1) + f(n-i), \end{aligned}$$

where the solution to this recursion is $f(n)$, given in (3.1). Hence q_n is bounded above by $f(n)$ and equality happens when in-degrees and out-degrees are permutation of $(0, 1, \ldots, n-1)$. □

[25, 22, 29] show different concentration bounds for quick-sort. In particular, [29] show that the probability of quick-sort algorithm requiring more comparisons than $(1+\epsilon)$ times its expected query complexity is $n^{-2\epsilon \ln\ln n + \mathcal{O}(\ln\ln\ln n)}$. Observe that for the non-adaptive adversarial model, the chance of a random pivot cutting inputs into balanced sets increases. As a result, one can show that the analysis in [29] follows automatically. In particular, Lemmas 2.1 and 2.2 in [29], which are the basis of their analysis, are valid for our non-adaptive adversarial model. Therefore, their tight concentration bound for quick-sort algorithm can be applied to our non-adaptive adversarial model.

Acknowledgements

Chapter 3 is adapted from Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Sorting with adversarial comparators and application to density estimation", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 2014; and Jayadev Acharya, Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "Maximum Selection and Sorting with Adversarial Comparators and Application to Density Estimation", In Preparation, 2015; The dissertation author was the primary researcher and author of these papers.

Part II

On the Verification Query of Symmetric Functions

Chapter 4

Introduction

4.1 background

A basic model of multivariate function computation is the *decision-tree complexity* or *worst-case query complexity* [8, 10, 38, 33]. In this model, we find the function value by adaptively choosing the inputs queried while optimizing the maximum number of queries needed to determine the function value. For example, the worst-case query complexity of a Boolean function $xy \vee \overline{x}z$ is 2, as the value of x determines which of y or z needs to be queried to determine the function value. For a survey of decision-tree complexity, please see [14].

A multivariate function is *symmetric* if its output remains unchanged under all input permutations. Many functions encountered in engineering and science are symmetric, including *parity*, *threshold*, and *delta*, as well as most statistical measures such as *median*, *mode*, *max*, *etc.* [39, 30, 6] considered the complexity of symmetric functions.

It has been shown, *e.g.* [20], that the worst-case query complexity of all non-constant symmetric functions is n. On the other hand, average-case query complexity received less attention. Perhaps because it is difficult to find a fixed and robust probabilistic model for the data, or because minimizing the expected depth of a decision tree is more difficult to analyze than minimizing the maximum depth.

Despite getting less attention, *expected* or average-case query complexity

for computing a function is more important than the worst-case in most of the problems. For example, when an airline decides to set a price for a ticket, often the expected earning is more of attention than the worst-case or when a gambler wants to decide a strategy, often the focus is on increasing expected gain than worst-case.

[9, 37, 11] considered the expected query complexity. [7] showed that for the expected query complexity under the uniform distribution, quantum algorithms can be exponentially faster than classical algorithms. [26] considered the expected query complexity of computing symmetric functions. For the expected query complexity, the optimal query order depends not only on the function, but also on the underlying distribution of variables. [26] found an optimal query order for threshold functions of independent but not necessarily identical Bernoulli random variables. In particular, they showed that for threshold functions of independent Bernoulli random variables, the optimal query order does not depend on the precise probabilities of inputs, but only on which is the largest, the second largest, *etc.*

To simplify and extend arguments for finding the optimal query order, [2] defined the expected *verification* query complexity of a function to be the lowest expected number of inputs that need to be revealed to convince an observer of the value of the function. For example, consider the logical OR function $X_1 \vee \ldots \vee X_n$, where each $X_i \sim \text{bernoulli}(p_i)$ and independently. To verify that the OR function is 1, it suffices to show that one of the inputs is 1; hence for moderate values of p_i's, the expected number of inputs that need to be revealed is small, whereas verifying that the OR function is 0, requires checking that all inputs are 0, hence all the n inputs must be queried. Note that verification complexity differs from *certificate* complexity [1, 14, 8], where all input values are known in advance and can be used to determine the optimal query order.

We show that for all symmetric functions of independent binary inputs, the optimal expected verification and computation complexities are equal. We use this result to simplify the proof of the optimal query complexity of threshold functions presented in [26]. We observe that the value of all binary-input symmetric

functions depends only on the number of ones, or *weight*, of the input, and use this property to find an optimal query order for all delta functions and for all symmetric functions that are not constant over any three consecutive input weights. We also show that after relaxing any of the symmetry, independence, or binary inputs restrictions, there are functions whose verification complexity is strictly lower than their computation complexity.

This part of the dissertation is organized as follows. In Section 4.2, we formally define the problems of computation and verification. In Section 4.3, we make a few observations to simplify our analysis. In Section 5.1, we show that verification method can be used to find the query order for computing threshold and delta functions. In Section 5.2, we show the equality of verification and computation for general symmetric functions of independent binary inputs. In Section 5.3, we demonstrate that symmetry, independency, and binary restrictions are all necessary to have equal verification and computation complexities.

4.2 Notation and formulation

Except Section 5.3, we assume that f is a symmetric function of n binary inputs $\mathbf{X} \stackrel{\text{def}}{=} X_1, X_2, \ldots, X_n$, where $X_i \sim \text{bernoulli}(p_i)$ independently, and the p_i's are *known* in advance. Without loss of generality, assume that $1 > p_1 \geq p_2 \geq \ldots \geq p_n > 0$. $[i, j]$ denotes the set of integers between and including i and j. For any set $\mathcal{S}$, $|\mathcal{S}|$ denotes the number of element in $\mathcal{S}$. Let $\bar{p}_i \stackrel{\text{def}}{=} 1 - p_i$.

To compute $f(\mathbf{X})$, we query the inputs sequentially. A *policy* $\mathcal{P}$ is a rule that at any given time, based on prior query outcomes, determines whether querying should stop or continue, and if the latter, which input should be queried next. $\mathcal{P}$ *computes* f, if for all values of $\mathbf{X}$, when $\mathcal{P}$ stops querying, f can be determined.

Let $Q_{\mathcal{P}}(\mathbf{x})$ be the number of inputs a policy $\mathcal{P}$ queries for input $\mathbf{x}$. The expected query complexity of $\mathcal{P}$ is

$$C(\mathcal{P}) \stackrel{\text{def}}{=} \mathbb{E}[Q_{\mathcal{P}}(\mathbf{X})] = \sum_{\mathbf{x}} p(\mathbf{x}) Q_{\mathcal{P}}(\mathbf{x}),$$

and the *computation complexity* of f is

$$C(f) \stackrel{\text{def}}{=} \min_{\mathcal{P}} C(\mathcal{P}) = \min_{\mathcal{P}} \sum_{\mathbf{x}} p(\mathbf{x}) Q_{\mathcal{P}}(\mathbf{x}), \tag{4.1}$$

where the minimum is taken over all the policies $\mathcal{P}$ computing f. Any policy that computes f with complexity $C(f)$ is an *optimal* computation policy. In general, there might be several optimal computation policies.

Example 21. *Consider the* threshold *function,*

$$\Pi_\theta(w) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } w \geq \theta, \\ 0 & \text{otherwise}, \end{cases}$$

and let $f(x_1, x_2) = \Pi_1(x_1 + x_2)$ *determine if at least one of* x_1 *and* x_2 *is 1. If* X_i *is queried first, the expected number of queries is* $1 \cdot p_i + 2 \cdot (1 - p_i) = 2 - p_i$*, since a policy can stop after querying an input with value one and it has to query the second input if it queries an input with value zero. As a result, the optimal policy should query* X_1 *first if* p_1 *is strictly greater than* p_2*.*

In theory, it is possible to express the computation complexity of any function and policy in terms of the input probabilities and optimize the query order. But since the number of policies is exponential in n, this may be computationally inefficient.

An alternative approach was proposed in [2]. Instead of finding an optimal policy to compute a function, they considered the *simpler* problem of finding an optimal policy to verify the function value. They found a class of functions for which the two policies coincide.

In the *verification* of a function f, we are given the value of $f(\mathbf{X})$, and are asked to query the inputs to verify that this is indeed the function value. As with computation, we apply a policy that determines which inputs to query and when to stop so that the function value can be determined. The only difference is that we have a freedom to use different policies for different function values.

It is easy to see that a *verification policy* is just a collection of computation policies, one for each value of f, and the advantage of verification is that for

each value of f we can choose a policy that minimizes the expected number of queries for that value of f. The difference between verification and computation complexities is perhaps easier to demonstrate via a non-symmetric function of dependent random variables.

