Tighter MA/1 Circuit Lower Bounds from Verifier Efficient PCPs for PSPACE

Joshua Cook
Department of Computer Science, University of Texas Austin, TX, USA

Dana Moshkovitz
Department of Computer Science, University of Texas Austin, TX, USA

Abstract

We prove that for some constant $a > 1$, for all $k \leq a$,

$$\mathbf{MATIME}[n^{k+o(1)}]/1 \not\subset \mathbf{SIZE}[O(n^k)],$$

for some specific $o(1)$ function. This is a super linear polynomial circuit lower bound.

Previously, Santhanam [29] showed that there exists a constant $c > 1$ such that for all $k > 1$:

$$\mathbf{MATIME}[n^{ck}]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

Inherently to Santhanam's proof, c is a large constant and there is no upper bound on c. Using ideas from Murray and Williams [26], one can show for all $k > 1$:

$$\mathbf{MATIME}[n^{10k^2}]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

To prove this result, we construct the first **PCP** for **SPACE**$[n]$ with quasi-linear verifier time: our **PCP** has a $\tilde{O}(n)$ time verifier, $\tilde{O}(n)$ space prover, $O(\log(n))$ queries, and polynomial alphabet size. Prior to this work, **PCP**s for **SPACE**$[O(n)]$ had verifiers that run in $\Omega(n^2)$ time. This **PCP** also proves that **NE** has **MIP** verifiers which run in time $\tilde{O}(n)$.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Circuit complexity; Theory of computation → Interactive proof systems

Keywords and phrases MA, PCP, Circuit Complexity

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2023.55

Category RANDOM

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2022/014/`

Funding This material is based upon work supported by the National Science Foundation under grant number 1705028.

1 Introduction

Some of the most fundamental problems in complexity theory are proving circuit lower bounds for uniform complexity classes. One such conjecture is that **NP** does not have polynomial size circuits, which is a strong version of $\mathbf{P} \neq \mathbf{NP}$. Very little is known on such lower bounds. In particular, there are no known proofs that **NEXP** does not have polynomial sized circuits! However, there are some closely related results that could be loosely seen as relaxations.

One can strengthen **NP** slightly by giving the non-deterministic algorithm access to randomness, as well as an extra bit of trusted advice. This gives the complexity class **MA**/1. We can weaken polynomial sized circuits to circuits of fixed polynomial size: **SIZE**$[n^k]$ for constant k.

Santhanam [29] proved that for any constant k, $\mathbf{MA}/1 \not\subseteq \mathbf{SIZE}[n^k]$. The $\mathbf{MA}/1$ algorithm runs in time n^{ck} for a large $c > 1$. In fact, inherently to Santhanam's proof, there is no upper bound on c (We will explain why when we describe Santhanam's proof in Section 1.2.1). One can use ideas from Murray and Williams [26] to get, for some explicit c with $2 < c < 10$, the result $\mathbf{MATIME}[n^{ck^2}]/1 \not\subset \mathbf{SIZE}[n^k]$.

The goal of this paper is to prove a fine grained separation of $\mathbf{MA}/1$ from fixed polynomial size circuits, namely,

$$\mathbf{MATIME}[n^{k+o(1)}]/1 \not\subset \mathbf{SIZE}[n^k].$$

We believe that the gold standard for separations should be fine grained separations. Fine grained separations are necessary for key results in complexity theory, e.g., Williams' program (See, e.g., [33]) and optimal derandomization [14].

Some fine grained separations are known, namely, hierarchy theorems that show that giving algorithms more time allows them to solve more problems [18, 11]. Hierarchy theorems are known for many complexity classes. While no hierarchy theorems are known for $\mathbf{MA}$, they are known for $\mathbf{MA}/1$. Fortnow, Santhanam, and Trevisan showed that $\mathbf{MA}$ with a small amount of advice can solve more problems when given more time [16]. Van Melkebeek and Pervyshev showed that for any $1 < b < d$, $\mathbf{MATIME}[n^b]/1 \subsetneq \mathbf{MATIME}[n^d]/1$ [32]. In particular, they imply that even $\mathbf{MATIME}[n^{2k}]/1$ is much larger than $\mathbf{MATIME}[n^{k+o(1)}]/1$.

1.1 Results

In this work, we give a fine grained separation for $\mathbf{MA}/1$ and $\mathbf{SIZE}[n^k]$. We show that for at least some $k > 1$, there is an $\mathbf{MA}$ protocol with one bit of advice whose verifier has time almost n^k such that any circuit solving the same problem also requires size n^k. Formally:

▶ **Theorem 1** (Fine Grained **MA** Lower Bound). *There exists a constant $a > 1$, such that for all $k < a$, for some $f(n) = o(1)$,*

$$\mathbf{MATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

We stress that we give **super linear** polynomial lower bounds. Our result holds for some k *strictly* greater than 1, even though we don't know which k. This result removes the large polynomial factor in the gap between the $\mathbf{MA}/1$ time and the circuit size in Santhanam's result. It may be the case that a is small, like $a = 1.0001$. But in that case, we get the following result for all k:

▶ **Theorem 2** (**MA** Lower Bound for Small a). *If the a from Theorem 1 is finite, then for all $k > 0$, for some $f(n) = o(1)$,*

$$\mathbf{MATIME}[O(n^{ak+f(n)})]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

This gives us a win-win scenario: if a is large, we get a strong result for a large range of k, but if a is small we get a similar result for all k.

When we describe our proof we will explain why we only get separations for $k < a$ for an (unknown) $a > 1$ and not for all $k > 1$. For now we would like to stress that: (1) under plausible complexity assumptions the upper bound a is in fact super-constant in n; (2) even the case of a constant $a > 1$ as promised in our theorem is highly interesting, since it is unknown how to prove that $\mathbf{NP} \not\subseteq \mathbf{SIZE}[n^k]$ for *any* $k > 1$.

Santhanam's original proof uses an interactive protocol for **PSPACE**. To prove our circuit lower bound, we replace the interactive protocol with a new, more efficient **PCP**.

To get our fine grained results, we need a **PCP** for space $S = O(n)$ and time $T = 2^{O(n)}$ algorithms, where the verifier simultaneously has $\tilde{O}(n)$ time and $\mathbf{poly}(\log(n))$ many queries. Further, the **PCP** needs a prover that can compute any bit of the proof in $\tilde{O}(n)$ space. Notably, we do not need any bounds on the proof length.

The **PCP** given by Babai, Fortnow, and Lund in their proof that **MIP** = **NEXP** [4] required $\Omega(\log(T))$ queries, while we want $O(\log(\log(T)))$ queries.

Holmgren and Rothblum in their work on delegated computation [20] improved on the BFL **PCP** in several ways that can[1] be used to give a **PCP** with verifier time $\tilde{O}(n+\log(T))$. Unfortunately, it still requires $\Omega(\log(T))$ queries.

Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan [5] gave a **PCP** that uses a constant number of queries, but has verifier time $\mathbf{poly}(\log(T))$, while we need $\tilde{O}(n+\log(T))$ verifier time. Similar results were given by subsequent work [23, 8, 6].

The small space requirement for the prover is achieved by Holmgren and Rothblum [20]. In some **PCP**s, like the **PCP** in Ben-Sasson, Chiesa, Genkin, and Tromer's work on the concrete efficiency of **PCP**s [6], the prover requires space $\Omega(T)$. In contrast, our result needs prover space $\tilde{O}(S+n)$.

A sufficiently efficient **PCP** was not known, so we construct a new **PCP**.

▶ **Theorem 3** (Verifier Efficient **PCP**). *Let $S, T = \Omega(n)$ be functions, and L be any language computed by a simultaneous time T and space S algorithm. Let $\delta \in (0, 1/2)$ be a constant. Then there is a* **PCP** *for L with:*

1. *Verifier time $\tilde{O}(n+\log(T))$.*
2. *Query time $\tilde{O}(\log(T))$.*
3. *$O(\log(n)+\log(\log(T)))$ queries.*
4. *Alphabet Σ with $\log(|\Sigma|) = O(\log(\log(T)))$.*
5. *Log of proof length $\tilde{O}(\log(T))$.*
6. *Prover space $\tilde{O}(S)$.*
7. *Perfect completeness and soundness δ.*

We believe we can achieve a similar verifier time, query time and prover space while also achieving constant number of queries and alphabet size. We do not need these improvements for our main result, so we only prove this simpler result.

Only our prover requires the space bound for its efficient computation. If we remove this space limitation, we get a similar **PCP** for nondeterministic algorithms.

▶ **Theorem 4** (Verifier Efficient **PCP** for Nondeterministic Algorithms). *Let $T = \Omega(n)$, $\delta \in (0, 1/2)$ be a constant, and $L \in \mathbf{NTIME}[T]$. Then there is a* **PCP** *for L with:*

1. *Verifier time $\tilde{O}(n+\log(T))$.*
2. *Query time $\tilde{O}(\log(T))$.*
3. *$O(\log(n)+\log(\log(T)))$ queries.*
4. *Alphabet Σ with $\log(|\Sigma|) = O(\log(\log(T)))$.*
5. *Log of proof length $\tilde{O}(\log(T))$.*
6. *Perfect completeness and soundness δ.*

An immediate corollary of Theorem 4 is a more fine grained equivalence between **MIP** and **NEXP**.

