Todo list

TFM Title

Félix Moreno Peñarrubia

Master's Thesis

Master's Degree in Advanced Mathematics and Mathematical Engineering

Facultat de Matemàtiques i Estadística, UPC

June 2024

Director: Albert Atserias

write title, abstract, keywords...

Abstract

TODO

Keywords: TODO

MSC2020 codes: TODO

Contents

1
Introduction

Write Chapter 1

1.1 Overview

High-level introduction to the topic at hand.

1.2 Preliminaries

Definitions of basic concepts we will use in the thesis.

1.3 Our Countributions

What is novel in this work.

2

Constructive Separations

Write Chapter 2

2.1 Basic Definitions

Basic definition of constructive separation as in [Che+24].

Definition 2.1. For a function $f\colon 0,1^* \to 0,1$ and an algorithm $\mathcal{A}$, a P-refuter for f against $\mathcal{A}$ is a deterministic polynomial time algorithm R that, given input 1^n, prints a string $x \in \{0,1\}^n$, such that for infinitely many n, $A(x) \neq f(x)$.

A BPP-refuter for f against $\mathcal{A}$ is a randomized polynomial time algorithm R that, given input 1^n, prints a string $x \in \{0,1\}^n$, such that for infinitely many n, $A(x) \neq f(x)$ with probability at least $2/3$.

A ZPP-refuter for f against $\mathcal{A}$ is a randomized polynomial time algorithm R that, given input 1^n, prints $x \in \{0,1\}^n \cup \{\perp\}$, such that for infinitely many n, either $x = \perp$ or $A(x) \neq f(x)$, and $x \neq \perp$ with probability at least $2/3$.

Definition 2.2. For $\mathcal{D} \in \{\mathsf{P}, \mathsf{BPP}, \mathsf{ZPP}\}$ and a class of algorithms $\mathcal{C}$, we say there is a $\mathcal{D}$-constructive separation of $f \notin \mathcal{C}$, if for every algorithm $\mathcal{A}$ computable in $\mathcal{C}$, there is a refuter for f against A that is computable in $\mathcal{D}$.

2.2 Main Results of [Che+24]

Briefly state the results of [Che+24], pointing to further discussion in the next chapters.

2.2.1 Conjectured Separations Can Be Made Constructive

Results about constructive separations for conjectured uniform separations.

2.2.2 Constructive Separations for Known Results Imply Breakthroughs

Results

State the results we will discuss in this thesis.

Theorem 2.3 (Theorem 1.6 from [Che+24])**.** *Let ϵ be a function of n satisfying $\epsilon \leq 1/(\log n)^{\omega(1)}$, and $1/\epsilon$ is a positive integer computable in poly$(1/\epsilon)$ time given n in binary.*

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

Theorem 2.4 (Theorem 3.4 from [Che+24])**.** *The following hold:*

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

- *If there is a* P*-constructive separation of* PAL *from nondeterministic* $O(n^{1.1})$ *time one-tape Turing machines, then* $\mathsf{E} \not\subset \mathsf{SIZE}[2^{\delta n}]$ *for some constant* $\delta > 0$*.*

- *If there is a* $\mathsf{LOGSPACE}$*-constructive separation of* PAL *from nondeterministic* $O(n^{1.1})$ *time one-tape Turing machines, then* $\mathsf{PSPACE} \not\subset \mathsf{SIZE}[2^{\delta n}]$ *for some constant* $\delta > 0$*.*

Lemmas About Circuit Complexity of Refuters

Review the lemma and implement the changes suggested by Atserias

Lemma 2.5. *Let* $T(n), s(n) \geq n$ *be functions and assume* $\mathsf{DTIME}(T(2^n)) \subseteq \mathsf{SIZE}(s(n))$*. Then, for every algorithm* R *running in time* $T(n)$ *and outputting* n *bits, the output of* $R(1^n)$ *can be computed by a circuit of size* $s(2\lceil \log n \rceil)$ *(which takes as input an index* $0 \leq i < n$ *and outputs the* i*-th bit of* $R(1^n)$*).*

Proof. Consider the function $f_R(m, i)$ which, given m and i in binary, outputs the i-th bit of $R(1^m)$. The inputs to f_R can be encoded in $2\lceil \log m \rceil$ bits so that the input size is the same for each input (m, i) with the same m. $f_R \in \mathsf{DTIME}(T(2^N))$, so f_R has circuits of size $s(N)$, where $N = 2\lceil \log m \rceil$ is the input size. We get the desired result by considering, for each n, the circuit for f_R with input size $2\lceil \log n \rceil$ which we modify by fixing the first $\lceil \log n \rceil$ input bits to the value of n. □

2.3 Additional Aspects of Refuters

Explain additional aspects such as list refuters, black-boxness/gray-boxness/explicit constructions, certifiability, "minimum refuting size vs. minimum counterexample size", etc. Mention results of [Ats06] and [BTW10].

3

Refuters for Query Algorithms

In this chapter, we study constructive separations in the query algorithm setting. In particular, we study the problem PromiseMAJORITY of distinguishing determining whether a binary string $x \in \{0,1\}^n$ has at least $(\frac{1}{2}+\epsilon)n$ ones or at most $(\frac{1}{2}-\epsilon)n$ ones, under the promise that it is one of the two cases.

Finish writing Chapter 3

We will prove the $\Omega(1/\epsilon^2)$ lower bound in the number of queries for PromiseMAJORITY, show a randomized constructive separation against algorithms using fewer queries, and prove and discuss Theorem 2.3, the result from [Che+24] showing that breakthrough lower bounds follow from sufficiently constructive separations of PromiseMAJORITY.

3.1 Lower Bounds Against Query Algorithms for PromiseMAJORITY

The obvious algorithm for PromiseMAJORITY consists in randomly sampling values and answering according to the majority of the values we have obtained. According to well-studied bounds on the binomial distribution, $\Theta(1/\epsilon^2)$ samples are necessary and sufficient to make the probability of error smaller than any fixed constant following this strategy.

In this section, we will prove the very intuitive result that this is essentially the best that can be done, that is, that there is no query algorithm making $o(1/\epsilon^2)$ queries that gives the right answer with small probability of error. This was proved in [CEG95] in the more general context of *samplers*: algorithms that compute the average of a real-valued function $f\colon \{0,1\}^n \to [0,1]$ by sampling some of the inputs.

The result that we will prove here is stronger than the one stated in [CEG95]: we will prove not just that such query algorithms require $\Omega(1/\epsilon^2)$ in the worst case in order to have a high success probability *for all* inputs; instead, we will prove that $\Omega(1/\epsilon^2)$ queries are required for *random* inputs, where the probability of success is over the randomness of both the input and the algorithm. This will allow us to obtain a constructive separation.

3.1.1 Proof of Lower Bounds for Random Inputs

Definition 3.1. A probability distribution $\mathcal{D}$ on $\{0,1\}^n$ has *homogeneous symmetry* if for every $y \in \{0,1\}^*$ and every pair of subsets $S, S' \subset [n]$ with $|S| = |S'| = |y|$, $\Pr_{x\leftarrow\mathcal{D}}[x[S] = y] = \Pr_{x\leftarrow\mathcal{D}}[x[S'] = y]$. That is, the probability $\Pr_{x\leftarrow\mathcal{D}}[x[S] = y]$ does not depend on the set S. Denote by $F_{\mathcal{D}}(y)$ the probability $\Pr_{x\leftarrow\mathcal{D}}[x[S] = y]$ for any S with $|S| = |y|$.

A probability distribution $\mathcal{D}$ on $\{0,1\}^n$ has *complementary symmetry* if for every $yin\{0,1\}^n$, $\Pr_{x\leftarrow\mathcal{D}}[x = y] = \Pr_{x\leftarrow\mathcal{D}}[x = \bar{y}]$, where $\bar{y}$ is the complementary vector of y.

A probability distribution $\mathcal{D}$ on $\{0,1\}^n$ is *symmetric* if it has homogeneous symmetry and it has complementary symmetry.

Fix n and ϵ, and let $X_\epsilon^+ = \{x \in \{0,1\}^n | w_1(x) \geq (\frac{1}{2} + \epsilon)n\}$ and $X_\epsilon^- = \{x \in \{0,1\}^n | w_1(x) \leq (\frac{1}{2} - \epsilon)n\}$. For $\mathcal{D}$ a probability distribution on $\{0,1\}^n$ for which the events $x \in X_\epsilon^+$ and $x \in X_\epsilon^-$ have nonzero probability (where x is the string of n bits sampled by the distribution), denote by $\mathcal{D}^+$ the probability distribution conditioned on $x \in X_\epsilon^+$, and by $\mathcal{D}^-$ the probability distribution conditioned on $x \in X_\epsilon^-$.

define w1, etc in introduction

Lemma 3.2. *Let $\mathcal{D}$ be a distribution on $\{0,1\}^n$ which has homogeneous symmetry.*

1. *The probability $F_\mathcal{D}(y)$ only depends on the number of zeros and ones of y: $F_\mathcal{D}(y) = F_\mathcal{D}(0^{w_0(y)}1^{w_1(y)})$. We will denote by $f_\mathcal{D}(a,b)$ the value of $F_\mathcal{D}(0^a1^b)$.*
2. *If the events $x \in X_\epsilon^+$ and $x \in X_\epsilon^-$ have nonzero probability (i.e. $\mathcal{D}^+$ and $\mathcal{D}^-$ are well-defined), the distributions $\mathcal{D}^+$ and $\mathcal{D}^-$ have homogeneous symmetry.*
3. *If $\mathcal{D}^+$ and $\mathcal{D}^-$ are well-defined and $a \leq b$, then $f_{\mathcal{D}^+}(b,a) \leq f_{\mathcal{D}^+}(a,b)$.*
4. *If $\mathcal{D}$ has complementary symmetry and $\mathcal{D}^+$ and $\mathcal{D}^-$ are well-defined, then $f_{\mathcal{D}^+}(a,b) = f_{\mathcal{D}^-}(b,a)$ for all $a,b \geq 0$.*

Find a way to phrase this without the awkward variable x?

Proof. Let us prove the first item. For $|y| = 1$ it is clear that the value of $F_\mathcal{D}(y)$ only depends the number of zeros and ones of y, since $y = 0$ or $y = 1$. Now we want to prove that for every y with $|y| \geq 2$, $F_\mathcal{D}(y) = F_\mathcal{D}(0^{w_0(y)}1^{w_1(y)})$. To prove that, it is enough to prove that we can perform "swaps" for any substring 10 in y, that is, that if $y = y_0||10||y_1$, then $F_\mathcal{D}(y_0||10||y_1) = F_\mathcal{D}(y_0||01||y_1)$. By a sequence of such swaps, we can always reach the string $0^{w_0(y)}1^{w_1(y)}$. This is proved as follows:

$$\begin{aligned}
F_\mathcal{D}(y_0||10||y_1) &= \Pr\left[x[S] = y_0||10||y_1\right] \\
&= \Pr\left[x[S] = y_0||10||y_1\right] + \Pr\left[x[S] = y_0||00||y_1\right] - \Pr\left[x[S] = y_0||00||y_1\right] \\
&= \Pr\left[x[S \setminus \{i\}] = y_0||0||y_1\right] - \Pr\left[x[S] = y_0||00||y_1\right] \\
&= \Pr\left[x[S \setminus \{j\}] = y_0||0||y_1\right] - \Pr\left[x[S] = y_0||00||y_1\right] \\
&= \Pr\left[x[S] = y_0||01||y_1\right] + \Pr\left[x[S] = y_0||00||y_1\right] - \Pr\left[x[S] = y_0||00||y_1\right] \\
&= F_\mathcal{D}(y_0||01||y_1),
\end{aligned}$$

where i and j are the indices of S corresponding to the 10 subsequence.

