Computational Complexity; slides 8, HT 2019
A Brief Introduction to randomisation

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

HT 2019

Randomised Algorithms

Randomised algorithms have access to a stream of random bits.

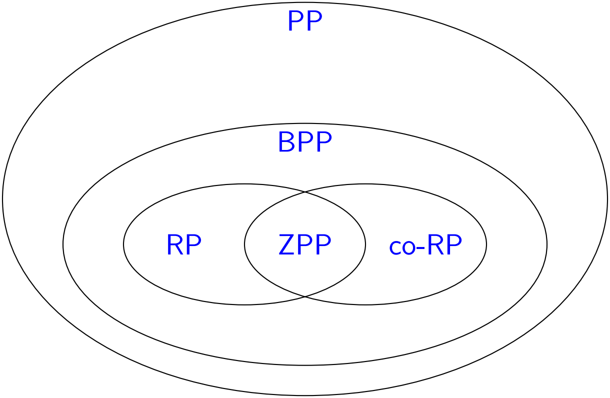
The running time and even the outcome may depend on random choices.

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$\rightarrow$"no", yes-instance$\rightarrow$"yes" with probability$\geq p$ (for some constant $p > 0$)
PP: "majority-P", contains NP, within PSPACE
BPP: allow error either way (constant $< \frac{1}{2}$)

Usage of randomised algorithms

In practice, not so much for language recognition, more for simulation, 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Miller-Rabin test for primality, subsequently superseded by 2002 AKS primality test (deterministic)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$\subseteq$NP: accepting computation of an RP machine is a certificate of yes-instance.

It's unknown whether BPP$\subseteq$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.
- 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 $\varepsilon < \frac{1}{2}$.

RP: one-sided error; no-instance$\rightarrow$"no", yes-instance$\rightarrow$"yes" with probability$\geq p$ (for some constant $p > 0$)

Useful? (even if, say, $p = 10^{-6}$ for some RP problem, or error probability is $\frac{1}{2} - 10^{-6}$ for some BPP problem?)

Probability amplification

BPP: problems that can be solved by a randomised algorithm

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

RP: one-sided error; no-instance→"no", yes-instance→"yes" with probability$\geq p$ (for some constant $p > 0$)

Useful? (even if, say, $p = 10^{-6}$ for some RP problem, or error probability is $\frac{1}{2} - 10^{-6}$ for some BPP problem?)

For problem X with RP algorithm having $p = 10^{-6}$, 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 $\frac{1}{2} - \varepsilon$: suppose we run it 3 times and take majority vote.

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

Lemma. 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 (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 $c \le i$ so that $\mathcal{B}$ gives the wrong answer.

The probability p_S for any bad sequence S to occur is

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

$S := a_1, \ldots, a_{2k}$: 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 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

$\alpha = -\log_2(4\varepsilon(1-\varepsilon))$ and choose $k \geq p(n)/\alpha$. □

General note

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

But still. Is this useful?

To design algorithms that may go wrong any sense?

General note

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

But still. Is this useful?

To design algorithms that may go wrong any sense?

Possible answers. One might argue that

- 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
  - ...

Also, it is certainly better to have an efficient algorithm that goes wrong every 2^{100} times than to have no algorithm.

Non-Determinism as Randomisation

Definition. Let $\mathrm{RP}(f(n))$ be the class of problems that can be solved by a randomised algorithm

- with polynomial worst-case running time such that
- every input that should be rejected is rejected with certainty and
- every input of length n that should be accepted is rejected with probability $\leq f(n)$.

Here, $f(n)$ is a function with $0 \leq f(n) < 1$ for all n.

Non-Determinism as Randomisation

Definition. Let $\mathrm{RP}(f(n))$ be the class of problems that can be solved by a randomised algorithm

- with polynomial worst-case running time such that
- every input that should be rejected is rejected with certainty and
- every input of length n that should be accepted is rejected with probability $\leq f(n)$.

Here, $f(n)$ is a function with $0 \leq f(n) < 1$ for all n.

