The Coin Problem, and Pseudorandomness for Branching Programs

Joshua Brody
Computer Science Department
Swarthmore College
Swarthmore, PA, USA
joshua.e.brody@gmail.com

Elad Verbin
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 $1/2 + \beta$ 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 recurfsive 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.

In December 2017, Avishay Tal brought to our attention a bug in the proof of our width-elimination lemma (Lemma 11 in the conference version, Lemma 10 in the full version). As a result, we retract our claim on this lemma, as well as the overall Coin Problem lower bound for ROBPs (Theorem 5 in both versions). Note: this lower bound still holds; we retract only the proof. In fact, this result has already been reproved and strenghthened by Steinberger (CCC 2013), who replaces our random restriction with a novel "interwoven hybrid" argument. In his proof, Steinberger uses our Collision Lemma, which we do not retract. We encourage the interested reader to consult the Steinberger paper for a proof of the Coin Problem lower bound for constant-width ROBPs.

The Coin Problem, and Pseudorandomness for Branching Programs

Joshua Brody
Department of Computer Science
Dartmouth College
jbrody@cs.dartmouth.edu

Elad Verbin
ITCS
Tsinghua University, 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 be correct 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.

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, then the dice cannot be distinguished by a small-width read-once branching program.

We suggest one application for this kind of theorems: we prove that Nisan's Generator fools width-w read-once *permutation* 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 [5] (which might be submitted to this FOCS).

1 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 whether the coin is biased toward heads or biased toward tails? 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.

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 constant-width, read-once branching programs (ROBPs). Informally, branching programs are layered directed acyclic graphs. States are labeled with variables x_i, and edges are labeled with possible values for x_i and transition to some state in the next layer. The first layer contains a single start state, and the last layer contains accept and reject states. Thus, a path from the start vertex to an end state describes a computation where variables are queried, and the state changes based on the values of these variables. The *width* of a branching program is the maximum number of states per layer. Branching programs with small width capture the notion of computation using limited space. (The space, in bits, is the logarithm of the width). In a *read once* branching program, all states in layer i are labeled with x_i. Thus, the branching program reads x_i exactly once, at layer i.

Note that non-read-once branching programs are extremely strong, as shown by Barrington [3]: he proved that width-5 polynomial-sized branching programs recognize any language in NC^1; in particular, they can compute Majority. In contrast, the model we are interested in is *read-once* programs, which are much weaker. They can be thought of as automata which has a different transition function at each point in time. Still, these machines are quite strong. For example, we show in this work that branching programs can solve the coin problem surprisingly well. In this work we do bound their power, 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.

1.1 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. 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

- $\beta = \Omega\left((\log n)^{-2w}\right)$ *in the coin problem,*
- $\beta = \Omega\left((\log n)^{-3w}\right)$ *in the dice problem.*

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

Theorem 3. *(Coin Problem Upper Bound). 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 $1/2 - O\left((\log n)^{2-w}\right)$.*

1.2 Work Related to the Coin Problem

To the best of our knowledge, this is the first work to consider constant-width branching programs that distinguish two close distributions; however, constructing distinguishers in general has a long history in the literature.

Our upper bound can be interpreted as an attempt to approximate the majority function using small space. Several recent papers consider the problem of approximating MAJ under different computation models. The closest to our work is the recent work of Amano [2], which provides size/depth tradeoffs for AC^0 circuits that approximate majority. O'Donnell and Wimmer [13] previously provided lower bounds on the size required to approximate majority with depth-d circuits; Amano's work gives a matching upper bound. The work of Viola [15] also has similar spirit. Also see the references therein. We remark that the notion of approximation considered in the coin problem is of dual nature: it both concentrate on inputs that are significantly biased, and it allows mistake of $1/3$ (in the distributional sense).

To the best of our knowledge, the current paper is the first work to study approximate-majority-type problems on read-once branching programs.

1.3 Techniques

In this section, we briefly describe the techniques used to prove Theorem 1, for the case of the coin problem, where the probability of heads is either $1/2 + \beta$ or $1/2 - \beta$.

In the proof, we first perform a monotonization step. Roughly speaking, we order the states at each level by how confident we are that the distribution comes from world 1 instead of world 2, conditioned on having

reached the state. We prove that in the *best* branching program, i.e. the one with the highest probability of success on the coin problem, there is a way to order the states such that the 1-transition coming out of a state always goes *above or equal to* the 0-transition coming out of the same state. Otherwise, we could change transitions and increase the success probability.

Our second step is to perform a series of $O(w)$ random restrictions. We show that with high probability, after the random restrictions, we are left with a branching program whose input only depends on few variables. Note that for this to work, the monotonization step was necessary: a (non-monotone) width-2 ROBP can compute the XOR of all the input bits, and the XOR function is incredibly resilient to random restrictions. The monotonization step allows us to assume that the ROBP is monotone, and such programs are, as we show, "killed" by random restrictions.

Finally, we show that a ROBP that is "killed" by random restrictions cannot distinguish two close distributions. We do this by an easy reduction argument, that in fact works for any function class. This finishes the proof of the coin theorem.

To generalize this proof for the case of dice, 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" 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 3.

1.4 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 [1], but for width 3 the problem is already wide open. (See more background in Section 6).

In Section 6 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 *permutation* read-once branching programs of width w. Here, ε is the error parameter of the PRG. A *permutation* read-once branching program (pROBP) is a ROBP where every state has exactly two edges entering it: one of them labeled '0' and one labeled '1'. The generator we use is Nisan's generator [12] or the INW generator [8]. 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, see Section 6.

Section 6 also contains an extensive intuitive discussion of why and how the coin problem is related to the INW generator.

1.5 Relation to the work of Braverman, Rao, Raz and Yehudayoff

In recent work of Braverman et al [5], the authors of [5] prove that Nisan's generator fools the class of *regular read-once branching programs* (which is similar, but more general, than permutation read-once branching programs). It seems prudent to discuss the relation between the two papers.

In [5], the authors achieve a seed length of $O((\log w + \log\log n + \log(1/\varepsilon))\log n)$. This is *better* than the seed length that we achieve. Furthermore, they have achieved the result chronologically earlier: Elad Verbin has spoken to two of the authors of that paper in November 2009, where they have already informed him that they are able to fool this class with good seedlength using Nisan's generator by an approach based on information theory, although no details beyond that were discussed. The approach of fooling read-once

branching programs based on the coin problem have been proposed by Verbin as far back as 2008, and much of the approach (but not the solution of the coin problem itself) was proposed back then, so we consider the current paper to be independent work with respect to [5].

Furthermore, we believe that the approach in the current paper is potentially stronger than that of [5], in the sense that we concentrate on the coin problem and the dice problem, and the results on fooling pROBPs are just a product of these theorems. In particular, we believe the coin problem and dice problem have potential to produce PRGs for non-permutation read-once branching programs, and for larger values of w than we currently know how to achieve. Furthermore, we believe that the coin problem might have other implications, for small-space computation as well as for other "low" computational models. Thus we believe that the current paper, even if considered as subsequent work to that of [5], is still a significant contribution.

It seems to us that the work of [5] concentrates on pROBPs (or *regular* ROBPs), while ours is more concentrated on ROBPs. The reason our results only give PRGs for permutation ROBPs is a technical (but deep) reason, explored in Section 7. Therefore, we believe that with some more ideas, the PRG approach based on the coin/dice problem could also be made to work for non-permutation ROBPs.

1.6 Organization of the Paper

In Section 2, we formalize many of the concepts and tools used in the rest of the paper. Sections 3 and 4 develop our lower bounds, and Section 5 constructs the upper bound. In Section 6 we show our results on PRGs for pROBPs. Section 7 explains the barrier stopping the PRG results from giving PRGs for ROBPs. Finally, Section 8 concludes the paper and presents some open problems. Some technical proofs are left to Appendix A.

2 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 use two different notions of what it means to distinguish. We say that f *weakly* distinguishes X, Y if $\Delta(f(X), f(Y)) > 1/3$, and that f *strongly* distinguishes X, Y if it accepts with probability at least $2/3$ when the strings come from distribution X, and rejects with probability $2/3$ when the strings come from Y. Note that the former is a necessary condition for the latter. We give our lower bound in terms of weak distinguishers, and our upper bound in terms of strong distinguishers. It will be clear from context which notion we're using, and for that reason, we drop the weak/strong distinction and discuss only distinguishers.

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.

2.1 Probabilities, Transitions, and Support

In this subsection, we define and describe many of the concepts 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

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

3 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 $\Delta(f(X), f(Y))$.

We prove the collision lemma by demonstrating that the transitions are *monotonic* in a weak sense. Specifically, we order the states at each level such that the transitions that are more likely in distribution Y will always precede transitions that are more likely in distribution X.

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 becomes takes will come from the average of X_i and Y_i. We then condition on both which variables get sampled, and the

value they take. We show that after each conditioning, the support of each level decreases by 1 (with high probability).

After $w - 1$ steps, each remaining free level is likely to have support on only *one state*. Since the branching program reaches this state no matter what happens in earlier levels, it follows that the function computed by the branching program is independent of all preceding levels. This implies that the function depends only on the few levels that follow the last level whose support contains a single state. 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 3.1, we establish our notion of monotonicity and prove the collision lemma. Section 3.2 uses the collision lemma to prove the main theorem. This proof requires a lemma that is technical. For this reason, we defer the lemma's proof until Section 3.3.

3.1 The Collision Lemma

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 k 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 $f(X|s_i) < f(X|s_j)$ for all $i < j$.[1] We call this the *canonical ordering* of $\{s_1, \ldots, s_w\}$.

Our next lemma states that this ordering holds for Y as well.

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

Proof. Suppose for the sake of contradiction that there exist states $s < t$ such that $f(Y|s) > f(Y|t)$. Fix some s' and b such that $s'(b) = s$. Then, moving this transition such that $s'(b) := t$ increases $f(X|s')$ and decreases $f(Y|s')$. Using equations (1) and (2), it follows that $f(X)$ increases and $f(Y)$ decreases. Hence, we improve $\Delta(f(X), f(Y))$, which contradicts the optimality of f. □

The proof of the above lemma also shows that it is safe to assume that the canonical ordering is strict; that is, that $f(X|s_j) < f(X|s_{j+1})$ for all $1 \leq j < m$. Otherwise, we could take all transitions that go into s_j and move them to point to s_{j+1} without changing $\Delta(f(X), f(Y))$.