Second item:

$$\begin{aligned}
\Pr_{x\leftarrow\mathcal{D}^+}\left[x[S]=y\right] &= \Pr_{x\leftarrow\mathcal{D}}\left[x[S]=y|x\in X_\epsilon^+\right]\\
&= \sum_{\substack{z\in X_\epsilon^+\\ z[S]=y}} \Pr_{x\leftarrow\mathcal{D}}\left[x=z\right]\\
&= \sum_{\substack{z\in X_\epsilon^+\\ z[S]=y}} F_{\mathcal{D}}(0^{w_0(z)}1^{w_1(z)}) && \text{(first item on set } S=[n])\\
&= \sum_{\substack{z\in X_\epsilon^+\\ z[S']=y}} F_{\mathcal{D}}(0^{w_0(z)}1^{w_1(z)})\\
&= \Pr_{x\leftarrow\mathcal{D}^+}\left[x[S']=y\right].
\end{aligned}$$

Third item:

$$\begin{aligned}
f_{\mathcal{D}^+}(b,a) &= \Pr_{x\leftarrow\mathcal{D}^+}\left[x[[a+b]]=0^b1^a\right]\\
&= \frac{1}{\binom{n}{a+b}\binom{a+b}{b}} \sum_{S\in\binom{[n]}{a+b}} \sum_{\substack{y\in\{0,1\}^{a+b}\\ w_0(y)=b}} \Pr_{x\leftarrow\mathcal{D}^+}\left[x[S]=y\right] && \text{(homog. sym.)}\\
&= \frac{1}{\binom{n}{a+b}\binom{a+b}{b}} \sum_{S\in\binom{[n]}{a+b}} \sum_{\substack{y\in\{0,1\}^{a+b}\\ w_0(y)=b}} \sum_{\substack{z\in X_\epsilon^+\\ z[S]=y}} \Pr_{x\leftarrow\mathcal{D}^+}\left[x=z\right]\\
&= \frac{1}{\binom{n}{a+b}\binom{a+b}{b}} \sum_{z\in X_\epsilon^+} \binom{w_0(z)}{b}\binom{w_1(z)}{a} \Pr_{x\leftarrow\mathcal{D}^+}\left[x=z\right] && \text{(swap sums)}\\
&\leq \frac{1}{\binom{n}{a+b}\binom{a+b}{b}} \sum_{z\in X_\epsilon^+} \binom{w_0(z)}{a}\binom{w_1(z)}{b} \Pr_{x\leftarrow\mathcal{D}^+}\left[x=z\right] && (*)\\
&= f_{\mathcal{D}^+}(a,b),
\end{aligned}$$

where the inequality at $(*)$ uses that if $x\leq y$ and $a\leq b$, then $\binom{x}{a}\binom{y}{b}\geq\binom{x}{b}\binom{y}{a}$.

Fourth item:

$$\begin{aligned} f_{\mathcal{D}^+}(a,b) &= \Pr_{x\leftarrow\mathcal{D}}\left[x[[a+b]] = 0^a1^b | x\in X_\epsilon^+\right] \\ &= \frac{1}{\Pr_{x\leftarrow\mathcal{D}}\left[x\in X_\epsilon^+\right]} \sum_{\substack{y\in X_\epsilon^+ \\ y[[a+b]]=0^a1^b}} \Pr_{x\leftarrow\mathcal{D}}\left[x=y\right] \\ &= \frac{1}{\Pr_{x\leftarrow\mathcal{D}}\left[x\in X_\epsilon^-\right]} \sum_{\substack{y\in X_\epsilon^- \\ y[[a+b]]=1^a0^b}} \Pr_{x\leftarrow\mathcal{D}}\left[x=y\right] && \text{(comp. sym.)} \\ &= \Pr_{x\leftarrow\mathcal{D}}\left[x[[a+b]] = 1^a0^b | x\in X_\epsilon^-\right] = f_{\mathcal{D}^-}(b,a). \end{aligned}$$

□

Lemma 3.3. *Let $\mathcal{D}$ be a symmetric distribution supported on $X_\epsilon^+ \cup X_\epsilon^-$, and let $\mathcal{A}$ be a probabilistic query algorithm that always does t queries and solves* PromiseMAJORITY *with error probability δ on inputs sampled from $\mathcal{D}$. Then:*

$$\sum_{i=0}^{\lceil t/2\rceil-1} \binom{t}{i} f_{\mathcal{D}^+}(i, t-i) \leq \delta.$$

Proof. Denote by x the input, and let $Y = y_1 \dots y_t$ denote the random variable corresponding to the values of the t sampled points. Note that Y depends on x and also on the randomness over the execution of Y_r be the random variable corresponding to the sampled points where the randomness of $\mathcal{A}$ has been fixed to be r. Y_r only depends on the choice of x.

Let $P^+ = \Pr\left[\mathcal{A}(x) = 0 | x\in X_\epsilon^+\right]$ the probability that the algorithm errs conditioned on x having a majority of ones, and similarly let $P^- = \Pr\left[\mathcal{A}(x) = 1 | x\in X_\epsilon^-\right]$ be the probability that the algorithm errs conditioned on x having a majority of zeros.

We have:

$$P^+ = \sum_{y\in\{0,1\}^t} \Pr\left[Y = y | x\in X_\epsilon^+\right] \Pr\left[\mathcal{A}(x) = 0 | Y = y, x\in X_\epsilon^+\right].$$

Now note the following:

- Once y is fixed, the result of $\mathcal{A}$ does not depend on x, but only on y and r. Hence $\Pr_{x,r}\left[A(x) = 0 | Y = y, x \in X_\epsilon^+\right] = \Pr_r\left[\mathcal{A}(x) = 0 | Y = y\right]$. We denote this probability by $G_{\mathcal{A}}(y)$.
- Since $\mathcal{D}$ is symmetric, the event $Y = y$ does not depend on the randomness r of the algorithm and in fact $\Pr\left[Y = y | x \in X_\epsilon^+\right] = F_{\mathcal{D}^+}(y)$. This is because, fixing some randomness r in the algorithm, the event that the sequence of t sample points is equal to y is equivalent to the event that a subsequence of $x[S]$ of x of size t is equal to a particular permutation of $\sigma(y)$ of y. By Lemma 3.2 (1), this probability does not depend on S or σ.

Thus,

$$P^+ = \sum_{y \in \{0,1\}^t} F_{\mathcal{D}^+}(y) \cdot G_{\mathcal{A}}(y).$$

Similarly:

$$P^- = \sum_{y \in \{0,1\}^t} F_{\mathcal{D}^-}(y) \cdot (1 - G_{\mathcal{A}}(y)).$$

Adding them and applying Lemma 3.2, we get the desired result:

$$\begin{aligned}
\delta &\geq \frac{1}{2}\left(P^+ + P^-\right) \\
&= \frac{1}{2} \sum_{y \in \{0,1\}^t} F_{\mathcal{D}^+}(y) \cdot G_{\mathcal{A}}(y) + F_{\mathcal{D}^-}(y) \cdot (1 - G_{\mathcal{A}}(y)) \\
&\geq \frac{1}{2} \sum_{y \in \{0,1\}^t} \min\{F_{\mathcal{D}^+}(y), F_{\mathcal{D}^-}(y)\} \\
&= \frac{1}{2} \sum_{i=0}^{t} \binom{t}{i} \min\{f_{\mathcal{D}^+}(i, t-i), f_{\mathcal{D}^+}(t-i, i)\} && \text{(Lemma 3.2 (4))} \\
&\geq \sum_{i=0}^{\lceil t/2 \rceil - 1} \binom{t}{i} f_{\mathcal{D}^+}(i, t-i). && \text{(Lemma 3.2 (3))}
\end{aligned}$$

□

Theorem 3.4. *Let $\mathcal{A}$ be a query algorithm that solves the* PromiseMAJORITY *problem on inputs uniformly drawn from $S_n = \{x \in \{0,1\}^n | w_1(x) \in \{\lfloor(1/2 - \epsilon)n\rfloor, \lceil(1/2+\epsilon)n\rceil\}\}$ with failure probability $\delta(n)$ using always at most $t(n)$ queries, with $t(n) \leq \frac{1}{4}\sqrt{n}$. There exists an absolute constant C so that:*

$$t(n) \geq \frac{1}{4\epsilon^2}\log\left(\frac{C}{\delta}\right)$$

for all sufficiently large n.

Proof. First, we can assume that the algorithm always samples exactly $t(n)$ distinct points. Otherwise, we can create an algorithm A' which samples additional points before answerting until exactly $t(n)$ points have been sampled.

For each n, the uniform distribution U_{S_n} on S_n is a symmetric distribution. Therefore, by Lemma 3.3, we have:

$$\sum_{i=0}^{\lceil t/2\rceil-1}\binom{t}{i} f_{U^+_{S_n}}(i, t-i) \leq \delta(n).$$

Let $k = \lceil(1/2+\epsilon)n\rceil$. We can compute $f_{U^+_{S_n}}$:

$$f_{U^+_{S_n}}(w, t-w) = \frac{\binom{n-t}{k-w}}{\binom{n}{k}} = \frac{(n-t)!k!(n-k)!}{n!(k-w)!(((n-k)-(t-w))!} = \frac{(k)_w}{(n)_w}\frac{(n-k)_{t-w}}{(n-w)_{t-w}} \tag{3.1}$$

Where $(n)_h = \frac{n!}{h!}$ is the falling factorial. We use the following inequalities (TODO references) for the falling factorial:

references for factorial inequalities

$$n^h e^{-\frac{h^2}{2(n-h)}} \leq (n)_h \leq n^h e^{-\frac{h(h-1)}{2n}}$$

So we have:

$$\frac{(k)_w}{(n)_w} \geq \frac{k^w e^{-\frac{w^2}{2(n-w)}}}{n^w e^{-\frac{w(w-1)}{2(n-w)}}} = \left(\frac{k}{n}\right)^w e^{-w\left(\frac{w}{2(k-w)}-\frac{w-1}{2n}\right)}$$

Recall that $w \le t \le \frac{1}{4}\sqrt{n}$, and that $k > \frac{1}{2}n$. Using this, we can see that for sufficiently large n, we have that $e^{-w\left(\frac{w}{2(k-w)} - \frac{w-1}{2n}\right)} \ge 1/\sqrt{2}$, so:

$$\frac{(k)_w}{(n)_w} \ge \frac{1}{\sqrt{2}} \left(\frac{k}{n}\right)^w$$

Similarly, for sufficiently large n we have:

$$\frac{(n-k)_{t-w}}{(n-w)_{t-w}} \ge \frac{1}{\sqrt{2}} \left(\frac{n-k}{n-w}\right)^{t-w} > \frac{1}{\sqrt{2}} \left(\frac{n-k}{n}\right)^{t-w}$$

Substituting in (3.1), we have:

$$f_{U^+_{S_n}}(w, t-w) \ge \frac{1}{2} \left(\frac{k}{n}\right)^w \left(\frac{n-k}{n}\right)^{t-w}$$

So that:

$$\begin{aligned}
\delta &\ge \sum_{i=0}^{\lceil t/2 \rceil - 1} \binom{t}{i} f_{U^+_{S_n}}(i, t-i) \\
&\ge \sum_{i=0}^{\lceil t/2 \rceil - 1} \frac{1}{2} \binom{t}{i} \left(\frac{k}{n}\right)^i \left(\frac{n-k}{n}\right)^{t-i} \\
&= \frac{1}{2} \Pr_{X \sim \mathrm{Bin}(t, k/n)} \left[X < \left\lceil \frac{t}{2} \right\rceil\right]
\end{aligned}$$

By standard bounds on the tail of the binomial distribution, (TODO reference and details) we have that there exists a constant C so that $\Pr_{X \sim \mathrm{Bin}(t,k/n)} \left[X < \left\lceil \frac{t}{2} \right\rceil\right] \ge Ce^{-4\epsilon^2 t}$. Rearranging, we get our desired result.

reference for tails of binomial distribution

□

Corollary 3.5. *Let $\epsilon(n)$ be a function satisfying $\epsilon(n) = \omega(n^{-1/4})$ and $\epsilon(n) = o(1)$. Then, all query algorithms $\mathcal{A}$ solving* PromiseMAJORITY *with bounded failure probability for all inputs do at least $\Omega(1/\epsilon(n)^2)$ queries.*

Proof. TODO □

prove corollary, and state it more precisely

3.1.2 Randomized Constructive Separation

Explain how to make a randomized constructive separation from above (amplify error probability).

Theorem 3.6.

3.2 Sufficiently Constructive Separations Imply Lower Bounds

State and prove possible strengthenings of the result of [Che+24].

Theorem 2.3 (Theorem 1.6 from [Che+24])**.** *Let ϵ be a function of n satisfying $\epsilon \leq 1/(\log n)^{\omega(1)}$, and $1/\epsilon$ is a positive integer computable in poly$(1/\epsilon)$ time given n in binary.*

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

3.3 Constructive Separations Against Weaker Query Algorithms

In Theorem 2.3, there are two important quantitative assumptions that must hold in order to obtain a lower bound:

1. We must have a refuter against all t-juntas with $t = O(\log \epsilon^{-1}/\epsilon^{1+\delta})$ for some $\delta > 0$.

