


Quadratic Time-Space Lower Bounds for Computing Natural Functions with a Random Oracle

Dylan M. McKay

EECS and CSAIL, MIT, 32 Vassar St., Cambridge MA, USA
dmmckay@mit.edu

Richard Ryan Williams¹

EECS and CSAIL, MIT, 32 Vassar St., Cambridge MA, USA
rrw@mit.edu

 <https://orcid.org/0000-0003-2326-2233>

Abstract

We define a model of size- S R -way branching programs with oracles that can make up to S distinct oracle queries over all of their possible inputs, and generalize a lower bound proof strategy of Beame [SICOMP 1991] to apply in the case of random oracles. Through a series of succinct reductions, we prove that the following problems require randomized algorithms where the product of running time and space usage must be $\Omega(n^2/\text{poly}(\log n))$ to obtain correct answers with constant nonzero probability, even for algorithms with constant-time access to a uniform random oracle (i.e., a uniform random hash function):

- Given an unordered list L of n elements from $[n]$ (possibly with repeated elements), output $[n] - L$.
- Counting satisfying assignments to a given 2CNF, and printing any satisfying assignment to a given 3CNF. Note it is a major open problem to prove a time-space product lower bound of $n^{2-o(1)}$ for the *decision version* of SAT, or even for the decision problem Majority-SAT.
- Printing the truth table of a given CNF formula F with k inputs and $n = O(2^k)$ clauses, with values printed in lexicographical order (i.e., $F(0^k), F(0^{k-1}1), \dots, F(1^k)$). Thus we have a $4^k/\text{poly}(k)$ lower bound in this case.
- Evaluating a circuit with n inputs and $O(n)$ outputs.

As our lower bounds are based on R -way branching programs, they hold for any reasonable model of computation (e.g. log-word RAMs and multitape Turing machines).

2012 ACM Subject Classification Theory of computation → Circuit complexity, Theory of computation → Oracles and decision trees

Keywords and phrases branching programs, random oracles, time-space tradeoffs, lower bounds, SAT, counting complexity

Digital Object Identifier 10.4230/LIPIcs.ITCS.2019.56

Funding Supported by NSF CCF-1741615 (CAREER: Common Links in Algorithms and Complexity).

Acknowledgements We thank the anonymous reviewers for helpful comments.

¹ Parts of this work were performed while visiting the Simons Institute for the Theory of Computing and the EECS department at UC Berkeley.



1 Introduction

Infamously little progress has been made towards the resolution of P vs NP. It is still open whether there are linear time algorithms for solving generic NP-hard problems such as CIRCUIT SAT although for certain NP-hard problems, non-linear lower bounds on multitape Turing machines are known [23, 18]. In fact it remains open whether CIRCUIT SAT has a generic algorithm in a random-access machine model running in $O(n^{1.9999})$ time and $O(\log n)$ additional space beyond the input.² Currently the best known time lower bound for solving SAT in $O(\log n)$ space is $n^{2 \cos(\pi/7) - o(1)} \geq n^{1.801}$ ([27, 14], building on [17]).

Other prominent work has used combinatorial methods to prove time-space lower bounds for decision problems in P [10, 3, 12, 11, 25, 4, 22]. For general random-access models of computation, modest super-linear (e.g. $n \log n$ -type) time lower bounds for explicit problems in P are known when the space is restricted to $n^{.99}$ or less (for randomized models as well [11, 22]). Thus for such problems, the time-space product can be lower-bounded to nearly $\Omega(n^2)$ when the space is close to n ; however, when the space is $O(\log n)$, the best time lower bound against RAMs appears to be the aforementioned $n^{1.801}$ bound for SAT.

All the above cited lower bounds are for *decision* problems. For example, the SAT lower bound applies to algorithms which are only required to determine if a given formula is satisfiable or not. There are several well-studied extensions of SAT which are *function* problems, outputting multiple bits:

1. **How difficult is it to *print* a satisfying assignment, when one exists?** For 3CNF formulas, call this problem PRINT-3SAT. Of course, PRINT-3SAT has a polynomial-time algorithm if and only if P = NP, but perhaps it is easier to prove concrete lower bounds for it, compared to the decision version.
2. **How difficult is #2SAT, in which we *count* the number of satisfying assignments to a 2CNF?** Since #2SAT is #P-complete, the problem should intuitively be harder to solve than 3SAT.
3. **How difficult is it to print the truth table of a CNF?** Given a CNF formula F of n variables and up to 2^n size, F can be evaluated on all possible 2^n inputs in $4^n \cdot \text{poly}(n)$ time and $\text{poly}(n)$ space (using a linear-time, log-space algorithm for evaluating a CNF on a given input). Call this problem TTPRINT (for truth-table printing).

Is this quadratic running time optimal? This question is considerable interest for SAT algorithms; for some circuit classes, the only known SAT algorithms beating exhaustive search (e.g., [28]) proceed by reducing SAT to a quick truth table evaluation.

In this paper, we prove that for essentially any generic randomized computation model using $n^{o(1)}$ space but having $O(1)$ -access to a random oracle (i.e., a random string of $2^{n^{o(1)}}$ bits), the above three problems all require nearly quadratic time to compute with nonzero constant probability. We revisit a quadratic time-space lower bound of Beame [8] for the UNIQUE ELEMENTS problem, show that his technique can be used to prove an analogous lower bound for an even “simpler” problem that we call NON-OCCURRING ELEMENTS, extend the lower bound’s reach to include random oracles, and give succinct reductions from NON-OCCURRING ELEMENTS to the above problems, proving hardness for them.

² Note that for multitape Turing machines, such lower bounds are not hard to show [24]. However, these lower bounds rely on the sequential access of Turing machines on tapes; once random access is allowed, these lower bounds break.

Lower Bound for Non-Occurring Elements

We define the NON-OCCURRING ELEMENTS problem to be: *given an unordered list L of n elements from $[n]$, output $[n] - L$ in any order.*³ That is, we simply wish to output the elements that do not occur in the given list.

Observe it is easy to get a space- $O(s(n))$ algorithm on the $\log(n)$ -word RAM for computing NOE with running time $O(n^2/s(n))$. Start by partitioning $[n]$ into $n/s(n)$ blocks of $s(n)$ elements each. Do $n/s(n)$ passes over L , where in the i th pass, check which numbers in the i th block do not occur in L : this can be done in $s(n)$ bits of space by simply keeping a bit vector. Hence we would say NOE has *time-space product* $O(n^2)$ for all $n \leq t(n) \leq n^2$.

Our first main theorem builds on the aforementioned work of Beame to show that this time-space product is optimal, even when programs have access to a random oracle and can err with high probability.

► **Theorem 1.** *For all $p \in (0, 1]$, every random oracle n -way branching program family of size $2^{s(n)}$ and height $t(n)$ computing NOE with success probability p has $t(n) \cdot s(n) \geq \Omega(n^2)$ for all sufficiently large n .*

(Note, we need to formally define what a “random oracle branching program” even means. For now, think of it as a non-uniform version of space-bounded computation with constant-time access to a uniform random $2^{s(n)}$ -bit string. The preliminaries in Section 2 give full detailed definitions.) Using a standard translation of word-RAMs into branching programs, it follows that every probabilistic word-RAM with a random oracle and wordsize $\log(n)$ computing NOE in time $t(n)$ and space $s(n)$ must have $t(n) \cdot s(n) \geq \Omega(n^2)$ in order to have any non-zero constant success probability.

Lower Bound for Sorting and Circuit Evaluation

Informally, we say that a reduction from a problem A to a problem B is *succinct* if any particular output bit of the reduction can be computed in $\text{poly}(\log n)$ time with random access to the input. (Definitions can be found in the Preliminaries.) As a warm-up, in Theorem 24 of the paper, we use a simple succinct reduction to obtain an analogous random oracle lower bound for the problem of sorting n unordered elements from $[n]$, which we call SORT. (A tight lower bound without random oracles was proved by Beame [8].) The sorting lower bound is combined with another simple reduction to obtain an analogous lower bound for evaluating a circuits. Let FCIRCEVAL be the problem: *Given a circuit of size n with at most n inputs (all fixed to 0-1 values) and at most n outputs, determine its output.* The decision version of FCIRCEVAL is well-known to be P-complete.

► **Theorem 2.** *For all finite Σ , there is a $k > 0$ such that the random oracle Σ -way branching program time-space product of FCIRCEVAL is at least $\Omega(n^2/\log^k n)$.*

As in the case of NOE, Theorem 2 implies analogous time-space lower bounds on probabilistic random access machines computing FCIRCEVAL. We note that without the “random oracle” modifier, Theorem 2 is almost certainly folklore; it follows from our simple reduction and the $\tilde{\Omega}(n^2)$ time-space product lower bound on sorting of Borodin and Cook [13].

