COMPUTABILITY AND COMPLEXITY 24
RANDOMNESS

AMIR YEHUDAYOFF

1. The basics

Remark. *Computational complexity studies the resources that are needed to achieve computational tasks. On a high-level, computational devices have costs (like time, memory size, energy, randomness, training data, etc.), and computational tasks have complexities (the minimum cost that is needed to achieve it). We now move to focussing on randomness.*

Remark. *What is "randomness"? We shall use the language of mathematics. In mathematics, randomness corresponds to a probability space. We shall work only with finite spaces.*

Remark. *What is randomness good for? It can help to hide things. It can help in algorithm-design. It can help to avoid "worst-case" choices.*

Remark. *How do we generate randomness? We can toss coins; this requires some device or person. We can use some internal "noise" in computers. We can use physical phenomena, like radioactive decay. We are not going to discuss any of these "engineering" problems.*

Remark. *We shall introduce randomness into the Turing machine model as follows. A probabilistic Turing machine (PTM) M has three tapes: input tape, working tape, and randomness tape. There are two types of inputs to the machine. The usual x and a random string R. The string R will consist of i.i.d. uniform bits (but this could be chanced according to context).*

Definition 1. *For $T : \mathbb{N} \to \mathbb{N}$, a language $L \subseteq \{0,1\}^*$ is in $\mathsf{BPTIME}(T(n))$ if there is a PTM M so that for all $x \in \{0,1\}^n$,*

$$\Pr[M(x,R) = L(x)] \geq \frac{2}{3}$$

where

- *R is uniformly distributed in $\{0,1\}^{O(T(n))}$.*
- *$\mathsf{TIME}(M,x,R) \leq O(T(n))$ for all R.*

Remark. *As we shall see later on, the $\frac{2}{3}$ is not important.*

Definition 2. *The class of bounded-error probabilistic polynomial time language is*

$$\mathsf{BPP} = \bigcup_{k \in \mathbb{N}} \mathsf{BPTIME}(n^k).$$

Remark.

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

The question

is $\mathsf{P} = \mathsf{BPP}$*?*

is central and open. It asks whether we can always efficiently de-randomize computations.

2. An example: polynomial identity testing (PIT)

Remark. *If* $A, B \in \{0,1\}^n$*, how much time does it take to check if* $A = B$ *or not? Can randomness help to reduce the running time?*

Remark. *Polynomials are an example where randomness can help. This property is very important; for example, for error correction.*

Remark. *We work with* n *variables* $x_1, \ldots, x_n$ *over the field of rational numbers (it could also be other fields). A monomial is an expression of the form*

$$x_1^{e_1} x_2^{e_2} \ldots x_n^{e_n}$$

where e_i *is a non-negative integer and by convention* $x^0 = 1$*. The degree of this monomial is* $\sum_{i \in [n]} e_i$*. We shall denote this monomial by* x^e *where* $e = (e_1, \ldots, e_n)$*. A polynomial is a (finite) linear combination of monomials*

$$p(x) = \sum_e \alpha_e x^e$$

where $\alpha_e \in \mathbb{Q}$*. That number* α_e *is called the coefficient of* x^e *in* p*. The degree of* p *is the maximum degree of a monomial that appears in* p*.*

Example 3. *The degree of* $x_1 + x_1 x_2 - x_1 x_2^3$ *is four.*

> We would like to check if two polynomials p, q are equal or not. Over the rationals, there are two ways to think about this equality. First, as formal sums. Second, as functions $\mathbb{Q}^n \to \mathbb{Q}$. (Over other field, the two notions may not be the same.)

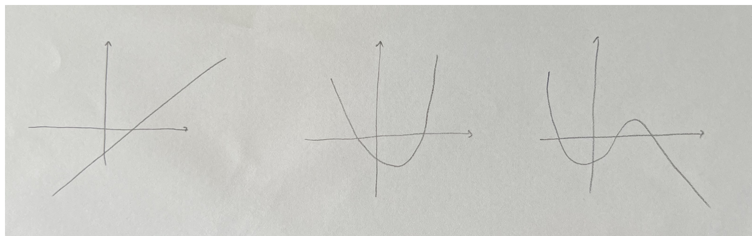
Remark. *Let's say we have black-box access to* p, q*. That is, we can choose* $a \in \mathbb{Q}^n$ *as we wish and ask "is* $p(a) = q(a)$*?"*

How many questions of this form we need to perform?

