Randomized Algorithms 2020-1
Lecture 2

2-SAT, 3-SAT and Complexity Classes *

Moni Naor

Our emphasis would be worst case data where the randomness is created independently of it. That is we assume the algorithm or computing device in addition to the inputs gets also a random 'tape' (like the other tapes of the Turing Machine, but this one with truly random symbols).

One nice feature that some randomized algorithms have is that they are simple. We demonstrated this in two algorithms (actually got only to see the min-cut algorithm).

Randomized algorithms existed for a long time, since the dawn of computing (for instance the numerical "Monte Carlo Method"[1] from the 1940's or Shannon's work [5], also from that time.

Later, when people started arguing rigorously about the running time of programs, the idea of complexity classes of probabilistic machines emerged. We mentioned three such classes: RP, $Co-RP$ and BPP.

Recall that $L \in RP$ if there is an algorithm (Probabilistic Turing Machine M) s.t. for $x \in L$ we have that $Prob[M(x)$ outputs 'yes'$] \geq 1/2$ and for $x \notin L$ we have $Prob[M(x)$ outputs 'no'$] = 1$. We say that $L \in BPP$ if there is a Probabilistic Turing Machine M s.t. for $x \in L$ we have that $Prob[M(x)$ outputs 'yes'$] \geq 2/3$ and for $x \notin L$ we have $Prob[M(x)$ outputs 'no'$] \geq 2/3$.

One important thing we did not mention explicitly is that the constants $2/3$ and $1/2$ in the definitions of RP and BPP are arbitrary. for RP any constant $\delta > 0$ would define the same class. For BPP any constant $1/2 + \gamma$ for $\gamma > 0$ would yield the same class. This is an important example of amplification.

Question: Show that this is indeed the case. Can you replace the constants δ and γ with a function of n? Which one? For the BPP case you will probably need Chernoff Bounds. Read about them and find the relevant ones.

The common conjecture is that $P = BPP$ and it is known to hold under the assumption that there are sufficiently good pseudo-random generators, that is a function $G : \{0,1\}^* \mapsto \{0,1\}^*$ that stretches its input significantly and the output cannot be distinguished from random. The weakest assumption under which the latter exist is that E (the class of deterministic $2^{O(n)}$) has a problem

*These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. In the interest of brevity, most references and credits were omitted.

[1] Do not confuse with the term "Monte Carlo Algorithm" which is a general name for an algorithm whose running time is deterministic (usually polynomial) but may err.

with *circuit complexity* $2^{\Omega(n)}$ [1].

Question: Hitting set for RP. For any language $L \in RP$ a hitting set for input size n is a collection $C_n = \{R_1, R_2, \ldots R_m\}$ where for every $x \in L$ s.t. $|x| = n$ there is an $R_i \in C_n$ such when the input x is executed with random tape R_i the result is correct. Show that for any language $L \in RP$ there is a hitting set of size $m = n$.

One of the proponents of the power of randomized algorithms in the 1970s was Michael Rabin who published a very influential paper [3] on randomized algorithms. In that paper he gave a randomized algorithm for testing the primality of a number (based on Miller's deterministic test which assumed the 'Extended Riemann Hypothesis') that ran in polynomial time as well as a linear expected time algorithm for finding the closest pair of points from a given set. The primality test was (one directional) 'Monte Carlo' - it would always output 'prime' on a prime input and would output 'non-prime' with high probability on a composite. Since then several randomized algorithms for primality have been discovered as well as a deterministic one (see Schoof [4]). The fact that fast algorithms for primality exist made the RSA cryptosystem feasible (it was suggested not long after and based on picking two random primes P and Q and making public their product $N = P \cdot Q$) .

A simple example we skipped was for checking matrix multiplication: given three $n \times n$ matrices A, B and C how do you check that $A \cdot B = C$, say over the finite field $GF[2]$? To recompute the product $A \cdot B$ is relatively expensive: the asymptotic time it takes is denoted as $O(n^\omega)$ where the current (as of 2014) best value for ω is ≈ 2.3728639). But in 1977 Freivalds suggests and $O(n^2)$ algorithm for verification: picking at random a vector $r \in \{0,1\}^n$ and compute $A(Br))$ and Cr and compare the two resulting vectors. If $AB = C$ then the algorithms always says 'yes' and if $A \cdot B \neq C$ then the algorithm says 'no' with probability at least $1/2$.

This algorithm is a good example for the theory of program checking.

The main algorithms we saw were for 2-SAT and 3-SAT. The second one is due to Uwing Schöning You can find a description in Chapter 7 of Mitzenmacher and Upfal. The complexities of the algorithms are roughly $O(n^2)$ and $O((4/3)^n)$ respectively.

A famous conjecture, called the Exponential Time Hypothesis (ETH), states that 3-SAT cannot be solved in sub-exponential time, i.e., in times less than $(1 + \alpha)^n$ for some $\alpha > 0$.

References

[1] Russell Impagliazzo and Avi Wigderson, *P = BPP if E Requires Exponential Circuits: Derandomizing the XOR Lemma,* STOC 1997, pp. 220–229.

[2] Dick Lipton's blog, "Rabin Flips a Coin", March 2009
`https://rjlipton.wordpress.com/2009/03/01/rabin-flips-a-coin/`

[3] Michael Oser Rabin, *Probabilistic algorithms.* In Algorithms and complexity: New Directions and Recent Results, pages 21 - 39. Academic Press, New York.

[4] Rene Schoof, *Four primality testing algorithms*, `http://arxiv.org/abs/0801.3840`

[5] Claude Shannon, *A Mathematical Theory of Communication.* Bell System Technical Journal

27 (3): 379 - 423, (July/October 1948).
http://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf