CS711008Z Algorithm Design and Analysis

Lecture 12. Randomized algorithm: a brief introduction [1]

Dongbo Bu

Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China

[1] The slides were made based on Chapter 13 of Algorithm design, Randomized Algorithm by R. Motwani and P. Raghavan.

Outline

- Introduction and nine categories proposed by R. Karp;
- The first example: GLOBALMINCUT problem;
- Randomized algorithm in protocol design for distributed system;
- Randomization in approximation algorithm: LP+Random Rounding;
- Randomization coupled with divide-and-conquorer;
- Hashing

Why randomized algorithm? Simplicity and speed. For many applications, a randomized algorithm is the simplest algorithm available, the fastest, or both.

A brief introduction

How to deal with hard problems? Trade-off "quality" and "time"

We have a couple of options:

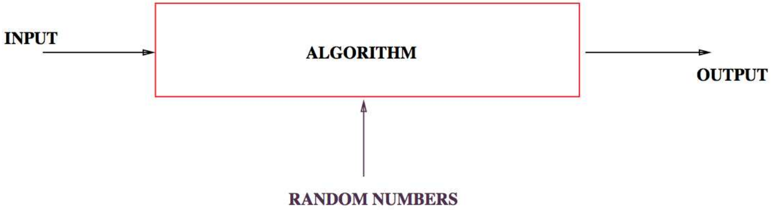
1. Give up polynomial-time restriction: hope that our algorithms run fast on the practical instances. (e.g. branch-and-bound, branch-and-cut, and branch-and-pricing algorithms are used to solve a TSP instance with over 24978 Swedish Cities. See http://www.tsp.gatech.edu/history/pictorial/sw24978.html)
2. Give up optimum restriction: from "optimal" solution to "nearly optimal" solution in the hope that "nearly optimal" is easy to find. e.g., approximation algorithm (with theoretical guarantee), heuristics, local search (without theoretical guarantee);
3. Give up deterministic restriction: the expectation of running time of a randomized algorithm might be polynomial;
4. Give up worst-case restriction: algorithm might be fast on special and limited cases;

Deterministic algorithm

- Goal: To prove that the algorithm solves the problem correctly (always) and quickly (typically, the number of steps should be polynomial in the size of the input)

(Excerpted from slides by P. Raghavan)

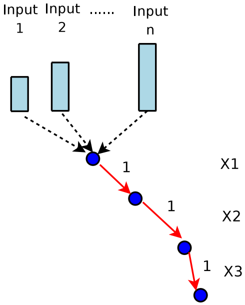
Randomized algorithm

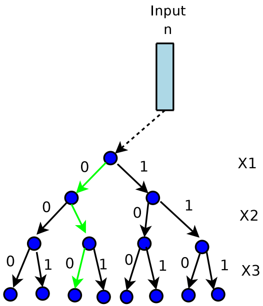


- In addition to input, algorithm takes a source of random numbers and makes random choices during execution.
- Behavior can vary even on a fixed input

Two views of randomness in the context of computation

1. The world is random: our algorithm is a deterministic algorithm that confront randomly generated input, and we can study the behavior of an algorithm on an "average" input rather than the worst-case input.
2. The algorithm is random: the world provides the same worst-case input as always; however, we allow our algorithm to make random decisions during execution.





Two types of randomzied algorithms

1. Las Vegas: always correct. Analyze its expected running-time.
2. Monte Carlo: correctness depends on the random choice. Analyze its error probability.

Note: still for worst-case input. ($\max_{Instance}$ expected time, or $\max_{Instance} \Pr[error]$);

Paradigms for randomized algorithms

Paradigms for randomized algorithm (by R. Karp) I

A handful of general principles lies at the heart of almost all randomized algorithms, despite the multitude of areas in which they find application.