The next lemma uses wire-switching to show a weak form of monotonicity. We leave the proof to the Appendix.

Lemma 7. *Fix some level k of the branching program, and let s be a state at level k. Assume states are ordered canonically, and let $a, b \in [m]$ be such that $p_{i,a} \geq q_{i,a}$ but $p_{i,b} \leq q_{i,b}$. Then, in any good branching program, $s(a) \geq s(b)$.*

Our final lemma in this section is a collision lemma. Formally, for $e \in [m]$, an *e-collision* is a pair of states s and t such that $s(e) = t(e)$. 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)

[1] Technically at this point the inequality should not be strict; however, we'll soon see that we can assume strict inequalities without loss of generality.

must occur. The next lemma characterizes when collisions occur if the number of states does not decrease across levels.

Lemma 8 (Collision Lemma). *Let $s_1, \ldots, s_d$ and $t_1, \ldots, t_d$ denote the support of two consecutive levels of the 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 $1 \le i \le d$ and for all $e \in [m]$.*

3.2 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 9. *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. *$p_{i,e}, q_{i,e} \ge \delta$ whenever $p_{i,e}$ and $q_{i,e}$ are nonzero.*
3. *$\Delta(X_i, Y_i) \le \beta$.*

Then we have $\Delta(f(X), f(Y)) \le \beta \cdot p^{-(w)}$.

Our main theorem follows directly from Theorem 9.

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 9, $\Delta(f(X), f(Y)) \le \beta \cdot p^{-w}$. Hence, we have $\beta = \Omega(p^w) = \Omega\left((\log n)^{-w}\right)$. □

We will prove Theorem 9 using a coupling method. It's well known that under any coupling ω of distributions μ, ν, we have $\Delta(X, Y) \le \Pr_\omega[X \ne Y]$. 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 $S_2 \subseteq S \setminus S_1$ 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 $i \in S \setminus \left(\bigcup_j S_j\right)$ 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 $\Delta(X'_i, Y'_i) \le \beta'$. Simple calculations show that

$$\Pr[X'_i = e] = (1 - p^{w-1}) r_{i,e} + p^{w-1}(r_{i,e} + \delta_{i,e} p^{-w+1}) = r_{i,e} + \delta_{i,e} = p_{i,e} \,,$$
$$\Pr[Y'_i = e] = (1 - p^{w-1}) r_{i,e} + p^{w-1}(r_{i,e} - \delta_{i,e} p^{-w+1}) = r_{i,e} - \delta_{i,e} = q_{i,e} \,.$$

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 10 (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 on $\Omega\left(n/(\log n)^{w-1}\right)$ variables with probability greater than $1 - 1/n$.*

We defer this proof until Section 3.3. The rest of the theorem follows from the following lemma:

Lemma 11. *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. We use a hybrid argument. Without loss of generality, assume that f depends on the first k variables. Let $Z_0 := Y$, and for $1 \leq i \leq k$, let $Z_i := X_1 X_2 \ldots X_i Y_{i+1} \ldots Y_k$. Note that $Z_k = X$. It's not hard to see that for all $1 \leq i \leq k$, we have $\Delta(Z_{i-1}, Z_i) = \Delta(X_i, Y_i) \leq \beta$. Then, by the triangle inequality, we have

$$\Delta(g(X), g(Y)) \leq \Delta(X, Y) = \Delta(Z_0, Z_k) \leq \sum_{i=1}^{k} \Delta(Z_{i-1}, Z_i) \leq k\beta \; .$$

□

Proof of Theorem 9. 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 10, 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 11. □

3.3 Proof of Lemma 10

Definition 12. *We say that level k is* inactive *if for all states s_i, s_j at level k, and for all $a, b \in [m]$, we have $s_i(a) = s_i(b), s_j(a) = s_j(b)$, and $s_i(a) \neq s_j(a)$. A level is* active *if it is not inactive.*

Unless otherwise specified, assume that a level is active. Inactive levels are effectively identity permutations – each state in the support of level k maps directly to some state in the next level. Moreover, it is easy to see that the output of the branching program is independent of x_k. Also, note that levels that are active may become *inactive* after a conditioning, since conditioning can decrease the support of a level. In our analysis, we shall perform a series of conditionings. For the most part, we *ignore* levels that become inactive.

Proof of Lemma 10. Let us call an event $\mathcal{E}$ *unlikely* if $\Pr[\mathcal{E}] \leq 1/n^3$. Through out this proof, we carry an implicit assumption that we are conditioning on the event that *none* of the unlikely events happen. At the end, we will show via a union bound that this assumption is reasonable, i.e., with high probability none of the unlikely events will happen.

Our proof uses induction, and can be seen at a high level as a form of width elimination. We shall show that after conditioning on $S_1, \ldots, S_k$, each of the remaining unrestricted levels is likely to have support on only $w - k$ states.

Let $a := (50 \log n)/\delta$. Through these conditionings, we wish to maintain the invariant that *except* for the first a free layers, the layers are likely to have small width. Specifically, we wish to maintain the following invariant:

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

We say that x_i is *nice* after conditioning on $S_1, \ldots, S_k$ if the support of level i is at most $w - k$. When $k, S_1, \ldots, S_k$ are clear from context, we simply say that x_i is *nice*.

As a base case, consider the first conditioning. Pick an arbitrary free level x_i, and consider the a active levels that precede it. Then, the probability that all a preceding levels are restricted is

$$(1-p)^a \geq e^{-2ap} \quad = \quad e^{-0.1/\delta} > 0.9\,, \tag{3}$$

where the first inequality holds because $1 - x \geq e^{-2x}$ for all $0 \leq x < 1/2$.

Next, consider the levels that remain free, and consider any consecutive pair i_1, i_2 such that at least a restricted levels lie between them. We wish to show that the support of the layer i_2 is at most $w-1$. To prove the support of layer i_2 is at most $w - 1$, we use a pebbling argument. Specifically, place a pebble at each state in the support of level $(i_1 + 1)$, and move these pebbles down the layers, along whichever transitions are dictated by the conditioning. Note that whenever two pebbles arrive at the same state, they stay together for the remainder. By the Collision Lemma, at any active level there exists an e-collision for some $e \in [m]$ such that $r_{i,e} > \delta$. Therefore, each active level will have a collision with probability at least δ.

Note that in between two free layers, if a collision exists on any layer, then at most $w - 1$ states in the free level at the bottom will receive pebbles. We are guaranteed to never reach state(s) that receive no pebbles, and so the support of level i_2 is precisely the set of states receiving pebbles. If a active restricted levels lie between a free level i and the free level that preceded it, then the probability that level i will *not* decrease in width is $(1 - \delta)^a \leq e^{-\delta a} = e^{-50 \log n} = n^{-50/\ln 2}$, which is unlikely.

Therefore, each level that remains free after the first conditioning is likely to have support $\leq w-1$ as long as there are $a = (50 \log n)/\delta$ levels preceding it. Trivially, at most a levels do *not* have a levels preceding them. Hence, the invariant holds after the first conditioning.

The logic behind the invariant for subsequent conditionings has the same flavor, but there are a couple of subtle complications, which we present below.

Assume for the sake of induction that the invariant holds after conditioning on $S_1, \ldots, S_k$. Let F_k denote the set of free levels after the kth conditioning. Note that $F_{k+1} \subseteq F_k$. Now, take any $j \in F_{k+1}$. There are two cases.

For the first case, suppose that prior to conditioning on S_{k+1}, at least $2a$ free (active) levels precede j, and consider the a free levels immediately preceding j. Then, as in equation (3), with probability > 0.9, these layers *all* become restricted. Suppose that this is indeed the case. Now, if any of these layers had support $w' < w-k$, then by the pebbling argument, layer j will also have width at most $w' \leq w-k-1$. Otherwise, by the invariant, each of these layers are *nice* with probability > 0.9. Therefore, we expect $0.9a$ of them to be nice. Since the probability that each of these layers is nice is independent of the others, by another Chernoff bound, the event that less than $0.6a$ of them are nice is unlikely. On the other hand, with $0.6a$ nice layers, there are at least $0.1a$ consecutive pairs of nice layers. Each of these pairs have support $w-k$, and since they are active, each layer of transitions must have an e-collision. Hence, the probability that there are *no* collisions in the a restricted layers preceding j is at most $(1-\delta)^{-0.1a} \leq e^{-5 \log n/} = n^{-5/\ln 2}$, which is unlikely. It follows that with high probability, j will have support at most $w-k-1$.

In the second case, suppose that at most $2a$ free levels precedes j. Then, we make no guarantees on the support of j. On the other hand, we will show there will not be too many such free j. Consider layers $i_1, \ldots, i_{2a}$. Conditioning on S_{k+1}, each layer remains free with probability p, and these probabilities are independent. Let $\mathcal{E}$ be the event that at least half of $i_1, \ldots, i_{2a}$ remain free after conditioning on S_{k+1}. By a Chernoff bound,

$$\Pr[\mathcal{E}] \leq \exp\left(-\frac{2a(1/2-p)^2}{2(1-p)}\right) \leq \exp\left(-a \cdot 0.49^2\right) \leq n^{-12} .$$

This event is unlikely. Therefore, with high probability, we make no guarantees about at most a free levels after conditioning on S_{k+1}. Together, these cases prove that the invariant holds after conditioning on S_{k+1}.

By induction, the invariant holds after conditioning on $S_1, \ldots, S_{w-1}$. Now consider what remains. Let $\hat{f}$ denote f after conditioning on $S_1, \ldots, S_{w-1}$. By the invariant, if more than a free layers exist, then the later layers have a support of 1 with probability greater than 0.9. Now, let x^* denote the last free active layer with support 1. Note that since $\hat{f}$ sends all probability to a single state in x^*, then the output is independent of the layers before x^*. Suppose that x^* is the Dth free layer from the end of the branching program. Since each of these layers has support 1 independently with probability greater than 0.9, then $\Pr[D > \log n] = (0.1)^{\log n} < n^{-3}$. Hence this event is unlikely. It follows that $\hat{f}$ depends on at most $a + \log n < (51 \log n)/\delta$ variables, i.e., $\hat{f}$ is an $O(\log n)$-junta.

