Amplifying Circuit Lower Bounds Against Polynomial Time, With Applications*

Richard J. Lipton
College of Computing
Georgia Institute of Technology
Atlanta, GA
rjl@cc.gatech.edu

Ryan Williams
Computer Science Department
Stanford University
Stanford, CA
rrw@cs.stanford.edu

Abstract

We give a self-reduction for the Circuit Evaluation problem (CircEval), and prove the following consequences.

- **Amplifying Size-Depth Lower Bounds.** If CircEval has Boolean circuits of n^k size and $n^{1-\delta}$ depth for some k and δ, then for every $\varepsilon > 0$, there is a $\delta' > 0$ such that CircEval has circuits of $n^{1+\varepsilon}$ size and $n^{1-\delta'}$ depth. Moreover, the resulting circuits require only $\tilde{O}(n^{\varepsilon})$ bits of non-uniformity to construct. As a consequence, strong enough depth lower bounds for Circuit Evaluation imply a full separation of P and NC (even with a weak size lower bound).
- **Lower Bounds for Quantified Boolean Formulas.** Let $c, d > 1$ and $e < 1$ satisfy $c < (1 - e + d)/d$. Either the problem of recognizing valid quantified Boolean formulas (QBF) is not solvable in $\mathsf{TIME}[n^c]$, or the Circuit Evaluation problem cannot be solved with circuits of n^d size and n^e depth. This implies unconditional polynomial-time uniform circuit lower bounds for solving QBF. We also prove that QBF does not have n^c-time uniform NC circuits, for all $c < 2$.

1 Introduction

Recently, Allender and Koucký [AK10] have proved that if certain weak lower bounds hold at all, they can be *amplified*, in the sense that lower bounds of the form $n^{1+\varepsilon}$ can be extended to arbitrary n^k lower bounds for every k. For example, Allender and Koucký show that if the NC^1-complete problem Balanced Formula Evaluation (BFE) is contained in TC^0, then BFE has TC^0 circuits of size $n^{1+\varepsilon}$, *for every* $\varepsilon > 0$. It follows that even (seemingly) modest TC^0 lower bounds on BFE would imply that $\mathsf{TC}^0 \neq \mathsf{NC}^1$. Similar results are proved for other problems within NC^1 and NL. All of them have the form: "an $n^{1+\varepsilon}$ lower bound against constant-depth class $\mathcal{C}$ implies arbitrary polynomial lower bounds against the same constant-depth class $\mathcal{C}$."

*A preliminary version of this paper appeared in the 2012 IEEE Conference on Computational Complexity [LW12].

These results suggest an intriguing approach to separating complexity classes: find a problem for which it is possible to prove small concrete lower bounds, then find an "amplifying" result that extends the small bound to arbitrary polynomials. For instance, if we could prove that

$$\text{SAT} \in \mathsf{TIMESPACE}[n^{O(1)}, \log n] \Rightarrow \text{SAT} \in \mathsf{TIMESPACE}[n^{1.1}, n^{.7}],$$

then we would separate NP from LOGSPACE, using known time-space tradeoff lower bounds [For00, FLvMV05, BW12]. It is unclear whether we should expect to prove such an amplification result for SAT. Allender and Koucký show that any problem with their "oracle self-reduction" property must already be in NC. (However, this does not rule out the possibility of *using the assumption* that SAT is contained in logspace, to obtain an amplifying reduction for SAT.)

Our Results First, we prove an amplification lemma for a P-complete problem, the Circuit Evaluation problem (CircEval), relative to low-depth circuit lower bounds. (The Circuit Evaluation problem is: given a circuit C and input x, determine if $C(x) = 1$.)

Let $\mathsf{SIZEDEPTH}[s(n), d(n)]$ be the class of languages recognized by $s(n)$-size $d(n)$-depth circuits.

Theorem 1.1 (Section 3)**.** *The following are equivalent:*

- *There is a $k \geq 1$ and $\delta > 0$ such that*

$$\text{CircEval} \in \mathsf{SIZEDEPTH}[n^k, n^{1-\delta}].$$

- *For all $\varepsilon > 0$, there is a $\delta' > 0$ such that*

$$\text{CircEval} \in \mathsf{SIZEDEPTH}[n^{1+\varepsilon}, n^{1-\delta'}].$$

  Moreover, these CircEval *circuits need at most $\tilde{O}(n^{\varepsilon})$ bits of non-uniformity; that is, the circuits can be generated in polynomial time by an algorithm given only $n^{\varepsilon} \cdot poly(\log n)$ bits of advice.*

It follows that, in order to separate P from polynomial-size, $n^{o(1)}$-depth circuits, it suffices to prove that CircEval is not in $\mathsf{SIZEDEPTH}[n^{1.001}, n^{1-o(1)}]$ for circuits with $n^{.001}$-bit descriptions. That is, any nontrivial size lower bound for sufficiently large depth implies NC $\neq$ P. This is counterintuitive: we believe that considering $n^{1.001}$ size lower bounds should be an easier task than considering arbitrary polynomial-size lower bounds. Theorem 1.1 says that they are equivalent tasks for P problems in $n^{1-\delta}$ depth.

The amplifying reduction of Theorem 1.1 has applications to proving *unconditional* lower bounds. As the resulting small circuits have $\tilde{O}(n^{\varepsilon})$ bits of non-uniformity (even if the original polynomial size circuits did not), this reduction can be combined with other ingredients to prove unconditional time lower bounds for constructing low-depth circuits.

It has been known since Kannan [Kan82] that for every k, there is an $L_k \in \mathsf{NP}$ that does not have P-uniform circuits of size $O(n^k)$. (That is, no polynomial time algorithm can construct $O(n^k)$-size circuits computing L_k.) However, it is an open problem to produce a *natural* problem of interest (such as SAT) that does not have such circuits (for some $k \geq 1$). Even producing a

natural problem in PSPACE with this property is open. Perhaps the most promising candidate is the canonical PSPACE-complete problem of Quantified Boolean Formulas (QBF). Although we do not believe QBF has *non-uniform* NC circuits (i.e., we believe PSPACE $\not\subset$ NC/poly), and we know that QBF does not have LOGSPACE-uniform NC circuits (by the space hierarchy theorem), it is open whether PSPACE has P-uniform NC circuits [All89]. Indeed, it was even open if QBF had $n^{1+o(1)}$-time uniform NC circuits (Eric Allender, personal communication).

Combining the ideas in Theorem 1.1 with other ingredients, we can obtain non-linear time lower bounds for generating low-depth circuits solving QBF:

Theorem 1.2 (Section 4)**.** *Let $c, d \geq 1$ and $e < 1$ satisfy the inequality $c < (1 - e + d)/d$. Either*

- *QBF $\notin$ TIME$[n^{c+o(1)}]$, or*
- CircEval *does not have circuits of $n^{d+o(1)}$ size and $n^{e+o(1)}$ depth.*

Three interesting settings to c, d (letting $e = 0$) yield the following corollaries for multitape Turing machines:

Corollary 1.1. *QBF does not have $O(n^{1.618})$-time uniform circuits of depth $n^{o(1)}$.*

Corollary 1.2. *QBF does not have $O(n^{2-\varepsilon})$-time uniform circuits of $n^{1+o(1)}$ size and $n^{o(1)}$ depth, for all $\varepsilon > 0$.*

Corollary 1.3. *Either QBF cannot be solved in $n^{1+\varepsilon}$ time for some $\varepsilon > 0$, or P $\not\subset$ NC/poly.*

The proof of Theorem 1.2 is by contradiction: assuming QBF can be efficiently solved and Circuit Evaluation has small low-depth circuits, we show how to simulate every language recognized by alternating machines in t time, using only $t^{1-\varepsilon}$ time for some $\varepsilon > 0$, contradicting the time hierarchy for alternating machines. This simulation is rather non-trivial as the low-depth circuits for CircEval are neither assumed to be uniform nor are they assumed to have a short description; they may take considerable resources to construct. However, by exploting alternation, we can guess-and-verify very small circuits for CircEval, and apply the ideas of Section 3 to yield larger circuits for CircEval which, combined with the assumption on QBF, yields a faster alternating simulation.

It is worth contrasting Theorem 1.2 with the seminal result of Hopcroft, Paul, and Valiant [HPV77] who showed that TIME$[t] \subseteq$ SPACE$[t/\log t]$. The proof of Theorem 1.2 yields a conditional improvement: if CircEval were in n^d size and n^e depth, then

$$\text{TIME}[t] \subseteq \text{ATIME}[t^{1/c}] \subseteq \text{SPACE}[t^{1/c}].$$

That is, low-depth (non-uniform) simulation of P implies a certain *uniform* low-depth simulation of P (as alternating time is a uniform analogue of circuit depth).

Our final theorem is an improvement on the time lower bound of Corollary 1.1:

Theorem 1.3 (Section 5)**.** *For all $\varepsilon > 0$, QBF does not have $n^{2-\varepsilon}$-time uniform circuits of $n^{2-\varepsilon}$ size and polylog depth.*

The argument is an intricate extension of Theorem 1.2; it is again a proof by contradiction, but this proof exploits the fact that the low-depth circuits can be constructed efficiently, to recursively build an alternating simulation that does not have to directly guess small circuits for CircEval.

