Algorithms and Computation

(grad course)

Lecture 10: Randomized Algorithms I

Instructor: Hossein Jowhari

jowhari@kntu.ac.ir

Department of Computer Science and Statistics
Faculty of Mathematics
K. N. Toosi University of Technology

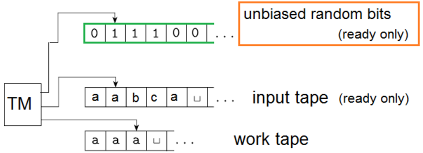
Fall 2021

Probabilistic Turing Machines

Randomized algorithms

A *probabilistic Turing machine* is a Turing machine augmented with the ability to generate an unbiased coin flip in one step. It corresponds to a randomized algorithm. On any input x, a probabilistic Turing machine accepts x with some probability, and we study this probability.

Randomized algorithms, Motwani and Raghavan



- Deterministic algorithm (correct answer always)
- Non-deterministic algorithm (wild guessing the answer)
- Randomized algorithm (correct answer most of the time)

Randomized algorithms are faster than their deterministic counterparts at the expense of making error in some trails.

Randomized complexity classes

- **RP** (Randomized Polynomial time)
- **coRP** (Complement of Randomized Polynomial time)
- **ZPP** (Zero-error Polynomial time)
- **PP** (Probabilistic Polynomial time)
- **BPP** (Bounded-error Probabilistic Polynomial time)

One-sided Error Algorithms

Definition: The class **RP** (Randomized Polynomial time) consists of all languages L that have a randomized algorithm A that runs in polynomial time such that for any input $x \in \Sigma^*$

- $x \in L \Rightarrow Pr[A(x) \text{ accepts}] \geq \frac{1}{2}$
- $x \notin L \Rightarrow Pr[A(x) \text{ accepts}] = 0$

Definition: The class **coRP** consists of all languages L that have a randomized algorithm A that runs in polynomial time such that for any input $x \in \Sigma^*$

- $x \in L \Rightarrow Pr[A(x) \text{ accepts}] = 1$
- $x \notin L \Rightarrow Pr[A(x) \text{ accepts}] \leq \frac{1}{2}$

The randomized algorithm A is called a **one-sided error algorithm**.

Example: Verifying Polynomial Identities

Input: Two polynomials $F(x)$ and $G(x)$ of degree d where $F(x)$ is given as a product and $G(x)$ is given in its canonical form.

Output: Accept if and only if $F(x) \equiv G(x)$

Example:

$$\underbrace{(x+1)(x-2)(x+3)(x-4)(x+5)(x-6)}_{\text{product form}} \stackrel{?}{\equiv} \underbrace{x^6 - 7x^3 + 25}_{\text{canonical form}}$$

Observation: The product form can be converted to the canonical form using $O(d^2)$ multiplications. (why?)

Verifying Polynomial Identity can be solved in $O(d^2)$ time.

A randomized algorithm for Verifying Polynomial Identity

- Suppose d is the maximum degree.
- Choose an integer r uniformly at random from the range $\{1, \ldots, 100d\}$
- Compute $F(r)$ and $G(r)$.
- Accept if $G(r) = F(r)$ otherwise reject.

Analysis:

- If $G(x) \equiv F(x)$, the algorithm always accepts.
- If $G(x) \not\equiv F(x)$, the algorithm may err and wrongly accept.

If $G(x) \not\equiv F(x)$ then $Q(x) = F(x) - G(x)$ is a non-zero polynomial of degree at most d.

The algorithm errs when accidentally it picks a root of $Q(x)$.

By the *fundamental theorem of algebra*, a polynomial of degree d has at most d roots. Therefore

$$Pr[\text{algorithm accepts when } F(x) \equiv G(X)] = 1$$

$$Pr[\text{algorithm accepts when } F(x) \not\equiv G(X)] \le \frac{d}{100d} \le \frac{1}{100}$$

Running time: $O(d)$ multiplications (when computing $F(r)$ and $G(r)$).

Here we have assumed picking a random integer from the desired range takes constant time.