[1] The **PCP** constructed by Holmgren and Rothblum was built to have no signalling soundness and has many steps that take longer than $\tilde{O}(\log(T))$ time to compute. Still, the basic elements of of their **PCP** needed for a standard **PCP** are computable in $\tilde{O}(n+\log(T))$ time.

▶ **Corollary 5** (Fine Grained Equivalence of **MIP** = **NEXP**). *For any time constructable function $p(n) = \Omega(n)$, language $L \in \mathbf{NTIME}[2^{\tilde{O}(p(n))}]$ if and only if there is a two prover, one round* **MIP** *protocol for L whose verifier runs in time $\tilde{O}(p(n))$.*

Note this equivalence implies a hierarchy theorem for **MIP** since there are hierarchy theorems for **NTIME** [11, 30, 34, 15].

A special case is **MIP** protocols for **NE**.

▶ **Corollary 6** (**NE** Has Quasi-linear Time Verifiers). *For any language $L \in \mathbf{NE}$, there is a two prover, one round* **MIP** *protocol for L whose verifier runs in time $\tilde{O}(n)$.*

Note this verifier time is nearly optimal since the verifier requires linear time to read its entire input.

All previous **PCP**s fail to achieve such an efficient **MIP** verifier. If the original **PCP** makes $\Omega(n)$ queries of size $\Omega(n)$, then it takes $\Omega(n^2)$ time to send the queries even if we allow more provers. And all previous **PCP**s with fewer queries require verifier time $\Omega(n^2)$ to either verify the response or compute the queries.

1.2 Proof Idea

1.2.1 MA Lower Bounds Using PCP

We first review Santhanam's original proof.

Santhanam's original result uses the fact that if **PSPACE** $\subset$ **P/poly**, then **PSPACE** = **MA**. This follows from the famous result that **IP** = **PSPACE** [31, 22]. The idea is that if **PSPACE** $\subset$ **P/poly**, then an **MA** protocol can guess a circuit computing any problem in **PSPACE**. The prover in the interactive protocol for **PSPACE** is also computable in **PSPACE**. So to solve any **PSPACE** problem in **MA**, the **MA** protocol first guesses the circuit for a prover, then simulates the verifier using the circuit we guessed as the prover.

Using this, Santhanam's original proof then considered two cases: either **PSPACE** $\subset$ **P/poly**, or **PSPACE** $\not\subset$ **P/poly**.

If **PSPACE** $\subset$ **P/poly**, then we already know **PSPACE** = **MA**. Now we just need a problem not computable by a size n^k circuit. But there is a straightforward algorithm that exhaustively finds a circuit of size larger than n^k that computes a function that cannot be computed by a smaller circuit. In fact, such an algorithm only requires space $\tilde{O}(n^k)$. So **PSPACE** $\not\subset$ **SIZE**$[n^k]$. In this case, **PSPACE** = **MA**, so **MA** $\not\subset$ **SIZE**$[n^k]$.

If **PSPACE** $\not\subset$ **P/poly**, then we know a hard problem that is not in **SIZE**$[n^k]$, namely any **PSPACE** complete problem. Let us take a **PSPACE** complete, downward self reducible language, Y. Now Y may be too hard for **MA** to solve, but if we give it enough padding, eventually the padded version of Y will be computable by size n^k circuits. But for this amount of padding, **MA** can pull the same trick it does in the **PSPACE** $\subset$ **P/poly** case. Namely, guess a circuit for Y and then simulate the **IP** protocol for Y. For some **PSPACE** complete Y, the language itself is its proof and this works. The trick is to use just the right amount of padding so it requires circuits of at least size n^k, but not much larger. Santhanam uses the single bit of advice in a clever way to figure out when there is just the right amount of padding.

In either case, the time of this protocol is roughly the time of the verifier in the **IP** protocol, plus the size of the prover circuit times the number of times the prover is queried.

There are two reasons the **MA** protocol could take polynomially more time than the size of the circuits it wants to compute in the case **PSPACE** $\subset$ **P/poly**. One is that the **IP** from the original Santhanam result has polynomial verifier time and a polynomial time interaction with the prover, making the verifier in the **MA**/1 protocol take polynomially longer than

the circuit complexity of the problem being solved. By using a **PCP**, we get better results. The other is that the prover circuit complexity could be large, depending on the circuit size required for **PSPACE** (could be any polynomial when **PSPACE** $\subset$ **P/poly**). This is the reason there is no upper bound on the polynomial run time of the **MA**/1 protocol in Santhanam's proof. To avoid this issue we consider a finer case analysis.

We break the problem into three cases. For some **SPACE**$[O(n)]$ complete language, X, we have one[2] of the following:

1. $X \notin$ **P/poly**.
2. $X \in$ **SIZE**$[n^{1+o(1)}]$.
3. $X \in$ **SIZE**$[n^{a+o(1)}] \setminus$ **SIZE**$[n^{a-o(1)}]$ for some $a > 1$.

The original proof only used the two cases $X \notin$ **P/poly** and $X \in$ **P/poly**. The case where $X \notin$ **P/poly** is completely unchanged. Note that this is the plausible case, and here there is no constant upper bound a on k.

If $X \in$ **P/poly**, we use our efficient **PCP**, Theorem 3, instead of the **IP** Santhanam uses. With this substitution, the case where $X \in$ **SIZE**$[n^{1+o(1)}]$ is almost unchanged from the original proof. By separating this into it's own case, we get tight bounds for all k in this case.

If $X \in$ **SIZE**$[n^{a+o(1)}] \setminus$ **SIZE**$[n^{a-o(1)}]$ for some $a > 1$, then we use the same padding technique we use if $X \notin$ **P/poly**, just using our new **PCP**. In this case, we can only do this if for some $k < a$, we are trying to show **MATIME**$[n^{k+o(1)}]/1 \not\subset$ **SIZE**$[n^{k-o(1)}]$. This is the case where a is finite, but in this case, we can use Santhanam's argument using our **PCP** to get Theorem 2.

To see why $k > a$ poses a difficulty, suppose **SPACE**$[O(n)] \not\subseteq$ **SIZE**$[o(n^2)]$, but **SPACE**$[O(n^2)] \subseteq$ **SIZE**$[O(n^2)]$. Then to get a language requiring size n^3 circuits, we need to use a space n^3 algorithm. But the prover for a space n^3 language is a language running on an input with length n^3, and using space linear in its input length. Thus we may need a size $(n^3)^2 = n^6$ circuit for our prover. So the verifier takes time at least n^6 to even read the prover circuit, thus can't run in time n^3. See Item 2 in our open problems for further explanation.

▶ Remark 7. We note our verifier in Theorem 1 is a RAM machine, *not* a standard Turing Machine. This is because we know how to efficiently simulate a circuit on a RAM machine, but not on a standard Turing Machine.

1.2.2 Verifier Efficient PCP

Now we explain the **PCP** we actually use in the **MA** protocol. We start with a **PCP** similar to [20] and [4] that we refer to as our base **PCP**. This **PCP** has a verifier that runs in time $\tilde{O}(n + \log(T))$ and uses $O(\log(T))$ queries. To reduce the number of queries, we use **PCP** composition [3, 7, 13, 25, 12].

To perform **PCP** composition, we need a robust **PCP**. Loosely, a robust **PCP** is a **PCP** so that when $x \notin L$, for any proof, most sets of queries to that proof return not only a rejected response, but a response that is far from any accepted response. To make our base **PCP** robust, we use the aggregation through curves technique [2]. Now we briefly explain how to use aggregation through curves to convert our base **PCP** into a robust **PCP**.

An honest proof for our base **PCP** is a single low degree polynomial. Suppose our base **PCP** has q queries. To make our **PCP** robust, we first choose the randomness for the base **PCP**, and another random point in the **PCP** proof. Then we find the degree q curve that

[2] This is a trichotomy in an asymptotic sense: for every constant a, either $X \in$ **SIZE**$[O(n^a)]$ or it is not. See Section 3.5 for details.

goes through all these points. Then we check if the proof, restricted to this curve, is a low degree polynomial, and whether the base **PCP** would have accepted on this input. Since a low degree polynomial is an error correcting code, this gives robustness.

One concern one might have with this robust **PCP** is that it actually requires $\Omega(\log(T)^2)$ queries. We don't need to actually calculate all of these query locations. Since we reduce the actual number of queries with **PCP** composition, we only need to be able to calculate any individual query location quickly. To find these query locations requires us to compute a point on the degree q curve going through each of our q points our base **PCP** queries plus a random point. In our base **PCP**, $q = O(\log(T))$ and our proof has dimension $O(\log(T))$. So the naive way to compute this curve is to calculate each coordinate independently, which would take time $\tilde{O}(\log(T)^2)$.

To efficiently compute low degree curves through points, or to extrapolate a function going through those points, we introduce the concept of time extrapolatable functions.

