Electronic Colloquium on Computational Complexity, Report No. 159 (2021)

Constructive Separations and Their Consequences

Lijie Chen
MIT
lijieche@mit.edu

Ce Jin
MIT
cejin@mit.edu

Rahul Santhanam
University of Oxford
rahul.santhanam@cs.ox.ac.uk

Ryan Williams*
MIT
rrw@mit.edu

Abstract

For a complexity class $\mathcal{C}$ and language L, a **constructive separation of** $L \notin \mathcal{C}$ gives an efficient algorithm (also called a **refuter**) to find counterexamples (bad inputs) for every $\mathcal{C}$ algorithm attempting to decide L. We study the questions: Which lower bounds can be made constructive? What are the consequences of constructive separations? We build a case that "constructiveness" serves as a dividing line between many weak lower bounds we know how to prove, and strong lower bounds against P, ZPP, and BPP. Put another way, constructiveness is the opposite of a complexity barrier: it is a property we want lower bounds to have. Our results fall into three broad categories.

- **For many separations, making them constructive would imply breakthrough lower bounds.** Our first set of results shows that, for many well-known lower bounds against streaming algorithms, one-tape Turing machines, and query complexity, as well as lower bounds for the Minimum Circuit Size Problem, making these lower bounds constructive would imply breakthrough separations ranging from $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$ to even $\mathsf{P} \neq \mathsf{NP}$. For example, it is well-known that distinguishing binary strings with $(1/2 - \varepsilon)n$ ones from strings with $(1/2 + \varepsilon)n$ ones requires randomized query complexity $\Theta(1/\varepsilon^2)$. We show that a sufficiently constructive refuter for this query lower bound would imply $\mathsf{P} \neq \mathsf{NP}$.
- **Most conjectured uniform separations can be made constructive.** Our second set of results shows that for most major open problems in lower bounds against P, ZPP, and BPP, including $\mathsf{P} \neq \mathsf{NP}$, $\mathsf{P} \neq \mathsf{PSPACE}$, $\mathsf{P} \neq \mathsf{PP}$, $\mathsf{ZPP} \neq \mathsf{EXP}$, and $\mathsf{BPP} \neq \mathsf{NEXP}$, any proof of the separation would further imply a *constructive* separation. Our results generalize earlier results for $\mathsf{P} \neq \mathsf{NP}$ [Gutfreund, Shaltiel, and Ta-Shma, CCC 2005] and $\mathsf{BPP} \neq \mathsf{NEXP}$ [Dolev, Fandina and Gutfreund, CIAC 2013]. Thus any proof of these strong lower bounds must also yield a constructive version, compared to many weak lower bounds we currently know.
- **Some separations cannot be made constructive.** Our third set of results shows that certain complexity separations cannot be made constructive. We observe that for $t(n) \geq n^{\omega(1)}$ there are no constructive separations for $\mathsf{R}_{\mathsf{K}^t}$ (which is known to be not in P) from any complexity class, unconditionally. We also show that under plausible conjectures, there are languages in $\mathsf{NP} \setminus \mathsf{P}$ for which there are no constructive separations from any complexity class.

*Supported by NSF CCF-2127597 and CCF-1909429. Part of this work was completed while the author was visiting the Simons Institute for the Theory of Computing, participating in the *Theoretical Foundations of Computer Systems* and *Satisfiability: Theory, Practice, and Beyond* programs.

ISSN 1433-8092

1 Introduction

A primary goal of complexity theory is to derive strong complexity lower bounds for natural computational problems. When a lower bound holds for a problem Π against a model $\mathcal{M}$ of algorithms, this implies that for each algorithm A from $\mathcal{M}$, there is an infinite sequence of *counterexamples* $\{x_i\}$ for which A fails to solve Π correctly.[1] In this paper, we study the question: can such a family of counterexamples be constructed efficiently, for fixed Π and a given algorithm A in $\mathcal{M}$? We call a positive answer to this question a *constructive* separation of Π from $\mathcal{M}$.

There are several motivations for studying this question in a systematic way for natural problems Π and models $\mathcal{M}$. Computer science is inherently a constructive discipline, and it is natural to ask if a given lower bound can be made constructive. Indeed, this can be seen as an "explicit construction" question of the kind that is studied intensively in the theory of pseudorandomness, where we may have a proof of existence of certain objects with optimal parameters, e.g., extractors, and would like to construct such objects efficiently. At a high level, cryptography is based on the constructiveness of lower bounds: we need lower bounds to exist, and we also need to sample hard instances efficiently.[2]

Our primary motivation is to understand the general lower bound problem better! Constructive lower bounds have led to some recent resolutions of lower bound problems in complexity theory, and we believe they will lead to more. In his Geometric Complexity Theory approach, Mulmuley [Mul10] suggests that in order to break the "self referential paradox" of P vs NP and related problems[3], one has to shoot for **algorithms** which can efficiently find counterexamples for any algorithms claiming to solve the conjectured hard language. This view has been dominant in the GCT approach towards the VNP vs. VP problem [Mul07, Mul12, IK20].

The ability to "construct bad inputs for a hard function" has also been critical to some recent developments in (Boolean) complexity theory. Chen, Jin, and Williams [CJW20] studied a notion of constructive proof they called *explicit obstructions*. They show several "sharp threshold" results for explicit obstructions, demonstrating (for example) that explicit obstructions unconditionally exist for $n^{2-\varepsilon}$-size DeMorgan formulas, but if they existed for $n^{2+\varepsilon}$-size formulas then one could prove the breakthrough lower bound $\mathsf{EXP} \not\subset \mathsf{NC}^1$. (We discuss the differences between their work and ours in Section 2.5, along with other related work.)

Constructive lower bounds have also been directly useful in proving recent lower bounds. Chen, Lyu, and Williams [CLW20] recently showed how to strengthen several prior lower bounds for E^{NP} based on the algorithmic method to hold *almost everywhere*. A key technical ingredient in this work was the development of an *constructive* version of a nondeterministic time hierarchy that was already known to hold almost everywhere [FS16]. The "refuter" in the constructive lower bound (the algorithm producing counterexamples) is used directly in the design of the hard function in E^{NP}. This gives a further motivation to study when lower bounds can be made constructive.

[1] If the family of counterexamples was finite, we could hard-code them into the algorithm A to give a new algorithm A' that solves Π correctly, for most "reasonable" models $\mathcal{M}$.

[2] A superficial difference is that in cryptography, we would like to construct instances that are hard for *any* efficient algorithm, whereas in our setting, there is a fixed algorithm, and we would like to construct instances that are hard for it. This difference vanishes when the problem Π has an optimal algorithm in the sense of Levin [Lev73], since any instance that is hard for the optimal algorithm is hard for *all* efficient algorithms.

[3] Namely, since the P vs. NP problem is a universal statement about mathematics that says that discovery is hard, why could it not preclude its own proof and hence be independent of the axioms of set theory?

The Setup. More formally, for a function $f\colon \{0,1\}^\star \to \{0,1\}$ and algorithm A, we define the *search problem* $D_{f,A}$ *of counterexamples* to be $D_{f,A} := \{(1^n, x) \mid x \in \{0,1\}^n \wedge f(x) \neq A(x)\}$. Intuitively, a *refuter* for f against A is an algorithm for the search problem $D_{f,A}$, proving in an algorithmic way that the algorithm A cannot compute f. (This notion seems to have first been introduced by Kabanets [Kab00] in the context of derandomization; see Section 2.5 for more details.)

Definition 1.1 (Refuters and Constructive Separations)**.** An algorithm R is a *refuter for f against A* if there are infinitely many n such that $(1^n, R(1^n)) \in D_{f,A}$. For complexity classes $\mathcal{C}$ and $\mathcal{D}$, we say there is a *$\mathcal{D}$-constructive separation of $f \notin \mathcal{C}$* if for every algorithm A computable in $\mathcal{C}$ there is a refuter for f against A that is computable in $\mathcal{D}$.

Note that we allow the refuter algorithm to depend on the algorithm A. The notion of refuter can also be extended naturally to randomized algorithms. Formally, we say a randomized algorithm R *solves* $D_{L,A}$ infinitely often, if for infinitely many integers n, $(1^n, R(1^n)) \in D_{L,A}$ with probability at least $2/3$. If for these infinitely many integers n, it holds in addition that $R(1^n)$ either outputs $\perp$ or a counterexample such that $(1^n, R(1^n)) \in D_{L,A}$, we say R is a *zero-error randomized algorithm solving* $D_{L,A}$.

At this point it is natural to ask:

Question 1: Which lower bounds imply a corresponding *constructive* lower bound?

Naively, one might expect that the answer to Question 1 is positive when the lower bound is relatively easy to prove. We show that this intuition is wildly inaccurate. On the one hand, we show that for many natural examples of problems Π and weak models $\mathcal{M}$, a lower bound is easily provable (and well-known), but *constructivizing* the *same* lower bound would imply a breakthrough separation in complexity theory (a much stronger type of lower bound). On the other hand, we show that for many "hard" problems Π and strong models $\mathcal{M}$, a lower bound for Π against $\mathcal{M}$ *automatically* constructivizes: the existence of the lower bound alone can be used to derive an algorithm that produces counterexamples. So, in contrast with verbs such as "relativize" [BGS75], "algebrize" [AW09], and "naturalize" [RR97], we *want* to prove lower bounds that constructivize! We are identifying a *desirable* property of lower bounds.

We now proceed to discuss our results in more detail, and then give our interpretation of these results.

1.1 Most Conjectured Poly-Time Separations Can Be Made Constructive

Generalizing prior work [GST07, DFG13], we show that for most major open lower bound problems regarding polynomial time, their resolution implies corresponding *constructive* lower bounds for most complete problems.

Theorem 1.2. *Let $\mathcal{C} \in \{\mathsf{P}, \mathsf{ZPP}, \mathsf{BPP}\}$ and let $\mathcal{D} \in \{\mathsf{NP}, \Sigma_2\mathsf{P}, \ldots, \Sigma_k\mathsf{P}, \ldots, \mathsf{PP}, \mathsf{PSPACE}, \mathsf{EXP}, \mathsf{NEXP}, \mathsf{EXP}^{\mathsf{NP}}\}$. Then $\mathcal{D} \not\subseteq \mathcal{C}$ implies that for every paddable $\mathcal{D}$-complete language L, there is a $\mathcal{C}$-constructive separation of $L \notin \mathcal{C}$.*[4] *Furthermore, $\oplus\mathsf{P} \not\subseteq \mathcal{C}$ implies that for every paddable $\oplus\mathsf{P}$-complete language L, there is a* BPP*-constructive separation of $L \notin \mathcal{C}$.*

[4] Throughout this paper when we say a language L is $\mathcal{D}$-complete, we mean it is $\mathcal{D}$-complete under polynomial-time many-one reductions. A language L is *paddable* if there is a deterministic polynomial-time algorithm that receives $(x, 1^n)$ as input, where string x has length at most $n-1$, and then outputs a string $y \in \{0,1\}^n$ such that $L(x) = L(y)$.

In other words, for many major separation problems such as $\mathsf{PP} \neq \mathsf{BPP}$, $\mathsf{EXP} \neq \mathsf{ZPP}$, and $\mathsf{PSPACE} \neq \mathsf{P}$, proving the separation automatically implies constructive algorithms that can produce counterexamples to any given weak algorithm. We find Theorem 1.2 to be mildly surprising: intuitively it seems that proving a constructive lower bound should be strictly stronger than simply proving a lower bound. (Indeed, we will later see other situations where making *known* lower bounds constructive would have major consequences!) Moreover, for separations beyond $\mathsf{P} \neq \mathsf{NP}$, the polynomial-time refuters guaranteed by Theorem 1.2 are producing hard instances for a problem that does not have short certificates, in general.

1.2 Unexpected Consequences of Making Some Separations Constructive

Given Theorem 1.2, we see that most of the major open problems surrounding polynomial-time lower bounds would yield constructive separations. Can *all* complexity separations can be made constructive? It turns out that for several "weak" lower bounds proved by well-known methods, making them constructive requires proving *other* breakthrough lower bounds!

Thus, there seems to be an algorithmic "dividing line" between many lower bounds we are able to prove, and many of the longstanding lower bounds that seem perpetually out of reach. The longstanding separation questions (as seen in Theorem 1.2) *require* a constructive proof: an efficient algorithm that can print counterexamples. Here we show that many lower bounds we are able to prove do not require constructivity, but if they could be made constructive then we would prove a longstanding separation! In our minds, these results confirm the intuition of Mulmuley that we should "go for explicit proofs" in order to make serious progress on lower bounds, especially uniform ones.

Constructive Separations for (Any) Streaming Lower Bounds Imply Breakthroughs. It is well-known that various problems are *unconditionally* hard for low-space randomized streaming algorithms. For example, from the randomized communication lower bound for the Set-Disjointness (DISJ) problem [KS92, Raz92, BJKS04], it follows that no $n^{1-\varepsilon}$-space randomized streaming algorithm can solve DISJ on $2n$ input bits.[5]

Clearly, every $n^{o(1)}$-space streaming algorithm for DISJ *must* fail to compute DISJ on some input (indeed, it must fail on many inputs). We show that efficient refuters against streaming algorithms attempting to solve *any* NP problem would imply a breakthrough lower bound on *general* randomized algorithms, not just streaming algorithms.

Theorem 1.3. *Let $f(n) \geq \omega(1)$. For every language $L \in \mathsf{NP}$, a P^{NP}-constructive separation of L from uniform randomized streaming algorithms with $O(n \cdot (\log n)^{f(n)})$ time and $O(\log n)^{f(n)}$ space*[6] *implies* $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$.

Essentially every lower bound proved against streaming algorithms in the literature holds for a problem whose decision version is in NP. Theorem 1.3 effectively shows that if *any* of these lower bounds can be made constructive, even in a P^{NP} sense, then we would separate randomized polynomial time from $\mathsf{EXP}^{\mathsf{NP}}$, a longstanding open problem in complexity theory. A more constructive separation (with an algorithm in a lower complexity class than P^{NP}) would imply a stronger separation. We are effectively showing that the counterexamples printed by such a refuter algorithm must encode a function that is hard for *general* randomized algorithms.

[5] Recall in the DISJ problem, Alice is given an n-bit string x, Bob is given an n-bit string y, and the goal is to determine whether their inner product $\sum_{i=1}^{n} x_i y_i$ is nonzero.

[6] That is, for every such randomized streaming algorithm A, there is a P^{NP} refuter B such that $B(1^n)$ prints an input x of length n such that A decides whether $x \in L$ incorrectly, for infinitely many n.

Stronger lower bounds follow from more efficient refuters for DISJ against randomized streaming algorithms. At the extreme end, we find that uniform circuits refuting DISJ against randomized streaming algorithms would even imply $\mathsf{P} \neq \mathsf{NP}$.

Theorem 1.4. *Let $f(n) \geq \omega(1)$. A polylogtime-uniform-AC^0-constructive separation of* DISJ *from randomized streaming algorithms with $O(n \cdot (\log n)^{f(n)})$ time and $O(\log n)^{f(n)}$ space*[7] *implies* $\mathsf{P} \neq \mathsf{NP}$.

To recap, it is well-known that DISJ does not have randomized streaming algorithms with $O(n \cdot (\log n)^{f(n)})$ time and $O(\log n)^{f(n)}$ space, even for $f(n) \leq o(\log n / \log\log n)$, by communication complexity arguments. We are saying that, if (given the code of such an algorithm) we can efficiently construct hard instances of DISJ for that algorithm, then strong lower bounds follow. *That is, making communication complexity arguments constructive would imply strong unconditional lower bounds.*

Constructive Separations for One-Tape Turing Machines Imply Breakthroughs. Next, we show how making some rather old lower bounds constructive would imply a circuit complexity breakthrough. It has been known at least since Maass [Maa84] that nondeterministic one-tape Turing machines require $\Omega(n^2)$ time to simulate nondeterministic multitape Turing machines. However, those lower bounds are proved by non-constructive counting arguments. We show that if there is a P^{NP} algorithm that can produce bad inputs for a given one-tape Turing machine, then E^{NP} requires exponential-size circuits. This in turn would imply $\mathsf{BPP} \subseteq \mathsf{P}^{\mathsf{NP}}$, a breakthrough simulation of randomized polynomial time.