Finally, we performed $w-1$ conditionings, and each time we defined $O(n)$ unlikely events. Hence, by the union bound, the probability that some unlikely event occurred is at most $O(w/n^2) = O(1/n)$. □

4 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.

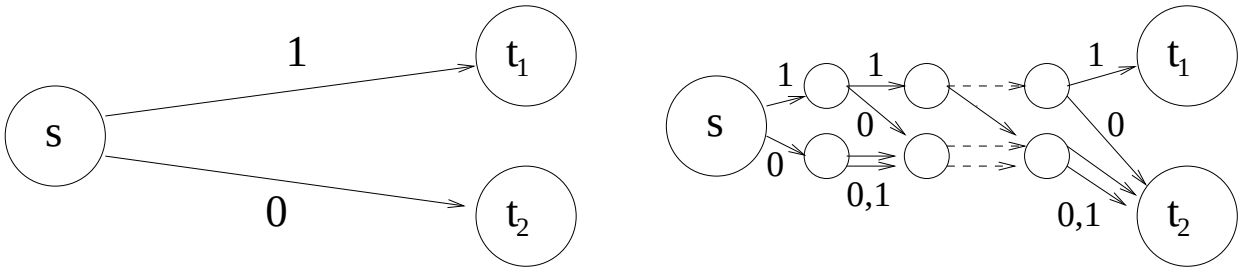

Figure 1: Replacing each $(s(0), s(1))$ pair with a width-2 balanced ROBP

Theorem 13. *Suppose $X = (X_1, \ldots, X_n)$ and $Y = (Y_1, \ldots, Y_n)$ are collections of mutually independent random coins such that*

$$\frac{1}{1+\beta} \leq \frac{\Pr[X_i = 0]}{\Pr[Y_i = 0]}, \frac{\Pr[X_i = 1]}{\Pr[Y_i = 1]} \leq 1 + \beta\,.$$

Then, if a width-w ROBP distinguishes X and Y, then $\beta = \Omega\left((\log n)^{-2w}\right)$.

Proof. 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

$$\begin{aligned} \Delta(f(X), f(Y)) &\leq \Delta(f(X), f(X')) + \Delta(f(X'), f(Y')) + \Delta(f(Y'), f(Y)) \\ &\leq \Delta(f(X'), f(Y')) + 2/n. \end{aligned}$$

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 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. □

We obtain a similar result for the dice problem.

Theorem 14. *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*

$$\frac{1}{1+\beta} \leq \frac{p_{ie}}{q_{ie}} \leq 1 + \beta$$

for all $1 \leq i \leq n$ and $e \in [m]$. If a width-w ROBP distinguishes X and Y, then $\beta = \Omega\left((\log(mn))^{-3w}\right)$.

As in the proof of Theorem 13, we reduce from Theorem 5. For completeness, we provide the proof in Appendix B.

5 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 15. *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 Appendix C.

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 [6, 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 $\beta = O\left((\log n)^{-(w-2)}\right)$. For this, we closely follow Amano's construction [2] of AC^0 circuits that approximate MAJ. Our construction is left to Appendix C.

6 The INW Generator Fools pROBP 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 pROBPs.

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. s is called the *seed length*.

We are interested in constructing explicit generators G that fools 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 are known are that for width-2 ROBP, $O(\log n)$ seed length is achievable by using epsilon-biased generators, see e.g. [1], for an extension see [4] and that for the class C of width-$poly(n)$ ROBP, Nisan's generator [12] can fool them with seed length $O(\log^2 n)$. An improvement/variant on Nisan's generator is the INW generator [8]. See below for a definition of the INW generator. It is a famous open problem even to get a PRG for width-3 ROBP that surpasses Nisan's generator. In this section we offer a partial solution to this problem, which only fools pROBPs, and achieves seed length $O(\log n \log\log n)$, when ε and the width are both constants. Our generator is the INW generator, but our proof is novel. Some more recent results touching on this problem, see e.g. [11, 10]. For more background on pseudorandomness, see e.g. the survey by Luby and Wigderson [9].

Given a distribution D, we say the *mass* of an element x with respect to D is the probability that D assigns to x.

We will deal extensively with ROBPs and pROBPs. These were defined in Section 1.

In this section, we prove the following:

Theorem 16. *The INW generator ε-fools the class of width-w* permutation *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 $O(\log^2 n)$.

It should be noted that all of our discussion applies both for Nisan's generator and for the INW generator, except where otherwise noted. We find the INW generator easier for the purpose of exposition, so we base our results on them.

In the following we prove Theorem 16. We first introduce the INW generator, and explain the intuition behind our improvement. We then show our improvement in the simple setting of the *baby-INW* generator, which we define, and then apply the same ideas to improve the INW generator itself.

6.1 PRGs: Some Preliminaries

We start by defining some parameters. n is the length of the string which the PRG outputs. d is the degree of the expanders; we leave it un-set for now. The seedlength of the generator is $O(\log n \log d)$. $s_0 = \log n$ is

the length of a bottom-level seed. $s_\Delta = \log d$ is the length of the seed needed for each level; this is because $\log d$ is the number of bits needed to choose an outgoing edge in a d-regular graph. $\ell = \log(n/\log n)$ is the number of levels of the generator (also called the depth of the recursion, or the depth of the *INW tree*). $s_i = s_0 + is_\Delta$ is the length of the seed at level i. The length of the overall seed is $s = s_\ell = s_0 + \ell s_\Delta$.

We now define the INW generator. The generator is defined using ℓ expanders, $G_1, \ldots, G_\ell$, $G_i = (V_i, E_i)$. Each of the expanders is d-regular. The expander G_i has $|V_i| = 2^{s_i} = nd^i$ vertices; it is a Ramanujan or near-Ramanujan graph. The eigenvalue gap of such expanders is $\lambda_2 \approx 1/\sqrt{d}$. We do not define λ_2 or other notions related to expanders here, since they will be of little use: for our analysis we just need the expander mixing lemma. For more information on expanders, see e.g. [7].

The INW generator works as follows. Start with a uniformly random seed of length $s = s_\ell$. Partition it to a first part, consisting of $s_{\ell-1}$ bits, and to a second part, consisting of s_Δ bits. Use the first part to select a vertex, v, in G_ℓ, and the second part to select an edge, (v, u) coming out of this vertex. v and u are individually uniform in V_ℓ, but they are dependent. Now, consider v as a seed for the $(\ell-1)$-th level, and proceed recursively with the expander $G_{\ell-1}$. Similarly, consider u as a seed for the $(\ell-1)$-th level, and proceed recursively, also with $G_{\ell-1}$. Concatenate the two $n/2$-bit strings you get from these two recursive operations, to produce one n-bit string; this string is the output of the generator. In other words, given seed x, the output $INW_\ell(x)$ is the concatenation of $INW_{\ell-1}(v)$ with $INW_{\ell-1}(u)$. The base case is that $INW_0(x) = x$, i.e. when the bottom-most level 0 of the generator gets a seed, it simply returns it as-is.

Notice that implicit in the definition of the INW generator is a one-to-one labeling of the vertices of each V_i by $(\log |V_i|)$-bit strings, as well as a one-to-one labeling of the edges touching each vertex v by $(\log d)$-bit strings. We refer to these two labelings together as the *expander-labeling*. The labeling of the edges is defined for each vertex separately, and the labeling of en edge does not have to be consistent on both sides: i.e. the edge (u, v) might be labeled '4' if coming from v but '7' when coming from u. It is not particularly important what these labelings are as long as they are one-to-one, because our proofs work no mater what the labeling is. Therefore we do not discuss the labeling here; the labeling does, however, play a role in the negative result in Section 7.

The INW generator implicitly defines a rooted ordered complete binary tree of depth ℓ. We call this the *INW tree*. Each node in the tree is associated with some seed. The seed of the root is chosen uniformly at random, and the seeds in the rest of the tree are computed deterministically from the root-seed. The seeds at the leaves are concatenated to get the output of the generator.

The original Nisan's generator [12] uses hash functions rather than expanders, but the idea is very similar. We do not discuss it further here.

The traditional theorem about the INW generator is the following:

Theorem 17. *When $d = poly(nw/\varepsilon)$ then the INW generator ε-fools the class of width-w read-once branching programs of length n.*

While our theorem states:

Theorem 18. *When $d = (\log n)^{\Theta(w^4)}/\varepsilon^2$ then the INW generator ε-fools the class of width-w* permutation *read-once branching programs of length n.*

With the traditional setting of parameters, the INW generator uses a seed length of $O(\log n \log d) = O(\log n \log(nw/\varepsilon))$ bits. We now shortly outline the traditional proof of Theorem 17. We need this, since our proof of Theorem 18 is obtained by an improvement of it. To show the proof, we start by quoting the Expander Mixing Lemma:

Lemma 19 (Expander Mixing Lemma)**.** *For every d-regular graph $G = (V, E)$ and for every two sets $A, B \subseteq V$, it holds that*

$$\left| \frac{|(A \times B) \cap E|}{|E|} - \frac{|A|\,|B|}{|V|^2} \right| \leq \frac{\lambda_2(G)}{d} \cdot \sqrt{\frac{|A|\,|B|}{|V|^2}} \leq \frac{\lambda_2(G)}{d} \,.$$

In the last equation, $\lambda_2(G)$ is the normalized value of the second-largest (in absolute value) eigenvalue of the adjacency matric of G. We do not include a proof of the expander mixing lemma here. For details, see [7]. When the graph G is a Ramanujan expander, then $\lambda_2(G) \leq 2/\sqrt{d}$, so the right hand side of this equation is at most $2/\sqrt{d}$. Denote $\beta_0 = 2/\sqrt{d}$.

The expander mixing lemma states that for any cut in the expander, the probability that a uniformly-sampled edge from E belongs to this cut is roughly equal to the probability that a uniformly sampled edge from the complete graph belongs to the cut. More accurately, it states that the difference between these two probabilities is at most β_0.

