Pseudorandom Bits for Oblivious Branching Programs

Rohit Gurjar* Ben Lee Volk*

Abstract

We construct a pseudorandom generator which fools read-k oblivious branching programs and, more generally, any linear length oblivious branching program, assuming that the sequence according to which the bits are read is known in advance. For polynomial width branching programs, the seed lengths in our constructions are $\tilde{O}(n^{1-1/2^{k-1}})$ (for the read-k case) and $O(n/\log\log n)$ (for the linear length case). Previously, the best construction for these models required seed length $(1-\Omega(1))n$.

*Department of Computer Science, Tel Aviv University, Tel Aviv, Israel, E-mails: rohitgurjar0@gmail.com, benleevolk@gmail.com. The research leading to these results has received funding the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 257575 and from the Israel Science Foundation (grant number 552/16).

1 Introduction

A *Pseudorandom Generator* (PRG, for short), for a class of boolean functions $\mathcal{C}$, is a family of efficiently computable functions $G_n : \{0,1\}^s \to \{0,1\}^n$ which fools functions in the class $\mathcal{C}$, in the sense that for all $f : \{0,1\}^n \to \{0,1\}$ in $\mathcal{C}$,

$$\left| \Pr_{x \sim U_s} [f(G_n(x)) = 1] - \Pr_{x \sim U_n} [f(x) = 1] \right| \leq \varepsilon,$$

where U_k denotes the uniform distribution on $\{0,1\}^k$, and $s = s(n, \varepsilon)$ is called the *seed length*.

A long line of work in complexity theory studies construction of PRGs for restricted classes of functions. One concrete motivation for these results is obtaining a "black box" derandomization of randomized algorithms from these restricted classes. More generally, this research program is aimed at shedding light on the power of randomness in computation in general, and on the limits of the use of randomness in algorithms with bounded resources.

One notable example in this area in Nisan's PRG for logarithmic space machines [Nis92]: Nisan constructed a PRG with seed length $O(\log^2(n))$ which fools RL machines, that is, logarithmic space machines with read-once access to its randomness. More generally, Nisan's PRG also works in the non-uniform setting, and fools any function which is computed by a small width *read once oblivious branching program* (see Section 2.2 for a formal definition).

In this more general setting, the seed length of Nisan's generator is $O(\log(n) \cdot (\log(n/\varepsilon) + \log(w)))$, where w denotes the *width* of the branching program. Impagliazzo, Nisan and Wigderson [INW94] gave a different construction with matching parameters, but to this day, and despite a large body of work on this topic, there is no better construction known for this model.

In the lack of better results, one possible avenue for improvement would be to obtain improved bounds in more restricted settings. One such challenge is to obtain an improved seed length in the bounded width case, i.e., when $w = O(1)$. Indeed, some progress was made in this setting, assuming more restrictive properties on the branching program ([BRRY14, De11, KNP11, RSV13, Ste12, SVW14]).

Another way to extend these results is to obtain PRGs against stronger models of computation. The saving in randomness in the works of Nisan [Nis92] and Impagliaazo, Nisan and Wigderson [INW94] follows from the fact that in the execution of a read-once branching program on a specific input, each bit is accessed only once, and furthermore, the order of access is known in advance to the designer of the PRG. It is natural to ask to what extent these restrictions can be removed, en route to constructing PRGs against more general classes of computation. In Section 1.3, we review some of the progress made in this setting.

1.1 Algebraic vs. Boolean Pseudorandomness

We now make a small detour and review some relevant results from algebraic complexity. Polynomial Identity Testing (PIT) is the problem of deciding, given an algebraic computation device which computes a formal polynomial using the arithmetic operations $+$ and $\times$, whether it computes the zero polynomial. This problem admits an easy randomized algorithm which follows from the Schwartz-Zippel-DeMillo-Lipton Lemma [Zip79, Sch80, DL78], and it is a major open problem to find an efficient deterministic algorithm, even for restricted classes of algebraic computation.

The algebraic analog of constructing PRGs is black-box identity testing: here, the goal is to construct a *hitting set*, which is a small and efficiently constructible set $\mathcal{H}$ such that for every non-zero polynomial f in the class, there exists $\alpha \in \mathcal{H}$ such that $f(\alpha) \neq 0$. It is not hard to show (see, e.g., [SY10]) that this is equivalent to constructing a *generator*, which is a polynomial map

$G : \mathbb{F}^s \to \mathbb{F}^n$ of small degree, such that for every non-zero f, $f \circ G$ is not the zero polynomial. The quality of the generator is measured by the seed length s and the degree of the polynomial map G.

The algebraic analog of a read-once oblivious branching program is a model called read-once oblivious *algebraic* branching programs (ROABPs). We omit the exact definition of this model from this informal introduction. Forbes and Shpilka [FS13] obtained a hitting set of quasi-polynomial size for this model, or equivalently, a generator whose number of variables s is $O(\log n)$, and whose degree is polynomial in the number of variables n, the width w and the degree d of the ROABP. Quantitatively, this is comparable to Nisan's generator, and indeed, the intuition behind the Forbes-Shpilka generator is similar to Nisan's proof.

