2010 IEEE 51st Annual Symposium on Foundations of Computer Science

The Coin Problem, and Pseudorandomness for Branching Programs

Joshua Brody[1]
Department of Computer Science
Dartmouth College
Hanover, NH, USA
jbrody@cs.dartmouth.edu

Elad Verbin[2]
ITCS
Tsinghua University
Beijing, China
elad.verbin@gmail.com

***Abstract*—The *Coin Problem* is the following problem: a coin is given, which lands on head with probability either $1/2+\beta$ or $1/2-\beta$. We are given the outcome of n independent tosses of this coin, and the goal is to guess which way the coin is biased, and to answer correctly with probability $\geq 2/3$. When our computational model is unrestricted, the majority function is optimal, and succeeds when $\beta \geq c/\sqrt{n}$ for a large enough constant c. The coin problem is open and interesting in models that cannot compute the majority function.**

In this paper we study the coin problem in the model of *read-once width-w branching programs*. We prove that in order to succeed in this model, β must be at least $1/(\log n)^{\Theta(w)}$. For constant w this is tight by considering the recursive tribes function, and for other values of w this is nearly tight by considering other read-once AND-OR trees.

We generalize this to a *Dice Problem*, where instead of independent tosses of a coin we are given independent tosses of one of two m-sided dice. We prove that if the distributions are too close and the mass of each side of the dice is not too small, then the dice cannot be distinguished by small-width read-once branching programs.

We suggest one application for this kind of theorems: we prove that Nisan's Generator fools width-w read-once *regular* branching programs, using seed length $O(w^4 \log n \log\log n + \log n \log(1/\varepsilon))$. For $w = \varepsilon = \Theta(1)$, this seedlength is $O(\log n \log\log n)$. The coin theorem and its relatives might have other connections to PRGs. This application is related to the independent, but chronologically-earlier, work of Braverman, Rao, Raz and Yehudayoff [1].

***Keywords*-pseudorandomness; branching programs; lower bounds**

I. Introduction

Suppose you have an unfair coin that is either slightly biased toward heads, or slightly biased toward tails, and the goal is to determine in which direction the coin is biased. What is the best strategy for determining which way the coin is biased? The naive solution is to flip the coin several times and guess heads if the results of the coin flips were heads more often than tails. If the outcome is tails more often than heads, guessing tails is the right choice.

[1]Supported in part by NSF Grant CCF-0448277.

[2]This research was supported in part by the National Natural Science Foundation of China Grant 60553001, and the National Basic Research Program of China Grant 2007CB807900, 2007CB807901.

In this game, the naive solution turns out to be optimal. Furthermore, if the goal is to guess correctly most of the time, there is a well-known tradeoff between the error probability and the number of coin flips required. Specifically, if the coin is heads with probability $1/2+1/t$ or heads with probability $1/2-1/t$, then $\Theta(t^2)$ flips are needed to guess the correct bias with $2/3$ confidence. Conversely, if you are restricted to flipping the coin at most n times, and you want to be $2/3$ confident of guessing correctly, the bias must $\Omega(1/\sqrt{n})$.

There is a subtle drawback to the solution described above: in order to output whether at least half the coin flips were heads, one must count the number of heads, using $\log n$ bits of space; n could be quite large, and in such cases we would want to use less space. At a high level, we consider the following:

> **Question:** Suppose n coin flips are available, but computation is restricted to a machine M that uses only a constant amount of space. What is the smallest β such that M distinguishes a coin which lands on head with probability $1/2+\beta$ from one which lands on heads with probability $1/2-\beta$?

Several possibilities to model constant-space computation exist. In this work, we consider width-w read-once branching programs (ROBPs). These machines are just like w-state automata, except that the transition function can be different at each time-step. Specifically, read-once branching programs are layered directed acyclic graphs. There are $n+1$ layers. The first layer consists of one vertex (the start state), the last layer consists of two vertices (the accept state and the reject state), and each other layer consists of w vertices. Each vertex in layer i has two edges leaving it: one labeled '0' and the other labeled '1'; both edges go into vertices in level $i+1$. The vertices at the final layer have out-degree zero. Given an input $x \in \{0,1\}^n$ the program starts at the start state, and proceeds according to the bits of x in the natural way (the edge taken at layer i is chosen according to the bit x_i). The output of the program is 'accept' or 'reject' according to which state this walk ends at. Such machines can be thought of as performing "streaming computation" with $\log w$ bits of memory.

0272-5428/10 $26.00 © 2010 IEEE
DOI 10.1109/FOCS.2010.10

IEEE computer society

Two special classes of ROBPs are of particular interest: A *regular* read-once branching program (rROBP) is a ROBP where every state has in-degree 0 or 2. (Only the nodes in the first layer have in-degree 0.) A *permutation* read-once branching programs (pROBP) is a regular ROBP where for every state with in-degree 2, the two edges entering it are labeled '0' and '1'. These kinds of programs have appealing properties, and several recent PRG results have concentrated on them.

Read-once branching programs seem quite weak; for example, by a simple communication complexity argument it is easy to prove that they can compute the majority function only if $w \geq n$. However, these machines are not as weak as they may seem: For example, a width-3 ROBP can compute the TRIBES function, and the TRIBES function can solve the coin problem with parameter $\beta = \Theta(1/\log n)$ (see Section V). In contrast, Cover and Hellman [2] proved that an automaton would need a superconstant number of states to solve the coin problem with such β. We discuss this further in Section I-B. In the main part of this paper, we bound the power of ROBPs, by proving they cannot solve the coin problem when β is too small.

The results that we prove on the coin problem and its relatives might have various applications to the study of small-space computation, such as in the field of streaming algorithms. Furthermore, the ideas we introduce here might be useful for studying other "low" models such as AC^0, ACC^0, and low-degree polynomials, among others. (Here, by "low" model, we mean a model that does not seem to be able to compute the majority function.) Finally, the coin problem seems particularly relevant in the study of pseudorandom generators, and we present an application of our results in that field.

A. Our Results on the Coin and Dice Problems

Let $X = (X_1, \ldots, X_n)$ be a product distribution, i.e. the distribution of the coordinates is mutually independent. Similarly, let $Y = (Y_1, \ldots, Y_n)$ be a product distribution. Our task is to construct width-w ROBPs that distinguish X and Y, when for each i, X_i is close to Y_i. In the **coin problem**, each input is a random coin, i.e., X_i and Y_i take values in $\{0, 1\}$. We also consider a more general problem, where the inputs take values in $\{1, \ldots, m\}$. We think of these variables as m-sided dice, and call the problem of distinguishing such distributions the **dice problem**. In either case, we wish to determine how close X and Y can be and still be distinguishable by a width-w ROBP f. Note that we do not require coordinates to correspond to the "same" coin/die – X_i and X_j may have different distributions. Our only requirement is that the distributions are independent.