The intuition is that the expander mixing lemma is useful for the proof because each node of a branching program can be implicitly seen as a *cut-distinguisher*, i.e. a function that checks if an edge (which is somehow inferred from the input string) is in a given cut. The whole branching program can be seen as a collection of such cut-distinguishers, trying to tell if the input comes from the uniform world or from the PRG world. (Here, the "uniform world" means the uniform distribution over n-bit strings, and the "PRG world" means the distribution induced by the PRG). Once these ideas are understood, the analysis of the INW generator is easy: roughly speaking, the proof is simply an application of the triangle inequality on nw such cut-distinguishers, where each cut-distinguisher, because of the expander mixing lemma, contributes only a difference of $\beta_0 = \varepsilon/nw$.[2]

The proof of Theorem 17 proceeds as follows (still not entirely formally): consider the middle layer of the branching program. The middle level is the only way to "communicate" between the left half and the right half. Each of the nodes in the middle layer can, at most, act as a cut-distinguisher. Thus, the total difference amassed in the middle layer is at most $w\beta_0$. Since this is the only way to communicate between the two sides, then each side can now be assumed to be independently uniform, and the same arguments can be used recursively. We end up with a total difference of at most $wn\beta_0$, and this gives Theorem 17. The formal proof is a little more involved, since one needs to formalize the phrase "can now be assumed to be independently uniform" and similar vague concepts, but this level of detail suffices as background for our purposes. The interested reader is referred to [9] for more details. (When we prove Theorem 16 we will not make such intuitive claims, and we will prove everything formally).

6.2 Some High-Level Arguments Behind Our Improvements

The standard proof that the INW generator fools ROBPs has a striking place where it seems non-tight: It performs a hybrid argument, that states that if the statistical difference in each node is β_0, then the overall statistical difference at the end is upper-bounded by $nw\beta_0$. This is true of course, but does not seem tight for small-width programs, especially in light of our results on the coin problem. This hybrid argument is the reason that the analysis in Theorem 17 requires the degree of the expanders to be $poly(nw/\varepsilon)$, rather than, say, just $poly(w/\varepsilon)$. This is the weakness that we improve in this section. We show that since branching programs cannot boost/amplify, then there is no way to amplify a difference of β_0 in each state to an overall difference of $nw\beta_0$. Sweeping many technical details under the rug, what happens is that we

[2] This use of the triangle inequality can alternatively be thought of as a union bound or, most accurately, as an application of a hybrid argument.

think of the ROBP as being applied to roughly nw "coins", each with statistical difference β_0; then, by the coin theorem, the statistical difference in the output should be only about $(\log n)^{O(w)}\beta_0$, assuming various technical conditions such as the "gap" being around the middle, and so on. To get this difference to be at most ε, we need to choose $\beta_0 = \varepsilon/(\log n)^{O(w)}$, and since $d = poly(1/\beta_0)$, we get that the seedlength is $O(\log n \log d) = O(\log n(w \log\log n + \log(1/\varepsilon)))$. This is more or less the seedlength that we promise in Theorem 18. The discrepancy between the two quantities (w is replaced by w^4) is due to various technical complications.

In order to go through with the rough argument shown in the last paragraph, two major technical difficulties must be overcome. First, the coin theorem requires that the location of the gap be bounded away from 0 and 1. We need to prove this property, which actually *does not* hold for ROBPs. It does hold for pROBPs, and we show this in Lemmas 21 and 22. (This is the only place where the proof fails for ROBPs). Second, when trying to fool pROBPs (or ROBPs), they should be thought of as dual-duty machines: they are used simultaneously both to find the differences between the uniform world and the PRG world, and in order to boost/amplify those differences. Due to this dual nature, it is hard to apply the ideas of the coin theorem directly on the pROBP: one of the most obvious reasons is because we cannot make the program monotone or weakly-monotone without losing its distinguisher properties (these concepts, "monotonicity" and "weak monotonicity" are described in the proof of the coin and dice theorems). To circumvent this difficulty, we strengthen the model of pROBPs to a stronger model called super-pROBPs, where the two duties are well-separated. These super-pROBPs are "primary" pROBP, that gets as input the output of "secondary" pROBPs. The primary pROBP has the role of statically boosting the differences, while the secondary ones have the role of creating the differences in the first place. We prove in Lemma 20 that super-pROBPs can simulate pROBPs (this is not obvious), and then our analysis applies the dice theorem to the primary pROBP, treating the secondary pROBPs as the dice.

6.3 The Baby-INW Generator, Super-pROBPs, and Some More Intuition

We now set some parameters. These parameters are used throughout the rest of this section. The degree of the expanders is chosen to be $d = (\log n)^{\Theta(w^4)}/\varepsilon^2$. Note that here and in the rest of the arguments we hide some constants in Oh-notation; This hiding will be done in a consistent way, and no hairy issues will arise. Next, the "error" parameter β_0 is chosen to be $1/\sqrt{d} = \varepsilon/(\log n)^{\Theta(w^4)}$. Another error parameter is $\beta = w^3\beta_0 = \varepsilon/(\log n)^{\Theta(w^4)}$. A final error parameter is $\delta = 1/4^{w^4}$. It can be seen that the seedlength that we declared in Theorem 18 is just $\log n \log d$, thus to prove Theorem 16 it suffices to prove that an INW generator with degree d indeed ε-fools width-w pROBPs.

We now define the baby-INW generator. The baby-INW generator only performs the INW procedure on the bottom level of the INW-tree, and is uniform in all other levels. It is equivalent to choosing G_1 just like in the INW generator, and choosing the other expanders $G_2, \ldots, G_\ell$ to be complete graphs.

We first prove in Section 6.6 that the baby-INW generator (ε/ℓ)-fools width-w pROBPs. Then, in Section 6.7 we show how to generalize this proof for the actual INW generator; this step simply consists of applying a hybrid argument on top of the proof of the baby-INW generator. The hybrid argument will have ℓ steps, because the INW generator has ℓ levels, and thus the final error will be ε. We believe that getting better analysis of the baby-INW generator might be an important step in future work, since it provides a simple "training-ground".

We now define the baby-INW generator more concretely. Let $m = n/2\log n$. The baby-INW generator produces $2m$ elements, divided into pairs. We call each pair a *chunk*. Each chunk consists of the $\log n$-bit labelings of two vertices v, u in G_1, chosen by drawing v uniformly at random, and then choosing u to be a uniformly random neighbor of v in G_1. Thus, we see that the output of the baby-INW generator consists

of m independent chunks, each corresponding to one edge of the expander. It should be obvious why such a generator is easier to analyze than the INW generator. This generator has awful seedlength (more than $n/\log n$), but analysis on it can in some cases be generalized to hold for the actual INW generator. Furthermore, it should be relatively easy to see why this generator is related to the coin problem: consider the i-th chunk. The expander mixing lemma shows that the statistical distance created by any cut-distinguisher running on the i-th chunk in the uniform world versus in the PRG world is β_0. Think of this as a "coin". The approach of the classical analysis of the INW generator would be just to apply a triangle inequality on the wm differences, and get a total difference of at most $wm\beta_0$. Instead, our analysis exploits the fact that a pROBP cannot boost/amplify the statistical distances generated by the cut-distinguishers.

We now define super-pROBPs. A *width-w super-pROBP* consists of a width-w length-m pROBP, whose inputs, rather than being bits, are 2^{w^3}-sided dice (more accurately, they are random variables over the universe $\{0,1\}^{w^3}$). In the i-th die, the j-th of these bits is the output of *another width-w pROBP*, call it $f_{i,j}$, which is applied only on the i-th chunk of the string. To summarize, a super-pROBP is a length-m pROBP, each of whose inputs consists of w^3 bits, which are the outputs of individual pROBPs run on the corresponding chunk. Intuitively, one should think of the *primary* pROBP as the part that intends to boost the error, and the *secondary* pROBPs as the cut-distinguishers. Note that in the baby-INW generator, the inputs of the primary pROBP are *mutually independent* random variables over $\{0,1\}^{w^3}$. This independence is crucial, since we are going to think of these inputs as the dice in the dice theorem, and the dice must be independent.

6.4 Simulating pROBPs by super-pROBPs

We now prove that any width-w pROBP can be simulated by a width-w super-pROBP. Note that this is not entirely trivial, since in a super-pROBP, the main pROBP does not get direct access to the input, but is restricted to access the input only through the secondary pROBPs, which, in their turn, are restricted to just look at one chunk each.

Lemma 20. *A width-w pROBP can be simulated by a width-w super-pROBP. Also, in this simulation, each secondary pROBP $f_{i,j}$ is a cut-distinguisher on chunk i.*

Note that when we say "simulate" we mean that the simulating super-pROBP and the simulated pROBP always return the same output given the same input. I.e., they compute the same function. In particular, this lemma does not assume that the input comes from the INW generator. It does use the notion of chunks (in the definition of super-pROBPs), but it just thinks of the chinks as a partitioning of the input, so it does not make any assumptions about the input.

Proof. Denote the original pROBP by f. Consider the behavior of f on the i-th chunk. For each node v of f in the layer just before the beginning of the chunk, and for every node u of f in the layer in the exact middle of the chunk and for every node w of f in the layer just after the end of the chunk, define a function $f_{i,j}$, where $j=(v,u,w)$, i.e. j is indexed by the nodes v, u and w. $f_{i,j}$ returns 1 if when the program starts from node v and proceeds according to the branching program f and reads the i-th chunk of the input, it goes through both node u and node w; $f_{i,j}$ returns 0 otherwise. Note that in particular $f_{i,j}$ is just a cut-distinguisher applied on chunk i. Also note that for every v, there is only one choice for a pair (u,w) that makes $f_{i,(v,u,w)}$ equal to 1.

Now, to simulate the program f on input x, we consider the start vertex v_0, and choose the nodes u_0, w_0 that make $f_{i,(v_0,u_0,w_0)}=1$. (By the previous discussion there is only one choice for these u_0, w_0). Now we set $v_1=w_0$ and find the nodes u_1, w_1 that make $f_{i,(v_1,u_1,w_1)}=1$, and so on. In the end we get a final vertex $v_{m-1}=w_{m-2}$ in the output-layer, and we return it. Obviously, this procedure returns the correct $f(x)$.

