CSC 5450 Randomness and Computation

Week 1: Introduction

Today's plan

① course overview

② conditional probability: minimum cut, string equality

③ linearity of expection: quicksort, max k-SAT, Ramsey graphs

---

Course Information

Objective: to use probabilistic tools in solving problems.

Course content: ① learn different tools and see their uses (about half the course).

② see applications in different areas (about half the course).

Course homepages: http://www.cse.cuhk.edu.hk/~chi/csc5450

http://www.cse.cuhk.edu.hk/~chi/csc5450-2011 (old course notes)

Course requirements: ① homework 50% ~3 assignments

② course project 50%

Course project: to study one specific topic in depth

- see some suggested topics in course homepage
- encouraged to be close to your research
- mathematical and theoretical
- survey report
- no presentation, no programming
- one person for graduate students, two persons for undergraduate students.

Schedule: 14 weeks, 3-4 hours per week (try to make last hour optional).

Prerequisties: working knowledge in probability (important). Try homework 0.

feel comfortable in data structures and algorithms (optional).

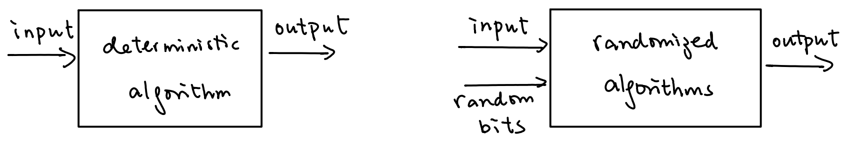
I will try to make the notes as self-contained as possible.

References: Mitzenmacher, Upfal. Probability and Computing **[MU]**

Motwani, Raghavan. Randomized Algorithms. [MR]

Randomized Algorithms

Informally, randomized algorithms are algorithms that flip coins.



In deterministic algorithm, given an input, it outputs the unique output all the time.

In randomized algorithms, the output is not only a function of the input, but also a function of the random bits. So the output is a random variable. Also the running time is a random variable.

Why would flipping coins help?

The main difference is that we allow the algorithm to make errors sometimes, and only requires that the output is correct with high probability. Or, we allow the randomized algorithm to be slow sometimes, but it is fast with high probability. Think of high probability as 99.99999999 %.

Surprisingly this tradeoff could bring us a significant improvement in the running time, and sometimes could do something impossible for deterministic algorithms.

- Faster algorithms : minimum spanning tree (almost linear to linear)
  minimum cut ($O(|V||E|)$ time to almost linear $\tilde{O}(|E|)$)
  primality testing ($O(\log^6 N)$ to $O(\log^2 N)$ where N is the number)
  approximate counting (exponential to polynomial)

- Simpler algorithms : almost always the randomized algorithms are (significantly) simpler than the the deterministic algorithms (e.g. MST, primality testing, 3SAT, etc).

- Save communications : In parallel and distributed computing, much communications are required to get the correct answers. On the other hand, if we just do things randomly, then almost surely we won't be in a bad situation.
  e.g. packet routing in distributed network to avoid congestion,
  random network coding in distributed manner,
  efficient parallel algorithm for finding a maximum matching.

- Beating adversary : For deterministic algorithms, there could be an adversary who can predict your action, and gives the worst scenario for you, e.g. online algorithms, sublinear algorithms, and many things could be proved to be impossible to do. But if we assume that we have some private randomness that the adversary doesn't know, then the power of the adversary to manipulate is much weaker, e.g. online paging algorithms, data streaming, etc.

- Probabilistic methods: Sometimes you could prove that some crazy objects exist by using a probabilistic argument, while deterministic arguments are much harder or not known (e.g. good error correcting codes, expander graphs, etc).

Let me emphasize that randomized algorithms could do much better because we have weaker requirements (e.g. to allow errors), or we have stronger assumptions (e.g. private randomness). But in almost all scenarios this tradeoff is more preferable.

Overview

The following is a tentative plan for the remaining lectures; some are undecided yet.

(2) moments and derivations: Markov and Chebyshev inequalities, balls and bins, power of 2 choices

(3) tail inequalities: Chernoff bounds, packet routing, congestion minimization, graph sparsification (negative correlations)

(4) probabilistic method: first moment method, expander graphs, superconcentrator entropy, randomness extraction, compression, Shannon's coding theorem

(5) probabilistic method: second moment method, threshold phenomenon in random graphs local lemma, K-SAT, algorithmic aspects

(6) random walk: 2-SAT & 3-SAT, stationary distribution, pagerank, electrical network convergence rate, spectral gap, conductance.