Our first theorem shows that if f distinguishes X from Y, then the statistical distance between pairs of variables (X_i, Y_i) cannot be too small.

Theorem 1. *(Main Theorem, informally stated). If a width-w read once branching program distinguishes X and Y such that $\Delta(X_i, Y_i) \leq \beta$ for all i, then $\beta = \Omega\left((\delta/\log n)^w\right)$. This assumes that for each outcome e, $\Pr[X_i = e], \Pr[Y_i = e] \geq \delta$.*

In this formulation we required that the "mass" of the sides of the dice not be too small (or, in coin problem language, that the "gap" of the coins not be around 0 or 1). As an extreme case of the coin problem where this *does not* hold, consider the coin problem where the coin in world 1 is heads with probability $1/n$, and in world 2 the coin is heads with probability zero.[1] Then, a width-2 ROBP can distinguish X and Y by computing the OR of the input variables. We thus see that branching programs can exploit small differences in probabilities to distinguish X and Y, when the probabilities themselves are small. To avoid this case, we must either require a lower bound on the masses of the elements in the coin/dice, as we did in Theorem 1, or define β based on *ratios* of probabilities, e.g., $\Pr[X_i = e]/\Pr[Y_i = e]$ instead of absolute differences, as in the following theorem.

Theorem 2. *(Lower Bound, relative version, informally stated). If a read once branching program distinguishes X from Y such that*

$$\frac{1}{1+\beta} \leq \frac{\Pr[X_i = e]}{\Pr[Y_i = e]} \leq 1 + \beta$$

for all i and all outcomes e, then

- $\beta = \Omega\left((\log n)^{-2w}\right)$ *when* $m = 2$.
- $\beta = \Omega\left((\log n)^{-3w}\right)$ *for any* m.

For the case of coins, we give an almost matching upper bound.

Theorem 3. *(Coin Problem Upper Bound). For any constant w, there exists a width-w read once branching program that distinguishes coins that are heads with probability $1/2 + O\left((\log n)^{2-w}\right)$ from coins that are heads with probability $1/2 - O\left((\log n)^{2-w}\right)$.*

A similar result holds for super-constant w. We defer the details of this to the full version.

B. Work Related to the Coin Problem

The problem of distinguishing two distributions has been widely studied in the literature, under various names. Cover and Hellman, in a series of works in the late '60s and early '70s, studied the question of how well small-space machines can distinguish between two distributions. See for example [2], [3], [4]. The problems they studied are more general than the coin problem, but their work in particular

[1] To reduce mathematical notation, we sometimes refer to coins from "world 1" and "world 2" instead of referring to the distributions X and Y that these coins might take

shows if a w-state automaton solves the coin problem with parameter β, then $w = \Omega(1/\beta)$ (see [2]). Cover also shows that constant-space ROBPs can perform significantly better than this [3]. In other works from that period Cover and Hellman studied how well *randomized* automata can distinguish between two distributions.

The recent paper of Braverman et al [1] contains an impossibility result for the coin problem on *regular* ROBPs. They prove that to solve the coin problem, w must be at least $\Omega(1/\beta)$.

Both of the above results are tight up to polynomial factors, by considering appropriate modifications of the majority function over w (or $\sqrt{w}$) variables. We omit the details here.

In the quantum world, things are radically different: Aaronson and Drucker [5] show a 3-state *quantum* ROBP that can solve the coin problem with $\beta = \Theta(1/\sqrt{n})$. Disregarding constant factors, this matches the majority function! In fact, the quantum ROBP that they give is actually a *reversible quantum automaton*; this class can be thought of as the quantum analogue of pROBPs, where in addition the transition function at each time-step is the same.

For AC^0, Aaronson [6] showed that if a depth-d AC^0 circuit solves the coin problem with parameter β, then $\beta \geq \Omega(1/\log^{d+2} n)$. Amano's work [7] constructs AC^0 circuits that can solve the coin problem; for constant depth d, his constructions solve the coin problem with parameter $\beta = \Theta(1/\log^{d-1} n)$.

The coin problem can be thought of as approximating the majority function in a restricted computational class. The approximate-majority problem is well-studied in the literature, for various notions of approximation. Some recent works on this topic are O'Donnell and Wimmer [8], and Viola [9]. Also see the references therein. We remark that the notion of approximation considered in the coin problem is of dual nature: it both concentrates on inputs that are significantly biased, and it allows mistake of $1/3$ (in the distributional sense); this typically makes proving impossibility results more *difficult* than in other notions of approximation.

C. Outline of the Proof

In this subsection, we briefly outline the proof of Theorem 1, for the case of the coin problem, where the probability of heads is either $1/2 + \beta$ or $1/2 - \beta$.

A ROBP is called *weakly monotone* if for each state s, the arrow of the 1-transition out of s always points above or equal to arrow of the 0-transition out of s. Note that such ROBPs can in fact compute non-monotone functions.[2]

In the first step of the proof of the coin theorem, we perform a monotonization step, that shows that without loss of generality we can just consider weakly monotone ROBPs (henceforth called mROBPs). To prove this, we show that for any width-w ROBP M, there is a weakly-monotone width-w ROBP M' such that if M solves the coin problem with parameter β then M' also solves the coin problem with parameter β. We now show how to get M' from M. We show this only for the problem of distinguishing a fair coin from a β-biased coin: in this case the transformation from M to M' is particularly simple. To get M' from M, we start with M, and re-order the states s of each layer from the top to the bottom, according to the probability that if we start at state s and we are in the β-biased world, then the program will answer correctly. Then, if for any state s its 0-transition goes above its 1-transition, then flip the two such that the 0-transition becomes the 1-transition and vice-versa. The new branching program is M'. M' is weakly monotone. It is easy to see that if the input comes from the fair coin, the distribution of the output did not change. Also, it is easy to see that if the input comes from the β-biased coin, then M' outputs the correct answer with at least the same probability that M did. Therefore, M' is at least as good as M.

The second step of the proof is to show that in an mROBP, for each variable x_i, either x_i is irrelevant (meaning it has no influence on the output), or there is a 0-collision at layer x_i (meaning there are two 0-edges that point to the same state), or there is a 1-collision at layer x_i (meaning there are two 1-edges that point to the same state). This is an easy combinatorial statement; the reader is referred to Lemma 9.