However, it interesting to note that both the challenges that were mentioned earlier in the context of boolean pseudorandomness have been met in the algebraic world: Forbes, Shpilka and Saptharishi [FSS14] obtained a hitting set of quasi-polynomial size which works even when the order in which the variables are read is unknown. The construction was later improved by Agrawal, Gurjar, Korwar and Saxena [AGKS15], whose hitting set size matches the hitting set for the known order case.

In the bounded width setting, Gurjar, Korwar and Saxena [GKS17] obtained a hitting set of polynomial size (over characteristic 0, and assuming the order is known) by leveraging intuition for the INW generator [INW94].

It is thus interesting to see to what extent the progress in the algebraic world can help in obtaining improved PRGs for boolean computational devices.

1.2 Results and Techniques

In [AFS+16], Anderson et al. obtained a subexponential time PIT algorithm for the model of read-k oblivious algebraic branching program. Here, we adapt their techniques to the boolean analog of this model, and prove the following.

Theorem 1.1. *For every $k \geq 2$, there exists an efficiently computable function $G : \{0,1\}^s \to \{0,1\}^n$, where*

$$s = O\left(\exp(k^2) \cdot n^{1-1/2^{k-1}} \cdot \log(n) \cdot (\log(n/\varepsilon) + k \log w)\right),$$

which ε-fools every function f which is computable by a width-w oblivious read-k branching program, when the sequence according to which the variables are read in the branching program is known in advance.

The saving in randomness is more noticeable when k is small. However, by exploiting the fact that the bound on s remains sublinear even for slightly superconstant k, we can prove the following.

Theorem 1.2. *There exists an efficiently computable function $G : \{0,1\}^s \to \{0,1\}^n$, for $s = O(n/\log\log n)$, which ε-fools every function f which is computable by an oblivious branching program of length $O(n)$ and width $\mathsf{poly}(n)$, when the sequence according to which the variables are read in the branching program is known in advance.*

Our techniques are mostly adaptation of the techniques used by [AFS+16] in the algebraic setting. To illustrate them, consider first a branching program that reads its variables twice in the order $x_1, x_2, \ldots, x_n, x_1, x_2, \ldots, x_n$. In the algebraic case, is it not very hard to show that such a width w algebraic branching program can be simulated by a width $\mathsf{poly}(w)$ ROABP in the variable order $x_1, \ldots, x_n$, and this fact serves as the starting point of the construction in [AFS+16]. This fact, however, is no longer true for boolean branching programs. As an example, consider the "address function" which receives as an input $y \in \{0,1\}^n$ and $z \in \{0,1\}^{\log n}$, interprets z as an

integer in $[n]$ and outputs y_z. In the variable order $y_1, y_2, \ldots, y_n, z_1, \ldots, z_{\log n}$, this function requires exponential width, since the branching program essentially has to remember all y bits before it sees z bits. But if the branching program is allowed to read the input twice in this order, polynomial width suffices.

Fortunately, it turns out that the generator by Impagliazzo, Nisan and Wigderon [INW94] also fools branching programs that read their input twice in the same order. This is essentially because this generator is useful against any model which can be simulated by a "low communication" protocol on a "simple" network topology (see Section 2.4 for further discussion). Thus, this model already has a PRG with seed length $\mathsf{polylog}(n)$ (assuming $w = \mathsf{poly}(n)$).

Generalizing a bit further, we can consider branching programs that read their input in the order $x_1, x_2, \ldots, x_n, x_{\pi(1)}, \ldots, x_{\pi(n)}$ for some arbitrary permutation π. Here, the basic idea in [AFS+16] was to argue, using the Erdős-Szekeres Theorem, that the sequence $x_{\pi(1)}, \ldots, x_{\pi(n)}$ must contain either a monotonically increasing sequence of length $\sqrt{n}$, or a monotonically decreasing sequence of the same length. Assuming that the sequence is increasing (the decreasing case is handled similarly), we obtain a set of $\sqrt{n}$ variables $y_1, \ldots, y_{\sqrt{n}}$ such that the branching program, restricted to only these variables, is exactly of the form required by the previous argument.

We continue inductively to find a (slightly shorter) monotone sequence in the remaining variables. This process can be shown to terminate after $O(\sqrt{n})$ applications of the Erdős-Szekeres Theorem, and the final generator is obtained by applying the INW generator with an independent seed to each of the $O(\sqrt{n})$ sets obtained in this process, for a total seed length of $O(\sqrt{n} \cdot \mathsf{polylog}(n))$.

