1 Lower bounds on all input lengths

Recall that the conclusion of the algorithmic method looks like $\mathcal{NEXP} \not\subset \mathcal{P}/\text{poly}$ or $\mathcal{NP} \not\subset \mathcal{SIZE}[n^k]$. In particular, it says that every small circuit family does not compute the hard function, or equivalently that there are infinitely many input lengths on which the circuit family errs.

One may hope for a better result, asserting that every small circuit family fails on all input lengths (except, at most, finitely many). This expectation makes intuitive sense, because for natural functions, it seems odd to have input-length-specific optimizations that matter (i.e., save significant time), are not efficiently computable (i.e., they pop-up only in the non-uniform model), and exist only for some input lengths. Moreover, a lower bound on all input lengths matters when deriving algorithms from circuit lower bounds, e.g. for pseudorandom generators [NW94; IW97].

We will now see a flavor of the algorithmic method that deduces lower bounds on all input lengths, at the cost of deducing a lower bound in $\mathcal{E}^{\mathcal{NP}}$ instead of $\mathcal{NEXP}$. The point isn't to educate you about "infinitely often" vs "almost always" specifically (although that is interesting), but rather to rehash the algorithmic method from another perspective, and to introduce an interesting notion called refuters.

Theorem 1 ([CLW20]). *If $\mathsf{CAPP}_{1,1/2}$ for n-bit circuits of size $2^{\epsilon \cdot n}$ can be solved in time $2^{(1-\epsilon)\cdot n}$ (for some $\epsilon > 0$), then there is a problem in $\mathcal{E}^{\mathcal{NP}}$ that is hard on all but finitely many input lengths for circuits of size $2^{\delta \cdot n}$, for some $\delta = \delta(\epsilon) > 0$.*

1.1 The bottleneck: Non-deterministic time hierarchy

Revisiting the proof of the algorithmic method, the loss appears when deducing witness lower bounds (i.e., we deduce that easy witnesses don't exist, but only for infinitely many input lengths). We do this because to get a contradiction to the $\mathcal{NTIME}$ hierarchy, we need an algorithm that works on all input length. Indeed, the $\mathcal{NTIME}$ hierarchy only asserts that every algorithm fails infinitely often, and thus to contradict it we need an algorithm that works on all input lengths.

Open Problem 1. *Prove that $\mathcal{NTIME}[t^2]$ is hard for $\mathcal{NTIME}[t]$ on all input lengths (except, at most, finitely many), for any time bound t.*

It's useful to recall the indirect diagonalization proof of the $\mathcal{NTIME}$ hierarchy [SFM78; Žák83].[1] We define a hard unary problem L by designing a machine M_L as follows.

Associate every t-time machine M with infinitely many intervals of input lengths, such that the intervals are disjoint.[2] In each interval, starting with input length n_0, for

[1] This is the standard proof often taught in textbooks and courses. An earlier proof by Cook [Coo73] is also interesting, and an accessible overview appears on my website. Cook's proof also only gives an infinitely-often separation.

[2] That is, all intervals for M are pairwise-disjoint, and they are also pairwise-disjoint from all intervals defined for any other machine M'.

each $i = 1, ..., T = 2^{3t(n_0)}$ the machine M_L on input 1^i mimics the behavior of M on input 1^{i-1}. On input 1^{n_T} the machine M_L is deterministic, simulates $M(1^{n_0})$ with all witnesses, and answers the opposite of the value of $M(1^{n_0})$.

If M agrees with L on this interval, then M accepts 1^{n_0} iff it accepts 1^{n_1}, and so on until 1^{n_T}, which is a contradiction. Hence, for infinitely many intervals, there is some input in the interval on which M and M_L disagree.

Remark 2. *The input lengths on which failure happens can be exponentially far apart (i.e., failure at n_1 and n_2 only guarantees that $n_2 \leq 2^{O(n_1)}$), since intervals are of exponential length.*

1.2 An almost-always time hierarchy when the witnesses are small

Fortnow and Santhanam [FS16] proved an almost-always $\mathcal{NTIME}$ hierarchy against verifiers that receive only a short (sub-linear) witness.

Theorem 3 ([FS16; CLW20])**.** *For every time-constructible $n \leq T(n) \leq 2^{\text{poly}(n)}$ there is $L \in \mathcal{NTIME}[T] \setminus \texttt{i.o.}\mathcal{NTIMEGUESS}[o(T), n/10]$. Moreover, there is an algorithm that, when given 1^n and M that runs in time $o(T)$ and makes at most $n/10$ non-deterministic guesses, runs in time $\text{poly}(T)$, makes oracle queries to $\mathcal{NP}$, and outputs x such that $M(x) \neq L(x)$.*

The reason for stating the "moreover" part, which was proved in [CLW20], is not clear at the moment. We will explain it when we plug this result into the algorithmic method and see the challenges. For now, let us note that such an algorithm is called a refuter: fixing some hard problem L, the refuter gets a machine M that we know fails to compute L, and the refuter efficiently finds an input on which M fails. It is a good question which lower bounds can have refuters, and which cannot. We will revisit this question in the final presentation of Mustafa and William.