³ As usual, we define $[n] := \{1, \dots, n\}$.

Lower Bound for Printing SAT Assignments

Any practical algorithm for Satisfiability would need to print satisfying assignments when they exist. We give a succinct reduction from Circuit Evaluation to the printing problem for 3SAT, proving a nearly-quadratic time lower bound in the $n^{o(1)}$ space setting.

► **Theorem 3.** *For all finite Σ , there is a $k > 0$ such that the random oracle Σ -way branching program time-space product of PRINT-3SAT is at least $\Omega(n^2/\log^k n)$.*

Lower Bound for Computing Truth Tables of CNFs

We show that the trivial algorithm for computing the truth table of a large CNF has optimal time-space product, even for randomized algorithms.

► **Theorem 4.** *For all finite Σ , there is a $k > 0$ such that the random oracle Σ -way branching program time-space product of TTPRINT for CNF formulas with n clauses and $\log(n) + \log \log(n)$ many variables is at least $\Omega(n^2/\log^k n)$.*

The proof of Theorem 4 works by giving a succinct reduction from NOE to TTPRINT, constructing a CNF whose truth table encodes (at the bit level) the non-occurring elements of a given list.

Lower Bound for #2SAT

Finally, we give a succinct reduction from the truth table problem TTPRINT for CNF formulas to *counting* SAT assignments to a given 2CNF, implying an analogous lower bound for #2SAT. In particular, we encode the output 2CNF in such a way that the bits of its number of satisfying assignments encode the *value* of the original CNF on various inputs.

► **Theorem 5.** *For all finite Σ , there is some $k > 0$ such that the random oracle Σ -way branching program time-space product of #2SAT is at least $\Omega(n^2/\log^k(n))$.*

Lower Bounds Beyond?

Finally, we note that the random oracle model we consider may not be the most general possible one. We could also consider oracles where the queries are on write-only storage that does not count towards the space bound. Although such oracles are sometimes used in space-bounded complexity ([20]), it is hard for us to see how such queries could help in the *random* oracle setting. In the paper’s conclusion (Section 5) we outline how to coherently define this sort of oracle access in branching programs, and conjecture that our lower bounds also hold in this “extended oracle” model.

1.1 Intuition for the NOE Lower Bound

Our proof of the NOE lower bound (Theorem 1) starts from Beame’s lower bound for UNIQUE ELEMENTS [8]. We generalize his proof, and use our generalization to show that any function problem satisfying two basic properties has a non-trivial branching program lower bound, even when the branching program has access to a huge number of random bits.

► **Theorem 6.** *Let $\{f_n : \Sigma_n^n \rightarrow \Sigma_n^*\}$ be a family of functions, $\{D_n\}$ be a family of distributions, and let $g : \mathbb{N} \rightarrow \mathbb{N}$ satisfy the following properties.*

1. [***f* typically has “long” outputs**]. For all $\varepsilon > 0$, there is an $n_0 \geq 0$ such that for all $n > n_0$, there is a $\delta > 0$ such that

$$\Pr_{x \in D_n} [|f_n(x)| > \delta g(n)] > 1 - \varepsilon.$$

2. [**Short random-oracle branching programs have low probability of printing long substrings of *f***]. Let U_n be the uniform distribution over Σ_n and let $N \leq 2^{s(n)}$ be an integer. There is an $\varepsilon > 0$ such that for all Σ_n -way branching programs P of height at most $n/4$,

$$\Pr_{(x,r) \sim D_n \times U_n^N} [(P(xr) \text{ is a substring of } f_n(x)) \wedge (|P(xr)| \geq m)] < e^{-\varepsilon m}.$$

Then, for all $n > n_0$ and $p \in (0, 1]$, and for every random oracle Σ_n -way branching program of size $2^{s(n)}$ and height $t(n)$ computing f_n with success probability at least p on inputs drawn from D_n , it must be that $t(n) \cdot s(n) \geq \Omega(n g(n))$.

The intuition behind Theorem 6 is that, if a function f on input x requires a long output (Property 1), then some (possibly many) subprograms of an efficient branching program P computing f will need to produce somewhat-long output. But by Property 2, this is “hard” for all short subprograms even with a random oracle: they have low probability of correctly answering a large fraction of f .

Beame’s original lower bound for UNIQUE ELEMENTS [8] follows a similar high-level pattern (as we review in Theorem 21). One of our insights is that the NON-OCCURRING ELEMENTS problem satisfies stronger versions of the properties used in his proof for UNIQUE ELEMENTS; these stronger properties give us extra room to play with the computational model in our lower bound against NON-OCCURRING ELEMENTS. Another insight is that the conditioning on uniform random input in his argument can be modified to accommodate very long auxiliary random inputs. Together, this allows us to extend the lower bounds to models equipped with random oracles.

Why Random Oracles?

Let us give one comment on the model itself. One may wonder why it is even necessary to add random oracles to branching programs (BPs), which are a *non-uniform* model of computation. Can’t we use Adleman’s argument [5] to show that the randomness can be hard-coded in the non-uniform model, similarly to how $\text{BPP} \subset \text{P/poly}$?

Indeed, a random-oracle BP can always be simulated by a deterministic BP; however, the running time of the deterministic BP increases by a multiplicative factor of $\Omega(n)$ (hence, a time lower bound of $\Theta(n^2)$ on a deterministic BP says nothing *a priori* about randomized BPs). Given a randomized BP for a decision problem with constant success probability $1/2 < p < 1$, it is derandomized by taking $\Omega(n)$ copies of the BP (with independent random bits filled in each copy), and computing the majority value of the BP outputs. The $\Omega(n)$ copies are needed because one needs to guarantee correctness over all 2^n possible inputs: this increases the running time by an $\Omega(n)$ multiplicative factor. In our case, the situation is even more complicated because we are concerned with function problems. The upshot is that n^2 lower bounds on random-oracle BPs give strictly more information than a typical randomized model with one-way access to random bits, or a deterministic model.

Another randomized branching program model considered in prior work (e.g., [9]) is to define a randomized time- T space- S branching program to be a *distribution* \mathcal{D} of deterministic time- T and space- S BPs. Such a model is said to compute a function f with error ε if, on

every input x , a randomly drawn BP from \mathcal{D} outputs $f(x)$ with probability at least $1 - \varepsilon$. This model looks even more general than our random oracle model, since no resources are spent accessing randomness (the randomness is “drawn” prior to any computation, then fixed for free). An anonymous reviewer for ITCS observed that our time-space lower bound for NON-OCCURRING ELEMENTS also holds in this BP model. The idea is to apply Yao’s principle [29] which reduces worst-case lower bounds for distributions of BPs to finding a hard distribution of inputs for deterministic BPs, and to verify that our proofs essentially show that the uniform distribution is hard for deterministic BPs.

2 Preliminaries

We assume basic familiarity with computational complexity [5]. Here we recall (and introduce) various computational models and notations needed in the paper.

All of our lower bounds are on the product of time and space for various problems. For clarity, we define the time-space product as follows.

► **Definition 7.** Let f be a function. We say that *the time-space product of f is at least $b(n)$* if for every algorithm running in $t(n)$ time and $s(n)$ space for f , it must be the case that $t(n) \cdot s(n) \geq \Omega(b(n))$.

Random Access Machines With Oracles

For this work, we consider random access machines (RAMs) with access to an oracle and a write-only output tape. More precisely, our RAMs can only append new characters to the end of their output tape.

► **Definition 8.** Let O be a language over a finite alphabet. An **oracle RAM** M^O is a random access machine with an additional *oracle tape*. During a time step, in addition to any normal RAM operations, an oracle RAM can read or write a character to the oracle tape, can move the oracle tape head left or right, or can query the oracle with the contents of the tape to obtain a uniform random bit. Additionally, M^O has a write-only output tape to which it can append a character in any step.

R-Way Branching Programs With Oracles and Output

Our lower bounds for NON-OCCURRING ELEMENTS (Theorem 1) will be against a generalization of branching programs with a (large) alphabet of size R , often called R -way branching programs [13]. We recall the definition here.

