Computational Complexity; slides 12, HT 2022
A Brief Introduction to randomisation

Prof. Paul W. Goldberg (Dept. of Computer Science, University of Oxford)

HT 2022

Randomised Algorithms

Randomised algorithms have access to a stream of random bits.

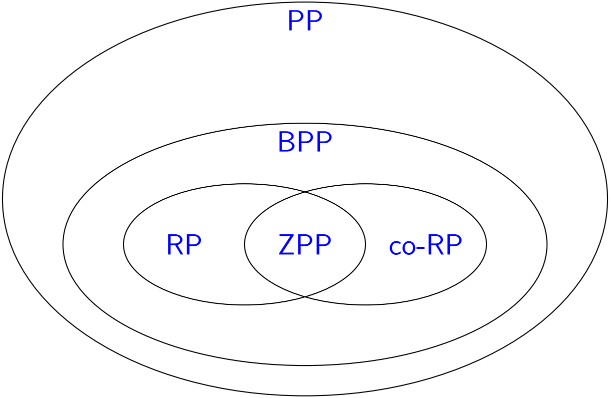
The running time and even the outcome may depend on random choices.

We may allow randomised algorithms to

- produce the wrong result, but only with small probability.
- take more than polynomially many steps, but "not too often"

  $\rightsquigarrow$ expected running time is polynomial.

Some randomised classes



ZPP: "Las Vegas algorithms"; contains P. Poly *expected* time
RP: one-sided error; no-instance$\mapsto$"no", yes-instance$\mapsto$"yes" with probability$\geq p$ (for some constant $p > 0$)
PP: "majority-P", contains NP, within PSPACE
BPP: allow error either way (constant probability $< \frac{1}{2}$)

Usage of randomised algorithms

In practice, not so much for language recognition, more for simulation, crypto, stats/ML, or sampling for probability from probability distributions of interest

search for approximate average via sampling

Find median element of list $\{a_1, \ldots, a_n\}$: To find k-th highest element, randomly select "pivot" element and find k'-th highest element of sublist (for suitable k')

Miller-Rabin test for primality, subsequently superseded by 2002 AKS primality test (deterministic)

- given prime number as input, says "prime"
- Given composite number as input, with prob. $1/4$ says "prime" (correct with prob. $3/4$).

One-sided error; co-RP. Run it k times, say "composite" if we ever get that result, else "prime". Error prob is only $(1/4)^k$.

Language recognition problem where randomisation seems to help

Polynomial identity testing:

E.g. $(x^2 + y)(x^2 - y) \equiv x^4 - y^2$
where $\equiv$ means equality holds for $x, y \in \mathbb{N}$.

In general, if we have many variables, no known deterministic and efficient algorithm, but notice you can try plugging in random x, y and checking for equality: if we find answer is "no" we are done; moreover it turns out that for all no-instances you have good chance of verifying that.

works for arithmetic circuits; consider question $p(x_1, \ldots, x_n) \equiv 0$ for circuit with n inputs, 1 output, gates are $+, -, \times$.

Randomised Complexity Classes

RP⊆NP: accepting computation of an RP machine is a certificate of yes-instance.

It's unknown whether BPP⊆NP, but we argue that BPP represents problems that are in a sense solvable in practice (we expect NP-complete problems to lie outside BPP).

PP (Gill, 1977):
Languages recognised by a probabilistic TM for which yes-instances are accepted with prob. $> \frac{1}{2}$; no-instance with prob. $\leq \frac{1}{2}$.

- PP contains BPP (almost follows directly from the definitions)
- It also contains NP: we can make a PP algorithm that solves SAT. (consider $X \vee \varphi$ where φ is a SAT-instance)
- PP is a subset of PSPACE.

Probability amplification

BPP: problems that can be solved by a randomised algorithm

- with polynomial worst-case running time
- which has an error probability of $\varepsilon < \frac{1}{2}$.

For RP, easy to see how we can improve error probability of algorithm (and evaluate the improvement):
RP: one-sided error; no-instance$\mapsto$"no", yes-instance$\mapsto$"yes" with probability$\geq p$ (for some constant $p > 0$)

For problem X with RP algorithm having (say) $p = 10^{-6}$, run the algorithm 10^6 times, finally output "yes" iff we see at least one "yes" output. Error probability goes down to $< \frac{1}{2}$!

co-RP algorithm: similar trick, output "no" iff we see at least one "no"

Probability Amplification

Corollary for RP algorithms:

Suppose $\mathcal{A}$ solves problem X in polynomial time $p(n)$ and the probability that a yes-instance gives answer "yes" is only $1/p'(n)$ (p' a polynomial), and no-instances always give answer "no". Then $X \in$RP.

Probability Amplification

Corollary for RP algorithms:

Suppose $\mathcal{A}$ solves problem X in polynomial time $p(n)$ and the probability that a yes-instance gives answer "yes" is only $1/p'(n)$ (p' a polynomial), and no-instances always give answer "no". Then $X \in$RP.