Example 22. *Let $n > 1$ be an integer. Let $\mathbf{e}_i$ be the unit vector in $\mathbb{R}^n$, whose ith component is 1 and all others are 0. Let $\mathbf{X} = \mathbf{e}_i$ with probability $\frac{1}{n}$,* i.e., *one of the n unit vectors with equal probability. Let $f : \{0,1\}^n \to \{1,...,n\}$ be such that $f(\mathbf{e}_i) = i$. For example, for $n = 3$, $f(100) = 1$, $f(010) = 2$, and $f(001) = 3$, and $Pr(100) = Pr(010) = Pr(001) = 1/3$.*

To compute f, we must find an i such that $X_i = 1$. Therefore, we need to query the inputs till we find the input whose value is 1 or find the $n-1$ inputs whose values are 0. Hence the computation complexity is

$$\frac{1}{n}\left(1 + 2 + \ldots + (n-1) + (n-1)\right) = \frac{(n-1)(n+2)}{2n}.$$

However, for verification we are given $f(\mathbf{X}) = i$ for some i and we can verify $X_i = 1$ by only querying X_i. Hence the verification complexity is 1.

For a more precise definition, the expected query complexity of policy $\mathcal{P}$ when $f(\mathbf{X}) = j$ is

$$C(\mathcal{P}|f(\mathbf{X}) = j) \stackrel{\text{def}}{=} \mathbb{E}\left[Q_{\mathcal{P}}(\mathbf{X})|f(\mathbf{X}) = j\right] = \sum_{\mathbf{x}:f(\mathbf{x})=j} \frac{p(\mathbf{x})Q_{\mathcal{P}}(\mathbf{x})}{\Pr\left(f(\mathbf{X}) = j\right)}.$$

The verification complexity of f when $f(\mathbf{X}) = j$ is the smallest expected number of inputs that need to be queried to verify that $f(\mathbf{X}) = j$, *i.e.,*

$$V_j(f) \stackrel{\text{def}}{=} \min_{\mathcal{P}} C(\mathcal{P}|f(\mathbf{X}) = j) = \min_{\mathcal{P}} \sum_{\mathbf{x}:f(\mathbf{x})=j} \frac{p(\mathbf{x})Q_{\mathcal{P}}(\mathbf{x})}{\Pr\left(f(\mathbf{X}) = j\right)},$$

where the minimum is taken over all policies that verify $f(\mathbf{X}) = j$. Equivalently we can take the minimum over all policies computing f since for the values of f other than j the behavior of the computation policy is not important.

The minimum expected verification query complexity or simply the *verification complexity* of f is

$$V(f) \stackrel{\text{def}}{=} \sum_j \Pr(f(\mathbf{X}) = j)V_j(f) = \sum_j \min_{\mathcal{P}} \sum_{\mathbf{x}:f(\mathbf{x})=j} p(\mathbf{x})Q_{\mathcal{P}}(\mathbf{x}), \tag{4.2}$$

where for each j, we find a potentially different policy $\mathcal{P}$ minimizing $V_j(f)$.

An optimal verification policy is one whose expected query complexity is $V(f)$. As with its computation counterpart, f may have several optimal verification policies.

In the next section, we make a few observations about computation and verification complexity and in Section 5.2, we show that for all symmetric functions of independent binary inputs, $V(f) = C(f)$.

4.3 Preliminary observations

Since computation is one way of verification, or equivalently, a verification policy is a set of computation policies, one for each value of f, the verification complexity is at most the computation complexity.

Observation 23. *For all f, $V(f) \leq C(f)$.*

Proof. Comparing Equations (4.1) and (4.2),

$$V(f) = \sum_j \min_{\mathcal{P}} \sum_{\mathbf{x}:f(\mathbf{x})=j} p(\mathbf{x})Q_{\mathcal{P}}(\mathbf{x}) \leq \min_{\mathcal{P}} \sum_{\mathbf{x}} p(\mathbf{x})Q_{\mathcal{P}}(\mathbf{x}) = C(f),$$

since the sum of minimums is at most the minimum of sum. □

Recall that symmetric functions of binary inputs are determined by the input's weight $w \stackrel{\text{def}}{=} w(\mathbf{X}) \stackrel{\text{def}}{=} \sum_{i=1}^{n} X_i$. With a slight abuse of notation, we use $f(w(\mathbf{X}))$ and $f(\mathbf{X})$ interchangeably. The following observation shows that when a policy computing f stops, the value of f is constant for all possible weights.

Observation 24. *If for inputs $\mathbf{x}$, a policy stops after querying n_0 zeros and n_1 ones, then $w(\mathbf{x})$ can take any value in the contiguous interval $[n_1, n - n_0]$. Furthermore, if the policy computes f, then $f(\mathbf{x})$ is constant for all $\mathbf{x}$ such that $w(\mathbf{x}) \in [n_1, n - n_0]$.*

Proof. After querying n_0 zeros and n_1 ones, $\sum_i x_i$ can take any value in $[n_1, n-n_0]$ depending on the values of the unknown inputs. Since when the policy stops, the function value is determined, regardless of the unknown inputs, $f(\mathbf{x})$ must be the same for all inputs with $w(\mathbf{x}) \in [n_1, n - n_0]$. □

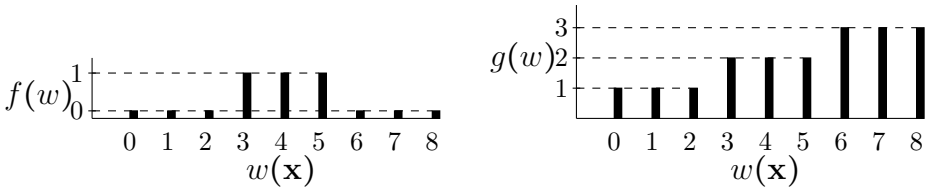


Figure 4.1: Example of function g

Let the *interval indicator function* of f be the function $g : [0, n] \to [1, n+1]$, defined by $g(0) \stackrel{\text{def}}{=} 1$ and the following recursion,

$$g(i+1) - g(i) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } f(i+1) = f(i), \\ 1 & \text{if } f(i+1) \neq f(i). \end{cases}$$

$g(w)$ indicates which peacewise-constant interval of f, $w(\mathbf{X})$ belongs to. Figure 4.1 demonstrates the relation between f and g by example. Define $\mathcal{I}_j \stackrel{\text{def}}{=} \{w | g(w) = j\}$ to be the j^{th} peacewise-constant interval of the function g. The next observation, lower bounds the number of inputs needed in order to compute g.

Observation 25. *If $w(\mathbf{x}) \in \mathcal{I}_j$ then for any policy $\mathcal{P}$, $Q_{\mathcal{P}}(\mathbf{x}) \geq n - |\mathcal{I}_j| + 1$.*

Proof. From Observation 24, we conclude that $g(x) = j$ for $j \in [n_1, n - n_0]$, where n_0 is the number of queried zeros and n_1 is the number of queried ones. Hence, $[n_1, n - n_0] \subset \mathcal{I}_j$. Therefore, $n - n_0 - n_1 + 1 \leq |\mathcal{I}_j|$. Equality is achieved, when $[n_1, n - n_0] = \mathcal{I}_j$. □

We divide intervals into two types. An interval $\mathcal{I}_j$ is *large* if $|\mathcal{I}_j| \geq \frac{n+1}{2}$ and *small* otherwise. In Section 5.2, we consider the behavior of optimal verification policies for these two types of intervals separately. The next observation shows that f and g have the same computation complexity.

Observation 26. *For any f, a symmetric function of independent binary random variables,*

$$C(f) = C(g).$$

Proof. For every symmetric function f, the value of g determines the value of f. Therefore, any policy that computes g also computes f. Hence $C(f) \leq C(g)$.

Conversely, while the value of f may not determine the value of g, *e.g.,* for the parity function, by Observation 24, when the value of f is determined, $w(\mathbf{X})$ lies in a known interval over which f is constant, and hence g is determined as well. Therefore, a policy that computes f also computes g and $C(g) \leq C(f)$. □

While the same result also holds for verification complexity, only one direction is easy to prove.

Observation 27. *For any f, a symmetric function of independent binary random variables,*

$$V(g) \leq V(f).$$

Proof. Recall that verification policy for f is a collection of computation policies, one for each value of f. Since g determines f, and a verification policy for f is also a verification policy for g, $V(g) \leq V(f)$. □

Our main result, Theorem 35, shows that for all symmetric functions f of independent binary random variables, $V(g) = C(g)$. Combining the above observations, we obtain $C(g) = V(g) \leq V(f) \leq C(f) = C(g)$ and hence,

Corollary 28. *For all symmetric functions f of independent binary random variables,*

$$V(g) = V(f) = C(f) = C(g).$$

In general, verification complexity appears to be easier to determine than computation complexity, and verification complexity of the interval indicator functions seems easier to determine than the verification complexity of symmetric functions.