▶ **Definition 8** (Extrapolatability). *For any $n, q, t > 0$, and field $\mathbb{F}$, we call $Q : [q] \to \mathbb{F}^n$ "t extrapolatable" (or time t extrapolatable) if there is a time t algorithm taking any $v \in \mathbb{F}^q$, that outputs*

$$\sum_{i \in [q]} v_i Q(i).$$

Equivalently, if we think of Q as outputting the columns of a matrix, then we say Q is time t extrapolatable if one can multiply a vector with it in time t. An important property of extrapolatable functions is that an extrapolation of an extrapolatable function can be computed efficiently. This is where it gets its name.

Our base **PCP** is just a sum check and a few point checks. Each of these are time $\tilde{O}(\log(T))$ extrapolatable. Our robust **PCP** only queries locations easily computable given the extrapolation of our base **PCP** query locations. Extrapolations of extrapolatable functions are easy to compute, so we can easily compute the query locations of the robust **PCP**.

We also introduce the concept an extrapolatable **PCP** (**ePCP**) as one where an honest proof is a low degree polynomial, and the query locations after fixing a choice of randomness are extrapolatable. We show that any **ePCP** can be extended into a robust **PCP** where the query locations of that robust **PCP** can be computed efficiently.

1.3 Generalization And Sharpness

We actually prove a stronger result than Theorem 1 that is sharp. First, our **MA** protocol is input oblivious: the message from Merlin is just a program for computing a **PSPACE** complete language and doesn't depend on the specific input, just its length. Second, the hardness is against the model used in Merlin's message. We used circuits, but we can describe a randomized algorithm directly to save some polynomial factors.

We define input oblivious Merlin-Arthur time, **OMATIME**, the same way as Fortnow, Santhanam, and Williams [17]. Input oblivious Merlin-Arthur are languages solvable with untrusted advice, where the advice only depends on the input length. In our case, Merlin gets to send a long, untrusted message for every input length, and Arthur also gets a single bit of trusted advice. Note that Santhanam's original proof implicitly also uses input oblivious **MA**.

The main property of circuits we use is that a randomized algorithm can efficiently simulate it. We can instead use $\mathbf{BPTIME}[n^k]/n^k$, that is, randomized algorithms running in time n^k with description length n^k. This uses the same model of computation as our verifier, allowing it to more efficiently simulate **OMATIME**.

Using **OMATIME** instead of **MATIME** and **BPTIME** instead of **SIZE**, we can follow the same proof as our main result to show:

▶ **Theorem 9 (OMATIME** Lower Bound Against **BPTIME).** *There exists constant $a > 1$, such that for all $k < a$, for some $f(n) = o(1)$,*

$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPTIME}[O(n^k)]/O(n^k).$$

This result is tight in the sense that for any function $f(n)$, we have

$$\mathbf{OMATIME}[f(n)]/1 \subseteq \mathbf{BPTIME}[O(f(n))]/(f(n)+1).$$

To get stronger results, we need to use nondeterminism that depends on the input. So one could say our result is less about the power of nondeterminism, and more about the power of trusted versus untrusted advice. Specifically: trusting advice doesn't always buy (much) time in the randomized setting, as long as we have *some* trusted advice.

Let us briefly outline what would need to change in our main proof to prove Theorem 9, and justify why those changes would work.

First we need to make a class of randomized programs that act more like circuits. Consider the class of programs, $\mathcal{C}$, that contain randomized algorithms that work only a specific input length. For any $C \in \mathcal{C}$, we say $C(x)$ is the random variable that simulates C on input x for time $|C|$, and outputs what C does if C terminates in time $|C|$, and outputs 0 otherwise. See that $\mathcal{C}$ behaves like circuits in the following important ways:

1. Given a program $C \in \mathcal{C}$, a randomized algorithm can calculate the random variable $C(x)$ in time $O(|C|)$.
2. For any function $f(n)$ and language $L \in \mathbf{BPTIME}[f(n)]/f(n)$, for every n, there is a $C_n \in \mathcal{C}$ such that $|C_n| = O(f(n))$ and with high probability $C_n(x) = 1_{x \in L}$.

A few notes on using $\mathcal{C}$ in our proof, as opposed to circuits.

1. First, see that $\mathbf{SPACE}[O(n^k)] \not\subseteq \mathbf{BPTIME}[o(n^k)]/o(n^k)$. This follows using the same exhaustive search type algorithm used for $\mathbf{SIZE}[o(n^k)]$.
   For any polynomial n^k, there is a deterministic program, A, with length $O(n^k)$ running in time $O(n^k)$, but is not computable with high probability by any program $C \in \mathcal{C}$ with length $o(n^k)$. This follows from a simple counting argument: length n^k deterministic programs contain more than $2^{\alpha n^k}$ functions for some constant α (just use some lookup table), while there are only $2^{\alpha n^k}$ length αn^k programs.
   Such an A can still be found by exhaustive search in space $O(n^k)$, since given $C \in \mathcal{C}$, we can space efficiently check every choice of randomness and calculate majority. This gives us that

   $$\mathbf{SPACE}[O(n^k)] \not\subseteq \mathbf{BPTIME}[o(n^k)]/o(n^k).$$

2. Given that $L \in \mathbf{BPTIME}[f(n)]/f(n)$, then for any n, there is some advice (notably, a program $C_n \in \mathcal{C}$ with size $|C_n| \leq f(n)$) such that a randomized algorithm given that advice can compute whether $x \in L$ with probability $1 - \epsilon$ in time $O(f(n)\log(\frac{1}{\epsilon}))$.
   This allows our verifier to efficiently compute a $\mathbf{SPACE}[O(n)]$ complete problem, L, in time nearly $f(n)$ if $L \in \mathbf{BPTIME}[f(n)]/f(n)$, given correct advice.
3. We also note that for some programs $C \in \mathcal{C}$, for some inputs x, our program C evaluated on x may answer one or zero with very close to half probability. That is, the syntax of $\mathcal{C}$ does not only give bounded error randomized algorithms, just a randomized algorithm. This is not an issue, because in the completeness case, there will be a program C that does have bounded error the prover should provide. And in the soundness case, the soundness of our **PCP** holds against any multi-prover strategy, even a randomized strategy. So no program provided will convince the verifier with high probability.

Finally, see that in all cases of our proof, Arthur only asks Merlin for a program computing some **SPACE**$[O(n)]$ complete problem. This advice does not depend on the specific input, only on the input size.

2 Preliminaries

We assume some familiarity with basic complexity theory. See Arora and Barak's book for background [1]. In this paper, by algorithm, we mean algorithm on a RAM machine, and by circuit, we mean a fan in 2 circuit with unbounded depth. A randomized algorithm is a deterministic algorithm with an extra input for randomness. We will assume in this paper that all time and space bounds for algorithms are sufficiently easily computable.

Now recall that **MA** is the complexity class of problems with polynomial sized certificates that can be verified with bounded error by a randomized, polynomial time algorithm. This is like **NP** with a randomized verifier. Then we define **MATIME** in an analogous way to **NTIME**. Our results have perfect completeness, so we only define **MATIME** with perfect completeness.

▶ **Definition 10 (MATIME/1).** *For any function $f : \mathbb{N} \to \mathbb{N}$, define* **MATIME**$[f(n)]/1$ *as the set of languages, L, such that there is a function $b : \mathbb{N} \to \{0,1\}$ and a time $f(n)$ randomized algorithm M taking four inputs, an input x, a random input r, a witness w, and an advice bit such that*

Completeness *If $x \in L$ and $n = |x|$, then there exists w with $|w| \leq f(n)$ such that*

$$\Pr_r[M(x, r, w, b(n)) = 1] = 1.$$

Soundness *If $x \notin L$ and $n = |x|$, then for every w,*

$$\Pr_r[M(x, r, w, b(n)) = 1] < 1/2.$$

We let **SIZE** denote the class of languages with circuits of a given size.

▶ **Definition 11 (SIZE).** *For any function $f : \mathbb{N} \to \mathbb{N}$,* **SIZE**$[f(n)]$ *is the class of languages, L, where for each input length n, there is a circuit of size $f(n)$ with n inputs computing L for inputs of length n.*

Further, **SIZE**$[O(f(n))]$ *is the class of languages, L, such that for some $g(n) = O(f(n))$, we have $L \in$* **SIZE**$[g(n)]$*. Similarly for* **SIZE**$[o(f(n))]$*.*

In this paper, we will focus on time and space efficient, non-adaptive **PCP**s with perfect completeness. Because we need to pay close attention to the amount of time it takes to make a single query to the proof, we separate the algorithm for producing queries, Q, from the algorithm for verifying the response, V. We also separate the function that gives all the query locations for a choice of randomness, I, from the algorithm that gives a single one of those query locations, Q.

So at a high level, a **PCP** protocol does the following:

1. Chooses a common random string, r.
2. Runs query function Q with randomness r for $q(n)$ many times to get all query locations, I.
3. Looks up all query locations, I, into a provided proof, π, to get proof window π_I.
4. Runs verifier V with randomness r and proof window π_I and outputs if V accepts.

Now we formally define a **PCP**.