(7) Markov chain Monte Carlo: DNF counting, network reliability, approximate sampling & approximate counting, coupling, shuffling cards, random spanning trees.

(8) algebraic techniques: polynomial identity testing, Schwarz-Zippel lemma, determinant, bipartite matching, network coding, matrix rank

(9) hashing: pairwise independence, universal hash families, perfect hashing, k-wise and almost k-wise independence.

(10) sublinear algorithms: data streaming, distinct elements, frequency moments, heavy hitters sublinear time: property testing, constant time algorithms

There are four more weeks. The following is a list of possible topics.

(a) geometry: dimension reduction, computational geometry, linear programming

(b) graphs: minimum spanning tree, all pairs shortest path, color coding, graph sparsification

(c) online algorithms: paging, ski-rental, secretary problem, minimum Steiner tree

(d) number theory: primality testing, and something else...

(e) concentration inequalities: martingales, bounded differences, isoperimetric inequality

(f) complexity: derandomization, pseudorandom generator, lower bound, probabilistic checkable proofs

(g) randomized rounding: linear programming, semidefinite programming, max cut

(h) web graphs: power law, preferential attachment, small world graphs

I will give you a form to indicate your preferences, and I'm also open to other suggestions.

Quick basic probability review (references: [1], MU 1, MR appendix B & C)

Sample space Ω: set of all possible outcomes, each with a "probability" associated with it.

Event E: subset of outcomes. $\Pr(E)$ = sum of the probabilities of its outcomes

Axioms: ① $0 \le \Pr(E) \le 1$ ② $\Pr(\Omega) = 1$ ③ $\Pr(\bigcup_i E_i) = \sum_i \Pr(E_i)$ for disjoint E_i.

Union bound: $\Pr(\bigcup_i E_i) \le \sum_i \Pr(E_i)$

Inclusion-exclusion principle: $\Pr(\bigcup_i E_i) = \sum_i \Pr(E_i) - \sum_{i,j} \Pr(E_i \cap E_j) + \sum_{i,j,k} \Pr(E_i \cap E_j \cap E_k) - \dots$
$\uparrow$
(see homework 0)
$+ (-1)^{l+1} \sum_{1 < i_2 < \dots < i_l} \Pr(\bigcap_{r=1}^{l} E_{i_r}) \dots$

Conditional probability: $\Pr(E|F) = \Pr(E \cap F) / \Pr(F)$

Independence: $\Pr(\cap E_i) = \prod_i \Pr(E_i)$

Total probability: Let E_i be disjoint and $\bigcup_i E_i = \Omega$.

Then $\Pr(B) = \sum_i \Pr(B \cap E_i) = \sum_i \Pr(B|E_i)\Pr(E_i)$

Baye's law: Let E_i be disjoint and $\bigcup_i E_i = \Omega$.

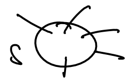
Then $\Pr(E_j|B) = \dfrac{\Pr(B|E_j)\Pr(E_j)}{\sum_i \Pr(B|E_i)\Pr(E_i)}$ (homework 0)

Minimum Cut (MU 1.4)

Given an undirected graph $G = (V,E)$, find a subset of edges F of minimum cardinality so that $G' = (V, E-F)$ is disconnected.

Notation: Given $S \subseteq V$, let $\delta(S)$ be the set of edges with one endpoint in S and the other

Notation: Given $S \subseteq V$, let $\delta(S)$ be the set of edges with one endpoint in S and another endpoint in $V-S$.



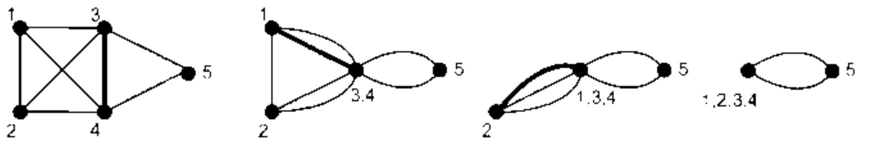
The minimum cut problem can be equivalently stated as finding $S \subseteq V$ that minimizes $|\delta(S)|$.