Prior work has shown how slightly-faster-than-brute-force algorithms for hard problems can imply circuit lower bounds for $\mathsf{EXP}^{\mathsf{NP}}$ and NEXP (e.g., [IKW02, KI04, Wil11]). At the other end of the spectrum, one may investigate the consequences of ultra-efficient algorithms for hard problems, such as nearly-linear time algorithms (which may construct low-depth circuits) for QBF, to explore the space of absurdities that result. Such algorithms can sometimes yield circuit lower bounds so strong that they contradict the existence of the algorithm. This is another way of looking at Theorem 1.2 and its corollaries.

2 Preliminaries

All functions are considered to be time constructible, unless otherwise specified. We use the notation $\mathrm{poly}(x)$ to denote expressions of the form $x^c + c$ where c is a universal constant, independent of x. We use $\tilde{O}$ notation to suppress $\mathrm{poly}(\log n)$ factors in the running time (this is standard notation in algorithms).

In this paper, our model of computation is the multitape Turing machine, although sometimes random-access machines suffice (as we will now explain). When we consider *alternating* computation, it is important to keep in mind that multitape Turing machines and random access machines are known to be equivalent in running time (up to polylogarithmic extra factors). That is, every alternating random-access machine running in time t (where $t(n) \geq n$) can be simulated by an alternating multitape TM running in time $t \cdot \mathrm{poly}(\log t)$ (Gurevich and Shelah [GS89] proved this result for nondeterministic computation, but their proof extends to alternating computation straightforwardly).

Moreover, recall that, via the Hennie-Stearns simulation [HS66], multitape TMs and two-tape TMs are time-equivalent up to polylog factors, and this simulation holds equally well for alternating computation (the two-tape TM may write a record of all the alternating "guesses" at the beginning of the computation, then simulate the multitape TM on those guesses). Hence we may assume the two-tape Turing machine model in our simulations. In any case, for simplicity we often say "algorithm" instead of the specific computational model.

In this paper, bounds of the form $n \cdot \mathrm{poly}(\log n)$ are called *quasi-linear*. We define DQL (Deterministic Quasilinear Time) to be the class of languages solvable by a multitape Turing machine in $n \cdot \mathrm{poly}(\log n)$ time [Sch78, NRS95].

We say that a language $L^\star$ is DQL-*hard* if for all $L \in \mathsf{DQL}$, there is a *polylog-time* reduction R from L to $L^\star$. More precisely, there is a constant c and an algorithm A that is given random access to (x, i) where i ranges from 1 to $n \cdot (\log n)^c$, such that A outputs the ith bit of $R(x)$ in $O((\log n)^c)$ time. (An alternative definition is that A can be implemented by a multitape Turing machine in $\mathrm{poly}(\log n)$ time, given $O(n)$ initial steps to first position the tape heads, during which period the TM cannot write.)

Our specific formulation of the quantified Boolean formula problem (QBF) is: given a sentence

of the form

$$\psi = (\exists x_1)(\forall x_2)\cdots(\exists x_n)\phi(x_1,\ldots,x_n)$$

where ϕ is a 3-CNF formula over Boolean variables $x_1,\ldots,x_n$, determine if ψ is true. It is well known that QBF is PSPACE-complete. A key property we use of QBF can be derived from the DQL-hardness of CircEval. Define $\mathsf{ATIME}[t(n)]$ to be the class of problems solvable in $t(n)$ time on an alternating Turing machine. Define AQL (Alternating Quasilinear Time) as the class

$$\mathsf{AQL} := \bigcup_{k\geq 1} \mathsf{ATIME}[n\cdot(\log^k n)].$$

QBF is known to be complete for AQL under a strong reducibility notion. The following theorem is implicit in work of Gurevich and Shelah [GS89]; see also Fortnow [For00].

Theorem 2.1. *QBF is* AQL*-complete for polylog-time reductions.*

Some Circuit Definitions Let $\mathsf{SIZEDEPTH}[s(n),d(n)]$ be the class of languages recognized by a $s(n)$-size $d(n)$-depth family of circuits. For a machine-based complexity class $\mathcal{C}$, a circuit family $\{C_n\}$ is *$\mathcal{C}$-uniform* if there is a multitape Turing machine M obeying the constraints of class $\mathcal{C}$ which produces a description of C_n on the input 1^n. Note that, if $\mathcal{C}\subseteq\mathsf{TIME}[t(n)]$ for some t, then the number of gates in such a C_n can be no more than $t(n)$.

We say that a circuit family $\{C_n\}$ *has $b(n)$ bits of non-uniformity* if there is an infinite family of strings $\{s_n\}$, where $|s_n|\leq b(n)$, and a polynomial time algorithm A, such that $A(1^n,s_n)$ outputs C_n. That is, the n^{th} circuit in the family can be efficiently described with only $b(n)$ bits. By a counting argument, most $s(n)$-size circuit families do not have less than $O(s(n)\log s(n))$ bits of non-uniformity, so the set of circuit families with low uniformity (say, $n^{.001}$ bits) is a fairly restricted class.

Some Related Prior Work Lipton and Viglas [LV03] proved that if $\mathsf{P}\subset\mathsf{NC/poly}$ then $\mathsf{TIME}[t]\subseteq\mathsf{SPACE}[t^{1-\varepsilon}]$ for some $\varepsilon>0$. (Compare with Corollary 1.3.) Their proof applies block-respecting Turing machines in a similar but not identical way.

Several uniform circuit lower bounds are known for SAT and the Permanent function. Allender and Koucký [AK10] prove there exists a $\delta>0$ such that SAT cannot be solved with $n^{1+\delta}$ size LOGTIME-uniform TC^0 circuits. In the pioneering time-space tradeoffs work of Fortnow [For00], he also proved that SAT does not have LOGSPACE-uniform branching programs of $n^{1+o(1)}$ size, and proves a result (attributed to Buhrman) that SAT does not have LOGSPACE-uniform NC^1 circuits of $n^{1+o(1)}$ size. Allender *et al.* [AKRRV01] extended this to prove that SAT does not have LOGSPACE-uniform SAC^1 circuits of $n^{1+o(1)}$ size.

For the Permanent function, Allender [All99] proved that the Permanent does not have LOGTIME-uniform TC^0 circuits of quasi-polynomial size, which uses the fact that the Permanent is #P-complete under efficient reductions. Koiran and Perifel [KP09] extended this lower bound, showing that the Permanent cannot be computed by LOGTIME-uniform polynomial-size threshold circuits of $o(\log\log n)$ depth. Very recently, Chen and Kabanets [CK12] and Kinne [K12] have proved that the Permanent does not have LOGTIME-uniform TC^0 circuits where the connection language is polynomial-time computable with $n^{o(1)}$ bits of non-uniformity.

To compare, we show that QBF does not have n^e depth circuits constructible in n^c time, for various $c > 1$ and $e < 1$. That is, we consider a more powerful circuit class with a more powerful uniformity notion, but prove lower bounds for an apparently harder problem. This requires a different approach from the earlier lower bounds, which crucially rely on either clever simulations of severely space-restricted machines or the extreme (logtime) uniformity of the circuit classes.

3 Amplifying CircEval Lower Bounds

We start with a completeness result for CircEval that is apparently folklore, but not well known:

Theorem 3.1 (Folklore)**.** CircEval *is* DQL*-complete for polylog-time reductions. That is, for all* $L \in$ DQL*, there is a quasi-linear size circuit family* $\{C_n\}$ *such that:*

- $x \in L$ *if and only if* $C_{|x|}(x) = 1$*, and*
- $\{C_n\}$ *is polylog-time uniform: given* n *and a gate index* i *of* C_n *(both encoded in* $O(\log n)$ *bits), in poly*$(\log n)$ *time we can compute the tuple* $\langle j_1, j_2, g\rangle$ *where* j_1 *and* j_2 *are the indices of the two possible gates in* C_n *which are input to gate* i*, and* g *is the gate type of gate* i*.*

Proof. Let M be a quasilinear time machine. We say that M is *oblivious* if for every n and input x of length n, the tape head movements of $M(x)$ depend only on n. That is, M makes precisely the same sequence of head movements on every input x of length n. Hennie and Stearns [HS66] showed that for every multitape Turing machine M running in $O(t(n))$ time there is an equivalent *oblivious* M' that uses only two tapes and runs in $O(t(n) \log t(n))$ time. From their simulation, it is easy to construct (in quasilinear time) a circuit C' on $O(t(n) \log t(n))$ gates that simulates M' on n-bit inputs, completing the reduction from $L(M)$ to CircEval. (Note that Pippenger and Fischer [PF79] gave a simplified, recursive construction.)

