Lecture 17
Randomized Computation

October 13, 2014

1

In the previous lectures we have studied about Turing Machines whose execution is deterministic. That is, given an input to the turing machine, one can state with complete accuracy what the output of the turing machine will be. Also, for the same input, the machine always outputs the same value. In this lecture we study about Probabilistic Turing Machines whose execution is not deterministic but random.

Before we get to details of Probabilistic Turing Machine, a few basic questions about randomness need to be understood.

- What is randomness?
- How do we use randomness for computation?
- Does it help?

At an abstract level randomness is described in terms of outcomes of events which have a certain probability distribution or a certain measure. However, in practice it is not very clear if we can generate truly random bits. The classical example of tossing a coin the outcome of which is often considered to be random is also deterministic in the sense that given the initial physical state of the coin and the air you are tossing it into, using the laws of physics that govern the motion of the coin, one can determine the outcome of the toss with complete accuracy. The random number generator functions used in program compilers too are not truly random since they use certain deterministic functions to output the bit. Hence it is unknown if generating a random number is practically feasible or not. However, such sources of random bits are observed to work when used in programs. Thus pseudorandom bits appear to be sufficient to be used in practice.

However, in the context of computation, we shall assume that we have access to a sequence of *pure* random bits. A bit is said to be random if it is 0 with a probability $\frac{1}{2}$ and 1 with a probability $\frac{1}{2}$.

Definition 1.1 (Probabilistic turing Machine)**.** A TM M is a Probabilistic Turing Machine if it is defined by a 4-tuple $< \Gamma, Q, \delta_0, \delta_1 >$ such that at any step of execution on an input x, M applies the transition δ_0 or δ_1 with a probability $\frac{1}{2}$ each and goes to the next configuration. In the end, M halts at either the accept or reject state.

The tuple representing the PTM are the set of alphabets, the set of states and two transition functions.

Definition 1.2 (Time complexity of a PTM)**.** We say that a PTM M runs in time $T(n)$ if M on input x halts within $T(|x|)$ steps irrespective of the random choices made during the execution.

Definition 1.3 (Class BPTIME)**.** A language $L \subset \{0,1\}^*$ is said to be in class BPTIME(T(n)) if there is a PTM M such that the running time of M is $\mathcal{O}(T(n))$ and

$$\Pr_{\text{all random choices in execution of M}}[M(x) = L(x)] \geq \frac{2}{3} \ \ \forall x \in \{0,1\}^*$$

In the name BPTIME, B stands for bounded error and P stands for probabilistic. The error here represents the probability that that the machine outputs a wrong answer. This error is bounded away from half. The value 2/3 in the definition can be replaced by any constant strictly greater than 1/2.

Definition 1.4 (Class BPP)**.** The class BPP or Bounded-error Probabilistic Polynomial time is defined as

$$\text{BPP} = \cup_{c>0}\text{BPTIME}(n^c)$$

The advantage with randomized algorithms is that they are simple and fast. In some cases, it is easier to prove the correctness of the randomized version of an algorithm than the deterministic algorithm. An example of this is the k^{th} order statistics problem.

Also randomized versions of algorithms are generally faster than their deterministic counterparts. For instance, the randomized quicksort runs in an expected time $\mathcal{O}(n \log n)$ while a deterministic version of it runs in time $\mathcal{O}(n^2)$.

While randomization makes algorithms faster, it must be asked as to how much faster does it make an algorithm. The question to be asked is if a language that cannot be decided deterministically in polynomial time be decided probabilistically in polynomial time. This requires one to compare the probabilistic classes with deterministic classes.

How does BPP compare with P?

Given a language $L \in$ P, there is a TM $M =< \Gamma, Q, \delta >$ that decides L in time polynomial in size of input. We now define the PTM $M' =< \Gamma, Q, \delta, \delta >$. This machine runs in time at most twice the time taken by M. The execution is identical to M. Every time it needs to make a random choice it picks a random number and ignores it. M', on every input, outputs if it is in the language or not correctly always. So the probability over the random choices that the machine decides an input correctly is 1. Hence M' is a PTM that decides L in polynomial time. Hence the language L is in BPP.

Thus P$\subset$BPP.