Theorem 1.5. *For every language L computable by a nondeterministic $n^{1+o(1)}$-time RAM, a P^{NP}-constructive separation of L from nondeterministic $O(n^{1.1})$-time one-tape Turing machines implies $\mathsf{E}^{\mathsf{NP}} \not\subset \mathsf{SIZE}[2^{\delta n}]$ for some constant $\delta > 0$.*

Constructive Separations for Query Lower Bounds Imply Breakthroughs. Now we turn to query complexity. Consider the following basic problem $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ for a parameter $\varepsilon < 1/2$.

> $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$: *Given an input $x \in \{0,1\}^n$, letting $p = \frac{1}{n}\sum_{i=1}^{n} x_i$, distinguish between the cases $p < 1/2 - \varepsilon$ or $p > 1/2 + \varepsilon$.*

This is essentially the "coin problem" [BV10]. It is well-known that every randomized query algorithm needs $\Theta(1/\varepsilon^2)$ queries to solve $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ with constant success probability (uniform random sampling is the best one can do). That is, any randomized query algorithm making $o(1/\varepsilon^2)$ must make mistakes on some inputs, with high probability. We show that constructing efficient refuters for this simple sampling lower bound would imply $\mathsf{P} \neq \mathsf{NP}$!

Theorem 1.6. *Let ε be a function of n satisfying $\varepsilon(n) \leq 1/(\log n)^{\omega(1)}$.*

- *If there is a polylogtime-uniform-AC^0-constructive separation of $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ from randomized query algorithms A using $o(1/\varepsilon^2)$ queries and $\mathrm{poly}(1/\varepsilon)$ time, then $\mathsf{NP} \neq \mathsf{P}$.*
- *If there is a polylogtime-uniform-NC^1-constructive separation of $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ from randomized query algorithms A using $o(1/\varepsilon^2)$ queries and $\mathrm{poly}(1/\varepsilon)$ time, then $\mathsf{PSPACE} \neq \mathsf{P}$.*

Note that $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ can be easily computed in NC^1. If for every randomized query algorithm A running in n^α time and making n^α queries for some $\alpha > 0$, we can always find inputs in NC^1 on which A makes mistakes, then would separate P from PSPACE.

[7] That is, for every such randomized streaming algorithm A, there is a polylogtime-uniform AC^0 circuit family $\{C_n\}$ such that A fails to solve DISJ on $2n$-bit inputs correctly on the output $C_n(1^n)$ for infinitely many n.

Constructive Separations for MCSP Against AC^0 Imply Breakthroughs. Informally, the Minimum Circuit Size Problem (MCSP) is the problem of determining the circuit complexity of a given 2^n-bit truth table. Recent results on the phenomenon of hardness magnification [OS18, MMW19, CJW19, CHO+20, CJW20] show that, for various restricted complexity classes $\mathcal{C}$:

- Strong lower bounds against $\mathcal{C}$ are known for explicit languages.
- Standard complexity-theoretic hypotheses imply that such lower bounds should hold also for MCSP (and its variants).
- However, actually proving that $\mathsf{MCSP} \notin \mathcal{C}$ would imply a breakthrough complexity separation.

In such situations, there is also often a slightly weaker lower bound against $\mathcal{C}$ that can be shown for MCSP, suggesting that we are quantitatively "close" to a breakthrough separation in some sense.

We show that a similar phenomenon holds for constructive separations. It is well known that versions of MCSP are not in AC^0 [ABK+02], but strongly constructive separations are not known. We show that strongly constructive separations would separate P from NP, and that they exist under a standard complexity hypothesis. Moreover, we show that slightly weaker constructive separations *do* exist, and the strong constructive separations we seek do hold for other hard problems such as Parity.

In the following, $\mathsf{MCSP}[f(n)]$ is the computational problem that asks whether a Boolean function on n bits, represented by its truth table, has circuits of size at most $f(n)$.

Theorem 1.7. *Let $f(n) \geq n^{\log(n)^{\omega(1)}}$ be any time-constructive super-quasipolynomial function. The following hold:*

1. *(Major Separation from Constructive Lower Bound) If there is a polylogtime-uniform $\mathsf{AC}^0[\text{quasipoly}]$ refuter for $\mathsf{MCSP}[f(n)]$ against every polylogtime-uniform AC^0 algorithm, then $\mathsf{P} \neq \mathsf{NP}$.*
2. *(Constructive Lower Bound Should Exist) If $\mathsf{PH} \not\subseteq \mathsf{SIZE}(f(n)^2)$, then there is a polylogtime-uniform-$\mathsf{AC}^0[\text{quasipoly}]$ refuter for $\mathsf{MCSP}[f(n)]$ against every polylogtime-uniform AC^0 algorithm.*
3. *(Somewhat Constructive Lower Bound) There is a polylogtime-uniform-$\mathsf{AC}^0[2^{\text{poly}(f(n))}]$ refuter for $\mathsf{MCSP}[f(n)]$ against every polylogtime-uniform AC^0 algorithm.*
4. *(Constructive Lower Bound for a Different Hard Language) There is a $\text{quasipoly}(N)$-size polylogtime-uniform-$\mathsf{AC}^0[\text{quasipoly}]$-list-refuter for Parity against every polylogtime-uniform AC^0 algorithm.*

Note that in item 3, the input size N to the problem is $N = 2^n$, hence $2^{\text{poly}(f(n))}$ is only slightly super-quasipolynomial in N.

Comparison with Theorem 1.2. It is very interesting to contrast Theorem 1.2 with the various theorems of this subsection. On the one hand, Theorem 1.2 tells us that many longstanding open problems in lower bounds would automatically imply constructive separations, when resolved. On the other hand, we see that extending simple and well-known lower bounds to become constructive would resolve other major lower bounds! Taken together, we view the problem of understanding which lower bounds can be made constructive as a significant key to understanding the future landscape of complexity lower bounds.

1.3 Certain Lower Bounds Cannot Be Made Constructive

Finally, we can give some negative answers to our Question 1. We show that for some hard functions, there are *no* constructive separations from *any* complexity classes. Specifically, we show (unconditionally or under plausible complexity conjectures) that there are no refuters for these problems against a trivial decision algorithm that *always returns the same answer* (zero, or one). Hence, there can be no constructive separations of these hard languages from any complexity class containing the constant zero or constant one function. (All complexity classes that we know of contain both the constant zero and one function.)

For a string $x \in \{0,1\}^*$, the t-time-bounded Kolmogorov complexity of x, denote by $\mathsf{K}^{\mathsf{t}}(x)$, is defined as the length of the shortest program prints x in time $t(|x|)$. We use $\mathsf{R}_{\mathsf{K}^{\mathsf{t}}}$ to denote the set of strings x such that $\mathsf{K}^{\mathsf{t}}(x) \geq |x| - 1$. Hirahara [Hir20] recently proved that for any super-polynomial $t(n) \geq n^{\omega(1)}$, $\mathsf{R}_{\mathsf{K}^{\mathsf{t}}} \notin \mathsf{P}$. We observe that this separation cannot be made P-constructive.

Proposition 1.8. *For any $t(n) \geq n^{\omega(1)}$, there is no P-refuter for $\mathsf{R}_{\mathsf{K}^{\mathsf{t}}}$ against the constant zero function.*

Since $\mathsf{R}_{\mathsf{K}^{\mathsf{t}}}$ is a function in EXP, it would be interesting to find functions in NP with no constructive separations.[8] We show that under plausible conjectures, such languages in NP exist.

Theorem 1.9. *The following hold:*

- *If $\mathsf{NE} \neq \mathsf{E}$, then there is a language in $\mathsf{NP} \setminus \mathsf{P}$ that does not have P refuters against the constant one function.*[9]
- *If $\mathsf{NE} \neq \mathsf{RE}$, then there is a language in $\mathsf{NP} \setminus \mathsf{P}$ that does not have BPP refuters against the constant one function.*[10]

Thus, under natural conjectures about exponential-time classes, there are some problems in NP with no constructive separations at all, not even against the trivial algorithm that always accepts.

1.4 Intuition

Let us briefly discuss the intuition behind some of our results. We will first focus on the results showing that constructive separations of known lower bounds would imply complexity breakthroughs, as we believe these are the most interesting of our paper.

Constructive Separations of Known Lower Bounds Imply Breakthroughs. Suppose for example we want to show that a constructive separation of SAT from quick low-space streaming algorithms implies $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$. The proof is by contradiction: assuming $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{BPP}$, we aim to construct a streaming algorithm running in $n(\log n)^{\omega(1)}$ time and $(\log n)^{\omega(1)}$ space which solves $\mathsf{3SAT}$ correctly on all instances produced by P^{NP} algorithms. Given a P^{NP} algorithm R, $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{BPP}$ implies $\mathsf{EXP}^{\mathsf{NP}} \subset \mathsf{P}_{/\mathsf{poly}}$, which further implies that the output of $R(1^n)$ must have circuit complexity at most $\operatorname{polylog}(n)$ (construed as a truth table).

Extending work of McKay, Murray, and Williams [MMW19], we show that $\mathsf{NP} \subset \mathsf{BPP}$ (implied by $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{BPP}$) implies there is an $n(\log n)^{\omega(1)}$ time and $(\log n)^{\omega(1)}$ space randomized

[8] Note that $\mathsf{R}_{\mathsf{K}^{\mathsf{t}}}$ is in $\mathsf{coNTIME}[t(n)]$, but it is likely not in coNP.

[9] Here, $\mathsf{E} = \mathsf{TIME}[2^{O(n)}]$, the class of languages decidable in (deterministic) $2^{O(n)}$ time, and NE is the corresponding nondeterministic class.

[10] Here, $\mathsf{RE} = \mathsf{RTIME}[2^{O(n)}]$, the class of languages decidable in randomized $2^{O(n)}$ time with one-sided error.

algorithm with one-sided error for finding a $\mathrm{polylog}(n)$-size circuit encoding the given length-n input, if such a circuit exists. So given any input $R(1^n)$ from a potential refuter R, our streaming algorithm can first compute a $\mathrm{polylog}(n)$-size circuit C encoding $R(1^n)$, and it construes this circuit C as an instance of the Succinct-3SAT problem. Since Succinct-3SAT $\in$ NEXP $=$ BPP, our streaming algorithm can solve Succinct-3SAT(C) in $\mathrm{polylog}(n)$ randomized time, which completes the proof.

For our results on constructive query lower bounds, we use ideas from learning theory. Set $\varepsilon \ll 1/\mathrm{poly}(\log n)$. Assuming PSPACE $=$ P, we want to show that for every n-bit string printed by an uniform NC^1 circuit C on the input 1^n, we can decide the $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ problem with $o(1/\varepsilon^2)$ randomized queries in $\mathrm{poly}(1/\varepsilon)$ time. (Then, any sufficiently constructive lower bound that $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ requires $\Omega(1/\varepsilon^2)$ queries would imply P $\neq$ PSPACE.) PSPACE $=$ P implies that for every uniform NC^1 circuit C, its output can be encoded by some $\mathrm{polylog}(n)$-size circuit D. Now, also assuming PSPACE $=$ P, this circuit D can be PAC-learned with error $\varepsilon/2$ and failure probability $1/10$ using only $\mathrm{poly}\log(n)/\varepsilon$ queries (and randomness). Let D' be the circuit we learnt through this process; it approximates D well enough that we can make $O(1/\varepsilon^2)$ random queries *to the circuit D', without querying D* in $\mathrm{poly}(1/\varepsilon, \log n)$ time, and return the majority answer as a good answer for the original n-bit answer. Such an algorithm only makes $\mathrm{polylog}(n)/\varepsilon \ll o(1/\varepsilon^2)$ queries to the original input and runs in $\mathrm{poly}(1/\varepsilon)$ time.

Constructive Separations for Uniform Complexity Separations. Next, we highlight some insights behind the proof of Theorem 1.2. The proof is divided into several different cases (Theorem 5.3, Theorem 5.5, and Theorem 5.7), and we will focus on the intuition behind Theorem 5.5, which applies to all complexity classes with a downward self-reducible complete language (such as PSPACE or Σ_kP).

We take the PSPACE vs. P problem as an example. Gutfreund, Shaltiel, and Ta-Shma [GST07] showed how to construct refuters for P $\neq$ NP, but their proof utilizes the search-to-decision reduction for NP-complete problems, and no such reduction exists for PSPACE. We show how a downward self-reduction can be used to engineer a situation similar to that of [GST07].

Let M be a downward self-reducible PSPACE-complete language and let A be a P algorithm. We also let D be a polynomial-time algorithm defining a downward-self reduction for M, so that for all but finitely many $n \in \mathbb{N}$ and $x \in \{0,1\}^n$,

$$D(x)^{M_{\le n-1}} = M(x). \tag{1}$$

That is, D can compute $M(x)$ given access to an M-oracle for all strings of length less than $|x|$. Our key idea is that (1) *also* defines M. Assuming the polynomial-time algorithm A cannot compute M, it follows that (1) does not always hold if M is replaced by A. In particular, the following NP statement is true for infinitely many n:

$$\exists x \in \{0,1\}^n \text{ such that } D(x)^{A_{\le n-1}} \neq A(x). \tag{2}$$

Now we use a similar approach as in [GST07]: we use A and a standard search-to-decision reduction to find the shortest string x^* so that (2) holds. If A fails to do so, we can construct a counterexample to the claim that A solves the PSPACE-complete language M similarly to [GST07]. If A finds such an x^*, then by definition $A(y) = M(y)$ for all y with $|y| \le |x^*| - 1$ and we have $A(x^*) \neq M(x^*)$ from (2), also a counterexample.[11]

[11] Note the argument above only finds a single counterexample; using a paddable PSPACE-complete language, one can adapt the above argument to find infinitely many counter examples, see the proof of Theorem 5.5 for details.

1.5 Organization

In Section 2 we introduce the necessary definitions and technical tools for this paper, as well as review other related work. In Section 3 we show that making known streaming and query lower bounds constructive implies major complexity separations, and prove Theorem 1.3 and Theorem 1.4. In Section 4 we show that certain constructive separations for MCSP imply breakthrough lower bounds such as $\mathsf{P} \neq \mathsf{NP}$, and prove Theorem 1.7. In Section 5 we study constructive separations for uniform classes and prove Theorem 1.2. In Section 6 we show that several hard languages do not have constructive separations from any complexity class, and prove Proposition 1.8 and Theorem 1.9. Finally, in Section 7 we conclude with some potential future work.

2 Preliminaries

2.1 Notation

We use $\widetilde{O}(f)$ as shorthand for $O(f \cdot \text{polylog}(f))$ throughout the paper. All logarithms are base-2. We use n to denote the number of input bits. We say a language $L \subseteq \{0,1\}^\star$ is $f(n)$-sparse if $|L_n| \leq f(n)$, where $L_n = L \cap \{0,1\}^n$. We assume knowledge of basic complexity theory (see [AB09, Gol08]).

2.2 Other Refuter Notions

For some of our results, it will be useful to generalize the notion of a refuter to allow the production of a *list* of strings, such that at least one of them is a counterexample.

