

Lecture Notes for CS 509

Foundations of Computer Science

Lecture 6

Lecturer: Joe Kilian
Scribe: Thomas Walsh
Department of Computer Science
Rutgers, The State University of New Jersey

1 Bounded-error, Probabilistic, Polynomial Time (BPP)

Consider a probabilistic Turing Machine M , which runs in polynomial time with the following distribution on its outputs:

- if $x \in L$, $M(x)$ accepts with probability $\frac{3}{4}$
- if $x \notin L$, $M(x)$ accepts with probability $\frac{1}{4}$

Claim: BPP is contained in exponential time, that is $\text{BPP} \in \text{EXP}(2^{N^{O(1)}})$

This is a trivial brute force bound determined by running through all the possible coin tosses and counting how many accept and how many fail. Viewing it this way means we can construct a machine: $M(x, r)$ where r represents the outcomes of our coin toss(es).

It is widely believed that $x \in \text{BPP} \implies x \in \text{P}$, but there are some interesting things we *can say* definitively about BPP:

We can replace the $\frac{3}{4}$ and $\frac{1}{4}$ in the above example (which seemed fairly arbitrary) with an arbitrarily larger gap (such as $\frac{1}{99}$ and $\frac{98}{99}$) by computing $M(x, r_1), M(x, r_2) \dots M(x, r_k)$ for independently drawn r and then accepting if the majority accept. The probability of this is exponential in k , so we can drive these numbers to whatever gap we need, such as:

- if $x \in L$, $M(x)$ accepts with probability $1 - 2^{-|x|^2}$
- if $x \notin L$, $M(x)$ accepts with probability $2^{-|x|^2}$

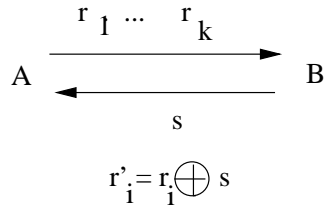


Figure 1: Alice and Bob's game to see if $M(x, r)$ accepts

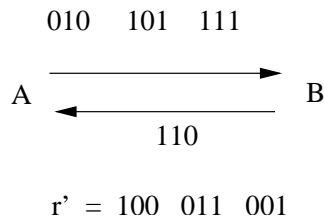


Figure 2: Alice and Bob's game with $k = m = 3$

but we can't get the probability down to 0. Here is yet another interesting result:

$$\text{BPP} \in \Sigma_2^{\text{Poly}}$$

Why is this case? Well, take a language that takes input x and random coin toss r , $M(x, r)$, such that an extraordinarily high number of coin tosses give us the right answer (like $1 - 2^{-|x|^2}$ as in the example above). Now we'll play the following game between Alice and Bob, illustrated in Figure 1. Alice sends k r 's (represented as binary strings because each one is really a series of coin tosses) over to Bob, who now masks them by sending back a bit string s which is used to construct a set of k new r' values, each one being defined as $r'_i = r_i \oplus s$, where \oplus is a bitwise or. An example for $k = 3$ and bit strings of length 3 is provided in Figure 2.

Now we have the following result: Alice wins $\iff M(x, r'_i)$ accepts for some r' . (remembering that Alice winning means we accept). The reasoning behind this result is the following:

If Alice and Bob play randomly and $x \in L$ then Alice will win with high probability (because it is unlikely Bob can mask all the strings Alice is sending to make M reject every time). But, if $x \notin L$, Bob wins with high probability. Why is this the case?

Well consider what the chances are that $M(x, r'_i)$ accepts. The actual numeric answer depends on the strategies of the players, but for now we'll just assume that Bob picks s at

randoms. If we consider any single string r'_i given that s is uniformly distributed, then we'd like to say that each r'_i would be uniformly distributed as well, but actually, Alice could be clever in her picking of each r_i and force the r'_i values to have correlations. Instead, we turn to the following result: No matter how r_i is chosen:

$$Pr(\exists i, M(x, r'_i) \text{ accepts}) \leq \sum_i Pr(M(x, r'_i) \text{ accepts})$$

(because the probability of one event in a set occurring is bounded by the probability of all of them occurring). That is, the probability that any r'_i leads to acceptance is bounded by $k * Pr(M \text{ accepting})$ no matter what k Alice actually picked, as long as k is not exponentially big.

Combining that result with the useful fact “If a random string wins with non-zero probability then Bob can win with probability 1”, we get that if $x \notin L$ then Bob will win.

But now what if $x \in L$? In the previous analysis, Bob had the distinct advantage of going second. We notice that if Bob is acting randomly as before, and $x \in L$ then Bob will almost certainly lose. But what if Bob acts “better” than randomly? What if Bob, seeing r_1 , chooses s to turn it into some r_{bad} such that $M(x, r_{bad})$ rejects (that is $r_1 \oplus s = r_{bad}$). Why then can't Bob force all $r_1 \dots r_k$ to r_{bad} ?

Claim: if $|s| = m$, Bob can improve his chance of winning by at most 2^m over $s = 0^m$ strategy (or $s = \text{random}$ strategy). That is, the random strategy does within 2^{-m} of the optimal choice. So Alice should just choose k to be $k \gg m$ (but still polynomial). Then, the probability that Bob wins with a random strategy $< 2^{-k}$, and using the previous result, we see Bob's best chance of winning is 2^{m-k} , so Alice can now ensure Bob's chance of winning is 0.