2. The function $\epsilon(n)$ must go to zero sufficiently fast: $\epsilon(n) = 1/(\log n)^{\omega(1)}$.

We can wonder whether those conditions are *tight*, in the sense that if we relax them slightly then we can have a constructive separation. We will show that the answer is positive for both of those conditions.

3.3.1 Refuters for $t \leq C/\epsilon$

In Theorem 2.3, we need a refuter against k-juntas for $k = \Omega(1/\epsilon^{1+\delta})$ for any $\delta > 0$ in order to obtain a breakthrough. This condition is tight in the sense that we can easily construct list-refuters for any query algorithms making less than C/ϵ queries (for some C depending on the probability gap of the algorithms).

Theorem 3.7. *Let $\epsilon(n)$ be a function so that $\epsilon(n) = o(1)$, $\epsilon(n) > 1/n$ and the integer $1/\epsilon$ is computable in polynomial time in n. There exists a polynomial-time algorithm R that, on input 1^n, outputs a list of $O(1/\epsilon)$ strings of length n, so that every query algorithm using $\frac{1}{16\epsilon}$ queries fails on one of the strings in the list with probability at least* $1/3$.

Proof. The algorithm R outputs $1 + \left\lfloor \frac{n}{4\lceil \epsilon n \rceil} \right\rfloor$ strings.

The first string consist of $\lfloor (1/2 - \epsilon)n \rfloor$ ones, followed by zeros. The i-th next string ($1 \leq i \leq \left\lfloor \frac{n}{4\lceil \epsilon n \rceil} \right\rfloor$) consists of $\lfloor (1/2 - \epsilon)n \rfloor$ ones followed by zeros in all positions except for the block from position $\lfloor (1/2)n \rfloor + (i-1) \cdot 2\lceil \epsilon n \rceil$ to $\lfloor (1/2)n \rfloor + i \cdot 2\lceil \epsilon n \rceil - 1$. That is, each string has a block of $2\lceil \epsilon n \rceil$ consecutive ones on its right half, disjoint with the blocks of all other strings.

Let $\mathcal{A}$ be any query algorithm as in the statement, and consider its execution on the first string. With any fixed randomness r for which $\mathcal{A}$ answers correctly on that string, it will answer incorrectly on at least half of the remaining $\left\lfloor \frac{n}{4\lceil \epsilon n \rceil} \right\rfloor < \frac{1}{8\epsilon}$ strings, since $\mathcal{A}$ can hit at most $\frac{1}{16\epsilon}$ distinct blocks with its queries. Therefore, if $\mathcal{A}$ answers correctly on the first string with probability at least $2/3$, it will answer incorrectly with probability at least $1/3$ on at least one of the other strings.

□

Note that this is an "explicit obstructions" result: the list refuter does not depend on the algorithm being refuted. Note also that the refuter works for any query algorithm, not just juntas, and also the refuter is quite simple and therefore can be implemented in restricted circuit models if they allow computation of $1/\epsilon$ and simple arithmetic operations with it.

3.3.2 Derandomization of Query Algorithms and Refuters for $\epsilon = 1/\mathbf{polylog}(n)$

One possible approach to obtaining deterministic constructive separations is to try to derandomize the constructive separation of Theorem 3.6. Pseudorandom generators fooling query algorithms are well-studied [HH23], and they will allow us to obtain a quasipolynomial list-refuter when $\epsilon = 1/(\log(n))^{O(1)}$.

Definition 3.8. A set $X_1, \ldots, X_n$ of random variables is k-wise independent if for any $I \subseteq [n]$ with $|I| = k$ and any values $\{x_i\}_{i\in I}$, we have that the events $\{X_i = x_i\}_{i\in I}$ are independent.

We say that a probability distribution $\mathcal{D}$ on $\{0,1\}^n$ is k-wise independent if the n random variables corresponding to each of the bits are k-wise independent.

We say that a probability distribution $\mathcal{D}$ on $\{0,1\}^n$ is k-wise independent probability p Bernoulli if it is k-wise independent and $\Pr[x_i = 1] = p$ for all $i = 1, \ldots, n$.

Think of better/shorter terms to use to refer to k-wise independent prob p Bernoulli.

It is clear that k-juntas can not distinguish between k-wise independent Bernoulli distributions and n independent Bernoulli random variables with the same probability. Interestingly, these distributions also fool any query algorithm using k queries.

Definitions of query algorithms

Theorem 3.9. *Let $\mathcal{A}$ be a query algorithm using at most k queries on $\{0,1\}^n$, let $\mathcal{D}$ be the distribution of n independent Bernoulli variables with probability p on $\{0,1\}^n$, and let $\mathcal{D}'$ be a k-wise independent probability p Bernoulli distribution on $\{0,1\}^n$. Then:*

$$Pr_{x\leftarrow\mathcal{D}}[\mathcal{A}(x) = 1] = Pr_{x\leftarrow\mathcal{D}'}[\mathcal{A}(x) = 1].$$

Proof. Note that it is sufficient to prove the result assuming $\mathcal{A}$ is a (deterministic) depth-k decision tree, since if $\mathcal{A}$ is probabilistic, the result for each of the decision trees resulting from each fixed randomness implies the desired result for the probabilistic algorithm.

Let $\mathcal{L}$ be the set of leaves of the decision tree, and for each $u \in \mathcal{L}$, let $\mathcal{A}_u(x)$ be equal to 1 if $\mathcal{A}(x) = 1$ and $\mathcal{A}$ ends at leaf u of the tree on input x, and 0 otherwise. Note that $\mathcal{A}_u(x)$ is a k-junta and that $\mathcal{A}(x) = \sum_{u \in \mathcal{L}} \mathcal{A}_u(x)$. Therefore:

$$\mathbb{E}_{x \leftarrow \mathcal{D}}\left[\mathcal{A}(x)\right] = \sum_{u \in \mathcal{L}} \mathbb{E}_{x \leftarrow \mathcal{D}}\left[\mathcal{A}_u(x)\right] = \sum_{u \in \mathcal{L}} \mathbb{E}_{x \leftarrow \mathcal{D}'}\left[\mathcal{A}_u(x)\right] = \mathbb{E}_{x \leftarrow \mathcal{D}'}\left[\mathcal{A}(x)\right],$$

and we have the desired result since $\Pr\left[\mathcal{A}(x) = 1\right] = \mathbb{E}\left[\mathcal{A}(x)\right]$, because $\mathcal{A}(x)$ takes values 0 and 1. □

Now, we show that there is an efficient pseudorandom generator which samples from a k-wise independent distribution with seed length $O(k \log n)$.

Theorem 3.10. *There is a polynomial-time algorithm $\mathcal{A}$ which, given as input integers n, k with $n \geq k \geq 1$, an integer p with $0 \leq p \leq 2^{\lceil \log_2 n \rceil}$, and $s = k\lceil \log_2 n \rceil$ random bits, outputs n bits with a k-wise independent probability $p/2^{\lceil \log_2 n \rceil}$ Bernoulli distribution.*

Proof. Let $q = 2^{\lceil \log_2 n \rceil}$, and let $\mathbb{F}_q$ be the finite field with q elements.

Let $\mathcal{P}_{k-1}$ be the set of univariate polynomials of degree at most $k-1$ in $\mathbb{F}_q$, and let $z_1, \ldots, z_n \in \mathbb{F}_q$ be distinct elements of the field. Define the function $G \colon \mathcal{P}_{k-1} \to \mathbb{F}_q^n$ by

$$G(p) = (p(z_1), \ldots, p(z_n)).$$

We claim that, when p is sampled uniformly at random from $\mathcal{P}_{k-1}$, the distribution of $G(p)$ is k-wise independent. This is because, for all $I \subset [n]$ with $|I| \leq k$ and all sequences of values $\{x_i\}_{i \in I}$ in $\mathbb{F}_q$, the number of $p \in \mathcal{P}_{k-1}$ so that $p(z_i) = x_i$ for all $i \in I$ is exactly $q^{k-|I|}$, since each polynomial $p \in \mathcal{P}_{k-1}$ is uniquely determined by its evaluation at k distinct points and conversely each possible sequence of values at k distinct points yields a unique polynomial. Thus:

$$\Pr\left[\bigwedge_{i \in I} p(z_i) = x_i\right] = \frac{1}{q^{|I|}} = \prod_{i \in I} \Pr\left[p(z_i) = x_i\right],$$

and therefore the $|I|$ events $p(z_i) = x_i$ are independent, as desired.

Now we describe the generator $\mathcal{A}$. We interpret the seed as the encoding of k elements of $\mathbb{F}_q$, determining a polynomial $p \in \mathcal{P}_{k-1}$, and we set the output bit to be 1 if $f(p(z_i)) \leq p$ and to be 0 otherwise, where f is some bijection from $\mathbb{F}_q$ to $\{1, \ldots, q\}$. Therefore, each bit has probability p/q of being equal to 1, and the k-wise independence of the distribution follows from the k-wise independence of $G(p)$. □

And finally, we can use this pseudorandom generator to derandomize Theorem 3.6. We do not prove all of the details, since the argument is similar to that of Theorem 3.4 but with a different probability distribution and with some additional complications.

Theorem 3.11. *Let $\epsilon = \epsilon(n) = 1/(\log n)^{O(1)}$ and let $\delta > 0$ be a sufficiently small constant. There exists a quasipolynomial-time quasipolynomial-list-refuter R for* PromiseMA[illegible]Y$_{n,\epsilon}$ *against all query algorithms using $t(n) = o(1/\epsilon^2)$ queries with probability gap bounded by $1 - 2\delta$.*

Clarify value of delta like in previous results

Proof sketch. The refuter is as follows: On input 1^n, set $k = 1/\epsilon^2$, $q = 2^{\lceil \log_2 n \rceil}$ and run the k-wise independent generator from Theorem 3.10 for all of the $2^{O(k \log n)} = 2^{\mathrm{polylog}(n)}$ seeds for each of $p = p_+ = \lceil q \cdot (1/2 + 2\epsilon) \rceil$ and $p = p_- = \lfloor q \cdot (1/2 - 2\epsilon) \rfloor$. Output all the results which satisfy the at least $(1/2 + \epsilon)n$ ones or at most $(1/2 - \epsilon)n$ ones promise.

Consider the uniform probability distribution on the outputs of the refuter $\mathcal{D}_1$. This distribution is statistically close to the uniform distribution $\mathcal{D}_2$ on all the outputs of the generator, without filtering those that do not satisfy the promise, since there are very few that do not satisfy the promise (a fraction of around $\Pr_{X \sim \mathrm{Bin}(n, p_+/q)}[X < (1/2 + \epsilon)n]$). In turn, distribution $\mathcal{D}_2$ is indistinguishable by algorithms making $t(n)$ queries from the corresponding distribution $\mathcal{D}_3$ in which either all n bits are sampled from independent Bernoullis with probability p_+/q or all n bits are sampled from Bernoullis with probability p_-/q, each of the two options with probability 1/2. And finally, distribution $\mathcal{D}_3$ is statistically close to distribution $\mathcal{D}_4$, which is distribution $\mathcal{D}_3$ conditioned on the output satisfying the promise. We conclude that query algorithms using $t(n)$ queries can not distinguish between distributions $\mathcal{D}_1$ and $\mathcal{D}_4$ except with some small probability (exponentially small in n, in fact).

Verify this claim.

Distribution $\mathcal{D}_4$ is a symmetric distribution on $X_\epsilon^+ \cup X_\epsilon^-$, and with similar arguments as in Theorem 3.4 we can prove that query algorithms with error probability 2δ with inputs over distribution $\mathcal{D}_4$ require $\Omega(1/\epsilon^2)$ queries. Therefore, algorithms with probability gap bounded by $1-2\delta$ must fail at least on a fraction δ of inputs from $\mathcal{D}_4$. Therefore, they must also fail at a constant fraction of inputs from $\mathcal{D}_1$ and in particular they will fail for at least one of the elements outputted by $R(1^n)$, for n big enough.

□

Write conclusion relating it to observations of the previous section.