▶ **Definition 12 (PCP).** *We say that a language L has a non-adaptive* **PCP***, A, with perfect completeness if there exists verifier V, prover P, index function I, and query function Q, such that, for some alphabet Σ, $\delta \in [0,1]$, and functions $r, l, q : \mathbb{N} \to \mathbb{N}$:*

1. *I takes 2 inputs, an input of length n and randomness of length $r(n)$, and outputs an element of $[l(n)]^{q(n)}$. That is, I outputs $q(n)$ indexes in a length $l(n)$ string,*
2. *Q is an algorithm with three inputs, an input x of length n, randomness r of length $r(n)$, and an index $i \in [q(n)]$ and outputs an element of $[l(n)]$ such that $Q(x, r, i) = I(x, r)_i$.*
3. *V is an algorithm with three inputs, an input of length n, randomness of length $r(n)$, and $q(n)$ symbols from Σ, and outputs either accept or reject.*
4. *P is an algorithm that takes two inputs, an input of length n, and an index $i \in [l(n)]$, and outputs a symbol from Σ.*

Completeness *If $x \in L$ and $n = |x|$, then there exists $\pi^x \in \Sigma^{l(n)}$ such that*

$$\Pr_r[V(x, r, \pi^x_{I(x,r)}) = 1] = 1,$$

and for every $i \in [l(n)]$, $P(x, i) = \pi^x_i$.

Soundness *If $x \notin L$ then for every π',*

$$\Pr_r[V(x, r, \pi'_{I(x,r)}) = 1] \leq \delta.$$

Then we also say:

1. *A has proof length $l(n)$.*
2. *A has alphabet Σ.*
3. *A has soundness δ.*
4. *A uses $q(n)$ queries.*
5. *A uses $r(n)$ bits of randomness.*
6. *If V runs in time $t(n)$, A has verifier time $t(n)$.*
7. *If V runs in space $s(n)$, A has verifier space $s(n)$.*
8. *If P runs in space $s'(n)$, A has prover space $s'(n)$.*
9. *If Q is computable in time $t'(n)$, A has query time $t'(n)$.*

3 Efficient PCP To Fine Grained Lower Bounds

We only give a detailed sketch here. The full version of our paper is available at [9].

Our analysis depends on the circuit complexity of some **PSPACE** complete problem. So we start by choosing a **SPACE**$[O(n)]$ complete problem. We use a version of `SPACE TMSAT` (on page 83 of [1]).

▶ **Definition 13** (Specific Problem). *`SPACE TMSAT` is the language*

$$\{(M, x, 1^n, 0^*) : \textit{Turing machine } M \textit{ accepts } x \textit{ using at most } n \textit{ space.}\}$$

Note: `SPACE TMSAT` $\in$ **SPACE**$[O(n)]$ and `SPACE TMSAT` is **SPACE**$[O(n)]$ complete. The 0^* is just there to make it explicit the language is paddable. In particular, this means that the circuit complexity of `SPACE TMSAT` is non-decreasing.

▶ **Lemma 14** (`SPACE TMSAT` Circuit Complexity is Non-Decreasing). *If $A'(n)$ is the size of the minimum circuit solving `SPACE TMSAT` for inputs of length n, then $A'(n)$ is non-decreasing.*

Proof. Let C be the circuit of size $A'(n+1)$ solving SPACE TMSAT for length $n+1$ inputs. Then to get a circuit for length n inputs, use C with an extra 0 hard coded into the last input. The resulting circuit will be at most the size of C and solve length n inputs. Thus $A'(n+1) \geq A'(n)$. ◀

Then using Theorem 3, we can get a **PCP** for SPACE TMSAT by setting $T = 2^{O(n)}$ and $S = O(n)$. This can be turned into a **PCP** with a binary alphabet by replacing every query for a symbol in Σ with $O(\log(n))$ queries to the individual bits of that symbol.

▶ **Corollary 15** (**PCP** for SPACE TMSAT)**.** *There is a* **PCP** *for SPACE TMSAT with:*

1. *Verifier time* $\tilde{O}(n)$.
2. *Query time* $\tilde{O}(n)$.
3. $\mathbf{poly}(\log(n))$ *queries.*
4. *Binary alphabet.*
5. *Log of proof length* $\tilde{O}(n)$.
6. *Prover space* $\tilde{O}(n)$.
7. *Soundness* $1/2$ *and perfect completeness.*

We prove three different **MATIME**/1 lower bounds that are based on three different hard problems. Different ones work better in different parameter regimes. After constructing them all, we show we always fall into some range of parameters so that we can get the lower bounds of Theorem 1.

3.1 Implicitly Encoding Advice in Input Length

In each of our cases, we will use advice to find the size of some prover circuit. To do this, we implicitly encode a number in the input length. If that implicitly encoded number describes the size, our advice bit will be 1. Otherwise, the advice bit is 0.

For any input length $n \in \mathbb{N}$, for some $l \in \mathbb{N}$, we have $n \in [2^l, 2^{l+1})$. For such an l, there is some $m \in \mathbb{N}$ such that $n = 2^l + m$. This m, or equivalently this l, is our implicitly encoded number. Because we will use this decomposition a lot, we will explicitly define some functions that perform this decomposition.

▶ **Definition 16** (Implicit Encoding In Input)**.** *For natural* $n \geq 1$*, let* $l \geq 0$ *be an integer so that* $n \in [2^l, 2^{l+1})$*, and* $m \geq 0$ *be an integer so that* $n = 2^l + m$*. Then define* $\mu(n) = m$ *and* $\rho(n) = l$.

There is a simple interpretation of this $m = \mu(n)$ and $l = \rho(n)$ in terms of the binary representation of n. You can think of l as the length of the binary number, and m the binary number after the top bit is removed.

3.2 SPACE TMSAT not in P/poly

In this case, we follow the proof in the original work [29] where $\mathbf{PSPACE} \not\subset \mathbf{P/poly}$. We present the same arguments here with more precise parameters.

When $\mathbf{PSPACE} \not\subset \mathbf{P/poly}$, the circuit complexity of SPACE TMSAT for different input sizes could change drastically and in a way that may be hard to analyze. This is an issue because the **PCP** for SPACE TMSAT needs a prover with a longer input than the input being verified, thus might require a much larger circuit.

Instead, we use a downward self reducible **PSPACE** complete language. Specifically, a language that has a sound interactive protocol with queries the same length as its input and whose prover is the language itself. We cite the result from Lemma 11 in [29]:

▶ **Lemma 17** (Same Size, Self Proving **PSPACE** Complete Language). *There is a* **PSPACE***-complete language Y and a probabilistic polynomial-time oracle Turing machine M such that for any input x:*

1. *M only asks its oracle queries of length $|x|$.*
2. *If M is given Y as oracle and $x \in Y$, then M accepts with probability 1.*
3. *If $x \notin Y$, then irrespective of the oracle given to M, M rejects with probability at least $1/2$.*

The important feature of language Y is that for an input x, the prover for x is the same language Y, and queries to the prover have the same length as x. This means Y, and the prover for Y, have the same circuit. Then if $\mathbf{PSPACE} \not\subset \mathbf{P/poly}$, then the polynomial overhead of the proof protocol grows much more slowly than the difficulty of Y. So a padded version of Y can be used as our language in this case. A proof sketch is in Appendix A, full proofs are in the full version [9].

▶ **Lemma 18** (Bound if **PSPACE** does not have Polynomial Sized Circuits). *If* $\mathit{SPACE\ TMSAT} \notin \mathbf{P/poly}$*, then for any $k > 0$, and some $f(n) = o(1)$:*

$$\mathbf{MATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

3.3 SPACE TMSAT in Almost Linear Size

The idea in this case is to use a brute force, small space algorithm that finds a problem not in a fixed polynomial size. In particular, for circuit size $S(n)$, the brute force algorithm uses space $O(S(n))$ to compute some function with minimum circuit size $\Theta(S(n))$. Then we want to simulate the **PCP** from Corollary 15 to prove the output of this algorithm. Since the **PCP** is efficient, the prover for this algorithm does not use much more space than the brute force algorithm itself.

If `SPACE TMSAT` has almost linear sized circuits, the prover doesn't require much larger circuits than the space of the prover. Finally, our **PCP** is efficient, so the time of the **MA** verifier isn't much more than the size of the prover circuit. So the **MA** protocol doesn't require much more time then the size of the circuit it proves the output of.

If `SPACE TMSAT` requires larger circuits, say quadratic circuits, then the size of the prover circuits would be quadratically larger than the input length of the prover. That is, the prover circuit would be quadratically larger than the circuit it is trying to prove. This would give quadratic overhead for the **MA** verifier time over the size of the circuit it verifies. So this construction only works well enough when `SPACE TMSAT` has almost linear sized circuits.

So this gives a proof when `SPACE TMSAT` has almost linear sized circuits. A proof sketch is in Appendix A, full proofs are in the full version [9].