► **Definition 9.** Let $R \geq 2$ be an integer. An **R -way branching program** with n inputs x_1, \dots, x_n is a directed acyclic graph with one source node in which every non-sink node has out-degree R . Every non-sink node is labeled with an index $i \in [n]$ corresponding to an input variable, and each edge is labeled with an element of $[R]$ such that no edge (u, v) and (u, w) where $v \neq w$ share a label. Additionally, each vertex v is labeled with an instruction to print a value $p(v)$ (which may be empty). The **height** of an R -way branching program P is the length of the longest path in its graph, and the **size** is the number of vertices. The **computation path of P on input x** is the unique path $\pi = (v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$ such that v_0 is the source node, v_k is a sink node, and for all $i \in \{0, \dots, k-1\}$, if vertex v_i is labeled j , then the label on (v_i, v_{i+1}) is the value of x_j . The **output of P on input x** , denoted as $P(x)$, is the concatenation of the values printed by the vertices along the computation path of P on input x , i.e., $p(v_0)p(v_1) \cdots p(v_k)$.

Let Σ be a finite alphabet. For notational convenience, we call a P a Σ -way branching program if P is a $|\Sigma|$ -way branching program with edges labeled with elements of Σ instead of $[|\Sigma|]$.

► **Definition 10.** A function $f : \Sigma^* \rightarrow \Sigma^*$ has a Σ -way branching programs of height $t(n)$ and size $S(n)$ if for all $n \geq 0$, there is a Σ -way branching program P_n of height $t(n)$ and size $S(n)$ such that for all $x \in \Sigma^n$, $P_n(x) = f(x)$.

We study a natural generalization of R -way branching programs with oracles whose queries count towards the space bound. This is a natural set-up for the random oracle setting, where random oracles model truly random hash functions. Let $O : \Sigma^* \rightarrow \{0, 1\}$ in the following.

► **Definition 11.** An O -oracle R -way branching program with n inputs is an R -way branching program with the following additional properties. Each vertex is labeled with either a *variable index* $i \in [n]$ or with a string $q_j \in \{0, 1\}^*$ (where j is an integer ranging from 1 to the number of vertices in the program); we call the latter Q -vertices (for “Query”). All Q -vertices have two outgoing arcs, labeled with the two possible query answers *yes* or *no* (i.e., 1 or 0). Computation on an O -oracle branching program is defined analogously to usual branching programs, with the following extra rule for Q -vertices. Each time a Q -vertex v is reached during a computation, the outgoing *yes* (i.e., 1) edge of v is taken in the computation path if and only if the label q_j of v satisfies $O(q_j) = 1$.

► **Remark.** Note that in the above definition, the oracle queries could be strings of *arbitrary* length, but in a size- S branching program we are only allowed S possible distinct oracle queries over all possible inputs. So the above definition is more general than the condition that “oracle queries count towards the space bound”. In the random oracle setting, this distinction will make little difference; we might as well think of the query strings as being of length about $\log_2(S)$.

It is natural to augment branching programs with oracles. Barrington and McKenzie [6], motivated by an approach to proving $\text{NC}^1 \neq \text{P}$, defined an oracle branching program model in terms of finite automata, where instead of branching on individual input bits, the program can branch on (in principle) all possible n -bit inputs, corresponding to oracle queries on the input. The usual branching program model is captured in their model by a branching program with an oracle for the predicate $\text{BIT}(x, i)$ which returns the i th bit of the input x . They proved exponential size lower bounds for branching programs with certain weak oracles. Our oracle model is designed to give a natural correspondence between O -oracle R -way branching programs and word RAMs with wordsize $\log(R)$ and oracle access to O .

It is helpful to think of oracle branching programs as simply branching programs *which receive the oracle as part of their input*. The following proposition formalizes this:

► **Proposition 2.1.** Let P^O be an O -oracle Σ -way branching program with n inputs, height T , and size S , and let q_1, \dots, q_m be the set of all possible oracle queries appearing at Q -vertices of P^O . There is a Σ -way branching program P' of height T and size S such that for all $x \in \Sigma^n$, $P^O(x) = P'(xy)$, where $y = O(q_1), \dots, O(q_m)$.

Proof. The branching program P' is simply a relabeling of P in which every Q -vertex with label q_j is instead labeled by the variable index $n + j$. ◀

Random Oracles

We will ultimately study R -way branching programs with random oracles. All of our random oracles will have the form $O : \Sigma^* \rightarrow \Sigma$ for a finite alphabet Σ . Thus we define random oracles as a random elements of the set $(\Sigma^* \rightarrow \Sigma)$.

► **Definition 12.** \mathcal{D}_Σ is the uniform distribution over functions in $(\Sigma^* \rightarrow \Sigma)$. That is, drawing some $f \sim \mathcal{D}_\Sigma$ is equivalent to, for each $x \in \Sigma^*$, choosing an element of Σ uniformly at random as $f(x)$.

We define computation with a random oracle branching program as follows.

► **Definition 13.** A function f has a **random oracle branching program family of height $t(n)$ and size $2^{s(n)}$ with success probability $p \in (0, 1]$** if there is a family of oracle branching programs $\{P_n^O\}$ such that the height of each P_n^O is at most $t(n)$, the size of each P_n^O is at most $2^{s(n)}$, and for all inputs x ,

$$\Pr_{O \sim \mathcal{D}_\Sigma} \left[P_{|x|}^O(x) = f(x) \right] \geq p.$$

Borodin and Cook show that R -way branching programs of height $O(t(n))$ and size $2^{O(s(n))}$ can simulate time- $t(n)$ space- $s(n)$ RAMs of wordsize $\lceil \log_2 R \rceil$ [13]. We observe that their proof relativizes to allow oracle R -way branching programs to simulate oracle RAMs with wordsize $\lceil \log_2 R \rceil$.

► **Lemma 14.** *For every language O and oracle RAM M^O running in time $t(n) \leq 2^{O(s(n))}$ and space $s(n) \geq \log(n)$ with wordsize $\log(\Sigma)$, there is an oracle Σ -way branching program of height $O(t(n))$ and size $2^{O(s(n))}$ such that $M^O(x) = P^O(x)$ for all inputs x .*

Proof. Without loss of generality, assume that at any step, M^O either accesses its input of length n by writing an index $j \in [n]$ to an “access register”, or M^O queries O by writing the character Q to the access register. Let the configuration of machine M contain the description of the random access memory of M , the state of M , and the character currently being output, if any. For integer $n \geq 1$, define a graph $G_n = (V_n, E_n)$ as follows. Let

$$V_n = \{(C, i, j) \mid C \text{ is a space-}s(n) \text{ configuration of } M, i \in \{0, 1, \dots, t(n)\}, j \in [n] \cup \{Q\}\}.$$

Our corresponding branching program, has each vertex (C, i, j) with $j \in [n]$ labeled by j , and those (C, i, j) with $j = Q$ are Q -vertices, which we label with a string q_j representing the content of the oracle tape of M^O in configuration C .

For $j \in [n]$, we put the edge $((C_1, i, j), (C_2, i + 1, j')) \in E_n$ and label it with a string p if and only if M^O in configuration C_1 would transition to configuration C_2 given that in configuration C_1 , j is put in the access register, $x_j = p$, and j' is put in the access register in C_2 . For configurations making an oracle query, put $((C_1, i, Q), (C_2, i + 1, j')) \in E_n$ and label it with $b \in \{0, 1\}$ if and only if M^O in configuration C_1 would transition to configuration C_2 given that in configuration C_1 , Q is written on the access register in C_1 , the oracle query O returns b , and j' is written on access register in C_2 .

Observe that $|V_n| \leq 2^{O(s(n))} \cdot t(n) \cdot n \leq 2^{O(s(n))}$, assuming the size of the tape alphabet and the number of states in the state machine of M^O are constants. Further observe that all paths in G_n have length at most $t(n)$, since every step in the path must be from some (C, i, j, k) to some $(C', i + 1, j', k')$ and $i, i + 1 \in [t(n)]$. Finally, the oracle branching program P_n^O is defined to be the Σ -way branching program which is the induced subgraph of G_n containing all vertices reachable from $v_0 = (C_0, 0, j_0)$ (with the labels prescribed above), where C_0 is the initial configuration of M^O , and j_0 is the index of the input read in the initial configuration. We see that by construction, for all $x \in \Sigma^n$, $P_n^O(x) = M^O(x)$. ◀

Lemma 14 immediately implies that R -way branching program lower bounds provide analogous RAM lower bounds:

► **Proposition 2.2.** *Let $f : \Sigma^* \rightarrow \Sigma^*$. Suppose there is no R -way branching program family of height $O(t(n))$ and size $2^{O(s(n))}$ computing f . Then there is no RAM of wordsize $\log(R)$ running in time $t(n)$ with space $s(n)$ computing f .*

Furthermore, if there is no random oracle R -way branching program family of height $O(t(n))$ and size $2^{O(s(n))}$ computing f with success probability p , then there is no oracle RAM M^O with wordsize $\log(R)$ such that for all $x \in \Sigma^$, $\Pr_{O \sim \mathcal{D}_\Sigma}[M^O(x) = f(x)] \geq p$.*

Succinct Reductions

After our lower bound for NOE has been proved, we can apply very efficient reductions from NOE to extend the lower bound to other problems. For this purpose, we need some notation. In what follows, let f , g , and π be functions from Σ^* to Σ^* .