However, we still need to show that this procedure can be simulated by a super-pROBP. We need to build the primary pROBP to correspond to this procedure. We do it as follows: The state between layers $i-1$ and i of the primary pROBP corresponds to the state of the simulated pROBP between chunks $i-1$ and i. Now, we just need to prove that for each possible w^3-bit input in the $(i-1)$-th layer, the transition is a permutation on the states. This can easily be seen to be the case, since for any pROBP, and in particular in the original pROBP f, any composition of layers is always a permutation as well. Informally, the input to the primary pROBP tells us which path we should traverse for any node that we start in, and no two of these paths end at the same node.

We remark that some choices of w^3-bit inputs are impossible to obtain with the simulation (for example ones that are all-0). For these choices we define the transition to be an arbitrary permutation.

We have thus seen how to simulate the pROBP by a super-pROBP with the same width, and have proved the lemma. □

6.5 A Useful Lemma about Mass of States in pROBPs

We now prove a lemma, which states that in a pROBP no state has too-small probability (unless it has probability 0). This is in fact the *only* place in this section that we use the fact that we are fooling pROBPs rather than ROBPs. The inspired reader might claim she has no desire for such a lemma, and could probably prove the pseudorandomness without it; the reason we need the lemma is that the coin theorem requires the "gap" to not only be small but to be "around the middle" and there is also an analogous restriction in the die theorem.[3] Furthermore, in Section 7, we show that in a strong sense, the INW generator simply *does not* work without a lemma such as this, or at least one can design a *specific expander-labeling* for which the INW generator fails.

Lemma 21. *Let f be a width-w pROBP. Consider running f on a uniformly random input x. Let v be a node in the program f. The probability of f applied on x to pass through v is either 0, or at least $1/2^w$.*

Proof. Call the probability to pass at a vertex v the *mass* of v. We will now prove a stronger claim than the lemma. We prove that for every layer, the following holds: assume the number of nodes with nonzero mass in the layer is r. Then for each node in the level, its mass is either 0, or at least $1/2^{r-1}$.

The proof proceeds by induction on the layers of the branching program. The base case is trivial (in the first layer one node has mass 1 and the other nodes have mass 0, so $r=1$). Fix a layer $i-1$ where the claim holds, and let us prove it for layer i. Denote by r_{i-1} and r_i the number of states with nonzero mass in layers $i-1$ and i, respectively. It is easy to see that the number of states with nonzero mass cannot decrease, thus $r_i \geq r_{i-1}$.

Consider a node v in layer i. If both of its in-arrows come from states with mass zero, then its mass is also zero and we are done. If both of its in-arrows come from states with mass nonzero, then by the induction hypothesis they both have mass $\geq 1/2^{r_{i-1}-1}$. But v's mass is the average of its in-neighbors mass (since the input is uniform), and since $r_i \geq r_{i-1}$ we are again finished.

If one of v's in-arrows come from a state with mass zero and one comes from a state with mass non-zero, then v's mass might be as small as $1/2^{r_{i-1}}$. But in this case, it is easy to see that $r_i > r_{i-1}$. One way to see this is that there are $2(w-r_{i-1})$ "available" arrows coming from the zero-mass states in layer $i-1$. v

[3] The lemma we are about to prove and its use shed some light on ideas of Reingold [14]. Reingold says that what is important in order to fool small-width models is to have good estimates of the probability masses of being in various states. The following lemma and its uses in our arguments suggest that what is important, at least in our setting, is to have some lower-bound on the mass of states; i.e. we just need that the masses are not too small, and we do not need good estimates of them.

took only one of them and a node in layer i that has zero mass must take two of them, thus there are at most $w - r_{i-1} - 1$ nodes with zero mass in layer i.

Overall, by induction we have the stronger claim, and the lemma follows from it. □

We now need a generalization of this lemma, for the case where k pROBPs are applied on the same input simultaneously.

Lemma 22. *Let $f_1, \ldots, f_k$ be k width-w pROBPs. Let x be a uniformly random input, and consider running $f_1, \ldots, f_k$ on this x. Let $v_1, \ldots, v_k$ be nodes in the programs such that v_i is in f_i and they are all in the same layer. Then the probability that for all i, f_i when applied to x passes through v_i, is either 0, or at least $1/2^{wk}$.*

Note that this lemma is not trivial, since the different pROBPs run on the same input, so their outputs are not independent.

Proof. The lemma proceeds just like the proof of the previous lemma, except that we work on all k programs at the same time. □

6.6 The Proof for The Baby-INW Generator

We now wish to prove that the inputs of the primary pROBP satisfy the conditions of the dice theorem.

Consider the i-th input to the primary pROBP. Denote it by X_i in the PRG world and Y_i in the uniform world. Recall that X_i and Y_i are random variables over the domain $\{0,1\}^{w^3}$.

Lemma 23. *The statistical difference between X_i and Y_i is at most $\beta = w^3 \cdot \beta_0$.*

The idea behind the proof is that for each $1 \leq j \leq w^3$, it holds by the expander mixing lemma that $\Delta(X_{i,j}, Y_{i,j}) \leq \beta_0$. By a "union bound" on the different coordinates we get the lemma. (Formally we do this using a simple coupling).

Proof. For any j, the value of $f_{i,j}$ is a deterministic function of the i-th chunk. The Expander Mixing Lemma says that $\Delta(X_{i,j}, Y_{i,j}) \leq \beta_0$. (Note that $X_{i,j}$ is the output of $f_{i,j}$ applied to the i-th chunk in the PRG world, and similarly $Y_{i,j}$ is the output of $f_{i,j}$ applied to the i-th chunk in the uniform world). We can easily define a coupling of X_i and Y_i where for each j, $\Pr(X_{i,j} \neq Y_{i,j}) \leq \beta_0$. We now union-bound over the different j and get that under this coupling, $\Pr(X_i \neq Y_i) \leq w^3\beta_0$, and by the standard property of couplings, we get the lemma. □

Lemma 24. *For each element in the output domain $\{0,1\}^{w^3}$, either its mass in X_i and Y_i is both zero, or its mass is at least $\delta = 1/4^{w^4}$ in both of them.*

Proof. Consider Y_i. We have seen in Lemma 23 that the probability of Y_i to take each value is either 0, or at least $1/2^{w^4}$. If the probability of Y_i to take a certain value is 0, then X_i never takes that value as well, because the paths that the branching program can take in the PRG world are a subset of the paths it can take in the uniform world. On the other hand, if the probability of taking the value is at least $1/2^{w^4}$ in the uniform world, then by the previous lemma, this means that the probability that X_i takes this value is at least $1/2^{w^4} - \beta$, which is $\geq 1/4^{w^4} = \delta$. Thus we are finished. □

We now get the final result of this subsection: that the baby-INW generator fools super-pROBPs, and by Lemma 20, it thus fools pROBPs.

Theorem 25. *The baby-INW generator with our parameters $(\varepsilon/\log n)$-fools width-w pROBPs.*

Proof. Simulate the pROBP by a super-pROBP of the same width. By the two previous lemmas, the inputs of the primary pROBP have all the properties required of the dice in the dice theorem. Apply Theorem 9 on the primary pROBP to get the conclusion. □

6.7 The Full Proof for the INW Generator

In this subsection we get the main result of this paper by considering the INW generator rather than its baby version. This is a simple hybrid argument over the layers, where for any layer we use essentially the argument of the last subsection.

Consider the INW generator. Let X be the output of the generator, and Y be a uniformly random string of length n. Consider a sequence of random variables, $X^0, X^1, \ldots, X^\ell$. X^0 is the same as X, while X^ℓ is the same as Y. X^i is obtained by using full randomness for the first i levels of the INW-tree with full randomness, and performing the rest of the layers according to the procedure of the INW generator. This is equivalent to choosing $G_1, \ldots, G_i$ to be complete graphs, and choosing $G_{i+1}, \ldots, G_\ell$ just like in the INW generator.

Consider a pROBP f. We wish to prove that the statistical distance between $f(X^0)$ and $f(X^\ell)$ is at most ε. This is the statement of the main theorem of this paper.

Theorem 26. *The statistical distance between $f(X)$ and $f(Y)$ is at most ε.*

By the triangle inequality, it suffices to prove the next lemma.

Lemma 27. *The statistical distance between $f(X^i)$ and $f(X^{i+1})$ is at most $\varepsilon/\log n$*

Proof. Consider the distributions $f(X^i)$ and $f(X^{i+1})$. Since the top i layers use perfect randomness in both distributions, the distributions can be thought of as a concatenation of 2^i equal-length chunks, where the different chunks are mutually independent. In each chunk, X^i is drawn by iterating the INW procedure $\ell - i$ times, while X^{i+1} is drawn by choosing two independent seeds, and iterating the INW procedure on each of them, $\ell - i - 1$ times.

Now, simulate f by a super-pROBP, working on these chunks. The j-th input of the primary pROBP consists of w^3 bits derived from the j-th chunk. Each of these w^3 bits corresponds to a triplet of nodes of the original pROBP: one at the beginning of the chunk, one at the middle and one at the end. The left half of the j-th chunk is equidistributed in $f(X^i)$ and $f(X^{i+1})$, and so is the right half. Therefore we can follow the same steps as in the analysis of the baby-INW generator and get the lemma. □

7 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. The only place in the analysis of the PRG that we use the fact that we're working on pROBPs rather than ROBPs is in Lemmas 21 and 22. It is easy to construct a ROBP for which these lemmas fail, for example the ROBP that computes the OR function on n bits. In this section we show something more far-reaching: that the INW generator itself *fails* to $(1/2)$-fool the class of width-3 ROBPs if the seed length is $o(\log^2 n)$. We prove this only in the case that an evil adversary is allowed to design the expander-labeling as he pleases. Note that our results do not rule out the possibility that designing an expander-labeling carefully, or just picking the labeling in a pseudorandom way, will allow the INW

generator to fool ROBPs with small seedlength. We remark that the traditional proof of the INW generator, and our proof of the INW generator on pROBPs, work for *any* expander-labeling, even one designed by an evil adversary.

For simplicity, we prove the negative result for the baby-INW generator. It can be strengthened to apply for the INW generator in a straightforward way. Since we work on the baby-INW generator, we just need to specify how the evil adversary designs one expander: the one at the bottom level. Let $m = n/2 \log n$ be the number of chunks, and recall that the number of vertices of the expander is n and that it is d-regular. We wish to prove that $d \geq n^{1/3}$; once we prove this we will be able to get the negative result easily.