▶ **Lemma 19** (Bound if `SPACE TMSAT` has Size $n^{1+o(1)}$). *If for some $g(n) = o(1)$ and some non-decreasing function $A(n) = n^{1+g(n)}$ we have $\mathit{SPACE\ TMSAT} \in \mathbf{SIZE}[O(A(n))]$, then for any $k > 0$, there is an $f(n) = o(1)$ such that:*

$$\mathbf{MATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

3.4 SPACE TMSAT in Polynomial Size, but not Almost Linear

This is the "bad" case, where we can't prove the result for every constant k, only for $k < a$. This is the most complicated case, requiring us to both pad the input to get the correct problem difficulty, and use advice to get the size of the circuits for the prover. The idea is to solve `SPACE TMSAT` on a padded version of the input using our **PCP**. So we need the advice to tell us three things:

1. Some m so that SPACE TMSAT on length m inputs requires circuits of size $\omega(n^k)$.
2. Further, we need SPACE TMSAT on length m inputs to require circuits of size near m^a. This keeps the prover from requiring circuits too much larger than SPACE TMSAT on length m inputs does.
3. How big the circuit for the prover in Corollary 15 needs to be.

Similar to the previous cases, this advice will come implicitly from the input length, and the single advice bit will be 1 if and only if the input length encodes valid advice. A skectch is in Appendix A. For full details, see the full version [9].

▶ **Lemma 20** (Bound if SPACE TMSAT has size $n^{a+o(1)}$)**.** *Suppose for some function $h(n)$ with $|h(n)| = o(1)$ and for some constant $a > 1$, for some function $A(n)$ we have $A(n) = n^{a+h(n)}$. Then if $A(n)$ is non-decreasing and we have SPACE TMSAT $\in$ $\mathbf{SIZE}[O(A(n))] \setminus \mathbf{SIZE}[o(A(n))]$, then for any $k < a$, for some $f(n) = o(1)$,*

$$\mathbf{MATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

3.5 Altogether

Altogether, these three cases imply Theorem 1. More details can be found in the full version [9].

▶ **Theorem 1** (Fine Grained **MA** Lower Bound)**.** *There exists a constant $a > 1$, such that for all $k < a$, for some $f(n) = o(1)$,*

$$\mathbf{MATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

Proof. First, we will find the best polynomial approximation of the circuit complexity of SPACE TMSAT. So define set

$$S = \{a \in \mathbb{R} : \text{SPACE TMSAT} \in \mathbf{SIZE}[O(n^a)]\}.$$

If $S = \emptyset$, then there is no constant a such that SPACE TMSAT $\in \mathbf{SIZE}[O(n^a)]$. Then SPACE TMSAT $\notin \mathbf{P/poly}$, so we use Lemma 18.

So suppose $S \neq \emptyset$. One can show that SPACE TMSAT requires linear size circuits since it depends on all inputs. Thus for any $a < 1$, we know that SPACE TMSAT $\notin \mathbf{SIZE}[O(n^a)]$, that is $a \notin S$. So 1 is a lower bound for S.

Then the set S is nonempty and has a lower bound. So S has an infimum, a, so that for any constant $\epsilon > 0$, we have SPACE TMSAT $\in \mathbf{SIZE}[O(n^{a+\epsilon})]$, but SPACE TMSAT $\notin \mathbf{SIZE}[O(n^{a-\epsilon})]$. This implies that for some $h(n)$ with $|h(n)| = o(1)$ and $A(n) = n^{a+h(n)}$ that SPACE TMSAT $\in \mathbf{SIZE}[O(A(n))] \setminus \mathbf{SIZE}[o(A(n))]$.

If $a = 1$, we use Lemma 19. If $a > 1$, we use Lemma 20. ◀

To prove Theorem 2, we use a proof similar to Lemma 19. See the full paper [9] for details.

4 PCP Sketch

We give a brief overview of the **PCP** proof. A full proof is available at [9].

Our overall protocol constructs an extrapolatable **PCP** by observing that a simple BFL style **PCP** is extrapolatable. Then this can be turned into an **rPCP** with an efficient query function. Finally we can compose this with a decodable, BFL style **PCP** with a very fast verifier.

4.1 Extrapolatable PCPs

We start by showing some lemmas about extrapolatability that make the definition easy to work with. Notably, if 2 functions are extrapolatable, so is their concatenation. This allows us to show components of a **PCP** are extrapolatable individually to get an extrapolatable **PCP**.

▶ **Lemma 21** (Extrapolatability Combination). *For integers $n, q, q', t, t' > 0$, and field $\mathbb{F}$, if $p : [q] \to \mathbb{F}^n$ is t extrapolatable, and $p' : [q'] \to \mathbb{F}^n$ is t' extrapolatable, then $g : [q + q'] \to \mathbb{F}^n$ is $O(t + t' + n \log(\mathbb{F}))$ extrapolatable where*

$$g(i) = \begin{cases} p(i) & i \leq q \\ p'(i-q) & i > q \end{cases}.$$

Of course, extrapolatability's main purpose is to allow efficient extrapolation.

▶ **Lemma 22** (Efficient Polynomials From Extrapolatability). *For any $n, q, t > 0$, field $\mathbb{F}$ where $|\mathbb{F}| > q$, and t extrapolatable $Q : [q] \to \mathbb{F}^n$, there is a time $O(t + q\mathbf{polylog}(|\mathbb{F}|))$ algorithm computing the value of a degree $q - 1$ polynomial, g, such that for all $i \in [q]$ we have $g(i) = Q(i)$.*

Now we introduce extrapolatable **PCP**s and show they imply robust **PCP**s with fast query functions. Extrapolatable **PCP**s are **PCP**s whose proofs are low degree functions, only require soundness against proofs that are low degree functions, and whose query location function is extrapolatable.

In our notation, the query function Q returns a single query location of the verifier V on a choice of randomness, whereas I returns every query location for a choice of randomness. Query function, Q, needs to be extrapolatable, but index function, I, is more convenient for defining completeness and soundness. See standard **PCP**s, Definition 12, for reference.

▶ **Definition 23** (Extrapolatable **PCP**). *We say a non-adaptive **PCP**, A, for language L with verifier V, prover P, individual query function Q, and index function I is an extrapolatable **PCP** (**ePCP**) if for some m and d:*

1. *For some field $\mathbb{F}$, A uses alphabet $\mathbb{F}$.*
2. *The proof length is $|\mathbb{F}|^m$.*
   That is, any proof, π, can be viewed as a function $\pi : \mathbb{F}^m \to \mathbb{F}$.

Low Degree Completeness *If $x \in L$ and $n = |x|$, then there exists a polynomial $\pi^x : \mathbb{F}^n \to \mathbb{F}$ of degree at most d such that $\Pr_r[V(x, r, \pi^x(I(x, r))) = 1] = 1$, and for every $i \in [l(n)]$, we have $P(x, i) = \pi^x_i$.*

Low Degree Soundness *If $x \notin L$ then for every polynomial $\pi' : \mathbb{F}^n \to \mathbb{F}$ of degree at most d has $\Pr_r[V(x, r, \pi'(I(x, r))) = 1] \leq \delta$.*

Further, we say A has:

1. *Extrapolation time $t(n)$ if for any x, r, the function $Q_{x,r}(i) = Q(x, r, i)$ is time $t(n)$ extrapolatable.*
2. *Degree d and m variables.*
3. *Low degree soundness δ.*
4. *Perfect low degree completeness.*

The main application of an **ePCP** is a convenient primitive in constructing efficient robust **PCP**s (**rPCP**). Robust **PCP**s are a standard primitive in the literature and are commonly used in **PCP** composition [3, 7, 13, 25, 12]. This conversion of **ePCP** to **rPCP** is where all the low degree testing is performed. But low degree test queries are simple lines so are easy to compute. Note that our **rPCP**s constructed from an **ePCP** uses the same proof, so if the **ePCP** prover is efficient, so is the **rPCP** prover. See the full paper [9] for a proof.

▶ **Theorem 24** (**ePCP** gives efficient **rPCP**). *For any language L with an* **ePCP**, *A, with*

1. *Verifier time $t(n)$.*
2. *Verifier space $s(n)$.*
3. *Extrapolation time $t'(n)$.*
4. *Randomness $r(n)$.*
5. *Degree $d(n)$ and $m(n)$ variables.*
6. *$q(n)$ queries.*
7. *Alphabet $\mathbb{F}$ where $|\mathbb{F}| > 10q(n)d(n)$.*
8. *Prover P.*
9. *Low degree soundness* 0.1.
10. *Perfect low degree completeness.*

Language L has an **rPCP**, *B, with*

1. *Verifier time $O(t(n) + |\mathbb{F}|^3\mathbf{polylog}(|\mathbb{F}|))$.*
2. *Verifier space $O(s(n) + \log(|\mathbb{F}|))$.*
3. *Randomness $r(n) + O(m(n)\log(|\mathbb{F}|))$.*
4. *Query time $O(t'(n) + (q(n) + m(n))\mathbf{polylog}(|\mathbb{F}|))$.*
5. *$O(|\mathbb{F}|)$ queries.*
6. *Prover P with perfect completeness.*
7. *Soundness at most* 0.99.