The answer depends on the degrees. If both degrees are at most d*, then naively we need something like* d^n *questions, which is a lot. Randomness allows to reduce the number of questions! This relies on the fact that polynomials have few roots.*

Lemma 4. *A univariate polynomial* p *of degree* d *has at most* d *roots.*

Remark. *We shall not prove (do you know how to prove?).*



Lemma 5 (DeMillo–Lipton–Schwartz–Zippel)**.** *If $p(x)$ is a non-zero polynomial in n variables of degree d and $S \subseteq \mathbb{Q}$ is non-empty, then*

$$\Pr[p(R) = 0] \leq \frac{d}{|S|}$$

where $R = (R_1, \ldots, R_n)$ is uniformly distributed in S^n.

Remark. *The lemma shows that with just one random question (from a large collection of questions) we can check if $p = q$ or not.*

Proof. The proof is by induction on n. The base case $n = 1$ holds because univariate polynomial of degree d has at most d roots. The induction step is performed as follows. Write

$$p(x) = \sum_{i=0}^{d} p_i(x_1, \ldots, x_{n-1}) x_n^i$$

where each p_i is in $n - 1$ variables of degree at most $d - i$. Because p is non-zero, there is some i so that p_i is non-zero. Let i_* be the maximum i so that p_i is non-zero. By induction,

$$\Pr[p_{i_*}(R') = 0] \leq \frac{d - i}{|S|}$$

where $R' = (R_1, \ldots, R_{n-1})$. By the $n = 1$ case,

$$\Pr[p(R) = 0 | R', p_{i_*}(R') \leq 0] \leq \frac{i_*}{|S|}.$$

Overall,

$$\begin{aligned} \Pr[p(R) = 0] &= \Pr[p(R) = 0, p_{i_*}(R') = 0] + \Pr[p(R) = 0, p_{i_*}(R') \neq 0] \\ &\leq \Pr[p_{i_*}(R) = 0] + \Pr[p(R) = 0 | p_{i_*}(R') \neq 0] \\ &\leq \frac{d - i_*}{|S|} + \frac{i_*}{|S|} = \frac{d}{|S|}. \end{aligned}$$

□

3. Representing polynomials

Remark. *Polynomials are extremely useful in many areas (mathematics, CS, science,...).*

Example 6. *Two central examples are over the $n \times n$ matrix of variables $X = (x_{i,j})$:*

$$\mathsf{det}_n(X) = \sum_{\pi \in S_n} \mathsf{sign}(\pi) \prod_{i=1}^{n} x_{i,\pi(i)}$$

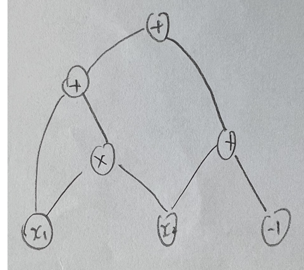
and

$$\mathsf{perm}_n(X) = \sum_{\pi \in S_n} \prod_{i=1}^{n} x_{i,\pi(i)}$$

where S_n is the (group of) permutations on $[n]$. The determinant is important in linear algebra and geometry (so in graphics, signal analysis, etc.). The permanent is important—it is the ultimate counting problem.

Remark. *We would like to represent polynomials efficiently. This leads to a new type of complexity theory: algebraic complexity.*

Definition 7. *An algebraic circuit is a DAG with in-degrees either zero or two. In-degree zero nodes are labelled by variables or field elements. In-degree two nodes are labelled by $+$ or $\times$. Circuits compute polynomials in the obvious way. The size of a circuit is the number of nodes in it.*



Definition 8. *The circuit complexity of p is the minimum size of a circuit computing p.*

Remark. *Again, devices have costs and tasks have complexities.*

Remark. *The circuit size of the det_n is at most $O(n^3)$ using Gaussian elimination. It is in fact at most $O(n^{2.5})$ which is much harder to see. The exact exponent is not known and the problem is important. (There is a long and interesting discussion here...)*

Remark. *The circuit complexity of perm_n is not known and it is one of the most important problems in theory of CS. But it is known that if there is an efficient way to compute perm_n then $\mathsf{P} = \mathsf{NP}$ and much more. (There is a long and interesting discussion here...)*

Remark. *Circuits have a different cost which captures the ability to perform the computation in parallel. The depth of the circuit is the length of the longest (directed) path in the graph.*

Remark. *Algebraic circuits have a very nice property: they can be balanced. If an algebraic circuit of size s computes a polynomial of degree d then there is circuit of size $\mathsf{poly}(n, s, d)$ and small depth $O(\log^2(sd))$. A similar result is not believed to hold for Boolean circuits.*

Example 9. *Here is a nice example of how PIT is related to other algorithmic problems, and how randomness could help. Let G be a bipartite graph where each color class is of size n. The problem we want to solve is*

"does G contain a perfect matching?"

