- In a probabilistic Turing machine (PTM), two transition functions δ_0 and δ_1 are given and one of these is chosen uniformly at random, independent of previous choices, at each step of computation. And, irrespective of the random choices it makes, PTM halts in $t(|x|)$ steps for any input x, where $t()$ is the time complexity of the algorithm.

- The complexity class *bounded-error probabilistic polynomial time* (BPP) consists of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any x in Σ^*, $x \in L \Rightarrow pr(A \text{ accepts } x) \geq \frac{3}{4}$, and $x \notin L \Rightarrow pr(A \text{ accepts } x) \leq \frac{1}{4}$. [1]

  Equivalently, language L $\in$ BPP whenever a DTM accepts $< x, r >$ with at least $\frac{2}{3}$ probability for every $x \in L$. Here, r is a string of bits, and $|r|$ is a polynomial in $|x|$. Each bit in r is chosen independently and uniformly at random. Every bit in r helps PTM to choose between δ_0 and δ_1.

  The BPP class captures languages that have Monte Carlo algorithms with two-sided errors.

  P $\subseteq$ BPP

  proof: an algorithm to claim $L \in$ P ignores random bits; and, in polynomial time, accepts $x \in L$ with prob 1 (which is $\geq 3/4$) and accepts $x \notin L$ with prob 0 (which is $\leq 1/4$)

  coBPP $\subseteq$ BPP

  proof: for any $L \in coBPP$, $\overline{L} \in BPP$; for any $x \in \overline{L}$, there is a polynomial time DTM M such that pr(M accepts x) $\geq 3/4$ and for any $x \in L$, pr(M accepts x) $\leq 1/2$; construct a DTM M' by exchanging accept and reject states of M; it is immediate to note M takes only polynomial time; for any $x \in L$, since pr(M accepts x) $\leq \frac{1}{4}$, pr(M' accepts x) $\geq \frac{3}{4}$; for any $x \notin L$, since pr(M accepts x) $\geq \frac{3}{4}$, pr(M' accepts x) $\leq \frac{1}{4}$

  BPP $\subseteq$ coBPP

  proof: analogous to coBPP $\subseteq$ BPP

  BPP $\subseteq$ PSPACE

  proof: for any language $L \in$ BPP, a PTM M for L makes $O(n^k)$ coin tosses along any branch of its computation when n is the input size; there are $2^{O(n^k)}$ computation paths; if the number of paths among these that accept is $\geq (3/4)(2^{O(n^k)})$, then accept, otherwise, reject; to simulate all possible computation paths while reusing the space requires only polynomial space

  BPP $\subset$? [2] NEXPTIME

  BPP $\subseteq$? NP

  BPP $\subseteq$? coNP

  BPP $=$? P

  No known BPP-complete language.

- The complexity class *randomized polynomial time* (RP) consists of every language L that has a randomized algorithm A running in worst-case polynomial time such that for any input x in Σ^*,
  if $x \in L$ then $pr(A \text{ accepts } x) \geq \frac{1}{2}$, and if $x \notin L$ then $pr(A \text{ rejects } x) = 1$. [3]

[1] These error probabilities can be reduced to $\frac{1}{2^n}$ with only a polynomial number of iterations; hence, the class BPP is robust with respect to specific error probabilities used in defining it.

[2] plausible but no proof known

[3] The choice of the bound on the error probability $\frac{1}{2}$ is arbitrary. The independent repetitions of the algorithm can reduce the probability of error with a change in the polynomial that bounds the running time.

The RP class captures languages that have Monte Carlo algorithms with one-sided errors.