Similarly, one can consider read-k branching programs whose reading order is

$$x_1, \ldots, x_n, x_{\pi_1(i)}, \ldots, x_{\pi_1(n)}, \ldots, x_{\pi_{k-1}(1)}, \ldots, x_{\pi_{k-1}}(n), \tag{1.3}$$

for $k-1$ permutations $\pi_1, \ldots, \pi_{k-1}$. By iteratively applying the Erdős-Szekeres Theorem, we can find a sequence of length $n^{1/2^{k-1}}$ which is monotone in each of the k reads, argue as before that the branching program restricted to these variables is fooled by the INW generator, and continue the argument as before. The iterative application of the Erdős-Szekeres argument accounts for most of the loss in the parameters, but it is unfortunately unavoidable in this approach (see the discussion in [AFS+16]).

More generally, we want to handle any sequence in which every variable appears at most k times, even if it is not of the form (1.3). To do this, Anderson et al. [AFS+16] defined the notion of a "k-regularly-interleaving sequence", which we define in Section 2.3, and enables us to similarly partition the set of variables X into t disjoint sets $Y_1, \ldots, Y_t$, such that the INW fools every branching programs in the variables of Y_i under any restriction of the variables in all other sets, while t remains sublinear in n.

1.3 Related Work

There are several works that consider the problem of constructing pseudorandom distributions for related models. Here we review some of the related results.

Impagliazzo, Meka and Zuckerman [IMZ12] constructed a very general PRG that fools *every* branching program with s vertices, with seed length $s^{1/2+o(1)}$. For the case of branching programs of length $O(n)$, the size is $O(w \cdot n)$, and thus this is meaningful only when $w = o(n)$, whereas our result remains non-trivial for any polynomial, and even super-polynomial, width.

Bogdanov, Papakonstantinou and Wan [BPW11, BPW12] constructed explicit PRGs for read-once branching programs and more generally for oblivious branching programs of linear length. In their construction, the seed length is $(1 - \Omega(1))n$, whereas our seed length is sublinear in n. However, an advantage of their construction is that it works even when the reading order of the

bits is unknown, while we require it to be known in advance. Haramaty, Lee and Viola obtained some improved bounds for the related model of *product tests* [**?**].

Finally, we discuss the pseudorandom generator of Impagliazzo, Nisan and Wigderson [INW94], which is also a useful tool in our construction. This work is often cited in the context of derandomizing logarithmic space or read-once oblivious branching programs, but is in fact applicable in other contexts which can be modelled as a network of processors, each flipping its own random bit. The parameters of the generator depend on the "simplicity" of the network graph, and the total communication between the processors.

In the common setting of read-once oblivious branching programs, the network consists of n processors, where the i-th processor reads the random bit x_i, and the network graph is a simple path. This graph is simple in the technical sense required by [INW94], and by the read-once property, the computation of a width w branching program can be simulated by each processor receiving at most one message and sending at most one message, each of length at most $\log(w)$. The message basically encodes the index of a node in the next layer of the branching program.

Considering read-twice branching programs, it is clear that the communication remains bounded if, for example, one considers branching programs which read their input twice in the same order (i.e., $x_1, x_2, \ldots, x_n, x_1, x_2, \ldots, x_n$). However, the situation changes when one considers general read-twice sequences: in this setting, it is always possible to bound the communication by allowing a more complicated network structures, e.g., a clique between the n processors which will allow each pair to communicate directly. However, in this case the network structure is no longer "simple" in the sense required by [INW94].

We insist on the network being a path, in which case it is convenient to model the variable access of the branching program as a Turing machine head, which can move at each step left or right (but cannot "jump" many cells in one step). This approach is also taken by [INW94], which show that their construction works as long as one can bound the number of times the Turing machine head visit each cell (see Theorem 2.5 for a formal statement). Their seed length is proportional to the maximum number of times a cell is visited.

To understand the subtleties, it is useful to consider the following read-twice sequence:

$$x_1, x_2, \ldots, x_n, x_1, x_n, x_2, x_{n-1}, \ldots, x_{n/2}, x_{n/2+1}.$$

If the order on the Turing machine tape is $(x_1, x_2, \ldots, x_n)$ then the Turing machine head reading the sequence would have to visit the $(n/2)$-th cell $\Omega(n)$ times, even though each element only appears twice in the sequence.

In this example, if the designer of the PRG is allowed to look at the sequence of bits — which is the case in our setting — they can "imagine" that the order of the bits on the tape is $x_1, x_n, x_2, x_{n-1}, \ldots, x_{n/2}, x_{n/2+1}$, so that the above sequence can be handled by a Turing machine head which reads each cell 3 times, and then instantiate the generator with this order.

Of course, in a more general case there is no guarantee that we can fix a order on the tape which ensures each cell is visited only a small number of times. In fact, there exists a read-twice sequence $x_1, x_2, \ldots, x_n, x_{\pi(1)}, \ldots, x_{\pi(n)}$ for some permutation π, such that for any fixed order on the tape, there will be a cell which would be visited $\Omega(n)$ times. Thus, one cannot hope to directly apply the INW generator. To overcome this, we follow Anderson et al. [AFS+16] and partition the set of variables X into t disjoint sets $Y_1, \ldots, Y_t$ (Theorem 2.3) using Erdős-Szekeres Theorem. The partition has the property that the given sequence restricted to any part Y_i can be traversed by Turing machine head while visiting each cell a bounded number of times (Lemma 3.2). Thus, it would suffice to plug in independent copies INW generator to each set Y_i (Lemma 2.6).