The question if BPP$\subset$P is not answered yet. It has been conjectured that BPP=P. This would imply that randomization gives only a polynomial factor advantage in running time of a TM and that every randomized algorithm can be derandomized efficiently. It also implies that there is no randomized polynomial time algorithm for NP-complete problems unless P=NP. A result supporting the conjecture is the following:

Theorem 1.5. Let E=DTIME($2^{\mathcal{O}(n)}$). If there's a language (equivalently a boolean function) in E that has circuit complexity $2^{\Omega(n)}$ then BPP=P.

The theorem appears in the paper 'P=BPP unless E has sub-exponential circuits: Derandomizing the XOR lemma' by Impagliazzo and Wigderson in 1997.

There are problems in BPP that are not known to be in P:

- Polynomial identity testing: Given a polynomial as an arithmetic circuit the algorithm checks if it is the zero polynomial or not.
- Univariate polynomial factoring over finite fields: Given a polynomial over a finite field, the algorithm needs to factorize the polynomial into irreducibles. It is theoretically known to be factorisable into irreducibles and the factorisation is unique.

Polynomial Identity Testing

Arithmetic circuit: An arithmetic circuit is like a boolean circuit in which the inputs are variables or constants, the OR gates are replaced by addition gates and the AND gates are replaced by multiplication gates. The edges between the gates can have constant scalars from the underlying field and they get multiplied to the value at the gate from which the edge starts. Every addition gate takes in two polynomials and outputs the addition of the polynomials. Similarly, every multiplication gate accepts two polynomials and outputs their product. Thus an arithmetic circuit computes a multivariate polynomial over the underlying field.

Now we consider the problem of polynomial identity testing:
Input: An arithmetic circuit
Output: 1 if the polynomial the circuit represents is identically zero and 0 otherwise

A polynomial is said to be identically zero if it is the zero polynomial.For example, $x_1x_2 - x_2x_1$ is identically zero.

Claim 1. PIT $\in$ BPP

Lemma 1.6 (Schwartz-Zippel lemma)**.** Let $g(x_1, x_2, \ldots, x_n)$ be any non-zero polynomial over field $\mathbb{F}$. Let $S \subset \mathbb{F}$ be a finite set. Let $\bar{a} = (a_1, a_2, \ldots, a_n) \in \mathbb{F}^n$ be a random point where every a_i is chosen uniformly at random from the set S. Let the total degree of the polynomial g be d. Then

$$\Pr_{\bar{a}}[\ g(\bar{a}) = 0\ ] \leq \frac{d}{|S|}$$

The degree of a multivariate term $x_1^{m_1}x_2^{m_2}\ldots x_n^{m_n} = \sum_{i=1}^{n} m_i$. If the multivariate polynomial $g = \sum_{j=1}^{k} x_1^{m_{j1}}x_2^{m_{j2}}\ldots x_n^{m_{jn}}$, then the total degree of $g = \max_j(\sum_{i=1}^{n} m_{ji})$.

Proof. The lemma can also be stated as, for a multivariate polynomial of degree d and any subset S, the number of zeros of g in S^n, is at most $d|S|^{n-1}$.

We prove this by induction on the number of variables in the polynomial, n.

If n=1, then g is a univariate polynomial. By the fundamental theorem of algebra, a univariate polynomial of degree d has at most d zeros. Thus the number of zeros in any set S is at most d. Hence the lemma holds true when n=1.

Now assume the statement for polynomials with at most $n-1$ variables.

Let $g = g_0(x_1, \ldots, x_{n-1}) + x_n g_1(x_1, \ldots x_{n-1}) + \ldots + x_n^t g_t(x_1, \ldots, x_{n-1})$ where $t \leq d$ and $g_t(x_1, \ldots, x_{n-1})$ is a non-zero polynomial.

Let $\bar{a} = (a_1, \ldots, a_n)$ be a point chosen uniformly at random from S^n.

Case 1: $g_t(a_1, \ldots, a_{n-1}) = 0$. g_t is a polynomial of degree at most $d - t$. By the induction hypothesis we have that this is true for at most $(d-t)|S|^{n-2}$ points in S^{n-1}. We can choose any value for x_{n-1} in $|S|$ ways. Thus the number of points where in S^n where $g = 0$ and $g_t = 0$ is at most $(d-t)|S|^{n-1}$.

Case 2: $g_t(a_1, \ldots, a_{n-1}) \neq 0$. This leaves us with a polynomial in x_n of degree at most t. By the induction hypothesis, the number of zeros of this polynomial is less than t. Hence the number of points where $g = 0$ and $g_t \neq 0$ is at most $t|S|^{n-1}$.