The third step of the proof is to prove that a random restriction with parameter $p = \Theta(1/log^{w-1} n)$ *kills* any width-w mROBP, in the sense that with high probability it turns the mROBP into a function that only depends on $O(\log n)$ variables.[3] (Note that this does not hold for ROBPs: for example, a width-2 ROBP can compute the parity function, which is not killed by random restrictions.) To show this, we perform a series of $w-1$ random restrictions with parameter $p' = 1/3 \log n$ each. (Overall this is equivalent to performing one random restriction with parameter $\Theta(1/log^{w-1} n)$.) We show that, essentially, each random restriction reduces the width of the mROBP by one. This is not strictly true and has some complications that we deal with in Section III, but we can show here a version of this argument, assuming that random variables actually behave like their expectation. Consider a relevant unfixed layer x_i, and let x_j be the first relevant unfixed layer following x_i. If random variables behave like their expectation, then between x_i and x_j there will be $3 \log n$ relevant fixed layers. By the previous

[2]Our concept of "weakly monotone ROBPs" is *not* the same as the concept of "monotone ROBPs" used by Meka and Zuckerman in [10].

[3]A random restriction with parameter p is a probabilistic operator on functions. It sets each variable independently to 1 with probability $(1-p)/2$, to 0 with probability $(1-p)/2$, and to '*' with probability p. Each variable which is set to 0 or 1 is fixed to that value and never changes, and each variable set to '*' stays unfixed. Therefore, after the restriction, we get a function of some number $k < n$ of variables; k is a random variable, its expectation is pn. The concept of random restriction was useful e.g. to prove lower bounds in AC^0, see for example [11]

paragraph, each of these layers has either a 0-collision or a 1-collision. Therefore, if we start in layer x_i and place a pebble on each state (w pebbles in total) and then we walk according to the values of the fixed variables, with probability $1 - 1/n^3$ two pebbles will collide, and by the time the pebbles arrive at layer x_j, at least one state will not get a pebble. The state that did not get a pebble will never be reached after applying the restriction, so layer x_j lost a state, and now its width is at most $w-1$. By union bounding over $\leq n$ events with probability $\leq 1/n^3$ each, we see that with probability $1/n^2$, all unfixed relevant layers lost a state, so after one random restriction the width of the mROBP is $w-1$. Continuing this argument $w-1$ times, we get that after $w-1$ random restrictions, with high probability the width of the program is 1, i.e. with high probability it is a constant function.

The argument in the last paragraph crucially relied on the "cheat" that we assume that random variables act according to their expectations. The argument above is not actually correct: for example, when performing a random restriction with parameter $1/3 \log n$, there will likely be $\Theta(n/\log^2 n)$ pairs of adjacent layers that are both unfixed. The argument in Section III deals with this kind of complications and that makes the argument significantly more complex.

Finally, the fourth step of our proof is to show that if a random restriction with parameter p of a function f turns it with high probability into a function with only k relevant variables, then f could only solve the coin problem when β is at least p/k. This is a rather straightforward argument, proved as part of the proof of Theorem 10.

To generalize this proof for the case of dice, or to the case of biases than $1/2 \pm \beta$, we use couplings instead of random restrictions. (Interestingly, couplings can be seen to be a natural generalization of random restriction in our setting.) We also need to generalize the above concepts appropriately. For the "relative" coin/dice theorem (Theorem 2), we perform a careful reduction to the non-relative case.

All of the above claims are formalized and proved in Section III.

D. Our results on PRGs

One of the main open questions in theoretical computer science is to derandomize log-space computations, namely to prove that L=RL. One approach for doing that is to construct pseudorandom generators (PRGs) for ROBPs with width $w = poly(n)$. The PRG should use seedlength $O(\log n)$, and the PRG itself should be computable in log-space. For width 2 it is known how to construct such PRGs with seedlength $O(\log n)$, see [12], but for width 3 the problem is already wide open. (See more background in Section VI).

In Section VI we take one step in this direction. We show how to achieve seedlength $O(w^4 \log n \log\log n + \log n \log(1/\varepsilon))$ for *regular* read-once branching programs of width w. Here, ε is the error parameter of the PRG. The generator we use is Nisan's generator [13] or the INW generator [14]. For $\varepsilon = O(1/\log n)$ and $w = O(1)$, the seedlength of our generator is just $O(\log n \log\log n)$, while the traditional Nisan's generator requires seedlength $O(\log^2 n)$. For more information and a more complete literature survey, see Section VI.

In Section VII we show why our techniques cannot be used to fool non-regular ROBPs, and in general, why the INW generator fails to fool even width-3 non-regular ROBPs with seed length $o(\log^2 n)$. In fact, we prove that INW with seedlength $o(\log^2 n)$ is not even a hitting set generator against width-3 ROBP.

We note that the work of Braverman et al [1] also shows that Nisan's generator fools small-width regular read-once branching programs, and their work achieve *better* seedlength than ours. The main advantage of our work is that all of Braverman et al's approach works only for regular ROBPs, while in our work, the coin and dice theorems work for all ROBPs. Thus, our work may be useful in the future in order to get PRGs even against non-regular ROBPs.

E. Organization of the Paper

In Section II, we formalize many of the concepts and tools used in the rest of the paper. Sections III and IV develop our lower bounds, and Section V constructs the upper bound. In Section VI we show our results on PRGs for rROBPs. Section VII explains the barrier stopping the PRG results from giving PRGs for ROBPs. Finally, Section VIII concludes the paper and presents some open problems.

Due to space restrictions, we have had to omit considerable parts of the paper from this version, most notably the later sections which deal with PRGs. We recommend the reader to read the full version, available online.

II. Preliminaries and Notation

In this section, we provide some technical background and concepts needed in the rest of the paper.

Definition 4 (Statistical Distance)**.** *Let X and Y be random variables that both take values on a finite set $\mathcal{V}$. The* statistical distance between X and Y *is defined as*

$$\Delta(X,Y) := \frac{1}{2} \sum_{v \in \mathcal{V}} |\Pr[X = v] - \Pr[Y = v]| \ .$$

We now formalize what it means for a branching program to distinguish two distributions. The branching programs we consider take as input n-bit strings, which correspond to n coin flips. We consider distributions $X = (X_1, \ldots, X_n)$ where the X_i are mutually independent. In the coin problem, all X_i come from the domain $\{0,1\}$; in the dice version, variables come from some finite domain $[m] := \{1, \ldots, m\}$.

We say that f distinguishes X and Y if $\Delta(f(X), f(Y)) > 1/3$. A branching program f *distinguishes* β if f distinguishes distributions X, Y such that $\Delta(X_i, Y_i) \leq \beta$ for all i. Width-w branching programs distinguish β if there exists

a width-w branching program that distinguishes β. Our goal is to determine the smallest β distinguishable by width-w branching programs.

Note that this definition of distinguishing two distributions is weaker than the version described in the introduction—if we output 1 (i.e. accept) with probability at least $2/3$ when the inputs come from X and at most $1/3$ when the inputs come from Y, then necessarily we have $\Delta(f(X), f(Y)) > 1/3$. This weaker notion of distinguishability only makes our lower bounds stronger. The upper bound we present uses the more robust notion; it creates a branching program that outputs 1 with probability $> 2/3$ when inputs are from world 1, and it outputs 1 with probability $< 2/3$ when inputs come from world 2.