This reduction can be made *polylog-time* uniform: if we want to compute the ith bit of the description of the resulting CircEval instance, this can be done in poly$(\log n)$ time (independently of the other bits). The proof that polylog-time uniformity is possible is essentially the same as the proof of Fortnow *et al.* [FLvMV05] that SAT is NP-complete under polylog-time uniform reductions. Their proof observes that the oblivious simulation of Hennie-Stearns and Pippenger-Fischer is extremely regular: given the index i of a given timestep (encoded in $O(\log n)$ bits), we can efficiently compute (in poly$(\log n)$ time) the head positions of the two tapes at timestep i. (We sketch some details of why this is true in the Appendix.) Therefore, we can efficiently produce the $O(1)$ gates of the circuit that simulate the transition function of M' at timestep i.

It remains to show that CircEval itself is in DQL. First, given a circuit C as a list of tuples (i, j_1, j_2, g), indicating that gate i has gate type g and takes its inputs from the outputs of gates j_1, j_2, topologically sort the nodes of the directed acyclic graph corresponding to C, in quasi-linear time. Then evaluate each gate of the circuit in increasing topological order, starting with the inputs. On a standard algorithm model, this is a simple dynamic program that can be solved in linear time. For multitape TMs, Fischer ([Fis74], pp.13–14) showed how to solve this problem in quasilinear time on a multitape Turing machine. (See also Pippenger [Pip77] for a proof that CircEval $\in$ DQL.)

In more detail, Fischer solved the *mail-carrier problem*: a mail carrier has a number of pushdown stacks containing letters to be delivered. Each letter has a number $i \in \{1, \ldots, n\}$ corresponding to a house number. The mail carrier visits houses $1, \ldots, n$ in that order, and must deliver and receive letters online (at house i, all letters numbered i must be delivered, and all letters that i wishes to send must be pushed on some stack before going to house $i+1$). Fischer shows that this problem can be solved in such a way that each letter is handled $O(\log n)$ times on a multitape Turing machine. To apply this to CircEval, imagine that we visit each gate of C in increasing topological order. Each gate i receives letters addressed to it (its input bits) and uses that to compute the output bit, which is sent in letters to the gates later in the topological order that i points to. □

Next, we prove that if CircEval has low-depth circuits at all, it must have small low-depth circuits. Moreover, these small low-depth circuits have sublinear-size representations. The proof of this implication stems from the fact that CircEval can be solved efficiently on multitape Turing machines, and that computation is rather local on multitape TMs: the computation can be divided into t/b time blocks of $O(b)$ steps each, where only $O(b)$ bits of input are read in each time block, and $O(b)$ bits of output are generated. By replacing each block with a low-depth circuit, we get a "somewhat low depth" circuit that has small size.

Lemma 3.1 (Size Reduction Lemma)**.** *If* CircEval *can be recognized in $t(n)$ time on a multitape Turing machine and* CircEval *has (non-uniform) circuits of size $s(n)$ and depth $d(n)$, then for all functions $b(n)$ such that $\Omega(\log n) \le b(n) \le t(n)$, there is a $\tilde{b}(n) \le b(n) \cdot \mathit{poly}(\log b(n))$ such that* CircEval *has circuits of*

$$
\begin{array}{l}
O(t(n) \log t(n) \cdot s(\tilde{b}(n))) \text{ size,} \\
O(\frac{t(n) \cdot \log t(n)}{b(n)} \cdot d(\tilde{b}(n))) \text{ depth, and} \\
O(s(\tilde{b}(n)) \log s(\tilde{b}(n))) \text{ bits of non-uniformity.}
\end{array}
$$

Moreover, assuming we are given these bits of non-uniformity as additional input, the resulting circuits for CircEval *are constructible in $O(t(n) \cdot \mathit{poly}(\log t) \cdot s(\tilde{b}(n)))$ time.*

Proof. Let M be a multitape Turing machine for CircEval that runs in $O(t(n))$ time. First, we convert M into an oblivious M', as in Theorem 3.1. M' runs in $t'(n) = O(t(n) \log t(n))$ time. As mentioned in the proof of Theorem 3.1, this oblivious two-tape simulation is polylog-time uniform, in that we can determine the head positions in step i in $\mathrm{poly}(\log t)$ time.

Next, the computation of M' can be partitioned using an old trick of Hopcroft, Paul, and Valiant [HPV77], who call the resulting machines *block-respecting*. The result is a reimplementation of M' so that its computation of $O(t'(n))$ time can be partitioned into $\frac{t'(n)}{b(n)}$ time blocks of $O(b(n))$ steps each, and each tape is partitioned into $O(\frac{t'(n)}{b(n)})$ tape blocks of $O(b(n))$ cells each. For every time block, and each tape head, the head stays within exactly one tape block. So we can think of each time block as running on an input of $O(b(n))$ bits, and each block outputs $O(b(n))$ bits (the new content of those tape blocks). Note that the Hopcroft-Paul-Valiant simulation maintains the obliviousness of M'.[1]

[1] We remark that, in principle, this additional block-respecting condition is not required, although we think it does simplify the overall picture. After the machine is made two-tape and oblivious, we can still partition the computation

Given that M' is oblivious, the block-respecting simulation is also uniform, in the following sense: for every n-bit input x to M' we can efficiently construct a *computation graph* $G_{M'}$ on $N = \frac{t'(n)}{b(n)}$ nodes, where each node $v \in \{1, \ldots, N\}$ represents a time block v, and there is an edge (v, v') if the $O(b(n))$ bits of content output by block v is needed as input to the computation of block v'. More precisely, an edge (v, v') can arise in two ways. Either:

- v immediately precedes v' in the computation, in which case the final state and head positions of the computation in block v is needed to give the initial state and head positions of block v', or
- block v writes content to some tape block, and this tape block is not read again until block v'.

Because M' is oblivious, the graph $G_{M'}$ is the same for all n-bit inputs. Observe that $G_{M'}$ has $O(N)$ edges: for every node v of $G_{M'}$ and for each tape of M', there are at most two time blocks (the block immediately prior to block v, and the last time block accessing the same tape block) that could affect the input to block v, so each v has at most 2 incoming edges for each of the two tapes of M'.

Since CircEval is DQL-hard, the problem of computing the outputs of one block (given the inputs) can be efficiently reduced to $O(b(n))$ instances of CircEval, each instance having size $\tilde{b}(n) = b(n) \cdot \mathrm{poly}(\log b(n))$. Furthermore, if CircEval has $s(n)$-size $d(n)$-depth circuits, then each of these $O(b(n))$ instances can be computed with an $S = s(\tilde{b}(n))$ size, $D = d(\tilde{b}(n))$ depth circuit, call it C.

Suppose we construct a Boolean circuit C' which replaces all nodes of $G_{M'}$ with copies of C, feeding the $O(b(n))$-bit outputs of the various copies of C to the appropriate inputs of other copies, as dictated by the edges of $G_{M'}$. Note that by properties of the block-respecting simulation, C' can be uniformly constructed, given access to C. As there are $N = \frac{t'(n)}{b(n)}$ blocks and $O(b(n))$ circuits per block, the final circuit for CircEval has $O(t(n) \log t(n) \cdot S)$ size and $O(\frac{t(n) \log t(n)}{b(n)} \cdot D)$ depth, and the amount of non-uniformity needed is just $O(S \log S)$ bits, to encode C. □

Since every language in $\mathsf{TIME}[n^c]$ has $n^{c+o(1)}$ size circuits, the proof of Lemma 3.1 yields the following important corollary:

Corollary 3.1. *Let $T(n) \geq n$. Assume* CircEval *has (non-uniform) circuits of size $s(n)$ and depth $d(n)$. Then all $b(n)$ such that $\Omega(\log T(n)) \leq b(n) \leq T(n)$, every $L \in \mathsf{TIME}[T(n)]$ has circuits of $T(n) \cdot s(b(T(n))^{1+o(1)})$ size and $\frac{T(n)^{1+o(1)}}{b(T(n)^{1+o(1)})} d(b(T(n))^{1+o(1)})$ depth, with $s(b(T(n))^{1+o(1)})$ bits of non-uniformity.*

into t'/b blocks of b steps each, and partition the two tapes into blocks of b cells, but it is not necessary to force the two tape heads to stay in one tape block during each time block. Indeed, within b time steps, at most two blocks of tape could possibly be visited by a tape head; therefore, at most four tape blocks (two from each tape), as well as the current state and head position, are needed to compute the input/output of any given time block. (This observation seems to have first been made by Paul and Reischuk [PR80].) This affects the rest of the proof only in minor ways; the input/output size of each block computation increases by a small constant factor due to the extra tape blocks. The definition of the computation graph $G_{M'}$ remains the same, but now the indegree of a node could be 5 instead of 4 (for a node v, each of the two tapes can be responsible for two incoming edges, and there may be a 5th incoming edge from the node corresponding to the block immediately prior to v).

Proof. By Theorem 3.1, CircEval is in $t(n) = n^{1+o(1)}$ time, and every $L \in \mathsf{TIME}[T(n)]$ can be reduced to CircEval instances of $Z = T(n)^{1+o(1)}$ size. By Lemma 3.1 and the assumption of the theorem, CircEval on inputs of size Z has circuits of $Z^{1+o(1)} \cdot s(b(Z)^{1+o(1)})$ size and $\frac{Z^{1+o(1)}}{b(Z)} \cdot d(b(Z)^{1+o(1)})$ depth, for $b(n)$ in the appropriate range. So L has circuits of size $T(n)^{1+o(1)} \cdot s(b(T(n))^{1+o(1)})$ and depth $\frac{T(n)^{1+o(1)}}{b(T(n)^{1+o(1)})} \cdot d(b(T(n))^{1+o(1)})$. □

Lemma 3.1 also has the following intriguing consequence which, on the face of it, has nothing to do with Turing machines.