Deterministic algorithms:	naive : $\binom{n}{2}$ max flow $\Rightarrow$	$O(n^3 m)$ time
	less naive : $n-1$ max flow $\Rightarrow$	$O(n^2 m)$ time
	not naive : one max flow $\Rightarrow$	$O(mn)$ time

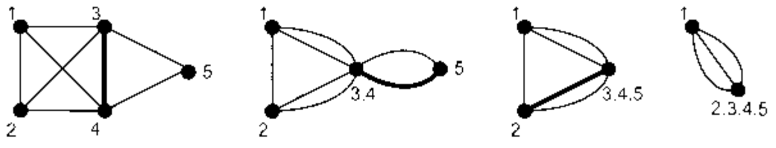
Randomized algorithms :	simple : $O(n^4)$	
	less simple : $\tilde{O}(n^2)$	(in homework 1 ?)
	not simple : $\tilde{O}(m)$	[Karger 2000]

Algorithm

```
Repeat
        pick a random edge e
        contract the two endpoints of e
until only 2 vertices left
output the edges between these 2 vertices
```



(a) A successful run of min-cut.



(b) An unsuccessful run of min-cut.

(illustration from MU)

Observation: Each node in an intermediate graph is a subset of nodes in the original graph.

So, each cut in an intermediate graph corresponds to a cut in the original graph.

Therefore, $\overset{\text{the size of}}{\vee}$ a minimum cut in an intermediate graph is at least the size of a minimum cut in the original graph.

Theorem The algorithm outputs a minimum cut with probability at least $2/n(n-1)$.

Proof Let k be the size of a minimum cut.

Then each vertex is of degree at least k, otherwise there is a singleton cut of size $<k$.

The the graph has at least $kn/2$ edges.

Consider a minimum cut F (a subset of k edges).

If the edges in F survive to the end (i.e. we don't contract them), then we win.

Let E_i be the event that F survives in the i-th iteration.

$Pr(E_1) \geq 1 - (k)/(kn/2) = 1 - \frac{2}{n}$.

In the i-th iteration, since min-cut $\geq k$ by the observation, there are at least $(n-i+1)k/2$ edges.

In the i-th iteration, since min-cut $\geq k$ by the observation, there are at least $(n-i+1)k/2$ edges.

So $\Pr(E_i \mid E_1 \cap E_2 \cap \dots \cap E_{i-1}) \geq 1 - \frac{2}{n-i+1}$.

Hence, $\Pr(\text{we win}) = \Pr(E_1 \cap \dots \cap E_{n-2})$

$$= \Pr(E_{n-2} \mid E_1 \cap \dots \cap E_{n-3}) \cdot \dots \cdot \Pr(E_2 \mid E_1) \cdot \Pr(E_1)$$

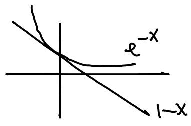
$$= (1-\tfrac{2}{3}) \cdot (1-\tfrac{2}{4}) \cdot \dots \cdot (1-\tfrac{2}{n-1}) \cdot (1-\tfrac{2}{n})$$

$$= \frac{1}{3} \cdot \frac{2}{4} \cdot \frac{3}{5} \cdot \dots \cdot \frac{n-3}{n-1} \cdot \frac{n-2}{n} = \frac{2}{n(n-1)}. \quad \square$$

Improving success probability. A simple way is to repeat the whole procedure many times.

Suppose we repeat for k times. Then the failure probability is at most $\left(1 - \frac{2}{n(n-1)}\right)^k$

Recall that $1 - x \leq e^{-x}$,



which can be derived from Taylor expansion (good to review some basic calculus, e.g. Stirling's approx)

Therefore, $\left(1 - \frac{2}{n(n-1)}\right)^k \leq e^{-\frac{2k}{n(n-1)}}$. So, if we set $k = 10n(n-1)$, then failure probability $\leq e^{-20}$.

Running time: One iteration can be implemented in $O(n^2)$ time, so total running time $= O(n^4)$.

Number of minimum cuts: A very interesting consequence is that an undirected graph has at most $n(n-1)/2$ different minimum cuts. This is because the surviving probability of one minimum cut is at least $2/n(n-1)$, and the events that two different cuts survive are disjoint. This statement is non-trivial to prove directly. Try it.

Minimum k-cut: This approach can be generalized to the minimum k-cut problem, where the objective is to remove the minimum number of edges to create k components. (homework 1?) Again it is not easy to solve the problem by a deterministic algorithm.