Now number of points at which $g = 0$ is at most $(d-t)|S|^{n-1} + t|S|^{n-1} = d|S|^{n-1}$. This proves the induction hypothesis.

Thus for any polynomial g and any subset S, the probability that for a point a, chosen randomly from S, $g(a) = 0$, $\leq \frac{d|S|^{n-1}}{|S|^n} = \frac{d}{|S|}$

□

Proof. We now prove that PIT $\in$ BPP.

Consider a PTM M whose working is as follows: Given a polynomial g, it chooses a subset S, and picks a point $\bar{a}$ uniformly at random. Now it evaluates the value of the polynomial at $\bar{a}$. If it is 0, it outputs 1 and if the polynomial evaluates to a non-zero value, it outputs 0.

Analyzing correctness of the algorithm:

Case 1: If g is identically zero: In this case, irrespective of the point we choose g evaluates to 0. Thus the algorithm outputs the correct value always.

Case 2: g is not identically zero. If the polynomial does not evaluate to 0 at $\bar{a}$ then the algorithm outputs correctly. If the polynomial evaluates to 0 at $\bar{a}$, then the algorithm outputs the wrong answer. The probability that at $\bar{a}$, g evaluates to 0 is $\leq \frac{d}{|S|}$ by the Schwartz-Zippel lemma.

$\Pr_{\bar{a}}[M(g) = 0 \ \& \ g = 0] = 1$ and $\Pr_{\bar{a}}[M(g) = 1 \ \& \ g \neq 0] \geq (1 - \frac{d}{|S|})$

Now if we choose S so that $|S| \geq 10d$, we get

$$\Pr_{\bar{a}}[M(g) \textit{ outputs correctly}] \geq \frac{9}{10}$$

Now, we need to ensure that M is of size polynomial in the input size. Let the input size, the size of the arithmetic circuit be n. The depth is also at most n.

The degree of the polynomial can be exponential in the depth of the circuit. Thus the degree of the polynomial is $\mathcal{O}(2^n)$. While numerically computing the value of the polynomial, a term of the form $x_k^{2^n}$ will need $n2^n$ bits to store its value which is exponential in the size of the input and hence not possible. The value of the polynomial can be as large as 2^{n2^n}.

We use the following properties of modular arithmetic to simplify this problem:
$a + b \pmod k = a \pmod k + b \pmod k$
$ab \pmod k = a \pmod k \times b \pmod k$

Thus we can compute the value of the polynomial modulo k by computing the value modulo k at each of the intermediate gates.

Now, we choose uniformly at random $10n^3$ numbers independently from $\{1, 2, 3, \ldots, 10.2^{n^2}\}$. Each of these numbers have a size at most $\mathcal{O}(n^2)$. We compute the value of the polynomial modulo each of these numbers. The number of primes in the set is $\frac{10.2^{n^2}}{n^2+\log 10}$. The number of distinct prime factors of $2^{n2^n} < \log 2^{n2^n} = n2^n$. Thus the number of primes that do not divide the value of the polynomial but are in the set are $\frac{10.2^{n^2}}{n^2+\log 10} - n2^n > \frac{8.2^{n^2}}{n^2}$. Since none of these $\frac{8.2^{n^2}}{n^2}$ numbers divide the value of the polynomial at the point, the value of the polynomial modulo any of these numbers will be non-zero. Let them be called *good* numbers. If a *good* number is chosen in at least one of the $10n^3$ trials, the output of the machine will be correct.

The probability of choosing one of the *good* numbers is $\frac{8.2^{n^2}/n^2}{10.2^{n^2}} = \frac{4}{5n^2}$. The probability of choosing a number that is not *good* is $1 - \frac{4}{5n^2}$. The probability a bad number is chosen in all the $10n^3$ trials $= (1 - \frac{4}{5n^2})^{10.n^3} = ((1 - \frac{4}{5n^2})^{-5.n^2/4})^{-8n}$. Asymptotically, this probability is $\leq e^{-8n}$.

Thus if the polynomial represented by the arithmetic circuit is non-zero, then the $\Pr[M(g) = 0] = \Pr[$ One of the good numbers is chosen and the polynomial evaluates to a non-zero value at the point $\bar{a}$ chosen]. The set of *good* numbers is determined once the value of the polynomial is evaluated. Hence the two events are independent. Hence $\Pr[M(g) = 0] \geq (1 - e^{-8n})9/10 \geq \frac{4}{5}$.