Reminder of Theorem 1.1 *The following are equivalent:*

- *There is a $k \geq 1$ and $\delta > 0$ such that*

$$\text{CircEval} \in \mathsf{SIZEDEPTH}[n^k, n^{1-\delta}].$$

- *For all $\varepsilon > 0$, there is a $\delta' > 0$ such that*

$$\text{CircEval} \in \mathsf{SIZEDEPTH}[n^{1+\varepsilon}, n^{1-\delta'}].$$

  Moreover, these CircEval *circuits need at most $\tilde{O}(n^{\varepsilon})$ bits of non-uniformity; that is, the circuits can be generated in polynomial time by an algorithm given only $n^{\varepsilon} \cdot poly(\log n)$ bits of advice.*

Proof. One direction (the second implies the first) is obvious. For the other, we set $b(n) = n^{\frac{\varepsilon}{k}}$ in the Size Reduction Lemma (Lemma 3.1); by Theorem 3.1, it suffices to use $t(n) = n^{1+o(1)}$ in the Lemma. Letting $\delta' = \delta \cdot \varepsilon/k$, we obtain circuits for CircEval with size

$$n^{1+o(1)} \cdot \left(n^{\varepsilon/k}\right)^k = n^{1+\varepsilon+o(1)}$$

and depth

$$\frac{n^{1+o(1)}}{n^{\varepsilon/k}} \cdot \left(n^{\varepsilon/k+o(1)}\right)^{1-\delta} \leq n^{1-(\varepsilon/k)\delta+o(1)} = n^{1-\delta'+o(1)},$$

with $n^{\varepsilon+o(1)}$ non-uniformity. The $o(1)$ factors can be neglected by simply adjusting δ' and $b(n)$ to be a little smaller. □

Corollary 3.2. *Suppose* CircEval *does not have circuits of $n^{1.1}$ size and $n^{1-\delta}$ depth, for every $\delta > 0$. Then $\mathsf{NC} \neq \mathsf{P}$.*

Hence even a modest size lower bound for $o(n)$-depth circuits is enough to separate P and NC.

4 Uniform Circuit Lower Bounds for Quantified Boolean Formulas

It is a major open problem to prove that QBF is not solvable in $n^{1+\varepsilon}$ time, for some $\varepsilon > 0$. The only relevant result (to our knowledge) is a barely-superlinear time lower bound for a non-standard encoding of QBF [Will08]. A preliminary step would be to establish that QBF does

not have $n^{1+\varepsilon}$ time algorithms under further restrictions on the algorithms. For one-tape Turing machines, it has been known since the 1960's that PALINDROMES cannot be solved in $O(n^2)$ time, so QBF lower bounds are easy in that setting [Cob66]. For algorithms using only $n^{1+\varepsilon}$ time and $n^{o(1)}$ space simultaneously, it is also easy to prove lower bounds: it is well known that PSPACE $\not\subseteq$ LOGSPACE-uniform NC, simply because the latter is contained in polylogarithmic space.

An interesting extension would be to prove that PSPACE $\not\subseteq$ P-uniform NC. (The class is also called PUNC [All89].) This separation problem asks if we can solve QBF efficiently in parallel, provided we are allowed polynomial-time preprocessing (in serial) to prepare the parallel algorithm for its computation. Such a separation would bring us a little closer to proving P $\neq$ PSPACE. However, currently the separation of PSPACE from P-uniform NC is still an open problem.

The amplification lemma of the previous section can be applied to prove new uniform circuit lower bounds for solving quantified Boolean formulas. At a high level, the general idea is similar to that of time-space tradeoff lower bounds for SAT [For00, FLvMV05, vMe06]. First we use the assumption that QBF has nice circuits to "speed up" generic deterministic time-bounded computations (using alternations in our sped-up simulation of deterministic time). Then we use the assumption again to remove these alternations efficiently, resulting in a contradiction to the time hierarchy theorem. However, the actual arguments involved in our lower bounds are ultimately quite different from those of the SAT lower bounds. In particular, the "speed-up" portion of our simulation only works conditioned on the assumption that P has low-depth circuits, unlike in the time-space setting (where the "speed up" simulation used there works unconditionally).

Reminder of Theorem 1.2 *Let $c, d \geq 1$ and $e < 1$ satisfy $c < (1 - e + d)/d$. Either*

- *QBF $\notin$ TIME$[n^{c+o(1)}]$, or*
- CircEval *does not have circuits of $n^{d+o(1)}$ size and $n^{e+o(1)}$ depth.*

Proof. Suppose that both

(A) QBF $\in$ TIME$[n^{c+o(1)}]$ and

(B) CircEval has circuits of $n^{d+o(1)}$ size and $n^{e+o(1)}$ depth.

We wish to conclude a contradiction.

Combining assumption (A) with the completeness of CircEval for quasilinear time (Theorem 3.1), we can infer that every QBF instance of length n can be reduced in $n^{c+o(1)}$ time to a CircEval instance of size $n^{c+o(1)}$ with n input bits. The completeness of QBF for alternating quasi-linear time (Theorem 2.1) means that, if QBF is solvable in $n^{c+o(1)}$ time, then every language in ATIME$[O(n)]$ is in $n^{c+o(1)}$ time. By a standard padding/translation argument, this entails that for all polynomials $t(n)$ we have

$$\text{ATIME}[t] \subseteq \text{TIME}[t^{c+o(1)}]. \tag{1}$$

Assuming (B) is true, we will derive a "fast" simulation of deterministic time:

$$\text{TIME}[t^{c+o(1)}] \subseteq \text{ATIME}[t^{c(1-\varepsilon/d)+ec\varepsilon/d+o(1)} + t^{c\varepsilon+o(1)}], \tag{2}$$

for all $\varepsilon \in (0, d)$. To minimize the running time of the simulation, we set

$$c(1 - \varepsilon/d) + ec\varepsilon/d = c\varepsilon.$$

and obtain

$$\varepsilon = d/(d + 1 - e).$$

That is, from assumption (B) we conclude that there is a simulation of every t^c time multitape Turing machine that runs in alternating time $t^{cd/(d+1-e)+o(1)}$. But this exponent is less than 1, provided that $c < (1 - e + d)/d$. Hence the combination of (1) and (2) yields a simulation of every language in alternating time $t(n)$ that runs in alternating time $o(t(n))$. This is a contradiction to the alternating time hierarchy, which follows from a direct diagonalization, and says that

$$\mathsf{ATIME}[o(t(n)] \subsetneq \mathsf{ATIME}[t(n)].$$

Let us discuss how to prove (2). First, note we may assume without loss of generality that our alternating simulation of CircEval runs on an alternating *random-access* machine. (As mentioned in the Preliminaries, alternating RAMs running in time t can be simulated by an alternating 2-tape Turing machine running in $t \cdot \mathrm{poly}(\log t)$ time; this is due to Gurevich and Shelah [GS89]. See also Paul, Prauss, and Reischuk [PPR80] and Fortnow and Lund [FL93].)

Let M be a machine running in $t^{c+o(1)}$ time that we wish to simulate quickly. By Corollary 3.1 of Lemma 3.1 (letting $s(n) = n^{d+o(1)}$, $d(n) = n^{e+o(1)}$, and $b(n) = n^{\varepsilon/d}$) and assumption (B), there is a circuit D that can simulate M on inputs of length n, where D has at most $t^{O(dc)}$ size, $t^{c(1-\varepsilon/d)+ec\varepsilon/d+o(1)}$ depth, and $t^{c\varepsilon+o(1)}$ bits of non-uniformity. The bits of non-uniformity encode a small $t^{c\varepsilon+o(1)}$-size circuit D' for CircEval on instances of size $t^{c\varepsilon/d}$. Given free access to D', we can efficiently construct the circuit D in $t^{O(dc)}$ time, by constructing the computation graph G with $t^{c+o(1)-c\varepsilon/d}$ nodes from Lemma 3.1, and replacing each of its nodes with copies of D'.

We would like to evaluate D efficiently. One could evaluate D in the standard gate-by-gate way, but as the size of D is $t^{O(1)}$, the standard simulation runs too slowly. Instead, we will exploit the low depth of D to get a faster alternating simulation.