String Equality [MR 7.4] This is a striking use of randomness in computation.

Suppose a company maintains multiple copies of the same huge data set.

From time to time they want to check that the copies are still the same.

Think of a data set is an n-bit string, for some very large n.

A naive way is to transmit all the n-bits from one server to another server.

Actually, one can prove that no deterministic algorithms could do better than that.

But a randomized algorithm could do much better, in $O(\log n)$ bits!

Think of the n-bits in server A to be $a_1 a_2 \dots a_n$ and in server B to be $b_1 b_2 \dots b_n$.

One way is to consider the polynomials $A(x) = \sum_{i=1}^{n} a_i x_i$ and $B(x) = \sum_{i=1}^{n} b_i x_i$.

One way is to consider the polynomials $A(x) = \sum_{i=1}^{n} a_i x_i$ and $B(x) = \sum_{i=1}^{n} b_i x_i$.

Choose a large enough finite field $\mathbb{F}$ (e.g. modular arithmetic over prime p).

Pick a random element $r \in \mathbb{F}$, evaluate $A(r)$, send r and $A(r)$ to server B.

Server B checks whether $A(r) = B(r)$. If so, then "consistent"; otherwise "inconsistent".

That's the algorithm. How to analyze the success probability?

If the two strings are the same, then it will always be "consistent". So, if the algorithm says "inconsistent", it will never a mistake (okay, assuming no transmission error).

But the algorithm may make mistake when the two strings are not the same and it says "consistent". We would like to upper bound this error probability.

If the two strings are not the same, then $A(x) \not\equiv B(x)$ but we have chosen r s.t. $A(r)=B(r)$.

What is this error probability? It only happens when r is a root of $(A-B)(x)$.

Since $A(x)$ and $B(x)$ are polynomials of degree n, so is $(A-B)(x)$.

There are at most n roots of a degree n polynomial.

So, the error probability is at most $n/|\mathbb{F}|$.

If we choose $|\mathbb{F}| = 1000n$, then this is at most 0.001.

And the algorithm only needs to send $O(\log n)$ bits!

If we like, we can also set $|\mathbb{F}| = 1000n^2$, then error probability $\leq 0.001/n$ and still $O(\log n)$ bits.

This is an example of algebraic algorithms, hashing, sublinear algorithm.

Random variables and expectations [MU 2.1]

Random variable: X is a function from $\Omega \to \mathbb{R}$. $Pr(X=a) = \sum_{s\in\Omega : X(s)=a} Pr(s)$.

Independence: X and Y are independent if and only if $Pr(X=x \cap Y=y) = Pr(X=x) \cdot Pr(Y=y)$.

Expectation: $E[X] = \sum_i i Pr(X=i)$.

Linearity of expectation: $E[\sum_i X_i] = \sum_i E[X_i]$.

It is important to note that this holds even for dependent variables, e.g $E[X_i]+E[X_i^2]=E[X_i+X_i^2]$.

Conditional expectation: $E[Y|Z=z] = \sum_y y\, Pr(Y=y|Z=z)$

$E[Y|Z]$ is a random variable of Z that takes on the value $E[Y|Z=z]$ if $Z=z$.

Please review some basic random variables such as binomial and geometric random variables [MU 2.2-2.4].

Randomized Quicksort [MU 2.5]

Task: Given n distinct numbers $x_1, x_2, \dots, x_n$, output the numbers in sorted order.

Algorithm: ① Pick a random number x.

② Construct the sets $S_{<x}$ and $S_{>x}$.

③ Return $\{$ quicksort $(S_{<x})$, x, quicksort $(S_{>x})\}$.

Intuition: Most of the time, we break the sequence "evenly" (not necessarily $(n/2, n/2)$, enough to have $(n/10, 9n/10)$ most of the time).

Then the recursion depth is $O(\log n)$, and the total running time is $O(n \log n)$.

It can be analyzed rigorously using a probabilistic recurrence (e.g. see [MR 1.4]).

Linearity of expectation There is a simple and elegant analysis using linearity of expectation.

The key point is to define the random variables cleverly.

Let $y_1, y_2, \dots, y_n$ be the sorted sequence.

Let X_{ij} be the indicator variable such that $X_{ij} = \begin{cases} +1 & \text{if } y_i \text{ and } y_j \text{ are compared.} \\ 0 & \text{otherwise.} \end{cases}$

Let X be the total number of comparisons, i.e. the total running time of the algorithm.

