TFNP Hardness Talk Background

Presenter: Akshat Yaparla
Supporter: Mark Chen

8 February 2024

This document provides a background for this Thursday's (8 February) lecture on "The Journey from NP to TFNP hardness" [HNY17]. The first section provides the reader with an overview on the various PRGs and what they imply in terms of Impagliazzo's worlds. It draws heavily on notes from Prof. Dana Moshkovitz's course "Advanced Complexity Theory" ([2], [3]). The second section provides a detailed overview on interactive proof systems, concluding with Zaps. The lecture will utilize background from at least one of these sections.

Contents

1 Evolution of PRGs

1.1 Yao's Theorem

Theorem 1 (Yao)**.** *If a polynomial time $\frac{1}{10}$-PRG $G_c : \{0,1\}^{s(n)} \to \{0,1\}^n$ against $O(n^c)$ size circuits, then*

$$\mathsf{BPP} \in \mathsf{DTIME}\left(2^{s(p(n))} \cdot p(s(p(n)) \cdot p(n)\right).$$

Proof. Recall the definition of a BPP. Given an algorithm $A \in \mathsf{BPP}$ and suppose $L \in A$ is a language in A, then we have the following:

$$x \in L \iff \Pr_{y \in \{0,1\}^{p(n)}} [A(x, y) = 1] > \frac{2}{3}$$
$$x \notin L \iff \Pr_{y \in \{0,1\}^{p(n)}} [A(x, y) = 1] < \frac{1}{3}$$

The goal is to construct a deterministic process to derandomize A using our polynomial time $\frac{1}{10}$-PRG. If we want to pick a seed $y \leftarrow \{0,1\}^{p(n)}$ at random, we can use this PRG to achieve a similar effect of the uniform sampling by definition, which, on average, should have

$$\Pr_{z \in \{0,1\}^{s(p(n))}} [A(x, G(z)) = 1] .$$

By the definition of a PRG, we know that

$$\left| \Pr_{z \in \{0,1\}^{s(p(n))}} [A(x, G(z)) = 1] - \Pr_{y \in \{0,1\}^{p(n)}} [A(x, y) = 1] \right| < \frac{1}{10},$$

implying

$$x \in L \iff \Pr_{z \in \{0,1\}^{s(p(n))}} [A(x, G(z)) = 1] > \frac{2}{3} - \frac{1}{10} > \frac{1}{2}$$
$$x \notin L \iff \Pr_{z \in \{0,1\}^{s(p(n))}} [A(x, G(z)) = 1] < \frac{1}{3} + \frac{1}{10} < \frac{1}{2}$$

With these two bounds, we can do standard amplification and confirm that this algorithm is indeed also in BPP. To make it deterministic, we simply run all possible instances in $\{0,1\}^{s(p(n))}$ (i.e. $2^{s(p(n))}$ instances). □

Runtime

We need to run all the $2^{s(p(n))}$ instances of z, each of which takes $\text{poly}(s(p(n)))$-time to generate. Also, in each iteration, we need to run $A(x, G(z))$, which is also poly-time, so it takes $p(n)$-time. The total runtime is thus:

$$\mathsf{BPP} \in \mathsf{DTIME}\left(2^{s(p(n))} \cdot p(s(p(n)) \cdot p(n)\right)$$

Observation 2. *If our seed length was* $\log n$*, that is,* $s(n) = O(\log(n))$*,*

$$\begin{aligned} \mathsf{BPP} &\in \mathsf{DTIME}\left(2^{s(p(n))} \cdot p(s(p(n)) \cdot p(n)\right) \\ &= \mathsf{DTIME}\left(2^{O(\log(p(n)))} \cdot p(O(\log(p(n)))) \cdot p(n)\right) \\ &\subseteq \mathsf{P} \end{aligned}$$

Since $\mathsf{P} \subseteq \mathsf{BPP}$ *is a trivial direction, we then have*

$$\mathsf{P} = \mathsf{BPP}.$$

1.2 Blum-Micali Generators

Blum, Micali, and Yao did not believe that such a PRG could exist as $s(n) = O(\log(n))$ is a small constraint. As a result, they eased the requirement for the generator to

$$G : \{0,1\}^{\sqrt{n}} \to \{0,1\}^n.$$

In this case, they were able to use the idea of one-way permutations to let G run in $\text{poly}(n)$ time but fool all $\text{poly}(n)$ time distinguishers.