1. Foiling an adversary: The classical adversary argument for a deterministic algorithm establishesa lower bound on the running time by constructing an input on which the algo fares poorly. While the adversary may be able to construct an input to foil one deterministic algo, it is difficult to devise a single input that is likely to defeat a randomized algo. (online algo, efficient proof verification)
2. Random sampling: a random sample from a population is representative of the whole population;

Paradigms for randomized algorithm (by R. Karp) II

3. Abundance of witnesses: To find a witness of a property of an input (say, “x is prime”). If the witness lies at a search space that is too large to search exhaustively, it suffices to randomly choose an element if the space contains a large number of witnesses.
4. Fingerprinting and hashing: to represent a long string by a short fingerprint using a random mapping. If two fingerprints are identical, the two strings are likely to be identical.
5. Random re-ordering input: After the re-ordering step, the input is unlikely to be in one of the orderings that is pathological for the naive algorithm;

Paradigms for randomized algorithm (by R. Karp) III

6. Rapidly mixing Markov chains: To count the number of combinatorial objects, we can randomly sample an appropriately defined population, which in turn relies on the information of the number. Solution: defining a Markov chain on the elements, and showing a short random walk is likely to sample the population uniformly.
7. Probabilistic methods and existence proofs: To establish that an object with cetain property exists by arguing that a randomly chosen object has the property with positive probability.
8. Load balancing: To spread load evenly among the resources in a paralell or distributed environment, where resource utilization decisions have to be made locally without reference to the global impact of these decisions. To reduce the amount of explicit communication or synchronization.

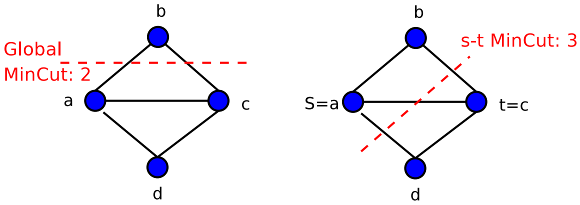
9. Isolation and symmetry breaking: In parallel environment, it is usually important to require a set of processors to find the same solution (consensus): choosing a random ordering on the feasible solutions, and tehn requiring the processors to find the solution with the lowest rank.

The first example: GLOBALMINCUT problem

INPUT:

A graph $G =< V, E >$

OUTPUT: a cut $c =< A, B >$ such that the size of c is minimized. Here, A, B are non-empty vertex sets and $V = A \cup B$.



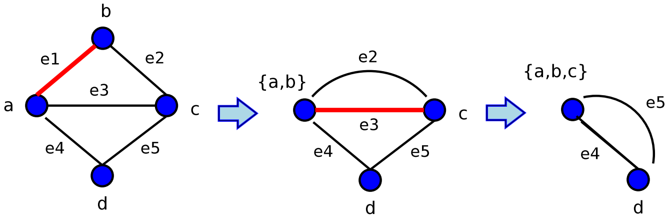
Note: The difference from the $s - t$ *cut* problem, where two vertex s and t are given, and we restrict the cut: $s \in A$, and $t \in B$.

A deterministic trial

- Basic idea: Transfering undirected graph G to a directed graph G' by replacing an edge $e = (u, v)$ with $e' = (u, v)$ and $e'' = (v, u)$, each of capacity 1. Let s be an arbitrary node. For $t = 2$ to n, call maximum-flow algorithm to calculate the minimum $s - t$ cut, and report the minimal one.
- Intuition: If (A, B) is a minimum $s - t$ cut in G', (A, B) is also a minimum cut among all those that separates s from t. We need a cut to seperate s from something.
- Time-complexity: $O(n^4)$.
- Note: Gloabl minimum cut can be found as efficiently as $s - t$ cut by techniques that didn't requires augmentation-path or a notion of flow.

Randomized algorithm (D. Karger, '92)

Advantage: qualitatively simpler than all previous algorithms.