Finally, we define several concepts that we'll use to analyze branching programs. For any state s and any $e \in [m]$, let $s(e)$ denote the state reached by following the e-transition from s.

For any branching program f and any input x, we define $f(x) := 1$ if f accepts x, and $f(x) := 0$ if f rejects. For a random variable X and state s, let $f(X|s)$ denote the expected output of f given X, conditioned on the event that we reach state s. Also, let $\beta_X(s) := \Pr_X[f \text{ reaches } s]$. Define the *support of f at level k given X* to be the set of states at level k that are reachable from the start state, given X. Usually, both f and X will be clear from context; in this case, we say "the support of level k".

The probabilities $\{\beta_X(s)\}$ provide a convenient way of expressing $f(X)$ in terms of the states at a particular level. Specifically, let $s_1, \ldots, s_w$ denote the states at some arbitrary level k. Then, we have

$$f(X) = \sum_{j=1}^{w} \beta_X(s_j) f(X|s_j) \ . \tag{1}$$

Suppose that s is a state at level k. It's not hard to see that $f(X|s)$ is a convex combination of $\{f(X|s(e)) : e \in [m]\}$. Specifically, we have

$$f(X|s) = \sum_{e \in [m]} \Pr[X_i = e] f(X|s(e)) \ . \tag{2}$$

III. The Lower Bound

In this section, we prove the following theorem, which is the main result of our paper:

Theorem 5 (Main Theorem). *Suppose $X = (X_1, \ldots, X_n)$ and $Y = (Y_1, \ldots, Y_n)$ are collections of independent random variables, all on a finite set $[m]$. Further suppose that for all $e \in [m]$ and $i \in [n]$*

1) $\Pr[X_i = e] = 0$ *if and only if* $\Pr[Y_i = e] = 0$,
2) $\Pr[X_i = e], \Pr[Y_i = e] \geq \delta$ *whenever they are nonzero,*
3) $\Delta(X_i, Y_i) \leq \beta$.

Then, for all constant w, if a width-w ROBP distinguishes X from Y, then $\beta = \Omega\left((\delta/(\log n))^w\right)$.

We prove this theorem in three steps. First we prove a *collision lemma*. A branching program has the collision property if the transitions from any level k to the next level $k+1$ either form an identity permutation, or for some $e \in [m]$, the e-wires collide; that is, there are s, t such that $s(e) = t(e)$. In the collision lemma, we show that any *good* branching program has this property—if a branching program f does not have the collision property, then we can replace it with one that does, and simultaneously increase the statistical distance $\Delta(f(X), f(Y))$.[4]

In the second step, we use the coupling method, together with a an iterative sampling process—each variable x_i will be sampled with probability $1 - p$, and if a variable is sampled, what value it takes will come from the average of X_i and Y_i. We show that after conditioning on $w - 1$ such samplings, the function is a $O(\log n)$-junta with high probability.

Our final step shows that any function that depends on a few variables cannot distinguish two close distributions.

To prove the main theorem, we must surmount several technicalities. For this reason, we divide the proof into sections. In Section III-A, we establish our notion of monotonicity and prove the collision lemma. Section III-B uses the collision lemma to prove the main theorem. This proof requires a technical lemma; we sketch the proof in Section III-C.

A. The Collision Lemma

In this section, we develop our notion of monotonicity and state our collision lemma. Because of a lack of space, we defer proofs of the lemmas in this section to the full paper.

We wish to determine the smallest β such that $\Delta(f(X), f(Y)) > 1/3$. Without loss of generality, assume that the branching program attempts to accept strings from world 1, and reject when strings come from world 2. Thus we may assume that $\Pr[f(X) = 1] > \Pr[f(Y) = 1]$. It follows that $\Delta(f(X), f(Y)) = \Pr[f(X) = 1] - \Pr[f(Y) = 1]$.

For any level of the branching program, define an ordering on the states at that level in terms of $f(X|s)$. Specifically, define $s \leq t$ if and only if $f(X|s) \leq f(X|t)$. Next, label the states $s_1, \ldots, s_w$ in increasing order of $f(X|s)$. Thus, we have $f(X|s_i) \leq f(X|s_j)$ for all $i < j$. We call this the *canonical ordering* of $\{s_1, \ldots, s_w\}$. Our next lemma states that this ordering holds for Y as well.

[4] In fact, we prove several lemmas in terms of good branching programs; in all cases, the meaning of "good branching program" is similar. Specifically, we take the statement "any good branching program f has property $\mathcal{P}$" as shorthand for "if a branching program does not have property $\mathcal{P}$ then there exists a branching program g that has $\mathcal{P}$ such that $\Delta(f(X), f(Y)) \leq \Delta(g(X), g(Y))$"

Lemma 6. *In any good branching program, $s < t$ implies $f(Y|s) < f(Y|t)$.*

Next, we show that good programs obey a certain form of monotonicity. We first formalize this notion.

Definition 7. *A branching program is* weakly monotone *if for all levels i and for all $a, b \in [m]$ such that $\Pr[X_i = a] \geq \Pr[Y_i = a]$ but $\Pr[X_i = b] \leq \Pr[Y_i = b]$, it holds that $s(a) \geq s(b)$.*

Lemma 8. *All good branching programs are* weakly monotone.

Our final lemma in this section is a collision lemma. In general, only collisions among states in the support of a level concern us, since the other states are unreachable. Now, it's easy to see by the pigeonhole principle that if there are d states in the support of level k and fewer than d states in the support of level $k+1$, then some collision(s) must occur. The next lemma characterizes when collisions occur in the case that consecutive levels have the same support size.

Lemma 9 (Collision Lemma). *Let $s_1, \ldots, s_d$ and $t_1, \ldots, t_d$ denote the support of two consecutive levels of a good branching program. Suppose there are no collisions among $s_1, \ldots, s_d$. Then, the transitions from $\{s_i\}$ to $\{t_i\}$ form an identity permutation; that is, $s_i(e) = t_i$ for all $1 \leq i \leq d$ and for all $e \in [m]$.*

We call a layer *irrelevant* if the transitions from $\{s_i\}$ to $\{t_i\}$ form an identity permutation, and relevant otherwise. Thus, all relevant layers of a good branching program have collisions.

B. Proof of Main Theorem

Given distributions X, Y, define $p_{i,e} := \Pr[X_i = e]$ and $q_{i,e} := \Pr[Y_i = e]$. Let $r_{i,e} := (p_{i,e} + q_{i,e})/2$ denote the average of the probabilities $p_{i,e}$ and $q_{i,e}$, and let $\delta_{i,e} := (p_{i,e} - q_{i,e})/2$.

In this section, we prove the following theorem.