2 Preliminaries

2.1 Notation

Let $[n] = \{1, 2, \ldots, n\}$. For a partition of $[n] = A_1 \sqcup A_2 \sqcup \cdots \sqcup A_r$ with $|A_i| = m_i$, and distributions D_i on $\{0,1\}^{m_i}$, we denote by

$$D_1^{A_1} \times D_2^{A_2} \times \cdots \times D_r^{A_r}$$

the distribution on $\{0,1\}^n$ obtained by sampling *independently* a vector $\mathbf{b}_i \in \{0,1\}^{m_i}$ from D_i, for $i \in [r]$, and obtaining a string $\mathbf{b} \in \{0,1\}^n$ by plugging $\mathbf{b}_i$ in the coordinates indexed by A_i.

This notation is also used for functions in a natural way: if $G_i : \{0,1\}^{s_i} \to \{0,1\}^{m_i}$ is any function, $G_1^{A_1} \times \cdots \times G_r^{A_r}$ is a function from $\{0,1\}^{s_1+\cdots+s_r}$ to $\{0,1\}^n$ obtained by applying G_1 to the first s_1 input bits and plugging the result in the coordinates of A_1, and so on.

We often consider models which compute boolean functions over a variable sets $X = \{x_1, \ldots, x_n\}$ and various partitions of the sets of variables $Y_1 \sqcup Y_2 \sqcup \cdots \sqcup Y_r$. By considering the indices of the variables in each set, such a partition corresponds naturally to a partition of $[n]$ and thus we use similar notations as above with the Y_i's in the superscript.

2.2 Computational Models

A branching program B on a variable set $X = \{x_1, \ldots, x_n\}$ is a directed acyclic graph, with a unique source vertex, two sink vertices labeled "accept" and "reject", and where every non-sink vertex is labeled by one of the n variables and has exactly two outgoing edges, labeled 0 and 1. B naturally define a boolean function $B : \{0,1\}^n \to \{0,1\}$ by considering the path an input x induces in the graph. In our case, the branching programs will be *layered*, that is, the vertex set can be partitioned into m layers, with every edge going from layer $i-1$ to i.

Such a branching program is said to be *oblivious* if on every layer, all the vertices are labelled by the same variable, namely, the program reads its input in a fixed order which is independent in the value it has read so far. An oblivious branching program is said to be also *read-once* if every variable appears as a label in at most one layer, and more generally *read-k* if every variable appears in at most k layers.[1] Without loss of generality, we may assume each variable is read exactly k times.

2.3 Read-k sequences

Let $X = \{x_1, \ldots, x_n\}$, and $S \in X^m$ be a sequence of m elements from X. S is said to be a read-k sequence over X if every element $x \in X$ appears exactly k times (and in that case $m = nk$). For a set $Y \subseteq X$ we let $S|_Y$ denote the subsequence of S which is obtained by keeping only the elements in Y, and erasing all other elements. For $i \in [k]$, we denote by $S^{(i)}$ the subsequence of S which consists of the i-th occurrences of the elements. In other words, $S^{(i)} \in X^n$ is a permutation of X according to the order of their i-th occurrences. Without loss of generality and by renaming variables, if necessary, we always assume $S^{(1)} = (x_1, \ldots, x_n)$ is the identity permutation. Similarly, for $i \neq j \in [k]$, we use the notation $S^{(i,j)}$ for the subsequence of S which consists of the i-th and j-th occurrences of all elements. We also associate a natural linear order on X by letting $x_1 < x_2 < \cdots < x_n$.

We now cite relevant definitions from [AFS+16]. We begin with the definition of a per-read-monotone sequence.

[1] The modifiers read-once and read-k can be used, and have been used, also in the context of non-oblivious branching programs. In the more general context, one has to distinguish between a syntactic definition and a semantic definition. As the two definitions coincide in the oblivious case, which is the only case we consider, we omit this discussion.

Definition 2.1. *A read-k sequence is said to be* per-read-monotone *if for all $i \in [k]$, $S^{(i)}$ is either monotonically increasing or monotonically decreasing.* ◊

Similarly, a read-k sequence is said to be *per-read-increasing (or per-read-decreasing)* if for all $i \in [k]$, $S^{(i)}$ is monotonically increasing (or decreasing).