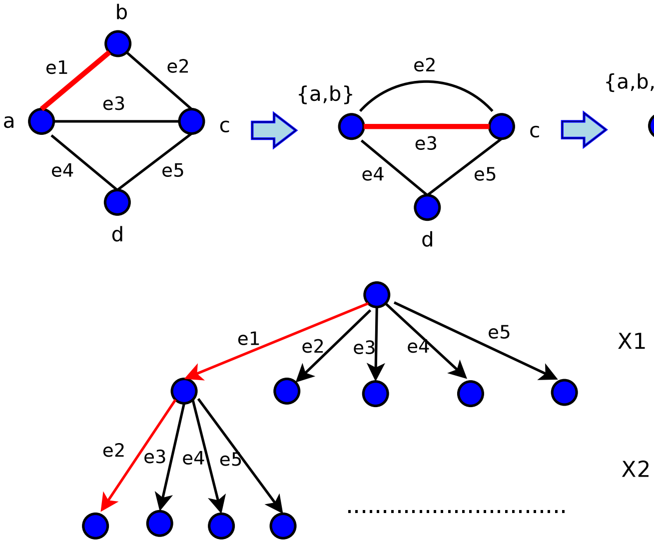


Contraction Algo(G)
Initially, S(v) = {v};
//S(v) to recorder nodes that have been contracted to v;

If G have two nodes v1 and v2
then return S(v1) and S(v2);

Choose an edge e=<u,v> uniformly at random;
Contract G to G', where a new node Zuv replace u and v;
Remove all edge between u and v;
S(Zuv) = S(u) + S(v);
Contraction(G');

A Las Vegas algorithm for GLOBALMINCUT problem



Theorem

The contraction algorithm returns a global min-cut with probabilty at least $\frac{1}{C_n^2}$.

Note: a bit counter-intuitive since there are exponentially many possible cuts of G, and thus the probabilty seems to be exponentially small.

Proof:

- Suppose (A, B) is a global min-cut with k edges. We want a lower bound of the probability that Contraction algo returns (A, B).
- Complement: failure due to an edge $e = (u, v)$, $u \in A$, $v \in B$ is selected for contracting;
- Let F_i be the event that an edge e in the cut is selected at the i-th iteration. We have:

- (The 1-st iteration) $Pr[F_1] = \frac{k}{|E|} \leq \frac{k}{(1/2)kn} = \frac{2}{n}$.
  (Reason: Each node v has a degree at least k. Otherwise, the cut $(v, V - v)$ has a size less than k. Thus, the edge number is at least $(1/2)kn$.)
- (The j-th iteration) $Pr[F_j|\overline{F_{j-1}}...\overline{F_1}] \leq \frac{2}{n-j}$.
  (Reason: same argument to G', where only $n - j$ supernodes are left.)

$$
\begin{aligned}
& Pr[\overline{F_{n-2}} \cap ... \cap \overline{F_1}] & (1)\\
= \; & Pr[\overline{F_1}]Pr[\overline{F_2}|\overline{F_1}]...Pr[\overline{F_{n-2}}|\overline{F_{n-3}} \cap ... \cap \overline{F_1}] & (2)\\
\geq \; & (1 - \frac{2}{n})(1 - \frac{2}{n-1})...(1 - \frac{2}{3}) & (3)\\
= \; & \frac{n-2}{n}\frac{n-3}{n-1}\frac{n-4}{n-2}...\frac{1}{3} & (4)\\
= \; & \frac{2}{n(n-1)} & (5)
\end{aligned}
$$

Further reduce failure probability via repeating

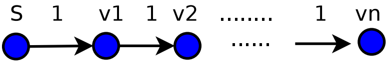
Basic idea: running Contraction algo r times will increase the probabilty to find a global min-cut.

- $r = C_n^2$: $Pr(FAILURE) \leq (1 - \frac{1}{C_n^2})^{C_n^2} \leq \frac{1}{e}$.
- $r = C_n^2 \ln n$: $Pr(FAILURE) \leq (1 - \frac{1}{C_n^2})^{C_n^2 \ln n} \leq \frac{1}{e^{\ln n}} = \frac{1}{n}$.

Time complexity: $O(rm)$ (Contraction algo costs $O(m)$ time.)

Extension: the number of GloablMinCut

- Question: what is the maximum number of global min-cuts an undirected graph G can have?
- Not obvious. Consider a directed graph as follows: s together with any subset of $v_1, ..., v_n$ constitutes a minimum $s - t$ cut. (2^n cuts in total.)



Extension: the number of GloablMinCut cont'd

Theorem

An undirected graph G on n nodes has at most C_n^2 global min-cuts.