2 Complete Problems

$L_1 \leq_l L_2$ if there exists an “efficient” mapping f such that $(x \in L_1 \iff f(x) \in L_2)$. The definition of “efficient” can change depending on the complexity class we are investigating. For instance, if $L_1 \in P$ and $L_2 \in P$ then “efficient” could mean “in polynomial time”. But, if $L_1, L_2 \in \text{LOG-SPACE}$ then “efficient” might mean “in log space”. (Note: LOG-SPACE describes Turing machines with working tapes that are only of size $\log(n)$ where n is the size of the input.

2.1 A Complete Problem for P

Given $\langle C \rangle$, which is a description of a circuit, and an input x , the question “is $C(x) = 1$?” is P-Complete. Consider the following generic Turing tableau depicted in Figure 3 where

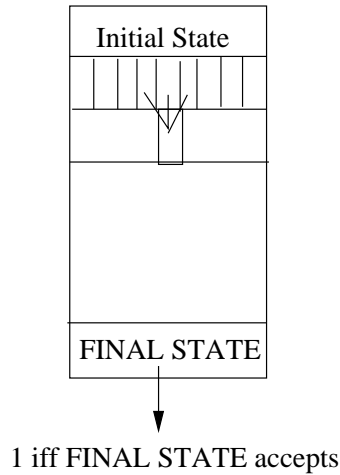


Figure 3: A Turing tableau, each single bit entry depends only on the three entries above, above-left, and above-right

the rows of the tableau represent the configuration of the TM after each step. Notice that each entry in each row is dependent solely on the three entries appearing above it, and that such a transformation can be represented by a circuit. The output of the total circuit C is 1 iff the final state is an accepting state, so the problems are equivalent.

2.2 A Complete Problem for NP

Recall that we can write an NP language as $x \in L \iff \exists y D(x, y)$ for polynomial time decidable D . Since $D \in P$ we can express D as a circuit with known inputs x and some unknown inputs y . Thus, we can express $D(x, y)$ as $D_x(y)$, the circuit with the inputs x hard wired in. So one NP-Complete problem is: $\exists y, C(y)$ outputs 1? (where C is again a circuit).

2.3 A more interesting NP-Complete problem

We now derive a more interesting NP-Complete problem from the simple one above, namely the classic 3-SAT problem.

Claim: Given $C(x)$ we can create $C'(x, A)$ such that $\exists x C(x) \iff \exists x, A, C'(x, A)$ where C' is a *shallow* formula. How? Well a circuit is just a bunch of gates like in Figure 4. We can make auxillary variables corresponding to the *output* of each gate, call them a_g for

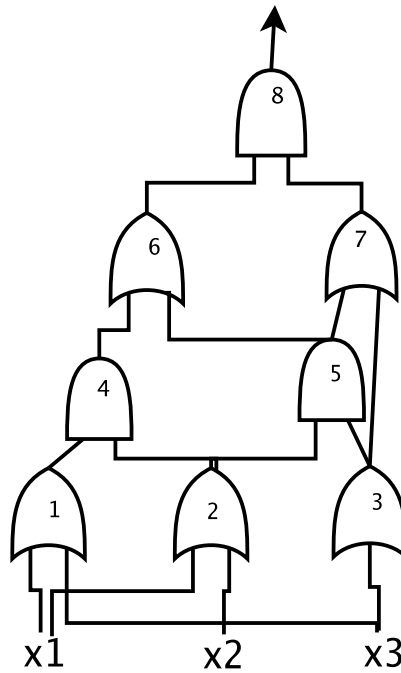


Figure 4: Sample circuit

gate g , and the the set $A = \{a_g\}$. So for instance, the circuit in Figure 4 can be encoded as $C'(x, A) = 1 \iff a_{g8} = 1, a_{g1} = x_1 \vee x_3, a_{g4} = a_{g1} \wedge a_{g2} \dots a_{g6} = a_{g4} \vee a_{g5}$. This allows us to view the variables as $x_1 \dots x_k, a_1 \dots a_m$. The full circuit can thus be compressed down to one giant AND of all the x 's and a 's (so it is an incredibly shallow circuit. In fact, you can collapse this down to a formula in Clausal Normal Form (CNF), which brings us to the definition of 3-SAT:

3-CNF is a special case of clausal normal form with 3 clauses:

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\dots \vee \dots \vee) \wedge (\dots \vee \dots)$$

3-Sat is then the following problem:

$$\{3\text{-CNF} \mid \exists x_1 \dots x_m F(x_1 \dots x_m)\}$$

which is “the mother of all NP-Complete Problems”.

2.4 A PSPACE Complete Problem

We make use of PSPACE being Alternating P-TIME. That is, with two players, Alice and Bob, alternating turns, each eventually making m moves with m being polynomial in the input size, a referee R makes a decision as to who “wins” based on all $2m$ moves:

$$\exists a_1 \forall b_1 \dots \exists a_m \forall b_m R(a_1 \dots a_m, b_1, \dots b_m)$$

We can consider, without loss of generality, all the moves to be one bit moves. Since R is polynomial it can be represented as a circuit, giving us:

$$\exists a_1 \forall b_1 \dots \exists a_m \forall b_m C(a_1 \dots a_m, b_1, \dots b_m)$$

We call this a **Quantified Boolean Circuit** (QBC). We can squash the circuit into a formula to get:

$$\exists a_1 \forall b_1 \dots \exists a_m \forall b_m \exists AF(a_1 \dots a_m, b_1, \dots b_m)$$

Consider now the **True Quantified Boolean Formula** (TQBF) = {QBF F that are true} Thus, any problem in PSPACE can be converted into a game and then converted into TQBF, so TQBF is a PSPACE-Complete problem.