Definition 2.1 (List-Refuters)**.** For a function $s\colon \mathbb{N} \to \mathbb{N}$, a language L and an algorithm A that fails to solve L, an s-size $\mathcal{D}$*-list-refuter* (where $\mathcal{D} \in \{\mathsf{P}, \mathsf{BPP}, \mathsf{ZPP}\}$) for L against A is a $\mathcal{D}$-algorithm B that, given input 1^n, prints a list of $s(n)$ strings $x_n^{(1)}, x_n^{(2)}, \ldots, x_n^{(s(n))}$ of lengths $n^{\Omega(1)}$, such that for infinitely many n, the following hold:

1. If $\mathcal{D} = \mathsf{P}$ there is an $i \in [s(n)]$ for which $A(x_n^{(i)}) \neq L(x_n^{(i)})$.
2. If $\mathcal{D} = \mathsf{BPP}$, with constant probability there exists $i \in [s(n)]$ for which $A(x_n^{(i)}) \neq L(x_n^{(i)})$.
3. If $\mathcal{D} = \mathsf{ZPP}$, then either the algorithm outputs "fail" or there exists $i \in [s(n)]$ for which $A(x_n^{(i)}) \neq L(x_n^{(i)})$, and the latter event happens with constant probability.

Refuters for Non-Uniform Models. We can also define refuters for circuit families. For a circuit class $\mathcal{C}$, we use $D_{L,\mathcal{C}}$ to denote the family $\{D_{L,A}\}_{A \in \mathcal{C}}$. We say a deterministic oracle algorithm R solves the search problem family $D_{L,\mathcal{C}}$ infinitely often, if for every $\{C_n\}_{n\in\mathbb{N}} \in \mathcal{C}$, there are infinitely many integers n such that $(1^n, R(1^n)) \in D_{L,\{C_n\}_{n\in\mathbb{N}}}$. We use $R^{\{\mathsf{desc}(C_n)\}_{n\in\mathbb{N}}}$ to denote that the oracle algorithm R gets access to the descriptions of the circuit family $\{C_n\}$, instead of only black box query access to it.

We can similarly generalize the above to randomized or zero-error randomized algorithms in the natural way.

Definition 2.2 (Refuters and Constructive Separations for Language L against Nonuniform Class $\mathcal{C}$)**.** For a language L, a $\mathcal{D}$ refuter R for L against circuit class $\mathcal{C}$ is a $\mathcal{D}$ algorithm solving $D_{L,\mathcal{C}}$ infinitely often. We also say that R gives a $\mathcal{D}$-constructive separation $L \notin \mathcal{C}$.

We can extend the above definitions to list-refuters by allowing the corresponding algorithm to output a (polynomial-size) candidate list instead of a single counterexample. And we say a P list-refuter R solves $D_{L,A}$ infinite often if for infinitely many n, there exists $a \in R(1^n)$ such that $(1^n, a) \in D_{L,A}$. One can also similarly define BPP or ZPP list-refuters.

Finally, for a list-refuter according to Definition 2.2, we say it is an oblivious list-refuter, if it does not need access to $\{\mathsf{desc}(C_n)\}_{n\in\mathbb{N}}$.

2.3 Definitions of MCSP and time-bounded Kolmogorov complexity

The Minimum Circuit Size Problem (MCSP) [KC00] and t-time-bounded Kolmogorov complexity (K^t) are studied in this paper. We recall their definitions.

Definition 2.3 (MCSP)**.** Let $s\colon \mathbb{N} \to \mathbb{N}$ satisfy $s(m) \geq m - 1$ for all m.
Problem: $\mathsf{MCSP}[s(m)]$.
Input: A function $f\colon \{0,1\}^m \to \{0,1\}$, presented as a truth table of $n = 2^m$ bits.
Decide: Does f have a (fan-in two) Boolean circuit C of size at most $s(m)$?

We will also consider search-MCSP, the search version of MCSP, in which the small circuit C must be output when it exists.

For a time bound $t\colon \mathbb{N} \to \mathbb{N}$, recall that the K^t complexity (t-time-bounded Kolmogorov complexity) of string x is the length of the shortest program which outputs x in at most $t(|x|)$ time.

Definition 2.4 ($\mathsf{R}_{\mathsf{K}^\mathsf{t}}$)**.** Let $t\colon \mathbb{N} \to \mathbb{N}$.
Problem: $\mathsf{R}_{\mathsf{K}^\mathsf{t}}$.
Input: A string $x \in \{0,1\}^n$.
Decide: Does x have $\mathsf{K}^\mathsf{t}(x)$ complexity at least $n - 1$?

2.4 Implications of Circuit Complexity Assumptions on Refuters

The following technical lemma shows that, assuming uniform classes have non-trivially smaller circuits, the output of a refuter may be assumed to have low circuit complexity. This basic fact will be useful for several proofs in the paper.

Lemma 2.5. *Let $s\colon \mathbb{N} \to \mathbb{N}$ be an increasing function. The following hold:*

1. *Assuming $\mathsf{E}^{\mathsf{NP}} \subset \mathsf{SIZE}[s(n)]$, then for every P^{NP} algorithm A such that $A(1^n)$ outputs n bits, it holds that $A(1^n)$ has circuit complexity at most $s(O(\log n))$.*

2. *Assuming $\mathsf{E} \subset \mathsf{SIZE}[s(n)]$, then for every P algorithm A such that $A(1^n)$ outputs n bits, it holds that $A(1^n)$ has circuit complexity at most $s(O(\log n))$.*

3. *Assuming $\mathsf{SPACE}[O(n)] \subset \mathsf{SIZE}[s(n)]$, then for every LOGSPACE algorithm A such that $A(1^n)$ outputs n bits, it holds that $A(1^n)$ has circuit complexity at most $s(O(\log n))$.*

Proof. In the following we only prove the first item, the generalization to the other two items are straightforward.

Consider the following function $f_A(n, i)$, which takes two binary integers n and $i \in [n]$ as inputs, and output the i-th bit of the output of $A(1^n)$. The inputs to f_A can be encoded in $O(\log n)$ bits in a way that all inputs (n, i) with the same n has the same length.

Since A is in P^{NP}, we have $f_A \in \mathsf{E}^{\mathsf{NP}}$. By our assumption and fix the first part of the input to f_A as n, it follows that $A(1^n)$ has circuit complexity at most $s(O(\log n))$. □

The following simple corollary of Lemma 2.5 will also be useful.

Corollary 2.6. *If* $\mathsf{E}^{\mathsf{NP}} \subset \mathsf{P}_{/\mathsf{poly}}$ *(*$\mathsf{E} \subset \mathsf{P}_{/\mathsf{poly}}$ *or* $\mathsf{SPACE}[O(n)] \subseteq \mathsf{P}_{/\mathsf{poly}}$*), then for every* P^{NP} *(*P *or LOGSPACE) algorithm* A *such that* $A(1^n)$ *outputs* n *bits, it holds that* $A(1^n)$ *has circuit complexity at most* $\mathrm{polylog}(n)$*.*

We also observe that $\mathsf{P} = \mathsf{NP}$ has strong consequences for polylogtime-uniform AC^0 circuits.

Lemma 2.7. *The following hold:*

1. *Assuming* $\mathsf{P} = \mathsf{NP}$*, then for every polylogtime-uniform* AC^0 *algorithm* A *such that* $A(1^n)$ *outputs* n *bits, it holds that* $A(1^n)$ *has circuit size complexity at most* $\mathrm{polylog}(n)$*.*
2. *Assuming* $\mathsf{P} = \mathsf{PSPACE}$*, then for every polylogtime-uniform* NC^1 *algorithm* A *such that* $A(1^n)$ *outputs* n *bits, it holds that* $A(1^n)$ *has circuit size complexity at most* $\mathrm{polylog}(n)$*.*

Proof. Let B be a polylogtime-uniform algorithm that, on the integer n (in binary) and $O(\log n)$-bit additional input, reports gate and wire information for an AC^0 circuit A_n. Consider the function $f(n, i)$ which determines the i-th output bit of the circuit A_n on the input 1^n, given n and i in binary. The function f is a problem in PH: given input of length $m = O(\log n)$, by existentially and universally guessing and checking gate/wire information (and using the $\mathrm{polylog}(n)$-time algorithm B to verify the information), the A_n of $n^{O(1)}$ size can be evaluated in $\Sigma_d\mathsf{TIME}[m^k]$ for a constant d depending on the depth of A_n, and a constant k depending on the algorithm B. Since $\mathsf{P} = \mathsf{NP}$, f is computable in P, i.e., f is in time at most αm^{α} for some constant α depending on k, d, and the polynomial-time SAT algorithm. Therefore f has a circuit family of size at most m^c for some fixed c, where $m = c \log n$. Thus the output of such a family always has small circuits.

The same argument applies if we replace AC^0 by NC^1 and replace PH by PSPACE. □

2.5 Other Related Work

As mentioned in the introduction, Kabanets [Kab00] defined and studied refuters in the context of derandomization. A primary result from that paper is that it is possible to simulate one-sided error polynomial time (RP) in zero-error subexponential time (ZPSUBEXP) on all inputs produced by refuters (efficient time algorithms that take 1^n and output strings of length n).[12] In other words, nontrivial derandomization is indeed possible when we only consider the outputs of refuters: there is **no** constructive separation of $\mathsf{RP} \not\subset \mathsf{ZPSUBEXP}$. This result contrasts nicely with some of our own, which show that if we could prove (for example) $\mathsf{EXP} = \mathsf{ZPP}$ holds with respect to refuters, then $\mathsf{EXP} = \mathsf{ZPP}$ holds unconditionally. (Of course this is a contrapositive way of stating our results; we don't believe that $\mathsf{EXP} = \mathsf{ZPP}$ holds!) Kabanets' work effectively shows that if $\mathsf{RP} \not\subseteq \mathsf{ZPSUBEXP}$ implied a *constructive separation* of $\mathsf{RP} \not\subseteq \mathsf{ZPSUBEXP}$, then $\mathsf{RP} \subseteq \mathsf{ZPSUBEXP}$ holds unconditionally (because there is no constructive separation of RP from ZPSUBEXP).

Chen, Jin, and Williams [CJW20] studied a notion of constructive proof they called *explicit obstructions*. Roughly speaking, an explicit obstruction against a circuit class $\mathcal{C}$ is a (deterministic) polynomial-time algorithm A outputting a list L_n of input/output pairs $\{(x_i, y_i)\}$ with distinct x_i, such that all circuits in $\mathcal{C}$ fail to be consistent on at least one input/output pair. Chen, Jin, and Williams show several "sharp threshold" results for explicit obstructions, demonstrating

[12] The exact statement involves an "infinitely-often" qualifier, which we omit here for simplicity. A version of the simulation that removes the restriction to refuters, with the addition of a small amount of advice, was given in [Wil16].

(for example) that explicit obstructions unconditionally exist for $n^{2-\varepsilon}$-size DeMorgan formulas, but if they existed for $n^{2+\varepsilon}$-size formulas then one could prove the breakthrough lower bound $\mathsf{EXP} \not\subset \mathsf{NC}^1$. In this work, we are considering a "uniform" version of this concept: instead of outputting a list of bad input/output pairs (that do not depend on the algorithm), here we only have to output one bad instance that depends on the algorithm given.

An additional motivation for studying constructive proofs comes from proof complexity and bounded arithmetic. A circuit lower bound for a language $L \in \mathsf{P}$ can naturally be expressed by a Π_2 statement S_n that says: "For all circuits C of a certain type, there exists x of length n such that $C(x) \neq L(x)$". In systems of bounded arithmetic such as Cook's theory PV_1 [Coo75] (formalizing poly-time reasoning) or Jeřábek's theory APC_1 [Jeř07] (formalizing probabilistic poly-time reasoning), a proof of S_n for infinitely many n immediately implies a constructive separation. The reason is that these theories have efficient witnessing: any proof of a Π_2 statement $\forall x \exists y R(x, y)$ (for R that can be expressed purely with bounded quantifiers and poly-time concepts) in these theories constructs an efficiently computable function f such that $R(x, f(x))$ holds. Here the function f plays the role of the refuter in a constructive separation. Therefore, situations in which constructive separations are unlikely to exist may provide clues about whether complexity lower bounds could be independent of feasible theories. Conversely, the constructiveness of a separation is a precondition for the provability of that separation in these feasible theories.[13]

Hardness Magnification. Another related line of work is hardness magnification [OS18, MMW19, OPS19, CHO+20]. This line of work shows how very minor-looking lower bounds actually hide the whole difficulty of P vs NP and related problems. However, one might say that those results simply illuminate large holes in our intuition: those minor-looking lower bounds are far more difficult to prove than previously believed. One has to be skeptical in considering hardness magnification as a viable lower bounds approach, because we really don't understand how difficult the "minor-looking" lower bounds actually are.

In this paper, in contrast, we are mainly focused on situations where we already *know* the lower bound holds (and can prove that in multiple ways), but we are striving to prove the known lower bound in a more constructive, algorithmic way. This sort of situation comes up routinely in applications of the probabilistic method, where an object we want can be constructed with randomness, but it is a major open problem to construct it deterministically. Our results indicate that there is a deep technical gap between the major complexity class separation problems, versus many lower bounds we know how to prove. The former type of lower bound problem automatically has constructive aspects built into it, while the latter type of lower bound requires a breakthrough in derandomization in order to be made constructive.

3 Constructive Separations for Streaming and Query Algorithms imply Breakthrough Lower Bounds

Streaming lower bounds and query complexity lower bounds are often regarded as well-understood, and certain lower bounds against one-tape Turing machines have been known for 50 years. In this section we show that surprisingly, making these separations constructive would imply breakthrough separations such as $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$ or even $\mathsf{P} \neq \mathsf{NP}$.

[13] We note, however, that these connections depend on the complexity classes being separated. A circuit lower bound for an NP problem does not have an obvious Π_2 formulation, so the efficient witnessing results mentioned above do not directly apply. More complicated witnessing theorems might still be relevant; we refer to [MP20] and the recent book on Proof Complexity by Krajíček [Kra19] for a more detailed discussion of these matters.

3.1 Making Most Streaming Lower Bounds Constructive Implies Breakthrough Separations

We show that if randomized streaming lower bounds for *any* language L in NP can be made constructive, even with a P^{NP} refuter, then $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$.

Reminder of Theorem 1.3. *Let $f(n) \geq \omega(1)$. For every language $L \in \mathsf{NP}$, a P^{NP}-constructive separation of L from uniform randomized streaming algorithms with $O(n \cdot (\log n)^{f(n)})$ time and $O(\log n)^{f(n)}$ space implies $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$.*

Remark 3.1. *Let $V(x,y)$ be a verifier for L, and assume that the witness length $|y|$ is at most $|x|$.*[14] *Then the randomized streaming algorithms considered in Theorem 1.3 can be further assumed to solve the search-version of L with one-sided error in the following sense: (1) A is also required to output a witness y when it decides $x \in L$ (2) whenever A outputs a witness y, we have $V(x,y) = 1$.*

We need the following lemma for solving search-MCSP, which adapts an oracle algorithm from [MMW19]. The original algorithm of [MMW19] has two-sided error: that is, when $x \notin \mathsf{MCSP}[s(n)]$, there is a small probability that the algorithm outputs an incorrect circuit. We modify their approach with a carefully designed checking approach so that the algorithm has only one-sided error.

Lemma 3.2 ([MMW19, Theorem 1.2], adapted)**.** *Assuming $\mathsf{NP} \subseteq \mathsf{BPP}$, for a time-constructive $s \colon \mathbb{N} \to \mathbb{N}$, there is a randomized streaming algorithm for* search-$\mathsf{MCSP}[s(n)]$ *on n-bit instances with $O(n \cdot s(n)^c)$ time and $O(s(n)^c)$ space for a constant c such that the following holds.*

- *If the input $x \in \mathsf{MCSP}[s(n)]$, the algorithm outputs a circuit C of size at most s computing x with probability at least $1 - 1/n$.*
- *If the input $x \notin \mathsf{MCSP}[s(n)]$, the algorithm always outputs NO.*

Alternatively, if we assume $\mathsf{NP} = \mathsf{P}$ instead, the above randomized streaming algorithm can be made deterministic.