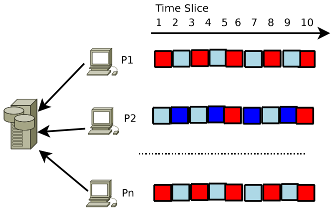
Proof:

- Suppose there are r global min-cut $c_1, ..., c_r$;
- Let C_i denote the event that c_i is reported, and C denote the success of Contraction algo;
- For each i, we have $Pr[C_i] \geq \frac{1}{C_n^2}$.
- Thus $Pr[C] = Pr[C_1 \cup ... \cup C_r] = \sum_i Pr[C_i] \geq r\frac{1}{C_n^2}$. (Note: $=$ since all C_i are disjoint.)
- We get $r \leq C_n^2$. $(r\frac{1}{C_n^2} \leq 1.)$

Randomization in distributed computing

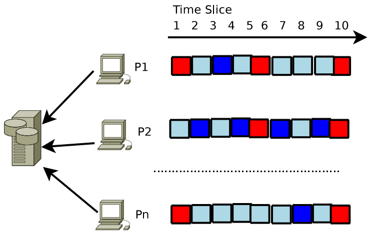
INPUT:
Suppose we have n nodes $M_1, ..., M_n$, each competing for access to a single shared database. The database can be accessed by at most one node in a single time slice; if two or more nodes attempt to access it simultaneously, then all nodes are "locked out" for the duration of that slice.

GOAL:
to design a protocol to divide up the time slices among the nodes in an equitable fashion. (suppose that the nodes cannot communicate with one another at all.)



Protocol design: ContentionResolution

- A randomized algorithm: *Each node will attempt to access the database at each slice with probability p, independently of the decisions of others.*
- Intuition: each node randomizes its behavior.
- Symmetry-breaking strategy: If all nodes operated in lockstep, repeatedly trying to access the database at the same time, there'd be no progress; but by randomizing, they "smooth out" the contention.



Waiting for a particular node to succeed.

Theorem

After $\Theta(n)$ *time slices, that probability that* M_i *has not yet succeeded is less than a constant; and after* $\Theta(n \ln n)$ *time slices, the probability drops to a quite small quantity.*

Proof.

- Let $A(i,t)$ denote the event that M_i attempts to access DB at time t, and $S(i,t)$ denote the success of the access.
- We have:
  $Pr[S(i,t)] = Pr[A(i,t)] \times \prod_{j \neq i} Pr[\overline{A(j,t)}] = p(1-p)^{n-1}$
- By setting the derivative to 0, we get $p = \frac{1}{n}$. And the maximum of $Pr[S(i,t)]$ is achieved: $Pr[S(i,t)] = \frac{1}{n}(1-\frac{1}{n})^{n-1}$.
- $\frac{1}{en} \leq Pr[S(i,t)] \leq \frac{1}{2n}$. (Reason: As n increases from 2, $(1-\frac{1}{n})^n$ coverges monotonically from $\frac{1}{4}$ to $\frac{1}{e}$, and $(1-\frac{1}{n})^{n-1}$ coverges monotonically from $\frac{1}{2}$ to $\frac{1}{e}$.)
- Let $F(i,t)$ denote the "failure event" that P_i does not succeed in any of the slices 1 through t;
- $Pr[F(i,t)] = \prod_{r=1}^{t} Pr[\overline{S(i,r)}] = (1-\frac{1}{n}(1-\frac{1}{n})^{n-1})^t$.
- A simpler estimation: $Pr[F(i,t)] = \prod_{r=1}^{t} Pr[\overline{S(i,r)}] \leq (1-\frac{1}{en})^t$.
- $Pr[F(i,t)] \leq (1-\frac{1}{en})^t \leq \frac{1}{e}$ when setting t to en.
- $Pr[F(i,t)] \leq (1-\frac{1}{en})^t \leq (\frac{1}{e})^{c \ln n} = n^{-c}$ when setting t to $cen \ln n$.

Waiting for all nodes to succeed.

Theorem

With probability at least $1 - n^{-1}$, all nodes succeed in accessing the DB at least once within $t = 2en \ln n$ time slices.

Proof.

