Derandomization from Hardness

CS254 Final Paper

March 2019

Introduction

Randomness has long been viewed as an indispensable component of efficient computation for many natural problems. We constantly rely on randomness to solve otherwise infeasible problems, and to model situations in almost every area of science. We would like to better understand the true computational power of randomness but the relationship between randomized and deterministic computation remains a mystery. However, several remarkable theorems showed us that randomness may not be as inherently powerful as we previously believed if we make some intuitive assumptions about hardness. The ability to remove randomness from computation without sacrificing efficiency is supported by natural assumptions about the hardness of problems. These results gave derandomizations of BPP, the class of languages decidable with randomized algorithms in polynomial time, into various deterministic computation classes based on various assumptions about both uniform and non-uniform hardness.

The most significant results to the effect of derandomizing BPP came from Impagliazzo and Wigderson. Based on a non-uniform hardness assumption, they showed that "P=BPP unless E has sub-exponential circuits" [4]. Based on a uniform hardness assumption, they showed that "if BPP $\neq$ EXP, then every problem in BPP can be solved deterministically in subexponential time on almost every input" [2]. Although these results seem to foreshadow a proof that BPP equals P or something close to P, such a result has been shown to have strong implications. In particular, Kabanet, Impagliazzo, and Wigderson related the derandomization of a class related to BPP to circuit

lower bounds for nondeterministic time classes [3]. This essentially suggests that the power of randomness is inversely proportional to the power of non-uniformity in relation to uniform computation. This paper will focus on the side of this trade-off that gets derandomization from hardness by looking at Impagliazzo and Wigderson's derandomization of BPP using circuit lower bounds.

Definitions

Most of these definitions are taken from [4].

- For a function $f : \{0,1\}^n \to \{0,1\}^\ell$, $S(f)$ denotes the worst-case circuit complexity of f: the minimum number of gates for a circuit that computes f correctly on every input.
- Let f be a Boolean function (a function with codomain $\{0,1\}$). The success $SUC_s(f)$ is defined as the maximum over all Boolean circuits C of size at most s of $Pr[C(x) = f(x)]$, where x is chosen uniformly at random from $\{0,1\}^n$. This corresponds to how easy f is to approximate for circuits of size at most s. If $SUC_s(f) = 1$ then $S(f) \geq s$. Note that $SUC_s(f) \geq 1/2$ for any s and f because a size 1 circuit could just always output a 1 or a 0 and thus match f at least half of the time.
- A *pseudorandom generator* (PRG) is, informally, a function that takes a small amount of random bits and outputs a large amount of bits that are indistinguishable from random for circuits of restricted size, or satisfy certain properties of looking random.

Derandomization from Hardness

In 1997, Impagliazzo and Wigderson established a strong relationship between the power of non-uniform computation and the power of randomness in the paper "P = BPP if E Requires Exponential Circuits: Derandomizing the XOR Lemma". The result of this paper is revealed by the title, but more formally:

Theorem 1 *If there is a Boolean function $f \in E = DTIME(2^{O(n)})$ where computing f on input size n requires circuits of size $2^{\Omega(n)}$, then BPP=P.*

This result can also be generalized so that stronger circuit-complexity assumptions implies weaker derandomization results [6, p. 260]. Reaching BPP = P from the existence of high-circuit-complexity problems is achieved in three broad steps:

1. If there is a function in E that requires circuits of size $2^{\Omega(n)}$, then there is a function in E that is mildly hard to approximate for any circuit of size $s(n)$ for some $s(n) = 2^{\Omega(n)}$.

2. If there is a function that is mildly hard to approximate, then we can amplify this unpredictability to get a function that is significantly hard to approximate. We will achieve this by applying Yao's XOR Lemma, which makes a function that requires a large input and then "derandomizing" the XOR Lemma. This means that we can generate a significantly unpredictable function from a mildly unpredictable function without blowing up the size of the input.

3. If there is a function that is hard to approximate for any circuit of size $s(n)$ for some $s(n) = 2^{\Omega(n)}$, then we can make a pseudo-random generator (PRG) whose output looks random to sufficiently small circuits. We can then use this PRG to deterministically simulate randomized algorithms with only a polynomial factor slowdown.