Since the expander is d-regular, it contains a matching; choose a matching of size $nd/2m$. Let A denote the left side of the matching and B denote its right side (assign vertices to sides arbitrarily, as long as the two endpoints of each edge in the matching are on different sides). Let $k = \log_2(2m/d)$. (We assume for simplicity that $2m/d$ us a power of 2). Label the vertices of A by all labels that start with k 0's, and label the vertices of B by all labels that start with k 1's. Label the other vertices, and all the edges, arbitrarily.

Now, design a ROBP as follows: the ROBP goes over each chunk, and checks if the first k bits in the left half of the chunk are 0 and if the first k bits in the right half of the chunk are 1. If there is a chunk where both of these conditions hold then the ROBP accepts, otherwise it rejects. It is not hard to see that this function can be implemented by a width-3 ROBP. (In short: keep one sink state exclusively as an "accept track", and make the required checks over each chunk using the other two states).

Now, the probability of accepting in the uniform world is, by the union bound, at most

$$n \cdot (d/2m)^2 \leq O\left(\frac{d^2 \log n}{m}\right) .$$

If $d \leq n^{1/3}$ then this is $o(1)$. On the other hand, in the PRG world, the probability of accepting is $\geq 1 - (1 - 1/m)^m \geq 1 - 1/e$. Therefore, to have any hope to fool width-3 ROBPs, the expander must have degree $d \geq n^{1/3}$. Since in the INW generator the seedlength is roughly $\log n \log d$, we get:

Theorem 28. *The INW generator fails to $(1/2)$-fool width-3 ROBPs, unless its seedlength is $\Omega(\log^2 n)$, as long as an evil adversary is allowed to design the expander-labeling.*

Strictly speaking, to prove this theorem as it is stated for the INW generator rather than the baby-INW generator, we must design the expanders of all levels, and show that all of them must have large degree. The details are routine and we omit them here.

We believe that the expanders can be labeled either carefully or pseudorandomly to bypass the barrier posed by the last theorem, but we do not know how to do this.

8 Conclusions

In this paper, we provide tight upper and lower bounds on how close two distributions can be and still be distinguishable by constant-width branching programs. On one hand, we prove that even with just three states, branching programs can distinguish two distributions that are within a factor of $O(1/\log n)$. On the other hand, we show that any width-w branching program cannot distinguish two distributions that are $o\left((\log n)^{-w}\right)$ from each other.

The above results are nearly tight for constant w. It would be interesting to investigate the case in which w is nonconstant. It would also be interesting to consider the coin and dice problems on other small-space models of computation, such as w-state automata or permutation branching programs. These models are

more restrictive than constant-width branching programs, but are still interesting from the perspective of pseudo-random generators.

Our work seems to suggest that small-space machines are *strictly* stronger when given access to a "clock", which counts the inputs we have seen, and allows us to decide which operation to do, based on the clock's value. It would be interesting to study the influence of a clock on the streaming model: what functions can be approximated using sublogarithmic space, plus a clock? Which problems cannot be solved even with a clock? What interesting generalizations of the proof technique of this paper are needed for such impossibility results?

How well can AC^0, ACC^0 and low-degree polynomials (over $\mathbb{F}_2$) solve the coin problem, and what does this entail? Of particular interest 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). It could be interesting to try to attack this problem using couplings. 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 together with other ideas, give a PRG against small-width ROBP with good seedlength?

Acknowledgements

Both authors are grateful to Sourav Chakraborty, and 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 first author would like to thank David Barrington and the Dartmouth Theory Reading Group for helpful discussions. The second author would like to thank Boaz Barak, Swastik Kopparty, Shachar Lovett, Avi Wigderson and David Xiao for helpful discussions. Some of this research was done while the authors were visiting Peter 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] 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*, pages 544–553, 1990.

[2] Kazuyuki Amano. Bounds on the size of small depth circuits for approximating majority. In *Proc. 36th International Colloquium on Automata, Languages and Programming*, pages 59–70, 2009.

[3] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *JCSS*, 38(1):150–164, 1989.

[4] Andrej Bogdanov, Zeev Dvir, Elad Verbin, and Amir Yehudoff. Pseudorandomness for width-2 branching programs. Manuscript, 2009.

[5] Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudoff. Pseudorandom generators for regular branching programs. Manuscript, 2010.

[6] Funda Ergün, Ravi Kumar, and Ronitt Rubenfeld. On learning bounded-width branching programs. In *Proc. 8th International Conference on Learning Theory*, pages 361–368, 1995.

[7] Shlomo Horry, Nathan Linial, and Avi Wigderson. Expander graphs and their applications, 2006.

[8] Russel Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proc. 26th Annual ACM Symposium on the Theory of Computing*, pages 356–364, 1994.

[9] Michael Luby and Avi Wigderson. Pairwise independence and derandomization. *Foundations and Trends in Theoretical Computer Science*, 1(4):237–301, 2006.

[10] Raghu Meka and David Zuckerman. Small-bias spaces for group products. In *Proc. 13th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 658–672, 2009.

[11] Raghu Meka and David Zuckerman. Pseudorandom generators for polynomial threshold functions. In *Proc. 42nd Annual ACM Symposium on the Theory of Computing*, 2010. To Appear.

[12] Noam Nisan. RL $\subseteq$ SC. In *Proc. 24th Annual ACM Symposium on the Theory of Computing*, pages 619–623, 1995.

[13] Ryan O'Donnell and Karl Wimmer. Approximation by DNF: Examples and counterexamples. In *Proc. 34th International Colloquium on Automata, Languages and Programming*, pages 195–206, 2007.

[14] Omer Reingold. Randomness vs memory: Prospects and barriers, 2009. `http://intractability.princeton.edu/attachments/omer_reingold.pptx`.

[15] Emanuele Viola. On approximate majority and probabilistic time. *Computational Complexity*, 18(3):337–375, 2009.

A Technical Lemmas

Lemma 29 (Restatement of Lemma 6)**.** *In any good branching program for the coin problem, $s < t$ implies $f(Y|s) < f(Y|t)$.*

Proof. Suppose for the sake of contradiction that there exist states $s < t$ such that $f(Y|s) > f(Y|t)$. Fix some s' and b such that $s'(b) = s$. Then, moving this transition such that $s'(b) := t$ increases $f(X|s')$ and decreases $f(Y|s')$. Using equations (1) and (2), it follows that $f(X)$ increases and $f(Y)$ decreases. Hence, we improve $\Delta(f(X), f(Y))$, which contradicts the optimality of f. □

Lemma 30 (Restatement of Lemma 7)**.** *Let s be some state at level i, and assume all states are ordered canonically. Suppose $a, b \in [m]$ such that $p_{i,a} \geq q_{i,a}$ but $p_{i,b} \leq q_{i,b}$. Then in any* good *branching program, $s(a) \geq s(b)$.*

Proof. Fix two states t_1, t_2 in level $(i+1)$ such that $t_1 > t_2$, and consider the following cases:

1. $s(a) = t_2,\ s(b) = t_1$
2. $s(a) = s(b) = t_2$
3. $s(a) = s(b) = t_1$

Note that $s(a) < s(b)$ only in the first case. Our goal is to show that this case is *never* optimal. Combining equations 1 and 2, we see that

$$f(X) = \sum_{j=1}^{w} \beta_X(s_j) \sum_{e \in [m]} p_{i,e} \cdot f(X|s(e)) \; .$$

A similar equation exists for $f(Y)$. To show that case 1 is never optimal, it suffices to show that $\Delta(f(X), f(Y)) = f(X) - f(Y)$ is always maximized in one of the other cases.

Let S denote the set of states at level i except state s. Let $M := [m] \setminus \{a, b\}$ be the set of outcomes that are neither a or b. Then, let

$$A := \sum_{j \in S} \beta_X(s_j) f(X|s_j) - \beta_Y(s_j) f(Y|s_j)$$

and

$$B := \sum_{e \in M} \beta_X(s) p_{i,e} f(X|s(e)) - \beta_Y(s) q_{i,e} f(Y|s(e)) \; .$$

Intuitively, A is the contribution to $f(X) - f(Y)$ from states in S, and B is the contribution from state s and outcomes that are neither a nor b. Note that since we change only $s(a)$ and $s(b)$ as we change cases, A and B remain invariant, and we may write $f(X) - f(Y)$ as

$$\begin{aligned} f(X) - f(Y) = A + B \quad &+ \quad \beta_X(s) \left[p_{i,a} f(X|s(a)) + p_{i,b} f(X|s(b)) \right] \\ &- \quad \beta_Y(s) \left[q_{i,a} f(Y|s(a)) + q_{i,b} f(Y|s(b)) \right] . \end{aligned}$$

To calculate $f(X) - f(Y)$ for each case, a few more definitions will aid the computation. Let $\delta_X := f(X|t_1) - f(X|t_2)$ and $\delta_Y := f(Y|t_1) - f(Y|t_2)$. Note that the canonical ordering of states in level $(i+1)$ gives $\delta_x, \delta_y \geq 0$. Finally, let

$$Z := \beta_X(s) \cdot f(X|t_2) \cdot (p_{i,a} + p_{i,b}) - \beta_Y(s) \cdot f(Y|t_2) \cdot (q_{i,a} + q_{i,b}) \; .$$

Then, in each case, we have:

1. $f(X) - f(Y) = A + B + Z + \beta_X(s) \delta_X p_{i,b} - \beta_Y(s) \delta_Y q_{i,b}$.
2. $f(X) - f(Y) = A + B + Z$.
3. $f(X) - f(Y) = A + B + Z + \beta_X(s) \delta_X (p_{i,a} + p_{i,b}) - \beta_Y(s) \delta_Y (q_{i,a} + q_{i,b})$.