Proof. At a high-level, we perform indirect diagonalization again. However, for each machine A, instead of associating A with an interval $[n, ..., 2^{O(t)}]$ of input lengths and performing indirect diagonalization in the search-space $1^n, 1^{n+1}, ..., 1^{2^{O(t)}}$, for each and every input length $n \in \mathbb{N}$ we perform indirect diagonalization against A in a search space of size $2^{n/10}$ that is a subset of $\{0,1\}^n$. Details follow.

Fix A working in non-deterministic time $o(T)$ and with $n/10$ guesses. We build B that diagonalizes against A; then later we explain how to diagonalize against all A's simultaneously.

On input x, let w_x be the $(n/10)$-bit prefix of x. Then, B works as follows:

- If $A(0^n, w_x) = 1$, reject.
- If $x = 1^n$, accept.
- Otherwise, mimic A on input $x + 1$ (including guessing a witness), where "+" is interpreted lexicographically.

Assume tac that A and B agree on all n-bit inputs. There are two cases:

1. A accepts 0^n. Since A and B agree, B also accepts 0^n. By definition of B, it means that A accepts $0^n + 1$. Continuing this logic, B accepts all n-bit inputs. However, this means that A rejects all witnesses for 0^n, a contradiction.

2. A rejects 0^n. Since A and B agree, B also rejects 0^n. By definition of B, and since there are no witnesses causing A to accept 0^n, on all $x \neq 1^n$ we know that B mimics A on input $x + 1$, and hence A and B reject all n-bit inputs. This contradicts the definition of B on input 1^n.

To diagonalize against all A's simultaneously, parse $x = (\langle A \rangle, w_x, ...)$ where $|\langle A \rangle| = \log^*(n)$, and use the search-space of w_x's per each A separately.

Turning to the "moreover" part, how do we construct a refuter? First find out whether A accepts 0^n.

Case 1: A rejects 0^n. Consider the list

$$A(0^n), A(0^n + 1), ..., A(1^n), B(1^n) ,$$

and note that the first value in the list differs from the last value. Hence, there are two adjacent values in the list that differ. If these are the last two values, then A and B disagree on 1^n. Otherwise, $A(x) \neq A(x + 1)$ for some $x < 1^n$. Since B mimics A for every $x < 1^n$ (because there are no witnesses causing A to accept 0^n), we know that $B(x) = A(x + 1)$. Hence, $A(x) \neq B(x)$.

The refuter checks if $A(1^n) \neq B(1^n)$ using its $\mathcal{NP}$ oracle, and if so, it outputs 1^n. Otherwise, it uses binary search to find two adjacent values in the list that differ. The list is of size 2^n and the refuter has query access to it (using its $\mathcal{NP}$ oracle), hence the refuter can perform the binary search in time $\text{poly}(n)$.

Case 2: A accepts 0^n. Let x^* be the first input rejected by B, or equivalently the first input such that $A(0^n, w_{x^*}) = 1$. Consider the list

$$A(0^n), A(0^n + 1), ..., A(x^*), B(x^*) .$$

This list is of size at most $2^{n/10}$, and the first and last elements differ. Hence, there are two adjacent values that differ, and the refuter can find them using binary search (in time $\text{poly}(n)$ and with an $\mathcal{NP}$ oracle). If $A(x^*) \neq B(x^*)$ the refuter outputs x^*, and otherwise $A(x) \neq A(x + 1)$ for some $x < x^*$. But we know that $B(x) = A(x + 1) = 0$ (due to the minimality of x^*), and hence $B(x) \neq A(x)$ and the refuter outputs x. ■

1.3 Plugging into the algorithmic method

We want to plug Theorem 3 into the algorithmic method and deduce almost-always circuit lower bounds. The obvious challenge is that the problem from Theorem 3 is hard only against algorithms making a sub-linear number of guesses.

We handle this by choosing the parameters differently. Specifically, instead of starting from a hard problem $L \in \mathcal{NTIME}[T]$ for $T = 2^{O(n)}$, we start from a hard problem $L \in \mathcal{NTIME}[T]$ for $T(n) = n^{O_\epsilon(1)}$. Assume tac that the PCP verifier of [BGH+05] for L has witnesses of size $T^\delta < n/10$, where $\delta > 0$ is small enough.