3.3.3 Limits of Explicit Obstructions

One salient aspect of the list-refuters described in the two previous subsections is that they are explicit obstructions: they do not depend on the algorithm being refuted. This is much more restrictive than the requirement to obtain a breakthrough in Theorem 2.3, in which the refuters are allowed to depend on the algorithm being refuted. The following results displays the limits of explicit obstructions for this problem:

Theorem 3.12. *Let R be an algorithm that, on input 1^n, outputs a list $L = L_n$ of strings of size n in time poly$(|L|)$. There is a probabilistic query algorithm $\mathcal{A}$ running in time poly$(n, |L|, 1/\epsilon)$ and making $O\left(\frac{\log(|L|)\cdot\log(\log(|L|))}{\epsilon}\right)$ queries which correctly solves* $\text{PromiseMAJORITY}_{n,\epsilon}$ *with probability at least $2/3$ on each of the outputs produced by R.*

Proof. The algorithm $\mathcal{A}$ is described in Algorithm 1. First, $\mathcal{A}$ computes L and divides it into L^+ and L^-, depending on whether the majority is 1 or 0. L^+ and L^- will mantain at all times the elements of the list that are consistent with the information obtained by the queries. If at any point L^+ or L^- becomes empty, the answer is determined and the algorithm can output it and stop.

$\mathcal{A}$ computes m^+ and m^-, the bitwise majority of all the elements of L^+ and L^-, respectively. That is, for each $i = 1, \ldots, n$, m_i^+ is equal to 1 if there are more elements of L^+ with i-th bit equal to 1 than elements with i-th bit equal to 0, and is equal to 0

Algorithm 1 Algorithm for Theorem 3.12

```
Input: $n$
Program:
    $L \leftarrow R(1^n)$
    $(L^+, L^-) \leftarrow$ FILTER($L$)    ▷ $L^+$ are the elements of $L$ with $\geq (1/2 + \epsilon)$ ones, $L^-$ the ones with $\leq (1/2 - \epsilon)n$ ones
    while $L^+ \neq \emptyset$ and $L^- \neq \emptyset$ do
        $m^+ \leftarrow$ MAJORITY($L^+$)
        $m^- \leftarrow$ MAJORITY($L^-$)
        if $m^+ \neq m^-$ then
            $i \leftarrow$ index such that $m_i^+ \neq m_i^-$.
            $v \leftarrow$ QUERY($i$)
            $(L^+, L^-) \leftarrow$ UPDATE($i, v, (L^+, L^-)$)    ▷ Returns elements from the lists consistent with $x[i] = v$
        else
            $S_0 \leftarrow \{i : m_i^+ = 0\}$
            $S_1 \leftarrow \{i : m_i^+ = 1\}$
            $c \leftarrow \text{argmax}_{i \in \{0,1\}} |S_i|$
            found $\leftarrow$ false
            repeat $\lceil (\log(6 \log |L|)) \cdot 1/\epsilon \rceil$ times:
                $i \leftarrow$ RANDOMELEMENT($S_c$)
                $v \leftarrow$ QUERY($i$)
                $(L^+, L^-) \leftarrow$ UPDATE($i, v, (L^+, L^-)$)
                if $v \neq c$ then
                    found $\leftarrow$ true
                end if
            end
            if not found then
                output $c$
                HALT
            end if
        end if
    end while
    output 0 if $L^+ = \emptyset$, 1 if $L^- = \emptyset$
    HALT
```

otherwise. If $m^+ \neq m^-$, $\mathcal{A}$ queries one of the bits where m^+ and m^- differ, discards the elements of L^+ and L^- inconsistent with this information and starts over again. Note that this will result in halving the size of L^+ or L^-.

If $m^+ = m^-$, then $\mathcal{A}$ determines the sets of indices where both L^+ and L^- have majority 0 and where they both have majority 1 and samples $\lceil (\log(6\log|L|)) \cdot 1/\epsilon \rceil$ random points from the biggest set of those two. If all the sampled points coincide with the majority value on that set, $\mathcal{A}$ outputs the answer corresponding to that value. Otherwise, $\mathcal{A}$ starts over again; note that in this case, the sizes of both L^+ and L^- are halved.

We proceed to the analysis of the algorithm. We have to argue that it performs $O(\log|L|\log\log|L|/\epsilon)$ queries, and that it answers correctly with probability at least 2/3. The bound on the number of queries is clear: the algorithm performs at most $O(\log\log|L|/\epsilon)$ queries in each iteration of the main loop and it does at most $2\cdot\log|L|$ iterations, since in each iteration the size of L^+ or L^- is halved and it exits the loop when one of them becomes empty.

Suppose that we have an x with $\geq (1/2+\epsilon)$ ones, and we will bound the probability that $\mathcal{A}$ outputs 0 (the case where x has majority 0 is symmetric). It is clear that if $\mathcal{A}$ exits the loop because L^- has become empty, then it always gives the correct answer. Then, $\mathcal{A}$ can only give a wrong answer when it outputs 0 after sampling $\log(6\log|L|)/\epsilon$ points in S_0 and finding that they are all equal to 0. Note that, if $|S_0| \geq 1/2$, then $x[S_0]$ must have at least ϵn bits equal to 1, and therefore the probability of sampling a bit equal to 1 is at least ϵ. Therefore, the probability that all sampled bits are equal to 0 is at most $(1-\epsilon)^{\log(6\log|L|)/\epsilon} \leq \frac{1}{6\log|L|}$. Since the algorithm performs at most $2\log|L|$ iterations, by the union bound the probability that $\mathcal{A}$ outputs 0 is at most $2\log|L| \cdot \frac{1}{6\log|L|} = \frac{1}{3}$, as desired.

□

Theorem 3.12 shows that any constructive separation for a class of query algorithms including the algorithm described in the statement must necessarily have refuters that depend on the algorithm being refuted. Note that if $\epsilon = 1/(\log n)^{\omega(1)}$ and $|L|$ is quasipolynomial, then $O((\log|L|\log\log|L|)/\epsilon) = o(1/\epsilon^{1+\delta})$ for all $\delta > 0$. This suggests that that refuters that would achieve the breakthroughs of Theorem 2.3, if they exist, should depend on the algorithm being refuted. However, there are some details that could potentially make this not be the case: recall that in Theorem 2.3 we only require refuters against uniform distributions of k-juntas running in time $\mathrm{poly}(1/\epsilon)$, while in Theorem 3.12 the algorithm is an adaptive query algorithm running in time $\mathrm{poly}(n, |L|, 1/\epsilon)$.

There seems to be some room for improvement in Theorem 3.12. There is a $\log(1/\epsilon)\log\log(1/\epsilon)$ gap between the number of queries in the algorithms refuted by the explicit obstructions of Theorem 3.7 and the number of queries in the algorithm $\mathcal{A}$ of Theorem 3.12. The repeated sampling of $O(\log\log|L|/\epsilon)$ points seems to be somewhat wasteful, and perhaps with a different approach we could be able to get an $\mathcal{A}$ which makes e.g. $O(1/\epsilon + \log|L|)$ queries, which would close this gap. It also seems feasible to eliminate the adaptivity of the queries by allowing m^+ and m^- to differ in a small number of bits.

Open Question 1. What is the precise frontier of explicit obstructions for $\mathrm{PromiseMAJORITY}_{n,\epsilon}$? Can we improve Theorem 3.12 or Theorem 3.7?

4
Refuters for One-Tape Turing Machines

In this chapter, we will study constructive separations of the language of palindromes

$$\mathrm{PAL} = \{ww^R : w \in \{0,1\}^*\} \cup \{wbw^R : w \in \{0,1\}^*, b \in \{0,1\}\}$$

from one-tape Turing machines running in time $o(n^2)$. In [Che+24] it was proved that constructive separations against nondeterministic machines yielded breakthrough results in circuit lower bounds (recall Theorem 2.4). In this chapter we will explore this setting more deeply, and explain when we can get circuit lower bounds, when we can actually get constructive separations, and even when we can prove that no constructive separations can exist.

For simplicity, through the chapter we will often only consider the language of *even* palindromes $\{ww^R : w \in \{0,1\}^*\}$ in the proofs. Recall that in Definition 2.1 we only require that the refuter works for infinitely many values of n, so in particular a refuter that only works for even palindromes and ignores the odd case is valid. However, the arguments that we make for the even case in this chapter can always be modified slightly to work for the odd case too. We omit the explanations of which particular changes would be necessary for each of our results for brevity, since they are rarely illuminating.

4.1 Lower Bounds Against 1-TMs for PAL

In this section, we prove the $\Omega(n^2)$ lower bound for 1-TMs computing PAL, due to Hennie [Hen65]. The proof of this fact lends itself to a simple *probabilistic* constructive separation that we also discuss.

4.1.1 Crossing Sequences and Lower Bounds

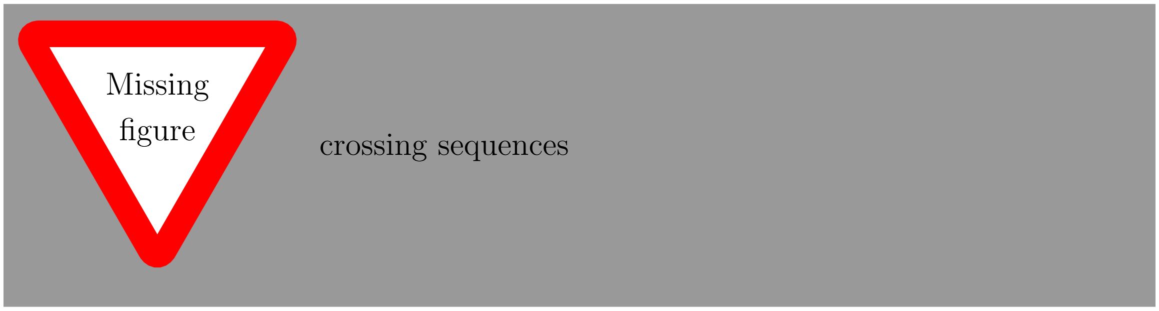
Definition 4.1. Let M be a 1-TM, and let $E = ((q_1, p_1), (q_2, p_2), \dots, (q_t, p_t))$, with $(q_i, p_i) \in Q \times \mathbb{Z}$, be a sequence describing an execution of M, that is, E is the sequence of states and positions of the tape head of M when it is executed on some input and (possibly) with some given non-deterministic choices. The *crossing sequence at position* i, denoted by $C_i(E)$, is the sequence of states for which the tape head crosses from position i to position $i+1$ or vice versa; that is, the sequence of states q_j such that $p_{j-1} = i$ and $p_j = i+1$ or $p_{j-1} = i+1$ and $p_j = i$.

explain indexing wrt to how input is written in the tape in the introduction

If M is a deterministic 1-TM and $x \in \{0,1\}^*$, we denote by $C_{M,i}(x)$ the crossing sequence at position i of the execution of M on input x.

If M is a non-deterministic 1-TM, $x \in \{0,1\}^*$ and $b \in \{0,1\}^*$ is a sequence of non-deterministic choices for the execution of M with input x, we denote by $C_{M,i}(x, b)$ the crossing sequence at position i of the execution of M on input x with non-deterministic choices b.

When the 1-TM M is clear from context, we will usually write $C_i(x)$ instead of $C_{M,i}(x)$. Also, for a non-deterministic 1-TM M and an accepted word x, we will usually write just $C_i(x)$, omitting the description of the non-deterministic choices, to



refer to the crossing sequence at position i on input x for *some* accepting execution of x. That is, for each accepted word x, we arbitrarily choose some accepting execution (e.g. the lexicographically smallest one), and use $C_i(x)$ to refer to the crossing sequences of that execution. This is because, for the purposes of proving lower bounds for PAL, the fact that there might be multiple executions (and therefore multiple crossing sequences) for each accepted word does not hurt, and we can prove the lower bounds only assuming that each accepted word has *at least* one accepting execution.

Intuitively, crossing sequences are a way to quantify the way a machine "carries information" from one side of the boundary to the other one via its internal states. Note that, in a crossing sequence at position $i \geq 1$, each odd crossing is from left to right and each even crossing is from right to left. See Section 4.1.1 for a pictorial depiction of crossing sequences.

Lemma 4.2. *Let $a = a_1 \dots a_i a_{i+1} \dots a_n$ and $b = b_1 \dots b_i b_{i+1} \dots b_m$ be two words accepted by a 1-TM M so that $C_i(a) = C_i(b)$. Then the word $c = a_1 \dots a_i b_{i+1} \dots b_m$ is also accepted by M.*

Proof. Let $E^a = ((q^a_1, p^a_1), \dots (q^a_{t^a}, p^a_{t^a}))$ be the accepting execution for a in M and let $E^b = ((q^b_1, p^b_1), \dots (q^a_{t^b}, p'_{t^b}))$ be the accepting execution for b. Let $j^a_1, j^a_2, \dots, j^a_{|C_i(a)|}$ and $j^b_1, j^b_2, \dots, j^b_{|C_i(a)|}$ be the indices corresponding to the states in $C_i(a)$ in E^a and E^b, respectively.