In the next chapter we give examples that demonstrates this simplicity. For these examples, we can find the computation policy by only considering the verification one. A generalization of them is given in Section 5.2.

Acknowledgements

Chapter 4 is adapted from Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Shengjun Pan, Ananda Theertha Suresh, "On the query computation and

verification of functions", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 2012; and Jayadev Acharya, Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "On the Computation and Verification Query complexity of Symmetric Functions", In preparation, 2015; The dissertation author was the primary researcher and author of these papers.

Chapter 5

Comparing Verification and Computation

5.1 Simplicity of verification

In this section, we find the optimal computation policies for the threshold and *delta* functions. We start with the threshold function and find a specific optimal verification policy for it. Next we show that there exists a computation policy which follows that specific optimal verification policy. As a result, the computation policy should be optimal since computation complexity is at least verification complexity. Finally we generalize the result to delta functions. We start by a simple lemma about the property of optimal verification policies.

Consider the case when the threshold function $\Pi_\theta(\mathbf{X})$ has value 1. This can be verified only when θ ones have been observed. Therefore, an optimal verification policy should minimize the expected time to observe θ ones. It is intuitive to first query the input with highest probability, *i.e.,* X_1, and then X_2 if it is necessary and so on. The following lemma formalizes this intuition.

Lemma 29. *For $\Pi_\theta(\mathbf{X}) = 1$, an optimal verification policy is to query in the order $X_1, X_2, \ldots$ until θ ones are observed and for $\Pi_\theta(\mathbf{X}) = 0$, the order is $X_n, X_{n-1}, \ldots$ until $(n - \theta + 1)$ zeros are observed.*

Proof. We prove the optimality when $\Pi_\theta(\mathbf{X}) = 1$. A similar argument holds when

$\Pi_\theta(\mathbf{X}) = 0$. It suffices to show that querying X_1 as the first input is optimal since after the first query, the problem reduces to computing another threshold function for the rest of the inputs.

We prove optimality of querying X_1 by induction on (n, θ). If $n < \theta$ the function is trivially zero. If $n = \theta$, all inputs must be queried to verify that $\Pi_\theta(\mathbf{X}) = 1$. Therefore, X_1 can be queried first.

For $\theta \geq 2$, suppose X_k is queried first. If $X_k = 1$, we have to find an optimal verification policy for $n-1$ inputs and threshold $\theta - 1$, and if $X_k = 0$, we have to find an optimal verification policy for $n-1$ inputs and threshold θ. By induction hypothesis, for both these cases there is an optimal hypothesis in which X_1 is queried next. This implies that there is an optimal policy for our problem in which the first two observed inputs are X_k and X_1, and since $\theta \geq 2$ at least two inputs should be queried. Thus the order of X_k and X_1 is immaterial and X_1 could be queried first, followed by X_k and no further changes to the policy.

This leaves us with the case when $\theta = 1$. For $n = 1$ it is trivial to ask X_1. For $n = 2$, we proved in Example 21 that X_1 should be queried first. For $n > 2$ and $\theta = 1$, we again use induction to prove X_1 should be queried first. Suppose an optimal policy $\mathcal{P}_1$ that queries $X_k \neq X_1$ first. If $X_k = 1$ then policy $\mathcal{P}_1$ should stop querying the inputs. On the other hand, if $X_k = 0$, then by induction hypothesis, policy $\mathcal{P}_1$ should query X_1 next. Now consider the new policy $\mathcal{P}_2$ which queries X_1 and then X_k if needed, and from the third step onwards, the policy follows the decision of the optimal policy $\mathcal{P}_1$. If $(X_1, X_k) = (0,0)$ or $(1,1)$, then $\mathcal{P}_1$ and $\mathcal{P}_2$ query the same number of inputs. Therefore, we only consider the case when $\{X_1, X_k\} = \{0,1\}$, *i.e.,* one and only one of the X_1 and X_k is 1. Let

$$\alpha \stackrel{\text{def}}{=} \Pr\left(X_1 = 1, X_k = 0 | \{X_1, X_k\} = \{0,1\}\right) = \frac{p_1 \bar{p}_k}{p_1 \bar{p}_k + \bar{p}_1 p_k} \geq \frac{1}{2}.$$

Observe that $\mathbb{E}\left[Q_{\mathcal{P}_2}(\mathbf{X}) | \{X_1, X_k\} = \{0,1\}\right] = \alpha + 2\bar{\alpha}$ and $\mathbb{E}\left[Q_{\mathcal{P}_1}(\mathbf{X}) | \{X_1, X_k\} = \{0,1\}\right] = 2\alpha + \bar{\alpha}$. Since $\alpha \geq \frac{1}{2}$, the expected query complexity of policy $\mathcal{P}_2$ is smaller than or equal to the expected query complexity of policy $\mathcal{P}_1$. Equality only happens when $p_1 = p_k$. As a result, an optimal policy to verify $\Pi_1(\mathbf{X}) = 1$ could query X_1 first. Hence the lemma is proved. □

The next lemma finds an input which can be queried first in an optimal verification policy for threshold function. This lemma helps us to find the optimal computation policy.

Lemma 30. *An optimal verification policy queries X_θ regardless of value of $\Pi_\theta(\mathbf{X})$.*

Proof. In Lemma 29 we showed that if $\Pi_\theta(\mathbf{X}) = 1$, an optimal verification policy queries $X_1, X_2, \ldots$ till finding θ ones. As a result, if $\Pi_\theta(\mathbf{X}) = 1$, an optimal verification policy should query X_θ at some point. Similarly, if $\Pi_\theta(\mathbf{X}) = 0$, an optimal verification policy queries X_θ at some point. □

Lemma 30 shows that if we query X_θ first, we can *track* an optimal verification policy without knowing the function value. After observing the value of X_θ, the threshold function will reduce to a new threshold function over $n-1$ inputs. Therefore, we can continue on tracking the optimal verification policy. Pseudocode of this verification policy is described in OPTIMAL-POLICY-THRESHOLD.

```
let i = j = θ and Y = 1
while θ ones or n − θ + 1 zeros are not queried
    if Y = 1
        query X_i, let Y = X_i, and decrement i
    else
        query X_j, let Y = X_j, and increment j
end while
output Y
```

Algorithm 10: OPTIMAL-POLICY-THRESHOLD

Observe that OPTIMAL-POLICY-THRESHOLD is an optimal verification policy for both $\Pi_\theta(\mathbf{X}) = 0$ and $\Pi_\theta(\mathbf{X}) = 1$. This policy does not need to know the function value to proceed and therefore, it is also a computation policy. Since computation complexity is at least verification complexity, OPTIMAL-POLICY-THRESHOLD is an optimal computation policy.

In the following we generalize this result to the delta function which is defined as

$$\Delta_\theta(w) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } w = \theta, \\ 0 & \text{otherwise.} \end{cases}$$

When a policy computes $\Delta_\theta(w)$ it also shows us whether $w < \theta$, $w = \theta$ or $w > \theta$. This can be seen by Observation 24. We consider the problem of verification for these cases separately. Using the results obtained for threshold functions, we obtain the following optimal verification policies.

- $w = \theta$: In this case all inputs must be queried to verify that the weight is exactly θ.
- $w < \theta$: This problem is identical to the threshold problem with θ as the threshold value. We know that an optimal verification policy is to query $X_n, X_{n-1}, \ldots$ until $n - \theta + 1$ zeros are observed.
- $w > \theta$: This problem is identical to the threshold problem with $\theta + 1$ as the threshold value. We know that an optimal verification policy is to query $X_1, X_2, \ldots$ until $\theta + 1$ ones are observed.

By comparing these optimal verification policies to optimal verification policies for the threshold function we can easily show that OPTIMAL-POLICY-THRESHOLD can be used for optimal computation policy for delta function with only changing the stopping criterion. The new stopping criteria is when $\theta + 1$ ones or $n - \theta + 1$ zeros are observed, or all inputs have been queried. For the sake of completeness, the Pseudocode for this policy is given in OPTIMAL-POLICY-DELTA.

5.2 Functions with same verification and computation

In this section we consider the general symmetric functions of independent binary inputs. Similar to Section 5.1, we show that verification and computation complexities of these functions coincide. In Theorem 35 we show that regardless

```
let i = j = θ and Y = 1
while θ + 1 ones or n − θ + 1 zeros are not queried
    if all the inputs are queried
        output 1 and stop
    if Y = 1
        query X_i, let Y = X_i, and decrement i
    else
        query X_j, let Y = X_j, and increment j
end while
output 0
```

Algorithm 11: OPTIMAL-POLICY-DELTA

of the function value, there is an optimal verification policy where some inputs can be always queried first. This result yields to an optimal computation policy that is also an optimal verification policy for all the function values.