4.2 Constructing our Extrapolatable PCP

Our **ePCP** is a BFL [4] style **PCP**. We start by converting our algorithm into a cellular automata so that it has very uniform, local constraints, similar to the Cook-Levin theorem [10, 21], or what was done in [20]. In particular, the constraints can be described by a simple polynomial of the computation history of the cellular automata.

Then our **ePCP** will ask for a multilinear extension of the computation history, a low degree polynomial encoding the constraint function of that computation history, and a sum check [22] style proof verifying that constraint is always satisfied. Then the verifier just needs to check the computation history is consistent with the input, the constraint polynomial is consistent with the computation history polynomial, and the sum check of the constraint polynomial succeeds. All low degree testing is handled in the conversion from an **ePCP** to an **rPCP**.

Now we verify that a variation on standard sum check [22, 4] is extrapolatable. Since **ePCP**s only need soundness against low degree proofs, our base **ePCP** only needs to perform constantly many queries besides the sum check. So the check sum is the most difficult thing to prove the extrapolatability of.

▶ **Lemma 25** (Sum Check Queries Are Extrapolatable). *For any variable number $n \in \mathbb{N}$, degree $d \in \mathbb{N}$, field $\mathbb{F}$ with $|\mathbb{F}| > \max\{d, n\} + 1$, and randomness r, the query location function, Q_r, used in the sum check protocol for a degree d and n variable function are $O(nd\mathbf{polylog}(|\mathbb{F}|))$ extrapolatable.*

The proof is straightforward and comes from a close observation of the specific sum check query locations.

We need a low space prover to use in our **MA** lower bounds. Creating the computation history itself takes about as much space as the original algorithm. Multilinear extensions can be done straightforwardly in small space if one only needs one symbol, or does not care about time.

▶ **Lemma 26** (Multilinear Extensions Require Low Space)**.** *Suppose function* $G : \{0,1\}^n \to \{0,1\}$ *is computable in space* S*. Then the multilinear function* g *consistent with* G *on Boolean inputs is computable in space* $O(n \log(|\mathbb{F}|) + S)$*.*

Similarly, individual symbols from the constraint polynomial can be computed in small space from the multilinear extension of the computation history. The sum check proof is essentially partial linearizations of the constraint polynomial, so can also be done in small space.

4.3 Decodable PCP

After that, we use a similar **PCP** for the decodable inner **PCP** (**dPCP**), and apply a standard robust, decodable **PCP** composition [12]. Essentially only two changes need to be made to get our **dPCP**. First we need to only check consistency of the computation history with the explicit input (and *not* the implicit input). Second, we need to do an additional query to an implicit input to decode from it. This does not require any new tricks.

Finally, we do a standard **PCP** composition of a robust **PCP** with a decodable **PCP** to do query reduction. We note that we can not do the standard trick of having the outer **rPCP** output a circuit that the inner **dPCP** proves the output of as that circuit description would be too large for a verifier to compute. Instead this must be done implicitly, similar to [5]. We also note that to keep the space of the inner **PCP** prover small requires the space of the outer **PCP** prover, the space of the outer query function, *and* the space of the outer **PCP** verifier to be small.

4.4 Fine Grained MIP = NEXP

We note that the low space and deterministic algorithm requirements were only to keep the prover space low. Dropping the prover space requirement and using the same proof gives Theorem 4. Then using Theorem 4, one can use a standard technique to get $\frac{1}{\mathbf{polylog}(n)}$ error, 2 query **PCP**. Combining this with a repetition theorem [28, 19, 27] gives the 2 query, constant error **MIP** stated in Corollary 5.

5 Open Problems

There are several ways we would like to improve the circuit lower bounds.

1. Remove the advice bit.
   We still had to use advice, a limitation from the original Santhanam result. It would be nice if we could get lower bounds on **MA** with no non-uniformity.
2. Prove tight bounds for all k.
   Another limitation of our circuit lower bound is that it does not prove this tight bound for all $k > 1$, just for some k.

The major barrier is in the case that **SPACE**$[n]$ algorithms may require super linear, but polynomial, sized circuits. Then the circuit size required for any given space may change in a strange way. For example, suppose for some $a > 1$

$$\mathbf{SPACE}[n] \subseteq \mathbf{SIZE}[O(n^a)] \setminus \mathbf{SIZE}[o(n^a)].$$

What we would like, but this does not obviously imply, is that for all $b > 1$:

$$\mathbf{SPACE}[n^b] \subseteq \mathbf{SIZE}[O(n^{ab})] \setminus \mathbf{SIZE}[o(n^{ab})].$$

While a padding argument shows $\mathbf{SPACE}[n^b] \subseteq \mathbf{SIZE}[O(n^{ab})]$, it does not show that $\mathbf{SPACE}[n^b] \not\subseteq \mathbf{SIZE}[o(n^{ab})]$.

3. Make lower bound more frequent.
   We only prove that infinitely often this lower bound occurs, but it may have even super exponential gaps between input sizes where the circuit lower bound holds. Murray and Williams [26] gave a refinement of the Santhanam circuit lower bounds, that is incomparable to ours, which gives circuit lower bounds that holds for input lengths only polynomially far apart.
4. Prove exponential lower bounds for **MAEXP**.
   A similar problem is to prove exponential circuit lower bounds for the exponential version of **MA**, known as **MAEXP**. The best circuit lower bounds known for **MAEXP** are "half-exponential" by Miltersen, Vinodchandran, and Watanabe [24]. Loosely, a function is half exponential if that function composed with itself is exponential.

Another open problem is to give a **PCP** whose verifier time matches ours with only quasilinear proof length. Known **PCP**s with proof length $\tilde{O}(T)$ have verifier time $\Omega(n + \log(T)^2)$. We suspect ideas from theorem 2.6 in [5][3] (which extends the results of [7] from **NP** to **NEXP**) may give a **PCP** with verifier time near $\tilde{O}(n + \log(T))$ while giving a proof of length $T^{1+o(1)}$. But it would not have proof length $\tilde{O}(T) = T\mathbf{polylog}(T)$. We stress that close analysis of the verifier time for the **PCP** of [5] has not been performed, and that **PCP** is much more complex than ours.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

2 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998. doi:10.1145/278298.278306.

3 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122, January 1998. doi:10.1145/273865.273901.

4 L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 16–25 vol.1, 1990. doi:10.1109/FSCS.1990.89520.

5 E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Short pcps verifiable in polylogarithmic time. In *20th Annual IEEE Conference on Computational Complexity (CCC'05)*, pages 120–134, 2005. doi:10.1109/CCC.2005.27.

[3] We emphasize this is the theorem labeled "Efficient PCPPs with small query complexity", which does not have quasi-linear proof length.

6 Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 585–594. Association for Computing Machinery, 2013. doi:10.1145/2488608.2488681.

7 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust pcps of proximity, shorter pcps and applications to coding. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 1–10. Association for Computing Machinery, 2004. doi:10.1145/1007352.1007361.

8 Eli Ben-Sasson and Emanuele Viola. Short pcps with projection queries. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 163–173. Springer, 2014. doi:10.1007/978-3-662-43948-7_14.

9 Joshua Cook and Dana Moshkovitz. Tighter ma/1 circuit lower bounds from verifier efficient pcps for pspace, 2022. URL: https://eccc.weizmann.ac.il/report/2022/014/.

10 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. Association for Computing Machinery. doi:10.1145/800157.805047.

11 Stephen A. Cook. A hierarchy for nondeterministic time complexity. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, STOC '72, pages 187–192. Association for Computing Machinery, 1972. doi:10.1145/800152.804913.

12 Irit Dinur and Prahladh Harsha. Composition of low-error 2-query pcps using decodable pcps. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 472–481, 2009. doi:10.1109/FOCS.2009.8.

13 Irit Dinur and Omer Reingold. Assignment testers: towards a combinatorial proof of the pcp-theorem. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 155–164, 2004. doi:10.1109/FOCS.2004.16.

14 Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1057–1068. IEEE, 2020.

15 Lance Fortnow and Rahul Santhanam. Robust simulations and significant separations. In *Proceedings of the 38th International Colloquim Conference on Automata, Languages and Programming - Volume Part I*, ICALP'11, pages 569–580. Springer-Verlag, 2011.

16 Lance Fortnow, Rahul Santhanam, and Luca Trevisan. Hierarchies for semantic classes. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 348–355. Association for Computing Machinery, 2005. doi:10.1145/1060590.1060642.

17 Lance Fortnow, Rahul Santhanam, and Ryan Williams. Fixed-polynomial size circuit bounds. In *2009 24th Annual IEEE Conference on Computational Complexity*, pages 19–26, 2009. doi:10.1109/CCC.2009.21.

18 J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965. URL: http://www.jstor.org/stable/1994208.

19 Thomas Holenstein. Parallel repetition: Simplifications and the no-signaling case. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 411–419. Association for Computing Machinery, 2007. doi:10.1145/1250790.1250852.

20 Justin Holmgren and Ron Rothblum. Delegating computations with (almost) minimal time and space overhead. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 124–135, 2018. doi:10.1109/FOCS.2018.00021.