Proof. We first recall the $\Sigma_3\mathsf{P}$ problem Circuit-Min-Merge introduced in [MMW19]; here, we will only consider the version with two given input circuits. In the following we identify the integer i from $[2^m]$ with the i-th string from $\{0,1\}^m$ (ordered lexicographically).

Circuit-Min-Merge$[s(n)]$

Input: Given two circuits C_1, C_2 on $m = \log n$ input bits and three integers $\alpha < \beta \leq \gamma \in [2^m]$.
Output: The lexicographically first circuit C' such that for all $\alpha \leq z \leq \beta - 1$, $C'(z) = C_1(z)$, and for all $\beta \leq z \leq \gamma$, $C'(z) = C_2(z)$. If there are no such circuits, it outputs an all-zero string.

Algorithm 1: Circuit-Min-Merge

Note that since $\mathsf{NP} \subseteq \mathsf{BPP}$, it follows that Circuit-Min-Merge is also in BPP. We can without loss of generality assume we have a BPP algorithm for Circuit-Min-Merge with error at most $1/n^3$.

After processing the first $p \in [2^m]$ bits of the input x, our streaming algorithm maintains a list of at most m circuits. Specifically, let $p = \sum_{k=0}^{m} a_k \cdot 2^k$ be the binary representation of

[14] That is, $x \in L$ if and only if there exists $y \in \{0,1\}^*$ such that $|y| \leq |x|$ and $V(x,y) = 1$.

p, for each $k \in [m]$. We maintain a circuit C_k that is intended to satisfy $C_k(z) = x_z$ for all $\sum_{\ell > k} a_\ell \cdot 2^\ell < z \leq \sum_{\ell \geq k} a_\ell \cdot 2^\ell$. Note that when $a_k = 0$, there is indeed no requirement on the circuit C_k and we can simply set it to a trivial circuit.

Now, suppose we get the $p+1$ bit of the input x. We update the circuit list via the following algorithm.

- We initialize D to be the linear-size circuit which outputs x_{p+1} on the input $p+1$, and outputs 0 on all other inputs.
- For k from 0 to m:
  - If $a_k = 1$, we set $D = \text{Circuit-Min-Merge}(C_k, D, \alpha, \beta, \gamma)$ with suitable α, β, γ, and set $a_k = 0$ and C_k to be a trivial circuit. We next check whether D is indeed the correct output of $\text{Circuit-Min-Merge}(C_k, D, \alpha, \beta, \gamma)$ by going through all inputs in $[\alpha, \gamma]$. We output NO and halt the algorithm immediately if we found D is not the correct output (if $\text{Circuit-Min-Merge}(C_k, D, \alpha, \beta, \gamma)$ outputs the all-zero string, we also output NO and halt the algorithm).
  - If $a_k = 0$, we set $C_k = D$, and halt the update procedure.

After we have processed the 2^m-bit of x, we simply output C_n. If $x \in \mathsf{MCSP}[s(n)]$, then by a simple union bound, with probability at least $1 - 1/n$, all calls to our BPP algorithm for Circuit-Min-Merge are answered correctly. In this case C_n is a correct algorithm computing the input x. If $x \notin \mathsf{MCSP}[s(n)]$, since we have indeed checked the output of all Circuit-Min-Merge calls, our algorithm will only output the circuit C_n if it is indeed of size at most $s(n)$ and computes x exactly. Since $x \notin \mathsf{MCSP}[s(n)]$ implies there is no such circuit C_n, our algorithm always outputs NO in this case.

For the running time, note that the above algorithm calls Circuit-Min-Merge at most $n \cdot \log n$ times on input of length $\widetilde{O}(s(n))$. Therefore calling Circuit-Min-Merge only takes $n \cdot \mathrm{poly}(s(n))$ time in total. Note that merging C_k and D takes $2^k \cdot \mathrm{poly}(s(n))$ time to verify the resulting circuit, but this only happens at most $n/2^k$ times. So the entire algorithm runs in $n \cdot \mathrm{poly}(s(n))$ time and $\mathrm{poly}(s(n))$ space as stated. □

Now we are ready to prove Theorem 1.3.

Proof of Theorem 1.3. The idea is to show that if $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{BPP}$ then we can construct a randomized streaming algorithm for $L \in \mathsf{NP}$ that "fools" all possible P^{NP} refuters. Interestingly, the assumption is used in three different ways: (1) to bound the circuit complexity of the outputs of P^{NP} algorithms, (2) to obtain a randomized streaming algorithm that finds a small circuit encoding the input, and (3) to get an efficient algorithm to find a small circuit encoding a correct witness when it exists.

Let $L \in \mathsf{NP}$, and $V(x, y)$ be a polynomial-time verifier for L. Assuming $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{BPP}$, we are going to construct a randomized streaming algorithm A, such that it solves L correctly on all possible instances which can be generated by a P^{NP} refuter.

Let B be an arbitrary P^{NP} refuter. First, by Corollary 2.6, $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{BPP} \subset \mathsf{P}_{/\mathsf{poly}}$ implies that for all $n \in \mathbb{N}$, $B(1^n)$ has a circuit complexity of $w(n) = \mathrm{polylog}(n)$.

Second, note that $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{BPP}$ also implies that $\mathsf{NP} \subseteq \mathsf{BPP}$. Let $f(n) \geq \omega(1)$ and $s(n) = (\log n)^{f(n)/c_1}$ for a sufficiently large constant $c_1 > 1$. By Lemma 3.2, we have a one-sided error randomized streaming algorithm A_{MCSP} for search-$\mathsf{MCSP}[s(n)]$ with running time $n \cdot s(n)^{O(1)}$ and space $s(n)^{O(1)}$. Since $w(n) \leq s(n)$, we apply A_{MCSP} to find an $s(n)$-size circuit C encoding $B(1^n)$.

Now, we have an $s(n)$-size circuit encoding the n-bit input $B(1^n)$, and we wish to solve the Succinct-L problem[15] on this circuit. Note that Succinct-L is a problem in NEXP.

$\mathsf{EXP}^{\mathsf{NP}} = \mathsf{BPP}$ implies $\mathsf{NEXP} \subset \mathsf{P}_{/\mathsf{poly}}$, so every Succinct-$L$ instance has a succinct witness with respect to the verifier V: this follows from the easy witness lemma of [IKW02]. Formally, there exists a universal constant $k \in \mathbb{N}$ such that, for every $s(n)$-size circuit D such that $\mathrm{tt}(D) \in L$, there exists an $s(n)^k$-size circuit E such that $V(\mathrm{tt}(D), \mathrm{tt}(E)) = 1$.

We consider the following problem:

> Given an $s(n)$-size circuit D with truth-table length n and an integer $i \in [\log(s(n)^k)]$, exhaustively try all circuits of size at most $s(n)^k$, find the first circuit E such that $V(\mathrm{tt}(D), \mathrm{tt}(E)) = 1$, and output the i-th bit of the description of E.

Note that the above algorithm runs in $2^{\mathrm{poly}(s(n))}$-time on $\mathrm{poly}(s(n))$-bit inputs, hence it is in EXP. Since $\mathsf{EXP} = \mathsf{BPP}$, this problem is also in BPP. Therefore there is a BPP algorithm which, given a Succinct-L instance D of size $s(n)$, outputs a description of a canonical circuit of size $s(n)^k$ which encodes a witness for input $\mathrm{tt}(D)$ with respect to verifier V.

Thus we obtain a randomized algorithm for L on all instances with $s(n)$-size circuits. When the witness for x has length at most $|x| = n$, the algorithm can take $n \cdot \mathrm{poly}(s(n))$ time to output the found witness, by outputting the truth-table of the circuit encoding the witness.

Setting c_1 to be large enough and putting everything together, we get the desired randomized streaming algorithm which solves all instances generated by P^{NP} refuters, which is a contradiction to our assumption. Therefore, it follows that $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$. □

3.2 Separating P and NP via Uniform-AC0-Constructive Separations

Now we discuss a different setting, in which the existence of particular refuters would even imply $\mathsf{P} \neq \mathsf{NP}$.

It is well-known (via communication complexity arguments) that DISJ does not have efficient streaming algorithms; in fact, any streaming algorithm must give incorrect answers on many inputs. So it is clear that counterexamples to DISJ exist, for every candidate streaming algorithm. But how efficiently can they be constructed? We show that the ability to construct counterexamples in uniform AC^0 would actually imply $\mathsf{P} \neq \mathsf{NP}$.

Reminder of Theorem 1.4. *Let $f(n) \geq \omega(1)$. A polylogtime-uniform-AC^0-constructive separation of* DISJ *from uniform randomized streaming algorithm with $O(n \cdot (\log n)^{f(n)})$ time and $O(\log n)^{f(n)}$ space implies* $\mathsf{P} \neq \mathsf{NP}$.

Proof. We prove the contrapositive. Assuming $\mathsf{P} = \mathsf{NP}$, we will show that there is an efficient streaming algorithm that solves all disjointness instances that are generated by polylogtime-uniform AC^0 circuit families.

From Lemma 2.7, we know that the output string of any polylogtime-uniform AC^0 circuit family has circuit size complexity at most $c(\log n)^c$.

Next, by Lemma 3.2 we know that $\mathsf{P} = \mathsf{NP}$ implies that search-MCSP on input strings with circuits of size $c(\log n)^c$ can be solved by a streaming algorithm in $n \cdot (\log n)^{kc}$ time and $O(\log n)^{kc}$ space for some k. Also assuming $\mathsf{P} = \mathsf{NP}$, DISJ on any n-bit input represented by a $c(\log n)^c$-size circuit can be solved in $ck(\log n)^{ck}$ time for some k; indeed, the "Succinct-DISJ" problem *given a*

[15] Here, we define "Succinct-L" to be: given a circuit C with ℓ input bits, decide whether $\mathrm{tt}(C) \in L$, where $\mathrm{tt}(C)$ is the truth table of C.

circuit C on $n+1$ inputs, does its truth table on 2^{n+1} inputs encode two 2^n-bit strings which are disjoint? is a coNP problem.

For every function $f(n) \geq \omega(1)$, we can therefore design a streaming algorithm for DISJ as follows. First, on an input x, the algorithm solves search-MCSP using $n \cdot (\log n)^{f(n)}$ time and $(\log n)^{f(n)}$ space to get an $O((\log n)^{f(n)})$ size circuit C encoding x. (We could simulate this by trying $f(n) = 1, 2, 4, 8, \ldots$, until we successfully obtain a circuit.) Then, we run a $(\log n)^{O(f(n))}$-time algorithm for Succinct-DISJ on the circuit C. This will correctly decide disjointness on all inputs x that are generated by a polylogtime-uniform AC^0 circuit family. □

3.2.1 Constructive Separations in Query Complexity

Finally we show certain uniform-AC^0-constructive separations in query complexity would imply $\mathsf{P} \neq \mathsf{NP}$.

Reminder of Theorem 1.6. *Let ε be a function of n satisfying $\varepsilon \leq 1/(\log n)^{\omega(1)}$.*

- *If there is a polylogtime-uniform-AC^0-constructive separation of $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ from randomized query algorithms A using $o(1/\varepsilon^2)$ queries and $\mathrm{poly}(1/\varepsilon)$ time, then $\mathsf{NP} \neq \mathsf{P}$.*
- *If there is a polylogtime-uniform-NC^1-constructive separation of $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ from randomized query algorithms A using $o(1/\varepsilon^2)$ queries and $\mathrm{poly}(1/\varepsilon)$ time, then $\mathsf{PSPACE} \neq \mathsf{P}$.*

Proof. Assuming $\mathsf{P} = \mathsf{NP}$, we will show that there is an efficient query algorithm that solves all $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ instances that are generated by polylogtime-uniform AC^0 circuit families.

From Lemma 2.7, if $\mathsf{P} = \mathsf{NP}$, then for every polylogtime-uniform AC^0 circuit family $\{C_n\}$, the n-bit output of $C_n(1^n)$ has circuit size $(c \log n)^c$ for some constant c. (The same size bound also holds for polylogtime-uniform NC^1 circuits, under the stronger assumption $\mathsf{P} = \mathsf{PSPACE}$.) Such a circuit can be PAC-learned with error $\varepsilon/2$ and failure probability $\delta = 1/10$ using $O(\varepsilon^{-1} \cdot ((c \log n)^c + \log(1/\delta)))$ samples (random queries) (see e.g. [MRT18, Theorem 2.5] on learning a finite class of functions). The learning algorithm achieving this sample complexity simply computes a minimum-size circuit that is consistent with all the observed samples.

Under the assumption of $\mathsf{P} = \mathsf{NP}$, this learning algorithm can be executed in $\mathrm{poly}(1/\varepsilon, \log n)$ time. In particular, the following problem is in the polynomial-time hierarchy:

> *Given a set $\{(x_i, y_i)\} \subset \{0,1\}^n \times \{0,1\}$, a positive integer s, and an index j, output the j-th bit of the lexicographically first circuit C of size at most s such that $C(x_i) = y_i$ for all i.*

Assuming $\mathsf{P} = \mathsf{NP}$ (or $\mathsf{P} = \mathsf{PSPACE}$) the above problem is in P. Therefore we can find a minimum-size circuit consistent with any given input/output sample in $\mathrm{poly}(\log n, 1/\varepsilon)$ time. Let D be the circuit we have learned.

Next, we decide $\mathsf{PromiseMAJORITY}_{n,\varepsilon/2}$ on the truth table of D, by computing its average output value on $\Theta(1/\varepsilon^2)$ uniform random inputs. This process takes $\mathrm{poly}(1/\varepsilon)$ time, and makes no queries to the original input string. Since the learned circuit D only has error $\varepsilon/2$ compared with the original input string, we can simply return the result as our answer to the original $\mathsf{PromiseMAJORITY}_{n,\varepsilon}$ problem. This algorithm has success probability $2/3$, sample complexity $o(1/\varepsilon^2)$, and time complexity at most $\mathrm{poly}(1/\varepsilon)$, since $1/\varepsilon \gg \mathrm{polylog}(n)$. □

3.3 Constructive Separations for One-Tape Turing Machines imply Breakthrough Lower Bounds

Maass [Maa84] showed that a one-tape nondeterministic Turing machine takes at least $\Omega(n^2)$ time to decide the language of palindromes $\mathsf{PAL} = \{x_n \cdots x_1 x_1 \cdots x_n \mid x_1, \ldots, x_n \in \{0,1\}^n, n \in \mathbb{N}\}$. This is a very basic lower bound that is often cited as a canonical application of communication complexity. In this subsection, we show that a constructive proof of this lower bound would imply a breakthrough circuit lower bound.

In fact, we will prove a much more general statement. We will also generalize the proof to show that for every language L computable by nondeterministic $n^{1+o(1)}$-time RAMs, a constructive proof that "L cannot be decided by $n^{1.1}$-size nondeterministic one-tape Turing machines" would yield uniformly-computable functions with exponential circuit complexity. That is, we would obtain major circuit lower bounds even from the task of distinguishing RAMs from one-tape Turing machines in a constructive way.

We begin by a simple lemma showing that nondeterministic one-tape Turing machines can solve PAL on inputs that have small circuits.