The proof of Theorem 35 is evolved and Figure 5.1 demonstrates the main steps of it. Recall that we defined large and small intervals in Section 4.3. In Lemma 33, we consider the optimal verification policy when the weight of the inputs belongs to a large interval and we find some inputs of which one can be queried first in an optimal verification policy. Conversely, when the weight of the inputs belongs to a small interval, Lemma 34 finds some inputs any of which can be queried first in an optimal verification policy. A combination of these two lemmas finds an input that can be queried first in an optimal verification policy for all function values. Lemmas 31 and 32 are the main tools for the proof of Lemmas 33 and 34 and are stated later.

Every policy starts by querying a fixed input X_i and the next query is a function of the value of X_i. An optimal policy is called *second-input-fixed* if the second input queried is some fixed X_j independent of the value of X_i. On the other hand, a policy may query X_i first and then X_j or X_k next depending on whether $X_i = 0$ or $X_i = 1$. Such a policy is called *second-input-varies*.

The next lemma is about the second-input-fixed optimal policies. Suppose there is a function for which there exists a second-input-fixed optimal policy, that

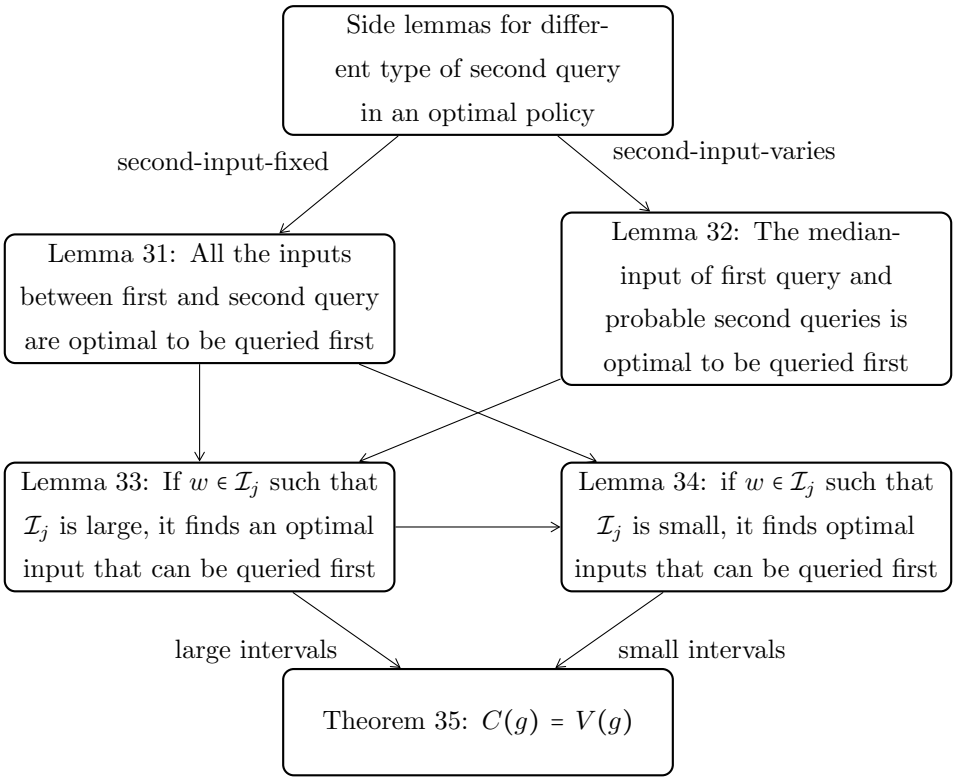


Figure 5.1: Proof scheme for Theorem 35

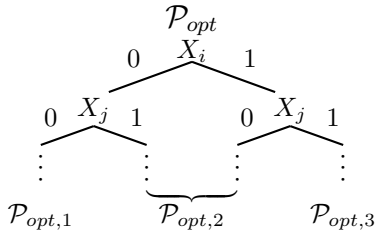


Figure 5.2: Second-input-fixed policy

queries some two inputs X_i and X_j. Using the fact that the input probabilities are sorted and the function is symmetric, we show that for any index k between i and j there exists an optimal policy that queries X_k first.

Lemma 31. *Let $\mathcal{P}_{opt}$ be a second-input-fixed optimal policy (verification or computation) that queries X_i and X_j as the first two inputs. Then for any k between i and j (inclusive), there exists an optimal policy that queries X_k first.*

Proof. Figure 5.2 describes $\mathcal{P}_{opt}$ for computing the value of g or verifying it. Let $\mathcal{P}_{opt,1}, \mathcal{P}_{opt,2}$, and $\mathcal{P}_{opt,3}$ be policies followed after querying of X_i and X_j, depending on the values observed. Since the function is symmetric, when $X_i + X_j = 1$, the same policy works. Without loss of generality, we assume that $i < j$. For any $k \in [i, j]$, consider four policies that are shown in Figure 5.3 and note that they query X_k first. We briefly explain these four policies.

The first policy queries X_k and then X_i. It then follows $\mathcal{P}_{opt}$ and queries X_j when $\mathcal{P}_{opt}$ requires querying X_k. In the second policy X_k is queried first, and depending on its value we query X_i or X_j and follow $\mathcal{P}_{opt}$. When $\mathcal{P}_{opt}$ requires querying X_k, we query one of X_i and X_j that has not been queried yet. The remaining two policies are similar.

We relate the expected query complexities of the various policies described above and show that at least one of the four policies defined, performs as well as $\mathcal{P}_{opt}$.

Using the linearity of expectation, the complexity can be written as

$$\begin{aligned} C(\mathcal{P}_{opt}) =& \bar{p}_i \bar{p}_j (a_{00}\bar{p}_k + b_{00}p_k) + \bar{p}_i p_j (a_{01}\bar{p}_k + b_{01}p_k) + \\ & p_i \bar{p}_j (a_{01}\bar{p}_k + b_{01}p_k) + p_i p_j (a_{11}\bar{p}_k + b_{11}p_k), \end{aligned}$$

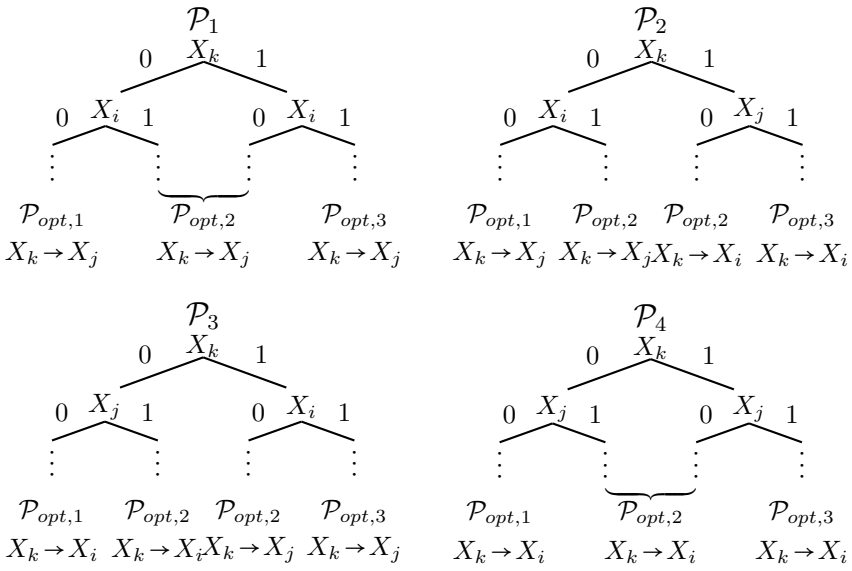


Figure 5.3: Candidates for the optimal policy if second input is fixed

where $a_{r,t}$ and $b_{r,t}$ are non-negative numbers depending on the structure of $\mathcal{P}_{opt}$, for $r, t \in \{0, 1\}$, independent of p_i, p_j and p_k. The complexity of the four defined policies can be written as,

$$\begin{aligned}
C(\mathcal{P}_1) =& \bar{p}_k\bar{p}_i(a_{00}\bar{p}_j + b_{00}p_j) + \bar{p}_k p_i(a_{01}\bar{p}_j + b_{01}p_j) + \\
& p_k\bar{p}_i(a_{01}\bar{p}_j + b_{01}p_j) + p_k p_i(a_{11}\bar{p}_j + b_{11}p_j), \\
C(\mathcal{P}_2) =& \bar{p}_k\bar{p}_i(a_{00}\bar{p}_j + b_{00}p_j) + \bar{p}_k p_i(a_{01}\bar{p}_j + b_{01}p_j) + \\
& p_k\bar{p}_j(a_{01}\bar{p}_i + b_{01}p_i) + p_k p_j(a_{11}\bar{p}_i + b_{11}p_i), \\
C(\mathcal{P}_3) =& \bar{p}_k\bar{p}_j(a_{00}\bar{p}_i + b_{00}p_i) + \bar{p}_k p_j(a_{01}\bar{p}_i + b_{01}p_i) + \\
& p_k\bar{p}_i(a_{01}\bar{p}_j + b_{01}p_j) + p_k p_i(a_{11}\bar{p}_j + b_{11}p_j), \\
C(\mathcal{P}_4) =& \bar{p}_k\bar{p}_j(a_{00}\bar{p}_i + b_{00}p_i) + \bar{p}_k p_j(a_{01}\bar{p}_i + b_{01}p_i) + \\
& p_k\bar{p}_j(a_{01}\bar{p}_i + b_{01}p_i) + p_k p_j(a_{11}\bar{p}_i + b_{11}p_i).
\end{aligned}$$