- Let $F(t)$ denotes the event that some nodes have not yet accessed DB after t time slices;
- $Pr[F(t)] = Pr[\bigcup_{i=1}^{n} F(i,t)] \leq \sum_{i=1}^{n} Pr[F(i,t)]$
- $Pr[F(t)] \leq n \times n^{-c}$ after $t = cen \ln n$ time slices.
- In particular, $Pr[F(t)] \leq \frac{1}{n}$ after $t = 2en \ln n$ time slices.

$\square$

INPUT: n processors $P_1, ..., P_n$, and m jobs arrive in a stream and need to be processed immediately;
GOAL: to design a protocol to distribute jobs among processors evenly. (Assuming no central controller again.)

Randomized algorithm: *assign each job to one of the processors uniformaly at random.*

Analysis: how well does this simple algo work? I

Theorem

(A simple case: $m = n$) With probability at least $1 - n^-1$, there is no processor that was assigned with over $e\gamma(n) = \Theta(\frac{\log n}{\log \log n})$ jobs.

Analysis: how well does this simple algo work? II

Proof.

- Let X_i denote the number of jobs assigned to P_i. Define an index random variable Y_{ij} as follows: $Y_{ij} = 1$ when job j is assigned to P_i, and 0 otherwise.
- We have: $X_i = \sum_{j=1}^{n} Y_{ij}$.
- Then,

$$
\begin{aligned}
E(X_i) &= E(\sum_{j=1}^{n} Y_{ij}) \\
&= \sum_{j=1}^{n} E(Y_{ij}) \\
&= \sum_{j=1}^{n} Pr(Y_{ij} = 1) \\
&= n \times \frac{1}{n} = 1
\end{aligned}
$$

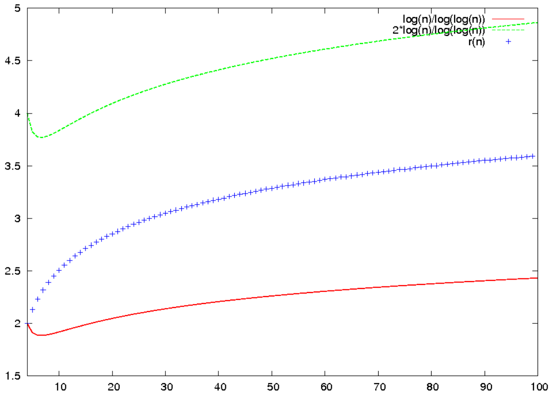
- Thus $Pr[X_i > c] < \frac{e^{c-1}}{c^c}$ (by Chernoff bound.)
- Suppose we have a c such that $Pr[X_i > c] < \frac{e^{c-1}}{c^c} \leq \frac{1}{n^2}$, then $Pr[\exists i, X_i > c] \leq n \times \frac{1}{n^2} = \frac{1}{n}$.

$\square$

The remaining difficulty: how to choose a c?

- Let $\gamma(n)$ be the solution to $x^x = n$. The estimation of $\gamma(n)$ can be given as follows:
- Taking logarithm of $x^x = n$ gives: $x \ln x = \ln n$.
- Taking logarithm again: $\ln x + \ln\ln x = \ln\ln n$.
- We have: $\ln x \leq \ln\ln n = \log x + \ln\ln x \leq 2\ln x$ (by $log(log(x)) \leq log(x)$ Dividing the equation: $x \ln x = \ln n$ (by $lnln(n) \geq 0$ when $n \geq e^e$) , we get:
- $\frac{1}{2}x \leq \frac{\ln n}{\ln\ln n} \leq x = \gamma(n)$.
- Setting $c = e\gamma(n)$, we have: $Pr[X_i > c] < \frac{e^{c-1}}{c^c} < (\frac{e}{c})^c = (\frac{1}{\gamma(n)})^{e\gamma(n)} < (\frac{1}{\gamma(n)})^{2\gamma(n)} = \frac{1}{n^2}$.