Hard to compute function → Mildly unpredictable function

Here, a Boolean function f being mildly unpredictable means that for some size $s(n) = 2^{\Omega(n)}$, any circuit of size $s(n)$ on inputs of size n will differ from f on some number of inputs. To construct a PRG, we need a function that is not just hard to compute in the worst case, but is also hard to compute on most inputs. As worst-case behavior is the focus of complexity theory and is often easier to reason about, we aren't satisfied basing our derandomization on an average-case assumption. Thankfully, we can construct unpredictability from worst case hardness. For the purposes of proving Theorem 1, Impagliazzo and Wigderson use the following hardness-to-unpredictability relation:

Theorem 2 *If there is a Boolean function f in E with $S(f) = 2^{\Omega(n)}$, then there is a function $h \in E$ such that the success, $SUC_{s(n)}(h_{cn}) \leq 1 - n^{-2}$ for some constant c and some $s(n) = 2^{\Omega(n)}$.*

Note that the size of the input (the subscript) can grow by a constant factor. There are several ways of reaching this kind of reduction that draw on seemingly distant fields. In [6, p. 240-250], Salil uses error correcting codes to construct a generalization of Theorem 2. Error correcting codes (ECC) are essentially encodings of messages with additional redundant information about the message such that some bits of the code can be randomly or adversarial tampered with and the original message can still be efficiently recovered. ECCs are vital for many types of data transmission such as internet routing and satellite communication. Anyways, the idea of the construction simple:

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$ with $S(f) > s$, view f as a message of size $L = 2^n$. Then define we apply error correcting code $ENC : \{0,1\}^L \to \Sigma^{\hat{L}}$ to f and interpret the result $ENC(f)$ as a function $\hat{f} : \{0,1\}^{\log \hat{L}} \to \Sigma$. If the ECC satisfies certain achievable properties, then $SUC_{s'}(\hat{f}) \leq 1 - \delta$ for some $s' = s^{\Omega(1)}$ and some constant δ that depends on the properties of the ECC [6, p. 241].

The argument is roughly as follows. Assume for the sake of contradiction that $SUC_{s'}(\hat{f}) < 1 - \delta$. So there is a circuit C of size s' that correctly computes $\hat{f}$ on a $> 1-\delta$ fraction of the inputs. The function C can be viewed as a message in $\Sigma^{\hat{L}}$. Particularly, C is a message that differs from $\hat{f}$ in a small $< \delta$ fraction of bits. This means that we can run the efficient decoding algorithm DEC on C to recover the original function f and compute f correctly on any input. Assuming our decoding DEC yields a size $\leq s$ circuit for computing f, this contradicts our assumption of the worst case hardness of f. This argument is flawed because decoding the entire function f on every input (as you would do assuming a normal ECC) would require exponential time. However, we can use a special type of ECC called *Locally Decodable Codes* to efficiently decode only the part of C that corresponds to the value of $f(x)$ for our input x.

This is a neat way to construct functions that are hard to approximate from functions that are hard to compute exactly. There are several other constructions, including one from Babai, Fortnow, Nisan, and Wigderson that uses *random self-reducablility* of functions in high complexity classes [1]. These reductions give us functions h such that $SUC_{s'}(h) \leq 1 - \delta$ for some small constant δ. However, for the purpose of constructing PRGs to fool s' sized circuits, we would like a function for which a size s' circuit can't do much better than randomly guessing. Salil gives a nice direct construction

of such a function from a worst case hard function using *list-decodable codes*, but more generally, there is a simple way we can amplify the unpredictability of any function at a cost of increasing the size of the input.

Mildly unpredictable function → Highly unpredictable function

Yao's XOR Lemma constructs a highly unpredictable function from a mildly unpredictable by simply taking the XOR of the function evaluated at multiple points. Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, let function $f^{\oplus(k)} : (\{0,1\}^n)^k \to \{0,1\}$ be given by

$$f^{\oplus(k)}(x_1, \cdots, x_k) = f(x_1) \oplus \cdots \oplus f(x_k)$$

Theorem 3 (Yao's XOR Lemma) *:*
If $SUC_s(f) \leq 1 - \delta$, then $SUC'_s(f^{\oplus(k)}) \leq 1/2 + \epsilon$ for any ϵ and any δ for some $k = O(-\log \epsilon/\delta)$ and $s' = s(\epsilon\delta)^{O(1)}$ [4].

Intuitively, if we have a circuit C that differs from f with probability δ on random inputs, then if we run C on k independent inputs and XOR the results together, we will get the correct value for $f^{\oplus(k)}$ if we were right every time (or wrong an even number of times), which happens with probability about $(1-\delta)^k$. Of course, we can have circuits that do something more sophisticated than simply running C k times so this intuition isn't completely applicable. Note that the XOR lemma amplifies hardness at the expense of increasing the input size by more than a constant factor. This is not yet a sufficient condition for our purposes.