We can define a new execution E^c by reproducing the execution E^a up to index

Missing figure

pasting together different executions

Figure 4.1: Illustration of Lemma 4.2.

j_1^a, then the execution E^b between indices j_1^b and j_2^b, and so on:

$$E^c = ((q_1^a, p_1^a), \ldots, (q^a_{j_1^a-1}, i), (q^b_{j_1^b}, i+1), (q^b_{j_1^b+1}, p^b_{j_1^b+1}), \ldots,$$
$$(q^b_{j_2^b-1}, i+1), (q^a_{j_2^a}, i), (q^a_{j_2^a+1}, p^a_{j_2^a+1}), \ldots).$$

See Section 4.1.1 for an illustration of execution E^c. It is not difficult to see that E^c is a valid accepting execution for M on input c, since the machine behaves as if it were reading input a to the left of index i and as if it were reading input b to the right of index i.

□

Corollary 4.3. *Let M be a 1-TM that accepts two palindromes $w_1w_2w_2^Rw_1^R$ and $w_3w_4w_4^Rw_3^R$ with $|w_1| = |w_3| > 0$, $|w_2| = |w_4|$, and $w_1 \neq w_3$. If the $C_{|w_1|}(w_1w_2w_2^Rw_1^R) = C_{|w_1|}(w_3w_4w_4^Rw_3^R)$, then M also accepts a non-palidrome of length $2|w_1| + 2|w_2|$.*

Proof. By Lemma 4.2, the non-palindrome $w_1w_4w_4^Rw_3^R$ is accepted. □

Now we are ready to prove the $\Omega(n^2)$ lower bound for PAL. We will prove a quantitative bound, which will be useful for our randomized constructive separation:

check that formulas are correctly updated everywhere

Theorem 4.4. *Let M be a non-deterministic 1-TM running in time $T(n)$ with s states that rejects every non-palindrome. Then, for every even n, the number of palindromes of length n accepted by M is bounded by $2^{\frac{(4\log s)T(n)}{n}+\frac{n+6}{4}}$.*

Proof. Let P_n be the set of palindromes of length n accepted by M.

Let

$$\ell_i = \frac{1}{|P_n|} \sum_{p \in P_n} |C_i(p)|$$

be the average length of the crossing sequence at position i on the tape for all palindromes in P_n. By Markov's inequality, there are at least $\frac{1}{2}|P_n|$ palindromes p from P_n for which $|C_i(p) \leq 2\ell_i$. Now, fix $i \in [1, n/2]$ and note that:

- There are $1 + s + \cdots + s^{2\ell_i} < 2s^{2\ell_i}$ crossing sequences at position i of length at most $2\ell_i$, where s is the number of states in M.
- There can be at most $2^{n/2-i}$ palindromes in P_n with the same crossing sequence at position i, since in the decomposition $p = w_1 w_2 w_2^R w_1^R$ with $|w_1| = i$, the prefix w_1 must be the same for all such palindromes because of Corollary 4.3 and the assumption that M rejects every non-palindrome.

Therefore:

$$\frac{1}{2}|P_n| < 2s^{2\ell_i} \cdot 2^{n/2-i}.$$

Rearranging:

$$\ell_i > \frac{\log |P_n| + i - n/2 - 2}{2 \log s}. \tag{4.1}$$

For each palindrome $p \in P_n$, we have that $\sum_{i=1}^{n/2} C_i(p) \leq T(n)$, so

$$\sum_{i=1}^{n/2} \ell_i = \frac{1}{|P_n|} \sum_{i=1}^{n/2} \sum_{p \in P_n} |C_i(p)| = \frac{1}{|P_n|} \sum_{p \in P_n} \sum_{i=1}^{n/2} |C_i(p)| \leq T(n)$$

Hence, summing over all i in (4.1):

$$T(n) > \sum_{i=1/}^{n/2} \frac{\log |P_n| + i - n/2 - 2}{2 \log s} = \frac{1}{2 \log s} \left(\frac{n}{2} \log |P_n| - \frac{n^2}{8} - \frac{3n}{4} \right)$$

And therefore:

$$2^{\frac{(4\log s)T(n)}{n}+\frac{n+6}{4}} > |P_n|$$

As desired.

□

Corollary 4.5. *Every non-deterministic 1-TM recognizing* PAL *must run in* $\Omega(n^2)$ *time.*

Proof. Such a machine M rejects every non-palindrome, so we can apply Theorem 4.4 to get that $2^{\frac{(4\log s)T(n)}{n}+\frac{n+6}{4}} \geq 2^{n/2}$, so $T(n) \geq \frac{n(n-6)}{16\log s} = \Omega(n^2)$. □

4.1.2 Randomized Constructive Separation

We prove the following:

Theorem 4.6.

- *There is a* $\mathsf{ZPP}^{\mathsf{NP}}$*-constructive separation of* PAL *from nondeterministic 1-TMs running in time* $o(n^2)$*.*
- *There is a* BPP*-constructive separation of* PAL *from nondeterministic 1-TMs running in time* $o(n^2)$ *which are promised to accept only palindromes.*

Let M be the non-deterministic Turing machine running in time $o(n^2)$ we are trying to refute, and let $M(x, b)$ be the result of the execution of M on input x and non-deterministic choices b. Our refuter R does the following on input 1^n:

- Uses the NP oracle to determine the truth of the statement

$$\exists x \in \{0,1\}^n \exists b \in \{0,1\}^{\leq n^2} : x \notin \text{PAL} \wedge M(x, b) \text{ accepts.}$$

  - If it is true, it performs a standard search-to-decision reduction with the NP oracle to determine such an x, and outputs x.
  - Otherwise, it chooses a uniformly random palindrome of length n and outputs it.

In the first case, it is clear that R outputs a counterexample for M. In the second case, we have that M does not accept any non-palindrome of length n, so by Theorem 4.4 it also rejects a fraction which goes to 1 when $n \to \infty$ of palindromes of length n, so the probability of outputting a counterexample will be greater than $2/3$ for sufficiently large n.

Thus, we have a $\mathsf{BPP}^{\mathsf{NP}}$-constructive separation. In fact, it can be made $\mathsf{ZPP}^{\mathsf{NP}}$-constructive, since we can verify the output to be a counterexample. Also note that the refuter as stated (without performing verification of the output) uses randomness and the NP oracle in a completely "disjoint" way: in the case where the machine M accepts a non-palindrome, it proceeds in a deterministic, "P^{NP}" way, while in the case where the machine M only accepts palindromes it just outputs a random palindrome without using the NP oracle. This differentiated behavior is of interest because, as we will see in the following sections, there are different consequences for constructive separations from 1-TMs which are promised to accept only palindromes and for constructive separations from 1-TMs which are promised to accept all palindromes.

4.2 Constructive Separations Imply Circuit Lower Bounds

Recall that in [Che+24], the following is proved:

Theorem 2.4 (Theorem 3.4 from [Che+24])**.** *The following hold:*

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

We will reproduce the proof argument here, but with some modifications to suit our purposes better. First, we prove the result with more generality, with the circuit lower bound we obtain being parametrized by the time complexity of the refuter, and not just stated in two discrete cases for P and P^{NP} (we will not discuss results about space complexity). This more granular statement will be useful to compare the result with the refuters we obtain in Section 4.4.

Second, we emphasize in the statement that it is enough to have refuters against 1-TMs which are promised to accept only palindromes, that is, that reject for every non-palindrome. This turns out to be an important aspect, because as we will see in Section 4.3, no P-constructive separation is possible even against deterministic 1-TMs running in time $O(n^{1+\epsilon})$ assuming the existence of a cryptographic primitive. Thus, the second and third bullet points of Theorem 2.4, as stated in [Che+24], are vacuous under this cryptographic assumption. By introducing this strengthening, we can still hope to find such constructive separations even if we believe that the cryptographic primitive exists.

Theorem 4.7. *Let $\{T_i(n)\}_{i\in\mathcal{I}}$ be a family of functions, let $\mathcal{O}$ be any language and let $\mathcal{C}^{\mathcal{O}}$ denote the class of algorithms running in time $T_i(n)$ for some $i \in \mathcal{I}$ with oracle access to $\mathcal{O}$. Let $f(n)$ be a function with $f(n) = \Omega(\log n)$ and $f(n+1) = O(f(n))$. If there is a $\mathcal{C}^{\mathcal{O}}$-constructive separation of* PAL *from 1-NTMs promised to accept only palindromes running in time $O(n \cdot f(n))$, then*

$$\bigcup_{i\in\mathcal{I}} \mathsf{DTIME}[T_i(2^n)]^{\mathcal{O}} \not\subset \mathsf{SIZE}[f(2^{n/2})^{\delta}]$$

for some universal constant $\delta > 0$.

Proof. Assume that $\bigcup_{i\in\mathcal{I}} \mathsf{DTIME}[T_i(2^n)]^{\mathcal{O}} \subseteq \mathsf{SIZE}[f(2^{n/2})^{\delta}]$. By Lemma 2.5, the output of any refuter $R \in \mathcal{C}^{\mathcal{O}}$ has circuit complexity $O(f(n+1)^{\delta}) = O(f(n)^{\delta})$. We will construct a 1-NTM M running in time $O(n \cdot f(n))$ that works correctly on the output of any such refuter, proving the contrapositive of the desired statement. First, M guesses a circuit C of size $s = O(f(n)^{\delta})$ and writes its description around the beginning of the tape. Then, M verifies that the circuit generates the input x by moving the circuit and a counter through the tape and checking that $C(i) = x_i$. Finally, M checks that $C(i) = C(n-i+1)$ for all $1 \leq i \leq n/2$, verifying that the input is indeed a palindrome. Moving the circuit through the input can be done in time $O(n \cdot (\log n + s))$ and each of the $O(n)$ evaluations of the circuit can be done in time $O(s^{O(1)})$, so if δ is small enough the total time complexity is $O(n \cdot f(n))$. □

Corollary 4.8. *The following hold:*

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

Proof. Take $\mathcal{I} = \mathbb{N}$, $T_i(n) = n^i$, $f(n) = n^{0.1}$ and $\mathcal{O} = \text{SAT}$ or $\mathcal{O} = \emptyset$ (respectively) in Theorem 4.7. □

Interestingly, those lower bounds for E^{NP} and E are precisely the ones that are needed for derandomization of algorithms:

Theorem 4.9 ([NW94; IW97])**.**

- *If* $\mathsf{E} \not\subset \mathsf{SIZE}[2^{\delta n}]$*, then there exists a pseudorandom generator* $G : \{0,1\}^s \to \{0,1\}^n$*, with* $n = 2^{\Omega(s)}$ *running in time* $2^{O(s)}$ *which fools circuits of size* n^3*. In particular, this implies* $\mathsf{BPP} = \mathsf{P}$*.*

- *If* $\mathsf{E}^{\mathsf{NP}} \not\subset \mathsf{SIZE}[2^{\delta n}]$*, then there exists a pseudorandom generator* $G : \{0,1\}^s \to \{0,1\}^n$*, with* $n = 2^{\Omega(s)}$ *running in time* $2^{O(s)}$ *with an* NP *oracle which fools circuits of size* n^3*. In particular, this implies* $\mathsf{BPP} \subseteq \mathsf{P}^{\mathsf{NP}}$*.*

Therefore, deterministic constructive separations of PAL from $O(n^{1+\epsilon})$ time 1-NTMs imply breakthrough results in derandomization. At the same time, such deterministic constructive separations could be obtained by derandomizing the probabilistic constructive separation of the previous section. The difference here between the two kinds of derandomization is that in the second case we would need to have pseudorandom generators against *nondeterministic* machines, which is a stronger requirement than the generators against deterministic circuits provided by Theorem 4.9.

On the other hand, our generator only needs to fool nondeterministic one-tape machines running in $O(n^{1+\epsilon})$ time, which is a much weaker model than general nondeterministic polynomial-size circuits. In this setting, a pseudorandom generator with seed length $\tilde{O}(\sqrt{T})$ against *deterministic* 1-TMs running in time $O(T)$ is known [INW94]. Also, while the generators of Theorem 4.9 and [INW94] fool all n^3-size circuits and $O(T)$-time 1-DTMs respectively, our generator can be *targeted*, that is, it can depend on which TM M we are trying to fool. And it does not need to be a full pseudorandom generator in the sense of indistinguishability from randomness; it can be a "hitting set generator" [ACR98] which generates at least one element outside the square-root-sized set defined by Theorem 4.4. All in all, it is not clear that the requierements to derandomize our constructive separation are much stronger than the results of Theorem 4.9 by achieving it.