Warm-up for BPP: BPP algorithm with error prob $\frac{1}{2} - \delta$: suppose we run it 3 times and take majority vote.

$$\Pr[error] = (\tfrac{1}{2} - \delta)^3 + 3(\tfrac{1}{2} - \delta)^2(\tfrac{1}{2} + \delta)$$
$$= (\tfrac{1}{2} - \delta)^2(\tfrac{1}{2} - \delta + \tfrac{3}{2} + 3\delta) = (\tfrac{1}{4} - \delta + \delta^2)(2 + 2\delta) = \tfrac{1}{2} - \tfrac{3}{2}\delta + 2\delta^3$$

Theorem. If a problem can be solved by a BPP algorithm $\mathcal{A}$

- with polynomial worst-case running time
- which has an error probability of $0 < \varepsilon < \frac{1}{2}$.

then it can also be solved by a poly-time randomised algorithm with error probability $2^{-p(n)}$ for any fixed polynomial $p(n)$.

Probability Amplification

Proof.

Algorithm $\mathcal{B}$: On input w of length n,

1. Calculate number k (to be determined; details to follow)
2. Run $2k$ independent simulations of $\mathcal{A}$ on input w
3. accept if more calls to the algorithm accept than reject.

Probability Amplification

$S := a_1, \ldots, a_{2k}$: sequence of results obtained by running $\mathcal{A}$ $2k$ times.

Suppose c of these are correct and $i = 2k - c$ are incorrect.

S is a bad sequence if $c \leq i$ so that $\mathcal{B}$ gives the wrong answer.

The probability p_S for any individual bad sequence S to occur is

$$p_S \leq \varepsilon^i (1-\varepsilon)^c \quad \leq \quad \varepsilon^k (1-\varepsilon)^k$$

Probability Amplification

$S := a_1, \ldots, a_{2k}$: sequence of results obtained by running $\mathcal{A}$ $2k$ times.

Suppose c of these are correct and $i = 2k - c$ are incorrect.

S is a bad sequence if $c \leq i$ so that $\mathcal{B}$ gives the wrong answer.

The probability p_S for any individual bad sequence S to occur is

$$p_S \leq \varepsilon^i (1-\varepsilon)^c \quad \leq \quad \varepsilon^k (1-\varepsilon)^k$$

Hence: $\Pr[\mathcal{B}$ gives wrong result on input w $] =$

$$\sum_{S \text{ bad}} p_S \;\leq\; 2^{2k} \cdot \varepsilon^k (1-\varepsilon)^k \;=\; (4\varepsilon(1-\varepsilon))^k$$

As $\varepsilon < \frac{1}{2}$ we get $4\varepsilon(1-\varepsilon) < 1$. Hence, to obtain probability $2^{-p(n)}$ we let

$$\alpha = -\log_2(4\varepsilon(1-\varepsilon)) \text{ and choose } k \geq p(n)/\alpha. \qquad \Box$$

General note

So, every problem that can be solved with error probability $\varepsilon < \frac{1}{2}$ can be solved with error probability $< 2^{-p(n)}$.

...practically useful?

General note

So, every problem that can be solved with error probability $\varepsilon < \frac{1}{2}$ can be solved with error probability $< 2^{-p(n)}$.

...practically useful?

Arguably yes:

- the probability that an algorithm with error probability of 2^{-100} has bad luck with the coin tosses is much smaller than the chance that any algorithm fails due to
  - hardware failures,
  - random bit mutations in the memory
  - ...

Hoeffding's inequality

Consider a (biased) coin that comes up heads with probability p. So, if we toss it n times, should get $p.n$ heads on average. Letting random variable $H(n)$ be number of heads seen after n coin tosses, it turns out that

$$\Pr[H(n) \leq (p-\varepsilon)n] \leq \exp(-2\varepsilon^2 n)$$

and similarly,

$$\Pr[H(n) \geq (p+\varepsilon)n] \leq \exp(-2\varepsilon^2 n)$$

Probability that we're off by a constant factor, is inverse-exponential in n. Often useful in analysing randomised algorithms!

Relationships to other complexity classes

Recall we noted that RP⊆NP.
(convert a randomised algorithm to a non-deterministic one by replacing coin flips with non-deterministic guesses.)

Doesn't work for BPP.

We do have BPP$\subseteq \Sigma_2^P \cap \Pi_2^P$ (Sipser-Gács-Lautemann theorem)
Consequently, if P=NP, it would follow that P=BPP since if P=NP, the polynomial hierarchy collapses to P.

We also know: BPP⊆P/poly (Adleman's theorem).
"Any BPP language has polynomial-size circuits."

Next: A randomised algorithms for reducing a (satisfiable) SAT instance to one having a unique solution

Then, a quick look at probabilistically checkable proofs