$\gamma(n)$: the solution to $x^x = n$

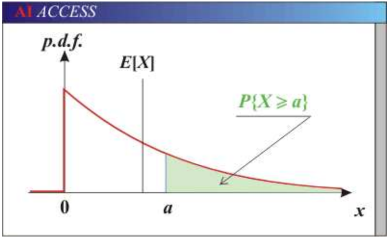


- More jobs: ($m = 6n \ln n$) The expected jobs number is: $\mu = 6 \ln n$. We have:
- $Pr[X_i > 2\mu] < (\frac{e}{4})^{6 \ln n} < (\frac{1}{e^2})^{\ln n} = \frac{1}{n^2}$. (by $(\frac{e}{4})^6 < \frac{1}{e^2}$.)

Bounding the sum of independent random variables

Bound 1: Markov inequality

- Suppose X is a non-negative random variable with mean $u = E(X)$.
- We have: $\Pr[X \geq t] \leq \frac{E(X)}{t}$.



Bound 2: Chebyshev inequality

- Suppose X is a random variable with mean $u = E(X)$, and variance $\sigma^2 = Var(X)$.
- We have: $\Pr[|X - u| \geq k\sigma] \leq \frac{1}{k^2}$.

Note: the non-negative requirement is removed, but need the information of variance.

Theorem

(Upper bound) Let $X = X_1 + X_2 + ... + X_n$*, where* X_i *is* $0/1$ *variable that takes* 1 *with probability* p_i*. Define* $\mu = E(X) = \sum_i p_i$*. For any* $\delta > 0$*, we have:* $Pr[X > (1+\delta)\mu] < (\frac{e^\delta}{(1+\delta)^{(1+\delta)}})^\mu$*.*

(Intuition: the flucuations of X_i are likely to be "cancelled out" as n increases.)

Bound 3: Chernoff bound (upper bound) II

Proof.

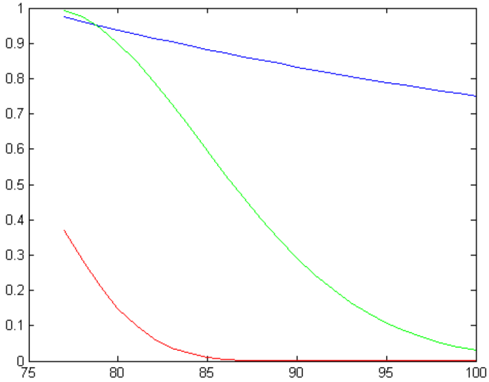
- Step 1: $Pr[X > (1+\delta)\mu] = Pr[tX > t(1+\delta)\mu] = Pr[e^{tX} > e^{t(1+\delta)\mu}] \leq \frac{E(e^{tX})}{e^{t(1+\delta)\mu}}$ for any $t > 0$. (Applying Markov inequality on e^{tX})
- Step 2: $E(e^{tX}) = E(e^{tX_1+...+tX_n}) = E(e^{tX_1}...e^{tX_n})$ (by independence of X_i.)
- Step 3: $E(e^{tX_i}) = e^t p_i + 1(1-p_i) = 1 + p_i(e^t - 1) \leq e^{p_i}(e^t - 1)$ (by $1 + x \leq e^x$, for $x > 0$.)
- Thus we have: $Pr[X > (1+\delta)\mu] \leq \frac{E(e^{tX})}{e^{t(1+\delta)\mu}} \leq \frac{\prod_i e^{p_i(e^t-1)}}{e^{t(1+\delta)\mu}} = \frac{e^{\mu(e^t-1)}}{e^{t(1+\delta)\mu}}$.
- Step 4: Setting $t = \ln(1+\delta)$.

□

Theorem

(Lower bound) Let $X = X_1 + X_2 + ... + X_n$, *where* X_i *is* $0/1$ *variable that takes* 1 *with probability* p_i. *Define* $\mu = E(X) = \sum_i p_i$. *For any* $\delta > 0$, *we have:* $Pr[X < (1-\delta)\mu] < e^{-\frac{1}{2}\mu\delta^2}$.

Comparison of Markov inequality and Chernoff bound



Trial: The number of heads in 100 tosses of a coin. $Pr[head] = 0.75$. Lines: The real probability (in red); Markov bound (in blue); Chernoff bound (in green).