Open Question 2. What are the relationships between constructive separations of PAL from 1-NTMs, (targeted) derandomization/hitting sets of $O(n^{1+\epsilon})$ 1-NTMs, and derandomization of deterministic machines/circuits? Can we find some sort of equivalence between them, or evidence than one of them is harder than the others?

4.3 Strongly Constructive Separations Do Not Exist Under Cryptographic Assumptions

In this section, we will see that BPP-constructive separations of PAL against $O(n^{1+\epsilon})$ time 1-TMs do not exist if we assume the existence of a certain cryptographic primitive. To show this, we will use an idea from [CLO24], where it is used to show that lower bounds for 1-TMs recognizing palindromes can not be proved in certain weak theories of arithmetic. Here we will reproduce their argument, adapting it to our setting.

The cryptographic primitives we will be using are *keyless collision-resistant hash functions.*

Definition 4.10 (Keyless Collision-Resistant Hash Function)**.** A keyless collision-resistant hash function with hash value length $m = m(n) < n$ is a polynomial-time computable function $h\colon \{0,1\}^* \to \{0,1\}^*$ such that for every x with $|x| = n$ we have $|h(x)| = m(n)$ and for every uniform probabilistic polynomial-time adversary $\mathcal{A}$ and every n, we have

$$\Pr\left[\mathcal{A}(1^n) \text{ outputs } \langle x_1, x_2\rangle \text{ such that } x_1 \neq x_2, |x_1| = |x_2| = n \text{ and } h(x_1) = h(x_2)\right] \leq \epsilon(n)$$

where $\epsilon(n)$ is a *negligible* function (that is, $\epsilon(n) = 1/n^{\omega(1)}$).

The idea is that, if we have a TM that hashes the left half and the reverse of the right half of a palindrome and accepts if they are equal, a refuter against this TM needs to find collisions in the hash function, which is supposed to be difficult. The main question here is whether we can implement this in a one-tape TM running in time $O(n^{1+\epsilon})$, only assuming the existence of polynomial-time hash functions h.

The answer is yes: we can do it using a technique from cryptography known as the *Merkle-Damgård construction* [Mer90; Dam90].

Theorem 4.11. *Suppose there exist keyless collision-resistant hash functions. Then, for all $0 < \delta, \epsilon < 1$, there exists a keyless collision resistant hash function $H = H_{\delta,\epsilon}$ with hash value length $m(n) < n^{\delta}$ which can be computed in time $O(n^{1+\epsilon})$ in a one-tape TM.*

Proof. Let h be a keyless collision-resistant hash function, computable in time $O(n^k)$ in a 1-TM for some k. Without loss of generality, we can assume that h has hash value length $n-1$ (we can always arbitrarily pad the output values mantaining collision resistance). Given an input size n, let $m = m(n) = \min\{\lfloor n^{\delta}\rfloor, \lfloor n^{\epsilon/k}\rfloor\}$ and define the following sequence of functions:

consistent notation for concatenation, also explain in introduc-

$$H_{-1}\colon \{0,1\}^{m-1} \to \{0,1\}^{m-1}\colon\ H_{-1}(x) = x$$

$$H_0\colon \{0,1\}^{m} \to \{0,1\}^{m-1}\colon\ H_0(x) = h(x).$$

$$H_1\colon \{0,1\}^{m+1} \to \{0,1\}^{m-1}\colon\ H_1(x||b) = h(H_0(x)||b).$$

$\ldots$

$H_i \colon \{0,1\}^{m+i} \to \{0,1\}^{m-1}\colon\ H_i(x||b) = h(H_{i-1}(x)||b)$.

$\ldots$

$H_{n-m} \colon \{0,1\}^{n} \to \{0,1\}^{m-1}\colon\ H_{n-m}(x||b) = h(H_{n-m-1}(x)||b)$.

figure? (maybe)

Take $H = H_{n-m}$. We have to prove that H is a collision-resistant hash function, and that it can be computed in time $O(n^{1+\epsilon})$ in a 1-TM.

We first prove that H is a collision-resistant hash function. Let $\mathcal{A}_H$ be a polynomial-time adversary outputting collisions for H. Consider the following algorithm $\mathcal{A}_h$: on input 1^m, run $\mathcal{A}(1^n)$ for all n so that $m(n) = m$ (there are polynomially many such values of n). If $\mathcal{A}_H$ outputs a valid collision $\langle x_1, x_2\rangle$ for H for some such value of n, do the following: compute the smallest value $i \in [0, n-m]$ such that $x_1[1\ldots m+i] \neq x_2[1\ldots m+i]$ but $H_i(x_1[1\ldots m+i]) = H_i(x_2[1\ldots m+i])$ and output $\langle H_{i-1}(x_1[1\ldots m+i-1])||x_1[m+i], H_{i-1}(x_2[1\ldots m+i-1])||x_2[m+i]\rangle$. We have that $h(H_{i-1}(x_1[1\ldots m+i-1])||x_1[m+i]) = h(H_{i-1}(x_2[1\ldots m+i-1])||x_2[m+i])$, and by the assumption that i was the smallest value with the specified property, at least one of $H_{i-1}(x_1[1\ldots m+i-1]) \neq H_{i-1}(x_2[1\ldots m+i-1])$ or $x_1[m+i] \neq x_2[m+i]$ hold. In any case, $\mathcal{A}_h(1^m)$ outputs a valid collision of length m for h. Hence:

$$\Pr\left[\mathcal{A}_H(1^n) \text{ outputs a collision for } H\right] \leq \Pr\left[\mathcal{A}_h(1^{m(n)}) \text{ outputs a collision for } h\right] \leq \epsilon_{\mathcal{A}_h}(m(n)),$$

where $\epsilon_{\mathcal{A}_h}(n)$ is a negligible function since h is collision-resistant, and therefore $\epsilon_{\mathcal{A}_h}(m(n))$ is also negligible and we get that the probability of success of any adversary is bounded by a negligible function, as desired.

Now we describe how to compute the function in a one-tape TM in time $O(n^{1+\epsilon})$:

First, we compute $n = |x|$, the length of the input. We iterate through the input from left to right, keeping a counter near the head in a "separate track" from the input (using a larger tape alphabet). The counter has at most $\log n$ bits and so updating it and moving it to the right takes $O(\log n)$ time, for a total complexity of $O(n \log n)$.

Second, we compute m and we put a mark on the m-th character of x on the tape. Computing m given n in binary can be done in polylog(n) time, since ϵ and δ are fixed rational numbers, and mantaining a counter until the m-th character can be done in $O(m \log m) = O(n)$, for a total complexity of $O(n)$ for this part.

Third, we compute the hash value in a process with $n - m + 1$ iterations. The i-th iteration ($0 \leq i \leq n - m$) is as follows:

1. At the beginning, the tape contains $H_{i-1}(x[1 \ldots m - 1 + i])$, followed by the $m + i$-th charcater of x with a mark on it, followed by the remaining characters of x.

2. We scan the tape, find the marked character, move the mark one cell to the right and copy the value $H_{i-1}(x[1 \ldots m - 1 + i])||x[m + i]$ into a separate track.

3. We compute $h(H_{i-1}(x[1 \ldots m - 1 + i])||x[m + i]) = H_i(x[1 \ldots m + i])$ in a the separate track, cleaning it after the execution.

4. We copy the value of $H_i(x[1 \ldots m + i])$ back into the main track, so that the tape now contains $H_i(x[1 \ldots m + i])$, followed by the $m + i + 1$-th character with a mark, followed by the rest of characters of x.

After $n - m + 1$ iterations, we have the value of $H_{n-m}(x)$ in the tape, as desired. Copying the values from one track to the other can be done in $O(m)$ time; the most expensive part of one iteration is computing h, which is done in $O(m^k)$ time. Therefore, the total time complexity is $O(n \cdot m^k) = O(n^{1+\epsilon})$.

□

Theorem 4.12. *Assuming the existence of keyless collision-resistant hash functions, there does not exist a* BPP*-constructive separation of* PAL *against 1-DTMs running in time $O(n^{1+\epsilon})$ for any $\epsilon > 0$.*

Proof. Construct a machine M that computes the hash function $H = H_{\epsilon,\epsilon}$ of Theorem 4.11 on the first half on the input and on the reverse of the second half of the input, and accepts if they are equal. Note that dividing the input into two halves can be done in $O(n \log n)$ time by moving a counter, and that the computation of H can be done on the reverse of the second half without needing to explicitly write the reverse of the second half of the tape. Comparing the two hashes can be done in $O(mn) = O(n^{1+\epsilon})$, so the machine M runs in time $O(n^{1+\epsilon})$.

Any counterexample against PAL for M must be of the form $x_1 x_2^R$ or $x_1 b x_2^R$, where $x_1 \neq x_2$ and $\langle x_1, x_2 \rangle$ is a collision for H. Thus, any polynomial-time refuter R that outputs counterexamples against M can be transformed into an efficient adversary that outputs collisions for H, contradicting that H is collision-resistant. □

Note that the machine M we constructed in Theorem 4.12 accepts all palindromes (and therefore fails only on non-palindromes), so the result also applies to BPP-constructive separations against 1-TMs promised to fail only on non-palindromes. This offers an interesting contrast with Theorem 4.7: refuting 1-NTMs promised to fail only on palindromes proves circuit lower bounds, while refuting 1-TMs promised to fail only on non-palindromes proves non-existence of cryptographic primitives, which is an "upper bound".

4.4 Constructive Separations Against Weaker 1-TMs

One can wonder whether the results of Theorem 4.7 are "tight", in the sense of requiring the minimal possible assumptions on the hypotheses (such as the resources used by the one-tape TMs being refuted) in order to obtain a breakthrough result via a constructive separation. It would be interesting to find refuters when each of the hypotheses are slightly weakened, in order to establish a "threshold" result which sets a dividing line between the situations where refuters can be found and the situations where proving the existence of refuters is as hard as major open problems.

In this section, we show that indeed there exist such refuters for certain weakenings of the hypotheses of Theorem 4.7.

4.4.1 Refuters Against $o(n \log n)$-time 1-TMs

The most natural weakening of the hypotheses of Theorem 4.7 is to consider deterministic 1-TMs rather than non-deterministic ones, since they are a more natural model of computation, and to consider deterministic refuters for them. The results of the above section show that we can not hope to find such refuters against $O(n^{1+\epsilon})$-time machines, but we can wonder what happens with, say, $O(n)$-time machines.

It turns out that even nondeterministic $O(n)$-time (even $o(n \log n)$-time) one-tape machines are very limited in computational power, which allows us to obtain simple deterministic refuters for nondeterministic machines.

Theorem 4.13 ([Har68])**.** *Any language recognized by a 1-NTM running in $o(n \log n)$ time is regular.*

Theorem 4.14. *There is a* P*-constructive separation of* PAL *against $o(n \log n)$-time 1-NTMs.*

As usual, for simplicity we will focus only on refuting for even n. To construct the refuter, we first show that "the first half" of the counterexamples are generated by finite automata:

Write in introduction notation used for automata

Lemma 4.15. *Given $L \in \mathsf{REG}$ over any alphabet Σ, the following languages are also regular:*

1. $L_1 := \{w : ww^R \in L\}$.
2. $L_2 := \{w : \exists \tilde{w} \in \Sigma^*, |w| = |\tilde{w}|, w \neq \tilde{w}, w\tilde{w}^R \in L\}$.

Proof. Let $A(L) = (Q^0, q_0^0, \delta^0, S^0)$ be the DFA for L.

We construct the DFA $A(L_1) = (Q^1, q_0^1, \delta^1, S^1)$ for L_1 by setting

$$Q^1 = Q^0 \times 2^{Q^0},$$
$$q_0^1 = (q_0^0, S^0),$$
$$S^1 = \{(q, S) : q \in S\},$$
$$\delta^1((q, S), x) = (\delta^0(q, x), \{q' : \delta^0(q', x) \in S\}).$$

Informally, we are simulating the DFA $A(L)$, but each time a character is read we are also "moving" the set of accepting states in the inverse direction of the transitions.