In particular, we can simulate the evaluation of D on an input x in alternating time

$$t^{c(1-\varepsilon/d)+ec\varepsilon/d+o(1)} + t^{c\varepsilon+o(1)},$$

as follows. First, *existentially* guess the computation graph G for M on $t^{c+o(1)-c\varepsilon/d} = t^{c(1-\varepsilon/d)+o(1)}$ nodes, with nodes indexed by time blocks. For each node v of G representing a time block, guess strings $s_{v,1}$ and $s_{v,2}$ of $O(\log t(n))$ bits indicating which $b(n)$-length block of tape cells (on each of the two tapes of M) is being accessed in time block v. Next, verify that G is consistent with respect to $s_{v,1}$ and $s_{v,2}$, for all v. That is, verify that:

- vertex 1 has $s_{1,1} = s_{1,2} = 1$
  (initially, the first block of cells is being read, on both of the tapes)

- for all i and $j = 1, 2$, $s_{i+1,j} \in \{s_{i,j} - 1, s_{i,j}, s_{i,j} + 1\}$
  (from one time block to the next, the index of the tape block can change by at most 1)

- for all $i > 1$, there are edges $(j_1, i), (j_2, i)$ in G where $j_1, j_2 < i$, and $s_{j_1,1} = s_{i,1}$, $s_{j_2,2} = s_{i,2}$ *(for each time block i, there are two previous time blocks j_1, j_2, the outputs of which are inputs to i, and time blocks j_1, j_2 wrote to the same two tape blocks that i is reading)*

If G does not satisfy these properties, then *reject*.[2]

Then, *existentially* guess a $t^{c\varepsilon+o(1)}$-bit string that is intended to represent D' which according to assumption (B) may be assumed to have $t^{ec\varepsilon/d+o(1)}$ depth. To check that this guess is correct, *universally* guess an $t^{c\varepsilon/d}$-bit input y to D', and verify (in $t^{c\varepsilon+o(1)}$ time) that $D'(y) = \textsc{CircEval}(y)$ (if not, then *reject*). Now we are assured that the guessed small circuit D' is correct.

The circuit D can be simulated on the input (C, x) with a $t^{c(1-\varepsilon/d)+ec\varepsilon/d+o(1)}$-depth uniform circuit using the small circuit D', along with the graph G and the sequences $s_{v,j}$ to route the outputs of $O(b(n))$ copies of D' (the output of a time block) to the appropriate inputs of other D' copies (the input to another time block). This low-depth simulation itself can be performed in $t^{c(1-\varepsilon/d)+ec\varepsilon/d+o(1)}$ alternating time on a random-access machine, via the standard simulation of uniform size-$t^{O(1)}$ depth-d circuits in alternating time $O(d + \log t)$ (cf. Borodin [Bor77]).

Note that, during the course of the low-depth simulation, it is possible that the guessed sequence $s_{i,j}$ is found to be incorrect for some i and j (e.g., the tape head of a block moves off the right at the end of the block, yet the corresponding index does not increase by 1). In such a case, the simulation rejects. □

Merging the two lower bounds of Theorem 1.2 into one, we obtain a non-linear time lower bound for constructing low-depth circuits solving quantified Boolean formulas:

Corollary 4.1. *For all $\varepsilon > 0$, QBF cannot be solved by circuits of n^d size and n^e depth constructible in n^c time, where $c, d \geq 1$ and $e < 1$ satisfy $c < (1 - e + d)/d$.*

Notice that we must have $d \leq c$, as no circuit of size greater than n^c can be constructed in n^c time.

Proof. The assumption that QBF has such circuits implies that QBF is in $n^{c+o(1)}$ time and that QBF has n^d size, n^e depth circuits. Since CircEval can be reduced in quasilinear time (and polylogarithmic time per bit) to QBF (Theorem 2.1), there is a polylogarithmic depth, quasilinear size circuit C which takes instances of CircEval and outputs equivalent instances of QBF. Composing this circuit with a circuit of n^d size and n^e depth for QBF, we conclude that CircEval has $n^{d+o(1)}$ size, $n^{e+o(1)}$ depth circuits. Hence the lower bound of Theorem 1.2 applies. □

Of course, we believe that *for all* c there is no n^c time algorithm for QBF (i.e., $\mathsf{P} \neq \mathsf{PSPACE}$), and that *for all* d, there are no n^d size, $n^{o(1)}$ depth circuits for QBF (i.e., $\mathsf{PSPACE} \not\subset \mathsf{NC/poly}$).

Let us instantiate some interesting values of c, d, e in Corollary 4.1. When $c = d$ and $e = o(1)$, then it suffices to set $c < \phi$, where $\phi = 1.618\cdots$ is the golden ratio.

[2] In principle, we do not have to guess and check the computation graph G. The oblivious Turing machine simulation (see the Appendix) allows us to efficiently compute the head positions and movements at the beginning and end of each time block, so we can in fact compute all edges of G in time that is quasi-linear in the number of its edges. However, including this guess-and-check of G does not affect the asymptotic running time of the simulation, so we keep it to clarify the presentation.

Reminder of Corollary 1.1 *QBF does not have $O(n^{1.618})$-time uniform circuits of depth $n^{o(1)}$.*

If $d = 1$ and $e = o(1)$, then $c = 2 - \varepsilon$ suffices to obtain a lower bound.

Reminder of Corollary 1.2 *QBF does not have $O(n^{2-\varepsilon})$-time uniform circuits of $n^{1+o(1)}$ size and $n^{o(1)}$ depth, for all $\varepsilon > 0$.*

Suppose $c = 1$ and $e = o(1)$. Then d can be any constant, and we can derive directly from Theorem 1.2 that:

Reminder of Corollary 1.3 *Either QBF cannot be solved in $n^{1+\varepsilon}$ time for some $\varepsilon > 0$, or $\mathsf{P} \not\subset \mathsf{NC}/poly$.*

Finally, any non-trivial depth reduction in circuits for solving QBF results in a non-trivial size lower bound:

Corollary 4.2. *For every $\varepsilon > 0$, there is a $c > 1$ such that QBF cannot be solved by circuits of n^c size and $n^{1-\varepsilon}$ depth, constructible in n^c time.*

Proof. Let $e = 1-\varepsilon$ and $c = d$. To obtain the lower bound from Corollary 4.1, we need $c < (\varepsilon+c)/c$, which holds for all sufficiently small c. □

5 A Nearly-Quadratic Lower Bound

Corollary 1.2 shows that $n^{2-o(1)}$ time is required to produce $n^{1+o(1)}$ size, $n^{o(1)}$ depth circuits solving QBF. But this is a somewhat unnatural statement: in $n^{2-\varepsilon}$ time we could, in principle, generate a circuit of size up to $n^{2-\varepsilon}$. Using a considerably more involved argument, we can in fact prove a $n^{2-\varepsilon}$ time lower bound for generating $n^{2-\varepsilon}$ size NC circuits solving QBF.

To build intuition for our approach, let us start by highlighting a weakness in the argument of Theorem 1.2, which established lower bounds of the form "either QBF is not in n^c time or CircEval doesn't have low-depth circuits of n^d size." Suppose that QBF is in $n^{2-\varepsilon}$-time uniform NC. The fast alternating simulation of determinism in the proof of Theorem 1.2 has two factors that dominate the running time:

(1) the size of the small low-depth circuit D' for computing blocks of the deterministic computation, and

(2) the depth of the circuit for the entire deterministic computation, which is obtained by replacing the block computations with copies of D'.

In the proof of Theorem 1.2, our alternating simulation guessed a description of D' and verified its correctness (this is part (1)), then used D' to reduce the overall depth of a larger circuit computing QBF (this is part (2)). To prove a stronger circuit lower bound, we show how to design a faster alternating simulation of determinism, under the assumption that QBF is in $n^{2-\varepsilon}$-time NC. Recall that the small circuit D' in part (1) is simulating a linear time computation: this linear time computation takes $O(1)$ blocks of length n as input, simulates for $O(n)$ steps, then produces $O(1)$ new blocks of output. Therefore (by assumption) the computation of D' on n-bit inputs also can

be modeled by $n^{2-\varepsilon}$-time uniform NC circuits: there is an algorithm $A_{D'}$ which, on 1^n, prints an NC circuit equivalent to D' on all n-bit instances.

Our key idea is this: rather than guessing an entire description of D', we instead design a fast alternating simulation of $A_{D'}$ which can efficiently compute gate information of D' on demand. This alternating simulation will be *recursive*, in that it divides the computation of $A_{D'}$ into even smaller blocks of length $n^{1-\delta}$ for some δ, and recursively runs a fast alternating simulation of those blocks in order to simulate them faster. This construction allows us to essentially construct the circuit D' in time that is quasi-linear in the *number of inputs* to D' (as opposed to quasi-linear in the *size* of D'), which yields a faster alternating simulation and a larger time lower bound.

Theorem 5.1. *For all $\varepsilon > 0$, QBF does not have $n^{2-\varepsilon}$-time uniform circuits of $n^{2-\varepsilon}$ size and polylog depth.*

Proof. First, note we can already assume $\varepsilon < .4$, from Theorem 1.2. Assume QBF has n^k-time uniform circuits of polylog depth; we want to establish a contradiction when $k = 2 - \varepsilon$. The assumption says that there is an $O(n^k)$ time algorithm G with the property that for all n, $G(1^n)$ outputs a circuit C_n of $z \leq O(n^k)$ size and $\text{poly}(\log n)$ depth, such that $C(F) = 1$ if and only if F is a valid quantified Boolean formula encoded in n bits. Without loss of generality, $G(1^n)$ outputs the circuit C_n encoded as a sequence of tuples of the form

$$\langle i, i_1, i_2, g \rangle,$$

for all $i = 1, \ldots, z$, where the two inputs to gate i in C_n are the outputs of gates i_1 and i_2, and the gate type of i is $g \in \{INPUT, AND, OR, NOT\}$. (If $g = INPUT$, then i_1 and i_2 can be arbitrary; if $g = NOT$ then i_2 can be arbitrary.)