Thus the working of M can be described as:

- It accepts an arithmetic circuit that represents a polynomial g, with $|g| = n$.
- The set $S = \{1, 2, \ldots, 10n\}$. A point $\bar{a}$ is chosen uniformly at random from S^k, where k is the number of variables.
- It chooses $10n^3$ numbers from $\{1, 2, 3, \ldots, 2^{n^2}\}$ independently and uniformly at random.
- The value of the polynomial at $\bar{a}$ modulo each of these numbers is computed.
- If any of the values is non-zero then the output is 0, else it outputs 1.

$\Pr[M(g) = 1] = 1$ if g is identically zero and $\Pr[M(g) = 0] \geq \frac{4}{5}$ if g is not identically zero. Thus $\Pr[M\text{decides correctly}] \geq \frac{4}{5}$.

An application of PIT: Perfect matching in bipartite graphs

A bipartite graph is a graph $G = (V_1, V_2, E)$ such that there is no edge between two vertices in V_1 or two vertices in V_2. A perfect matching of a graph is a choice of edges of the graph such that every vertex is the end point of exactly one edge in the collection. For a bipartite graph to have a perfect matching, we require $|V_1| = |V_2|$.

Given a graph G, we need to determine if it has a perfect matching or not.

This problem is known to be in P. The *augmenting path algorithm* solves the perfect matching in polynomial time. However the problem is not known to be in NC. It is however known to be in randomized-NC.

The bi-adjacency matrix B of a bipartite graph is an $n \times n$ matrix whose rows are labeled by the vertices in V_1 and columns by the vertices in V_2. The matrix is defined as:

$$B_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}$$

A perfect matching of the graph is a collection of n edges with no two edges having the same vertex. Thus for every vertex in V_1, we can map it to the vertex in V_2 that it shares an edge in the matching with. This map is bijective. Thus a perfect matching is basically a permutation of $[n]$. If the permutation is σ, then the edges in the perfect matching are $M = \{(i, \sigma(i)) | 1 \leq i \leq n\}$. Any set $M' = \{(i, \sigma(i)) | 1 \leq i \leq n \ \& \ (i, \sigma(i)) \in E\}$ is also a valid perfect matching.

The determinant of the matrix B is

$$\det B = \sum_{\sigma \in S_n} \text{sign}(\sigma) \Pi_{i=1}^{n} B_{i,\sigma(i)}$$

Thus for every perfect matching of the graph there is a term in the summation. If the graph has no perfect matching, then none of the terms in the summation survive and hence the determinant of the matrix is 0. However, if there are two or more perfect matchings of the graph, the permutation associated with them can have opposite signs due to which the corresponding terms may cancel each other and the determinant can still have a value of 0. If we are assured that the number of perfect matchings of the graph is either odd or zero, then the determinant will evaluate to 0 iff the graph has no perfect matchings. For the general case, to determine if the graph has a perfect matching we need to prevent cancellation of the terms. Hence we define the symbolic bi-adjacency matrix as follows:

$$B_{ij} = \begin{cases} x_{ij} & (i,j) \in E \text{ where } x_{ij} \text{ is a formal variable} \\ 0 & (i,j) \notin E \end{cases}$$

The determinant of this matrix is a multivariate polynomial of degree exactly n. If this polynomial is identically zero, then every term in the summation must be zero and hence there exists no perfect matching. However, if the polynomial is not identically zero, then at least one of the terms in the summation must be non-zero and that corresponds to a perfect matching of the graph. Thus the problem of determining if the graph has a perfect matching or not has been reduced to determining if a given polynomial is identically zero or not. We can use the polynomial identity testing to solve this problem now.

The degree of the determinant polynomial is n and the number of variables is the number of edges. We select the set $S = 1, 2, \ldots, 3n$. We randomly choose a point from $S^{|E|}$ and assign it to the variables This can be done all at once in one time step. The value of the resultant polynomial at the specific point is equal to the value of the determinant computed after assigning these values to the variables in the matrix. Computation of a numeric determinant has an NC algorithm. If the determinant is non-zero, output that there is a perfect matching and if it evaluates to 0, output that it has no perfect matching.

If the determinant is zero, then according to Schwartz-Zippel lemma, with a probability more than 2/3 the given graph has no perfect matching. Thus the above algorithm is a randomized algorithm that outputs the correct answer with a probability greater than 2/3. Thus the problem of determining if the graph has a perfect matching is in randomized NC.

□

Notes scribed by:
Sabareesh R
ugsabareesh@ug.iisc.in