Lemma 3.3. *For every constant $\delta \in (0,1]$, there is a nondeterministic $n^{1+O(\delta)}$-time one-tape Turing machines solving* PAL *on every x with circuit complexity at most $|x|^{\delta}$.*

Proof. Let $\delta \in (0,1]$. Our nondeterministic (one-tape) Turing Machine M runs as follows:

> M guesses a circuit C of size n^{δ}, and checks that $C(i)$ equals the i-th input bit for all $1 \le i \le n$, which can be done in $n \cdot n^{O(\delta)}$ time by moving the head on the tape from the first input bit to the last, while storing the n^{δ}-size circuit C in the cells close to the current position of the head. Finally M checks that the string $C(1)C(2)\cdots C(n)$ is a palindrome by evaluating C on every i and $n-i$, in $n \cdot n^{O(\delta)}$ total time. M accepts the input on a guess C if and only if all checks are passed.

Observe that M recognizes PAL correctly on every string x with circuit complexity at most n^{δ}, and its running time is bounded by $n^{1+O(\delta)}$. □

Now we show that breakthrough separations follow from constructive proofs of lower bounds for PAL.

Theorem 3.4. *The following hold:*

- *A P^{NP}-constructive separation of* PAL *from nondeterministic $O(n^{1.1})$ time one-tape Turing machines implies $\mathsf{E}^{\mathsf{NP}} \not\subset \mathsf{SIZE}[2^{\delta n}]$ for some constant $\delta > 0$.*
- *A* P*-constructive separation of* PAL *from nondeterministic $O(n^{1.1})$ time one-tape Turing machines implies* E $\not\subset$ SIZE$[2^{\delta n}]$ *for some constant $\delta > 0$.*
- *A LOGSPACE-constructive separation of* PAL *from nondeterministic $O(n^{1.1})$ time one-tape Turing machines implies* PSPACE $\not\subset$ SIZE$[2^{\delta n}]$ *for some constant $\delta > 0$.*

Proof. We will only prove the first item; it is straightforward to generalize to the other two items. Let $\delta > 0$ be a small enough constant such that, by Lemma 3.3, there is an nondeterministic $O(n^{1.1})$-time one-tape Turing machine solving PAL correctly on inputs x with circuit complexity at most n^{δ}.

Now suppose there is a P^{NP} refuter for M: a polynomial-time algorithm A with an NP oracle, which on input 1^n outputs an n-bit string. Assuming that $\mathsf{E}^{\mathsf{NP}} \subset \mathsf{SIZE}[2^{\delta_1 n}]$ for a constant

$\delta_1 > 0$ that is small enough compared to δ, by Lemma 2.5 there is a circuit C of size at most $n^{O(\delta_1)} \leq n^{\delta}$ that on input (n,i) computes the i-th bit of $A(1^n)$. That is, the output of any such A on 1^n has circuit complexity at most n^{δ}. By construction, M will always decide $A(1^n)$ correctly, contradicting the assumption that A is a refuter. Hence, there must exist a constant $\delta > 0$ such that $\mathsf{E}^{\mathsf{NP}} \not\subset \mathsf{SIZE}[2^{\delta n}]$. □

We say a family of 3-SAT formulas $\{C_n\}_{n\in\mathbb{N}}$ such that C_n has $S(n)$ clauses is *strongly explicit*, if there is an algorithm A such that $A(n,i)$ outputs the i-th clause of C_n in $\operatorname{polylog}(S(n))$ time. We need the following efficient reduction from nondeterministic $T(n)$-time RAMs to $T(n) \cdot \operatorname{polylog}(T(n))$-size 3-SAT instances.

Lemma 3.5 ([Tou01, FLvMV05]). *Let M be a $T(n)$-time nondeterministic RAM. There exists a strongly explicit family of 3-SAT formulas $\{C_n\}_{n\in\mathbb{N}}$ of $T \cdot \operatorname{polylog}(T)$ size, such that for every $x \in \{0,1\}^n$, $M(x) = 1$ if and only if there exists y such that $C_n(x,y) = 1$.*

Now we are ready to generalize Theorem 3.4 to other problems.

Reminder of Theorem 1.5. *For every language L computable by a nondeterministic $n^{1+o(1)}$-time RAM, a P^{NP}-constructive separation of L from nondeterministic $O(n^{1.1})$-time one-tape Turing machines implies $\mathsf{E}^{\mathsf{NP}} \not\subset \mathsf{SIZE}[2^{\delta n}]$ for some constant $\delta > 0$.*

Proof. Let M_{RAM} be a nondeterministic $n^{1+o(1)}$-time RAM for L. We apply Lemma 3.5 to obtain a strongly explicit family of 3-SAT formulas $\{C_n\}_{n\in\mathbb{N}}$ with $n^{1+o(1)}$ size and $s = n^{1+o(1)}$ variables.

Let $\delta_1 > 0$ be a small enough constant, and consider the following nondeterministic (one-tape) Turing machine M:

> M guesses a circuit D of size n^{δ_1}, and checks that $D(i)$ equals the i-th input bit for all $1 \leq i \leq n$, which can be done in $n \cdot n^{O(\delta_1)}$ time by moving the head on the tape from the first input bit to the last, while storing the n^{δ_1}-size circuit D in the cells close to the current position of the head.
>
> Next, M guesses a circuit E of size n^{δ_1}, and accepts if and only if
>
> $$D(1), \ldots, D(n), E(1), \ldots, E(s-n)$$
>
> satisfies C_n. Note that this can be checked in $n^{1+O(\delta_1)}$ time by enumerating all $n^{1+o(1)}$ clauses in C_n and evaluating D and E to obtain the assignments to the corresponding variables.

We take δ_1 to be small enough so that the above machine M runs in $O(n^{1.1})$ time. Suppose there is a P^{NP} refuter B for L against M, and we further assume towards a contradiction that $\mathsf{E}^{\mathsf{NP}} \subset \mathsf{SIZE}(2^{\delta n})$ for all $\delta > 0$.

By Lemma 2.5, it follows that $B(1^n)$ has an n^{δ_1}-size circuit. It also follows that if $B(1^n) \in L$, then the lexicographically first string $y_n \in \{0,1\}^{s-n}$ such that $C_n(B(1^n), y_n)$ has an n^{δ_1}-size circuit. By Lemma 3.5, this means that M solves $B(1^n)$ correctly, a contradiction. Hence, we have that $\mathsf{E}^{\mathsf{NP}} \not\subset \mathsf{SIZE}(2^{\delta n})$ for some $\delta > 0$. □

We conclude this section with a remark on the proofs. In the proofs of Lemma 3.3 and Theorem 1.5, we can naturally view our constructions as *nondeterministic streaming algorithms* with total time $n^{1+O(\delta)}$ and space $n^{O(\delta)}$. Hence, both results apply to low-space nondeterministic algorithms equally well. We only state the generalization of Theorem 1.5 below.

Remark 3.6. *For every language L computable by a nondeterministic $n^{1+o(1)}$-time RAM, a P^{NP}-constructive separation of L from nondeterministic $O(n^{1.1})$-time $n^{0.1}$-space streaming algorithms implies $\mathsf{E}^{\mathsf{NP}} \not\subset \mathsf{SIZE}[2^{\delta n}]$ for some constant $\delta > 0$.*

This remark is stronger than Theorem 1.5, as any $(n \cdot t)$-time t-space nondeterministic streaming algorithm can be simulated by an $n \cdot \mathrm{poly}(t)$ time nondeterministic one-tape Turing machine (see, e.g., [CHMY21, Lemma 9]). However, we have chosen not to emphasize it because the model of "nondeterministic streaming" is less common.

4 Constructive Separations for MCSP Imply Breakthrough Lower Bounds

In this section we show that constructive separations for MCSP against uniform AC^0 imply breakthrough lower bounds. In particular, we prove Theorem 1.7 (restated below for convenience).

Reminder of Theorem 1.7. *Let $f(n) \ge n^{\log(n)^{\omega(1)}}$ be any time-constructive super-quasipolynomial function. The following hold:*

1. *(Major Separation from Constructive Lower Bound) If there is a polylogtime-uniform $\mathsf{AC}^0[\text{quasipoly}]$ refuter for $\mathsf{MCSP}[f(n)]$ against every polylogtime-uniform AC^0 algorithm, then $\mathsf{P} \neq \mathsf{NP}$.*
2. *(An Constructive Lower Bound Should Exist) If $\mathsf{PH} \not\subseteq \mathsf{SIZE}(f(n)^2)$, then there is a polylogtime-uniform-$\mathsf{AC}^0[\text{quasipoly}]$ refuter for $\mathsf{MCSP}[f(n)]$ against every polylogtime-uniform AC^0 algorithm.*
3. *(Somewhat Constructive Lower Bound) There is a P-uniform-$\mathsf{AC}^0[2^{\mathrm{poly}(f(n))}]$ refuter for $\mathsf{MCSP}[f(n)]$ against every polylogtime-uniform AC^0 algorithm.*
4. *(Constructive Lower Bound for a Different Hard Language) There is a $\text{quasipoly}(N)$-size polylogtime-uniform-$\mathsf{AC}^0[\text{quasipoly}]$-list-refuter for Parity against every polylogtime-uniform AC^0 algorithm.*

Throughout this section, we use N to refer to the size of a truth table of a Boolean function on $n = \log(N)$ bits.

To prove Theorem 1.7, we will heavily use known results about pseudo-random generators against AC^0.

Theorem 4.1 ([Nis91, Vio05]). *Let d be any positive integer. There is a pseudo-random generator $G = \{G_N\}, G_N \colon \{0,1\}^{\log(N)^{O(d)}} \to \{0,1\}^N$, such that for each N, the PRG G $1/N$-fools depth-d AC^0 circuits of size N. Moreover, G is computable by polylogtime-uniform-AC^0 circuits of size $\mathrm{poly}(N)$, and $G_N(z)$ has circuits of size $\mathrm{polylog}(N)$ for each seed z of length $\log(N)^{O(d)}$.*

Corollary 4.2 ([All01]). *Let $f(N) \ge \log(N)^{\omega(1)}$ be any time-constructive function such that $f(N) \le o(N/\log(N))$. Then $\mathsf{MCSP}[f(N)]$ is not in AC^0.*

Corollary 4.2 follows from Theorem 4.1 by observing that $\mathsf{MCSP}[f(N)]$ distinguishes the uniform distribution on N bits from the output of G_N, since every output of G_N is a YES instance of $\mathsf{MCSP}[f(N)]$, while a random string of length N is a NO instance with high probability. In fact, it follows that for $f(N)$ quite close to maximum, the AC^0 lower bounds are exponential (but with an inverse dependence in the exponent on the circuit depth), similar to known lower bounds for Parity.

First, we show that uniform AC^0 refuters for separations of MCSP from uniform AC^0 would solve the main open problem in complexity theory. This establishes the first item of Theorem 1.7. We find it more convenient here to state the size bound for MCSP in terms of the input size $N = 2^n$ than in terms of n.

Theorem 4.3 (Item (1) of Theorem 1.7)**.** *Let $f(N) \geq 2^{\log\log(N)^{\omega(1)}}$ be any time-constructive function such that $f(N) \leq o(N/\log(N))$. If there is a polylogtime-uniform-AC^0[quasipoly] refuter for $\mathsf{MCSP}[f(N)]$ against every polylogtime-uniform AC^0 algorithm, then $\mathsf{P} \neq \mathsf{NP}$.*

Proof. Assume that $\mathsf{P} = \mathsf{NP}$ and that there is a polylogtime-uniform-AC^0 refuter for $\mathsf{MCSP}[f(N)]$ against every polylogtime-uniform AC^0 algorithm. We derive a contradiction. Indeed, consider the trivial algorithm that always outputs YES. Consider the polylogtime-uniform-AC^0 refuter R for this trivial algorithm. Using the same argument as in the proof of Theorem 1.4, the refuter R always outputs a string x of circuit complexity $2^{\log\log(N)^{O(1)}}$. But such a string is a YES instance of $\mathsf{MCSP}[f(N)]$ since $f(N) \geq 2^{\log\log(N)^{\omega(1)}}$. This contradicts the assumption that R refutes the algorithm that always outputs YES. □

By inspecting the proof carefully, it can be seen that the conclusion above holds even if the hypothesis is that there is a quasipolynomial-size uniform AC^0 list-refuter running in quasipolynomial time.

Next, we show that if a certain natural circuit lower bound assumption holds for the Polynomial Hierarchy, we do get the strongly constructive separations we seek. We obtain these separations by using a win-win argument: for any uniform AC^0 algorithm, either the algorithm outputs NO with noticeable probability, in which case the refuter exploits a PRG whose range is supported on strings of low circuit complexity, or it outputs YES with noticeable probability, in which case the refuter exploits a PRG (obtained using our assumption) whose range is supported on strings of high circuit complexity. This establishes the second item of Theorem 1.7.

Theorem 4.4 (Item (2) of Theorem 1.7)**.** *Let $f(N) \geq 2^{\log\log(N)^{\omega(1)}}$ be any time-constructive function such that $f(N) \leq o(N/\log(N))$. If $\mathsf{PH} \not\subseteq \mathsf{SIZE}(f(N)^2)$, then there is a polylogtime-uniform-AC^0[quasipoly] refuter for $\mathsf{MCSP}[f(N)]$ against every polylogtime-uniform AC^0 algorithm.*

Proof. Let f be as in the statement of the theorem, $F \in \mathsf{PH}$ be such that $F \notin \mathsf{SIZE}(f(N)^2)$, and let A be a polylogtime-uniform-AC^0 algorithm. We construct a polylogtime-uniform-AC^0[quasipoly] refuter R against A.

Let G be the PRG from Theorem 4.1 where d is the depth of the uniform AC^0 algorithm A, and let $G' = \{G'_N\}$ be the generator from $\log(N)^{O(d)}$ bits to N bits defined by $G'_N(z) = G_N(z) \oplus y_N$ for each seed z, where y_N is the truth table of F on $\lceil \log(N) \rceil$ bits truncated to the first N bits. The refuter R outputs the lexicographically first string x in the range of G such that $A(x) = 0$, or in case such a string does not exist, the lexicographically first string x' in the range of G' such that $A(x') = 1$. We will show that either x or x' exists. Note that R can be implemented by polylogtime-uniform quasipolynomial-size AC^0 circuits, since both G and G' have quasi-polynomial sized range and can be computed in uniform AC^0 - this is true for G by Theorem 4.1 and it is true for G' because the truth table of any PH function on $\log(N)$ bits can be computed by uniform AC^0 circuits of size $\mathrm{poly}(N)$.

Since G $1/N$-fools depth-d AC^0 circuits and G' is a linear translate of the range of G, G' also $1/N$-fools depth-d AC^0 circuits. We show that there is either a string x in the range of G such that $A(x) = 0$ or a string x' in the range of G' such that $A(x') = 1$. A either outputs NO with probability at least $1/2$ on randomly chosen input of length N, or it outputs YES with probability at least $1/2$. In the first case, since G $1/N$-fools A, there is a string x in the range of G such that $A(x) = 0$. Moreover, since every string in the range of G is a YES instance of $\mathsf{MCSP}[f(N)]$ by Theorem 4.1, we have that x refutes that A solves $\mathsf{MCSP}[f(N)]$ correctly. In the second case, since G' $1/N$-fools A, there is a string x' in the range of G' such that $A(x') = 1$. Moreover, since $F \notin \mathsf{SIZE}(f(N)^2)$ and every string in the range of G has $\mathrm{polylog}(N)$ size circuits, it follows that every string in the range of G' is a NO instance of $\mathsf{MCSP}[f(N)]$. Thus A makes a mistake on x', implying that R is a correct refuter. □

We also show that slightly weaker constructive separations than desired do hold unconditionally. The argument is similar to the argument in the proof of Theorem 4.4, but since we do not use an assumption, we need to argue differently in the case where the algorithm we are refuting outputs YES with high probability. We do so by exploiting the sparsity of the language against which we are showing a lower bound. This establishes the third item of Theorem 1.7.