Let $L \in \mathsf{ATIME}[n^{1+\varepsilon/2}]$. As in Theorem 1.2, we will show that our assumption implies a simulation of L in $\mathsf{ATIME}[o(n^{1+\varepsilon/2})]$, a contradiction.

First, using the efficient reduction from arbitrary languages in alternating linear time to QBF instances of $\tilde{O}(n)$ size (Theorem 2.1), the algorithm G can be used to construct an $\tilde{O}(n^{k(1+\varepsilon/2)})$-time algorithm Print-$L$-Ckt with the following specification:

- for all n, Print-L-Ckt(1^n) outputs a circuit C_n of $\tilde{O}(n^{k(1+\varepsilon/2)})$ size and $\text{poly}(\log n)$ depth, such that C_n recognizes L on all n-bit inputs.

An efficient algorithm for the connection language of $\{C_n\}$. Our goal will be to describe a decision algorithm Connect-C running in $\tilde{O}(n)$ alternating time and accepts the *connection language of the circuit family* $\{C_n\}$. That is:

- for all n, Connect-$C(1^n, \langle i, i_1, i_2, g \rangle) = 1$ if and only if the tuple $\langle i, i_1, i_2, g \rangle$ is output by Print-L-Ckt(1^n).

Given an $\tilde{O}(n)$ time algorithm for Connect-C, it is easy to construct an alternating algorithm running in $\tilde{O}(n)$ time for deciding L: as in the proof of Theorem 1.2, we carry out the usual simulation of a $\text{poly}(\log n)$-depth circuit in $\text{poly}(\log n)$ alternating time, but when the simulation

requests gate information for a gate i (the gates that feed into a gate i, and the type of gate i), the relevant tuple $\langle i, i_1, i_2, g\rangle$ can be existentially guessed, and then verified by running Connect-C (in $\tilde{O}(n)$ time) in a separate branch of the computation.[3] This implies that

$$\mathsf{ATIME}[n^{1+\varepsilon/2}] \subseteq \mathsf{ATIME}[n \cdot \mathrm{poly}(\log n)],$$

a contradiction to the alternating time hierarchy.

Now we set ourselves to the task of describing Connect-C. In the following, let $t \leq n^{k(1+\varepsilon/2)} \cdot \mathrm{poly}(\log n)$ be the running time of the algorithm Print-L-Ckt. Without loss of generality, we may assume Print-L-Ckt is oblivious, as this only increases the running time by a log factor. We may also assume that Print-L-Ckt(1^n) waits until the entire circuit description of C_n has been constructed on some internal tape, then spends its last $O(|C_n|)$ steps copying and pasting the description of C_n from its internal tapes to its output tape. (That is, Print-L-Ckt(1^n) only begins outputting a description over its last $O(|C_n|)$ steps.)

Suppose $(1^n, \langle i, i_1, i_2, g\rangle)$ is an input to Connect-C. We will efficiently verify that the tuple $\langle i, i_1, i_2, g\rangle$ is indeed part of the description of C_n, by verifying that this tuple is output by Print-L-Ckt(1^n) at some point. We will do this by simulating the algorithm Print-L-Ckt efficiently with alternations.

The algorithm Connect-C begins by partitioning the computation of Print-L-Ckt(1^n) into n time blocks of ρ steps each, where $\rho = t/n$. The algorithm then constructs (via Footnote 2 in Theorem 1.2) the $O(n)$-edge, n-node computation graph G, where each node of G corresponds to a time block of Print-L-Ckt(1^n), and arcs between nodes encode how the inputs and outputs of blocks relate to each other.

It will be very useful to think of the computation graph G itself as a circuit over a single type of gate, a function

$$\mathit{Block} : \{0,1\}^{O(1)} \times \{0,1\}^b \to \{0,1\}^{b'},$$

where b and b' are $\Theta(\rho)$. *Block* is computed at each node of G: it takes the description of a machine M (in this particular case, the machine is Print-L-Ckt, but it may be different), as well as b bits of input corresponding to a constant number of $O(\rho)$-bit blocks of tape. The *Block* gates in G perform linear time computations: *Block* simulates M for ρ steps on the tape blocks given as input, and outputs $O(\rho)$-bit blocks of new tape content, as well as all tuples $\langle i', i'_1, i'_2, g\rangle$ of the circuit C that written to output during the given time block. Without loss of generality, since tuples are only output at the very end of the computation in a copy-paste operation, we may assume that for each tuple of C there is exactly one time block in which it is output: no tuple is output partially in one time block, then output partially in a later time block, *etc.*

By the assumption that QBF has n^k-time uniform NC circuits, there is another $\tilde{O}(n^k)$-time algorithm Print-Block-Ckt with the specification:

- for all n, Print-Block-Ckt(1^n) outputs a circuit D_n of $\tilde{O}(n^k)$ size and $\mathrm{poly}(\log n)$ depth, such that for all y of length n and all $i = 1, \ldots, O(n)$, $D_n(M, i, y)$ equals the ith output bit of $\mathit{Block}(M, y)$.

[3] Note this sort of simulation will be described in further detail later, when we describe Connect-C itself.

That is, Print-Block-Ckt prints a low-depth circuit computing the *Block* function.[4] We now claim:

Proposition 1. *Let $k = 2 - \varepsilon$. Assuming* $\mathsf{TIME}[n]$ *is in n^k-time uniform* NC*, the connection language of the poly$(\log n)$-depth circuit family $\{D_n\}$ computing Block can be recognized in $\tilde{O}(n)$ alternating time. That is, there is an alternating algorithm Connect-D such that for all n, Connect-$D(1^n, \langle i, i_1, i_2, g\rangle) = 1$ if and only if $\langle i, i_1, i_2, g\rangle$ is a tuple in the description of D_n.*

Provided Proposition 1 is true, we can complete the description of Connect-C so that it is implementable in $\tilde{O}(n)$ time with alternations. In the proof of Theorem 1.2, we guessed and verified a full description of a small circuit D' computing *Block* for a smaller input size, and used this D' to efficiently simulate the entire computation with alternations. Connect-C will instead use the algorithm Connect-D from Proposition 1 to simulate random access to the small circuit D': rather than generate a full description of D', Connect-C guesses the relevant tuple of D', and runs the algorithm Connect-D to verify the guess.

More precisely, given the input tuple $\tau = \langle i, i_1, i_2, g\rangle$, Connect-$C$ existentially guesses:

- an integer $j \in \{1, \ldots, n\}$ corresponding to the time block in which τ is output by Print-L-Ckt, and
- an integer $p \in \{1, \ldots, b'\}$ corresponding to the bit position in the output of time block j where τ is located.

(Recall that, WLOG, each tuple τ is output in exactly one time block.) To check these guesses, Connect-C universally chooses a $z \in \{0, \ldots, |\tau| - 1\}$ and will attempt to verify that the $(p+z)$th bit output during the jth time block of Print-L-Ckt(1^n) equals the zth bit of τ. This will certify that τ is indeed output by Print-L-Ckt.

Now to verify the zth bit of τ, we do a full (alternating) simulation of Print-L-Ckt, going backwards from the end of the jth block. Using Connect-D to guide us, this simulation will be fast. Connect-C executes the following procedure P, which evaluates the circuit D_b (computing the *Block* function) on one ρ-step block:

- Let M be a description of Print-C-Ckt, and set i' to be the index of the output gate of D_b. Set τ' to be the zth bit of τ, the proposed output bit. Set $q = p + z$ and $g = AND$.[5]
- While $g \neq INPUT$,
  - Existentially guess the tuple $\langle i', i'_1, i'_2, g\rangle$ for the current gate i' of $D_b(M, q, \cdot)$
  - Universally guess a bit w.
    1. If $w = 0$, call Connect-$D(1^b, \langle i', i'_1, i'_2, g\rangle)$ to verify the tuple is correct. If so, *accept*.
    2. If $w = 1$, then there are several cases:
       If $g = NOT$ then update the current gate i' to be i'_1, and update τ' to be $\neg\tau'$.

[4] Note that such an algorithm Print-Block-Ckt exists, assuming only that $\mathsf{TIME}[n]$ is contained in $\mathsf{TIME}[n^k]$-uniform NC.

[5] Note this choice for g is arbitrary; it only has to be some value not equal to $INPUT$.

If $g = AND$ and $\tau' = 1$ then universally guess both i'_1 and i'_2 to be the new i' (i.e., verify both inputs are 1).
If $g = AND$ and $\tau' = 0$ then existentially guess one of i'_1 or i'_2 to be the new i' (i.e., verify some input is 0).
If $g = OR$ and $\tau' = 1$ then existentially guess one of i'_1 or i'_2 to be the new i' (i.e., verify some input is 1).
If $g = OR$ and $\tau' = 0$ then universally guess both i'_1 or i'_2 to be the new i' (i.e., verify both inputs are 0).