We can easily verify by induction that the following property is satisfied: $A(L_1)$ is in state (q, S) after reading word w if and only if $A(L)$ is in state q after reading w and S is the set of states s that satisfy that, if $A(L)$ reads w^R starting at state s, then it ends in an accepting state in S^0. As a corollary, $A(L_1)$ accepts w if and only if $A(L)$ accepts ww^R, as desired.

We construct the NFA $A(L_2) = (Q^2, q_0^2, \delta^2, S^2)$ for L_2 by setting

$$\begin{gathered}
Q^2 = Q^0 \times 2^{Q^0} \times \{0,1\}, \\
q_0^2 = (q_0^0, S^0, 0), \\
S^2 = \{(q, S, 1) : q \in S\}, \\
\delta^2((q,S,0),x) = \{(\delta^0(q,x), \{q' : \delta^0(q',x) \in S\}, 0), (\delta^0(q,x), \{q' : \exists y \neq x\, \delta^0(q',y) \in S\}, 1)\}, \\
\delta^2((q,S,1),x) = \{(\delta^0(q,x), \{q' : \exists y\, \delta^0(q',y) \in S\}, 1)\}.
\end{gathered}$$

Informally, we simulate the DFA $A(L)$ as before, but this time the word $\tilde{w}$ we are reading "backwards" (by moving the set of accepting states) needs to be different from the word w we are reading forwards, so we divide the states in two stages depending on whether there is already at least one different character in the partial words w and $\tilde{w}$ or not.

The invariant property that is satisfied in this case is:

- $A(L_2)$ can reach state $(q, S, 0)$ after reading word w if and only if $A(L)$ is in state q after reading w and S is the set of states s that satisfy that, if $A(L)$ reads w^R starting at state s, then it ends in an accepting state in S^0, and
- $A(L_2)$ can reach state $(q, S, 1)$ after reading word w if and only if $A(L)$ is in state q after reading w and S is the set of states s that satisfy that, there exists $\tilde{w} \in \Sigma^*, |w| = |\tilde{w}|, w \neq \tilde{w}$ so that if $A(L)$ reads $\tilde{w}^R$ starting at state s, then it ends in an accepting state in S^0.

This implies that there is an accepting execution for w in $A(L_2)$ if and only if there exists $\tilde{w} \in \Sigma^*, |w| = |\tilde{w}|, w \neq \tilde{w}$ so that $A(L)$ accepts $w\tilde{w}^R$, as desired.

□

Proof of Theorem 4.14. Let $L \in \mathsf{REG}$ be the language recognized by the 1-NTM. By Lemma 4.15, the languages $L_1 := \{w : ww^R \notin L\}$ and $L_2 := \{w : \exists \tilde{w} \in \Sigma^*, |w| = |\tilde{w}|, w \neq \tilde{w}, w\tilde{w}^R \in L\}$ are also regular. We assume we have access to the automata A_{L_i} recognizing those languages. Note that these automata are of size $O(1)$.

Our refuter is described in Algorithm 2.

Algorithm 2 Refuter of Theorem 4.14

```
Input: $1^n$
Require: $n$ even.
Hardcode: $A_{L_1} \leftarrow$ DFA corresponding to $L_1$.
Hardcode: $A_{L_2} \leftarrow$ DFA corresponding to $L_2$.
Program:
  $w \leftarrow \textsc{FindWord}(A_{L_1}, n/2)$  ▷ $\textsc{FindWord}(A, \ell)$ finds a word of length $\ell$ accepted by automaton $A$.
  if Found $w$ then
      output $ww^R$
      HALT
  end if
  $w \leftarrow \textsc{FindWord}(A_{L_2}, n/2)$
  if Found $w$ then
      $u \leftarrow \textsc{ExtendWord}(A_{L_2}, w, ww^R, n)$  ▷ $\textsc{ExtendWord}(A, p, v, \ell)$ finds a word of length $\ell$, with prefix $p$ and different from word $v$, accepted by automaton $A$.
      output $u$
      HALT
  end if
  HALT
```

FindWord can be implemented in time $O(n)$ by constructing the product automaton for the language $L(A) \cap \{0,1\}^\ell$. Similarly, ExtendWord can be implemented in time $O(n)$ by constructing the corresponding product automaton and finding a second accepting path if the first found one results in the undesired word (the graph of the product automaton is acyclic, so finding a second path can be done simply by e.g. a DFS traversal).

Thus, the refuter runs in polynomial (in fact, linear) time and that it outputs counterexamples for infinitely many values of n (for any even value of n for which there is a counterexample, it finds one, and for big enough n there are always counterexamples by Corollary 4.5). □

Note that the refuter can be made black-box by learning the regular language recognized by the TM using black-box queries. In fact, if we iterate over all automata of size (say) $\log\log n$ and output the counterexample found for each of the automata, our algorithm still runs in polynomial time, so we have an explicit obstruction against all $o(n \log n)$ 1-TMs.

can it be done in logspace??

4.4.2 Refuters Against $O(n \log n)$-time 1-TMs

One-tape Turing machines running in $n \log n$ time have much more power than finite automata. And non-deterministic machines running in $n \log n$ time are strictly more powerful than deterministic machines: for example, the language of non-palindromes can be recognized in $O(n \log n)$ non-deterministic time by guessing the index of the differing character and moving the $O(\log n)$ bits of the index around the tape, while $\Omega(n^2)$ is required for deterministic machines to recognize this language as it is the complement of a language requiring $\Omega(n^2)$ time.

Still, surprisingly, we can find a refuter against palindromes for these machines:

Theorem 4.16.

- *There exists a* P^{NP}*-constructive separation of* PAL *against one-tape nondeterministic Turing machines running in time* $O(n \log n)$*.*
- *There exists a* P*-constructive separation of* PAL *against one-tape deterministic Turing machines running in time* $O(n \log n)$*.*

We say that a crossing sequence is *short* if it has length at most $4K \log n$. Our refuter works by repeatedly generating palindromes and running the machine on them until either a non-accepting palindrome or a collision among short crossing sequences is found. The algorithmic procedure is described in detail in Algorithm 3; we proceed to explain how it works.

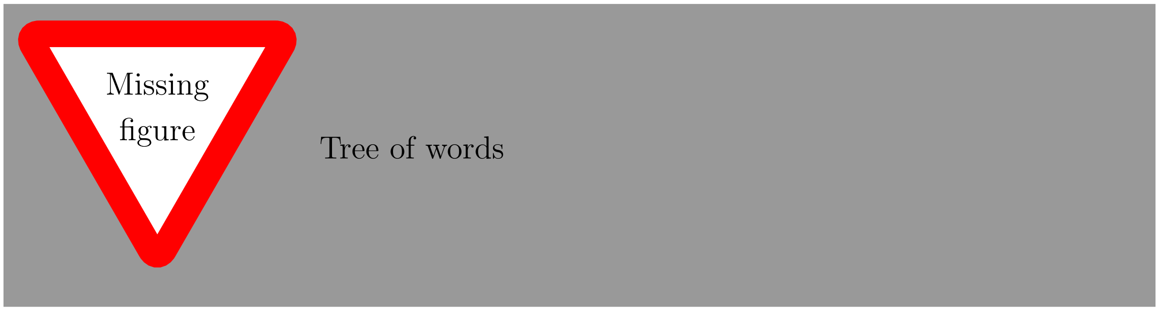
Explain empty word notation in introduction

Algorithm 3 Refuter of Theorem 4.16

```
Input: $1^n$
Require: $n$ even.
Hardcode: $M$ 1-TM to be refuted.
Hardcode: $K \leftarrow$ constant such that $M$ runs in $Kn \log n$ steps.
Hardcode: $s \leftarrow$ number of states of $M$.
Program:
  $L \leftarrow 4K \log n$
  MarkedPrefixes $\leftarrow \{\}$
  SequenceByPrefix $\leftarrow []$
  repeat $4 \cdot s^{L+1}$ times:
      $w \leftarrow$ ChoosePalindrome($n$, MarkedPrefixes)
      run $M$ on input $ww^R$, and set:
          Accepted $\leftarrow$ result (whether $M$ accepts or not).
          $C \leftarrow$ crossing sequences of the execution.
      end
      if Accepted then
          for $i \leftarrow 1 \dots n/2$ do
              $p \leftarrow w[1 \dots i]$
              if $|C_i| \leq L$ and $p \notin$ MarkedPrefixes then
                  for $\tilde{p} \in$ MarkedPrefixes do
                      if $|\tilde{p}| = i$ and SequenceByPrefix$[\tilde{p}] = C_i$ then
                          $v \leftarrow w(i \dots n/2]$
                          output $\tilde{p}vw^R$
                          Halt
                      end if
                  end for
                  MarkedPrefixes $\leftarrow$ MarkedPrefixes $\cup \{p\}$
                  SequenceByPrefix$[p] \leftarrow C_i$
              end if
          end for
      else
          output $ww^R$
          Halt
      end if
  end
  Halt
```

Algorithm 4 Procedure ChoosePalindrome for Algorithm 3

```
function ChoosePalindrome(n, $S$)
    Weight $\leftarrow$ []        ▷ Associative array of integers. If Weight is accessed for a key
  that has not been defined yet, it is initialized with default value 0.
    VerticesByLength $\leftarrow$ Array$(n/2)$                    ▷ Array of sets.
    for $v \in S$ do
        VerticesByLength$[|v|]$ $\leftarrow$ VerticesByLength$[|v|] \cup \{v\}$
    end for
    for $\ell \leftarrow n/2 \ldots 1$ do
        for $v \in$ VerticesByLength$[\ell]$ do
            if $v \in S$ then
                Weight$[v]$ $\leftarrow$ Weight$[v] + 1$
            end if
            $p \leftarrow$ Pop$(v)$
            if Weight$[p] = 0$ then
                Weight$[p]$ $\leftarrow$ Weight$[v]$
                VerticesByLength$[\ell - 1]$ $\leftarrow$ VerticesByLength$[\ell - 1] \cup \{p\}$
            else
                Weight$[p]$ $\leftarrow \min\{$Weight$[p]$, Weight$[v]\}$
            end if
        end for
    end for
    $w \leftarrow \lambda$
    repeat $n/2$ times:
        if Weight$[w||0] \leq$ Weight$[w||1]$ then
            $w \leftarrow w||0$
        else
            $w \leftarrow w||1$
        end if
    end
    return $w$
end function
```



The program starts by initializing L to the maximum length of short sequences (this will simply act as a shorthand for the expression $4K \log n$ later, it is not a variable), and by creating two empty containers, `MarkedPrefixes` and `SequenceByPrefix`. `MarkedPrefixes` will be a set containing prefixes of palindromes accepted by M for which the M generated a short crossing sequence at the right end of the prefix in an accepting execution. `SequenceByPrefix` will be an associative array mapping each prefix in `MarkedPrefixes` to the corresponding short crossing sequence.

Next, we have the main loop, in which in each iteration we choose a palindrome ww^R and run M on it. We choose the palindrome using the procedure ChoosePalindrome, described in Algorithm 4. This procedure returns a word w of length $n/2$, and the corresponding palindrome will be ww^R. What this procedure does is to choose a w minimizing the number of prefixes of w which are in `MarkedPrefixes`. It is easier to visualize it by thinking of a binary tree of height $n/2$ whose vertices correspond to all binary words of length up to $n/2$, the root being the empty word and the two children of each vertex corresponding to the word of the parent concatenated with each of the two characters in the alphabet.

In this setting, `MarkedPrefixes` would correspond to marked vertices in the tree, and we want to find a path from the root to a leaf visiting the minimum number of marked vertices. We can compute the minimum number of marked vertices in such a path by an easy recursive formula:

$$\texttt{Weight}[v] = \texttt{Weight}[\text{child}_1(v)] + \texttt{Weight}[\text{child}_2(v)] + [1\text{if } v \text{ is marked}],$$

and once we have computed it for all vertices, find a path with minimum weight by repeatedly descending to the child with smaller weight. However, we can not explore the whole tree since it is of exponential size. ChoosePalindrome does this in a "bottom-up" way so that only vertices with weight greater than 0 are explored. See the pseudocode description at Algorithm 4 for details.

After choosing the palindrome and running it, if M does not accept we can output it as a counterexample. If M does accept it, we analyze the crossing sequences at the first half of the input. If there is a prefix with a short crossing sequence that we have not marked yet, we iterate over all other marked prefixes of the same size. If we had marked a different prefix with the same short crossing sequence, we can replace the current prefix by that one and we obtain a non-palindrome that is accepted, which we can output as a counterexample. If not, we add it to the `MarkedPrefixes` and `SequenceByPrefix` containers.