Now we compare the complexity of the four new policies with the original optimal

policy by subtracting $C(\mathcal{P}_{opt})$ from them. After simplifying,

$$\begin{aligned}
C(\mathcal{P}_1) - C(\mathcal{P}_{opt}) &= \bar{p}_i(p_j - p_k)(b_{00} - a_{01}) + p_i(p_j - p_k)(b_{01} - a_{11}),\\
C(\mathcal{P}_2) - C(\mathcal{P}_{opt}) &= \bar{p}_i(p_j - p_k)(b_{00} - a_{01}) + p_j(p_i - p_k)(b_{01} - a_{11}),\\
C(\mathcal{P}_3) - C(\mathcal{P}_{opt}) &= \bar{p}_j(p_i - p_k)(b_{00} - a_{01}) + p_i(p_j - p_k)(b_{01} - a_{11}),\\
C(\mathcal{P}_4) - C(\mathcal{P}_{opt}) &= \bar{p}_j(p_i - p_k)(b_{00} - a_{01}) + p_j(p_i - p_k)(b_{01} - a_{11}).
\end{aligned}$$

By assumption, input probabilities are sorted in decreasing order, hence $p_i \geq p_k \geq p_j$. If any of p_i, p_j, and p_k are equal, the result follows trivially. Assume that they are all different. In the four equations above, observe that the coefficients multiplied by $(b_{00} - a_{01})$ and $(b_{01} - a_{11})$ take all possible negative and positive signs. Therefore, either all the equations are zero or at least one of them is negative. Hence, at least one of the four policies performs as well as the optimal policy. □

Next we consider second-input-varies policies. Suppose a policy that queries X_i first and then X_j or X_k depending on whether $X_i = 0$ or $X_i = 1$. For X_i, X_j, X_k the median index is the median of $\{i, j, k\}$. We call the corresponding input as the *median-input*.

Lemma 32. *For the second-input-varies policy defined above, there exists an optimal policy that queries the median-input of $\{X_i, X_j, X_k\}$ first.*

Proof. If $j < i < k$ or $k < i < j$, then by definition, the optimal policy queries the median-input, X_i, first. We consider the case when $i > \max\{j, k\}$. The case when $i < \min\{j, k\}$ is similar.
Figure 5.4 describes an optimal policy $\mathcal{P}_{opt}$. As in the proof of Lemma 31,

$$\begin{aligned}
C(\mathcal{P}_{opt}) =& \bar{p}_i\bar{p}_j(\bar{p}_k a_{00} + p_k b_{00}) + \bar{p}_i p_j(\bar{p}_k a_{01} + p_k b_{01}) +\\
& p_i\bar{p}_k(\bar{p}_j a_{10} + p_j b_{10}) + p_i p_k(\bar{p}_j a_{11} + p_j b_{11}).
\end{aligned}$$

Since $\mathcal{P}_{opt}$ is optimal, changing $\mathcal{P}_{opt,2}$ to $\mathcal{P}_{opt,3}$ in Figure 5.4 should not decrease the complexity. As a result,

$$\bar{p}_k a_{01} + p_k b_{01} \leq \bar{p}_k a_{10} + p_k b_{10}. \tag{5.1}$$

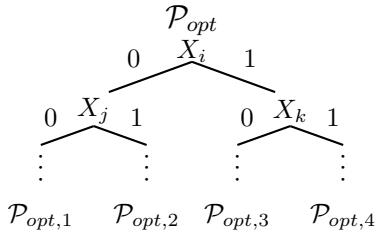


Figure 5.4: Second-input-varies policy

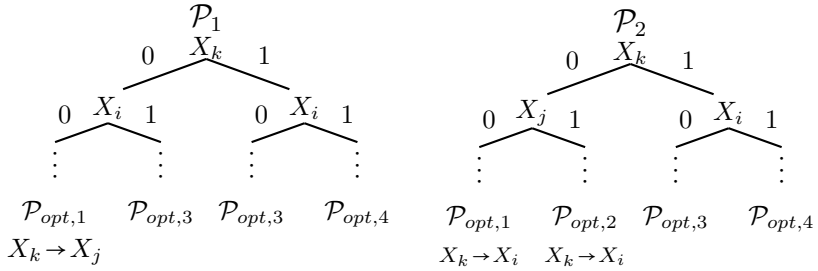


Figure 5.5: Candidates for the optimal policy if second input varies, $j < k$

Similarly,

$$\bar{p}_j a_{10} + p_j b_{10} \leq \bar{p}_j a_{01} + p_j b_{01}. \tag{5.2}$$

Consider the following two cases. For both of them, we show that there exists a policy that queries the median-input first.

Case 1 ($j < k < i$) We consider the policies in Figure 5.5 and show that at least one of them is as good as $\mathcal{P}_{opt}$. Note that both of these policies query the median-input first. By the linearity of expectation, the expected query complexity of the two policies in Figure 5.5 are

$$\begin{aligned}
C(\mathcal{P}_1) =& \bar{p}_k\bar{p}_i(\bar{p}_j a_{00} + p_j b_{00}) + \bar{p}_k p_i(\bar{p}_j a_{10} + p_j b_{10}) + \\
& p_k\bar{p}_i(\bar{p}_j a_{10} + p_j b_{10}) + p_k p_i(\bar{p}_j a_{11} + p_j b_{11}), \\
C(\mathcal{P}_2) =& \bar{p}_k\bar{p}_j(\bar{p}_i a_{00} + p_i b_{00}) + \bar{p}_k p_j(\bar{p}_i a_{01} + p_i b_{01}) + \\
& p_k\bar{p}_i(\bar{p}_j a_{10} + p_j b_{10}) + p_k p_i(\bar{p}_j a_{11} + p_j b_{11}).
\end{aligned}$$

We now show that either $C(\mathcal{P}_1) \leq C(\mathcal{P}_{opt})$ or $C(\mathcal{P}_2) \leq C(\mathcal{P}_{opt})$.

If $C(\mathcal{P}_1) > C(\mathcal{P}_{opt})$,

$$p_j\bar{p}_k\bar{p}_i b_{00} + \bar{p}_j p_k\bar{p}_i a_{10} + p_j p_k\bar{p}_i b_{10} > p_j\bar{p}_k\bar{p}_i a_{01} + \bar{p}_j p_k\bar{p}_i b_{00} + p_j p_k\bar{p}_i b_{01}.$$

After applying Equation (5.2),

$$p_j\bar{p}_k\bar{p}_i b_{00} + \bar{p}_j p_k\bar{p}_i a_{01} > p_j\bar{p}_k\bar{p}_i a_{01} + \bar{p}_j p_k\bar{p}_i b_{00}.$$

And since $p_j > p_k$,

$$b_{00} > a_{01}. \tag{5.3}$$

If $C(\mathcal{P}_2) > C(\mathcal{P}_{opt})$,

$$\begin{aligned}\bar{p}_j\bar{p}_k p_i b_{00} + p_j\bar{p}_k p_i b_{01} + \bar{p}_j p_k\bar{p}_i a_{10} + p_j p_k\bar{p}_i b_{10} > \\ \bar{p}_j\bar{p}_k p_i a_{10} + p_j\bar{p}_k p_i b_{10} + \bar{p}_j p_k\bar{p}_i b_{00} + p_j p_k\bar{p}_i b_{01}.\end{aligned}$$

This equation can be rewritten as

$$\bar{p}_j\bar{p}_k p_i b_{00} + p_j\bar{p}_k p_i b_{01} + (p_k\bar{p}_i - p_i\bar{p}_k)(\bar{p}_j a_{10} + p_j b_{10}) > \bar{p}_j p_k\bar{p}_i b_{00} + p_j p_k\bar{p}_i b_{01}.$$

Since $p_k\bar{p}_i - p_i\bar{p}_k \geq 0$, using Equation (5.2), we can simplify the above expression as

$$\bar{p}_j\bar{p}_k p_i b_{00} + p_j\bar{p}_k p_i b_{01} + (p_k\bar{p}_i - p_i\bar{p}_k)(\bar{p}_j a_{01} + p_j b_{01}) > \bar{p}_j p_k\bar{p}_i b_{00} + p_j p_k\bar{p}_i b_{01}.$$

By simplifying,

$$(p_k\bar{p}_i - p_i\bar{p}_k)\bar{p}_j a_{01} > (p_k\bar{p}_i - p_i\bar{p}_k)\bar{p}_j b_{00}.$$

Since $p_k\bar{p}_i - p_i\bar{p}_k \geq 0$, we have $a_{01} > b_{00}$, contradicting Equation (5.3).

