Introduction to Theoretical Computer Science

Lecture 16: Complexity of Randomized Computation

Richard Mayr

University of Edinburgh

Randomized RMs/TMs

Consider machines where the non-deterministic MAYBE instructions are replaced by probabilistic fair coin tosses.
Chance $1/2$ of Head and Tail each.
If Head goto location x.
If Tail goto location y.
Given such a machine M and input w, this yields a probability distribution over computations.
Nothing magic about the fair coin toss: One can easily create small probabilities by tossing the fair coin k times. Chance of all Heads is 2^{-k}.

Important: We still consider the worst case over all possible inputs of size n.
Nothing random about the input.
The randomness is inside the algorithm.

Here we consider **decision problems**. (Some classes like EP, ZPP, BPP can also be defined for general problems.)

Classes EP and ZPP

Class EP

EP is the class of problems that can be solved by an algorithm that runs in worst case expected polynomial time.

Clearly $P \subseteq EP$, but for EP it is allowed that a small minority of the computations can be very long.
Let n be the size of the input and $\varepsilon(n) < 1$ be an error function.
Intuitively, $\varepsilon(n)$ is the probability that something goes wrong.

Class ZPP

$ZPP(\varepsilon(n))$ is the class of problems that can be solved by a worst case polynomial time algorithm that never gives wrong answers, but may fail with some probability $\leq \varepsilon(n) < 1$.

ZPP = zero-error probabilistic polynomial, also called “Las Vegas algorithms”.
Trivially ZPP = coZPP.

EP vs. ZPP(1/2)

Theorem

EP=ZPP(1/2).

Proof ideas:

$EP \subseteq ZPP(1/2)$. If the algorithm stops in expected polynomial time $p(n)$ then the probability of running for longer than $2p(n)$ is $\leq 1/2$. Thus the alg. solves the problem in polynomial time $\leq 2p(n)$ with probability $\geq 1/2$.

$ZPP(1/2) \subseteq EP$. Just run the $p(n)$ ZPP alg. again and again. The probability that this yields an answer by time $k * p(n)$ is $\geq 1 - 2^{-k}$. Thus the expected stopping time is polynomial.

Classes RP and co-RP

The probabilistic version of NP.

Class RP

$RP(\varepsilon(n))$ is the class of problems that can be solved by a worst case polynomial time algorithm that always rejects false instances, but rejects true instances only with a probability $\leq \varepsilon(n) < 1$.

RP=Randomized polynomial time.
Admits only a one-sided error.

Dually for coRP, i.e., $L \in \text{coRP}(\varepsilon(n))$ iff $\overline{L} \in RP(\varepsilon(n))$.

Clearly $P \subseteq ZPP(1/2) = RP(1/2) \cap coRP(1/2)$.
Moreover, $RP(1/2) \subseteq NP$ and $coRP(1/2) \subseteq coNP$.

The class BPP

Class BPP

$BPP(\varepsilon(n))$ is the class of problems that can be solved by a worst case polynomial time algorithm that reports a wrong result only with a probability $\leq \varepsilon(n) < 1/2$.

Admits two-sided error, i.e., false positive and false negative.

BPP = bounded-error probabilistic polynomial, also called “Monte Carlo algorithms”.
Trivially BPP = coBPP.
Clearly,
$RP(1/2) = RP(1/3) \subseteq BPP(1/3)$ and
$coRP(1/2) = coRP(1/3) \subseteq BPP(1/3)$.

Robustness

Theorem

$ZPP(1 - 1/p(n)) = ZPP(1/2) = ZPP(2^{-q(n)})$ for polynomials p, q.

Proof ideas: For the first equality, re-run the algorithm $p(n)$ times, and consider $\lim_{k\to\infty}(1 - 1/k)^k = 1/e < 1/2$.
For the second equality, re-run the algorithm $q(n)$ times.

Likewise for RP and co-RP.
Also for BPP, provided that the error bounds are $< 1/2$ here, i.e., $BPP(1/2 - 1/p(n)) = BPP(2^{-q(n)})$.

General definitions

- $ZPP := ZPP(1/2)$.
- $RP := RP(1/2)$.
- $BPP := BPP(1/3)$.

Relation to classic complexity classes

$RP = RP(1/2) \subseteq NP$, but
$NP = RP(1 - 2^{-p(n)})$ for polynomials $p(n) \geq n$.

NP requires just one one accepting computation.
Even a single computation of polynomial length $p(n)$ with thus $\leq p(n)$ fair coin tosses has probability $\geq 2^{-p(n)}$.

So NP is just RP with a very weak (too weak?) error bound.

$P \subseteq ZPP \subseteq RP \subseteq NP \subseteq PSPACE$ and
$RP \subseteq BPP \subseteq \Sigma_2^p \cap \Pi_2^p$ (Sipser–Lautemann theorem)
and thus $\subseteq PSPACE$.

Open questions: Are any of these inclusions strict?
And what is the relationship between *BPP* and *NP* ?

BPP

Intuitively, $BPP = BPP(1/3)$ is the largest class of problems where the worst case can be solved efficiently by standard machines.
Just run the algorithm a few dozen times, take the majority vote as outcome, and the error bound becomes tiny.

Many experts conjecture that $BPP = P$.
(On the other hand, hardly any believe that $NP = P$.)

For quantum computation, the corresponding (extended) class is **BQP**.
E.g., the factorization problem for integers is in BQP.
(So a functioning quantum computer could break RSA encryption.)
$BQP \subseteq PSPACE$ (and it is still possible that $BQP = P$).

RP

Theorem

Polynomial identity testing is in coRP (and not known to be in P).

Consider two multivariate polynomials p, q, defined via arithmetic circuits in a field.
This representation acts like a black box.
(It takes too long to compute the coefficients of the polynomials from the given circuits.)
So you cannot look at the coefficients of the polynomials, but you can efficiently compute what $p(x_1, \ldots, x_k)$ and $q(x_1, \ldots, x_k)$ are for chosen inputs $x_1, \ldots, x_k$.
Checking whether $p = q$ is in coRP.

Why? If $p \neq q$ then it is fairly likely that $p(x_1, \ldots, x_k) \neq q(x_1, \ldots, x_k)$ for a randomly chosen $x_1, \ldots, x_k$.

Primality

Given a natural number x, is it a prime?

Note that the size of the instance is not x but $\log(x)$, the number of bits used to describe x.

Primality testing

- Trivially in coNP. Guess factors and multiply to check.
- Also in NP (Vaughan Pratt, 1975). Primality certificates.
- Also in coRP (Solovay–Strassen and Miller–Rabin, 1992).
- Even in RP and thus ZPP (Adleman–Huang, 1992).
- Even in P (Agrawal–Kayal–Saxena, 2002).
- Randomized algorithms like Miller–Rabin and Baillie–PSW are more efficient in practice.
- Slightly super-polynomial deterministic algorithms like ECPP and APR-CL also widely used. E.g., $(\log(n))^{\mathcal{O}(\log\log\log n)}$.

This is not factorization. Even if the test replies “not prime”, it does not tell you any factor.