► **Definition 15.** We say A has a $t(n)$ -time reduction with blowup $b(n)$ to B if there is a many-one reduction f reducing A to B such that $|f(x)| \leq b(|x|)$, and any character in the string $f(x)$ can be computed by an algorithm running in time $t(|x|)$, given the index $i = 1, \dots, |f(x)|$ of the character and random access to x .

Polylog-time reductions with quasi-linear blowup essentially preserve lower bounds for our branching program model, up to polylog factors. As the proof is relatively straightforward and our space is limited, the proof is omitted (but appears in the full version).

► **Lemma 16.** *Let π be an $O(\log^k(n))$ -time reduction with blowup $b(n)$ from f to g , and suppose g has an oracle branching program family $\{P_n^O\}$ of height $t(n)$ and size $2^{s(n)}$ with success probability $p > 0$. Then f has an oracle branching program family $\{Q_n^O\}$ of height $t(b(n)) \log^k(n)$ and size $2^{O(s(b(n)) + \log^k(n))}$ with success probability at least p .*

► **Definition 17.** The **random oracle Σ -way branching program time-space product of f is at least $b(n)$** if for every constant $p \in (0, 1]$ and every random oracle branching program family of height $t(n)$ time and size $2^{s(n)}$ computing f with success probability p , it must be that $t(n) \cdot s(n) \geq \Omega(b(n))$.

We also note (proof omitted in this version) that lower bounds on time-space products are roughly preserved by polylog-time reductions of quasi-linear blowup.

► **Lemma 18.** *Let π be an $O(\log^j(n))$ -time reduction with blowup $O(n \log^j(n))$ from f to g . Suppose there are k and d such that the random oracle Σ -way branching program time-space product of f is at least $\Omega(n^d) / \log^k(n)$. Then there is some k' such that the random oracle Σ -way branching program time-space product of g is at least $\Omega(n^d) / \log^{k'}(n)$.*

For our #2SAT lower bound, we inspect a reduction of Hoffmeister, and note that it is succinct.

► **Theorem 19** ([19]). *There is a $\log(n)$ -time reduction from #3-SAT to #2-SAT with blowup $\tilde{O}(n)$.*

Finally, our lower bound for FCIRCEVAL exploits the fact that there are highly efficient circuits for sorting n items from $[n]$.

► **Theorem 20** ([7, 26, 15]). *There is a k such that SORT has $O(\log^k(n))$ time uniform circuits of size $O(n \log^k(n))$.*

Other Related Work

Besides the work cited earlier, there is an extensive literature on proving $\Omega(\tilde{n}^2)$ time-space product lower bounds for computing functions in generic word-RAM-like models (usually by proving a R -way branching program lower bound). The functions include matrix multiplication and the discrete Fourier transform [30], generalized string matching [1], bit-vector convolution and integer multiplication [2], universal hashing from n bits to $O(n)$ bits [21], and computing various functions over sliding windows [9].

3 Lower Bound for NOE and Sorting

In this section, we abstract out key properties of the UNIQUE ELEMENTS problem that were used in Beame’s proof of an $\Omega(n^2)$ time-space product lower bound for UNIQUE ELEMENTS against R -way branching programs [8]. This abstraction is useful in two ways. First, it allows us to easily prove lower bounds for other problems, by simply verifying that the key properties hold. Second, this level of abstraction helps us identify stronger generalizations of the lower bounds: average-case lower bounds against R -way branching programs and RAMs with random oracles.

3.1 Beame’s Method (Without Random Oracles)

To give intuition for our lower bound theorem for branching programs with random oracles (Theorem 6), we begin with a similar but weaker theorem, which is an abstraction of the technique used by Beame [8].

► **Theorem 21.** *Let $\{f_n : \Sigma_n^n \rightarrow \Sigma_n^*\}$ be a family of functions, $\{D_n\}$ be a family of distributions, and $g : \mathbb{N} \rightarrow \mathbb{N}$ with the following properties.*

1. *[f typically has “long” outputs]. There is an $\varepsilon > 0$ and $\delta > 0$ such that*

$$\Pr_x [|f(x)| > \delta g(n)] > \varepsilon.$$

2. *[Short branching programs have low probability of printing long substrings of f]. There is an $\varepsilon > 0$ such that, for all Σ_n -way branching programs P of height at most $n/4$, and for all $m \geq 1$,*

$$\Pr_{x \sim D} [P(x) \text{ is a substring of } f(x) \wedge |P(x)| \geq m] < e^{-\varepsilon m}.$$

Then for $n > n_0$, every Σ_n -way branching program of size $2^{s(n)}$ and height $t(n)$ for f_n has $s(n)t(n) \geq \Omega(ng(n))$.

Theorem 21 is motivated by the observation that if a function f on input x requires a long output (Property 1), then some (possibly many) subprograms of a branching program P computing f need to output a large fraction of f . But by Property 2, this is “hard” for all short subprograms: they have low probability of correctly answering a large fraction of f .

Proof of Theorem 21. Let $f, D, g(n), \varepsilon, \delta$ be given as above. We prove the lower bound by demonstrating that any space- S branching program P of sufficiently low height T has a nonzero probability of error on an input x drawn from D . To do this, we lower bound the probability of error by

$$\Pr_{x \sim D} [|f(x)| > \delta g(n)] - \Pr_{x \sim D} [(|f(x)| > \delta g(n)) \wedge (P(x) = \text{NOE}(x))].$$

By Property 1 of the hypothesis, the first term is lower bounded by some constant $\varepsilon > 0$. The second term is at most $\Pr_{x \sim D}[|P(x)| > \delta g(n)]$, giving us an error probability of at least

$$\varepsilon - \Pr_{x \sim D}[|P(x)| > \delta g(n)].$$

We now upper-bound $\Pr_{x \sim D}[|P(x)| > \delta g(n)]$.

Without loss of generality, let P be layered, having size 2^S in each layer, and height T . Partition P into $4T/n$ layers of height $n/4$. By the pigeonhole principle, some layer must output at least $ng(n)/4T$ elements of the output. There are at most 2^S such subprograms in that layer, and the probability that P outputs at least $\delta g(n)$ elements correctly is upper bounded by the probability that some layer outputs $m := \delta ng(n)/4T$ elements correctly. As there are at most 2^S such subprograms, by a union bound over property 2 of the hypothesis, the probability that P outputs all elements correctly is upper bounded by

$$2^S e^{-\varepsilon' \delta ng(n)/4T}$$

for some $\varepsilon' > 0$. This implies that the error probability of P on an input x drawn from D is at least $\varepsilon - 2^S e^{-\varepsilon' \delta ng(n)/4T}$.

Finally, we observe that if $ST \leq \alpha \cdot ng(n)$ for sufficiently small $\alpha > 0$, the term $2^S e^{-\varepsilon' \delta ng(n)/4T}$ becomes less than ε . Thus there is some input length n such that the probability of error for P on an input x drawn from D is nonzero. This implies that $ST \geq \Omega(ng(n))$. ◀

Beame's lower bound against UNIQUE ELEMENTS follows from showing that UNIQUE ELEMENTS has the properties required by Theorem 21. In particular, on random inputs UNIQUE ELEMENTS has long outputs with high probability, and it is difficult to guess even a small number of elements in the output of a random input, without seeing most of the input. Instead of reproving the UNIQUE ELEMENTS lower bound, we give a lower bound against the problem NON-OCCURRING ELEMENTS (NOE) defined in the introduction, as it admits a slightly easier analysis.

First let us verify Property 1 of Theorem 21, for the uniform distribution U_n^n and $g(n) = n$.

