

2 Randomized Complexity Classes

Today's agenda roughly follows section 1.5 of the textbook:

- Randomized complexity classes: PP, BPP, RP, ZPP.

Recall that a randomized algorithm is one that takes in an input x and a (uniformly) random string r . What does it mean for a randomized algorithm to be correct and efficient? This is not as clear as worst-case bounds for deterministic algorithms.

To start, let us suppose a randomized algorithm is **efficient** (polynomial time) if the runtime of your algorithm $A(x, r)$ is at most $\text{poly}(|x|)$ for all x, r . Now suppose the algorithm is **correct** if it outputs the correct answer with probability $> \frac{1}{2}$. This interpretation gives us the class PP.

DEFINITION 2.1 (PP (PROBABILISTIC POLYNOMIAL)) A language L is in **PP** iff there exists a polynomial time randomized algorithm A such that

- If $x \in L$, then $\Pr_r(A(x) \text{ accepts}) > \frac{1}{2}$.
- If $x \notin L$, then $\Pr_r(A(x) \text{ accepts}) < \frac{1}{2}$.

As it turns out, PP is extremely powerful. For example, $\text{NP} \subset \text{PP}$.

THEOREM 2.2 $\text{NP} \subset \text{PP}$. In other words, PP is very powerful.

Proof. Suppose $L \in \text{NP}$. Then there exists a polynomial-time deterministic algorithm B and a polynomial p such that

$$x \in L \iff \exists y \in \{0, 1\}^{p(|x|)} \text{ such that } B(x, y) \text{ accepts.}$$

Construct a randomized algorithm $A(x, r)$ as follows:

- With probability $\frac{1}{2} - \epsilon$, accept.
- Otherwise, pick y at random and accept if $B(x, y)$ accepts.

If $x \notin L$, then no y makes $B(x, y)$ accept, so

$$\Pr_r[A(x) \text{ accepts}] = \frac{1}{2} - \epsilon < \frac{1}{2}.$$

If $x \in L$, then there is at least one y such that $B(x, y)$ accepts, so

$$\Pr_r[A(x) \text{ accepts}] \geq \frac{1}{2} - \epsilon + 2^{-|y|} \left(\frac{1}{2} + \epsilon \right) > \frac{1}{2}. \quad \square$$

Informally, this means $\text{PP} \approx \text{P}^\#$ (the \approx is here because PP is a decision problem but $\text{P}^\#$ is a counting problem, so this comparison is not well-defined), the class that counts the number of witnesses to an NP problem. This means that PP is not very practical, since we don't have an efficient way to distinguish acceptance probability $0.50\dots001$ and 0.5 , for example. The class BPP aims to solve this problem.

DEFINITION 2.3 (BPP (BOUNDED-ERROR PROBABILISTIC POLYNOMIAL)) A language L is in **BPP** iff there exists a polynomial time randomized algorithm A such that

- If $x \in L$, then $\Pr_r[A(x) \text{ accepts}] > \frac{2}{3}$.
- If $x \notin L$, then $\Pr_r[A(x) \text{ accepts}] < \frac{1}{3}$.

Our first remark is that $\frac{2}{3}$ and $\frac{1}{3}$ feel arbitrary. What matters is that there is a constant gap away from $\frac{1}{2}$. More generally, bounds of the form $> c + \epsilon$ and $< c - \epsilon$ for some constants $c, \epsilon \in \mathbb{R}$ are sufficient.

Indeed, we can **amplify** the success probability by running the algorithm multiple times with independent randomness. Suppose we run the algorithm k times and accept if the majority of runs accept. By a Chernoff bound, the probability that the majority vote is incorrect decreases exponentially in k . More precisely, to reduce the error probability to at most δ , it suffices to take

$$k = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right).$$

Then the majority vote is correct with probability at least $1 - \delta$.

Thus, we can make the error probability exponentially small while only increasing the runtime by a polynomial factor. However, no matter how many repetitions we perform, the error probability can never be reduced to 0.

But perhaps we may want probability 1 for one of our cases in a decision problem. Recall, for instance, the Solovay-Strassen primality test algorithm from last lecture. This algorithm only has 1-sided error: If n is prime, the algorithm always accepts (it claims it is prime). On the other hand, there is a chance that even if n is composite, we accept (this can happen when we randomly choose an a that satisfies the equation.)

DEFINITION 2.4 (RP (RANDOMIZED POLYNOMIAL TIME)) A language L is in **RP** iff there exists a polynomial-time randomized algorithm $A(x, r)$ such that

- If $x \in L$, then $\Pr_r(A(x) \text{ accepts}) > \frac{1}{2}$.
- If $x \notin L$, then $\Pr_r(A(x) \text{ accepts}) = 0$.

Thus **RP** is a strengthening of **BPP** that gives us one-sided error. Thus Solovay-Strassen witnesses that primality testing is in **coRP** (and the language of composites is in **RP**).

Now, what if we want no false negatives or false positives? There are a few interpretations of this question. If we want the runtime to be deterministically polynomial, this class is **P**; the randomness does not matter, and this is guaranteed to run in polynomial time and give the right answer. But what if we just require that the *expected* runtime is polynomial?

DEFINITION 2.5 (ZPP (ZERO-ERROR PROBABILISTIC POLYNOMIAL)) A language L is in **ZPP** iff there exists a randomized algorithm A that always computes the right answer and the expected runtime is $\mathbb{E}_r[\text{runtime}(A(x, r))] = \text{poly}(|x|)$.

THEOREM 2.6 $\text{ZPP} = \text{RP} \cap \text{coRP}$

Proof. ($ZPP \subseteq RP \cap \text{coRP}$) Let $L \in ZPP$ with algorithm A of expected runtime $R = \text{poly}(|x|)$. Define a new algorithm that runs A for at most $10R$ steps. If A has halted, return the output of A . Else, terminate and reject (output 0).

If $x \notin L$, then A always outputs 0, so the modified algorithm also always outputs 0. If $x \in L$, then A always outputs 1, unless we terminate early. By Markov's inequality,

$$\Pr[\text{runtime}(A) > 10R] \leq \frac{\mathbb{E}[\text{runtime}(A)]}{10R} \leq \frac{1}{10}.$$

Thus the algorithm outputs 1 with probability at least $\frac{9}{10}$, so $L \in RP$. By a symmetrical proof, $L \in \text{coRP}$.

($RP \cap \text{coRP} \subseteq ZPP$) Suppose $L \in RP \cap \text{coRP}$. Then there is an RP algorithm A (no false positives) and a coRP algorithm B (no false negatives). Construct a new algorithm that runs A and B independently (with fresh randomness) until one of them gives a definitive answer: if A accepts, output 1; if B rejects, output 0.

Since at least one of these events occurs with constant probability in each round, the expected number of rounds until termination is constant. Therefore, the expected runtime is polynomial, and the algorithm is always correct. Hence $L \in ZPP$. \square