There is a polynomial time algorithm for doing so. But it is not clear how to use randomness, or how can randomness be helpful for this problem. It turns out randomness is useful in allowing to solve the problem in "short parallel time with few processors".

Given an input graph G, define the $n \times n$ matrix of variables X:

$$X_{i,j} = \begin{cases} x_{i,j} & (i,j) \in E(G) \\ 0 & (i,j) \notin E(G) \end{cases}$$

Denote by $p(X)$ the determinant of X. The polynomial p has degree n in at most n^2 variables. A basic observation is that

$$G \text{ has a perfect matching (PM)} \iff \det(X) \neq 0.$$

If we set $S = [3n]$, and choose each $R_{i,j}$ i.i.d. uniform in S then

$$\Pr[p(R) = 0] \leq \frac{1}{3}.$$

In other words, the algorithm

if $p(R) = 0$ output "no PM" and if $p(R) \neq 0$ output "yes PM"

succeeds with probability at least $\frac{2}{3}$. As described above, $p(R)$ can be computed in $\mathsf{poly}(n)$-time and $O(\log^2(n))$-depth. This is an efficient algorithms that can be run in short parallel time.

4. Back to PIT

Remark. *Instead of working with polynomials as black-boxes, we wish to work with explicit representation of them as algebraic circuits.*

> Given two algebraic circuits C, C' over the integers $\mathbb{Z}$, we want to check if $C = C'$ in the sense that the polynomials they compute are the same.

Remark. *We think of the input length as the size s of the circuits.*

Claim 10. *If C has size s then its output has degree at most 2^s.*

Remark. *Using the previous approach (based on the DSZL lemma) would require computations with 2^s bits, which is too expensive. Again, randomness comes to the rescue. The idea is to perform the computation modulo some random large prime k.*

The algorithm

(1) Let S be the set of integers between 0 and $10 \cdot 2^s$.
(2) Choose $R = (R_1, \ldots, R_n)$ uniformly in S^n.
(3) Choose a prime $k \leq N := 2^{4s}$ uniformly at random.
(4) If $C(R) \mod k = C'(R) \mod k$, output "equal" and otherwise output "not equal".

Running time. The running time is now polynomial in s, because all computations are modulo k.

Equality case. If $C = C'$ the algorithm *always* outputs the correct answer.

Inequality case. If $C \neq C'$ then analyze the chance of failure as follows. Let

$$y = C(R) - C(R').$$

We know that

$$\Pr[y = 0] \leq \frac{2^s}{|S|} = \frac{1}{10}.$$

The integer y is at most $(10 \cdot 2^s)^{2^s}$ and so when we decompose it to a product of primes, there are at most $\log((10 \cdot 2^s)^{2^s}) \leq 2^{3s}$ prime factors. The number of primes at most N is at least

$$\Omega\Big(\frac{N}{\log N}\Big) = \Big(\frac{2^{4s}}{s}\Big).$$

(This was conjectured by Gauss and by Legendre, and proved by Hadamard and by de la Vallee Poussin). So,

$$\Pr[y \mod k = 0] \leq O\Big(\frac{2^{3s}}{2^{4s}/s}\Big) \leq \frac{1}{10}$$

for large s. Overall, by the union bound,

$$\Pr[\text{error}] \leq \frac{2}{10}.$$

□

Remark. *It is not trivial to sample a random prime efficiently, but it can be done.*

Remark. *We see how ideas from various areas of mathematics are deeply related to algorithm design.*

5. More classes

The PIT algorithm we say has "one sided" error. Namely, if $C = C'$ the output is always correct and if $C \neq C'$ the output is correct with probability at least $\frac{2}{3}$. This is a stronger guarantee than BPP requires. This type of guarantee is also natural and important, and there is a special term for it. It is called "randomized polynomial time" and denoted by RP.

Remark. *There are other natural classes for randomized computations but we shall not go over all of them in this course.*

6. Error reductions

Given a PTM M for a language L and input x, we can think of $M(x)$ as random variable that is typically equal to $L(x)$:

$$\Pr[M(x) = L(x)] \geq \frac{2}{3}.$$

How can we increase the chance of success?

Run M a few times.

If $y_1, \ldots, y_T$ is the outcomes of the T independent runs of M on x then $\mathsf{MAJ}(y_1, \ldots, y_T)$ is much more likely to be $L(x)$.