► **Proposition 3.1** (Property 1 holds for NOE). *For all $\varepsilon > 0$, there is a $\delta > 0$ such that for sufficiently large n , $\Pr_{x \in U_n^n}[|NOE(x)| > \delta n] > 1 - \varepsilon$.*

Proof. The desired bound reduces to a weak version of a well-known Balls-and-Bins bound (see [16, p.75] for a reference). In particular, when selecting m integers from $[n]$ uniformly at random, the number Z of integers in $[n]$ not selected (the non-occurring elements) is tightly concentrated around its mean:

$$\Pr[|Z - E[Z]| > t] \leq 2 \exp(-2t^2/m).$$

For our problem, we have $n = m$ and $E[Z] = n/e$. Let $c > 0$ be a parameter, and let $t = cn$. Therefore we have

$$\Pr[|Z - n/e| > cn] \leq 2 \exp(-2c^2n).$$

Complementing and rearranging variables, the inequality becomes $\Pr[Z \geq n/e - cn] \geq 1 - 2 \exp(-2c^2n)$. Finally, for any $\varepsilon > 0$, we can pick $c \in (0, 1/e)$ such that $\varepsilon > 2 \exp(-2c^2n)$ for sufficiently large n . Letting $\delta \in (0, 1/e - c)$, we have $\Pr[Z > \delta n] > 1 - \varepsilon$. ◀

Note that the bound of Proposition 3.1 is stronger than that required by Theorem 21. This will be useful for the extension to random oracles later (Theorem 6).

Next, we verify that for NON-OCCURRING ELEMENTS, Property 2 of Theorem 21 holds as well. In fact, we prove a stronger statement that allows for extra side randomness in the input (and therefore random oracle branching programs). This randomness can be ignored in the application of Theorem 21.

► **Proposition 3.2** (Property 2 holds for NOE). *For all integers $n, k, N \geq 0$ and all branching programs P of height at most $n/4$ computing a function with m outputs,*

$$\Pr_{x \sim U_n^N, r \sim U_k^N} [P(xr) \text{ outputs } m \text{ non-occurring elements of } x] < e^{-3m/4}.$$

Proof. The desired probability equals

$$\sum_{\text{paths } \pi \text{ in } P} \Pr_{x,r} [P(xr) \text{ follows } \pi] \cdot \Pr_{x,r} [P(xr) \text{ has } m \text{ NOEs of } x \mid P(xr) \text{ follows } \pi].$$

We show that for all such π ,

$$\Pr_{x,r} [P(xr) \text{ has } m \text{ non-occurring elements of } x \mid P(xr) \text{ follows } \pi] < e^{-3m/4}.$$

From this, it will follow that our desired probability is less than $e^{-3m/4}$.

For a path π , let q be the number of distinct variable queries to x and let q' be the number of distinct queries made to r . Notice that $q + q' \leq n/4$. To bound the probability that $P(xr)$ has at least m non-occurring elements of x , given that $P(xr)$ follows π , we simply count the relevant numerator and denominator.

The total number of xr that follow π is $n^{n-q} \cdot k^{N-q'}$: there are $n - q$ unqueried inputs of x , and $N - q'$ unqueried inputs of r . The total number of xr that follow π , and for which the m outputs of π are non-occurring elements of x , is at most $(n - m)^{n-q} \cdot k^{N-q'}$: since the m outputs are supposed to be non-occurring in x , none of the remaining $n - q$ unqueried variables can take on any of the m outputs. By simple manipulation, we have

$$\frac{(n - m)^{n-q} k^{N-q'}}{n^{n-q} k^{N-q'}} = (1 - m/n)^{n-q} \leq (1 - m/n)^{n-n/4} \leq (1 - m/n)^{3n/4} \leq e^{-3m/4}.$$

This completes the proof. ◀

3.2 Lower Bounds With Random Oracles

We are now ready to present our main lower bound theorem against branching programs with random oracles.

► **Reminder of Theorem 6.** *Let $\{f_n : \Sigma_n^n \rightarrow \Sigma_n^*\}$ be a family of functions, $\{D_n\}$ be a family of distributions, and let $g : \mathbb{N} \rightarrow \mathbb{N}$ satisfy the following properties.*

1. [***f* typically has “long” outputs**]. *For all $\varepsilon > 0$, there is an $n_0 \geq 0$ such that for all $n > n_0$, there is a $\delta > 0$ such that*

$$\Pr_{x \in D_n} [|f_n(x)| > \delta g(n)] > 1 - \varepsilon.$$

2. [***Short random-oracle branching programs have low probability of printing long substrings of f***]. *Let U_n be the uniform distribution over Σ_n and let $N \leq 2^{s(n)}$ be an integer. There is an $\varepsilon > 0$ such that for all Σ_n -way branching programs P of height at most $n/4$,*

$$\Pr_{(x,r) \sim D_n \times U_n^N} [(P(xr) \text{ is a substring of } f_n(x)) \wedge (|P(xr)| \geq m)] < e^{-\varepsilon m}.$$

Then, for all $n > n_0$ and $p \in (0, 1]$, and for every random oracle Σ_n -way branching program of size $2^{s(n)}$ and height $t(n)$ computing f_n with success probability at least p on inputs drawn from D_n , it must be that $t(n) \cdot s(n) \geq \Omega(ng(n))$.

Theorem 6 prescribes a general scheme for proving time-space product lower bounds against random oracle R -way branching programs, and thus against word RAMs with random oracles as well.

Proof of Theorem 6. The proof is similar to Theorem 21; we focus on highlighting what is different. First, we note that by Proposition 2.1, it is sufficient to demonstrate that for all $n > n_0$ and $p \in (0, 1]$, and for every Σ_n -way branching program of size $2^{s(n)}$ and height $t(n)$ where $\Pr_{x \sim D_n, r \sim U_n^N}[P(xr) = f_n(x)] > p$, it must be that $t(n) \cdot s(n) \geq \Omega(ng(n))$.

As in Theorem 21, we establish the lower bound by demonstrating that every $n/4$ -height branching program P has a large probability of error on an input $x \sim D_n$. In what follows, assume P successfully computes $f_n(x)$ on inputs (x, r) drawn from $D_n \times U_n^N$ with probability at least p . By property 1 of the hypothesis, letting $\varepsilon := p/2$, there is a δ such that $\Pr_{x \in D, r \sim U_n^N}[|f(x)| > \delta g(n)] > 1 - p/2$.

Analogously as in the proof of Theorem 21, we lower bound the probability of error of P on D_n by

$$\Pr_{x \sim D}[|f(x)| > \delta g(n)] - \Pr_{x \sim D, r \sim U_n^N}[|f(x)| > \delta g(n) \wedge P(xr) = f(x)];$$

note that this probability is at least $(1 - p/2) - \Pr_{x \sim D, r \sim U_n^N}[|P(xr)| > \delta g(n)]$.

Applying a union bound over all $2^{s(n)}$ subprograms of P as before, but substituting property 2 from the hypothesis, we determine by an analogous argument as Theorem 21 that $\Pr_{x \sim D, r \sim U_n^N}[|P(xr)| > \delta g(n)]$ is at most $2^{s(n)} e^{-\varepsilon \delta n g(n)/4t(n)}$. Therefore the error probability of P on D_n is at least

$$1 - p/2 - 2^S e^{-\frac{\varepsilon \delta n g(n)}{4t(n)}}.$$

Finally, if we assume $s(n)t(n) \leq \alpha n \cdot g(n)$ for all $\alpha > 0$, we can tune $2^{s(n)} e^{-\varepsilon \delta n g(n)/(4t(n))}$ to be arbitrarily small: it is at most $2^{s(n)} e^{-\varepsilon \delta s(n)/(4\alpha)}$. Thus we can make the error probability for P on an input x drawn from D_n to be at least $1 - 2p/3$, implying that the success probability is at most $2p/3 < p$. This is a contradiction, so there is some $\alpha > 0$ (depending on δ, ε , and p) such that $s(n)t(n) \leq \alpha n \cdot g(n)$. ◀

► **Remark.** If a function does not satisfy Property 2 of Theorem 6 but satisfies a slightly weaker property, namely that there is an $\varepsilon > 0$ such that for all $|\Sigma_n|$ -way branching programs P of height at most $n/4$,

$$\Pr_{x \sim D_n}[P(x) \text{ is a substring of } f(x) \wedge P(x) \geq m] < e^{-\varepsilon m},$$

we can still obtain an average case lower bound against f for inputs drawn from D_n , but not necessarily a lower bound against random-oracle branching programs. We omit an exposition of this result, because we have not yet found applications of it.

We conclude this subsection with the lower bound for NOE with a random oracle.

► **Reminder of Theorem 1.** For every $p \in (0, 1]$, every random oracle n -way branching program family of size $2^{s(n)}$ and height $t(n)$ computing NOE with success probability p must have $t(n) \cdot s(n) \geq \Omega(n^2)$ for all sufficiently large n .

Proof. Applying Proposition 3.1 and 3.2 and set $D_n := U_n^n$ and $\varepsilon := 3/4$. Then both properties 1 and 2 of Theorem 6 hold for NON-OCCURRING ELEMENTS. ◀

3.3 Sort and Random Oracles

By reducing from NON-OCCURRING ELEMENTS to SORT, it follows that random oracle n -way branching programs still require a time-space product of $\Omega(n^2)$ to sort a list $L \in [n]^n$. (The proof is omitted due to space restrictions.)