Definition. A problem belongs to the class RP^* if it belongs to $\mathrm{RP}(f(n))$ for some function $f(n)$.

Non-Determinism as Randomisation

Definition. Let $\mathsf{RP}(f(n))$ be the class of problems that can be solved by a randomised algorithm

- with polynomial worst-case running time such that
- every input that should be rejected is rejected with certainty and
- every input of length n that should be accepted is rejected with probability $\leq f(n)$.

Here, $f(n)$ is a function with $0 \leq f(n) < 1$ for all n.

Definition. A problem belongs to the class RP^* if it belongs to $\mathsf{RP}(f(n))$ for some function $f(n)$.

Lemma. $\mathsf{NP} = \mathsf{RP}^*$.

but this would use inverse-exponential $f(n)$, e.g. $f(n) = 2^{-n}$.

Reducing SAT to USAT with the aid of randomness

USAT: given a formula φ with at most 1 satisfying assignment, determine whether it is satisfiable. (U stands for "unique")
We reduce SAT to USAT.

Motivation: known algorithms for SAT take time $poly(n)2^n$. The "exponential time hypothesis" asserts that you *need* time proportional to 2^n.
But: note Grover's algorithm, a quantum algorithm solving USAT in time $poly(n)2^{n/2}$. Reducing SAT to USAT means that on a quantum machine, SAT is also solved in time $poly(n)2^{n/2}$!

Reducing SAT to USAT with the aid of randomness

USAT: given a formula φ with at most 1 satisfying assignment, determine whether it is satisfiable. (U stands for "unique")
We reduce SAT to USAT.

Motivation: known algorithms for SAT take time $poly(n)2^n$. The "exponential time hypothesis" asserts that you *need* time proportional to 2^n.
But: note Grover's algorithm, a quantum algorithm solving USAT in time $poly(n)2^{n/2}$. Reducing SAT to USAT means that on a quantum machine, SAT is also solved in time $poly(n)2^{n/2}$!

Challenge: Given φ, construct ψ such that ψ has a unique satisfying assignment iff φ is satisfiable.

Reducing SAT to USAT with the aid of randomness

USAT: given a formula φ with at most 1 satisfying assignment, determine whether it is satisfiable. (U stands for "unique")
We reduce SAT to USAT.

Motivation: known algorithms for SAT take time $poly(n)2^n$. The "exponential time hypothesis" asserts that you *need* time proportional to 2^n.
But: note Grover's algorithm, a quantum algorithm solving USAT in time $poly(n)2^{n/2}$. Reducing SAT to USAT means that on a quantum machine, SAT is also solved in time $poly(n)2^{n/2}$!

Challenge: Given φ, construct ψ such that ψ has a unique satisfying assignment iff φ is satisfiable.

Idea: $\psi := \varphi \wedge \rho$, where ρ is some other formula over the same variables.

Reducing SAT to USAT

Challenge: Given φ, construct ψ such that ψ has a unique satisfying assignment iff φ is satisfiable.
Idea: $\psi := \varphi \wedge \rho$, where ρ is some other formula over the same variables.
Extension of the idea: $\psi_1 := \varphi \wedge \rho_1$, ... ,$\psi_k := \varphi \wedge \rho_k$; look for satisfying assignment of any of these...

Problem: Think of φ as having been chosen by an opponent. Given a choice of $\rho_1, \ldots, \rho_k$, he can pick a φ that fails for your choice. This is where randomness helps!

(random) parity functions: let $x_1, \ldots, x_n$ be the variables of φ. Let $\pi := \oplus_{x \in R}(x) \oplus b$ where each x_i is added to R with prob. $\frac{1}{2}$, and b is chosen to be TRUE/FALSE with equal probability $\frac{1}{2}$.

Think of R as standing for "relevant attributes"

Reducing SAT to USAT

Q: Why are random parity functions great?

A: Consider φ with set S of satisfying assignments. For random p.f. π, the expected number of satisfying assignments of $\varphi \wedge \pi$ is $\frac{1}{2}|S|$.

To see this, note that any satisfying assignment of φ gets eliminated with probability $\frac{1}{2}$.