Definition 2.2. *A read-2 sequence S over $X = \{x_1, \ldots, x_n\}$ is said to be* 2-regularly-interleaving *if there exists a partition $X = X_1 \sqcup X_2 \sqcup \cdots \sqcup X_t$, such that for every $i \in [t]$, the following two conditions hold:*

1. *The sequence S can be partition into t read-2 sequences $\{S_i\}_{i\in[t]}$ such that $S_i \in X_i^{2|X_i|}$, and $S = (S_1, \ldots, S_t)$ is the concatenation of $S_1, \ldots, S_t$.*
2. *Each S_i as above can be partitioned into two subsequences $S_{i,1}$ and $S_{i,2}$, such that for $c \in \{1, 2\}$, $S_{i,c}$ contains the c-th occurrences of X_i, and S_i equals the concatenation of $S_{i,1}$ and $S_{i,2}$.*

A read-k sequence is k-regularly-interleaving *if for all $i \neq j \in [k]$, the subsequence $S^{(i,j)}$ is 2-regularly-interleaving.* ◊

The following theorem was proved in [AFS+16]. Roughly, it says that for small k, every read-k sequence can be partitioned into a sublinear number of subsequences, each of which is per-read-monotone and k-regularly-interleaving.

Theorem 2.3 ([AFS+16])**.** *Let S be a read-k sequences over $X = \{x_1, \ldots, x_n\}$. Then, X can be partitioned into t disjoint subsets $Y_1 \sqcup Y_2 \sqcup \cdots \sqcup Y_t$, such that*

1. *The subsequence $S_i = S|_{Y_i}$ is per-read-monotone and k-regularly interleaving.*
2. *$t \leq \exp(k^2) \cdot n^{1-1/2^{k-1}}$.*

Further, this partition can be computed, given S, in time $\mathsf{poly}(k, n)$.

The following observation about the structure of per-read-monotone sequences was also made in [AFS+16]. Intuitively, it says that in a per-read-monotone sequence, increasing and decreasing subsequences cannot intersect. That is, a per-read-monotone sequence can be partitioned into subsequences, which are alternately per-read-increasing and per-read-decreasing.

Proposition 2.4. *Let S be a read-k per-read-monotone sequence over $X = \{x_1, \ldots, x_n\}$. Then S is a concatenation of $t \leq k$ subsequences $S = (T_1, \ldots, T_t)$ such that:*

1. *$t \leq k$.*
2. *There exist $1 = i_1 < i_2 < i_3 < \ldots < i_{t-1} < i_t \leq k$ such that for all $i_j \leq c < i_{j+1}$, $S^{(c)}$ is contained in T_j.*
3. *For all odd j (even, respectively), all the subsequences $S^{(c)}$ that appear in T_j are monotonically increasing (decreasing, respectively).*

2.4 The Impagliazzo-Nisan-Wigderson Generator

As mentioned in Section 1.3, we use the INW generator as an important tool in our construction. Here we cite their specific construction which we use, which can handle multiple reads of the input, as long as the "total communication" is bounded.

Theorem 2.5 ([INW94], Theorem 3)**.** *There exists a generator $G^{\mathrm{INW}}_{n,d,\varepsilon} : \{0,1\}^s \to \{0,1\}^n$ which ε-fools every width w oblivious branching program, in which the reading order can be simulated by a Turing machine head which visits every cell at most d times. The seed length s is $O(\log n \cdot (d \log w + \log(n/\varepsilon)))$.*

2.5 Combining Generators

For an oblivious branching program B over the variable set X, a subset $Z \subseteq X$, and a bit vector $\mathbf{b} \in \{0,1\}^{|Z|}$, let $B|_{Z=\mathbf{b}}$ denote the branching program obtained by fixing the variables in Z according to the values given by $\mathbf{b}$. Observe that for all $\mathbf{b}$, $B|_{Z=\mathbf{b}}$ is an oblivious branching program over the variable set $X \setminus Z$.

Lemma 2.6. *Let B be an oblivious branching program of width w over the variable set $X = \{x_1, \ldots, x_n\}$. Let $Y \subseteq X$ be such that $|Y| = m$ and let $Z := X \setminus Y$. Let D be a distribution on $\{0,1\}^m$ that ε-fools $B|_{Z=\mathbf{b}}$, for all $\mathbf{b} \in \{0,1\}^{n-m}$, and let D' be* any *distribution on $\{0,1\}^{n-m}$. Denote $\mu_1 = U_m^Y \times D'^Z$ and $\mu_2 = D^Y \times D'^Z$. Then, it holds that*

$$\left| \Pr_{x \sim \mu_1}[B(x) = 1] - \Pr_{x \sim \mu_2}[B(x) = 1] \right| \leq \varepsilon$$

Proof. From the lemma hypothesis,

$$\left| \Pr_{y \sim U_m}[B|_{Z=\mathbf{b}}(y) = 1] - \Pr_{y \sim D}[B|_{Z=\mathbf{b}}(y) = 1] \right| \leq \varepsilon, \tag{2.7}$$

We observe that the distribution of $B|_{Z=\mathbf{b}}(y)$ where y is chosen according to U_m (or D, respectively) is the same as the marginal distribution of $B(x)$ conditioned on $Z = \mathbf{b}$, where x is chosen from μ_1 (or μ_2, respectively).