► **Theorem 22.** *For any constant $c \in (0, 1]$, let $\{P_n^O\}$ be an oracle n -way branching program family of size $2^{s(n)}$ and height $t(n)$ such that for all $n, x \in [n]^n$, $\Pr_{O \sim \mathcal{D}_{[n]}}[P_n^O(x) = \text{SORT}(x)] \geq c$. Then, $ST = \Omega(n^2)$.*

To prove our lower bounds for other problems in the following section, we require a somewhat stronger result: a nearly-quadratic time-space product lower bound for computing the non-occurring elements of a list of n items from $[n]$, as well as sorting n items from $[n]$ for lists encoded over a *finite* alphabet Σ . By a reduction, we can prove this using the general lower bounds of Theorem 1 and Theorem 22.

► **Lemma 23.** *Let $\{f_n : \Sigma_n^n \rightarrow \Sigma_n^*\}$ be a family of functions, and let $f_\Sigma : \Sigma^* \rightarrow \Sigma^*$ be such that $f(\langle x \rangle_\Sigma) = \langle f_{|x|}(x) \rangle_\Sigma$, where $\langle s \rangle_\Sigma$ is s encoded by a string over Σ . Suppose $\{f_n\}$ requires a random oracle Σ_n -way branching program time-space product of $\Omega(n^d)$. Then there is some $k > 0$ such that f_Σ requires a random oracle Σ -way branching program time-space product of $\Omega(n^d / \log^k(n))$.*

Proof. By contradiction. Suppose for all $k > 0$, f_Σ has random oracle Σ -way branching program family $\{P_n^O\}$ of height $t(n)$ and size $2^{s(n)}$ with success probability $p > 0$ where $t(n)s(n) \leq O(n^d / \log^k(n))$. We show that $\{f_n\}$ has a random oracle n -way branching program family $\{Q_n^O\}$ of height $t'(n)$ and size $2^{s'(n)}$ with success probability $p > 0$ where $t'(n)s'(n) \leq O(n^d / \log^k(n))$. To do this, we simply simulate P_n^O with Q_n^O . We first consider the input $l \in [n]^n$ a length $n \lceil \log_{|\Sigma|}(n) \rceil$ string where each number is represented by some string of characters from Σ . We then modify $P_{n \lceil \log_{|\Sigma|}(n) \rceil}^O$ as follows. For each vertex v which queries an input i , instead query input $\lfloor i / \log_{|\Sigma|}(n) \rfloor$. Then, for all edges (u, v) with label $\alpha \in \Sigma$ in $P_{n \lceil \log_{|\Sigma|}(n) \rceil}^O$, add an edge from u to v with label $\beta \in [n]$ for all β whose representation in base Σ contains α at position $i \bmod \lceil \log_{|\Sigma|}(n) \rceil$. Finally, we need only modify the output behavior of $P_{n \lceil \log_{|\Sigma|}(n) \rceil}^O$. It prints the representation of characters $\sigma \in \Sigma_n$ using characters from Σ . We need only $O(\log(n))$ extra bits of storage to remember the last $\lceil \log_{|\Sigma|}(n) \rceil$ characters $P_{n \lceil \log_{|\Sigma|}(n) \rceil}^O$ would have printed, and upon reaching enough characters, we simply print the corresponding element of Σ_n and remember that we have started a new character. As in the proof of Theorem 22, we can do this by creating $O(n)$ copies of our modified branching program and transitioning accordingly and letting this be Q_n^O .

Finally, we see that by construction, $\{Q_n\}$ computes $\{f_n\}$ with success probability p . Further, we see that the height of Q_n is $O(t(n \log(n)))$ and the size is $2^{O(s(n) + \log(n))}$. By our assumption, we see that $t(n \log(n))(s(n) + \log(n)) \geq \Omega(n^d)$ and $t(n \log(n))(s(n) + \log(n)) \leq O((n \log(n))^d / \log^{d+1}(n))$. This is a contradiction. ◀

From this we can conclude lower bounds for NON-OCCURRING ELEMENTS and SORT for strings over finite alphabets.

► **Lemma 24.** *Let $\text{NON-OCCURRING ELEMENTS}_\Sigma : \Sigma^* \rightarrow \Sigma^*$ be a function which maps the encoding in Σ of a list $\langle L \rangle$ (where $L \in [n]^n$ for some n) to $\langle \text{NON-OCCURRING ELEMENTS}(L) \rangle$, and let $\text{SORT}_\Sigma : \Sigma^* \rightarrow \Sigma^*$ be the function which maps the encoding of a list $\langle L \rangle$ where $\exists n \in \mathbb{N}(L \in [n]^n)$ to $\langle \text{SORT}(L) \rangle$. For all Σ , there is some $k > 0$ such that, the random oracle Σ -way branching program time-space product for both $\text{NON-OCCURRING ELEMENTS}_\Sigma$ and SORT_Σ are both at least $\Omega(n^2 / \log^k n)$.*

Proof. This follows directly from Theorems 1 and 22 and Lemma 23. ◀

► **Remark.** It will be important later that the branching program lower bounds we have established hold even if we allow the branching program to make small perturbations on the output. For example, all above lower bound proofs still go through, if we permit branching programs to output 0 at any node that does not already output some other number. Thus our lower bounds hold for any BP computing NON-OCCURRING ELEMENTS or NON-OCCURRING ELEMENTS $_{\Sigma}$ which prints the binary representation of a list L , such that L contains all non-occurring elements of x , along with any number of elements which are 0.

4 Reductions

We now give a series of reductions from NOE and SORT, showing nearly-quadratic time-space product lower bounds for several natural circuit-analysis problems. Each reduction is very efficient, requiring only $\text{poly}(\log n)$ time to look up any bit of the output of the reduction, and only creating problem instances of size $\tilde{O}(n)$ from inputs of size n .

As a warm-up, we observe a very simple reduction from SORT $_{\{0,1\}}$ (sorting $n \log(n)$ -bit strings) to FCIRCEVAL. Recall in FCIRCEVAL we are given a circuit of size n with at most n inputs (all fixed to 0-1 values) and at most n outputs, and want to determine its output.

► **Reminder of Theorem 2.** *For all finite Σ , there is some $k > 0$ such that the random oracle Σ -way branching program time-space product of FCIRCEVAL is at least $\Omega(n^2 / \log^k n)$.*

Proof. The idea is simple: on an unsorted input, we can succinctly produce a circuit for sorting and ask FCIRCEVAL for its output. We observe there is an $O(\log n)$ -time reduction with blowup $O(\log^k n)$ from SORT to FCIRCEVAL (for some k). Given an input list x to be sorted, we produce a circuit C for SORT with x hard-coded as the input. Any bit of the circuit description can be computed in $O(\log n)$ time, as SORT has $O(\log n)$ -time uniform circuits of size $\tilde{O}(n)$, and any bit of x can trivially be produced in $O(\log n)$ time. By Lemmas 24 and 18, we conclude a time-space lower bound of the form $\Omega(n^2 / \log^k n)$ for FCIRCEVAL. ◀

A straightforward corollary of Theorem 2 is a similar lower bound against a seemingly weaker version of 3SAT.

► **Definition 25.** Let PROMISE-PRINTING-UNIQUE-SAT be the problem: given a circuit C promised to have exactly one satisfying assignment, print the satisfying assignment of C .

► **Lemma 26.** *For all finite Σ , there is a $k > 0$ such that the random oracle Σ -way branching program time-space product of PROMISE-PRINTING-UNIQUE-SAT is at least $\Omega(n^2 / \log^k n)$.*

Proof. We give a reduction from FCIRCEVAL to PROMISE-PRINTING-UNIQUE-SAT. Given an instance C of FCIRCEVAL with only constant inputs and m output bits y_1, \dots, y_m , consider the circuit $C'(x)$ with m free input bits x_1, \dots, x_m and one output bit. C' can be made such that $C'(x_1, \dots, x_m) = 1$ if and only if $x_i = y_i$ by using only as many gates as an needed to construct C and $O(m)$ additional gates to check equality of the output bits of C with the free input bits and then to take the conjunction of these equalities. In total, we see then that $|C'| = O(|C|)$. Finally we see this gives us an $O(\log(n))$ -time reduction with blowup $O(n)$ from FCIRCEVAL to PROMISE-PRINTING-UNIQUE-SAT, since $\text{FCIRCEVAL}(C) = \text{PROMISE-PRINTING-UNIQUE-SAT}(C')$, and each bit of C' can be computed by simply looking up bits of C directly or deciding if they are part of the equality check or conjunction of the equality checks added to C . By Lemma 18, we can conclude that PROMISE-PRINTING-UNIQUE-SAT has a time-space lower bound of $\Omega(n^2 / \log^k n)$. ◀

To establish a connection with printing satisfying assignments for 3CNFs, we consider the problem PROMISE-PRINTING-UNIQUE-3SAT, which is just PROMISE-PRINTING-UNIQUE-SAT restricted to 3CNF formulas. By another straightforward reduction, it follows that PROMISE-PRINTING-UNIQUE-3SAT cannot be computed more efficiently than FCIRCEVAL.