Theorem 10. *Fix $p := \delta/(1000 \log n)$ for some constant δ, and suppose that $X = (X_1, \ldots, X_n)$ and $Y = (Y_1, \ldots, Y_n)$ are each a collection of mutually independent random variables such that for all $e \in [m]$ and $i \in [n]$ the following conditions hold:*

1) *$p_{i,e} = 0$ if and only if $q_{i,e} = 0$,*
2) *$p_{i,e}, q_{i,e} \geq \delta$ whenever $p_{i,e}$ and $q_{i,e}$ are nonzero,*
3) *$\Delta(X_i, Y_i) \leq \beta$.*

If f is a good branching program, then we have $\Delta(f(X), f(Y)) \leq \beta \cdot p^{-w}$.

Our main theorem follows directly from Theorem 10.

Proof of Theorem 5. If f distinguishes β, then there exist two β-close distributions X and Y such that $\Delta(f(X), f(Y)) > 1/3$. By Theorem 10, $\Delta(f(X), f(Y)) \leq \beta \cdot p^{-w}$. Hence, we have $\beta = \Omega(p^w) = \Omega\left((\log n)^{-w}\right)$. □

We will prove Theorem 10 using a coupling method. It's well known that under any coupling ω of distributions μ, ν, we have $\Delta(X, Y) \leq \Pr_\omega[X \neq Y]$. Fix $\beta' := \beta \cdot p^{-(w-1)}$. Define distributions X' and Y' in the following iterative manner. Create a set of indices $S_1 \subseteq [n]$ by placing each $i \in S_1$ independently with probability $1-p$. For each $i \in S_1$ and each $e \in [m]$ we set $X'_i := Y'_i := e$ with probability $r_{i,e}$.

From the coordinates *not* in S_1, we sample a new set of coordinates $S_2 \subseteq [n] \setminus S_1$ in the same manner, and again we set X'_i and Y'_i for each $i \in S_2$ in the same way.

We repeat this sampling process $w-1$ times, after which we set the remaining $i \in [n] \setminus \left(\bigcup_j S_j\right)$ such that $\Pr[X'_i = e] = r_{i,e} + \delta_{i,e} p^{-w+1}$, $\Pr[Y'_i = e] = r_{i,e} - \delta_{i,e} p^{-w+1}$, and the joint distribution (X'_i, Y'_i) is distributed according to a coupling ω_2 to be defined later. Note that no matter the choice of ω_2, we have $\Delta(X'_i, Y'_i) \leq \beta'$. Simple calculations show that

$$\begin{aligned} \Pr[X'_i = e] &= (1 - p^{w-1}) r_{i,e} \\ &\quad + p^{w-1}(r_{i,e} + \delta_{i,e} p^{-w+1}) \\ &= r_{i,e} + \delta_{i,e} \\ &= p_{i,e} \,. \end{aligned}$$

In the same way, we have $\Pr[Y'_i = e] = q_{i,e}$. Hence, X and X' are equidistributed, as are Y and Y'. It's not hard to see that the same holds for $f(X)$ and $f(X')$ and for $f(Y)$ and $f(Y')$. Hence, the joint distribution (X', Y') defines a coupling for $f(X)$ and $f(Y)$. For the sake of notation, we'll now drop the superscripts, and refer to X and Y instead of X' and Y'.

Our goal now will be to bound $\Pr[f(X) \neq f(Y)]$ by iteratively conditioning on $S_1, S_2, \ldots, S_{w-1}$. We write "conditioning on S_j" as shorthand for "conditioning on both the choice of $i \in S_j$ *and* on the setting of X_i, Y_i for $i \in S_j$." If $i \notin \bigcup_{1 \leq j \leq k} S_j$, then we say that X_i remains *free* after the first k conditionings. When k is clear from context, we simply say that X_i is *free*. If X_i is not free, we say that it is *restricted*.

In the next lemma, we prove that conditioned on $S_1, \ldots, S_{w-1}$, the branching program f is likely to be an $O(\log n)$-junta.

Lemma 11 (Width Elimination). *Suppose there is a width-w branching program that is weakly monotone. Then, after conditioning on $S_1, \ldots, S_{w-1}$, the branching program is an $O(\log n)$-junta with probability greater than $1 - 1/n$.*

This lemma crucially uses the Collision Lemma; we defer this proof until Section III-C. The rest of the theorem follows from the following lemma, whose proof is a simple hybrid argument.

Lemma 12. *Suppose that g is a k-junta and $\Delta(X_i, Y_i) \leq \beta$ for all relevant variables i. Then, we have*

$$\Delta(g(X), g(Y)) \leq k\beta \ .$$

Proof of Theorem 10. Let R_1 denote the random coins used to generate the sampling $S_1, \ldots, S_{w-1}$, and let R_2 denote the rest of the random coins, i.e., the randomness used to choose X_i and Y_i for those i that remain free after the conditioning is complete. Then, we have $\Pr_\omega[f(X) \neq f(Y)] = \Pr_{R_1,R_2}[f(X) \neq f(Y)]$.

For given random strings r_1, r_2, let $\mathcal{E}(r_1, r_2)$ be the event that $f(X) \neq f(Y)$, given that $R_1 = r_1$ and $R_2 = r_2$. Next, let $h(r_1, r_2)$ be an indicator variable for the event $\mathcal{E}(r_1, r_2)$, and set $\zeta(r_1) := E_{R_2}[h(r_1, R_2)]$. Then, we have $E_{R_1}[\zeta(R_1)] = \Pr_\omega[f(X) \neq f(Y)]$. Now, let us call r_1 *good* if $\zeta(r_1) \leq 2\ E_{R_1}[\zeta(R_1)]$, and *bad* otherwise. By Markov's Inequality, at most half the r_1s are bad. By the union bound and Lemma 11, there exists a good r_1 such that conditioned on $R_1 = r_1$, the branching program is an $O(\log n)$-junta. Fix this r_1.

We now construct a branching program g on $m := np^{w-1}$ variables that distinguishes β'. Let y be an input to g. We "embed" this into an input x for f: for each free variable x_i, we assign a variable from y. Finally, let ω_2 be a coupling such that $\Delta(g(X), g(Y)) = \Pr_{\omega_2}[g(X) \neq g(Y)]$. Note that $\Pr_{R_2}[f(X) \neq f(Y) \mid R_1 = r_1] = \Pr_{\omega_2}[g(X) \neq g(Y)]$. Therefore, we have

$$\begin{aligned}
\Delta(f(X), f(Y)) &\leq \Pr_{\omega}[f(X) \neq f(Y)] \\
&\leq 2 \Pr_{\omega_2}[g(X) \neq g(Y)] \\
&= 2\Delta(g(X), g(Y)) \\
&\leq 2\beta' \log n \\
&= O\left(\beta (\log n)^w\right) \ ,
\end{aligned}$$

where the first inequality follows from the coupling of ω, the second follows from our choice of r_1, the penultimate equality comes from the choice of ω_2, and the final inequality comes from Lemma 12. □

C. Proof Sketch of Lemma 11

We give a sketch of the proof of Lemma 11 here, and leave the complete proof to the full version of the paper.

