CMPT 407: Computational Complexity **Summer 2017**

Lecture 6: Randomized Computation (12 - 16 June)

Lecturer: Valentine Kabanets *Scribe: Lily Li*

The complexity classes we are going to consider in the next little while are these:

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

where BPP is the class of randomized polynomial time algorithms (assuming access to uniform random bits) and BQP is quantum polynomial time (assuming Quantum Mechanics is accurate and we can build large quantum systems).

Consider an example in Communication Complexity. Consider two parties *Alice* and *Bob* with strings a and b respectively. They both want to know if $a = b$. Let the cost of a protocol be the number of bits sent between the two parties. Consider the following deterministic protocol: *Alice* sends *Bob* string a. *Bob* compares a and b and sends *Alice* one bit denoting if they are equal or not. The complexity of the protocol is $n + 1$ for strings of length n. As it turns out, the best we can do with a deterministic protocol is n.

What if we use a randomized protocol? *Alice* picks a random prime number p in the range (n^2, n^3) and a random number r in the range $(1, p)$. For $a = a_1 \cdots a_n$, *Alice* calculates $A(r) = a_1 r^{n-1} + \cdot + a_n \mod p$ then sends *Bob* the string $(q, r, A(r))$ (actually p is not necessary but you might as well send it anyways). For $b = b_1 \cdots b_n$, *Bob* computes $B(r) = b_1 r^{n-1} + \cdots + b_n \mod p$ and compares $A(r)$ and $B(r)$ returning *Alice* 1 if $A(r) = B(r)$ and 0 otherwise. The cost of this protocol approximately $9 \log n$.

To see how good this protocol is let us calculate the error rate. If $a = b$ then $A(r) = B(r)$. Suppose $a \neq b$ i.e. $A(x) \neq B(x)$ as polynomials in the finite field $\mathbb{F}_p$ then what is the likelihood that $A(r) = B(r)$? Well $A(r) = B(r)$ if $a_1 r^{n-1} + \cdot + a_n \equiv b_1 r^{n-1} + \cdots + b_n \mod p$ or $C(r) = (a_1 - b_1) r^{n-1} + \cdot + (a_n - b_n) = 0$. This reduces to finding the number of roots of the polynomial $C(x)$. The maximum degree of C is $n - 1$ and r can take on at least n^2 different values so $\mathbf{Pr}[C(r) = 0] \leq \frac{n-1}{n^2} \leq \frac{1}{n}$ which is pretty good for large n.

6.1 Randomized Complexity Class

A more granular subdivision of Randomized Complexity classes is as follows:

$$\mathsf{ZPP} \subseteq \mathsf{RP} \subseteq \mathsf{BPP}$$

Definition 6.1 *A language $L \in \mathsf{RP}$ (one-sided error) if there is a polynomial time DTM $M(x, r)$ such that $\forall x \in L : \mathbf{Pr}_r[M(x, r)$ accepts$] \geq 1/2$ and $\forall x \notin L : \mathbf{Pr}_r[M(x, r)$ accepts$] = 0$. If $|x| = n$ then we have $|r| \leq n^c$ for some constant c. To decide if $x \in L$ the randomized algorithm R flips r fair coins and decides accept or reject in polynomial time. If R accepts then $x \in L$. If R rejects then R might be too cautious and be wrong a proportion of the time. Think of R as allowing false negatives.*

Similarly for two-sized error class, language $L \in \mathsf{BPP}$ if $\forall x \in L : \mathbf{Pr}_r[M(x, r)$ accepts$] \geq 3/4$ and $\forall x \notin L : \mathbf{Pr}_r[M(x, r)$ accepts$] = 1/4$.

Finally for the zero-error class, a language $L \in \mathsf{ZPP}$ if $M(x, r)$ produces an answer, then it is correct and $\forall x, \mathbf{Pr}[M(x, r) =?] \leq 1/3$ where ? is the output "I don't know".

Theorem 6.2 *If $L \in \mathsf{RP}$ then we can reduce the error to $\frac{1}{2^n}$ for any $n \in \mathbb{N}$.*

Proof: Quite straight forward. Just run the RTM M on input x a bunch of times. If any trial accepts then $x \in L$ since M cannot be wrong on rejecting inputs. ■

Theorem 6.3 *If $L \in \mathsf{BPP}$ (recall $\frac{1}{4}$ chance of error for both accepting and rejecting) then we can reduce the error to $\frac{1}{2^n}$ for any $n \in \mathbb{N}$.*

Now lets consider ZPP in-depth:

Theorem 6.4 $L \in \mathsf{ZPP} \iff L \in \mathsf{RP} \cap \mathsf{coRP} \iff$ *there exists a randomized algorithm that is always correct and has expected polynomial running time. Let the random variable $T(x, r)$ be the running time on input x with randomness r. The expected running time on x is $Exp_r[T(x, r)] = \sum_r T(x, r) \cdot Pr(r)$.*

Proof: To show that the first $\iff$ holds we will show that $L \in \mathsf{RP} \cap \bar{L} \in \mathsf{RP} \implies L \in \mathsf{ZPP}$. Let M_1 be an algorithm for L and M_2 be an algorithm for $\bar{L}$. We chose a random r and run both $M_1(x, r)$ and $M_2(x, r)$. If either accepts then we know if $x \in L$ for sure. In the $1/4$ of the time when both M_1 and M_2 reject, we say *"I don't know"*.

Next we have $L \in \mathsf{ZPP}$ and need to show that $L \in \mathsf{RP} \cap \bar{L} \in \mathsf{ZPP}$. Suppose that we have an algorithm M for $L \in \mathsf{ZPP}$. Again we choose a random r and run $M(x, r)$. If M accepts or rejects then we know if $x \in L$ or not. Otherwise we reject. It is clear that we will not be wrong in the cases when $x \notin L$.

For the next $\iff$ we show that $L \in \mathsf{ZPP} \iff$ there exists and algorithm which runs in an expected polynomial time. In the backwards direction, let M be the algorithm. To show that $L \in \mathsf{ZPP}$ we need an algorithm that runs in polynomial time. We can do so by timing M and stopping when the time is up. If M did not provide an answer in the alloted time, answer *"I don't know"*.

In the forward direction let M be an algorithm for L in ZPP. This time we have the opposite problem. Let us assume that $L \in \mathsf{EXP}$. We need the expected time to be polynomial, but on occasion we are allowed exponential running time. We achieve this as follows: let M' be a exponential TM for L. Run M' on input x and simultaneously generate random strings r to run M. If M accepts or rejects, do the same. This will happen most of the time. in the rare occasion that M fails to find an answer in exponential time, M' will provide the answer. ■

It is believed that $\mathsf{BPP} = \mathsf{P}$ so $\mathsf{BPP} \subseteq \mathsf{NP}$ but no

Theorem 6.5 *There exists a zero-error ZPP randomized algorithm for $k - \mathsf{SAT}$ in expected time $2^{n(1-\frac{\epsilon}{k})}$ with constant $\epsilon > 0$. It turns out that this algorithm allows for counting the number of satisfying assignments as well!*

Proof: Do you still remember the switching lemma? It will come in handy. The actual statement of the lemma is a bit stronger than was stated originally stated. The original was as follows: given a particular restriction ρ and DNF D, there exists a CNF logically equivalent to D with restriction ρ with high probability. The actual statement is:

Lemma 6.6 *$\forall k, t, p$ and $\forall k$-CNF ϕ on n variables:*

$$\mathbf{Pr}_{\rho \sim R_p}[\textit{depth of } C\Delta T(\phi|_\rho) > t] < (5pk)^t$$

Notice that instead of any old CNF, you get inparticular the canonical decision tree for our DNF is very likely to have low depth. Observe that there is an equivalence between devision trees of depth d and d-DNF. Given a depth d decision tree D

Next we need to define the notion of a **canonical decision tree**. Let ϕ be some k-CNF over variables $x_1, ..., x_n$. We assign an order to the variables, say $x_1 < x_2 < \cdots < x_n$, and an order for each clause, say lexicographical.

Finally, we will build the $k - \mathsf{SAT}$ *algorithm.* Let the k-CNF be over the variables $x_1, ..., x_n$.

1. For each $i \in \{1, ..., n\}$ decide with probability p, to be decided later, to put i in a set S
2. Iterate over all assignments a to the variables with indices not in S. We expect np items in S so there should be $n - np$ variables to assign. Check if $\phi|_a$ is satisfiable by computing $C\Delta T(\phi|_a)$. If *phi* is satisfiable then output *yes*, otherwise output *no*.

   Let us analysis this randomized algorithm. First note that the

■

The following is an application of our discussion thus far:

Theorem 6.7 $\mathsf{BPP} \subseteq \mathsf{PolySize}$

Proof: Suppose $L \in \mathsf{BPP}$. Then on input x, the probability that $M(x, r)$ is wrong can be made quite small: $\leq 2^{-2n}$. Consider the grid with all n bit string inputs $x_1, ..., x_{2^n}$ down the rows and all the random strings $r_1, ..., r_{2^{PolySize}}$ across the columns. Each cell (i, j) of the table has *accept* or *reject* depending on the value of $M(x_i, r_j)$. For each row the expected number of wrong results to be 2^{-2n}. Using union bound, expected number of random inputs r_j for which $M(x_i, r_j)$ is wrong for some x_i is $2^n/2^{2n} = 1/2^n$. Which is quite small. Thus there must exist some random string r^* which allows M to correctly evaluate every input string. By hard-coding r^* into the $\mathsf{PolySize}$ circuit which represents $M(x, r)$, we obtains a $\mathsf{PolySize}$ circuit for L which always outputs the right answer. ■

Note: this is an existential proof. We know that such a circuit must exists, but, in-practice, we have no way of finding r^* (we would need a bit chunk of the table to do so) and constructing such a circuit.