► **Lemma 27.** *For all finite Σ , there is a $k > 0$ such that the random oracle Σ -way branching program time-space product of PROMISE-PRINTING-UNIQUE-3SAT is at least $\Omega(n^2 / \log^k n)$.*

Proof. We present a reduction from the PROMISE-PRINTING-UNIQUE-SAT function to PROMISE-PRINTING-UNIQUE-3SAT. Consider the standard reduction from CIRCUIT SAT to 3SAT in which a circuit C with n variables x_1, \dots, x_n and m gates is mapped to a CNF ϕ with $O(m)$ clauses, variables x_1, \dots, x_n and m extra variables. Assume without loss of generality (blowing up only an additional constant factor in size otherwise) that C is comprised only of NAND gates. The typical reduction from CIRCUIT SAT to 3SAT uses an additional variable y_1, \dots, y_m for each gate g_1, \dots, g_m , and for each gate $g_i = \neg(g_j \wedge g_k)$, $g_i = \neg(g_j \wedge x_k)$, or $g_i = \neg(x_j \wedge x_k)$, we add to ϕ the constraints $(y_i = \neg(y_j \wedge y_k))$, $(y_i = \neg(y_j \wedge x_k))$, or $(y_i = \neg(x_j \wedge x_k))$ respectively, where each requires only $O(1)$ clauses of width 3.

Notice that printing the first n bits of a satisfying assignment to ϕ is sufficient for computing a satisfying assignment to C , and observe that this reduction can be done in $O(\log^k(n))$ time with blowup $O(\log^k(n))$ for some k . Analogously to Lemma 18, we can conclude that PROMISE-PRINTING-UNIQUE-3SAT requires a random oracle Σ -way branching program time-space product of $\Omega(n^2 / \log^{k'}(n))$ for some $k' > 0$. ◀

As a corollary, we can immediately conclude that the harder problem of PRINT-3SAT also has a time-space lower bound of $\Omega(n^2 / \log^k n)$ for some k .

► **Reminder of Theorem 3.** *For all finite Σ , there is a $k > 0$ such that the random oracle Σ -way branching program time-space product of PRINT-3SAT is at least $\Omega(n^2 / \log^k n)$.*

Proof. Follows from the trivial reduction from PROMISE-PRINTING-UNIQUE-3SAT to PRINT-3SAT and Lemma 18. ◀

Next, we show by a reduction directly from NON-OCCURRING ELEMENTS_{0,1} that printing the truth tables of CNF formulas with a small number of variables admits a time-space lower bound similar to those above.

► **Reminder of Theorem 4.** *For all finite Σ , there is some $k > 0$ such that the random oracle Σ -way branching program time-space product of TTPRINT for CNF formulas with n clauses and $\log(n) + \log \log(n)$ many variables is at least $\Omega(n^2 / \log^k n)$.*

Proof. We consider that the output of a machine computing NOE_{0,1} can be a list L which contains the non-occurring elements of its input as well as any number of elements which are 0, as in Remark 3.3. We give a $\text{poly}(\log(n))$ -time reduction with blowup $\tilde{O}(n)$ from NON-OCCURRING ELEMENTS_{0,1} to TTPRINT. That is, we will show that given a list L of n elements from $[n]$, we can produce a CNF formula whose truth table is a representation of a list L' of n strings each of $\log(n)$ bits, where the i -th string in L' equals i if $i \in [n] - L$, otherwise the i -th string equals all-zeroes. Then, the lower bound for NON-OCCURRING ELEMENTS_{0,1} will carry over to TTPRINT.

Given a list $L \in \{1, \dots, n\}^n$, we show how to efficiently construct a CNF formula F with $\log(n) + \log \log(n)$ variables and $O(n)$ clauses such that the truth table of F is the binary representation of the list of non-occurring elements of l separated by strings of 0's.

Denote the first $\log(n)$ variables of F by $x = x_1 \cdots x_{\log(n)}$. Letting $b \in \{0, 1\}^{\log(n)}$, we make a clause $C_b(x)$ expressing that $x \neq b$. In detail, suppose b is represented by the bit string $b_1, \dots, b_{\log(n)}$. Then

$$C_b(x) := ((x_1 \oplus b_1) \vee \dots \vee (x_{\log(n)} \oplus b_{\log(n)})).$$

Note for all i , we can think of C_b as containing the literal \bar{x}_i if $b_i = 1$, otherwise it contains the literal x_i .

Given a list $L = \ell_1, \dots, \ell_n$ of elements of $\{0, 1\}^{\log(n)}$, we first construct a CNF formula $F'(x)$ which says that x is a non-occurring element of ℓ :

$$F'(x) := C_{\ell_1}(x) \wedge \dots \wedge C_{\ell_n}(x).$$

Suppose we can construct a CNF formula $D(x, i)$ which is true exactly when the i -th bit of x is 1, and define our output formula to be

$$F(x, i) := F'(x) \wedge D(x, i)$$

where $i = i_1 \cdots i_{\log \log(n)}$. This F would have $\log(n) + \log \log(n)$ variables, and its truth table in lexicographical order could be viewed as a list of n different $\log(n)$ -bit strings, each of which are either a non-occurring element of ℓ , or the all-zeroes string. (This would complete our reduction.)

We show how to construct such a $D(x, i)$ with $|x|$ clauses. For each variable of x , and $j = 1, \dots, |x|$, we construct a clause E_j which is true if and only if either $i \neq j$ or $x_j = 1$. That is, we define $E_j = ((i_1 \oplus j_1) \vee \dots \vee (i_{\log(|x|)} \oplus j_{\log(|x|)}) \vee x_j)$. Then we can define

$$D(x, i) := \bigwedge_{j=1}^{|x|} E_j.$$

The final formula $F(x_1, \dots, x_{\log(n)}, i_1, \dots, i_{\log \log(n)})$ is a width- $(\log(n)+1)$ CNF of $O(n)$ clauses, and all of the clauses of F can be efficiently constructed in a local way. ◀

Using the lower bound on truth table printing, we can now show the lower bound for counting SAT assignments (Theorem 5). Again we do this by providing a $\text{poly}(\log(n))$ -time reduction with blowup $\tilde{O}(n)$ from one problem to another. We show how to modify a circuit C to efficiently produce another circuit C' , such that the bit representation of the number of SAT assignments of C equals the truth table of C .

► **Lemma 28.** *Let C be a circuit of size s with n input variables x_1, \dots, x_n . Then, there exists a circuit C' of size $O(s + 2^n)$ with input variables $x_1, \dots, x_n, y_1, \dots, y_{2^n}$ such that $\#(C') = TT(C)$. Moreover, there is an algorithm that given such a circuit and index i can produce the i -th bit of C' in time $\tilde{O}(\log(s + n + i))$.*

Proof. Let C be a circuit of size s with inputs x_1, \dots, x_n . First, we construct a new circuit $D(x_1, \dots, x_n, y_1, \dots, y_{2^n})$ which outputs 1 if and only if $y \leq 2^{\text{bin}(x)}$, where $\text{bin}(x)$ is the number in $\{1, \dots, 2^n\}$ represented by the binary string $x_1 \cdots x_n$. Note that D can be constructed with $O(2^n)$ gates (by standard arguments), and any particular gate of D can be constructed in $\text{poly}(\log(n))$ time. Finally, we define the circuit C' on variables $x_1, \dots, x_n, y_1, \dots, y_{2^n}$ by

$$C'(x, y) := C(x) \wedge D(x, y).$$

Observe that the number of assignments satisfying C' is $\sum_{x \in \{0, 1\}^n} C(x) \cdot 2^{\text{bin}(x)}$, as for each $x \in \{0, 1\}^n$ there are $2^{\text{bin}(x)}$ assignments to $y \in \{0, 1\}^{2^n}$ such that $C'(x, y) = 1$. This is exactly the number represented in binary by $TT(C)$. ◀

From Lemma 28 and Theorem 4, we can conclude Lemma 29.