Under these notations,

$$\Pr_{x \sim \mu_1}[B(x) = 1] = \sum_{\mathbf{b}} \Pr_{x \sim U_m}[B(x) = 1 | Z = \mathbf{b}] \cdot \Pr[Z = \mathbf{b}] = \sum_{\mathbf{b}} \Pr_{y \sim U_m}[B|_{Z=\mathbf{b}}(y) = 1] \cdot \Pr[Z = \mathbf{b}],$$

and similarly,

$$\Pr_{x \sim \mu_2}[B(x) = 1] = \sum_{\mathbf{b}} \Pr_{y \sim D}[B|_{Z=\mathbf{b}}(y) = 1] \cdot \Pr[Z = \mathbf{b}].$$

Thus, using (2.7) it follows that

$$\begin{aligned}
\left| \Pr_{x \sim \mu_1}[B(x) = 1] - \Pr_{x \sim \mu_2}[B(x) = 1] \right| &= \left| \sum_{\mathbf{b}} \left(\Pr_{y \sim U_m}[B|_{Z=\mathbf{b}}(y) = 1] - \Pr_{y \sim D}[B|_{Z=\mathbf{b}}(y) = 1] \right) \cdot \Pr[Z = \mathbf{b}] \right| \\
&\leq \sum_{\mathbf{b}} \left| \Pr_{y \sim U_m}[B|_{Z=\mathbf{b}}(y) = 1] - \Pr_{y \sim D}[B|_{Z=\mathbf{b}}(y) = 1] \right| \cdot \Pr[Z = \mathbf{b}] \\
&\leq \varepsilon.
\end{aligned}$$

□

3 Pseudorandom Generator for Read-k Oblivious Branching Programs

We begin by showing that the generator G^{INW} from Theorem 2.5 is pseudorandom against read-k oblivious branching programs that read their inputs in a per-read-monotone and k-regularly-interleaving fashion. To that end, we show that such sequences satisfy the properties required by the theorem.

Recall from Proposition 2.4, that any read-k, per-read-monotone sequence is a concatanation of subsequences which are alternately per-read-increasing and per-read-decreasing. The following Lemma from [AFS+16] says that in a per-read-increasing and k-regularly interleaving sequence there are no upward jumps (the case of per-read-decreasing is analogous).

Lemma 3.1. *Let S be a read-k, per-read-increasing, and k-regularly-interleaving sequence over $X = \{x_1, \ldots, x_n\}$. Let $\ell \in [kn]$ be an integer, and suppose that x_i appears in the ℓ-th position in S, and for some $j > i$, x_j appears in position $\ell + 1$. Then $j = i + 1$.*

We use this lemma to show that a per-read-monotone and k-regularly-interleaving sequence satisfies the properties required by Theorem 2.5.

Lemma 3.2. *Let S be a read-k, per-read-monotone, and k-regularly-interleaving sequence over $X = \{x_1, \ldots, x_n\}$. Then, when modelling the variable access of S as a Turing machine head, the head visits every cell at most $2k$ times.*

Proof. We first argue that it is enough to consider the case when we the given sequence is per-read-increasing. From Proposition 2.4, it follows that S is concatenation of t subsequences $S = (T_1, \ldots, T_t)$ such that for each $r \in [t]$, T_r is a read-k_r and k_r-regularly-interleaving sequence over X for some $k_1, \ldots, k_t$ with $k_1 + k_2 + \cdots + k_t = k$. Moreover, for all odd r (even, respectively), T_r is per-read-increasing (decreasing, respectively). In particular, this means that for all odd r, T_r starts with x_1 and ends with x_n. On the other hand for all even r, T_r starts with x_n and ends with x_1. Thus, when moving from T_r to T_{r+1} the Turing machine head does not visit any new cell. We claim that while traversing T_r, the Turing machine head visits every cell at most $2k_r$ times. This would imply that while traversing S, the head visits every cell at most $\sum_{r=1}^{t} 2k_r = 2k$ times.

Now, consider the sequence T_r, which is a read-k_r, per-read-increasing (the decreasing case is similar), and k_r-regularly-interleaving sequence. Obviously, the head needs to visit the i-th cell whenever x_i appears in the sequence. This can happen, by assumption, at most k_r times. However, it also needs to pass through x_i whenever, for $j < i < h$, x_j appears in the sequence and followed by x_h, or x_h is followed by x_j, and thus our goal is bound the number of times these can happen.

We claim the first transition cannot happen at all in S. For suppose x_j appears at position ℓ, and is immediately followed by x_h in position $\ell + 1$. Since $h > i > j$, this contradicts Lemma 3.1.

As for the second type of transition, we claim there can be at most k_r of these. By Lemma 3.1, for any $h' > i > j'$ and for any appearance of $x_{h'}$ after $x_{j'}$ in T_r, the element x_i must appear in between them. Since x_i can appear in T_r at most k_r times, this establishes the claim. □

We are now ready to present the construction of our pseudorandom generator for read-k oblivious branching programs.

