Probabilistic Computation - BPP and RP

Certain algorithms benefit from randomness. For example, the 'pivot' in quicksort should be chosen randomly. We can also potentially get drastic speedups if we accept some probability of our program producing the wrong answer.

a Brainstorm a few different ways we could incorporate the ability to make random choices into the formal definition of a TM. Are there any clear advantages or disadvantages of your different formalizations?

Solution:

- We can use a normal deterministic multi-tape machine where one of the tapes starts with an infinite random sequence of characters. Then whenever the TM wants to make a random choice it can read from that randomness tape.
- We can use a normal nondeterministic machine, except instead of saying that the machine accepts if any one branch accepts, we say that the machine accepts with probability equal to the chance it reaches an accept state. Sipser's definition (10.3) is similar to this one, except it also requires each branching step to only branch into two options.

b (Sipser p397) For $0 \leq \epsilon < \frac{1}{2}$, we say that (probabilistic) TM M decides language A with error probability ϵ if

- $w \in A$ implies $\Pr[M \text{ accepts } w] \geq 1 - \epsilon$, and
- $w \notin A$ implies $\Pr[M \text{ rejects } w] \geq 1 - \epsilon$

BPP is the class of languages that are decided by probabilistic polynomial time Turing machines with an error probability of $\frac{1}{3}$.

Show that if a language L is in BPP, then there is a probabilistic polynomial time TM which decides L with an error probability of $\frac{3}{10}$. Why do you think $\frac{1}{3}$ was chosen as the threshold for BPP?

Solution:

Since L is in BPP, there is an M that decides it with an error probability of $\frac{1}{3}$. Run M 3 times and *accept/reject* based on majority vote. This has error probability $(\frac{1}{3})^3 + 3 * (\frac{1}{3})^2 * \frac{2}{3} = \frac{7}{27} < \frac{3}{10}$. By a similar argument, any constant threshold strictly less than $\frac{1}{2}$ would have given us the same class BPP.

$\frac{1}{3}$ was presumably chosen by Sipser as the "simplest" fraction less than a half. (I have also seen sources that define BPP using $\frac{1}{4}$, presumably because it's the "simplest" fraction less than a half which is also a power of 2 since CS folks like those, and values like $\frac{499}{1000}$, presumably to emphasize the fact that any constant less than a half, even a tiny bit less, is enough.)

c Using only complexity classes from the list **L**, P, $PSPACE$, and $EXPTIME$, provide the tightest possible bounds on BPP (i.e. what's the strongest X and Y for which we have $X \subseteq BPP \subseteq Y$?)

Solution:

$P \subseteq BPP$, since any poly-time deterministic TM can be thought of as a probabilistic one that just isn't using its randomness powers (and thus has error probability of 0).

$BPP \subseteq PSPACE$, since using polynomial space we can deterministically simulate a BPP machine by iterating through all possible random seeds.

d Consider the problem where you are given black-box access to a polynomial $f(x)$ (i.e. you can evaluate it efficiently on inputs of your choice but you can't see its coefficients), and you need to determine whether $\forall x, f(x) = 0$. Outline an efficient probabilistic algorithm for solving this problem, in order to argue informally that this problem is in BPP. *(The full details are a bit more complicated to pin down.)*

Note:

This problem was underspecified; the key thing I was missing in our discussion in class is that the polynomial needs to have multiple variables - this is what prevents you from coming up with a trivial upper bound on the number of zeros and using that to create a zero-error solution.

Solution:

Try some tractable number of random inputs, and declare that the polynomial is identically zero if and only if all the inputs give zero as output. The only way this fails is if the polynomial is non-zero somewhere yet we get very unlucky and don't find any such inputs.

e Some algorithms in BPP (including probably the one you came up with in the previous part) have the property of *one-sided error*, captured in the following class definition: Define RP to be the class of languages decided by probabilitic poly-time TMs where inputs in the language are accepted with a probability of at least $\frac{1}{2}$, and inputs not in the language are rejected with a probability of 1.

Why was it ok to define RP with a "$\geq \frac{1}{2}$" rather than the "$\geq \frac{2}{3}$" we used for BPP? (What would have gone wrong if we'd used "$\geq \frac{1}{2}$" in the definition of BPP?)

Solution:

With one-sided error, we can use the amplification strategy from earlier to bring the $\frac{1}{2}$ error threshold lower - e.g. if you run the algorithm three times on an input that should be accepted, then the chance it's rejected has dropped exponentially to $\frac{1}{8}$, while the chance that an input which should be rejected is indeed rejected remains 1. So reject and accept both can become arbitrarily accurate signals. (Whereas if we'd tried to define BPP using $\frac{1}{2}$, then if a TM just outputs the result of a coin flip on every input, there is no way to 'amplify' that to get useful results.)

f Show that if $NP \subseteq BPP$ then $NP = RP$.

Solution:

First we note that RP is (unconditionally) a subset of NP. This is because if RP-TM M decides L, then there is a random seed which will cause M to accept a given w iff $w \in L$, so that seed is the poly-time verifiable certificate.

Now suppose $NP \subseteq BPP$, and it remains to show that $NP \subseteq RP$. We will do this by showing NP-complete language SAT is in RP. By the assumption, $SAT \in BPP$, so let M be a BPP-TM deciding SAT. Then we can leak a k-variable assignment one variable at a time by choosing a value for the variable and calling M on the remaining formula (calling it enough times that the error probability is $\leq \frac{1}{2k}$), rejecting if M says there is no satisfying assignment. Finally we can check the leaked assignment before accepting. So if there is no satisfying assignment, we reject with probability 1, and if there is a satisfying assignment, then we find it and accept with probability bounded above by $k * \frac{1}{2k} = \frac{1}{2}$.