Reducing SAT to USAT

Q: Why are random parity functions great?
A: Consider φ with set S of satisfying assignments. For random p.f. π, the expected number of satisfying assignments of $\varphi \wedge \pi$ is $\frac{1}{2}|S|$.

To see this, note that any satisfying assignment of φ gets eliminated with probability $\frac{1}{2}$.

Corollary: letting $\rho_k := \pi_1 \wedge \ldots \wedge \pi_k$ for independently randomly-chosen π_i, the expected number of satisfying assignments to $\varphi \wedge \rho_k$ is $|S|/2^k$.

Reducing SAT to USAT

Q: Why are random parity functions great?
A: Consider φ with set S of satisfying assignments. For random p.f. π, the expected number of satisfying assignments of $\varphi \wedge \pi$ is $\frac{1}{2}|S|$.

To see this, note that any satisfying assignment of φ gets eliminated with probability $\frac{1}{2}$.

Corollary: letting $\rho_k := \pi_1 \wedge \ldots \wedge \pi_k$ for independently randomly-chosen π_i, the expected number of satisfying assignments to $\varphi \wedge \rho_k$ is $|S|/2^k$.

This suggests the following approach:

- Generate ρ_k as above, for each $k = 1, 2, \ldots, n+1$.
- Search for a satisfying assignment to $\varphi \wedge \rho_k$.

Need to argue that for $k \approx \log_2 |S|$, we have reasonable chance of producing a formula with a *unique* s.a.

Pairwise independence of random p.f's:

Given $x \neq x' \in S$, and a random parity function π, we have:

$\Pr[x \text{ satisfies } \pi] = \frac{1}{2}$ $\Pr[x' \text{ satisfies } \pi] = \frac{1}{2}$

In addition:

$\Pr[x \text{ satisfies } \pi | x' \text{ satisfies } \pi] = \frac{1}{2}$

Proof:

For any x, $\pi(x) = v.x$ where v is characteristic vector of relevant attributes R of π.
($v.x$ denotes sum (XOR) of entries of x where corresponding entry of v is 1)
Let i be a bit position where $x'_i = 1$ and $x_i = 0$. i gets added to R with probability $\frac{1}{2}$, so value of $\pi(x')$ gets flipped with probability $\frac{1}{2}$.

Reducing SAT to USAT

For some k, we have $2^{k-2} \leq |S| \leq 2^{k-1}$.

Lemma: $\Pr$[there is unique $x \in S$ satisfying $\varphi \wedge \rho_k$] $\geq \frac{1}{8}$ (probability is w.r.t. random choice of ρ_k).

Proof: Let $p = 2^{-k}$ be the probability that $x \in S$ satisfies ρ_k.
Let N be the random variable consisting the number of s.a.'s of $\varphi \wedge \rho_k$.
$E[N] = |S|p \in [\frac{1}{4}, \frac{1}{2}]$.

$$\Pr[N \geq 1] \geq \sum_{x \in S} \Pr[x \models \rho_k] - \sum_{x < x' \in S} \Pr[x \models \rho_k \wedge x' \models \rho_k] = |S|p - \binom{|S|}{2} p^2$$

By pairwise independence and union bound, we have $\Pr[N \geq 2] \leq \binom{|S|}{2} p^2$. So

$$\Pr[N = 1] = \Pr[N \geq 1] - \Pr[N \geq 2] \geq |S|p - 2\binom{|S|}{2} p^2 \geq |S|p - |S|^2 p^2 \geq \frac{1}{8}.$$

(where the last inequality uses $\frac{1}{4} \leq |S|p \leq \frac{1}{2}$.)

Notes on BPP

- BPP$\subseteq \Sigma_2^P \cap \Pi_2^P$ (Sipser-Gács theorem)
- Class of problems having "useful" algorithms
- Not a "syntactic" complexity class: no obvious way to define a complete problem for BPP. (Similar point for RP: these are said to be "semantic" as opposed to "syntactic" classes.) P, NP, PSPACE, are syntactic. PP?

Next: TFNP (also not a syntactic class)