Under this "easy witnesses" assumption, we decide L in $\mathcal{NTIMEGUESS}[o(t), n/10]$ (and derive a contradiction) as follows:

Given x,

- Guess a witness-circuit C_w of size $T^\delta < n/10$.
- Construct $C_{x,C_w}(r) = V^{C_w}(x, r)$, which is of size $\text{poly}(n) \cdot T^{O(\delta)} < 2^{\epsilon \cdot \ell}$, where $\ell = \log(T) + O(\log\log T) = O(\log n)$ and the inequality relied on T being large enough and on δ being small enough.
- Run the $\mathsf{CAPP}_{1,1/2}$ algorithm on C_{x,C_w}, in time $2^{(1-\epsilon)\cdot\ell} = o(T)$.

The analysis is identical to what we've seen before, and we deduce that the PCP verifier for L doesn't have witnesses of size $T^\delta = 2^{\Omega(\ell)}$.

First attempt. How to go from a witness lower bound to circuit lower bounds? Certainly, we don't want to derandomize the verifier in the lower bound $\mathcal{MA}/1 \not\subset \mathcal{SIZE}[n^k]$, because the latter lower bound only holds infinitely-often.

If we'd have used to previous parameter setting with $T = 2^n$, going from a witness lower bound to a circuit lower bound with an $\mathcal{NP}$ oracle is easy.[3] Specifically, consider the function $f(x, i) = w_i^*$ where w^* is the lex-first convincing witness for x (or all-zeroes, if no convincing witness exists), and note that f is computable in time $2^{O(n)}$ with an $\mathcal{NP}$ oracle (i.e., we can find the lex-first convincing witness bit-by-bit), and that f has circuit complexity $T^\delta = 2^{\Omega(n)}$ (because for some x^* that has no easy witnesses, the partial truth-table $f(x^*, \cdot)$ has hardness T^δ).

However, with the current parameter setting, this does not work anymore. The truth-table of f is of size more than 2^n, but the hardness is only $T^\delta = o(n)$. Indeed, this is inherent to the setting, because the witnesses are of sub-linear size $o(n)$, and so their hardness is necessarily sub-linear in n.

Resolving this using the refuter. Let's fix x to some good value x^* (i.e., such that x^* has no easy witnesses), and consider the function $f_{x^*}(i) = w_i^*$. This function has exponential hardness (i.e., the truth-table is of length $\tilde{O}(T)$ and has hardness T^δ), but how do we efficiently compute f in this case, i.e. how do we find x^*?

This is where the refuter comes in. We feed the refuter 1^n and the algorithm M from the proof above (i.e., M tries to speed-up the computation of the hard problem using easy witnesses), and the refuter finds an n-bit input x^* on which M fails. Such x^* has no easy witnesses, otherwise M would've succeeded.

[3] Indeed, we could have done this in prior proofs of the algorithmic method, but then we'd have obtained lower bounds in $\mathcal{E}^{\mathcal{NP}}$ instead of $\mathcal{NTIME}$.

Hence, we define $f(i) = w_i^*$ where w^* is the lex-first convincing witness for x^* that the refuter finds. This algorithm runs in time $\text{poly}(n)$ and uses an $\mathcal{NP}$ oracle (both for the refuter and to find w^*), and the truth-table that it computes is of size $\tilde{O}(T) = \text{poly}(n)$ and has exponential hardness T^δ. This yields a problem in $\mathcal{E}^{\mathcal{NP}}$ that has no circuits of size $2^{\Omega(n)}$.

References

[BGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. "Short PCPs Verifiable in Polylogarithmic Time". In: *Proc. 20th Annual IEEE Conference on Computational Complexity (CCC)*. 2005, pp. 120–134.

[CLW20] Lijie Chen, Xin Lyu, and R. Ryan Williams. "Almost-Everywhere Circuit Lower Bounds from Non-Trivial Derandomization". In: *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2020, pp. 1–12.

[Coo73] Stephen A. Cook. "A hierarchy for nondeterministic time complexity". In: *Journal of Computer and System Sciences* 7 (1973), pp. 343–353.

[FS16] Lance Fortnow and Rahul Santhanam. "New non-uniform lower bounds for uniform classes". In: *Proc. 31st Annual IEEE Conference on Computational Complexity (CCC)*. 2016, 19:1–19:14.

[IW97] Russell Impagliazzo and Avi Wigderson. "P = BPP if E requires exponential circuits: derandomizing the XOR lemma". In: *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*. 1997, pp. 220–229.

[NW94] Noam Nisan and Avi Wigderson. "Hardness vs. randomness". In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149–167.

[SFM78] Joel I. Seiferas, Michael J. Fischer, and Albert R. Meyer. "Separating Nondeterministic Time Complexity Classes". In: *J. ACM* 25.1 (1978), pp. 146–167. DOI: 10.1145/322047.322061. URL: https://doi.org/10.1145/322047.322061.

[Žák83] Stanislav Žák. "A Turing machine time hierarchy". In: *Theoretical Computer Science* 26.3 (1983), pp. 327–333.