Suppose for the sake of contradiction that $f(X) - f(Y)$ is maximized in case 1. Then, comparing case 1 to case 2, we see that

$$\beta_S(s) \delta_X p_{i,b} > \beta_Y(s) \delta_Y q_{i,b} \iff \frac{\beta_X(s)}{\beta_Y(s)} > \frac{q_{i,b} \delta_Y}{p_{i,b} \delta_X} \; .$$

Comparing case 1 to case 2, we see that

$$\begin{aligned} &\beta_X(s) \delta_X p_{i,b} - \beta_Y(s) \delta_Y q_{i,b} \geq \beta_X(s) \delta_X (p_{i,a} + p_{i,b}) - \beta_Y(s) \delta_Y (q_{i,a} + q_{i,b}) \\ &\quad \iff \beta_X(s) \delta_X p_{i,a} > \beta_Y(s) \delta_Y q_{i,a} \\ &\quad \iff \frac{\beta_X(s)}{\beta_Y(s)} \geq \frac{\delta_Y q_{i,a}}{\delta_X p_{i,a}} \; . \end{aligned}$$

Combining these two inequalities, we see that

$$\frac{\delta_Y}{\delta_X} < \frac{q_{i,b}\delta_Y}{p_{i,b}\delta_X} < \frac{\beta_X(s)}{\beta_Y(s)} < \frac{q_{i,a}\delta_Y}{p_{i,a}\delta_X} < \frac{\delta_Y}{\delta_X}\,,$$

where the first and last inequalities come from our initial assumptions that $p_{i,a} > q_{i,a}$ but $p_{i,b} < q_{i,b}$. Thus, we have $\delta_Y/\delta_X < \delta_Y/\delta_X$, a contradiction. □

Lemma 31 (Restatement of Lemma 8)**.** *Let $s_1, \ldots, s_d$ and $t_1, \ldots, t_d$ denote the support of two consecutive levels of the 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 $1 \le i \le d$ and for all $e \in [m]$.*

Proof. Suppose for the sake of contradiction that there are no collisions among $s_1, \ldots, s_d$. Then for every t and every $e \in [m]$, there exists s such that $s(e) = t$. Consider any $a, b \in [m]$ such that a is more likely in X_i but b is more likely in Y_i. Fix i such that $s_i(b) = t_d$. By Lemma 7, it follows that $s_i(a) = t_d$ as well. Proving that $s_j(a) = s_j(b)$ for all other j follows by induction on $\{s_j : j \neq i\}$ and $\{t_j : j \neq d\}$. Hence, the transitions form a permutation. It remains to show this permutation is the identity. This follows from the ordering of $s_1, \ldots, s_d$ and from equation (2). □

B Dice Theorem for Small Probabilities

Theorem 32 (Restatement of Theorem 14)**.** *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*

$$\frac{1}{1+\beta} \le \frac{p_{ie}}{q_{ie}} \le 1 + \beta$$

for all $1 \le i \le n$ and $e \in [m]$. If a width-w ROBP distinguishes X and Y, then $\beta = \Omega\left((\log(mn))^{-3w}\right)$.

Proof. This proof closely follows the proof of the Coin Theorem. We begin by rounding our random variables. This time, there could be up to m small probabilities; we need to round each. Define X_i' to take the random variable taking value e with probability $p'_{i,e}$, where $p'_{i,e}$ is defined by:

$$p'_{ie} \;=\; \begin{cases} 0\,, & \text{if } \max(p_{ie}, q_{ie}) < 1/mn^2\,, \\ p_{ie} + \sum_{e:max(p_{ie},q_{ie})<1/mn^2} p_{ie}\,, & \text{if } e = e^*\,, \\ p_{ie} & \text{otherwise,} \end{cases}$$

where $e^* := \operatorname{argmax}_e\{p_{ie}\} \cup \{q_{ie}\}$ is the most likely value for X_i' or Y_i'. Define Y_i' in terms of Y_i in a similar manner. Simple calculations show that $\Delta(X_i, X_i') \le 1/n^2$, hence by a hybrid argument $\Delta(X, X') < 1/n$. The same bound holds for $\Delta(Y, Y')$. Hence, we can assume that the probabilities are never *too* low.

Suppose that f is a ROBP such that $\Delta(f(X), f(Y)) = o\left((\log(mn))^{-3w}\right)$. Then $\Delta(f(X'), f(Y')) = o\left((\log(mn))^{-3w}\right)$ as well. Next, we simulate each die roll in f by a ROBP that uses only balanced *coins*. There are many ways to perform this simulation; our primary goal is to minimize the blowup in width, since the lower bound we have from the Coin Theorem has a w-factor in the exponent. Our secondary motivation in this construction is to make things as simple as possible—aside from the emphasis on minimizing the width-blowup, we do not attempt to tighten bounds as much as possible.

Recall in the proof of the strong coin theorem that at each step, we "extracted" the low probability by constructing an AND of k completely fair coins with a final coin that was only slightly biased. We'll

simulate a die roll in a similar manner. Specifically, we'll serially extract p_{ie} for each $e \in [m]$. Because we extract these probabilities serially, we need an additional w accumulator states, one for each state in the next level of f. In our simulation, the transitions from these states all point directly downward; i.e., once we're in some accumulator state t_i, we remain there for the rest of the simulation of the current die.

To begin the simulation, pick some $e \in [m]$, and construct a width-2 ROBP that accepts with probability p_{ie} in the same manner as in the proof of the strong coin theorem. Replace the accept state with the accumulator state for $s(e)$. The reject state becomes the start state for the width-2 ROBP that computes the next $e \in [m]$.

One subtle complication arises. The start state for the next extraction is itself now biased—we reach it with probability $p'_i := 1 - p_{ie}$ in world 1, and with probability $q'_i := 1 - q_{ie}$ in world 2. Care must be taken in the ordering of which probabilities to extract to ensure that the bias does not become unexpectedly large. Suppose without loss of generality that $p' > q'$, and let e be the remaining element such that $p_{ie} > q_{ie}$ and p_{ie}/q_{ie} is maximized. This is the next element to extract. Let $k := \lceil -\log(p_{ie}/p') \rceil$, and construct a width-2 ROBP that computes the AND of $k+1$ coins. As in the proof of the strong coin theorem, the first k coins are fair coins in both worlds. The $(k+1)$th coin will have probability $2^{-k}p_{ie}/p'$ in world 1 and probability $2^{-k}q_{ie}/q'$ in world 2. Note that the bias in this last coin is $(p_{ie}/q_{ie})/(p'/q')$. Note that

$$\begin{aligned} 1 &\leq \frac{p'}{q'} \\ &= \frac{\sum_{e \in S} p_{ie}}{\sum_{e \in S} q_{ie}} \\ &\leq \frac{(p_{ie}/q_{ie}) \sum_{e \in S} q_{ie}}{\sum_{e \in S} q_{ie}} \\ &= \frac{p_{ie}}{q_{ie}}, \end{aligned}$$

where the final inequality follows from our choice of e. It follows that

$$1 \leq \frac{p_{ie}}{q_{ie}} \frac{p'}{q'} \leq \frac{p_{ie}}{q_{ie}}.$$

Therefore, the bias in each outcome remains at most $1 + \beta$.

Putting this together, we get that for each level in f, we use at most $O(m \cdot \log(mn))$ coins in the simulation, as we know by the rounding argument that $p_{ie}, q_{ie} > 1/mn^2$. Hence, the new ROBP has length $O(nm \log(mn))$. Furthermore, it has width-$3w$, since for each state in f we use a width-2 ROBP, and we require w additional accumulator states. Hence, there is a width-$3w$, length-$O(mn \log(mn))$ ROBP that computes f and uses only balanced coins. We know by theorem 5 that if such a ROBP distinguishes β, then

$$\beta = \Omega\left(\log(mn \log(mn))^{-3w}\right) = \Omega\left(\log(mn)^{-3w}\right).$$

The lower bound on β for f follows. □

C Proof of Upper Bound

For the sake of simplicity, we give the construction for widths $w = 3$ before generalizing to any constant width $w = O(1)$.

C.1 Width-3 Branching Programs Distinguish $\beta = O(1/\log n)$

Fix m such that $m \log m = n$, and consider an OR-AND tree consisting of a single OR gate at the root and m AND gates at the second level, each with fan-in $\log m$. Recall that a width-3 branching program can compute this function. In this subsection, we show an OR-AND tree that distinguishes $\beta = 1/\log m = O(1/\log n)$.

Suppose that the coins are heads with probability $1/2 + \beta$. Then, we have

$$\Pr\left[\wedge_{i=1}^{\log m} x_i = 1\right] = \left(\frac{1}{2}(1+2\beta)\right)^{\log m} = \frac{1}{m}\left(1+2\beta\right)^{\log m} \geq \frac{1}{m} e^{\beta \log m} = \frac{e}{m} ,$$

where the inequality uses $1 - x \geq e^{-x/2}$, which holds for all $0 < x < 1/2$. Therefore, we have

$$\begin{aligned} \Pr\left[\vee_{i=1}^{m} \wedge_{j=1}^{\log m} x_{i,j} = 0\right] &\leq \left(1 - \frac{e}{m}\right)^m \\ &\leq e^{\left(-\frac{em}{m}\right)} \\ &= e^{-e} < 0.066 . \end{aligned}$$

Therefore, when the coins are heads with probability $1/2 + \beta$, we accept with probability at least $1 - 0.066 = 0.934 > 2/3$. Alternatively, suppose that the coins are heads with probability $1/2 - \beta$. Then, we have

$$\Pr\left[\wedge_{i=1}^{\log m} x_i = 1\right] = (1/2 - \beta)^{\log m} = \frac{1}{m}\left(1-2\beta\right)^{\log m} \leq \frac{1}{e^2 m} .$$

Hence,

$$\Pr\left[\vee_{i=1}^{m} \wedge_{j=1}^{\log m} x_{i,j} = 0\right] \geq \left(1 - \frac{1}{e^2 m}\right)^m \geq e^{-\frac{1}{e^2}} \geq 0.934 .$$

C.2 Width-w Branching Programs

For the general case, we closely follow the approach of Amano [2]. Fix $W = (w_1, \ldots, w_d)$, and consider an AND-OR tree such that each node at level i has fan-in w_i.

Definition 33 (Amano [2]). *For $k = 0, \ldots, d$ and $i \in \{0, 1\}$, let $A_k^i : [0,1] \to [0,1]$ be a series of functions defined as follows:*

- $A_0^i(p) = p$ *for all* $p \in [0,1]$.
- $A_k^1(p) = \left(A_{k-1}^1(p)\right)^{w_k}$ *for all odd* k *and all* $p \in [0,1]$.
- $A_k^0(p) = \left(A_{k-1}^0(p)\right)^{w_k}$ *for all even* k *and all* $p \in [0,1]$.
- $A_k^0(p) + A_k^1(p) = 1$ *for all* k *and for all* $p \in [0,1]$.