Recall the definition of a one-way permutation (OWP).

Definition 3 (OWP)**.** *A bijective function $f : \{0,1\}^n \to \{0,1\}^n$ such that f is computable in* $\text{poly}(n)$*, but its inverse f^{-1} is not except for negligible probability over a uniform* BPP *challenger.*

The Blum and Micali generator first uses a random seed to generate a random string z. Then, utilizing an OWP f, the generator repeatedly applies f to the current string and uses the first bit to add to the output of the generator. A visual is given on Figure 1. Formally,

Definition 4. *The Blum-Micali (BM) Generator $g : \{0,1\}^k \to \{0,1\}^n$ is given by*

$$g(y)_i = f^{(i)}(z)_1,$$

where z is a string yielded from the seed $y \leftarrow \{0,1\}^k$.

Theorem 5. *The BM generator g is a pseudorandom generator for* P *using $f = RSA$ and some additional assumptions.*

Proof. Omitted. □

However, there is a drawback to the generator. If we desired a seed $k = O(\log(n))$ for the BM generator, then we can invert $f : \{0,1\}^k \to \{0,1\}^k$ in $\text{poly}(n)$ time by simply trying all possible inputs to f and seeing which one results in the string that we want to invert.

1.3 Nisan-Wigderson Generators

The *Nissan-Wigderson (NW) generator* aims to rectify this issue with the BM generator by using a function f that is easy to compute, but not *too* easy to compute. An NW generator G is one that

- runs slower than the D that G is trying to fool, but
- also stretches the randomness of the seed $s(n)$ such that $s(n) = O(\log(n))$. Hence, the time $t(n)$ of G is exponentially larger than $s(n)$.

Before introducing the assumptions under which such a generator can be realized, we introduce what it means for a function to be *hard* for a class of circuits.

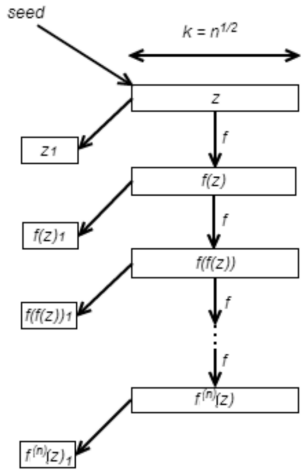

Figure 1: A visual depicting how the BM generator works.

Definition 6 (ϵ-hardness)**.** *A function $f : \{0,1\}^k \to \{0,1\}$ is said to be ϵ-hard for size m if for all circuits C of size at most m such that*

$$\left| \Pr_{x \leftarrow \{0,1\}^k} [f(x) = C(x)] \right| \leq \epsilon$$

Lemma 7. *If a function $f : \{0,1\}^k \to \{0,1\}$ is ϵ-hard for size 2^k for $\epsilon = \exp(-k)$, then the random bit string k looks random to $\mathsf{P/poly}$, the class of $\mathrm{poly}(n)$-sized circuits.*

Definition 8 (Nisan-Wigderson (NW) Assumption)**.** *A generator G can be constructed assuming that there exists a function $f : \{0,1\}^k \to \{0,1\}$ such that*

- *f is computable in 2^{100k}.*
- *f is ϵ-hard for size $2^{k/100}$ for any $\epsilon \leq \exp(-k)$.*

This assumption is called the NW assumption.

Theorem 9 (NW Generator)**.** *If the NW assumptions are true, then there exists $G : \{0,1\}^{O(\log(n))} \to \{0,1\}^{O(n)}$ s.t. G can be compute in poly-time and G 0.1-fools all n^2-size circuits.*

Proof. Provided in [2] and [3]. In summary, the NW PRG satisfies the bound concluded by Yao's theorem while avoiding the problem that exists for BM generators. □

1.4 PRGs in the Worlds of Cryptography

[HNY17] starts by only assuming NP-hardness, placing us in the world of *Pessiland*. From this, they deduce that TFNP is nonuniform hard. To extend this to hard-on-average TFNP, we have to assume the NW assumption to be true. Since this does not imply the existence of OWFs, we *could* still remain in Pessiland with this assumption.