► **Lemma 29.** *For all finite Σ , there is some $k > 0$ such that the random oracle Σ -way branching program time-space product of $\#\text{CIRCUIT SAT}$ is at least $\Omega(n^2/\log^k n)$.*

From Lemma 29, we can conclude Lemma 30.

► **Lemma 30.** *For all finite Σ , there is some $k > 0$ such that the random oracle Σ -way branching program time-space product of $\#\text{3SAT}$ is at least $\Omega(n^2/\log^k n)$.*

Proof. The standard reduction from $\#\text{CIRCUIT SAT}$ to $\#\text{3SAT}$ can be implemented as an $O(\log(n))$ -time reduction with blowup $\tilde{O}(n)$. ◀

From Lemmas 30 and 19, we can conclude Theorem 5.

► **Reminder of Theorem 5.** *For all finite Σ , there is some $k > 0$ such that the random oracle Σ -way branching program time-space product of $\#\text{2SAT}$ is at least $\Omega(n^2/\log^k n)$.*

Proof. Lemma 19 gives an $O(\log^k(n))$ time reduction with blowup $O(n\log^k(n))$ from $\#\text{3SAT}$ to $\#\text{2SAT}$. Hence, by Lemmas 30 and 18, we can conclude that the random oracle Σ -way branching program time-space product of $\#\text{2SAT}$ is at least $\Omega(n^2/\log^k(n))$. ◀

5 Conclusion

We extended a lower bound framework for branching programs, lifting it to random oracles. We demonstrated its utility for lower bound results by giving several unorthodox reductions which show sharp relationships between counting satisfying assignments, printing truth tables of CNFs, printing satisfying assignments, and evaluating circuits on given inputs. It would be interesting to find more applications of the encoding techniques used in our reductions. Perhaps they can be used to show lower bounds for other models, or better algorithms.

Another interesting direction would be to explore other lower bound methods, such as those for *decision* problems, and determine to what extent they can be “lifted” to lower bounds with random oracles. To give one tantalizing example, although it is known that deciding SAT requires $n^{1.8}$ time on deterministic $O(\log n)$ -space machines, and this paper shows that *printing* SAT assignments requires $n^{2-o(1)}$ time on *randomized* $O(\log n)$ -space machines with constant error probability, it is still open whether *deciding* SAT is in $O(n)$ time and $O(\log n)$ space on randomized machines with two-sided error(!). Perhaps it is easier to find bridges between function problems and decision problems when we consider efficient programs for NP problems (rather than decision problems in P).

Finally, our oracle model for branching programs is not the most general that one could imagine (although for the *random* oracle case, we believe it does not matter). One can define a sensible “extended oracle” model, where oracle queries can be as long as the height of the branching program. (This model is not very practical in a truly space-bounded setting, e.g., when we view a random oracle as a random hash function, but it would be interesting for lower bounds.)

At a high level, here is how such an “extended oracle” model can be defined. In each step, we allow the BP to output an “oracle character” σ from the input alphabet of the oracle (along with its usual outputs). Instead of labeling the query vertices of the BP with specific query strings (as in Definition 11), we instead label them with a special symbol Q . The Q -vertices still have two outgoing arcs for their yes/no query answers. However, each time a Q -vertex v is reached during a computation, the outgoing *yes* edge of v is now taken in the computation path if and only if the string of oracle characters $y = \sigma_1 \cdots \sigma_t$ output since the

previous Q -vertex (or source node, if there is no previous Q -vertex) satisfies $Q(y) = 1$. One can think of this as allowing the BP “append-only” access to an arbitrarily long oracle tape, for which it can ask queries, and for which the oracle tape is reset to blank after each query (as in the oracle model of Ladner and Lynch [20]).

With the above model, we can ask queries whose length is only bounded by the height of the branching program (rather than the logarithm of its size). We strongly believe that our lower bounds also hold for random oracles in this more powerful model. The main conceptual bottleneck is that, unlike normal branching programs, we cannot easily partition these extended-oracle branching programs into short independent branching programs: the oracle queries have “memory” that can stretch all the way back to the source node. It seems likely that lower bounds based on Yao’s principle [29] can be extended to this model, but we have not yet confirmed this.

References

- 1 Karl Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16(6):1039–1051, 1987.
- 2 Karl Abrahamson. Time-space tradeoffs for algebraic problems on general sequential machines. *Journal of Computer and System Sciences*, 43(2):269–289, 1991.
- 3 Miklós Ajtai. Determinism versus nondeterminism for linear time RAMs with memory restrictions. *Journal of Computer and System Sciences*, 65(1):2–37, 2002.
- 4 Miklós Ajtai. A Non-linear Time Lower Bound for Boolean Branching Programs. *Theory of Computing*, 1(8):149–176, 2005. doi:10.4086/toc.2005.v001a008.
- 5 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
- 6 David A. Mix Barrington and Pierre McKenzie. Oracle branching programs and Logspace versus P. *Information and Computation*, 95(1):96–115, 1991.
- 7 K. E. Batchner. Sorting Networks and Their Applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS ’68 (Spring), pages 307–314, 1968.
- 8 Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM Journal on Computing*, 20(2):270–277, 1991.
- 9 Paul Beame, Raphaël Clifford, and Widad Machmouchi. Element distinctness, frequency moments, and sliding windows. In *FOCS*, pages 290–299. IEEE, 2013.
- 10 Paul Beame, Thathachar S Jayram, and Michael Saks. Time-space tradeoffs for branching programs. *Journal of Computer and System Sciences*, 63(4):542–572, 2001.
- 11 Paul Beame, Michael Saks, Xiaodong Sun, and Erik Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *JACM*, 50(2):154–195, 2003.
- 12 Paul Beame and Erik Vee. Time-space tradeoffs, multiparty communication complexity, and nearest-neighbor problems. In *STOC*, pages 688–697. ACM, 2002.
- 13 Allan Borodin and Stephen Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM Journal on Computing*, 11(2):287–297, 1982.
- 14 Samuel R. Buss and Ryan Williams. Limits on Alternation Trading Proofs for Time-Space Lower Bounds. *Computational Complexity*, 24(3):533–600, 2015.
- 15 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2009.
- 16 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
- 17 Lance Fortnow, Richard J. Lipton, Dieter van Melkebeek, and Anastasios Viglas. Time-space lower bounds for satisfiability. *JACM*, 52(6):835–865, 2005.

- 18 Etienne Grandjean. Linear Time Algorithms and NP-Complete Problems. *SIAM J. Comput.*, 23(3):573–597, 1994.
- 19 Christian Hoffmann. Exponential Time Complexity of Weighted Counting of Independent Sets. *CoRR*, abs/1007.1146, 2010. [arXiv:1007.1146](https://arxiv.org/abs/1007.1146).
- 20 Richard E Ladner and Nancy A Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10(1):19–32, 1976.
- 21 Yishay Mansour, Noam Nisan, and Prasoos Tiwari. The computational complexity of universal hashing. *Theoretical Computer Science*, 107(1):121–133, 1993.
- 22 Jakob Pagter. On Ajtai’s Lower Bound Technique for R-way Branching Programs and the Hamming Distance Problem. *Chicago J. Theor. Comput. Sci.*, 2005.
- 23 Wolfgang J. Paul, Nicholas Pippenger, Endre Szemerédi, and William T. Trotter. On Determinism versus Non-Determinism and Related Problems (Preliminary Version). In *FOCS*, pages 429–438, 1983.
- 24 Rahul Santhanam. Lower bounds on the complexity of recognizing SAT by Turing machines. *Inf. Process. Lett.*, 79(5):243–247, 2001.
- 25 Martin Sauerhoff and Philipp Woelfel. Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions. In *STOC*, pages 186–195. ACM, 2003.
- 26 Dieter van Melkebeek. A survey of lower bounds for satisfiability and related problems. *Foundations and Trends in Theoretical Computer Science*, 2(3):197–303, 2007.
- 27 R. Ryan Williams. Time-Space Tradeoffs for Counting NP Solutions Modulo Integers. *Computational Complexity*, 17(2):179–219, 2008.
- 28 Ryan Williams. Nonuniform ACC Circuit Lower Bounds. *JACM*, 61(1):2, 2014.
- 29 Andrew C. Yao. Near-optimal time-space tradeoff for element distinctness. In *FOCS*, pages 91–97, 1988.
- 30 Yaacov Yesha. Time-space tradeoffs for matrix multiplication and the discrete Fourier transform on any general sequential random-access computer. *Journal of Computer and System Sciences*, 29(2):183–197, 1984.