Theorem 4.5 (Item (3) of Theorem 1.7)**.** *Let $f(N) \geq \log(N)^{\omega(1)}$ be any time-constructive function such that $f(N) \leq o(N/\log(N))$. There is a P-uniform-$\mathsf{AC}^0[2^{\mathrm{poly}(f(N))}]$ refuter for $\mathsf{MCSP}[f(N)]$ against every polylogtime-uniform AC^0 algorithm.*

Proof. Given a polylogtime-uniform AC^0 algorithm A, we define a uniform AC^0 refuter R running in time $2^{\mathrm{poly}(f(N))}$. For any d, let G^d be the PRG from Theorem 4.1 corresponding to depth d, and let $G = G^d$ where d is the depth of the uniform AC^0 algorithm A. Let G' be the generator with seed length $\mathrm{poly}(f(N))$ obtained by truncating the output of $G^{d'}_{f(N)^c}$ to N bits, where d' and c are to be specified later. R works as follows. It outputs the lexicographically first string x in the range of G for which $A(x) = 0$, and if such an x does not exist, it outputs the lexicographically first string x' in the range of G' that is not a YES instance of $\mathsf{MCSP}[f(N)]$ for which $A(x') = 1$. We show that such an x' always exists in the case that x does not, and that moroeover A is a correct refuter. Since G and G' can be computed by uniform AC^0 circuits of size exponential in $\mathrm{poly}(f(N))$ and moreover the YES instances of $\mathsf{MCSP}[f(N)]$ can be enumerated by uniform AC^0 circuits of size exponential in $\mathrm{poly}(f(N))$, we have that the refuter can be implemented by uniform AC^0 circuits of size exponential in $\mathrm{poly}(f(N))$. The uniformity condition here is P rather than polylogtime since the refuter needs to be able to compute f.

Either A outputs NO with probability greater than $1/2$ on a uniformly chosen input of length N, or it does not. In the first case, since G $1/N$-fools A, there must be a string x in the range of G for which $A(x) = 0$. Moreover, since every string in the range of G has circuit complexity $\mathrm{polylog}(N) \ll f(N)$, we have that x is a YES instance of $\mathsf{MCSP}[f(N)]$, and hence the refuter correctly outputs an input on which A makes a mistake in this case.

Suppose A outputs YES with probability at least $1/2$. We define a uniform AC^0 algorithm A' running in time $2^{\mathrm{poly}(f(N))}$ as follows. A' first enumerates all YES instances of $\mathsf{MCSP}[f(N)]$. Note that there are at most $2^{\mathrm{poly}(f(N))}$ YES instances, and they can be enumerated by an AC^0 algorithm in time $2^{\mathrm{poly}(f(N))}$ time by running over all circuits of size at most $f(N)$ and guessing and checking their computations. A' checks if its input x' is in the list of YES instances of $\mathsf{MCSP}[f(N)]$ or not. If it is, it outputs NO, otherwise it runs A on x' and outputs the answer. A' can be implemented by polylogtime-uniform AC^0 circuits of size $2^{\mathrm{poly}(f(N)}$ and constant depth. Now, by choosing the parameters c and d' in the first para large enough so that G' $1/N$-fools A', we have that at least a $1/2 - 1/N^{\omega(1)}$ fraction of outputs x' of G' have $A'(x') = 1$, and hence there is a lexicographically first such output. Moreover, since A' outputs NO on all YES instances of $\mathsf{MCSP}[f(N)]$, it must be the case that x' is a NO instance of $\mathsf{MCSP}[f(N)]$ and hence that A' makes a mistake on x' when trying to solve $\mathsf{MCSP}[f(N)]$. □

Finally, we observe that the strongly constructive separations we seek do hold in the case of the well-known lower bound for Parity against AC^0. Indeed, in this case we actually get an oblivious list-refuter (a.k.a. an explicit obstruction). This establishes the fourth item of Theorem 1.7.

Theorem 4.6 (Item (4) of Theorem 1.7, [Ajt83, FSS84, Yao85, Hås86])**.** *For each integer d, Parity does not have depth-$(d+1)$ AC^0 circuits of size $2^{O(N^{1/d})}$.*

Theorem 4.7. *There is a $\mathrm{quasipoly}(N)$-size polylogtime-uniform-$\mathsf{AC}^0[\mathrm{quasipoly}]$-list-refuter for Parity against every polylogtime-uniform AC^0 algorithm.*

Proof. In fact, we show that there is an oblivious list-refuter R that outputs a quasipoly-size set of strings of length N. The list-refuter R simply outputs the set of all strings of the form $y0^{N-\log(N)^d}$ where $y \in \{0,1\}^{\log(N)^d}$. Suppose, for the sake of contradiction, that there is a uniform AC^0 algorithm A that correctly solves Parity on all strings output by R. Then we can compute Parity by circuits of size $2^{O(m^{1/d})}$ on input y of length m as follows: pad y to length $2^{m^{1/d}}$ by suffixing it with zeroes, then run A on the padded string. This contradicts the lower bound of Theorem 4.6. □

5 Most Conjectured Uniform Separations Can Be Made Constructive

In this section we show many uniform separations imply corresponding refuters. We will prove Theorem 1.2 (restated below).

Reminder of Theorem 1.2. *Let $\mathcal{C} \in \{\mathsf{P}, \mathsf{ZPP}, \mathsf{BPP}\}$ and let $\mathcal{D} \in \{\mathsf{NP}, \Sigma_2\mathsf{P}, \dots, \Sigma_k\mathsf{P}, \dots, \mathsf{PP}, \mathsf{PSPACE}, \mathsf{EXP}, \mathsf{NEXP}, \mathsf{EXP}^{\mathsf{NP}}\}$. Then $\mathcal{D} \not\subseteq \mathcal{C}$ implies that for every paddable $\mathcal{D}$-complete language L, there is a $\mathcal{C}$-constructive separation of $L \notin \mathcal{C}$.*[16]

Furthermore, $\oplus\mathsf{P} \not\subseteq \mathcal{C}$ implies that for any paddable $\oplus\mathsf{P}$-complete language L, there is a BPP-constructive separation of $L \notin \mathcal{C}$.

We will prove the case of $\mathcal{D} \in \{\mathsf{PSPACE}, \mathsf{EXP}, \mathsf{NEXP}, \mathsf{EXP}^{\mathsf{NP}}\}$ in Section 5.1, $\mathcal{D} \in \{\Sigma_k\mathsf{P}\}_{k\geq 1}$ in Section 5.2, and $\mathcal{D} \in \{\mathsf{PP}, \oplus\mathsf{P}\}$ in Section 5.3.

We first provide a definition that is easier to work with, which is the constant size version of the list-refuters of Definition 2.1 with some additional requirements.

Definition 5.1 (Constant-size list-refuters)**.** For a language L and an algorithm A that fails to solve L, a constant-size *$\mathcal{D}$-list-refuter* (where $\mathcal{D} \in \{\mathsf{P}, \mathsf{BPP}, \mathsf{ZPP}\}$) for L against A is a $\mathcal{D}$-algorithm B such that given input 1^n, prints a list of c strings $x_n^{(1)}, x_n^{(2)}, \dots, x_n^{(c)} \in \{0,1\}^*$, such that for infinitely many n, there exists $i \in [c]$ for which $A(x_n^{(i)}) \neq L(x_n^{(i)})$ when $\mathcal{D} = \mathsf{P}$, for a universal constant $c > 0$ (independent of n). (The cases for $\mathcal{D} = \mathsf{BPP}$ or ZPP are similarly defined as in Definition 2.1.) Moreover, for every $i \in [c]$, there is a strictly increasing polynomial $\ell^{(i)} \colon \mathbb{N} \to \mathbb{N}$ such that $|x_n^{(i)}| = \ell^{(i)}(n) \geq n$ for all integers n.

Remark 5.2. *A constant-size P list-refuter implies a P-refuter. For every $i \in [c]$, let $B^{(i)}$ denote the algorithm which prints $x_n^{(i)}$ on input $1^{\ell^{(i)}(n)}$. Observe that at least one of $B^{(1)}, B^{(2)}, \dots, B^{(c)}$ prints valid counterexamples for infinitely many n.*

Similarly, if a constant-size BPP (or ZPP) list-refuter is pseudo-deterministic (i.e., the refuter outputs the canonical list with $1 - o(1)$ probability), then the same argument also applies, and we can obtain a BPP (or ZPP) refuter.

[16] Throughout this paper when we say a language L is $\mathcal{D}$-complete, we mean it is $\mathcal{D}$-complete under polynomial-time many-one reductions. A language L is *paddable* if there is a deterministic polynomial-time algorithm that receives $(x, 1^n)$ as input, where string x has length at most $n-1$, and then outputs a string $y \in \{0,1\}^n$ such that $L(x) = L(y)$.

5.1 Refuters for PSPACE, EXP, and NEXP

We first consider the case when $\mathcal{D}$ is a complexity class from $\{\mathsf{PSPACE}, \mathsf{EXP}, \mathsf{NEXP}\}$. Our proof below generalizes the refuter construction of [DFG13] which only discussed the case of $\mathcal{D} = \mathsf{NEXP}$.

Let $(\exists\, \mathrm{poly}(n))\mathcal{D}$ denote the complexity class that contains languages L satisfying the following property: there exists a polynomial $p(n)$ and a language $L' \in \mathcal{D}$ such that for all strings x, $L(x) = 1$ if and only if there exists $y \in \{0,1\}^{p(|x|)}$ such that $L'(x,y) = 1$. Similarly, we define the complexity class $(\forall\, \mathrm{poly}(n))\mathcal{D}$.

Theorem 5.3. *Let $\mathcal{C} \in \{\mathsf{P}, \mathsf{BPP}, \mathsf{ZPP}\}$, and $\mathcal{D}$ be a complexity class such that $\mathcal{C} \subsetneq \mathcal{D}$. Suppose $\mathcal{D}$ satisfies $(\exists\, \mathrm{poly}(n))\mathcal{D} \subseteq \mathcal{D}$ and $(\forall\, \mathrm{poly}(n))\mathcal{D} \subseteq \mathcal{D}$.*

If $\mathcal{D} \not\subseteq \mathcal{C}$, then for every paddable $\mathcal{D}$-complete language L, there is a $\mathcal{C}$-constructive separation of $L \notin \mathcal{C}$.

Proof. We first consider the case of $\mathcal{C} = \mathsf{P}$. Let A be a polynomial-time algorithm that fails to solve L on input length n. For $b \in \{0,1\}$, define the language

$$G_A^{(b)} := \{(1^n, x) : \text{there exists } y \in \{0,1\}^n \text{ with prefix } x \text{, such that } L(y) = b, A(y) = 1 - b\}.$$

Observe that $G_A^{(1)} \in (\exists\, \mathrm{poly}(n))\mathcal{D} \subseteq \mathcal{D}$, $G_A^{(0)} \in (\exists\, \mathrm{poly}(n))\mathrm{co}\mathcal{D} \subseteq \mathrm{co}\mathcal{D}$. Define

$$G_A := G_A^{(0)} \cup G_A^{(1)} = \{(1^n, x) : \text{there exists } y \in \{0,1\}^n \text{ with prefix } x \text{, such that } L(y) \neq A(y)\}.$$

Since L is $\mathcal{D}$-complete, there is a polynomial-time procedure R^L that can decide G_A by making two queries to an oracle for L. Since L is paddable, we may assume the queries to the L-oracle always have length exactly $\ell(n)$, for some strictly increasing polynomial $\ell\colon \mathbb{N} \to \mathbb{N}$. If we let R query the algorithm A instead of the L-oracle, then on any $(1^n, x)$, either R^A solves $G_A(1^n, x)$ correctly, or A gives the incorrect answer on at least one of the queries.

Our list-refuter performs a search-to-decision reduction which repeatedly calls $R^A(1^n, x)$ and extends the prefix x one bit at a time. It either eventually finds a string $y \in \{0,1\}^n$ such that $L(y) \neq A(y)$, or detects the inconsistency of A's answers. The pseudocode of this list-refuter is presented in Algorithm 2.

```
• Initialize $x$ as an empty string
• For $i \leftarrow 1, 2, \ldots, n$:
    – If $R^A(1^n, x \circ 1) = 1$:
        * $x \leftarrow x \circ 1$
    – Else if $R^A(1^n, x \circ 0) = 1$:
        * $x \leftarrow x \circ 0$
    – Else:
        * Return all the queries sent to $A$ by $R^A(1^n, x)$, $R^A(1^n, x \circ 0)$, and $R^A(1^n, x \circ 1)$
• Return $x$ and all the queries sent to $A$ by $R^A(1^n, x)$
```

Algorithm 2: The list-refuter against A

To prove the correctness of this list-refuter, we suppose for contradiction that A could correctly solve every string in the list. Consider three cases according to the final length of x when the refuter terminates:

(1) $|x| = 0$. Then $(1^n, 1)$ and $(1^n, 0)$ are not in G_A, which is impossible, since A cannot solve L correctly on every n-bit input.

(2) $1 \le |x| < n$. Then $(1^n, x) \in G_A$, but $(1^n, x \circ 1)$ and $(1^n, x \circ 0)$ are not in G_A. This is also impossible.

(3) $|x| = n$. Then $(1^n, x) \in G_A$, meaning that $L(x) \neq A(x)$. But x is also in the list and A should solve x correctly, a contradiction.

Hence, A answers incorrectly on at least one string in the list returned by Algorithm 2.

The list contains at most six strings, each of which has length n or $\ell(n)$. By Remark 5.2, this constant-size list-refuter can be converted into a refuter.

Now we consider the case of $\mathcal{C} = \mathsf{BPP}$. Since $A \in \mathsf{BPP}$, by standard amplification[17], there is another BPP algorithm A' which decides the same language as A and has success probability $1 - 2^{-2n}$. Then, for a uniformly chosen random seed r, with $1 - 2^{-n}$ probability, $A'(\cdot, r)$ decides the same language as A on input length n. From this point, we may apply the same proof of the $\mathcal{C} = \mathsf{P}$ case to $A'(\cdot, r)$. Hence we have a BPP-refuter against A. If we further assume $A \in \mathsf{ZPP}$, then the refuter also has zero error. Note that our randomized refuters are pseudo-deterministic. □

Corollary 5.4. *Let $(\mathcal{C}, \mathcal{D})$ be a pair of complexity classes from*

$$\{\mathsf{P}, \mathsf{ZPP}, \mathsf{BPP}\} \times \{\mathsf{PSPACE}, \mathsf{EXP}, \mathsf{NEXP}, \mathsf{EXP}^{\mathsf{NP}}\}.$$

Assuming $\mathcal{D} \not\subseteq \mathcal{C}$, for every paddable $\mathcal{D}$-complete language L, there is a $\mathcal{C}$-constructive separation of $L \notin \mathcal{C}$.

Proof. Note that all pairs $(\mathcal{C}, \mathcal{D})$ satisfy the requirements in Theorem 5.3 (where the inclusion $(\forall \operatorname{poly}(n))\mathsf{NEXP} \subseteq \mathsf{NEXP}$ follows from concatenating the witnesses for every possibility in the universal quantifier). □

5.2 Refuters for NP and the Polynomial Hierarchy

Now we move to the case that $\mathcal{D} = \Sigma_k\mathsf{P}$ for an integer k.

Theorem 5.5 (Adaptation of [GST07])**.** *Let $\mathcal{C} \in \{\mathsf{P}, \mathsf{BPP}, \mathsf{ZPP}\}$. Suppose $\mathsf{NP} \subseteq \mathcal{D}$, and there is a $\mathcal{D}$-complete language M which is downward self-reducible.*

If $\mathcal{D} \not\subseteq \mathcal{C}$, then for every paddable $\mathcal{D}$-complete language L, there is a $\mathcal{C}$-constructive separation of $L \notin \mathcal{C}$.