21 Leonid Levin. Universal'nyie perebornyie zadachi (universal search problems, in russian). *Problemy Peredachi Informatsii*, 9:265–266, 1973.

22 Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, October 1992. doi:10.1145/146585.146605.

23 Or Meir. Combinatorial pcps with efficient verifiers. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 463–471, 2009. doi:10.1109/FOCS.2009.10.

24 Peter Bro Miltersen, N. V. Vinodchandran, and Osamu Watanabe. Super-polynomial versus half-exponential circuit size in the exponential hierarchy. In *Proceedings of the 5th Annual International Conference on Computing and Combinatorics*, COCOON'99, pages 210–220. Springer-Verlag, 1999.

25 Dana Moshkovitz and Ran Raz. Two query pcp with sub-constant error. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 314–323, 2008. doi:10.1109/FOCS.2008.60.

26 Cody Murray and Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime: An easy witness lemma for np and nqp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 890–901. Association for Computing Machinery, 2018. doi:10.1145/3188745.3188910.

27 Anup Rao. Parallel repetition in projection games and a concentration bound. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 1–10. Association for Computing Machinery, 2008. doi:10.1145/1374376.1374378.

28 Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998. doi:10.1137/S0097539795280895.

29 Rahul Santhanam. Circuit lower bounds for merlin-arthur classes. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 275–283. Association for Computing Machinery, 2007. doi:10.1145/1250790.1250832.

30 Joel I. Seiferas, Michael J. Fischer, and Albert R. Meyer. Separating nondeterministic time complexity classes. *J. ACM*, 25(1):146–167, 1978. doi:10.1145/322047.322061.

31 Adi Shamir. Ip = pspace. *J. ACM*, 39(4):869–877, October 1992. doi:10.1145/146585.146609.

32 D. van Melkebeek and K. Pervyshev. A generic time hierarchy for semantic models with one bit of advice. In *21st Annual IEEE Conference on Computational Complexity (CCC'06)*, pages 14 pp.–144, 2006. doi:10.1109/CCC.2006.7.

33 Ryan Williams. Non-uniform acc circuit lower bounds. In *2011 IEEE 26th Annual Conference on Computational Complexity*, pages 115–125, 2011. doi:10.1109/CCC.2011.36.

34 Stanislav Žák. A turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, 1983. doi:10.1016/0304-3975(83)90015-4.

A MA Circuit Lower Bound Proofs

In this section, we sketch the proofs of the main lemmas of Section 3. Full proofs are available in the full version [9].

▶ **Lemma 18** (Bound if **PSPACE** does not have Polynomial Sized Circuits). *If* $\mathit{SPACE\ TMSAT} \notin \mathbf{P/poly}$*, then for any* $k > 0$*, and some* $f(n) = o(1)$*:*

$$\mathbf{MATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

Proof. The idea is to take length m inputs to Y, pad them up to length n inputs so that Y on length m inputs requires, and has, circuits of size just slightly larger than n^k. Then as long as n grows much faster than m, the overhead of the interactive protocol verifier will be small and we only need to focus on prover overhead, which is around n^k since the circuit for Y is the prover for Y.

First, I claim that, since $Y \notin \mathbf{P/poly}$, for some $g(n) = \omega(1)$, we have $Y \notin \mathbf{SIZE}[O(n^{g(n)})]$. If this was not true, then Y must have circuits of size n^k for some k upper bounding the best choice of $g(n)$.

Now we define our language, W, in $\mathbf{MATIME}[n^{k+f(n)}]/1$ but not in $\mathbf{SIZE}[O(n^k)]$. For any input size, n, using Definition 16, let $m = \mu(n)$ and $l = \rho(n)$. Let our advice bit be 1 if both:

1. Y on length m inputs has circuits with size $n^k \log(n)$. This makes sure there is a fast prover for Y.
2. And Y on length m inputs does not have circuits of size $m^{g(m)}$. This makes sure m grows much more slowly than n.

Then $x \in W$ for some x with $|x| = n$ if and only if the advice bit is 1 and for some $y \in Y$ with $|y| = m$ we have $x = y1^{n-m}$.

Let $a > 0$ be the constant so that the verifier (M in Lemma 17) for Y's interactive protocol runs in time $O(n^a)$. Define the **MA** protocol as follows. If the trusted advice bit is one, Merlin gives Arthur a length $n^k \log(n)$ circuit. Then Arthur simulates the interactive protocol for Y on y using the circuit from Merlin as the prover. This protocol runs in time $O(m^a n^k \log(n))$. By assumption we have $m^{g(m)} < n^k \log(n)$, which one can show implies $m < n^{f'(n)}$ for some $f'(n) = o(1)$. Thus the $\mathbf{MA}/1$ protocol runs in time $O(n^{f'(n)} n^k \log(n)^2) = n^{k+f(n)}$ for some $f(n) = o(1)$.

By choice of $g(m)$, for infinitely many m, the second condition is satisfied and $g(m) > k+1$. For such m, the smallest choice of $n = 2^l + m$ where Y on length m inputs has circuits of size $n^k \log(n)$ can not have circuits of size $(n/2)^k \log(n/2)$. For such n, language W does not have circuits of size $O(n^k)$, but does have an **MA** protocol running in time $n^{k+f(n)}$. ◀

▶ **Lemma 19** (Bound if `SPACE TMSAT` has Size $n^{1+o(1)}$)**.** *If for some $g(n) = o(1)$ and some non-decreasing function $A(n) = n^{1+g(n)}$ we have `SPACE TMSAT` $\in \mathbf{SIZE}[O(A(n))]$, then for any $k > 0$, there is an $f(n) = o(1)$ such that:*

$$\mathbf{MATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

Proof. The proof proceeds in five steps.

1. Find a language $L \in \mathbf{SPACE}[n^k \log(n)^2] \setminus \mathbf{SIZE}[n^k \log(n)/10]$. In particular, for every input length n, language L has circuits of size $n^k \log(n)$ but not $n^k \log(n)/10$.
2. Reduce L to `SPACE TMSAT` and use Corollary 15. In particular, find a circuit, C_n, for the prover in an **MA** protocol for L on length n inputs.
3. Define our advice bit to implicitly give an upper bound for the size of C_m for some m within a factor of 2 of n. Then we define W to be length m elements of L, padded to length n.
4. Show that infinitely often the advice bit is 1 and W does not have small circuits.
5. Show that W has an efficient **MA** protocol.

With that outline in mind, let us begin the proof.

1. Find a language $L \in \mathbf{SPACE}[n^k \log(n)^2] \setminus \mathbf{SIZE}[n^k \log(n)/10]$.
   From the non-uniform hierarchy (in Arora and Barak [1, Theorem 6.22]), there is a language $L \in \mathbf{SIZE}[n^k \log(n)] \setminus \mathbf{SIZE}[n^k \log(n)/10]$. In particular, for every n, language L on length n has circuits size $n^k \log(n)$ but not size $n^k \log(n)/10$. A brute force search can then find and simulate such a circuit using space $O(n^k \log(n)^2)$.
2. Reduce L to `SPACE TMSAT` and use Corollary 15.
   Since M only uses space $g(n) = \tilde{O}(n^k)$, we know $x \in L$ if and only if $(M, x, 1^{g(n)}, 0) \in$ `SPACE TMSAT`. We know `SPACE TMSAT` on length $\tilde{O}(n^k)$ inputs has a **PCP** protocol from Corollary 15 that uses $\mathbf{polylog}(n)$ many length $\tilde{O}(n^k)$ queries to a space $\tilde{O}(n^k)$ prover, P, where each query can be calculated by a time $\tilde{O}(n^k)$ algorithm, Q, and the results from P are verified by a time $\tilde{O}(n^k)$ verifier, V.

Now we reduce the prover P to SPACE TMSAT so we can use that SPACE TMSAT $\in$ **SIZE**$[O(n^{1+g(n)})]$ to get a circuit for P. A length $\tilde{O}(n^k)$ query, q, to P can be converted into a length $\tilde{O}(n^k)$ input, q', for SPACE TMSAT by providing the algorithm for P and $\tilde{O}(n^k)$ 1s. Call the circuit for SPACE TMSAT on length $|q'|$ inputs C_n. Since SPACE TMSAT $\in$ **SIZE**$[O(n^{1+g(n)})]$, we know C_n has size $(\tilde{O}(n^k))^{1+g(n)} = n^{k+g'(n)}$ for some $g'(n) = o(1)$.