P $\subseteq$ RP

proof: algorithm A to claim $L \in$ P accepts $x \in L$ with prob 1, which is $\geq 1/2$; and, A accepts any string in $\Sigma^* - L$ with prob 0, and it does so in polynomial time

RP $\subseteq$ BPP

proof: for $x \in L$, prob of acceptance of an algorithm to claim $L \in$ BPP is $\geq 3/4$, which is $\geq 1/2$ required to claim $L \in$ RP; for $x \notin L$, prob of acceptance of an algorithm to claim $L \in$ RP is zero, which is $\leq 1/4$ required to claim $L \in$ BPP

RP $\subseteq$ NP

proof: replace using a random bit in an RP machine with non-determinism

RP $\subseteq$? NP $\cap$ coNP

- The complexity class coRP consists of languages that have polynomial time randomized algorithms erring only when $x \notin L$. Specifically, $L \in RP$ iff $\overline{L} \in coRP$.

P $\subseteq$ RP $\cap$ coRP

proof: P consists of languages that have polynomial time algorithms (that do not require any random bits) and accept/reject with zero error prob

coRP $\subseteq$ BPP

proof: analogous to $RP \subseteq BPP$

coRP $\subseteq$ coNP

proof: analogous to showing $RP \subseteq NP$

RP =? coRP

- The *zero-error probabilistic polynomial time* (ZPP) is the class of languages that have Las Vegas algorithms running in expected polynomial time. For example, due to randomized quicksort algorithm, decision version of sorting problem belongs to ZPP.

A problem belonging to both RP and $coRP$ can be solved by a randomized algorithm with zero-sided error, i.e., a Las Vegas algorithm.

RP $\cap$ coRP $\subseteq$ ZPP

proof: construct the following DTM M

do {

- run the Monte-Carlo DTM M' that accepts L; if M' accepts, then M accepts
- if not, run the Monte-Carlo DTM M'' that accepts $\overline{L}$; if M'' accepts, then M rejects

}

the expected running time of M is $2p(n) + \frac{1}{2}(2p(n)) + \frac{1}{4}(2p(n)) + \ldots = 4p(n)$

ZPP $\subseteq$ RP $\cap$ coRP

proof: given L is accepted by a Las-Vegas DTM M_1 that runs in expected polynomial time, construct a Monte-Carlo DTM M_2 that does the following: if M_1 accepts w in $2p(|w|)$ time, so does M_2; otherwise, M_2 rejects

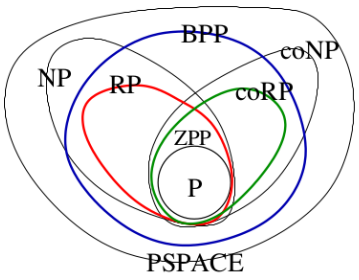
prob M_2 accepts w in $p(|w|)$ time is at least $1/2$: prob M_1 accepts w in $2p(|w|)$ steps is $\geq 1/2$; expected time of M_1 on w is $\geq (1/2)2p(|w|)$;

proving $ZPP \subseteq coRP$ is same as above except that, M_2 accepts when M_1 rejects and M_2 rejects when M_1 accepts

$P \subseteq ZPP$

proof: a DTM that does not use random bits is obviously a Las-Vegas polynomial-time bounded

- In conclusion,



- Are there any NP-hard problems for which there is an algorithm that takes expected polynomial time? Here, the expected time is over the distribution of random bits used by the algorithm.

  Are there any NP-hard problems for which there is an algorithm with average-case time? Here, the expected time is over random distribution of inputs.

  Are these two problems related? Via Yao's minimax principle?

  Yao's minimax principle states that, for any problem Π, the expected running time of the optimal deterministic algorithm for an arbitrarily chosen input distribution p is a lower bound on the expected running time of the optimal Las Vegas randomized algorithm for Π.

  That is, to establish a lower bound on the performance of randomized algorithms, it suffices to find an appropriate distribution p on (difficult) inputs, and to prove that no deterministic algorithm can perform well against that distribution. The power of this technique lies in the flexibility in the choice of p and, more importantly, the reduction to a lower bound on deterministic algorithms.

References:
- Randomized Algorithms by R. Motwani and P. Raghavan.
- Introduction to Automata Theory, Languages, and Computation by J. E. Hopcroft, R. Motwani, and J. D. Ullman.