Then $E[X] = E[\sum_{i<j} X_{ij}] = \sum_{i<j} E[X_{ij}]$ by linearity of expectation

$= \sum_{i<j} \Pr[y_i \text{ and } y_j \text{ are compared}]$

Note that y_i and y_j are compared iff y_i or y_j is selected as the random number, and $y_{i+1}, \dots, y_{j-1}$ are not selected yet.

Since each number is selected with equal probability, this happens with $2/(j-i+1)$.

So, $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2/(j-i+1) \leq 2n \log n$.

Monte Carlo, Las Vegas, RP, BPP [MR 1.2]

Monte Carlo: may make mistakes

It is important to note that the error is independent on the input, but just dependent on the random bits, i.e. works on every input, e.g. minimum cut.

Las Vegas: no mistakes, but running time is a random variable.

Note that a Monte Carlo algorithm can be changed to a Las Vegas algorithm if there is a quick algorithm to check whether the answer is correct. It is often not easy to have such a quick checking algorithm. A known example is a linear time minimum spanning tree verification algorithm (deterministic, very nontrivial), and this can turn a Monte Carlo linear time MST algorithm into Las Vegas.

RP (randomized polynomial time): one sided error

YES-instance $\Rightarrow$ Pr(algorithm says YES) $\geq \frac{1}{2}$

No-instance $\Rightarrow$ Pr(algorithm says YES) $= 0$

BPP (Bounded-error probabilistic polynomial time): two sided error

YES-instance $\Rightarrow$ Pr(algorithm says YES) $\geq \frac{1}{2} + \varepsilon$

No-instance $\Rightarrow$ Pr(algorithm says YES) $\leq \frac{1}{2} - \varepsilon$.

Can take majority to boost the success probability.

<u>Conjecture</u>: P = BPP. That is, every randomized algorithm can be derandomized.

<u>Probabilistic Methods</u> [MU 6.1, 6.2.2]

This is a powerful method to prove that something exists by a probabilistic argument.

<u>MAX k-SAT</u> Given a formula which is a conjunction of clauses, each is a disjunction of k variables.

e.g. $(x_1 \vee x_2 \vee x_3 \ldots \vee x_k) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_4 \vee \ldots \vee \bar{x}_{k+2}) \wedge \ldots$

This is a classical NP-hard problem.

Suppose we just want to satisfy many clauses.

It is easy to argue that there is an assignment that satisfies $(1-2^{-k})$ fraction of clauses.

Consider a random assignment, i.e. each variable is set to true/false with prob. ½.

Let X_i be the indicator variable where the i-th clause is satisfied.

Let X be the total number of clauses satisfied.

Then $E[X] = E[\Sigma X_i] = \Sigma E[X_i] = m(1-2^{-k})$, where m denotes the total number of clauses, and the last equality holds because there is only one out of 2^k assignments that violates the clause.

Since the expected value is $m(1-2^{-k})$, there exists one specific assignment that satisfies that many clauses. This is called the <u>first moment method.</u>

Finding such an assignment deterministically is not entirely trivial. Try it.

Ramsey Graphs

Can you color the edges of a complete graph of n vertices by two colors (say red and blue) so that there are no large monochromatic complete subgraphs?

This is one of the most famous problems in discrete mathematics, and is one of the early problems that the probabilistic method is developed.

Theorem If $\binom{n}{k} 2^{-\binom{k}{2}+1} < 1$, then it is possible to color the edges of K_n so that there are no monochromatic K_k.

Proof Consider a subset of k vertices S.

The probability that it is monochromatic is $2 \cdot 2^{-\binom{k}{2}}$.

There are $\binom{n}{k}$ subsets of size k.

So, by the union bound, if $\underset{\uparrow}{}\binom{n}{k} \cdot 2^{-\binom{k}{2}+1} < 1$, there is at least one coloring with no monochromatic K_k.

(the bad events don't cover the sample space)

$\square$

Note that the theorem is satisfied if $k = \Omega(\log_2 n)$ (check).

So, there is a coloring with no monochromatic $K_{\Omega(\log n)}$.

In fact, a random coloring will work with high probability.

Surprisingly, there are no deterministic method known to construct such a coloring.

In fact, it is already extremely difficult to get $k = \Theta(\sqrt{n})$ for a deterministic construction!

References

[1] Ross. A First Course in Probability.