Proof Sketch. Let A be any algorithm in $\mathcal{C}$. We will construct a refuter for L against A. Here we only prove the case of $\mathcal{C} = \mathsf{P}$. (For $\mathcal{C} \in \{\mathsf{BPP}, \mathsf{ZPP}\}$, we use the same proof as the $\mathcal{C} = \mathsf{P}$ case, and apply the amplification argument described at the end of the proof of Theorem 5.3.)

Since M is downward self-reducible, there is a polynomial-time procedure D such that for every $x \in \{0,1\}^m$, $M(x) = D^{M_{\le m-1}}(x)$. Let $p_n \colon \{0,1\}^{\le n} \to \{0,1\}^{q(n)}$ be a $\operatorname{poly}(n)$-time reduction such that $M(x) = L(p_n(x))$, where $q(n) \colon \mathbb{N} \to \mathbb{N}$ is some strictly increasing polynomial.

[17] We remark that [GST07] also studied the case where A does not have bounded probability gap, which we do not consider here.

For large enough n, there must exist an $x \in \{0,1\}^{\leq n}$ such that $A(p_n(x)) \neq M(x)$, since otherwise we would have a $\mathcal{C}$ algorithm that decides M, contradicting $\mathcal{D} \nsubseteq \mathcal{C}$. Hence, there is a string x of length $m \leq n$, such that

$$A(p_n(x)) \neq D^{O_{m-1}}(x), \text{ where } O_{m-1} := \{x \in \{0,1\}^{\leq m-1} : A(p_n(x)) = 1\}, \tag{3}$$

since otherwise the downward self-reducibility of M would imply $A(p_n(x)) = M(x)$ for all $x \in \{0,1\}^{\leq n}$. Let x^* be the shortest string x satisfying condition (3). Then $A(p_n(x^*)) \neq M(x^*) = L(p_n(x^*))$.

Observe that condition (3) can be checked in polynomial time, so such x^* can be found if we had an NP machine. Since A claims to decide an NP-hard language, we can try to find x^* by a search-to-decision reduction using A, which is the same as what we did in the proof of Theorem 5.3. If we find the string x^*, then we should add $p_n(x^*)$ to our list. The correctness of this list-refuter follows from the same argument in the proof of Theorem 5.3. □

The following Corollary follows immediately from Theorem 5.5 and the fact that $\Sigma_k\mathsf{P}$ has a downward self-reducible complete language $\Sigma_k\mathsf{SAT}$.

Corollary 5.6. *Let $(\mathcal{C}, \mathcal{D})$ be a pair of complexity classes from the following list*

$$\{\mathsf{P}, \mathsf{ZPP}, \mathsf{BPP}\} \times \{\Sigma_k\mathsf{P}\}_{k\geq 1}.$$

If $\mathcal{D} \nsubseteq \mathcal{C}$, then for every paddable $\mathcal{D}$-complete language L, there is a $\mathcal{C}$-constructive separation of $L \notin \mathcal{C}$.

5.3 Refuters for PP and Parity-P

Finally we prove Theorem 1.2 for the case $\mathcal{D} \in \{\mathsf{PP}, \oplus\mathsf{P}\}$.

Theorem 5.7. *Let $\mathcal{C} \in \{\mathsf{P}, \mathsf{BPP}, \mathsf{ZPP}\}$. If $\mathsf{PP} \nsubseteq \mathcal{C}$, then for every paddable PP-complete language L, there is a $\mathcal{C}$-constructive separation of $L \notin \mathcal{C}$.*

Proof. Let A be any $\mathcal{C}$-algorithm. We will construct a refuter for L against A. Here we only prove the case of $\mathcal{C} = \mathsf{P}$. (For $\mathcal{C} \in \{\mathsf{BPP}, \mathsf{ZPP}\}$, we use the same proof as the $\mathcal{C} = \mathsf{P}$ case, and apply the amplification argument described at the end of the proof of Theorem 5.3.)

We first review the well-known polynomial-time algorithm D^{PP} that solves #3SAT with the help of a PP oracle. Given a 3-CNF formula ϕ with n variables, let $c_n c_{n-1} \cdots c_0$ denote the number of satisfiable assignments of ϕ in binary, i.e.,

$$\#3\mathsf{SAT}(\phi) = \sum_{0\leq i\leq n} c_i \cdot 2^i,$$

where $c_i \in \{0,1\}$ for $i = 0, \ldots, n$. The algorithm computes the values of c_i in decreasing order of i: after $c_n, c_{n-1}, \ldots, c_{i+1}$ are determined, c_i is the truth value of the statement

$$\#3\mathsf{SAT}(\phi) \geq 2^i + \sum_{i+1\leq j\leq n} c_j \cdot 2^j,$$

which can be determined by the PP oracle. Hence D^{PP} can compute $\#3\mathsf{SAT}(\phi)$ using $n+1$ queries to a PP oracle. For all $0 \leq i < n$, the algorithm D^{PP} asked the query

$$\#3\mathsf{SAT}(\phi) \geq \sum_{i\leq j\leq n} c_j \cdot 2^j$$

to which the oracle answered 1, and it also asked the query

$$\mathsf{\#3SAT}(\phi) \geq 2^i + \sum_{i \leq j \leq n} c_j \cdot 2^j$$

to which the oracle answered 0.

Since A claims to decide a PP-complete language, we replace the PP oracle by A and try to use D^A to solve #3SAT on n variables. By padding, we assume the input strings received by A have length exactly $\ell(n)$, for some strictly increasing polynomial $\ell\colon \mathbb{N} \to \mathbb{N}$. The polynomial-time algorithm D^A cannot correctly solve #3SAT on all possible ϕ, since otherwise it would contradict the assumption that $\mathsf{PP} \not\subseteq \mathsf{P}$. Hence there exists a formula ϕ such that $D^A(\phi) \neq D^A(\phi_0) + D^A(\phi_1)$, where ϕ_b denotes the formula obtained by setting the first variable in ϕ to b. Since $\mathsf{NP} \subseteq \mathsf{PP}$, we can try to find such a ϕ by a search-to-decision reduction using A, analogously to the proof of Theorem 5.5 and Theorem 5.3. We either find such a ϕ, or detect inconsistency during the search and find a constant-size list that contains a counterexample.

Now suppose we have found such a ϕ with m variables satisfying $D^A(\phi) \neq D^A(\phi_0) + D^A(\phi_1)$. Then we know that A answered incorrectly on one of the $3(m+1)$ queries asked by D. In the following we show how to reduce the size of this list to $O(1)$.

Let $a_m \cdots a_0$, $b_m \cdots b_0$, $c_m \cdots c_0$ be the binary representation of $D^A(\phi_0), D^A(\phi_1)$, and $D^A(\phi)$, respectively (where $a_m = b_m = 0$). Since $D^A(\phi) \neq D^A(\phi_0) + D^A(\phi_1)$, we know

$$\sum_{0 \leq j \leq m} (c_j - a_j - b_j) \cdot 2^j \neq 0.$$

We assume $D^A(\phi) \leq 2^m$ (and similarly, $D^A(\phi_0), D^A(\phi_1) \leq 2^{m-1}$); otherwise A must have answered 1 to the query "$\mathsf{\#3SAT}(\phi) \geq S$" for some $S \geq 2^m + 1$, which is an obvious counterexample. Now consider two cases:

(1) $\sum_{0 \leq j \leq m} (c_j - a_j - b_j) \cdot 2^j \leq -1$. We know that A answered 0 to the query "$\mathsf{\#3SAT}(\phi) \geq 1 + \sum_{0 \leq j \leq m} c_j \cdot 2^j$", and answered 1 to the queries "$\mathsf{\#3SAT}(\phi_0) \geq \sum_{0 \leq j \leq m} a_j \cdot 2^j$" and "$\mathsf{\#3SAT}(\phi_1) \geq \sum_{0 \leq j \leq m} b_j \cdot 2^j$". Assuming that all three answers are correct, we have

$$\begin{aligned} 0 &= \mathsf{\#3SAT}(\phi) - \mathsf{\#3SAT}(\phi_0) - \mathsf{\#3SAT}(\phi_1) \\ &< (1 + \sum_{0 \leq j \leq m} c_j \cdot 2^j) - (\sum_{0 \leq j \leq m} a_j \cdot 2^j) - (\sum_{0 \leq j \leq m} b_j \cdot 2^j) \\ &= 1 + \sum_{0 \leq j \leq m} (c_j - a_j - b_j) \cdot 2^j \\ &\leq 0, \end{aligned}$$

a contradiction.

(2) $\sum_{0 \leq j \leq m} (c_j - a_j - b_j) \cdot 2^j > 0$. We know that A answered 1 to the query "$\mathsf{\#3SAT}(\phi) \geq \sum_{0 \leq j \leq m} c_j \cdot 2^j$", and answered 0 to the queries "$\mathsf{\#3SAT}(\phi_0) \geq 1 + \sum_{0 \leq j \leq m} a_j \cdot 2^j$" and "$\mathsf{\#3SAT}(\phi_1) \geq 1 + \sum_{0 \leq j \leq m} b_j \cdot 2^j$". Assuming that all three answers are correct, we have

$$\begin{aligned} 0 &= \mathsf{\#3SAT}(\phi) - \mathsf{\#3SAT}(\phi_0) - \mathsf{\#3SAT}(\phi_1) \\ &\geq (\sum_{0 \leq j \leq m} c_j \cdot 2^j) - (\sum_{0 \leq j \leq m} a_j \cdot 2^j) - (\sum_{0 \leq j \leq m} b_j \cdot 2^j) \\ &= \sum_{k \leq j \leq m} (c_j - a_j - b_j) \cdot 2^j \\ &> 0, \end{aligned}$$

a contradiction.

In either of the two cases, we obtain a list of three strings that contains at least one counterexample. This finishes our construction of the constant-size list-refuter, which can be converted into a refuter by applying Remark 5.2. □

Theorem 5.8. *Let $\mathcal{C} \in \{\mathsf{P}, \mathsf{BPP}, \mathsf{ZPP}\}$. If $\oplus\mathsf{P} \not\subseteq \mathcal{C}$, then for every paddable $\oplus\mathsf{P}$-complete language L, there is a* BPP*-constructive separation of $L \notin \mathcal{C}$.*

Proof Sketch. Let A be any algorithm in $\mathcal{C}$. We will construct a refuter for L against A. Here we only prove the case of $\mathcal{C} = \mathsf{P}$. (For $\mathcal{C} \in \{\mathsf{BPP}, \mathsf{ZPP}\}$, we use the same proof as the $\mathcal{C} = \mathsf{P}$ case, and apply the amplification argument described at the end of the proof of Theorem 5.3.)

The proof is similar to that of Theorem 5.7. Let R be a reduction from $\oplus$3SAT to L. Then there must exist a 3-CNF formula ϕ such that $A(R(\phi)) \neq A(R(\phi_0)) \oplus A(R(\phi_1))$, where ϕ_b denotes the formula obtained by setting the first variable in ϕ to b. If we can find such ϕ, then we immediately obtain three strings which contain a counterexample for A.

Our requirement for ϕ can be encoded as a SAT instance π. By the Valiant-Vazirani theorem [VV86] and the fact that $\mathsf{P}^{\oplus\mathsf{P}} = \oplus\mathsf{P}$ [PZ83], there is a polynomial-time reduction f with random seed r such that if $x \notin \mathsf{SAT}$, then $\Pr_r[f(x,r) \notin \oplus\mathsf{3SAT}] = 1$, and if $x \in \mathsf{SAT}$, then $\Pr_r[f(x,r) \in \oplus\mathsf{3SAT}] \geq 2/3$.[18] We pick a random seed r, and consider two cases:

- If $A(R(f(\pi,r))) = 1$, then we can use the downward self-reducibility of $\oplus$3SAT to perform a search-to-decision reduction using A (similar to the proof of Theorem 5.3). Either we find a satisfiable assignment for $f(\pi,r)$, or we detect that A's answers are inconsistent. In the first case, note that the reduction f of [VV86] is simple enough so that we can efficiently convert any satisfying assignment for $f(\pi,r)$ to a satisfying assignment for π, which can then be converted to a formula ϕ that satisfies the desired property $A(R(\phi)) \neq A(R(\phi_0)) \oplus A(R(\phi_1))$.
- If $A(R(f(\pi,r))) = 0$, then our refuter simply outputs $R(f(\pi,r))$. Observe that this string is indeed a counterexample for A if $f(\pi,r) \in \oplus\mathsf{3SAT}$, which happens with probability at least $2/3$. □

Remark: These refuters are non-black-box. Observe that all refuter constructions in this section do require access to the *code* of the algorithm A being refuted. (That is, our refuter constructions are not "black-box" in terms of the algorithm A.) Atserias [Ats06] constructed a black-box refuter for the separation $\mathsf{NP} \not\subseteq \mathsf{BPP}$, and it may be possible to improve our refuter constructions to be black-box as well. However, it seems challenging to use the techniques of [Ats06] for this, because he crucially relies on the $\mathsf{ZPP}^{\mathsf{NP}}$ learning algorithm for polynomial-size circuits [BCG+96]. It is unclear how one might prove P-constructive separations using such an algorithm.

6 Hard Languages With No Constructive Separations

In this section we show there are hard languages without constructive separation from any complexity class. We first observe there are no constructive separations for $\mathsf{R}_{\mathsf{K}^t}$ unconditionally.

[18] In more detail, Valiant-Vazirani says that there is a randomized Turing reduction from SAT to $\oplus$SAT such that a given formula x is reduced to a sequence of formulas $x_1, \ldots, x_{O(n)}$ which are called on $\oplus$SAT. We take the entire Turing reduction from SAT to $\oplus$SAT, with success probability increased to at least 2/3, and apply the fact that $\mathsf{P}^{\oplus\mathsf{P}} = \oplus\mathsf{P}$, to obtain a single $\oplus$SAT instance.

Reminder of Proposition 1.8. *For any $t(n) \geq n^{\omega(1)}$, there is no* P *refuter for* $\mathsf{R}_{\mathsf{K}^{\mathsf{t}}}$ *against the constant zero function.*

Proof. A P refuter for $\mathsf{R}_{\mathsf{K}^{\mathsf{t}}}$ against the constant zero function needs to output in $\mathrm{poly}(n)$ time an n-bit string y_n with K^{t} complexity at least $n-1$, for infinitely many integers n. But by the definition of K^{t} complexity, all these y_n can be computed in $\mathrm{poly}(n)$ time by a uniform algorithm given the input n of $\log n$ bits, hence $\mathsf{K}^{\mathsf{t}}(y_n) = O(\log n)$ for all n, a contradiction. □

Next we show that, under plausible conjectures, there are languages in $\mathsf{NP} \setminus \mathsf{P}$ with no constructive separations from any complexity class.

Reminder of Theorem 1.9. *The following hold.*

- *If* $\mathsf{NE} \neq \mathsf{E}$, *then there is a language in* $\mathsf{NP}\backslash\mathsf{P}$ *that does not have* P *refuters against the constant one function.*
- *If* $\mathsf{NE} \neq \mathsf{RE}$, *then there is a language in* $\mathsf{NP}\backslash\mathsf{P}$ *that does not have* BPP *refuters against the constant one function.*[19]

Proof. Assume $\mathsf{NE} \neq \mathsf{E}$, and let $L' \in \mathsf{NE}\backslash\mathsf{E}$. Suppose for some constant $c \geq 1$ there is a $2^{O(n)}$ time reduction $R\colon \{0,1\}^n \to \{0,1\}^{2^{cn}}$ such that $x \in L' \Leftrightarrow R(x) \in \mathsf{SAT}$.