Construction 3.3. *Let S be a read-k sequence over $X = \{x_1, \ldots x_n\}$, and let $Y_1, \ldots, Y_t$ be as promised by Theorem 2.3. Let $n_i = |Y_i|$, and s_i be the seed length of $G^{\mathrm{INW}}_{n_i, 2k, \varepsilon/n}(\cdot)$ as given by Theorem 2.5, that is, $s_i = O(\log(n) \cdot (\log(n/\varepsilon) + k \log w))$, and let $s = \sum_{i=1}^{t} s_i$. Define $G^k_\varepsilon : \{0,1\}^s \to \{0,1\}^n$, by*

$$G^k_\varepsilon(\mathbf{y}) = \left(G^{\mathrm{INW}}_{n_1, 2k, \varepsilon/n}(\mathbf{y}_1)\right)^{Y_1} \times \left(G^{\mathrm{INW}}_{n_2, 2k, \varepsilon/n}(\mathbf{y}_2)\right)^{Y_2} \times \cdots \times \left(G^{\mathrm{INW}}_{n_t, 2k, \varepsilon/n}(\mathbf{y}_t)\right)^{Y_t},$$

where $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_t)$ and $\mathbf{y}_i \in \{0,1\}^{s_i}$. ◊

Theorem 3.4. *Let S be a read-k sequence. The generator G^k_ε from Construction 3.3 ε-fools every read-k oblivious branching program which reads the variables in the order prescribed by S. The seed length s is*

$$O\left(t \cdot \log(n) \cdot (\log(n/\varepsilon) + k \log w)\right) = O\left(\exp(k^2) \cdot n^{1 - 1/2^{k-1}} \cdot \log(n) \cdot (\log(n/\varepsilon) + k \log w)\right).$$

Proof. The bound of the seed length follows directly from the construction.

The proof that it indeed ε-fools read-k oblivious branching program is by a standard hybrid argument, using Lemma 2.6.

Let B be any branching program as stated in the theorem, and let $Y_1, \ldots, Y_t$ be the partition as described in Construction 3.3. Recall that $|Y_i| = n_i$. Denote by U_{n_i} the uniform distribution on $\{0,1\}^{n_i}$, and by D_i the distribution of $G^{\mathrm{INW}}_{n_i,2k,\varepsilon/n}(\mathbf{y}_i)$, with $\mathbf{y}_i$ randomly and uniformly picked from $\{0,1\}^{s_i}$.

Now observe that the distribution of a randomly and uniformly seeded G^k_ε is given by $\mu_t := D_1^{Y_1} \times D_2^{Y_2} \times \cdots \times D_t^{Y_t}$, whereas $\mu_0 := U_n = U_{n_1}^{Y_1} \times U_{n_2}^{Y_2} \times \cdots \times U_{n_t}^{Y_{n_t}}$. Similarly, for every $0 \leq j \leq t$, define

$$\mu_j = D_1^{Y_1} \times \cdots \times D_j^{Y_j} \times U_{n_{j+1}}^{Y_{j+1}} \times \cdots \times U_{n_t}^{Y_t}.$$

Consider any $1 \leq j \leq t$. Let $Z_j = X \setminus Y_j$. Recall that for any bit vector $\mathbf{b} \in \{0,1\}^{n-n_j}$, the restriction $B|_{Z_j=\mathbf{b}}$ is a width w oblivious branching program over variables Y_j. From Theorem 2.3, the sequence $S|_{Y_j}$ is a read-k, per-read-monotone, k-regularly interleaving sequence. Thus, by Lemma 3.2 and Theorem 2.5, the distribution $D_j = G^{\mathrm{INW}}_{n_j,2k,\varepsilon/n}(\mathbf{y}_j)$ ε-fools the branching program $B|_{Z_j=\mathbf{b}}$. Now, we apply Lemma 2.6 on B with D as D_j and D' as $D_1^{Y_1} \times \cdots \times D_{j-1}^{Y_{j-1}} \times U_{n_{j+1}}^{Y_{j+1}} \times \cdots \times U_{n_t}^{Y_t}$. We get that for each $1 \leq j \leq t$,

$$\left| \Pr_{\mathbf{x} \sim \mu_{j-1}} [B(\mathbf{x}) = 1] - \Pr_{\mathbf{x} \sim \mu_j} [B(\mathbf{x}) = 1] \right| \leq \varepsilon/n,$$

which, by the triangle inequality, implies that

$$\left| \Pr_{\mathbf{x} \sim \mu_0} [B(\mathbf{x}) = 1] - \Pr_{\mathbf{x} \sim \mu_t} [B(\mathbf{x}) = 1] \right| \leq t \cdot \varepsilon/n \leq \varepsilon,$$

as in the statement of the theorem. □

4 Pseudorandom Generator for Linear Length Oblivious Branching Programs

Our generator for general linear length oblivious branching programs is based on the simple observation that in the generator from Section 3, the seed length remains sublinear even for $k = k(n)$ which is slightly super-constant, whereas if the length of an oblivious branching program is at most cn, the number of variables which appear more than k times is at most $\frac{c}{k}n$, which is sublinear. Thus, these variables can be just sampled uniformly.