In other words, to verify the output bit $\tau[z]$, procedure P starts at the output gate of D_b computing the *Block* function, and guesses the connection information for this gate. Then in parallel, it verifies the correctness of the guess in one branch, and continues evaluating the circuit D_b for output $\tau[z]$ in the other branch. As it continues traversing down the circuit from the output gate, it continues to guess connection information, verifying each guess in a separate computation branch. P continues until an input gate of $D_b(M, q, \cdot)$ is reached.

After P completes, the computation graph G is consulted to determine which block $j' < j$ produced the input bit i' to block j, and which output bit q' of block j' is the input bit i' to block j. (This part can be done efficiently by guessing j' and q', then universally both inspecting G to verify j', q' and continuing the simulation assuming correct guesses.) Then i' is updated to be the output gate of D_b again, the current block j is updated to be j', the output bit index q is updated to be q', and the procedure P is started again from the beginning of the While loop, with the current bit τ' and the new q and j.

This entire process is repeated until the 1st block is reached ($j = 1$), in which case the input bit i' refers to one of the original input bits to Print-L-Ckt; then, the bit can be verified directly (it must be a 1, as the input is assumed to be unary).

Let us analyze the running time of the alternating machine Connect-C. The initial processing to generate the computation graph G on n nodes takes $n \cdot \mathrm{poly}(\log n)$ time. There are n total blocks of computation, so the number of times P is executed in any branch of the computation is at most n. As the depth of the circuit D is $\mathrm{poly}(\log n)$, each execution of P takes alternating time

$$\mathrm{poly}(\log n) + \mathrm{time}_D(\rho),$$

where $\mathrm{time}_D(m)$ is the running time of Connect-D on input of length m. Every call to Connect-D happens in a separate branch of the computation, hence the running time is upper bounded by the time of one call to Connect-D plus $\mathrm{poly}(\log n)$ extra time for the depth. Indeed, this is true for the entire computation over all calls to P: every single call to Connect-D is executed in a separate branch of the alternating computation, so the running time $\mathrm{time}_D(\rho)$ contributes only an additive factor to the overall running time.

Therefore we can upper bound the running time of Connect-C as

$$t' \leq n \cdot \mathrm{poly}(\log n) + \mathrm{time}_D(\rho).$$

Recalling that $\rho = t/n$, $t = \tilde{O}(n^{k(1+\varepsilon/2)})$, and $k = 2 - \varepsilon$, we have $k(1 + \varepsilon/2) - 1 = 1 - \varepsilon^2/2$ and

$$t' \leq \tilde{O}(n) + \mathrm{time}_D(n^{1-\varepsilon^2/2} \cdot \mathrm{poly}(\log n)).$$

Assuming Proposition 1 and $\varepsilon > 0$, $\text{time}_D(n^{1-\varepsilon^2/2} \cdot \text{poly}(\log n)) \leq \tilde{O}(n)$, hence $t' \leq \tilde{O}(n)$. We therefore have an $n \cdot \text{poly}(\log n)$ time algorithm for recognizing $L \in \mathsf{ATIME}[n^{1+\varepsilon/2}]$ and our desired contradiction to the alternating time hierarchy.

Efficiently computing a circuit for the *Block* function. Now we prove Proposition 1, which will conclude the proof of the theorem. We want to describe the algorithm Connect-D which will determine in $\tilde{O}(n)$ alternating time whether a given input tuple $\langle i, i_1, i_2, g\rangle$ is part of the description of the circuit D_n, which computes the *Block* function on n-bit inputs. As mentioned above, by assumption we have an $\tilde{O}(n^k)$-time algorithm Print-Block-Ckt that for all n, outputs the circuit D_n when given the input 1^n. The algorithm Connect-D will simulate Print-Block-Ckt efficiently using alternations, analogously to the algorithm Connect-C. The difference is that now we are taking an algorithm which prints a circuit that processes blocks, partitioning that algorithm's computation into smaller blocks, then constructing circuits for those smaller blocks in order to construct the overall circuit. In this case, the call to Connect-D will become a *recursive* call.

Let $t' \leq n^k \cdot \text{poly}(\log n)$ be the running time of Print-Block-Ckt on the input 1^n. The computation Print-Block-Ckt(1^n) is partitioned into n time blocks as was with Print-L-Ckt, but now each block takes only $\tilde{O}(t'/n) \leq \tilde{O}(n^{1-\varepsilon})$ time steps each. The overall circuit D_n printed by this algorithm computes a *Block* computation on n bits, but the algorithm itself is divided up into n time blocks, each of which are *Block* computations on $\tilde{O}(n^{1-\varepsilon})$ bits.

The behavior of Connect-D is totally analogous that of Connect-C, but instead of simulating Print-L-Ckt(1^n) we now simulate the computation Print-Block-Ckt(1^n). We still make calls to Connect-D in order to simulate blocks efficiently, but now the calls are recursive. In more detail, Connect-D has an analogous initial processing of a computation graph G encoding the relations between the n time blocks; this processing runs in $\tilde{O}(n)$ time. For a given input tuple τ of D_n that Connect-D is supposed to verify, it guesses the time block $j \in \{1, \ldots, n\}$ of Print-Block-Ckt(1^n) in which τ is generated, and will make up to j calls to a procedure P (analogous to that of Connect-C) which processes a single time block of $\tilde{O}(n^{1-\varepsilon})$ steps. Each call to P takes $\text{poly}(\log n)$ alternations to evaluate the circuit D_b (where $b \leq \tilde{O}(n^{1-\varepsilon})$), and in each computation branch, the procedure makes at most one *recursive* call to Connect-D on inputs of length b, to verify a guessed tuple for some gate in the circuit D_b. As before, all calls to Connect-D occur in separate computation branches, so each one only contributes an additive factor to the overall running time. Analogously to Connect-C, the running time of Connect-D on any instance $(1^n, \langle i, i_1, i_2, g\rangle)$ can be upper bounded by the recurrence:

$$\text{time}_D(n) \leq \tilde{O}(n) + \text{time}_D(n^{1-\varepsilon} \cdot \text{poly}(\log n)),$$

which solves to $\text{time}_D(n) \leq \tilde{O}(n)$. This proves Proposition 1, and completes the proof of the theorem. □

6 Discussion

In this paper, we have applied some fairly old techniques to prove new lower bounds. Every component we use has been essentially known since the 1980's. Perhaps the major difference between then and now is our perspective. What else might we have missed? For example, could a

clever recursive argument finally show that PSPACE is not contained in P-uniform NC? Could the techniques here be used (along with other ideas) to finally prove that QBF is not in $n^{1.1}$ time?

An interesting consequence of the Size Reduction Lemma is that, to prove a non-uniform size-depth lower bound for CircEval, it suffices to prove one for circuits with succinct descriptions ($o(n)$ bits of non-uniformity). This is very intriguing, because it looks far easier to prove lower bounds against succinctly described circuits, compared to arbitrary circuits. For example, it is not difficult to show that there is a language in $O(n \log^4 n)$ time that does not have linear-size circuits with n-bit descriptions:

Proposition 2. $\mathsf{TIME}[n \log^4 n] \not\subseteq \mathsf{TIME}[O(n \log^2 n)]/n$.

Proof. (Sketch) The proof idea is identical to that of Tourlakis [Tou01], who proved $\mathsf{NTIME}[n^k] \not\subseteq \mathsf{coNTIME}[o(n^k)]/n$. The diagonalizing machine M running in $O(n \log^4 n)$ time simulates the $|x|$th linear time machine $M_{|x|}$ on input x, treating x also as the advice string, and outputs the opposite result. For every input length n and linear time M_n/y with advice string y, $M(y) \neq (M_n/y)(y)$.□

However, if the restriction to n-bit descriptions is removed, it is possible that *every* polynomial-time computable language has linear-size circuits. Therefore this restriction to low-uniform circuits is rather significant, with respect to what we can prove.

One intermediate direction for future work is to try to understand the space of these new lower bound proofs for QBF. For SAT time-space tradeoffs, the known proof methods are now extremely well-understood, in terms of their power and limits [BW12]. The proofs for QBF here have a similar flavor: one component turns alternating time computations into deterministic time computations, and another component simulates deterministic time computations faster using alternations. Can we generalize and rigorously formalize the methods used here, and build an automatic search that can find better lower bounds?

Acknowledgements

We are grateful to the referees for helpful comments which improved the presentation. R.W. was supported in part by a David Morgenthaler II Faculty Fellowship, and NSF Grant CCF-1212372.

References

[All89] E. Allender. P-uniform circuit complexity. *J. ACM* 36(4):912–928, 1989.

[All99] E. Allender. The Permanent Requires Large Uniform Threshold Circuits. *Chicago J. Theor. Comput. Sci.*, 1999.

[AKRRV01] E. Allender, M. Koucký, D. Ronneburger, S. Roy, and V. Vinay. Time-Space Tradeoffs in the Counting Hierarchy. *IEEE Conference on Computational Complexity*, 295–302, 2001.