We define a language L as following:

- For $m \in \mathbb{N}$, L is given the concatenated string
$$(t, w_0, w_1, \ldots, w_{2^m-1}, s) \in \{0,1\}^{2^m} \times \left(\{0,1\}^{2^{cm}}\right)^{2^m} \times \{0,1\}^{2^{c(m+1)}}$$
as input.

  Here, m is intended as the input length to the language L', t is interpreted as a potential truth table of L' on all m-bit inputs which needs to be verified, $w_0, \ldots, w_{2^m-1}$ are interpreted as potential witnesses for every m-bit inputs to L' to help the verification, and s is intended as an input to SAT.

- $L(t, w_0, w_1, \ldots, w_{2^m-1}, s) = 1$ if and only if all of the following conditions hold:
  1. For every $i \in \{0,1\}^m$ with $t_i = 1$, we have that $w_i \in \{0,1\}^{2^{cm}}$ is a correct witness of $R(i) \in \mathsf{SAT}$ (in particular, $i \in L'$).
  2. For every $i \in \{0,1\}^m$ with $t_i = 0$, we have $i \notin L'$.
  3. $s \notin \mathsf{SAT}$.

That is, L accepts the input $L(t, w_0, w_1, \ldots, w_{2^m-1}, s)$ if (1) t is the correct truth table of L' on all m-bit inputs and all the w_i are correct witnesses for the corresponding inputs to L' and (2) $s \notin \mathsf{SAT}$.

The first condition above means that every input accepted by L *must reveal the truth table of the language* L', which helps us to design an E (RE) algorithm for L' given a P (BPP) refuter for L. The second condition allows us to argue that if $\mathsf{P} \neq \mathsf{NP}$, then $L' \notin \mathsf{P}$.

The concatenated string has length $2^{\Theta(m)}$. We can verify the first condition in $2^{O(m)}$ time, and verify the other two conditions in $\mathsf{coNTIME}[2^{O(m)}]$, so $L \in \mathsf{coNP}$.

[19] Recall we have defined RE to be one-sided randomized time $2^{O(n)}$.

Claim 1. *If $L \in \mathsf{P}$, then $\mathsf{SAT} \in \mathsf{P}$.*

From Claim 1 we conclude $L \notin \mathsf{P}$, since otherwise it would imply $\mathsf{P} = \mathsf{NP}$ and consequently $\mathsf{E} = \mathsf{NE}$, contradicting our assumption. Hence, $\overline{L} = \{0,1\}^* \backslash L$ is a language in $\mathsf{NP} \backslash \mathsf{P}$.

We will show that $\overline{L}$ does not have P refuters against the constant one function. If there is such a refuter, then it must output in $2^{O(m)}$ time a string $(t, w_0, w_1, \ldots, w_{2^m-1}, s) \in L$. By the first two conditions in the definition of L, we have $t_i = L'(i)$ for all $i \in \{0,1\}^m$. Hence, we can use this refuter to decide L' on m-bit inputs in $2^{O(m)}$ time, contradicting $L' \notin \mathsf{E}$.

To prove the second statement of the theorem, we further assume $\mathsf{NE} \neq \mathsf{RE}$ and $L' \in \mathsf{NE} \backslash \mathsf{RE}$. Suppose $\overline{L}$ has a BPP refuter against the constant one function, which prints a string $(t, w_0, w_1, \ldots, w_{2^m-1}, s)$. With at least $2/3$ probability, the string is in L. On a given input $i \in \{0,1\}^m$, if $t_i = 1$ and w_i is a correct witness of $i \in L'$, then we return $L'(i) = 1$; otherwise, we return $L'(i) = 0$. This yields a one-sided error randomized algorithm that decides L' on m-bit inputs in $2^{O(m)}$ time, contradicting $L' \notin \mathsf{RE}$.

It remains to prove Claim 1.

Proof of Claim 1. Recall that $L' \in \mathsf{NE}$ and $R \colon \{0,1\}^n \to \{0,1\}^{2^{cn}}$ is a $2^{O(n)}$ time reduction such that $y \in L' \Leftrightarrow R(y) \in \mathsf{SAT}$. Assume $L \in \mathsf{TIME}[n^d]$. The recursive algorithm `Solve-SAT` (described in Algorithm 3) receives $m \in \mathbb{N}$ and $x \in \{0,1\}^{2^{cm}}$ as input, and outputs a pair $(\mathsf{SAT}(x), w)$, where $w \in \{0,1\}^{2^{cm}}$ is a correct witness if $\mathsf{SAT}(x) = 1$.

```
Solve-SAT($m, x$) :
  • If $m \leq O(1)$, then return the correct $(\mathsf{SAT}(x), w)$ in constant time
  • For $y \in \{0,1\}^{m-1}$:
      – Let $(t_y, w_y)$ := Solve-SAT($m-1, R(y)$)
  • Let answer := $\overline{L}(t, w_0, w_1, \ldots, w_{2^{m-1}-1}, x)$
  • If answer $= 1$, then find a correct witness $w$ of $x \in \mathsf{SAT}$ by a search-to-decision reduction
    which repeatedly calls $\overline{L}(t, w_0, \ldots, w_{2^{m-1}-1}, \cdot)$
  • Return (answer, $w$)
```

Algorithm 3: Solve-SAT

The correctness of `Solve-SAT` easily follows from the definition of L and an induction on m. The overall idea of this algorithm is to use $\overline{L}(t, w_0, \ldots, w_{2^{m-1}-1}, \cdot)$ as a SAT solver after we obtain the correct $t, w_0, \ldots, w_{2^{m-1}-1}$, which themselves can be found by solving smaller SAT questions.

To improve the running time of the algorithm, we implement `Solve-SAT` with memoization. That is, if (t_y, w_y) at the m-th level of the recursion is already computed, then later it can be directly accessed without recursively calling `Solve-SAT` again. Then, the total time of `Solve-SAT` is at most $\sum_{m' \leq m} 2^{m'-1} \cdot 2^{cm'} \cdot (2^{O(m')})^d \leq 2^{O(m)}$. Hence, we can solve $\mathsf{SAT}(x)$ in $\mathrm{poly}(|x|)$ time. □

This completes the proof of the overall theorem. □

7 Conclusion

Many interesting questions remain for future work. While we have given many examples of complexity separations that can automatically be made constructive, it is unclear how to extend our results to separations with complexity classes within P. For example, let L be a P-complete language. If L is not in uniform NC^1, does a P-constructive separation of L from uniform NC^1 follow? How about separations of P from LOGSPACE? Would establishing constructive separations in these lower complexity classes have any interesting consequences?

Note that there is no P-constructive separation of $\mathsf{MCSP}[s] \notin \mathsf{P}$ for super-polynomially large s, unless EXP requires super-polynomial size Boolean circuits. (A polynomial-time refuter for the trivial algorithm that always accepts, must print a hard function!) But do any interesting consequences follow from a constructive separation of *search versions* of MCSP from P? The same proof strategy (of applying the conjectured refuter for the trivial algorithm that always accepts) does not make sense in this case, as the only hard instances for search problems are YES instances.

It would also be interesting to examine which proof methods for circuit lower bounds can be made constructive. We list a few examples which should be particularly interesting:

(1) the $\widetilde{\Omega}(n^3)$ size lower bound against DeMorgan formulas for Andreev's function [Hås98, Tal14],

(2) the $\widetilde{\Omega}(n^2)$ size lower bound against formulas for Element-Distinctness [Nec66],

(3) $\mathsf{AC}^0[p]$ size-depth lower bounds via the approximation method [Raz87, Smo87].

Chen, Jin, and Williams [CJW20] showed that constructing corresponding explicit obstructions for (1) and (2) above would imply $\mathsf{EXP} \not\subset \mathsf{NC}^1$, but it is unclear whether one can get a P-constructive separation without implying a major breakthrough lower bound.

We remark that as shown in [CJW20], most lower bounds proved by random restrictions *can* be made constructive, by constructing an appropriate pseudorandom restriction generator. [CJW20] explicitly constructed an oblivious list-refuter for parity against subquadratic-size formulas, and we remark that a similar oblivious list-refuter for parity against polynomial-size AC^0 circuits follows from the pseudorandom restriction generator for AC^0 of [GW14].

Finally, it would be interesting to consider constructive separations against *non-uniform* algorithms. Should we expect a proof of $\mathsf{NP} \not\subset \mathsf{P}/\mathrm{poly}$ or $\mathsf{NEXP} \not\subset \mathsf{P}/\mathrm{poly}$ to imply a refuter of some kind? In such a setting, one would presumably need to feed the code of the non-uniform algorithm to the polynomial-time algorithm as part of its input (the algorithm should get the non-uniform code as advice, one way or another).

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[ABK+02] Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35(6):1467–1493, 2006. Preliminary version in FOCS'02.

[Ajt83] Miklós Ajtai. Sigma-formulae on finite structures. *Ann. Pure Appl. Log.*, 24(1):1–48, 1983.

[All01] Eric Allender. When worlds collide: Derandomization, lower bounds, and kolmogorov complexity. In *Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science (FST TCS 2001)*, volume 2245 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2001.

[Ats06] Albert Atserias. Distinguishing SAT from polynomial-size circuits, through black-box queries. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006)*, pages 88–95. IEEE Computer Society, 2006.

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1):2:1–2:54, 2009.

[BCG$^+$96] Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon. Oracles and queries that are sufficient for exact learning. *J. Comput. Syst. Sci.*, 52(3):421–433, 1996.

[BGS75] Theodore P. Baker, John Gill, and Robert Solovay. Relativizations of the $\mathsf{P} = ?\mathsf{NP}$ question. *SIAM J. Comput.*, 4(4):431–442, 1975.

[BJKS04] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.

[BV10] Joshua Brody and Elad Verbin. The coin problem and pseudorandomness for branching programs. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 30–39, 2010.

[CHMY21] Mahdi Cheraghchi, Shuichi Hirahara, Dimitrios Myrisiotis, and Yuichi Yoshida. One-tape turing machine and branching program lower bounds for MCSP. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021*, pages 23:1–23:19, 2021.

[CHO$^+$20] Lijie Chen, Shuichi Hirahara, Igor Carboni Oliveira, Ján Pich, Ninad Rajgopal, and Rahul Santhanam. Beyond natural proofs: Hardness magnification and locality. In *11th Innovations in Theoretical Computer Science Conference, ITCS*, pages 70:1–70:48, 2020.

[CJW19] Lijie Chen, Ce Jin, and R. Ryan Williams. Hardness magnification for all sparse NP languages. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1240–1255, 2019.

[CJW20] Lijie Chen, Ce Jin, and R. Ryan Williams. Sharp threshold results for computational complexity. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 1335–1348. ACM, 2020.

[CLW20] Lijie Chen, Xin Lyu, and R. Ryan Williams. Almost-everywhere circuit lower bounds from non-trivial derandomization. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1–12. IEEE, 2020.

[Coo75] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing, 1975*, pages 83–97. ACM, 1975.

[DFG13] Shlomi Dolev, Nova Fandina, and Dan Gutfreund. Succinct permanent is *NEXP*-hard with many hard instances. In *Algorithms and Complexity, 8th International Conference, CIAC 2013. Proceedings*, volume 7878 of *Lecture Notes in Computer Science*, pages 183–196. Springer, 2013.

[FLvMV05] Lance Fortnow, Richard J. Lipton, Dieter van Melkebeek, and Anastasios Viglas. Time-space lower bounds for satisfiability. *J. ACM*, 52(6):835–865, 2005.

[FS16] Lance Fortnow and Rahul Santhanam. New non-uniform lower bounds for uniform classes. In *31st Conference on Computational Complexity (CCC 2016)*, 2016.

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984.

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.

[GST07] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. If NP languages are hard on the worst-case, then it is easy to find their hard instances. *Computational Complexity*, 16(4):412–441, 2007.

[GW14] Oded Goldreich and Avi Wigderson. On derandomizing algorithms that err extremely rarely. In *Symposium on Theory of Computing, STOC 2014*, pages 109–118. ACM, 2014.

[Hås86] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 6–20. ACM, 1986.

[Hås98] Johan Håstad. The shrinkage exponent of de Morgan formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998.

[Hir20] Shuichi Hirahara. Unexpected hardness results for kolmogorov complexity under uniform reductions. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 1038–1051. ACM, 2020.

[IK20] Christian Ikenmeyer and Umangathan Kandasamy. Implementing geometric complexity theory: on the separation of orbit closures via symmetries. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 713–726. ACM, 2020.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002.

[Jeř07] Emil Jeřábek. Approximate counting in bounded arithmetic. *J. Symb. Log.*, 72(3):959–993, 2007.

[Kab00] Valentine Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *J. Comput. Syst. Sci.*, 63(2):236–252, 2001. A preliminary version appeared in CCC'00.

[KC00] Valentine Kabanets and Jin-yi Cai. Circuit minimization problem. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 73–79, 2000.

[Kra19] Jan Krajíček. *Proof complexity*, volume 170. Cambridge University Press, 2019.

[KS92] Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992.

[Lev73] Leonid Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[Maa84] Wolfgang Maass. Quadratic lower bounds for deterministic and nondeterministic one-tape turing machines (extended abstract). In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 401–408. ACM, 1984.

[MMW19] Dylan M. McKay, Cody D. Murray, and R. Ryan Williams. Weak lower bounds on resource-bounded compression imply strong separations of complexity classes. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 1215–1225. ACM, 2019.

[MP20] Moritz Müller and Ján Pich. Feasibly constructive proofs of succinct weak circuit lower bounds. *Ann. Pure Appl. Log.*, 171(2), 2020.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, second edition, 2018.

[Mul07] Ketan Mulmuley. Geometric complexity theory VI: the flip via saturated and positive integer programming in representation theory and algebraic geometry. *CoRR*, abs/0704.0229, 2007.

[Mul10] Ketan Mulmuley. Explicit proofs and the flip. *CoRR*, abs/1009.0246, 2010.

[Mul12] Ketan Mulmuley. The GCT program toward the *P* vs. *NP* problem. *Commun. ACM*, 55(6):98–107, 2012.

[Nec66] E. Neciporuk. On a boolean function. *Doklady of the Academy of the USSR*, 169(4):765–766, 1966.

[Nis91] Noam Nisan. Pseudorandom bits for constant depth circuits. *Comb.*, 11(1):63–70, 1991.

[OPS19] Igor Carboni Oliveira, Ján Pich, and Rahul Santhanam. Hardness magnification near state-of-the-art lower bounds. In *34th Computational Complexity Conference, CCC 2019*, pages 27:1–27:29, 2019.

[OS18] Igor Carboni Oliveira and Rahul Santhanam. Hardness magnification for natural problems. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 65–76, 2018.

[PZ83] Christos H. Papadimitriou and Stathis Zachos. Two remarks on the power of counting. In *Theoretical Computer Science, 6th GI-Conference, 1983, Proceedings*, volume 145 of *Lecture Notes in Computer Science*, pages 269–276. Springer, 1983.

[Raz87] Alexander A Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[Raz92] Alexander A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106(2):385–390, 1992.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987*, pages 77–82, 1987.

[Tal14] Avishay Tal. Shrinkage of de morgan formulae by spectral techniques. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 551–560, 2014.

[Tou01] Iannis Tourlakis. Time-space tradeoffs for SAT on nonuniform machines. *J. Comput. Syst. Sci.*, 63(2):268–287, 2001.

[Vio05] Emanuele Viola. The complexity of constructing pseudorandom generators from hard functions. *Comput. Complex.*, 13(3-4):147–188, 2005.

[VV86] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[Wil16] R. Ryan Williams. Natural proofs versus derandomization. *SIAM J. Comput.*, 45(2):497–529, 2016.

[Yao85] Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *26th Annual Symposium on Foundations of Computer Science, 1985*, pages 1–10, 1985.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il