The function $A_k^i(p)$ gives the probability that a subtree of depth k outputs i, given that each input is 1 with probability p. Amano used the above function and a careful choice of W to give an optimal size bound for depth d AC^0 circuits that approximate the Majority function.

Couched in our language, Amano's construction fixes a depth d and $\beta := \varepsilon/\sqrt{n}$ and gets an optimal $\exp\left(\Theta\left(n^{1/(2d-2)}\right)\right)$ circuit size to distinguish β. We wish to fix a depth $d := w - 1$ and input size n and determine the best β we can distinguish with circuits of such depth and size.

Fix $m := (\log n)/(d-1) - \log\log n + \log(d-1)$, and set $w_1 := m$, $w_k := (\ln 2) \cdot m \cdot 2^m$ for $2 \leq k \leq d-1$, and $w_d := (\ln 2) \cdot 2^m$. Note that the number of inputs to this circuit equals $\prod w_i$. Furthermore, note that

$$m = \log\left(n^{1/(d-1)}\right) + \log\left(\frac{d-1}{\log n}\right) = \log\left(\frac{(d-1)n^{1/(d-1)}}{\log n}\right) .$$

Therefore, we have $2^m = (d-1)n^{1/(d-1)}/\log n$, hence

$$\begin{aligned}
\prod_{1\leq k\leq d} w_i &= m \cdot ((\ln 2)m2^m)^{d-2} \cdot (\ln 2)2^m \\
&= ((\ln 2) \cdot m \cdot 2^m)^{d-1} \\
&< \left(\frac{\log n}{d-1} \cdot \frac{(d-1)n^{1/(d-1)}}{\log n}\right)^{d-1} \\
&= n .
\end{aligned}$$

Hence the circuit family that we describe accepts at most n inputs. We claim that these circuits distinguish $\beta = m^{-(d-1)} = O\left((\log n)^{-(w-2)}\right)$. To prove this claim, we require two lemmas that analyze the functions A_k^i. We leave their proofs to the Appendix and note that these lemmas closely follow Lemmas 3 and 4 of Amano [2].

Lemma 34. *Let $z := (\ln 2) \cdot m \cdot 2^m$. Suppose that*

$$A \geq 2^{-m}\left(1 + cm^{-k}\right)$$

for some $1 \leq k < d-1$ and some constant $c > 0$. If $k > 1$, then

$$(1-A)^z \leq 2^{-m}\left(1 - (c/3)m^{-k+1}\right).$$

If $k = 1$, then

$$(1-A)^z \leq 2^{-m} \cdot 2^{-c}.$$

Proof. Using $1 - x \leq e^{-x}$ for all real x, we have

$$\begin{aligned}
(1-A)^z &\leq \exp(-zA) \\
&\leq \exp\left(-(\ln 2) \cdot m \cdot 2^m \cdot \left(2^{-m}(1 + cm^{-k})\right)\right) \\
&= 2^{-m(1+cm^{-k})} \\
&= 2^{-m} \cdot 2^{-(cm^{-k+1})}.
\end{aligned}$$

If $k = 1$, then $(1-A)^z \leq 2^{-m} \cdot 2^{-c}$. Otherwise, using $1 - x \geq 2^{-3x}$, which holds for all real $0 \leq x \leq 1$, we have

$$(1-A)^z \leq 2^{-m} \cdot 2^{-cm^{-k+1}} \leq 2^{-m} \cdot \left(1 - \frac{c}{3}m^{-k+1}\right) .$$

□

Lemma 35. *Let $z := (\ln 2) \cdot m \cdot 2^m$. Suppose that*

$$A \leq 2^{-m}\left(1 - cm^{-k}\right)$$

for some $1 \leq k < d-1$ and some constant $c > 0$. If $k > 1$, then

$$(1-A)^z \geq 2^{-m}\left(1 + \frac{c}{3}m^{-k+1}\right).$$

If $k = 1$, then

$$(1-A)^z \geq 2^{-m} \cdot 2^{c/3}.$$

Proof. Using $1 - x \geq e^{-x-x^2} = e^{-x(1+x)}$, which holds for all $x > 0$, we have

$$\begin{aligned}
(1-A)^z &\geq \exp\left(-zA(1+A)\right) \\
&\geq \exp\left(-(\ln 2)m(1-cm^{-k})\left(1 + 2^{-m}(1-cm^{-k})\right)\right) \\
&= 2^{-m}\exp\left((\ln 2)\left(cm^{-k+1} - m2^{-m}(1-cm^{-k})^2\right)\right) \\
&\geq 2^{-m} \cdot 2^{\frac{c}{1.1}m^{-k+1}},
\end{aligned}$$

where the last inequality holds for sufficiently large n (and hence sufficiently large m). When $k = 1$, then we have $(1-A)^z \geq 2^{-m} \cdot 2^{c/1.1} > 2^{-m} \cdot 2^{c/3}$ and we're done. Otherwise, using $1 + x\ln 2 \leq 2^x$, we have

$$(1-A)^z \geq 2^{-m}\left(1 + \frac{c\ln 2}{1.1}m^{-k+1}\right) \geq 2^{-m}\left(1 + \frac{c}{3}m^{-k+1}\right).$$

Either way, the proof is complete. □

We are now ready to prove the main upper bound. We restate the theorem here for clarity.

Theorem 36. *For all constant w, there exists $\beta = O\left((\log n)^{-(w-2)}\right)$ such that width-w branching programs distinguish β.*

Proof. Construct a depth-$d := w-1$ AND-OR tree in the manner described above. Fix $\beta := c(\log n)^{-(w-2)}$ for some constant c to be determined later. Recall that width-w branching programs can simulate depth-$(w-1)$ AND-OR trees, and so it remains to show that

$$A_d^1\left(\frac{1}{2} + \beta\right) \geq 2/3 \quad \text{and} \quad A_d^0\left(\frac{1}{2} - \beta\right) \leq 1/3\,.$$

Assume without loss of generality that d is odd (the case in which d is even is proved similarly). Note that $A_0^1(1/2+\beta) = 1/2+\beta$ and $A_0^1(1/2-\beta) = 1/2-\beta$. Using $1+x \geq e^{x/2}$, which holds for all $0 < x < 1/2$, we have

$$A_1^1\left(\frac{1}{2} + \beta\right) = \left(\frac{1}{2}(1+2\beta)\right)^m \geq 2^{-m}\exp(\beta m) = 2^{-m}\exp(cm^{-w+3}) \geq 2^{-m}\left(1 + cm^{-w+3}\right).$$

Therefore, by Lemma 34, we have $A_2^0(1/2+\beta) \leq 2^{-m}\left(1 - (c/3)m^{-w+4}\right)$, hence by Lemma 35, we have $A_3^1(1/2+\beta) \geq 2^{-m}\left(1 - (c/9)m^{-w+5}\right)$. Continue alternating between Lemmas 34 and 35 until

$$A_{d-2}^1\left(\frac{1}{2} + \beta\right) \geq 2^{-m}\left(1 + \frac{c}{3^{d-3}}m^{-w+d}\right) = 2^{-m}\left(1 + \frac{c}{3^{d-3}}m^{-1}\right).$$

With one final invocation of Lemma 34, we get

$$A^0_{d-1}\left(\frac{1}{2}+\beta\right) \le 2^{-m}\cdot 2^{-c/3^{d-3}}.$$

Finally, because the root node has $(\ln 2)\cdot 2^m$ children, we have

$$\begin{aligned} A^1_d\left(\frac{1}{2}+\beta\right) &= \left(1-A^0_{d-1}\left(\frac{1}{2}+\beta\right)\right)^{(\ln 2)2^m} \\ &\ge \left(1-2^{-m}\cdot 2^{-c/3^{d-3}}\right)^{(\ln 2)2^m} \\ &\ge \exp\left(-(\ln 2)2^m\cdot 2^{-m}\cdot 2^{-c/3^{d-3}}\cdot 2\right) \\ &= 2^{-2^{1-c/3^{d-3}}}\ . \end{aligned}$$

Setting $c := 3^{d-2}$, we get that

$$A^1_d\left(\frac{1}{2}+\beta\right) \ge 2^{-1/4} > 2/3\ .$$

It remains to prove that $A^0_d(1/2-\beta)\ge 2/3$. Using $e^{-2x}\le 1-x\le e^{-x}$, which holds for all $0<x<1/2$, we see that

$$\begin{aligned} A^1_1\left(\frac{1}{2}-\beta\right) = \left(\frac{1}{2}(1-2\beta)\right)^m &\le 2^{-m}e^{-2\beta m} \\ &\le 2^{-m}(1-\beta m) \\ &= 2^{-m}\left(1-cm^{-w+3}\right)\ . \end{aligned}$$

By Lemma 35, it follows that $A^0_2(1/2-\beta)\ge 2^{-m}\left(1+(c/3)m^{-w+4}\right)$, and hence by Lemma 34, we have $A^1_3(1/2-\beta)\le 2^{-m}\left(1-(c/9)m^{-w+5}\right)$. Again alternating between Lemmas 34 and 35, we continue until

$$A^1_{d-2}\left(\frac{1}{2}-\beta\right) \le 2^{-m}\left(1-\frac{c}{3^{d-3}}m^{-1}\right)\ .$$

By Lemma 35, we see that

$$A^0_{d-1}\left(\frac{1}{2}-\beta\right) \ge 2^{-m}\cdot 2^{\frac{c}{3^{d-2}}}\ .$$

Recall that the root node has $(\ln 2)\cdot 2^m$ children. Hence, we see that

$$\begin{aligned} A^1_d\left(\frac{1}{2}-\beta\right) &= \left(1-A^0_{d-1}\left(\frac{1}{2}-\beta\right)\right)^{(\ln 2)2^m} \\ &\le \left(1-2^{-m}\cdot 2^{c/3^{d-2}}\right)^{(\ln 2)2^m} \\ &\le \exp\left(-(\ln 2)2^m\cdot 2^{-m}\cdot 2^{c/3^{d-2}}\right) \\ &= 2^{-2^{c/3^{d-2}}} \\ &= 2^{-2} < 1/3\ . \end{aligned}$$

□