[AK10] E. Allender and M. Koucký. Amplifying Lower Bounds by Means of Self-Reducibility. *JACM* 57(3), 2010.

[BGHSV05] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Short PCPs verifiable in polylogarithmic time. In *IEEE Conference on Computational Complexity*, 120–134, 2005.

[Bor77] A. Borodin. On Relating Time and Space to Size and Depth. *SIAM J. Comput.* 6(4): 733–744, 1977.

[BW12] S. R. Buss and R. Williams. Limits on Alternation-Trading Proofs for Time-Space Lower Bounds. To appear in *IEEE Conference on Computational Complexity*, 2012.

[CK12] R. Chen and V. Kabanets. Lower Bounds against Weakly Uniform Circuits. In *Proc. of COCOON*, 408–419, 2012.

[Cob66] A. Cobham. The recognition problem for the set of perfect squares. In *IEEE Conference on Switching and Automata Theory*, 78–87, 1966.

[Fis74] M. J. Fischer. Lecture Notes on Network Complexity. Yale Technical Report 1104, June 1974.

[For00] L. Fortnow. Time-Space Tradeoffs for Satisfiability. *J. Comput. Syst. Sci.* 60(2):337–353, 2000.

[FLvMV05] L. Fortnow, R. J. Lipton, D. van Melkebeek, and A. Viglas: Time-space lower bounds for satisfiability. *J. ACM* 52(6):835–865, 2005.

[FL93] L. Fortnow and C. Lund. Interactive proof systems and alternating time-space complexity. *Theoretical Computer Science A*, 113:55–73, 1993.

[GS89] Y. Gurevich and S. Shelah. Nearly-linear time. In *Logic at Botik*, Springer LNCS 363:108–118, 1989.

[HS66] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape Turing machines. *JACM* 13(4):533– 546, 1966.

[HPV77] J. Hopcroft, W. Paul, and L. Valiant. On time versus space. *JACM* 24(2):332–337, 1977.

[IKW02] R. Impagliazzo, V. Kabanets, and A. Wigderson. In Search of an Easy Witness: Exponential Time vs. Probabilistic Polynomial Time. *Journal of Computer and System Sciences* 65(4):672–694, 2002.

[KI04] V. Kabanets and R. Impagliazzo. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *Computational Complexity* 13(1-2):1–46, 2004.

[Kan82] R. Kannan. Circuit-Size Lower Bounds and Non-Reducibility to Sparse Sets. *Information and Control* 55(1-3):40–56, 1982.

[K12] J. Kinne. On TC0 Lower Bounds for the Permanent. In *Proc. of COCOON*, 420–432, 2012.

[KP09] P. Koiran and S. Perifel. A Superpolynomial Lower Bound on the Size of Uniform Nonconstant-depth Threshold Circuits for the Permanent. *IEEE Conference on Computational Complexity*, 35–40, 2009.

[LV03] R. J. Lipton and A. Viglas. Non-Uniform Depth of Polynomial Time and Space Simulations. *International Symposium on Fundamentals of Computation Theory*, 311–320, 2003.

[LW12] R. J. Lipton and R. Williams. Amplifying Circuit Lower Bounds Against Polynomial Time With Applications. In *IEEE Conference on Computational Complexity*, 1–9, 2012.

[vMe06] D. van Melkebeek. A Survey of Lower Bounds for Satisfiability and Related Problems. *Foundations and Trends in Theoretical Computer Science* 2(3):197–303, 2006.

[NRS95] A. V. Naik, K. W. Regan, and D. Sivakumar. On Quasilinear-Time Complexity Theory. *Theor. Comput. Sci.* 148(2):325–349, 1995.

[PPR80] W. J. Paul, E. J. Prauss, and R. Reischuk. On Alternation. *Acta Informatica* 14:243–255, 1980.

[PR80] W. J. Paul and R. Reischuk. On Alternation II: A Graph Theoretic Approach to Determinism Versus Nondeterminism. *Acta Informatica* 14:391–403, 1980.

[Pip77] N. Pippenger. Fast simulation of combinational logic circuits by machines without random-access storage. In *Proc. 15th Allerton Conference on Communication, Control, and Computing*, 25–33, 1977.

[PF79] N. Pippenger and M. J. Fischer. Relations Among Complexity Measures. *JACM* 26(2):361–381, 1979.

[Sch78] C. P. Schnorr. Satisfiability is quasilinear complete in NQL. *JACM* 25:136–145, 1978.

[Tou01] I. Tourlakis: Time-Space Tradeoffs for SAT on Nonuniform Machines. *J. Comput. Syst. Sci.* 63(2):268–287, 2001.

[Will08] R. Williams. Non-Linear Time Lower Bound for (Succinct) Quantified Boolean Formulas. *Electronic Colloquium on Computational Complexity* (ECCC), TR08-076, 2008.

[Wil11] R. Williams. Non-Uniform ACC Circuit Lower Bounds. *IEEE Conference on Computational Complexity*, 115–125, 2011.

A Polylog-time uniformity of Pippenger-Fischer

Here we give a few more details about why the tape head positions of the Pippenger-Fischer circuit construction (for multitape Turing machines running in time $n \cdot \mathrm{poly}(\log n)$) can be computed in $\mathrm{poly}(\log n)$ time, given the index of a timestep $t = 1, \ldots, n \cdot \mathrm{poly}(\log n)$. Without loss of generality, suppose the running time of the machine M is a power of two, 2^k. The first tape is for copying and pasting blocks of tape cells; the second tape holds the configurations of all tapes (their cell content and "virtual" head positions) on its tracks. Also on the second tape, there is a specially marked cell called the "home" cell. At the beginning, all "virtual" tape heads are positioned on the home cell.

First, the simulation of one step of M is done by direct simulation, moving the head one cell to the left (marking the tape for those tape heads that are supposed to move to the left) then

moving two steps to the right (for those tape heads that move to the right), then moving left once to recenter the tape head of the simulation at the home cell. (Note we are implicitly assuming that the virtual tape heads are within one cell of the home cell: in general, the invariant is that at the beginning of a 2^q-step simulation, all virtual tape heads are within 2^q cells of the home cell.)

Then a simulation of 2 steps is performed, followed by 4 steps, 8 steps, and so on, until 2^k steps are reached. The simulation of 2^q steps of M (for $q \geq 1$) takes two recursive calls to simulate 2^{q-1} steps, with two $O(2^q)$-time phases of "shifting" before the calls, in which the tape heads move obliviously. A shifting phase does a cyclic shift of the $2^{q+1}+1$ cells of the tape that are within 2^q cells of the current tape head position, either to the left or to the right, for 2^{q-1} cells (depending on the content and the state). (Recall the purpose of this shift is to bring the "virtual" heads to within 2^{q-1} cells of the home cell, to facilitate the recursive calls.) This involves sweeping the tape heads 2^q cells to the left of the current head position and 2^q cells to the right of this position, for a small constant number of times. After the second recursive call, two "cleanup" phases occur, where the cyclic shifts from the shifting phases are undone. The head movements here are analogous; we note that the two shifting phases need to record a few bits for the later cleanup phases.

To produce the head positions for a given timestep t, we only have to calculate where step t lies within the above procedure. The number of steps in the simulation of 2^q steps of M is precisely $c \cdot 2^{q-1}$ (the cost of the shifting phases and cleanup phases) plus twice the number of steps used to simulate for 2^{q-1} steps, for a small integer c. Hence each simulation of 2^q steps of M takes precisely $d \cdot q2^q$ steps, for a fixed integer d. This 2^q-step simulation is run in phases, for increasing $q = 0, 1, 2, \ldots$, until halting is reached. To determine which phase q that step t refers to, we only need to perform $\mathrm{poly}(\log n)$ arithmetic operations on $O(\log n)$-size integers. (For example, we could compute $t := t - d \cdot i2^i$ for increasing i, until $t < 0$. This yields the desired q as well as the step number s within the simulation of 2^q steps.) Once the phase q is known, determining where t lies within the simulation of 2^q steps again only requires $\mathrm{poly}(\log n)$ arithmetic operations on $O(\log n)$-size integers. (If $s < e \cdot 2^{q-1}$ for a certain integer e, then we are in the first shifting phase; if $s \in [e \cdot 2^{q-1}, e \cdot 2^{q-1} + d \cdot (q-1)2^{q-1}]$ we are in the first simulation of 2^{q-1} steps; if $s \in [e \cdot 2^{q-1} + d \cdot (q-1)2^{q-1} + 1, e \cdot 2^{q-1} + d \cdot (q-1)2^{q-1} + e \cdot 2^{q-1}]$ we are in the second shifting phase, *etc.* The point is that all of these checks can be done with a little arithmetic.)

It is worth noting that even stronger uniformity conditions are known to hold for the Pippenger-Fischer simulation. Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan [BGHSV05] have observed (for the purpose of providing efficiently described PCPs) that an $O(\log n)$-space uniform circuit of $\mathrm{poly}(n)$ size and $O(\log n)$ depth can implement the above reduction for simulating 2^n steps, by embedding the oblivious simulation of M into a computation on a butterfly graph (or de Bruijn graph) and using the recursive structure of butterfly graphs to ensure an easily-described circuit.