Construction 4.1. *Let $S \in X^{cn}$ be a sequence of length cn over $X = \{x_1, \ldots x_n\}$, and set $k = k(n) = (\log\log n)/2$. A variable is said to be* frequent *if it appears more than k times in S. Let F be the set of frequent variables, so we know that $|F| \leq cn/k$. Let s_1 be the seed length of G^k_ε from Construction 3.3, and $s_2 = |F|$.*

Define $G^{\mathrm{lin}} : \{0,1\}^s \to \{0,1\}^n$, by

$$G^{\mathrm{lin}}(\mathbf{y}) = \left(G^k_\varepsilon(\mathbf{y}_1)\right)^{X \setminus F} \times \left(\mathbf{y}_2\right)^F$$

where $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$ and $\mathbf{y}_i \in \{0,1\}^{s_i}$. ◊

Theorem 4.2. *Let $S \in X^{cn}$ be a sequence over $X = \{x_1, \ldots, x_n\}$ of length cn. The generator $G^{\mathrm{lin}} : \{0,1\}^s : \{0,1\}^n$ from Construction 4.1 ε-fools every oblivious branching program B of width w that reads its variables in the order prescribed by S. The seed length s is $O(\frac{n}{\log\log n})$ for $w = \mathsf{poly}(n)$.*

Proof. Since $S|_{X\setminus F}$ is a read-k sequence, from Theorem 3.4, the generator G^k_ε from Construction 3.3 ε-fools the branching program $B|_{F=\mathbf{b}}$ for any $\mathbf{b} \in \{0,1\}^{|F|}$. Thus, from Lemma 2.6, the generator G^{lin} from Construction 4.1 ε-fools B. The bound on the seed length follows from the seed length of Construction 3.3. □

Acknowledgment

We thank Andrej Bogdanov for useful comments on an earlier version of this text.

References

[AFS+16] Matthew Anderson, Michael A. Forbes, Ramprasad Saptharishi, Amir Shpilka, and Ben Lee Volk. Identity Testing and Lower Bounds for Read-k Oblivious Algebraic Branching Programs. In *Proceedings of the 31st Annual Computational Complexity Conference (CCC 2016)*, volume 50, pages 30:1–30:25, 2016.

[AGKS15] Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-Sets for ROABP and Sum of Set-Multilinear Circuits. *SIAM J. Comput.*, 44(3):669–697, 2015. Pre-print available at `arXiv:1406.7535`.

[BPW11] Andrej Bogdanov, Periklis A. Papakonstantinou, and Andrew Wan. Pseudorandomness for Read-Once Formulas. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, pages 240–246, 2011.

[BPW12] Andrej Bogdanov, Periklis A. Papakonstantinou, and Andrew Wan. Pseudorandomness for Linear Length Branching Programs and Stack Machines. In *Proceedings of the 16th International Workshop on Randomization and Computation (RANDOM 2012)*, volume 7408 of *Lecture Notes in Computer Science*, pages 447–458. Springer, 2012.

[BRRY14] Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudayoff. Pseudorandom Generators for Regular Branching Programs. *SIAM J. Comput.*, 43(3):973–986, 2014.

[De11] Anindya De. Pseudorandomness for Permutation and Regular Branching Programs. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity (CCC 2011)*, pages 221–231, 2011.

[DL78] Richard A. DeMillo and Richard J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Information Processing Letters*, 7(4):193–195, 1978.

[FS13] Michael A. Forbes and Amir Shpilka. Quasipolynomial-Time Identity Testing of Noncommutative and Read-Once Oblivious Algebraic Branching Programs. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 243–252, 2013. Full version at `arXiv:1209.2408`.

[FSS14] Michael A. Forbes, Ramprasad Saptharishi, and Amir Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 867–875, 2014.

[GKS17] Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Identity Testing for Constant-Width, and Any-Order, Read-Once Oblivious Arithmetic Branching Programs. *Theory of Computing*, 13(2):1–21, 2017. Preliminary version in the *31st Annual Computational Complexity Conference (CCC 2016)*.

[IMZ12] Russell Impagliazzo, Raghu Meka, and David Zuckerman. Pseudorandomness from Shrinkage. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, pages 111–119, 2012.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC 1994)*, pages 356–364, 1994.

[KNP11] Michal Koucký, Prajakta Nimbhorkar, and Pavel Pudlák. Pseudorandom generators for group products: extended abstract. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011)*, pages 263–272, 2011.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[RSV13] Omer Reingold, Thomas Steinke, and Salil P. Vadhan. Pseudorandomness for Regular Branching Programs via Fourier Analysis. In *Proceedings of the 17th International Workshop on Randomization and Computation (RANDOM 2013)*, pages 655–670, 2013.

[Sch80] Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, 1980.

[Ste12] Thomas Steinke. Pseudorandomness for Permutation Branching Programs Without the Group Theory. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:83, 2012.

[SVW14] Thomas Steinke, Salil P. Vadhan, and Andrew Wan. Pseudorandomness and Fourier Growth Bounds for Width-3 Branching Programs. In *Proceedings of the 18th International Workshop on Randomization and Computation (RANDOM 2014)*, pages 885–899, 2014.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic Circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, March 2010.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.