Theorem 11 (concentration of measure)**.** *If $Z_1, \ldots, Z_n$ are i.i.d. variables taking values in $[0,1]$ with expectation $\mu = \mathbb{E}Z_1$, then for all $t \geq 0$,*

$$\Pr\Big[\Big|\Big(\frac{1}{n}\sum_{i\in[n]} Z_i\Big) - \mu\Big| > t\Big] \leq 2e^{-2t^2 n}.$$

Remark. *There are many inequalities of this from (by Chernoff, Hoeffding, Azuma, Bernstein, and more). They are important in many areas.*

Remark. *This shows that in BPP the failure probability can be between $2^{-\mathsf{poly}(n)}$ and $\frac{1}{2} - \frac{1}{\mathsf{poly}(n)}$ and the meaning of BPP will stay the same.*

Remark. *There are several way to prove concentration bounds. Some use the "moment method" or "Fourier transform". Others are more "algorithmic".*

7. De-randomization conclusions

Remark. *De-randomization is a method for "efficiently simulating" a randomized algorithm by a deterministic one. There are many ingenuous ways to de-randomize algorithms, and there are still many basic related open problems. Here we shall see two examples.*

Theorem 12 (Adleman)**.** $\mathsf{BPP} \subseteq \mathsf{P/poly}$. *In other words, for every* $L \in \mathsf{BPP}$ *and* $n \in \mathbb{N}$*, there is a Boolean circuit computing*

$$\{0,1\}^n \ni x \mapsto L(x) \in \{0,1\}.$$

Sketch. The idea is that

(1) (as we said) we can assume that the error is smaller than 2^{-n}, and
(2) there are only 2^n inputs.

So, we can apply the union bound. (The details are left as an exercise.) □

Theorem 13 (Sipser-Gacs)**.**

$$\mathsf{BPP} \subseteq \Sigma_2.$$

Because BPP *is closed under complements, we can deduce*

$$\mathsf{BPP} \subseteq \Sigma_2 \cap \Pi_2.$$

Sketch. We have a PTM $P(x,R)$ that decides $x \in L$. The randomness R is in $\{0,1\}^m$ for $m = \mathsf{poly}(n)$. In fact, we can assume

$$x \in L \Rightarrow \Pr[M(x,R)=1] \geq \frac{1}{2}$$

and

$$x \notin L \Rightarrow \Pr[M(x,R)=1] < \frac{1}{2m+1}.$$

(See that you understand the latter condition when $x \notin L$.) We should construct a poly-time TM M so that for all inputs x,

$$x \in L \iff \exists a \forall b M(x,a,b) = 1.$$

What are a and b?

Fix $x \in \{0,1\}^n$. Consider the subset of strings that lead M to accept

$$S_x = \{R : M(x,R) = 1\}.$$

The size of S_x tells us if $x \in L$ or not. If $|S_x|$ is large, we should accept, and otherwise we should reject.

How can we do that?

Here is the main idea. For every $u \in \{0,1\}^m$, let

$$u + S_x = \{u + s : s \in S_x\}$$

where addition is in $\{0,1\}^m$ (entry-wise modulo two).

Claim 14. *If* $|S_x| \geq \frac{1}{2} \cdot 2^m$ *then there are* $t \leq m+1$ *vectors* $u_1, \ldots, u_t$ *so that*

$$\bigcup_{i \in [t]} (u_i + S_x) = \{0,1\}^m.$$

Remark. *If u is uniformly at random, then $u+S_x$ is a random subset of fractional size $\geq \frac{1}{2}$. It is not distributed uniformly at random, but each element is in it with probability at least a half.*

Proof idea. Taking $O(\log 2^m) = O(m)$ sets allows to cover the whole universe. With one set, we cover $\frac{1}{2}$. With two sets, we cover $1 - \frac{1}{4}$. With threes sets, we cover $1 - \frac{1}{8}$. And so forth, until we cover more than $1 - 2^m$ of the domain (at which point we are done). □

The witness a is now

$$a = (u_1, \ldots, u_t).$$

And b allows to check if indeed

$$\bigcup_{i \in [t]} (u_i + S_x) = \{0,1\}^m.$$

Namely, b is an element in $\{0,1\}^m$ and

$$M(x,a,b) = 1\big[\exists i \in [t]\ P(x, b + u_i) = 1\big].$$

We see that

$$x \in L \Rightarrow \exists a \forall b\ M(x,a,b) = 1.$$

What about the "no case"?

When $x \notin L$, as we already justified:

$$|S_x| < \frac{1}{t} 2^m$$

so that for all $u_1, \ldots, u_t$,

$$\Big| \bigcup_{i \in [t]} (u + S_x) \Big| < t \cdot |S_x| < 2^m;$$

here we used the property of M for $x \notin L$. Stated differently,

$$x \notin L \Rightarrow \forall a \exists b\ M(x,a,b) = 0.$$

□