Case 2 $(k < j < i)$ Inequalities (5.1) and (5.2) still hold. As before we have two policies in Figure 5.6 and show that at least one of them is as good as our optimal policy. By the linearity of expectation, the expected query complexity of the two policies are

$$\begin{aligned}C(\mathcal{P}_1) =& \bar{p}_j\bar{p}_i(\bar{p}_k a_{00} + p_k b_{00}) + \bar{p}_j p_i(\bar{p}_k a_{01} + p_k b_{01}) + \\ & p_j\bar{p}_i(\bar{p}_k a_{01} + p_k b_{01}) + p_j p_i(\bar{p}_k a_{11} + p_k b_{11}), \\ C(\mathcal{P}_2) =& \bar{p}_j\bar{p}_i(\bar{p}_k a_{00} + p_k b_{00}) + \bar{p}_j p_i(\bar{p}_k a_{01} + p_k b_{01}) + \\ & p_j\bar{p}_k(\bar{p}_i a_{10} + p_i b_{10}) + p_j p_k(\bar{p}_i a_{11} + p_i b_{11}).\end{aligned}$$

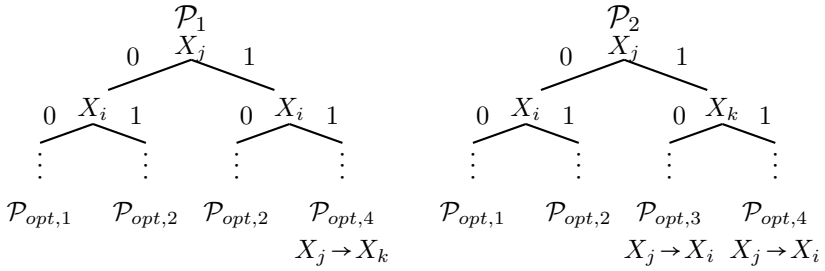


Figure 5.6: Candidates for the optimal policy if second input varies, $j > k$

If $C(\mathcal{P}_1) > C(\mathcal{P}_{opt})$,

$$\bar{p}_j\bar{p}_k p_i a_{01} + \bar{p}_j p_k p_i b_{01} + p_j\bar{p}_k p_i a_{11} > \bar{p}_j\bar{p}_k p_i a_{10} + \bar{p}_j p_k p_i a_{11} + p_j\bar{p}_k p_i b_{10}.$$

This equation can be rewritten as

$$\bar{p}_j\bar{p}_k a_{01} + \bar{p}_j p_k b_{01} - \bar{p}_j\bar{p}_k a_{10} > \bar{p}_j p_k a_{11} + p_j\bar{p}_k b_{10} - p_j\bar{p}_k a_{11}.$$

By subtracting $\bar{p}_j p_k b_{10}$ from both sides,

$$\bar{p}_j\left(\bar{p}_k a_{01} + p_k b_{01} - \bar{p}_k a_{10} - p_k b_{10}\right) > -\left(\bar{p}_j p_k - p_j\bar{p}_k\right)\left(b_{10} - a_{11}\right). \tag{5.4}$$

Let $E = \bar{p}_k a_{10} + p_k b_{10} - \bar{p}_k a_{01} - p_k b_{01}$. Equation (5.1) shows that $E \geq 0$. Observe that $\bar{p}_j p_k - p_j\bar{p}_k = p_k - p_j > 0$. Therefore, Equation (5.4) can be rewritten as

$$(p_k - p_j)(b_{10} - a_{11}) > \bar{p}_j \cdot E. \tag{5.5}$$

If $C(\mathcal{P}_2) > C(\mathcal{P}_{opt})$,

$$\begin{aligned}\bar{p}_j\bar{p}_k p_i a_{01} + &\bar{p}_j p_k p_i b_{01} + p_j\bar{p}_k\bar{p}_i a_{10} + p_j p_k\bar{p}_i a_{11} > \\ &\bar{p}_j\bar{p}_k p_i a_{10} + \bar{p}_j p_k p_i a_{11} + p_j\bar{p}_k\bar{p}_i a_{01} + p_j p_k\bar{p}_i b_{01},\end{aligned}$$

which can be rewritten as

$$\left(p_j\bar{p}_i - p_i\bar{p}_j\right)\left(\bar{p}_k a_{10} - \bar{p}_k a_{01} - p_k b_{01}\right) > -p_k(p_j - p_i)a_{11}.$$

Observe that $p_j\bar{p}_i - p_i\bar{p}_j = p_j - p_i > 0$. By adding $(p_j - p_i)p_k b_{10}$ to both sides of the equation and substituting $E = \bar{p}_k a_{10} + p_k b_{10} - \bar{p}_k a_{01} - p_k b_{01}$,

$$E > p_k(b_{10} - a_{11}). \tag{5.6}$$

Since $p_k > p_j$, after combining equation (5.5) and (5.6),

$$\begin{aligned} E &> \frac{p_k \bar{p}_j}{p_k - p_j} E \\ &= \frac{p_k - p_j p_k}{p_k - p_j} E \\ &> E, \end{aligned}$$

contribution. □

Recall that we divide the intervals into two types of small and large intervals. Lemmas 33 and 34 consider the first queried input when the input's weight lies in large and small interval respectively.

Lemma 33. *Suppose $w(\mathbf{X})$ belongs to a large interval with size L. There exists an optimal policy that queries some X_i first where $i \in [n - L + 1, L]$, and verifies the value of $g(w)$.*

Proof. Suppose $w(\mathbf{X}) \in \mathcal{I}_m$ where $\mathcal{I}_m$ is a large interval. Let $\mathcal{L}_{n,L} = \{X_{n-L+1}, \ldots, X_L\}$. Our goal is to show that there exists an optimal verification policy that queries some $X_i \in \mathcal{L}_{n,L}$ first. The proof is based on induction on n and first we consider some corner cases.

Consider a corner case when $n \in \mathcal{I}_m$. Then verifying the value of $g(w)$ is same as verifying $w(\mathbf{X}) \geq n - L + 1$ or $\Pi_{n-L+1}(w) = 1$. Lemma 30 shows that X_{n-L+1} can be queried first in an optimal verification policy and $X_{n-L+1} \in \mathcal{L}_{n,L}$. Hence the lemma holds for this corner case. The next corner case is when $0 \in \mathcal{I}_m$ and similarly, we can show that the lemma holds in this case. Next we prove the induction when 0 and n are not inside $\mathcal{I}_m$.

By induction, suppose the lemma holds for any function g' with $n' = n - 1$ inputs. Observation 25 shows that at least $n - L + 1$ inputs need to be queried. Since $0 \notin \mathcal{I}_m$ and $n \notin \mathcal{I}_m$, L is smaller than n and at least two inputs need to be queried.

Let $\mathcal{P}_{opt}$ be an optimal policy that queries X_i first. If $X_i \in \mathcal{L}_{n,L}$ then the induction is complete. If $X_i \notin \mathcal{L}_{n,L}$, then either $i > L$ or $i < n - L + 1$. We consider the case, when $i > L$ and the case $i < n - L + 1$ is similar.

Let $\mathbf{X}'$ be all the inputs except X_i. Verifying $w(\mathbf{X}) \in \mathcal{I}_m$ is same as verifying $w(\mathbf{X}') + X_i \in \mathcal{I}_m$. Recall that 0 and n are not inside $\mathcal{I}_m$. Therefore, after querying X_i, verifying $w(\mathbf{X}') + X_i \in \mathcal{I}_m$ is same as verifying $w(\mathbf{X}') \in \mathcal{I}'_m$ for some $\mathcal{I}'_m$ such that $|\mathcal{I}'_m| = L$.

Using induction, there is an optimal policy $\mathcal{P}'_{opt}$ that verifies $w(\mathbf{X}) \in \mathcal{I}_m$ and queries X_i first and then based on its value, it chooses one of the inputs inside $\mathcal{L}_{n-1,L}$ for the second query. Either the policy $\mathcal{P}'_{opt}$ is second-input-fixed or second-input-varies and we consider them separately.