2 Interactive Proof Protocols

An **interactive proof protocol** is a series of interactions between a verifier, denoted V, that is usually constrained to polynomial time computation, and a prover, denoted P, that has unbounded computational power. The basic principles behind such a protocol are

- (*Completeness*): Every true theorem should have a proof that can be accepted by the verifier.
- (*Soundness*): Every false theorem should not have a proof that can be accepted by the verifier.
- (*Efficiency*): No matter how hard a proof is for the prover to come up with, the verifier should be able to efficiently verify the proof that the prover gives.

Depending on the specific complexity class, either P or V may be the first to send a message to the other. An interaction between the prover and verifier is as follows. Say V acts first. Then, it is given an instance of a theorem to prove, say x. After k rounds of iteration, we then have

$$\begin{aligned} a_1 &= V(x) \\ a_2 &= P(x, a_1) \\ a_3 &= V(x, a_1, a_2) \\ &\vdots \\ a_{2k+1} &= V(x, a_1, \ldots, a_{2k}) \end{aligned}$$

Note that a "round of interaction" means that the prover and verifier exchange messages with each other once each. The final output of V after all rounds of interaction is then denoted as

$$\text{out}_P\langle V, P\rangle(x).$$

2.1 Complexity Classes

We can define a variety of complexity classes based on interactive proof systems. Each class differs by whether the verifier is allowed access to a random coin, the number of messages the prover and verifier exchange, the visibility of the verifier's random coins and whether the prover or verifier send the first message.

Deterministic Interactive Proof

In this protocol, the verifier is a P Turing Machine, or a deterministic polynomial time algorithm, and it sends the first message to the prover. The class of problems that can be decided on by such a protocol with polynomial rounds of interaction is called dIP. It ends up that

$$\mathsf{dIP} = \mathsf{NP}.$$

A simple proof is as follows. Note that we need to show both directions of inclusion.

- $\mathsf{NP} \subseteq \mathsf{dIP}$: For any $L \in \mathsf{NP}$, there must exist some certificate w that is checkable by a polynomial time verifier. We can then design a simple 1-message dIP protocol whereby the verifier sends the input $x \in L$ to the prover and the prover comes up with a certificate w to send back to the verifier. The verifier then runs $V(x, w)$, and returns 1 if w is a correct certificate.

- $\mathsf{dIP} \subseteq \mathsf{NP}$: In essence, all of the prover's messages to the verifier can just be minimized into one long, polynomial sized certificate $w = w_1 \dots w_k$, where $k = \text{poly}(n)$. Then, the verifier can collapse all of its decisions into a verification of this new long certificate. This turns out to just be an NP verifier.

Interactive Proof

This is the same as above except that the verifier is now a BPP algorithm rather than a P algorithm. In other words, it is allowed randomization. The analogous class to dIP is now called IP for all protocols of polynomial rounds of interaction. A rather surprising and important result is due to Shamir in 1992 which states that

$$\mathsf{IP} = \mathsf{PSPACE}.$$

$\mathsf{IP}[k]$ denotes an interactive proof protocol with k rounds of interaction.

Arthur-Merlin and Merlin-Arthur Protocols

Now, one can consider the classes where $\mathsf{IP}[k]$ is known to the prover. The main difference between these protocols are the order in who sends the first message, and the number of messages allowed to be sent.

- *Merlin-Arthur* protocols are those where Merlin (the prover) sends a message to Arthur (the verifier), and Arthur must then decide. The set of languages that have such a protocol belong to the class MA. Note that since Arthur does not communicate back with Merlin, he does not reveal his coins.

- *Arthur-Merlin* protocols are reversed from Merlin-Arthur, and associated languages are said to be in class AM. The main difference between Arthur-Merlin and IP is that Arthur must reveal his random coin flips to Merlin. A k-message AM protocol if referred to as $\mathsf{AM}[k]$, and if k is constant, it can be shortened down to 2 messages, similar to the method used to show $\mathsf{dIP} \subseteq \mathsf{NP}$. That is, $\mathsf{AM}[k] = \mathsf{AM}[2]$.

2.2 Public and Private Coin Protocols

Note that allowing the prover to flip coins gives it no advantage - since it has unbounded power, it can simply simulate outcomes of all coin flips. Since the verifier runs in polynomial time, it is allowed to flip coins, as this may provide it with more power (unless $\mathsf{P} = \mathsf{BPP}$). The verifier's coin can be either public or private. For a prover-verifier interactive proof (P, V),

- A *public-coin protocol* is one where the prover knows the result of the verifier's coin flips. That is, for the kth round of interaction, the verifier will send (a_{2k+1}, r) to P, where a_{2k+1} is the result of V's computation, and r are the random choices it makes. This is analogous to the AM protocol above.