(It is trivially the case that $\mathsf{RP} \subseteq \mathsf{NP}$. Why?) It is not know if $\mathsf{BPP} \subseteq NP$. Further it is not know (and not believed) if $\mathsf{NP} \subseteq \mathsf{BPP}$. And it believed, but now shown that randomized complexity gives no extra power i.e. $\mathsf{BPP} = \mathsf{P}$.

Theorem 6.8 $\mathsf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$. *(*$\mathsf{BPP}$ *is in "generalized"* NP*)*

Proof: First note that BPP is quite similar to P in the sense that BPP is equal to its complement. Thus it is sufficient to prove that $\mathsf{BPP} \subseteq \Sigma_2^p$ since its complement is $\mathsf{coBPP} \subseteq \Pi_2^p$.

$\forall L \in \mathsf{BPP}$, there exists a probabilistic TM M which decides L with error probability $\leq 1/2^n$. Let n be the input length of x and m be the length of the random string. We want to know if $x \in L$ and given that that $m = \mathsf{Poly}(n)$. Fix input x, and define $A = \{r \in 0, 1 : M(x, r) \text{ accepts }\}$. Observe the following cases:

1. $x \in L$: let $R = \{0, 1\}^m$ be the set of all random strings. In this case, $|A|/|R|$ is very close to 1 since the number of bad random strings is only $1/2^n$.
2. $x \notin L$: then similar to the above, but now $|A|/|R|$ is close to 0 since most of the random strings cause M to reject.

Consider a translation of A, namely given a random string $z \in \{0, 1\}^m$ let $A \oplus z = \{a \oplus z : \text{ for } a \in A\}$. We will use these shifts to cover the domain of all random strings.

Lemma 6.9 *If $|A|/|R| \geq 1 - 2^{-n}$, then there exist strings $s_1, ..., s_k$, for $k = |r|$, such that $R = \cup_{i=1}^{k}(A \oplus s_i)$.*

Proof: Use the probabilistic method. Again we will consider the two cases. First we suppose $x \in L$. Let $S = \cup_{i=1}^{k}(A \oplus s_i)$ What is the probability that $\mathbf{Pr}[r \notin S]$? Since the random strings $s_1, ..., s_k$ are chosen independently and we can apply union bound we have:

$$= \prod_{i=1}^{k} \mathbf{Pr}[r \oplus s_i \notin A] < (2^{-n})^k = 2^{-nk}$$
$$= \mathbf{Pr}[\exists r \in R \text{ such that } r \notin S] \leq |R|2^{-nk} = 2^{-n} << 1$$

Thus a randomly chosen set of strings $s_1, ..., s_k$ causes R to cover $\{0, 1\}$ with high probability.

Next we consider what happens when $x \notin L$. Then $|A|/|R| < 2^{-n}$ and it is possible to show that now set of k random strings $s_1, ..., s_k$, when shifting A, will cover the entire space of random strings. Since at most $|A| * k = k/2^n < 1$ percent of the random strings are covered by the shifted copies of A, some $r \in R$ will not be covered. ■

■

6.2 Randomness and Nondeterminism

Recall one definition of NP. It is the set of all languages such for which there exists a deterministic polynomial time verifier which checks the "proof" of an all powerful prover. To inject some randomness to the party, now the verifier is allowed to be randomized polynomial time. More formally, and we will be using some of the strange naming conventions, we will define the classes as follows:

Let the prover be Merlin and the verifier be Arthur. A language $L \in MA$ if there is a constant c, and a polytime relation $R(x, y, z)$ such that, for every x of length n,

$$x \in L \implies \exists y \mathbf{Pr}_z[R(x, y, z) = 1] \geq \frac{3}{4}$$
$$x \notin L \implies \forall y \mathbf{Pr}_z[R(x, y, z) = 1] \geq \frac{3}{4}$$

Similarly, a language $L \in AM$ if The difference here is that the verifier moves first showing the prover the chosen random string. This type of problem is know as Public coin Protocol.

Similarly there exists a Private-coin Protocol. This protocol makes use of the graph non-isomorphism problem.