There have been many proofs of Yao's XOR lemma that assume the inputs to $f^{\oplus(k)}$ are uniformly random (i.e. each input x_i is independently uniformly chosen). However, Impagilazzo and Wigderson cleverly constructed a proof that uses only two specific properties of the independence of the inputs x_i [4]. They then constructed a class of pseudorandom generators whose output maintains these specific properties, while of course losing true independence. Then, by composing $f^{\oplus(k)}$ with this PRG $G : \{0,1\}^m \to (\{0,1\}^n)^k$ where m is linear in n and k is as in the XOR Lemma, we get a function that has input size linear in n and is very unpredictable. In the next section, we see how to use this function to derandomize BPP.

Impagliazzo and Wigderson's proof of Yao's XOR Lemma is very involved, but it interestingly proves the contrapositive by using a circuit C that approximates $f^{\oplus(k)}$ with high success to construct a circuit C' that approximates f with high success using a strategy for a card betting game. The properties of the distribution of the inputs for $f^{\oplus(k)}$ that this proof requires are that it is (k', q, δ)-*hitting*, and M-*restrictable* for certain parameters k', q, δ, M. The *hitting* property essentially enforces that, for any k subsets $H_1, \cdots, H_k \subseteq \{0,1\}^n$ of size $\geq \delta 2^n$, then if we sample $x_1, \cdots, x_k$ from our distribution on $(\{0,1\}^n)^k$, then a good portion of these x_i's are expected to land in their corresponding H_i. The *restrictable* property of the generator is important for the correctness and feasibility of the implementation of the card game strategy on fixed size circuits. It is more complicated to define, but essentially allows us to efficiently sample $x_1, \cdots, x_k$, fixing a certain value $x_i = x$ for x and i of our choosing, and restricting the other values to some small subset of $\{0,1\}^n$. In the construction of C', we simulate C on inputs where one input x_i is the input to C' and the other inputs are coded into the circuit as "advice", but we want this input to have certain independence properties so that C estimate $f^{\oplus(k)}$ with high enough success.

Conveniently, we can get a generator with both the hitting and restrictable properties by simply taking the XOR of a generator with the hitting property and a generator with the restrictable property. To make generators with the hitting property, we use the common tool of walks on *expander graphs*. To achieve the restrictable property, we use another common generator: *nearly disjoint subsets*.

Now that we have an efficiently computable generator $G : \{0,1\}^m \to (\{0,1\}^n)^k$ with these nice properties, we can define a function $f' : \{0,1\}^m \to \{0,1\}$ as $f'(x) = f^{\oplus(k)}(G(x))$. Now if no circuit of size $s = 2^{\Omega(n)}$ can compute f on all but a small constant fraction of inputs, then for some $s' = 2^{\Omega(n)}$, no circuit of size s' can do better than guessing f' on more than a $2^{-\Omega(n)}$ fraction of inputs. This is exactly the condition required in the next section to simulate polynomial time randomized algorithms deterministically in polynomial time.

Unpredictable function $\to$ P = BPP

In the 1994 paper *Hardness vs Randomness*, Nisan and Wigderson showed that we can construct *pseudo-random generators* capable of efficiently sim-

ulating randomized algorithms from average-case hard functions [5]. This resulted in the average-case hardness vs randomness trade-off: if there is a function in E that has is hard to approximate some circuit size class, then we can derandomize BPP to some deterministic time class. Specifically, we get the following three results (converted to our notation) [5]:

Theorem 4 *The following trade-off:*

- *If there exists Boolean function $f \in E$ with $SUC_{s(n)}(f) \leq 1 - n^{-k}$ for any $s \in poly(n)$, then $BPP \subseteq SUBEXP = \bigcap_{\epsilon>0} DTIME(2^{n^\epsilon})$.*
- *If there exists $f \in E$ with $SUC_{s(n)}(f) \leq 1 - n^{-k}$ for $s = 2^{n^\epsilon}$ for some $\epsilon > 0$, then $BPP \subseteq DTIME(2^{(\log n)^c})$ for some c.*
- *If there exists $f \in E$ with $SUC_{s(n)}(f) < 1/2 + 2^{-\epsilon n}$ for $s(n) = 2^{\epsilon n}$ for some ϵ, then $BPP = P$. This last result, combined with the results from the previous sections gives us Theorem 1.*

We could imagine several important parameters for defining a PRG to fool circuits of a certain size, but for the purposes of derandomizing BPP, Nisan and Wigderson defined a pseudorandom generator $G : l \to n$ as a class of functions $G_n : \{0,1\}^{l(n)} \to \{0,1\}^n$ such that for any circuit C of size n:

$$\Big|\Pr\big[C(y) = 1\big] - \Pr\big[C(G(x))\big]\Big| < 1/n$$

Where x is uniform random in $\{0,1\}^{l(n)}$ and y is uniform random in $\{0,1\}^n$. In other words, for any circuit of size n, the probability that the circuit will output a 1 given a completely random input varies only slightly from the probability that it will output a 1 given a pseudorandom input.