If $\mathcal{P}'_{opt}$ is second-input-fixed, then suppose it queries X_j as the second input such that $X_j \in \mathcal{L}_{n-1,L}$. By Lemma 31, any member of $\{X_j, \ldots, X_i\}$ can be queried first in an optimal policy and $\{X_j, \ldots, X_i\} \bigcap \mathcal{L}_{n,L} \neq \varnothing$. If the policy is second-input-varies, then suppose it queries X_j if $X_i = 0$, and X_k if $X_i = 1$, such that $X_j, X_k \in \mathcal{L}_{n-1,L}$. Then, by Lemma 32, there exists an optimal policy which queries the median-input of $\{X_i, X_j, X_k\}$ first which is an input inside $\mathcal{L}_{n,L}$. Hence the lemma is proved for $i > L$ and similar proof holds for $i < n - L + 1$. □

Lemma 34. *Suppose $w(\mathbf{X})$ belongs to a small interval with size ℓ, then querying any one of $\{X_\ell, \ldots, X_{n-\ell+1}\}$ first, is optimal.*

Proof. The proof is based on induction on n and is similar to the proof of Lemma 33. We explain the general scheme of the proof.

Suppose an optimal policy that queries X_i first. The value of i can belong to three distinct sets, $[1, \ell-1]$, $[\ell, n-\ell+1]$, and $[n-\ell+2, n]$ and we consider these cases separately. For all these cases, by using Lemmas 31, 33 and induction, we can show that $\{X_\ell, \ldots, X_{n-\ell+1}\}$ are eligible choices to be queried first in an optimal policy. □

The following theorem is the main contribution of this part. It states that for all symmetric functions of independent binary inputs, there exists an optimal computation policy that is also an optimal verification policy.

Theorem 35. $C(g) = V(g)$.

Proof. Observe that all symmetric functions have at most one large interval except, $\Pi_{\frac{n+1}{2}}(x)$ for odd n. For this exception, the above theorem is proved in Section 5.1 and in the rest of the proof we assume that there is at most one large interval.

Let $\mathcal{A}_j$ be the set of inputs that can be queried first in the optimal verification policy when $w \in \mathcal{I}_j$. It suffices to show that $\bigcap_j \mathcal{A}_j \neq \varnothing$ since it implies that there is an input that can be queried first in an optimal verification policy for all function values. After the first query, the similar argument can be used for the rest of the inputs and we can *track* the optimal verification policies without knowing function value.

Let $\ell_j \stackrel{\text{def}}{=} |\mathcal{I}_j|$ and k be the index of the longest interval. If there is more than one longest interval, choose one of them arbitrary. We consider two cases, $\ell_k < \frac{n+1}{2}$ and $\ell_k \geq \frac{n+1}{2}$ separately. For both of these cases, we show that $\bigcap_j \mathcal{A}_j \neq \varnothing$.

- $\ell_k < \frac{n+1}{2}$, hence $\mathcal{I}_k$ is small interval. By Lemma 34, for all j,

  $$\{X_{\ell_j}, \ldots, X_{n-\ell_j+1}\} \subset \mathcal{A}_j.$$

  Hence, $\{X_{\ell_k}, \ldots, X_{n-\ell_k+1}\} \subset A_j$ and $\bigcap_j \mathcal{A}_j \neq \varnothing$.

- $\ell_k \geq \frac{n+1}{2}$, hence $\mathcal{I}_k$ is large interval. By Lemma 34, $\forall j \neq k$,

  $$\{X_{\ell_j}, \ldots, X_{n-\ell_j+1}\} \subset \mathcal{A}_j.$$

  Observe that,

  $$\{X_{n-\ell_k+1}, \ldots, X_{\ell_k}\} \subset \{X_{\ell_j}, \ldots, X_{n-\ell_j+1}\}.$$

  Therefore, $\forall j \neq k, \{X_{n-\ell_k+1}, \ldots, X_{\ell_k}\} \subset \mathcal{A}_j$. By Lemma 33,

  $$\{X_{n-\ell_k+1}, \ldots, X_{\ell_k}\} \bigcap \mathcal{A}_k \neq \varnothing.$$

  Hence $\bigcap_j \mathcal{A}_j \neq \varnothing$.

Hence the theorem is proved. □

From the proof of Theorem 35, we have the following observation. Let $\ell_{\max} \stackrel{\text{def}}{=} \max_j |\mathcal{I}_j|$.

Observation 36. *If* $\ell_{\max} < \frac{n+1}{2}$*, an optimal computation policy queries all the inputs* $X_{\ell_{\max}}, \ldots, X_{n-\ell_{\max}+1}$ *and if* $\ell_{\max} = \frac{n+1}{2}$*, it queries* $X_{\ell_{\max}}$*.*

Observation 36 can be used to find an optimal computation policy for a function with small $\ell_{\max}$. In particular, if $\ell_{\max}$ is small, using Observation 36, we reduce the number of inputs from n to $2\ell_{\max} - 2$. For a smaller number of inputs, it is possible to find the optimal computation policy by exhaustive search. In the following, we give an example to show how this result can be used in order to find an optimal computation policy.

Example 37. *In this example, we find the optimal computation policy for functions with* $\ell_{max} \leq 2$*. Observation 36 shows that the optimal computation policy should query all the inputs* $X_2, \ldots, X_{n-1}$*. As a result, only* X_1 *and* X_n *are left to be queried and the optimal computation policy for any symmetric function with two inputs is trivial.*

5.3 Functions with different verification and computation

Section 5.2 shows that computation and verification complexities are same for symmetric functions of independent binary inputs. We show that the symmetry, independence, and binary restrictions are necessary, and relaxing any of them may result in functions whose verification complexity is strictly lower than their computation complexity.

5.3.1 Non-symmetric Functions

The following example is a non-symmetric function of independent binary inputs that has different verification and computation complexities.

Example 38. *Let* X_1, X_2*, and* X_3 *be independent* bernoulli$(\frac{1}{2})$ *random variables. Consider the non-symmetric function* $f(\mathbf{x})$ *defined in Table 5.1. We show that its*

verification complexity is,

$$V(f) = \sum_{j=1}^{4} V_j(f) p(f(\mathbf{X}) = j) = \frac{2}{4} + \frac{2}{4} + \frac{2}{4} + \frac{3}{4} = \frac{9}{4}.$$

Table 5.1: Function in Example 38

$x_1x_2x_3$	000	001	100	110	011	111	010	101
$f(\mathbf{x})$	1	1	2	2	3	3	4	4

The first three terms correspond to function values 1, 2, and 3, each occurring with probability 1/4 *and verifying them requires to query exactly two inputs. The fourth term corresponds to function value 4. In that case, querying all the three inputs is necessary to verify the function value.*

On the other hand, we show that the computation complexity is $\frac{5}{2}$*. Suppose we start with querying* X_1 *as the first input. Querying* X_2 *or* X_3 *has a similar result. Based on the* X_1 *value, the function can be rewritten as in Table 5.2.*

Table 5.2: Induced functions in Example 38

$x_1 = 0$

x_2x_3	00	01	11	10
$f(\mathbf{x})$	1	1	3	4

$x_1 = 1$

x_2x_3	00	10	11	01
$f(\mathbf{x})$	2	2	3	4

The optimal computation policy that queries X_1 *first is shown in Figure 5.7. A simple calculation shows that the expected query complexity is* $\frac{5}{2}$*. As a result, computation and verification complexity are different for this example.*

Example 38 has a non-binary output. Next, we consider a binary-input and output function where the inputs are independent and the verification and computation complexities are different.

Example 39. *Let* X_1, X_2*, and* X_3 *be independent* bernoulli($\frac{1}{2}$) *and* X_4 *be* bernoulli($\frac{1}{10}$)*. Consider the non-symmetric function* $f(\mathbf{x})$ *defined in Table 5.3*

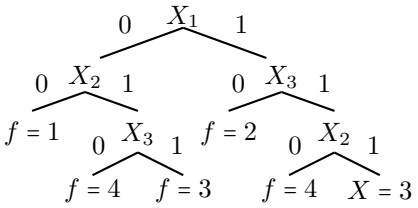


Figure 5.7: Optimal computation policy for Example 38

Table 5.3: Function in Example 39

$x_1x_2x_3x_4$	0000	1000	0100	1100	0010	1010	0110	1110
$f(\mathbf{x})$	0	0	1	1	0	1	0	1
$x_1x_2x_3x_4$	0001	1001	0101	1101	0011	1011	0111	1111
$f(\mathbf{x})$	1	0	1	1	1	0	1	1

With exhaustive search, we can show that if $f = 0$, then an optimal verification policy should *query X_4 first, however if $f = 1$ querying X_3 before X_4 results in a smaller expected query complexity. As a result, different function values imply different set of inputs to be queried first in an optimal verification policy. Therefore, computation and verification complexity are different for this example.*

5.3.2 Functions with dependent inputs

The next example is of a symmetric binary-input function that with dependent inputs has different verification and computation complexities.