Proof Sketch. We proceed by induction. We'd like to show that each conditioning reduces the support at each level by one, so after $w-1$ conditionings, each level has support size of one. Note that if level k has support size one, then f is trivially independent of all levels preceding k, as no matter what happens, we always arrive at one particular state at level k. Hence, if we were able to prove such a result, the lemma would follow easily.

Unfortunately and unsurprisingly, the support size is only *likely* to decrease by one after each conditioning. Instead, we maintain an invariant throughout the induction process. Let $a := (50 \log n)/\delta$. Our invariant is the following:

> **Invariant:** Let x_i be any variable that remains free after conditioning on $S_1, \ldots, S_k$. If there are at least a levels j such that x_j is free and relevant and $j < i$, then with probability at least 0.9, the support of level i is at most $w - k$. [5]

We call a layer x_i *nice* after k conditionings if it has support at most $w - k$. Note that throughout this proof, we carry an implicit assumption that none of several *unlikely* events occur. We call an event *unlikely* if it happens with probability at most n^{-3}. We condition on $O(wn)$ unlikely events, so by a union bound, the probability that none of the unlikely events happen is at least $1 - O(1/n)$.

Again, we prove the lemma by induction. As a base case, consider the first conditioning. Fix an arbitrary free level x_i, and consider the a relevant levels preceding it. With probability $(1-p)^a \geq e^{-2ap} > 0.9$, all a levels become restricted. Each of these levels is relevant, so by the Collision Lemma and the assumption that $p_{i,e}, q_{i,e} \geq \delta$, each level has a collision with probability $\geq \delta$. Note that a collision at any restricted level reduces the support size of the free level following it by one. Therefore, the probability that *none* of the restricted levels have collisions is at most $(1-\delta)^a \leq e^{-\delta a} = n^{-50/\ln 2}$, which is *unlikely*. It's easy to see that this collision reduces the support size by one.

Proving the induction step follows the same logic, but there are some subtle complications. Chief among them is that layers that become restricted during e.g., the $(k+1)$th conditioning are not *guaranteed* to have support size at most $w-k$; instead, they have such support only with probability > 0.9. To surmount this obstacle, proceed as before, considering a free layer x_j, and noting that with probability > 0.9 the preceding a free layers become restricted. Note that if any two *consecutive* layers x_i, x_{i+1} are nice, then a collision at x_i decreases support by one, and if the support decreases to $w-k-1$ at any of these restricted layers, the support at the next free layer is similarly at most $w-k-1$. Since each layer is nice with probability 0.9, we expect $0.9a$ layers to be nice. Therefore, the event that $< 0.6a$ layers are nice is unlikely. With $0.6a$ nice layers, there must exist at least $0.1a$ pairs of consecutive nice layers. A collision at any of these layers reduces the support at the next free layer to $w-k-1$. The probability that none of these layers have collisions is at most $(1-\delta)^{0.1a} \leq n^{-5/\ln 2}$, which is unlikely. It follows that with probability at least 0.9, x_j has support size at most $w-k-1$ after $k+1$ conditionings, as desired.

[5] the requirement that there are a free, relevant levels preceding x_i is largely a technical one. We need it because we need a free active levels preceding x_i to show that the width of x_i is likely to decrease. More subtly, we also need it because in the full proof we cannot guarantee that the width of the first a free active levels is nice. Hence, the invariant only speaks of free levels that have a free levels preceding it. A complete argument appears in the full paper.

The final step is to show that f is an $O(\log n)$-junta. After $w-1$ conditionings, let x^* be the last layer with support size 1. Since each remaining free layer has support size 1 with probability > 0.9, we expect very few layers to follow this. Indeed, we prove that the event that more than $O(\log n)$ free layers follow x^* is unlikely. But f now depends on only these layers, hence it is a $O(\log n)$-junta with high probability. □

IV. Lower Bounds – Relative Versions

In this section, we generalize and strengthen the lower bound from the previous section to handle outcomes that occur with $o(1)$ probability. We begin with the coin problem.

Theorem 13. *Suppose $X = (X_1, \ldots, X_n)$ and $Y = (Y_1, \ldots, Y_n)$ are collections of mutually independent random variables on a finite domain $[m]$ such that*

$$\frac{1}{1+\beta} \leq \frac{\Pr[X_i = e]}{\Pr[Y_i = e]} \leq 1 + \beta \ .$$

for all $1 \leq i \leq n$ and all $e \in [m]$. Then, if a width-w ROBP distinguishes X and Y, then

- $\beta = \Omega\left((\log n)^{-2w}\right)$ *if* $m = 2$,
- $\beta = \Omega\left((\log mn)^{-3w}\right)$ *if* $m > 2$.

Proof: For the sake of brevity, we prove only the result for the coin problem here. The case of the dice problem is proved in the same way. We reduce from Theorem 5. Specifically, we take a width-w branching program that distinguishes distributions where the coins can have very low probabilities, and construct a width-$(2w)$ branching program that distinguishes β, using only slightly unbalanced coins—each of the coins we use will have probability of heads between $1/4$ and $3/4$.

Let $p_i := \Pr[X_i = 1]$ and $q_i := \Pr[Y_i = 1]$. First, we handle the case when $p_i \notin (1/n^2, 1 - 1/n^2)$ by rounding. Specifically, define a distribution $X' = (X'_1, \ldots, X'_n)$ in the following manner. For each $1 \leq i \leq n$, set

$$X'_i = \begin{cases} 1, & \text{if } p_i > 1 - 1/n^2 \,, \\ 0, & \text{if } p_i < 1/n^2 \,, \\ X_i, & \text{otherwise.} \end{cases}$$

Clearly, $\Delta(X_i, X'_i) \leq 1/n^2$, so by a hybrid argument, $\Delta(f(X), f(X')) \leq \Delta(X, X') \leq 1/n$. Define Y' in a similar fashion. By the triangle inequality, we have $\Delta(f(X), f(Y) \leq \Delta(f(X'), f(Y')) + 2/n$.

Therefore, it suffices to bound $\Delta(f(X'), f(Y'))$. For the rest of the proof, we consider this case and assume that $p_i, q_i \in (1/n^2, 1 - 1/n^2)$. Our next step is to construct for each i a width-2, depth-$O(\log n)$ ROBP C_i that uses only balanced coins and accepts strings from world 1 with probability p_i and accepts strings from world 2 with probability q_i. Then, we replace the transitions in our unbalanced ROBP with these mini-ROBPs—for each state s, we first transition

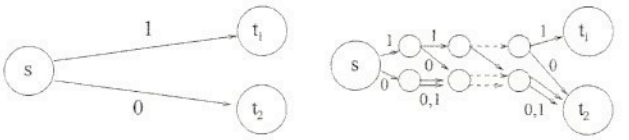


Figure 1. Replacing each $(s(0), s(1))$ pair with a width-2 balanced ROBP

to C_i, except we replace the accept and reject states in C_i with $s(1)$ and $s(0)$ respectively. The resulting branching program will have width $2w$ and length $O(n \log n)$, and the output distributions in worlds 1 and 2 will match those of the original ROBP.