We have been talking about circuit complexity this whole time because our assumption is based on circuit complexity, but we can finally relate this back to time complexity with the fact that any uniform algorithm (Turing Machine) running in time t can be simulated by a circuit of size t^2. We can then use a pseudorandom generator $G : l(n) \to n$ to convert a time t randomized algorithm R to a time $2^{O(l(t^2))}$ deterministic algorithm D as follows. First we convert R, which uses $O(t)$ random bits to an algorithm that uses $O(l(t^2))$ random bits by replacing R's random bits with the output of generator G. If this changed the behavior of R by a non-negligible amount, then we could use R to distinguish the output of G from truly random bits,

contradicting the fact that G is a pseudorandom generator. Then D can simply run this modified R on all possible $2^{O(l(t^2))}$ bit seeds and taking a majority vote on the outputs [5].

Now all that is left is to construct a pseudorandom generator from an unpredictable Boolean function $f : \{0,1\}^m \to \{0,1\}$. By definition, an unpredictable Boolean function can create a single pseudorandom bit, but the challenge is making lots of pseudorandom bits. Nisan and Wigderson achieve this using the *Nearly Disjoint Subsets* mentioned in the previous section. The Nearly Disjoint Subsets are a collection of n subsets $S_1, \cdots, S_n \subseteq \{1, \cdots, l\}$ each of size m such that any pair of sets has small intersection. Using random seed $x \in \{0,1\}^l$, we generate n pseudorandom bits, one for each set S_i: simply take the bits x_j such that $j \in S_i$ and pass that into our unpredictable function f. Finally, if $f \in E$ with $SUC_{s(n)}(f) < 1/2 + 2^{-\epsilon n}$ for $s(n) = 2^{\epsilon n}$ for some ϵ, then applying this construction gives us a pseudorandom generator $G : \emptyset(\log n) \to n$. We can then use this to simulate any polynomial time randomized algorithm deterministically in polynomial time. Combined with the results of the previous two sections, this gives us Theorem 1.

The first constructions of pseudorandom generators used one-way functions. Although we are willing to base the security of all our communications on the existence of one-way functions, this is still a very strong assumption and fortunately we do not need such a strong assumption for our purposes.

Conclusion

We now have a striking truth: if there are exponential time computable functions that require exponential sized circuits to compute, then BPP = P. More generally, the existence of exponential time computable functions that are harder to approximate weakens the power of randomness to speed up computation. This is striking because it can give us a new way to view randomness. Instead of insisting that randomness is truly random from any viewpoint, we often can label as *random* something that just looks random from a restricted viewpoint. Many things that we already consider truly random are in fact not purely random from a large enough view point. For example, the flip of a coin is considered random because it is hard for us to predict what the result will be from our observations, but with enough computational power and enough information about the position and strength of the flipper, we can determine the outcome with certainty.

This result is also striking in the more specific case because the existence of circuit size lower bounds is a very natural assumption. We are willing to base cryptography - the security of all our digital communications - on the existence of one-way functions, which is a much stronger assumption than what we would need to assume about circuit lower bounds to achieve BPP = P. Although it is already conjectured by complexity theorists, given how powerful randomness appears and how prevalent and indispensable randomized algorithms are in the modern world, BPP = P is a remarkable statement.

References

[1] Lźszló Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has Subexponential Time Simulations Unless EXPTIME has Publishable Proofs. *computational complexity*, 3(4):307–318, Dec 1993.

[2] R. Impagliazzo and A. Wigderson. Randomness vs. Time: De-Randomization Under a Uniform Assumption. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, pages 734–, Washington, DC, USA, 1998. IEEE Computer Society.

[3] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In Search of an Easy Witness: Exponential Time vs. Probabilistic Polynomial Time. *J. Comput. Syst. Sci.*, 65(4):672–694, December 2002.

[4] 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, pages 220–229, New York, NY, USA, 1997. ACM.

[5] Noam Nisan and Avi Wigderson. Hardness vs Randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, October 1994.

[6] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1â3):1–336, 2012.