3. Define our advice bit.
   Now an **MA** protocol can guess C_n, but we may not be able to compute how large C_n needs to be. The function $g'(n)$ may be hard to compute. So we use advice.
   Let $l = \rho(n)$, $m = 2^l$ and $t = \mu(n)$ so that $n = m + t$. Then let the advice bit be 1 if
   a. Circuit C_m has size $m^k 2^t$.
   b. For any natural t' less than t, circuit C_m does not have size $m^k 2^{t'}$.
      This condition allows us to use the smallest t possible for a given m.
   Then $x \in W$ for some x with $|x| = n$ if and only if the advice bit is 1 and for some $y \in$ SPACE TMSAT with $|y| = m$ we have $x = y1^{n-m}$.
4. Show W does not have small circuits.
   First see that for every large enough l, for $m = 2^l$, there will be one t such that this advice bit is 1. That is for some $t < m$, C_m has size $m^k 2^t$. Otherwise, C_m would have exponential size, but it only has almost linear size. Then for the minimum such t, the advice bit will be one. So infinitely often, the advice bit will be 1.
   When the advice bit is 1, the language W on length $n = m + t$ inputs is equal to L on length m inputs. Language L on length m inputs does not have circuits of size $m^k \log(m)/10$. See by choice of m that $2m > n$, so $n^k = o(m^k \log(m)/10)$. Thus infinitely often, W does not have size n^k circuits. Thus $W \notin$ **SIZE**$[O(n^k]$.
5. Show W has an efficient **MA** protocol.
   If the advice bit is 0, this is trivially true. For $n = 2^l + t$ so that the advice bit is 1 and $m = 2^l$, either
   $t = 0$: Then C_m has size $n^k = O(m^{k+g'(m)})$.
   $t \geq 1$: Then C_m has size $m^k 2^t$ but not $m^k 2^{t-1}$. Since C_m does have circuits of size $m^{k+g'(m)}$, we have $m^k 2^t = O(m^{k+g'(m)})$.
   In either case, an **MA** protocol can guess C_m with a circuit with size $O(m^{k+g'(m)})$.
   Then an **MA** protocol for $x = y1^{n-m}$ and an advice bit of 1 can verify if $y \in L$ by first guessing a circuit for C_m, then using it as the prover in the **PCP** protocol from Corollary 15.
   The **MA** verifier needs to calculate **polylog**(m) queries with Q, run C_m on each of those queries, and run V on those results. Since C_m has size $O(m^{k+g'(m)})$, and Q and V run in time $\tilde{O}(m^{k+g'(m)})$, calculating all query locations, running C_m on each of those locations, and V on those outputs takes time

$$\begin{aligned}&\textbf{polylog}(m)(\tilde{O}(m^k) + O(m^{k+g'(m)})) + \tilde{O}(m^k)\\ =&\tilde{O}(m^{k+g'(m)}).\end{aligned}$$

   Finally, since $m < n$, for some $f(n) = o(1)$, the **MA** verifier runs in time $n^{k+f(n)}$.
   The **MA** protocol is complete and sound since the **PCP** is. Thus

$$W \in \textbf{MATIME}[O(n^{k+f(n)})]/1.$$ ◀

▶ **Lemma 20** (Bound if SPACE TMSAT has size $n^{a+o(1)}$)**.** *Suppose for some function $h(n)$ with $|h(n)| = o(1)$ and for some constant $a > 1$, for some function $A(n)$ we have $A(n) = n^{a+h(n)}$. Then if $A(n)$ is non-decreasing and we have SPACE TMSAT $\in$ **SIZE**$[O(A(n))] \setminus$ **SIZE**$[o(A(n))]$, then for any $k < a$, for some $f(n) = o(1)$,*

$$\textbf{MATIME}[O(n^{k+f(n)})]/1 \not\subset \textbf{SIZE}[O(n^k)].$$

Proof. We want to solve a smaller instance of SPACE TMSAT that requires circuits of size $n^k \log(n)$, and we also need advice to tell us the size of circuits needed to prove SPACE TMSAT. The advice for this will come implicitly from the input length.

For input x of length n, (using ρ and μ from Definition 16) let $l = \rho(n)$, $l' = \rho(\mu(n))$, and $t = \mu(\mu(n))$ so that $n = 2^l + 2^{l'} + t$. We want to solve SPACE TMSAT on length $2^{l'}$ inputs, so we let $m := 2^{l'}$. Then $n = 2^l + m + t$ and our language will solve length m inputs for SPACE TMSAT using prover circuits of size $(2^l)^k 2^t$. Our advice bit will tell us when 2^l, m and t are appropriate.

So then m is the input length to SPACE TMSAT we want to solve, 2^l is how much padding is needed to make length m problems the right difficulty, and $(2^l)^k 2^t$ is the size of the circuits needed for our **PCP** prover.

The proof proceeds in 4 steps.

1. Define circuits C_m that prove SPACE TMSAT for length m inputs using our **PCP** and our theorem assumptions on circuits for SPACE TMSAT.
2. Define when the advice bit should be 1.
3. Show infinitely often the advice bit is 1 and $W \notin$ **SIZE**$[O(n^k)]$.
4. Show that W has an efficient **MA** protocol.

Now following this outline:

1. Define circuits C_m that prove SPACE TMSAT for length m inputs.
   Then SPACE TMSAT on length m inputs has a **PCP** protocol with verifier time $\tilde{O}(m)$, log of proof length $\tilde{O}(m)$, and prover space $\tilde{O}(m)$. Then the prover for SPACE TMSAT on length m inputs can be reduced to a circuit for SPACE TMSAT with length $\tilde{O}(m)$ inputs. Then SPACE TMSAT on length $\tilde{O}(m)$ inputs has a circuit, C_m, of size at most $\tilde{O}(m)^{a+h(m)}$. So for some $h'(m) = o(1)$, we have $|C_m| \leq m^{a+h'(m)}$.
2. Define when the advice bit should be 1.
   Since SPACE TMSAT $\notin$ **SIZE**$[o(A(n))]$, for some $c_1 > 0$, for some infinite set, U', for all $n' \in U'$, language SPACE TMSAT on length n' inputs does not have circuits with size $c_1 A(n')$.
   Let the advice bit be 1 if and only if each of the following hold:
   a. SPACE TMSAT on length m inputs does not have circuits with size at most $c_1 A(m/2)$. This restricts us to m where the circuits for SPACE TMSAT require size near our upper bound. This limits how much bigger C_m needs to be than the circuits for SPACE TMSAT on length m inputs.
   b. SPACE TMSAT on length m inputs does not have a circuit with size $(l-1)(2^{l-1})^k$.
   c. SPACE TMSAT on length m inputs does have a circuit with size $l(2^l)^k$.
   d. Circuit C_m has size $(2^l)^k 2^t$.
   e. Either $t = 0$, or C_m does not have size $(2^l)^k 2^{t-1}$.
   Then $x \in W$ for some x with $|x| = n$ if and only the advice bit is 1 and for some $y \in$ SPACE TMSAT with $|y| = m$ we have $x = y1^{n-m}$.
3. Now we will argue that infinitely often the advice bit is 1 and W does not have circuits with size $O(n^k)$. See the full version [9] for full details. We do this in a few steps:

- First restrict our focus to m large enough and where SPACE TMSAT on length m inputs requires circuits of size at least $c_1 A(m/2)$. This will be the set of input lengths, U. Since m can be any power of 2 and A is monotone, as long as A is tight, we can always find an infinite set of $m = 2^{l'}$ such that SPACE TMSAT on length m inputs requires circuits of size $A(m/2)$.
- For each $m \in U$, find appropriate l and t.
  Take the smallest l so that SPACE TMSAT on length m inputs does have a circuit of size $l(2^l)^k$. Note that $l > l' = \log(m)$, since SPACE TMSAT on length m inputs does not have circuits with size $m^k \log(m)$ since $k < a$.
  Let t be the smallest t such that C_m has size $(2^l)^k 2^t$. Since we know C_m has poly sized circuits, we know that some $t < m$ achieves this.
- Now for $n = 2^l + m + t$, the advice bit is 1 and language SPACE TMSAT on length n inputs does not have circuits with size $O(n^k)$.

4. Show that

$$W \in \mathbf{MATIME}\left[n^{k+f(n)}\right]/1.$$

If the advice bit is 0, this is trivial. Otherwise, assume for n the advice bit is 1.
When the advice bit is 1, we know C_m has size at most $(2^l)^k 2^t$ and either
$t = 0$: Then C_m has size $(2^l)^k = O(n^k)$.
$t \geq 1$: Then C_m does not have size $(2^l)^k 2^{t-1}$ by choice of t. Circuit C_m has size $m^{a+h'(m)}$. Further, SPACE TMSAT on length m inputs does not have circuits with size $c_1 A(m/2)$ since the advice bit is 1, but it does have circuits with size $l(2^l)^k$. Together this implies that

$$\begin{aligned} c_1 A(m/2)\frac{2^{t-1}}{l} &< m^{a+h'(m)} \\ 2^t &< \frac{2}{c_1}\frac{l m^{a+h'(m)}}{A(m/2)}. \end{aligned}$$

Then one can show that $2^t = m^{o(1)}$. So for some $f'(n) = o(1)$ we have $(2^l)^k 2^t = n^{k+f'(n)}$.

The verifier for SPACE TMSAT can be simulated in time $\tilde{O}(m)$, and the $\mathbf{poly}(\log(m))$ queries to the prover can be simulated in time

$$\mathbf{poly}(\log(m)) S(2^l) 2^t = n^{k+f(n)}$$

for some $f(n) = o(1)$. Thus

$$W \in \mathbf{MATIME}\left[n^{k+f(n)}\right]/1. \qquad \blacktriangleleft$$