We now show how to construct C_i. Assume without loss of generality that $q_i < 1/2$ and $p_i = q_i(1 + \beta)$. We construct C_i to be width-2 ROBP that uses $O(\log n)$ independent balanced coins and accepts strings from world 1 with probability p_i and accepts strings from world 2 with probability q_i. If $p_i > 1/2$, there is nothing to prove—X_i and Y_i are already balanced. Otherwise, Fix $k \geq 0$ such that $k + 1 \leq -\log p_i < k + 2$. Let C_i be the AND of $k + 1$ coins. The first k coins will be fair in both worlds 1 and world 2. The final coin will be heads with probability $2^k p_i$ in world 1 and $2^k q_i$ in world 2. It is easy to see that C_i accepts strings with probability $2^{-k} \cdot 2^k p_i = p_i$ in world 1 and $2^{-k} \cdot 2^k q_i = q_i$ in world 2.

The new branching program now has $n \cdot O(\log n) = O(n \log n)$ coins and width $2w$, and its output distributions given coins from world 1 or 2 matches those of the original ROBP. From Theorem 5 we know that if the new ROBP distinguishes β, then $\beta = \Omega\left((\log(n \log n))^{-2w}\right) = \Omega\left((\log n)^{-2w}\right)$. By reduction, the same bound holds for the original ROBP. ■

V. The Upper Bound

In this section, we construct width-w branching programs that distinguish $(1/2+\beta)$-biased coins from $(1/2-\beta)$-biased coins for some $\beta = O\left((\log n)^{-(w-2)}\right)$.

Theorem 14. *For all constant w, there exists $\beta = O\left((\log n)^{-(w-2)}\right)$ and a width-w branching program f such that, when fed a series of n independent β-biased coin flips, f accepts with probability at least $2/3$, and when fed a series of n independent $(-\beta)$-biased coin flips, f rejects with probability at least $2/3$.*

For the sake of brevity, we include only a proof sketch here and leave the full proof to the full version of the paper.

Our branching programs will compute the output of AC^0 circuits—specifically, of depth-$(w-1)$ AND-OR trees. First, we claim that width-w branching programs are able to compute depth-$(w-1)$ AND-OR trees. This is somewhat similar to a proof in [15, Sec. 5.1]. We argue this by induction. As a base case, consider the number of states

needed to compute the AND of m coins, i.e., $\wedge x_i$. A width-2 branching program can compute AND in the following manner: transition to state 2 if $x_1 = 1$ and to state 1 otherwise. For later levels, once we reach state 1, we stay there, and if we are in state 2, we transition to state 1 if $x_i = 0$ and remain in state 2 otherwise. In this way, state 1 represents the event that $x_i = 0$ for some i, and state 2 represents that $x_i = 1$ for all i so far. At the end, we reject from state 1 and accept if we remain in state 2 throughout. The OR function is computed in the same way, except that the roles of 0 and 1 are reversed.

For depth-$(w-1)$ AND-OR trees, suppose that the root node is an AND gate. Use $w-1$ states to compute each child; however, instead of accepting or rejecting, transition to state w if the subtree evaluates to 0, and transition to state 1 if the subtree evaluates to 1, allowing us to use $w-1$ states to compute the next subtree. If we ever reach state w, we reject. Otherwise, we accept.

It remains to carefully choose the number of subtrees at each level, and to show that these branching programs distinguish $1/2 \pm \beta$ coins for some $\beta = O\left((\log n)^{-(w-2)}\right)$. Our construction closely follows Amano's construction [7] of AC^0 circuits that approximate MAJ and is deferred to the full version of the paper. One interesting example is the case of $w = 3$, where the function is just the TRIBES function: it is an AND of 2^m clauses; each of the clauses is an OR of m variables. We choose m such that $m \cdot 2^m = n$. An easy computation, which we omit here, shows that this solves the coin problem with $\beta = \Omega(1/\log n)$.

VI. The INW Generator Fools rROBP with Small Seed

Let C be a set of functions from $\{0,1\}^n$ to $\{0,1\}$. For example, C might be the set of functions computable by width-w rROBPs.

Let $G : \{0,1\}^s \to \{0,1\}^n$ be another function, called the *pseudorandom generator*. G is said to *ε-fool* C, if for every $f \in C$,

$$\left|\mathbb{E}_{x\in\{0,1\}^n}[f(x)] - \mathbb{E}_{y\in\{0,1\}^s}[f(G(y))]\right| \leq \varepsilon \ .$$

In other words, G ε-fools C if for any function in the class C, the probability of f to be 1 when its input is uniform is up to ε from its probability to be 1 when its input is taken from the generator. The parameter s is called the *seed length*.

We are interested in constructing explicit generators G that fool interesting classes, and in making s as small as possible. It is well known, and easy to see, that no matter what C is, if G is taken to be a random function and $s = \Theta\left(\log\left(\frac{\log|C|}{\varepsilon^2}\right)\right)$ then with high probability, G ε-fools C. When C is the class of width-$poly(n)$ ROBP, the size of C is $2^{poly(n)}$, and thus there is a PRG that $1/poly(n)$-fools C with seed length $O(\log n)$. The challenge is to find such an explicitly-defined G, which is hopefully also computable in a low complexity class, such as LOGSPACE. Such a G will immediately imply that $L = RL$, a central outstanding open problem. The best results that were known before 2010 were that for width-2 ROBP, $O(\log n)$ seed length is achievable by using epsilon-biased generators, see e.g. [12], for an extension see [16]; and that for the class C of width-$poly(n)$ ROBP, Nisan's generator [13] can fool them with seed length $O(\log^2 n)$. An improvement/variant on Nisan's generator is the INW generator [14]. It is a famous still-open problem even to get a PRG for width-3 ROBP that surpasses Nisan's generator. For more background on pseudorandomness, see e.g. the survey by Luby and Wigderson [17].

So far in the year 2010, three works all prove that the INW generator, or Nisan's generator, fools similar restricted classes of ROBPs: Braverman et al [1] show that Nisan's generator fools rROBP even with seed length $O((\log w + \log\log n + \log(1/\varepsilon))\log n)$, and in subsequent work Koucky et al [18] show that the INW generator fools pROBP even with seed length $O((w! + \log(1/\varepsilon))\log n)$. The latter seed length is better when w is constant. In this section we show that the INW generator fools rROBPs using seed length $O(w^4 \log n(w^4 \log\log n + \log(1/\varepsilon)))$. We do this by relating it to the dice problem and then using the dice theorem.