Example 40. *Consider the symmetric binary function and input probabilities shown in Table 5.4, where ϵ is a small positive constant so we can ignore its effect on our calculation.*

If $f(\mathbf{X}) = 0$, in an optimal verification policy, all the inputs other than X_4 are eligible to be queried first. However, if $f(\mathbf{X}) = 1$, We can show that the first two steps of an optimal verification policy have to follow Figure 5.8. Therefore, based on the function value, an optimal verification policy should query different set of inputs first. Hence the verification and computation complexities are different.

Table 5.4: Function and input probabilities in Example 40

$f(\mathbf{x}) = 1$	$x_1x_2x_3x_4$	1100	1010	0110	0101	0011	1001
	$p(\mathbf{x})$	ϵ	ϵ	q	q	q	$3q$
$f(\mathbf{x}) = 1$	$x_1x_2x_3x_4$	1110	1101	1011	0111	1111	
	$p(\mathbf{x})$	ϵ	ϵ	ϵ	ϵ	ϵ	
$f(\mathbf{x}) = 0$	$x_1x_2x_3x_4$	0000	1000	0100	0010	0001	
	$p(\mathbf{x})$	ϵ	ϵ	ϵ	ϵ	q	

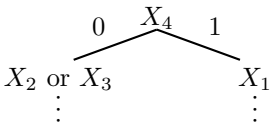


Figure 5.8: Optimal verification policy in Example 40 when $f(\mathbf{X}) = 1$

5.3.3 Non-binary inputs

Our last example is a symmetric function of ternary independent inputs where the verification and computation complexities are different.

Example 41. *Let four independent ternary random variables be distributed over ternary alphabet according to Table 5.5. And consider the symmetric function of the variable sum*

$$f(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_{i=1}^{4} x_i \leq 2, \\ 1 & \text{if } 3 \leq \sum_{i=1}^{4} x_i \leq 5, \\ 0 & \text{if } 6 \leq \sum_{i=1}^{4} x_i. \end{cases}$$

The minimum number of inputs to verify $f(\mathbf{X}) = 0$, namely $\sum_{i=1}^{4} X_i \leq 2$, is 3 and the optimal verification policy will query X_2, X_3 or X_4 as a first input but not X_1, since X_1 has the smallest probability of zero among the inputs. Similarly, If $\sum_{i=1}^{4} x_i \geq 6$ the optimal verification policy will query X_1, X_3 or X_4 as a first input but not X_2. With exhaustive search, we can show that if $3 \leq \sum_{i=1}^{4} x_i \leq 5$ the optimal verification policy should query X_1 first. Since the first queries for different function values does not intersect, verification and computation complexities are different.

Table 5.5: Input probabilities in Example 41

	0	1	2
x_1	2ϵ	$\frac{1}{2}-\epsilon$	$\frac{1}{2}-\epsilon$
x_2	$\frac{1}{2}-\epsilon$	$\frac{1}{2}-\epsilon$	2ϵ
x_3	$\frac{1}{2}-\epsilon$	2ϵ	$\frac{1}{2}-\epsilon$
x_4	0.1	0.7	0.2

Acknowledgements

Chapter 5 is adapted from Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Shengjun Pan, Ananda Theertha Suresh, "On the query computation and verification of functions", *Proceedings of IEEE Symposium on Information Theory (ISIT)*, 2012; and Jayadev Acharya, Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Ananda Theertha Suresh, "On the Computation and Verification Query complexity of Symmetric Functions", In preparation, 2015; The dissertation author was the primary researcher and author of these papers.

Bibliography

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

[2] Jayadev Acharya, Ashkan Jafarpour, and Alon Orlitsky. Expected query complexity of symmetric boolean functions. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 26–29. IEEE, 2011.

[3] Jayadev Acharya, Ashkan Jafarpour, Alon Orlitsky, and Ananda Theertha Suresh. Near-optimal-sample estimators for spherical gaussian mixtures. abs/1402.4746, 2014.

[4] Micah Adler, Peter Gemmell, Mor Harchol-Balter, Richard M. Karp, and Claire Kenyon. Selection in the presence of noise: The design of playoff systems. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia*, pages 564–572, 1994.

[5] Miklos Ajtai, Vitaly Feldman, Avinatan Hassidim, and Jelani Nelson. Sorting and selection with imprecise comparisons. abs/1501.02911, 2015.

[6] Andris Ambainis. Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. *Theory of Computing*, 1(1):37–46, 2005.

[7] Andris Ambainis and Ronald de Wolf. Average-case quantum query complexity. *Journal of Physics A: Mathematical and General*, 34(35):6741, 2001.

[8] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[9] Shai Ben-David, Benny Chor, and Oded Goldreich. On the theory of average case complexity. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 204–216. ACM, 1989.

[10] Yosi Benasher and Ilan Newman. Decision trees with boolean threshold queries. *Journal of Computer and System Sciences*, 51(3):495–502, 1995.

[11] Andrej Bogdanov and Luca Trevisan. Average-case complexity. *CoRR*, abs/cs/0606037, 2006.

[12] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs the method of paired comparisons. *Biometrika*, 39(3-4):324–345, 1952.

[13] Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms. January 20-22, 2008, San Francisco, California, USA*, pages 268–276, 2008.

[14] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[15] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[16] Constantinos Daskalakis, Ilias Diakonikolas, and Rocco A Servedio. Learning poisson binomial distributions. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 709–728. ACM, 2012.

[17] Constantinos Daskalakis and Gautam Kamath. Faster and sample near-optimal algorithms for proper learning mixtures of gaussians. 2014.

[18] Herbert Aron David. *The method of paired comparisons*, volume 12. DTIC Document, 1963.

[19] Luc Devroye and Gabor Lugosi. *Combinatorial Methods in Density Estimation*. Springer - verlag, New York, 2001.

[20] Anand K Dhulipala, Christina Fragouli, and Alon Orlitsky. Silence-based communication. *Information Theory, IEEE Transactions on*, 56(1):350–366, 2010.

[21] GÖSta Ekman. Weber's law and related functions. *The Journal of Psychology*, 47(2):343–352, 1959.

[22] Pascal Hennequin. Combinatorial analysis of quicksort algorithm. *Informatique théorique et applications*, 23(3):317–333, 1989.

[23] Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 881–890, 2007.

[24] Michael Kearns, Yishay Mansour, Dana Ron, Ronitt Rubinfeld, Robert E Schapire, and Linda Sellie. On the learnability of discrete distributions. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 273–282, 1994.

[25] Donald Ervin Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[26] Hemant Kowshik and PR Kumar. Optimal ordering of transmissions for computing boolean threshold functions. pages 1863–1867. IEEE, 2010.

[27] Satyaki Mahalanabis and Daniel Stefankovic. Density estimation in linear time. pages 503–512, 2008.

[28] CJH McDiarmid and Ryan B Hayward. Large deviations for quicksort. *journal of algorithms*, 21(3):476–507, 1996.

[29] Colin McDiarmid and Ryan Hayward. Strong concentration for quicksort. In *Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 27-29 January 1992, Orlando, Florida*, pages 414–421, 1992.

[30] Ashwin Nayak and Felix Wu. The quantum query complexity of approximating the median and related statistics. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 384–393. ACM, 1999.

[31] Sahand Negahban, Sewoong Oh, and Devavrat Shah. Iterative ranking from pair-wise comparisons. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2483–2491, 2012.

[32] Jelani Nelson. Personal communication. 2015.

[33] Miklos Santha. On the monte carlo boolean decision tree complexity of read-once formulae. *Random Structures & Algorithms*, 6(1):75–87, 1995.

[34] Henry Scheffe. A useful convergence theorem for probability distributions. In *The Annals of Mathematical Statistics*, volume 18, pages 434–438, 1947.

[35] Louis L Thurstone. A law of comparative judgment. *Psychological review*, 34(4):273, 1927.

[36] Leslie G. Valiant. A theory of the learnable. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 436–445, 1984.

[37] Jie Wang. Average-case computational complexity theory. *Complexity Theory Retrospective II*, pages 295–328, 1997.

[38] Ingo Wegener. *The complexity of Boolean functions.* Wiley-Teubner, 1987.

[39] Ingo Wegener. The complexity of symmetric boolean functions. In *Computation Theory and Logic, In Memory of Dieter Rödding*, pages 433–442, 1987.