We claim that, for all big enough n, we will always find a counterexample repeating this $4 \cdot s^{L+1}$ times, and that the whole algorithm always runs in polynomial time. Let us prove some simple observations first:

Lemma 4.17. *Let M be a 1-TM running in time $Kn \log n$, and let w be an accepted word of even length n. Then, for at least $n/4$ of the indices $1 \leq i \leq n/2$, $|C_i(x)| \leq 4K \log n$.*

Proof.

$$Kn \log n \geq \sum_{i=1}^{n/2} |C_i(x)| \geq 4K \log n \cdot (n/2 - |\{i : |C_i(x)| \leq 4K \log n\}|)$$

$$\implies |\{i : |C_i(x)| \leq 4K \log n\}| \geq n/4.$$

□

Lemma 4.18. *Let M be a 1-TM. For n large enough (depending on M), for every call to* ChoosePalindrome *during the execution of the refuter of Algorithm 3 against M, the word w returned by* ChoosePalindrome *satisfies the following: there exist at least $n/8$ indices $1 \leq i \leq n/2$ so that $|C_i(ww^R)| \leq 4Klogn$ and the prefix $w[1 \ldots i]$*

does not already belong to `MarkedPrefixes`*. (In other words, at least $n/8$ prefixes are added to* `MarkedPrefixes` *in each iteration, if no counterexample is found).*

Proof. There are $4 \cdot s^{L+1}$ iterations and in each iteration at most $n/2$ prefixes are added to `MarkedPrefixes`. Therefore, at any time, there are at most $2n \cdot s^{L+1}$ prefixes in `MarkedPrefixes`. We claim that the word returned by $\textsc{ChoosePalindrome}(n, S)$ has at most $1 + \log |S|$ prefixes in S, which for our calls with $S =$ `MarkedPrefixes` is at most $1 + \log(2n \cdot s^{L+1}) = O(\log n)$.

To prove so, consider the following greedy algorithm to find a path from the root of the tree to a leaf passing through not too many marked vertices which could have been an alternative implementation for ChoosePalindrome: at each vertex, descend to the child with the least amount of marked vertices in its subtree. It is clear that after each step the number of marked vertices in the current subtree is halved, so after $1 + \log |S|$ steps there are no marked vertices in the subtree. Therefore, there exists a path with at most $1 + \log |S|$ marked vertices, and since ChoosePalindrome chooses the path with the minimum number of marked vertices, its result must have at most that many marked vertices.

To finish, by Lemma 4.17 the result w has at least $n/4$ indices $0 \leq i \leq n/2$ with $|C_i(ww^R)| \leq 4K \log n$, so at least $n/4 - O(\log n) > n/8$ (for big enough n) of them are not in `MarkedPrefixes`. □

Lemma 4.19. *Let M be a 1-TM. For n large enough (depending on M), the execution of the refuter of Algorithm 3 against M outputs a word x such that $M(x) \neq \mathrm{PAL}(x)$.*

Proof. It is clear that whenever the refuter outputs a word, it is a correct counterexample. Therefore, we have to prove that for n large enough the refuter always outputs a counterexample. Suppose not, that for some n large enough to apply Lemma 4.18 the refuter does not output anything after the $4 \cdot s^{L+1}$ iterations. Then, by Lemma 4.18, `MarkedPrefixes` has at least $n/8 \cdot 4 \cdot s^{L+1}$ elements at the end of the execution. That means that for some $1 \leq i \leq n/2$, there must be at least s^{L+1} of length i. There are $1 + s + \ldots + s^L < s^{L+1}$ crossing sequences of length at most L, so there are two prefixes of length i with the same short crossing sequence. But that would have been detected during the execution of the program and a counterexample would have been printed, contradiction. □

Proof of Theorem 4.16. The correctness of the refuter is established by Lemma 4.19. We now need to prove that it runs in time polynomial in n. The number of iterations is $4{\cdot}s^{L+1} = 4{\cdot}n^{4K \log s}$, polynomial in n. Inside each iteration, $\textsc{ChoosePalindrome}(n, S)$ runs in $O(n{\cdot}|S|)$, which is polynomial (by the previous argument that `MarkedPrefixes` always has an at most polynomial number of elements), running M is $O(n \log n)$ (with possibly a call to an NP oracle), and the subsequent iteration over $i \leftarrow 1 \ldots n/2$ and all elements of `MarkedPrefixes` is also polynomial.

Therefore, the refuter runs in polynomial time and we have a P-constructive (or P^{NP}-constructive) separation. □

4.4.3 Additional Aspects and Consequences of the Refuters

We can modify our refuter to work against machines that run in more than $n \log n$ time; we just have to modify the definition of "short sequence" to ones that are shorter than $4T(n)/n$, where $T(n)$ is the runtime of the machine. Our refuter then no longer runs in polynomial time in that case, though:

Theorem 4.20. *For each 1-NTM M running in time $O(n \cdot f(n))$ with $f(n) = o(n)$, there is a refuter of* PAL *against M running in time $n \cdot 2^{O(f(n))}$ with a* NP *oracle.*

Corollary 4.21. $\mathsf{DTIME}[2^{n+O(f(2^n))}] \not\subset \mathsf{SIZE}[f(2^{n/2})^{\delta}]$.

Note that by taking $f(n) = (\log n)^k$, the DTIME-class bound we get is similar to the bound implicitly obtained by the classic diagonalization used to show $\Sigma_4^{\mathsf{P}} \not\subset \mathsf{SIZE}(n^k)$ [Kan82]. This refuter is too weak to give new circuit lower bounds, but on in some sense it seems to be "close". For example, if we could refute $O(n\text{polylog}(n))$ 1-NTMs still using polynomial time, then we would prove $\mathsf{E}^{\mathsf{NP}} \not\subset \mathsf{SIZE}(n^k)$, which is not known. We now have a refuter against all $O(n^{1+o(1)})$ 1-TMs using subexponential ($2^{n^{o(1)}}$) time; if we could refute $O(n^{1+\epsilon})$ in that time, we would get $\mathsf{DTIME}[2^{2^{o(n)}}] \not\subset \mathsf{SIZE}[2^{\delta n}]$.

Therefore, it seems difficult to improve this refuter. However, there is some evidence that it does admit improvements. First, for the ease of exposition here we have explained a version of the refuter which is white-box on the machine being refuted (indeed, it is a constructive separation in the sense of [Che+24] which has access to uncomputable information about the machine M, such as the constant K in its runtime), but it can be improved to be black-box (with oracle access to M). Here we briefly explain which aspects need to be modified:

- The refuter depends on the values of s and K, which can not be obtained by black-box access. However, since we can verify whether the output is a valid counterexample, we can just iterate through $(s, K) = (1, 1), (2, 2), \dots, (n, n)$, stopping as soon as we obtain a valid counterexample. This makes the refuter no longer be a polynomial-time algorithm as a general oracle-algorithm, but for any fixed 1-TM M given as an oracle, it will be a polynomial-time algorithm, since for n big enough the refuter will always stop at a fixed, constant value of s and K.

- The refuter depends on knowing the crossing sequences of the execution in order to choose a new palindrome which "minimizes the overlap" with the short crossing sequences obtained by previous palindromes. However, as we hinted at before, this is not really necessary: indeed, if we just generate a fixed set of palindromes in which the $O(\log n)$ first bits are all different, we still get that any two palindromes overlap in at most $O(\log n)$ prefixes, which is enough for our purposes.

- The refuter also depends on knowing the crossing sequences in order to find a collision and to construct the counterexample. However, we do know that a collision among the polynomially-many generated palindromes will exist after all the iterations, so we can just check all pairs of palindromes and all indices $i = 1, \dots, n/2$, in polynomial time in total.

Therefore, we have a polynomial-time refuter against $O(n \log n)$-time 1-TMs that only needs oracle access to M (and in particular it does not need an NP oracle for

1-NTMs). And we do not even take advantage of much information about the oracle calls to M: we simply use them to "test candidates" for counterexamples, stopping when we find a correct counterexample but otherwise not being adaptive at all in the queries!

However, do note that, since we need to stop at constant (s, K) in order to run in polynomial time, we do not have a polynomial-time explicit-obstructions refuter, like we did in the $o(n \log n)$ setting. So we do lose something when going from $o(n \log n)$ to $\Theta(n \log n)$. Still, given the austerity of our refuter when it comes to using information about the specific machine we are refuting, it is difficult to believe we can not do better.

Open Question 3. Is it possible to obtain a better P^{NP}-constructive separation, even if not good enough to obtain new circuit lower bounds, by taking advantage of white-box access and the NP oracle?

It is of note that our refuters work for non-deterministic 1-TMs in exactly the same way as they do for deterministic ones, and they work for all $O(n \log n)$ 1-TMs, not just those that accept only palindromes. Theorem 4.7 might reduce our hopes of obtaining great progress against non-deterministic machines, since that would imply breakthrough circuit lower bounds, but progress in refuters against deterministic 1-TMs doesn't seem easy either, and yet we do not have any explicit reason why. Theorem 4.12 explains why it shouldn't be possible to have a refuter for all $O(n^{1+\epsilon})$ deterministic 1-TMs, but in the setting of 1-TMs which only accept palindromes, which is linked to "plausible" derandomization, we do not have any obstruction.

Open Question 4. Can we get a better P-constructive separation against deterministic 1-TMs promised to accept only palindromes? If not, can we get any result saying that it should be as difficult as obtaining certain derandomization, like we have for non-deterministic machines?

Bibliography

[ACR98] Alexander E. Andreev, Andrea E. F. Clementi, and José D. P. Rolim. "A new general derandomization method". In: *J. ACM* 45.1 (Jan. 1998), pp. 179–213.

[Ats06] A. Atserias. "Distinguishing SAT from polynomial-size circuits, through black-box queries". In: *21st Annual IEEE Conference on Computational Complexity (CCC'06).* 2006, 8 pp.–95.

[BTW10] Andrej Bogdanov, Kunal Talwar, and Andrew Wan. "Hard Instances for Satisfiability and Quasi-one-way Functions". In: *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings.* Ed. by Andrew Chi-Chih Yao. Tsinghua University Press, 2010, pp. 290–300.

[CEG95] Ran Canetti, Guy Even, and Oded Goldreich. "Lower bounds for sampling algorithms for estimating the average". In: *Information Processing Letters* 53.1 (1995), pp. 17–25.

[Che+24] Lijie Chen et al. "Constructive Separations and Their Consequences". In: *TheoretiCS* Volume 3 (Feb. 2024).

[CLO24] Lijie Chen, Jiatu Li, and Igor C. Oliveira. "Reverse Mathematics of Complexity Lower Bounds". In: *Electronic Colloquium on Computational Complexity* (2024).

[Dam90] Ivan Bjerre Damgård. “A Design Principle for Hash Functions”. In: *Advances in Cryptology — CRYPTO’ 89 Proceedings*. Ed. by Gilles Brassard. New York, NY: Springer New York, 1990, pp. 416–427. ISBN: 978-0-387-34805-6.

[Har68] J. Hartmanis. “Computational Complexity of One-Tape Turing Machine Computations”. In: *J. ACM* 15.2 (Apr. 1968), pp. 325–339.

[Hen65] F. C. Hennie. “One-Tape, Off-Line Turing Machine Computations”. In: *Inf. Control.* 8.6 (1965), pp. 553–578.

[HH23] Pooya Hatami and William Hoza. “Theory of unconditional pseudorandom generators”. In: *Electron. Colloquium Comput. Complex., TR23-019.* 2023.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. “Pseudorandomness for network algorithms”. In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing.* STOC ’94. Montreal, Quebec, Canada: Association for Computing Machinery, 1994, pp. 356–364. ISBN: 0897916638.

[IW97] Russell Impagliazzo and Avi Wigderson. “P = BPP if E requires exponential circuits: derandomizing the XOR lemma”. In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing.* STOC ’97. El Paso, Texas, USA: Association for Computing Machinery, 1997, pp. 220–229. ISBN: 0897918886.

[Kan82] R. Kannan. “Circuit-size lower bounds and non-reducibility to sparse sets”. In: *Information and Control* 55.1 (1982), pp. 40–56.

[Mer90] Ralph C. Merkle. “One Way Hash Functions and DES”. In: *Advances in Cryptology — CRYPTO’ 89 Proceedings*. Ed. by Gilles Brassard. New York, NY: Springer New York, 1990, pp. 428–446. ISBN: 978-0-387-34805-6.

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