In this section, we prove the following:

Theorem 15. *The INW generator ε-fools the class of width-w* regular *read-once branching programs of length n, with seed length $O(w^4 \log n \log\log n + \log n \log(1/\varepsilon))$ bits.*

For $\varepsilon = O(1/\log n)$ and $w = O(1)$, this seedlength is just $O(\log n \log\log n)$, while the traditional proof of the INW generator requires seedlength $O(\log^2 n)$.

Our proof also works for Nisan's generator, but we find it conceptually easier to work with the INW generator.

Our proof actually holds even for ROBPs as long as the *mass* of every state is either 0 or not too small. Here, by the *mass* of a state we mean the probability that when applying the program to a uniformly random input, the program passes through that state. When the mass of every state is either 0 or $\geq \mu$, then our seed length is $O(\log n(w^4 \log\log n + w^3 \log(1/\mu) + \log(1/\varepsilon)))$. A similar result is found in [1].

Due to lack of space, and difficulty in even giving a sketch of the arguments without a comprehensive exposition of the background, we omit the rest of this section entirely, and encourage the reader to get the full version online.

VII. Why Our Technique Fails to Fool ROBP

In this section we explain why our technique fails to fool ROBPs, despite the fact that we do solve the coin and dice problems for ROBPs. We omit the content of this section because the required background takes up too much space, and refer the reader to the full version.

VIII. Conclusions

In this paper, we provide nearly-tight upper and lower bounds on how close two distributions can be and still be distinguishable by small-width read-once branching programs.

It would be interesting to make these results even tighter and extend them in various directions. It would also be interesting to study the coin problem on other classes of functions. One interesting class is low-degree polynomials over $\mathbb{F}_2$, where we conjecture that the best β would be only a function of the degree, and not of the number of variables. Another interesting class is ACC^0: could a technique based on couplings prove that ACC^0 cannot solve the coin problem? (This will in particular prove that majority is not in ACC^0, settling a major open problem.) Our work uses only couplings in which the coordinates are mutually independent, but to analyze the above classes, it might be useful to use more complicated couplings.

Finally, can the ideas in this paper, possibly along with other ideas, give a PRG against small-width ROBP with good seedlength? Note that the coin theorem implies that for any $\varepsilon > 0$, the PRG whose output is n coin-flips of a $\Theta(\varepsilon/\log^{2w} n)$-biased coin is a good PRG: it ε-fools width-w ROBPs. ([1] also prove something like this, but only for rROBPs.) This PRG has terrible seedlength, though. It might be possible to "derandomize the proof of the coin theorem" in some way to prove that a much more randomness-efficient PRG works as well.

Acknowledgements

The authors wish to thank Sourav Chakraborty for helpful discussions. The first author would like to thank David Barrington and the Dartmouth Theory Reading Group for helpful discussions. The second author is grateful to Brendan Juba, Jaikumar Radhakrishnan, Pranab Sen and John Steinberger, for stimulating discussions and ideas that helped progress this work. The second author would also like to thank Boaz Barak, Swastik Kopparty, Shachar Lovett, Avi Wigderson and David Xiao for helpful discussions. We thank the anonymous referees for pointing us to some important references. Some of this research was done while the authors were visiting Peter Bro Miltersen at the Center for Algorithmic Game Theory, Aarhus University. We would like to thank Peter and the group for their gracious hospitality.

References

[1] M. Braverman, A. Rao, R. Raz, and A. Yehudoff, "Pseudorandom generators for regular branching programs," in *Proc. 51st Annual IEEE Symposium on Foundations of Computer Science*, 2010, to appear.

[2] M. E. Hellman and T. M. Cover, "Learning with finite memory," *Ann. Math. Stat.*, vol. 41, no. 3, pp. 765–782, 1970.

[3] T. M. Cover, "Hypothesis testing with finite statistics," *Ann. Math. Stat.*, vol. 40, no. 3, pp. 828–835, 1969.

[4] M. E. Hellman, "Learning with finite memory," Ph.D. dissertation, Stanford University, Department of Electrical Engineering, 1969.

[5] S. Aaronson and A. Drucker, "When qubits go analog (powerpoint talk)," 2008, based on unpublished work. Slides available at http://www.scottaaronson.com/talks/analog.ppt.

[6] S. Aaronson, "BQP and the polynomial hierarchy," in *Proc. 42nd Annual ACM Symposium on the Theory of Computing*, 2010, pp. 141–150.

[7] K. Amano, "Bounds on the size of small depth circuits for approximating majority," in *Proc. 36th International Colloquium on Automata, Languages and Programming*, 2009, pp. 59–70.

[8] R. O'Donnell and K. Wimmer, "Approximation by DNF: Examples and counterexamples," in *Proc. 34th International Colloquium on Automata, Languages and Programming*, 2007, pp. 195–206.

[9] E. Viola, "On approximate majority and probabilistic time," *Computational Complexity*, vol. 18, no. 3, pp. 337–375, 2009.

[10] R. Meka and D. Zuckerman, "Pseudorandom generators for polynomial threshold functions," in *Proc. 42nd Annual ACM Symposium on the Theory of Computing*, 2010, pp. 427–436.

[11] M. L. Furst, J. B. Saxe, and M. Sipser, "Parity, circuits, and the polynomial-time hierarchy," *Mathematical Systems Theory*, vol. 17, no. 1, pp. 13–27, 1984.

[12] N. Alon, O. Goldreich, J. Hastad, and R. Peralta, "Simple construction of almost k-wise independent random variables," in *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science*, 1990, pp. 544–553.

[13] N. Nisan, "RL $\subseteq$ SC," in *Proc. 24th Annual ACM Symposium on the Theory of Computing*, 1995, pp. 619–623.

[14] R. Impagliazzo, N. Nisan, and A. Wigderson, "Pseudorandomness for network algorithms," in *Proc. 26th Annual ACM Symposium on the Theory of Computing*, 1994, pp. 356–364.

[15] F. Ergün, R. Kumar, and R. Rubenfeld, "On learning bounded-width branching programs," in *Proc. 8th International Conference on Learning Theory*, 1995, pp. 361–368.

[16] A. Bogdanov, Z. Dvir, E. Verbin, and A. Yehudoff, "Pseudorandomness for width-2 branching programs," 2009, ECCC Technical Report TR09-070.

[17] M. Luby and A. Wigderson, "Pairwise independence and derandomization," *Foundations and Trends in Theoretical Computer Science*, vol. 1, no. 4, pp. 237–301, 2006.

[18] M. Koucky, P. Nimbhorkar, and P. Pudlak, "Pseudorandom generators for group products," 2010, ECCC Technical Report TR10-113.