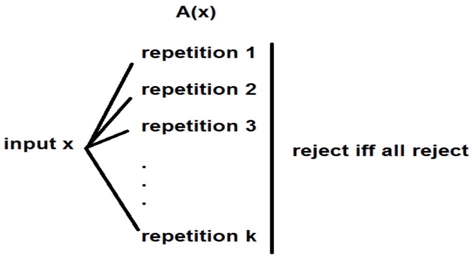
The constant $\frac{1}{100}$ in the randomized algorithm for Verifying Polynomial Identity is arbitrary. We can choose the random integer r from a larger rang, for example $\{1, \ldots, 1000d\}$, and bring down the error probability to $\frac{1}{1000}$. Note that as result the time we spend for picking the random number r and also computing $F(r)$ and $G(r)$ increases.

Repetition: boosting the success probability

Consider a one-sided randomized algorithm A for language L where

- $x \in L \Rightarrow Pr[A(x) \text{ accepts}] \geq \frac{1}{2}$
- $x \notin L \Rightarrow Pr[A(x) \text{ accepts}] = 0$

A common way of boosting the algorithm A is by repetition. We repeat A on input x a number of times (independently) and reject iff all the repetitions reject.



Let $A^{(k)}$ be the amplified algorithm with k repetitions. The algorithm $A^{(k)}(x)$ errs when $x \in L$ and all the k repetitions reject. This happens with probability at most $(1 - \frac{1}{2})^k$. Therefore

- $x \in L \Rightarrow Pr[A^{(k)}(x) \text{ accepts}] \geq 1 - (1 - \frac{1}{2})^k$
- $x \notin L \Rightarrow Pr[A^{(k)}(x) \text{ accepts}] = 0$

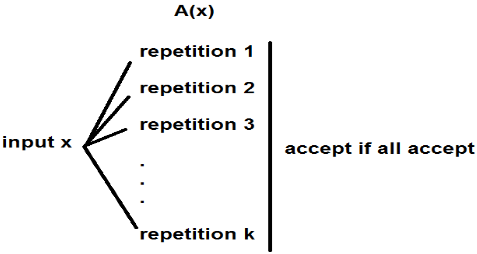
Independent Events: If $E_1, \ldots, E_t$ are independent events then

$$Pr[\underbrace{E_1 \wedge \ldots \wedge E_t}_{\text{all happen}}] = Pr[E_1] \times \ldots \times Pr[E_t]$$

Similarly consider a one-sided randomized algorithm A for language L where

- $x \in L \Rightarrow Pr[A(x) \text{ accepts}] = 1$
- $x \notin L \Rightarrow Pr[A(x) \text{ accepts}] \leq \frac{1}{2}$

We repeat A on input x a number of times (independently) and accept iff all the repetitions accept.



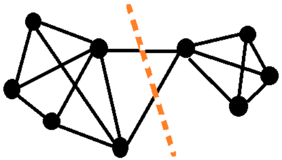
Let $A^{(k)}$ be the amplified algorithm with k repetitions. The algorithm $A^{(k)}(x)$ errs only when $x \notin L$ and all the k repetitions accept. This happens with probability at most $(\frac{1}{2})^k$. Therefore

- $x \in L \Rightarrow Pr[A^{(k)}(x) \text{ accepts}] = 1$
- $x \notin L \Rightarrow Pr[A^{(k)}(x) \text{ accepts}] \leq (\frac{1}{2})^k$

A faster algorithm for Min-Cut

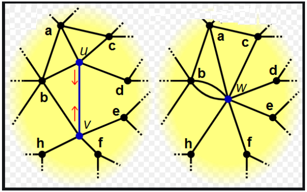
Input: Undirected graph $G = (V, E)$ on n nodes with m edges.

Problem: Is there a cut of G of size $\leq k$? In other words, can we disconnect G by removing at most k edges.



Fact: The minimum cut can be found deterministically by using a maximum flow algorithm. The running time is at least $O(nf(n))$ when $f(n)$ is the running time of the maximum flow algorithm.

An algorithm based on contracting edges

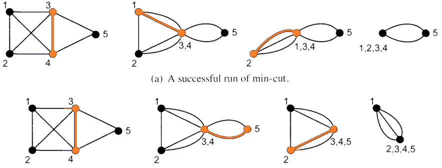


Contracting the edge $e = (u, v)$ in undirected graph G results in the undirected graph G' where

- The edge e is removed.
- The vertices u and v are joined to form a new vertex w in G'.
- All the edges on u and v (except e) are placed on the vertex w.

Consider the following randomized strategy (due to David Karger) for finding a minimum cut of graph G.

- The algorithm consists of $n - 2$ iterations.
- In each iteration, the algorithm picks an edge from the existing edges in the graph and contracts that edge.
- There are many possible ways one could choose the edge at each step. The randomized algorithm chooses the edge uniformly at random from the remaining edges.



(a) A successful run of min-cut.

(b) An unsuccessful run of min-cut.

Theorem: The algorithm arrives at a minimum cut with probability at least $\frac{2}{n(n-1)}$.

Proof:

- The graph G may have several min-cuts. We fix a min-cut $(S, V/S)$ and calculate the probability that the algorithm arrives at $(S, V/S)$.
- Let C be the edges crossing the cut $(S, V/S)$.
- If the algorithm never chooses an edge in C the algorithm at the end arrives at the cut $(S, V/S)$.
- (**Intuition**) If C is small the probability that the algorithm never picks an edge in C is large enough.

E_1 $\quad E_2$ $\quad \ldots$ $\quad E_{n-2}$

$e_1 \notin C$ $\quad e_2 \notin C$ $\quad \ldots$ $\quad e_{n-2} \notin C$

- Let E_i be the event that the edge contracted in the i-th iteration is not in C.
- Let $F_i = \cap_{j=1}^{i} E_j$ be the event that no edge of C is contracted in the first i iterations.
- We want to lower bound $Pr[F_{n-2}]$.

(**Base case**) $Pr[F_1] = Pr[E_1]$. What is the probability that the first edge is not in C?

Assuming $|C| = k$, all vertices in the graph has degree at least k. Therefore there are at least $\frac{nk}{2}$ edges in the graph.

$$Pr[E_1] \geq 1 - \frac{k}{nk/2} = 1 - \frac{2}{n}$$

- If E_1 happens the resulting graph has $n-1$ nodes with min-cut size k.
- $Pr[E_2 \mid F_1] \geq 1 - \frac{k}{(n-1)k/2} \geq 1 - \frac{2}{n-1}$
- $Pr[E_i \mid F_{i-1}] \geq 1 - \frac{k}{(n-i+1)k/2} \geq 1 - \frac{2}{n-i+1}$
- $Pr[F_{n-2}] = Pr[E_{n-2} \cap F_{n-3}] = Pr[E_{n-2} \mid F_{n-3}]Pr[F_{n-3}]$
- $= Pr[E_{n-2} \mid F_{n-3}]Pr[E_{n-3} \mid F_{n-4}] \ldots Pr[E_2 \mid F_1]Pr[F_1]$
- $\geq \prod_{i=1}^{n-2}(1 - \frac{2}{n-i+1}) = \prod_{i=1}^{n-2}(\frac{n-i-1}{n-i+1})$
- $\geq (\frac{n-2}{n})(\frac{n-3}{n-1})(\frac{n-4}{n-2}) \ldots (\frac{3}{5})(\frac{2}{4})(\frac{1}{3})$
- $= \frac{2}{n(n-1)}$

- The algorithm is one-sided error.
- If G has min-cut size $> k$ then the algorithm never accepts.
- If G has min-cut size $\leq k$ then the algorithm accepts with probability $\frac{2}{n(n-1)}$.
- If we repeat the algorithm $t = 10 \times \frac{n(n-1)}{2}$ times the probability of error would be at most

$$(1 - \frac{2}{n(n-1)})^t = (1 - \frac{2}{n(n-1)})^{\frac{n(n-1)}{2} \times 10} \leq e^{-10}$$

- Running time of the algorithm ?