- A *private-coin protocol* is one where the verifier does not make its random choices available to the prover. It will only send a_{2k+1} to P.

Below are private and public coin protocols for the NP language GraphNonIsomorphism (GNI).

- **(Private Coin)** Given graphs G_1 and G_2, the prover P wants to convince the verifier V that G_1 is not isomorphic to G_2.

Step 1: V flips a coin and get $i \in \{1, 2\}$ and chooses graph G_i. Then, it permutes G_i randomly with a polynomial number of coin flips. V sends the permuted graph H to P.

Step 2: P wants to decide which of G_1 or G_2 was permuted by V to create H. It chooses $j \in \{1, 2\}$ corresponding to graph G_j.

- If G_1 and G_2 are not isomorphic, then P correctly decides which graph was permuted to create H.
- If they are isomorphic, then the prover cannot decide which graph H is a permutation of. It will then choose j randomly, choosing correctly with probability 1/2. This process can be repeated to attain probability 1/4.

P then sends j to V. This protocol meets both completeness and soundness and thus is contained in private coin IP.

Step 3: V accepts if $i = j$ and rejects otherwise.

- **(Public Coin)** Consider that the prover also has knowledge about the coin-flipping result of the verifier in step 2, the above problem actually becomes in dIP. Here is how:

Step 1: The verifier flips a coin (then it can, but doesn't need to, do the same following steps as the previous case).

Step 2: The prover sees the coin flip, $i \in \{0, 1\}$. The prover, with unbounded power, figures out if G_1 is a graph isomorphism of G_2. Then, if it is, then the prover sends the verifier $j \neq i$; $i = j$ otherwise.

Step 3: Finally, the verifier accepts if $i = j$, otherwise it rejects.

This protocol satisfies both correctness and soundness, so we have that GNI $\in$ public coin IP.

It is important to note that the class of public-coin IP protocols is simply just an Arthur-Merlin protocol with polynomial rounds of interaction, or just AM[poly(n)]. Perhaps surprisingly, Goldwasser and Sipser in 1986 found that interactive proof protocols with polynomial number of messages with public or private coins are identical, meaning

$$\mathsf{AM}[\text{poly}(n)] = \mathsf{IP}.$$

Note that no such relation is known for a constant number of messages, for example $k = 2$.

2.3 Zero-Knowledge Proofs

Informally, a *Zero-Knowledge Proof* (ZKP) is a proof system where the prover P can prove to the verifier V the truth of some statement, while not leaking any additional information beyond the truth of the statement. ZKP are a subset of IP, where both the completeness and soundness requirements must be met, with an additional requirement of zero knowledge. That is, for all verifiers V', there exists a BPP algorithm S that can correctly simulate a "conversation" between P and V'.

2.4 Witness Indistinguishability

A *witness-indistinguishable proof* (WIP) is a type of ZKP with the "zero-knowledge" requirement relaxed. Here, any two proofs generated using two different witness are computationally indistinguishable. That is, a verifier that sees provers using different witnesses for the proof cannot distinguish between these proofs. Feige and Shamir, who developed WIPs, showed that with one-way functions, there exists a three-message WIP proof for every language in NP.

2.5 Zaps

The paper defines Zaps, using the Dwork and Naor ('07) definition, as "a two-message public-coin witness indistinguishable sceheme where he first message can be reused for all instances." Furthermore, it was shown that with a nonuniform distribution, the two messages can be condensed into a single message. Barak et al. showed that one message is also possible in the uniform setting by using a NW PRG for derandomizing known constructions of Zaps.

References

[1] Mark Braverman (2011) *COS597D: Information Theory in Computer Science*, Lecture 10, `https://www.cs.princeton.edu/courses/archive/fall11/cos597D/L10.pdf`

[2] Dana Moshkovitz (2012) *6.841: Advanced Complexity Theory*, Lecture 13, `https://people.csail.mit.edu/dmoshkov/courses/adv-comp/scribe13.pdf`

[3] Dana Moshkovitz (2012) *6.841: Advanced Complexity Theory*, Lecture 14, `https://people.csail.mit.edu/dmoshkov/courses/adv-comp/scribe14.pdf`