LP+Random rounding paradigm: MaxSAT problem

INPUT:
Given a set of clauses $C_1, ..., C_k$, each of length 3, over a set of boolean variables $X = \{x_1, ..., x_n\}$;
OUTPUT:
to find an assignment to maximize the number of satisfied clauses;

e.g.
$C1 : x_1 \vee \neg x_2 \vee x_3$
$C2 : \neg x_1 \vee \neg x_2 \vee x_4$
$C3 : \neg x_3 \vee \neg x_4 \vee x_5$
$C4 : \neg x_2 \vee \neg x_4 \vee x_7$

Algo1 (remarkably simple):

1: set each variable x_i to 1 with probability $\frac{1}{2}$.

Algo1 is good

Theorem

The expected number of clauses satisfied by Algo1 is within an approximation factor $\frac{7}{8}$ of optimal.

Proof.

- Let X be the number of satisifed clauses. X_i is an index variable such that $X_i = 1$ if C_i was satisfied, and $X_i = 0$ otherwise.
- $X = X_1 + ... + X_k$
- $E(X) = E(X_1 + ... + X_k) = E(X_1) + ... + E(X_k) = \frac{7}{8}k$ $(E(X_i) = \frac{7}{8})$
- and we have a lower bound: $OPT \leq k$.
- Thus, $E(X) \geq \frac{7}{8}OPT$.

$\square$

Extensions

Probability method:

Corollary

There exists at least an assignment to satisfy at least $\frac{7}{8}k$ clauses.

(Intuition: the expectation is over $\frac{7}{8}k$ clauses. Just an existence proof.)

Probability method again:

Corollary

All 3SAT instance with at most 7 clauses are satisfied.

(Intuition: The unsatisfied clause number is $\frac{1}{8}k = \frac{7}{8} < 1.$)

- Question: how to find an assignment to satisfy at least $\frac{7}{8}k$ clauses?
- Algo 2:

1: repeat Algo1 until at least $\frac{7}{8}k$ clauses are satisfied.

Find a good assignment: analysis I

Theorem

The expected running time of Algo2 is polynomial. In particular, the expected number of $repetition$ is less than $8k$.

Find a good assignment: analysis II

Proof.

- Let p be the probability that at least $\frac{7}{8}$ clauses are satisfied;
- It suffices to prove that $\frac{1}{p} \leq 8k$. (Reason: the expected waiting time of an event with probability p is $\frac{1}{p}$.)
- Let p_j be the probability that EXACTLY j clauses are satisfied. We have:
  1. $p = \sum_{j \geq \frac{7}{8}k} p_j$,
  2. $p + \sum_{j < \frac{7}{8}k} p_j = 1$;
  3. $E(X) = \frac{7}{8}k = \sum_{j=1}^{k} jp_j = \sum_{j \geq \frac{7}{8}k} jp_j + \sum_{j < \frac{7}{8}k} jp_j$.
- Thus,
  $\frac{7}{8}k \leq \sum_{j \geq \frac{7}{8}k} kp_j + \sum_{j < \frac{7}{8}k} k'p_j = kp + k'(1-p) \leq k' + kp$.
  (k' is the max number such that $k' < \frac{7}{8}k$.)
- Thus, $kp \geq \frac{7}{8}k - k'$, and $p \geq \frac{1}{8k}$. (since $\frac{7}{8}k - k' \geq \frac{1}{8}$.)

$\square$

Algo3: "LP+Random Rounding" strategy

ILP formulation

$C1 : x_1 \vee \neg x_2 \vee x_3$
$C2 : \neg x_1 \vee \neg x_2 \vee x_4$
$C3 : \neg x_3 \vee \neg x_4 \vee x_5$
$C4 : \neg x_2 \vee \neg x_4 \vee x_7$
ILP:

$$
\begin{array}{llrcr}
\max & z = & z_1 + z_2 + \ldots z_k & & \\
s.t. & & x_1 + (1 - x_2) + x_3 & \geq & z_1 \\
 & & (1 - x_1) + (1 - x_2) + x_4 & \geq & z_2 \\
 & & (1 - x_3) + (1 - x_4) + x_5 & \geq & z_3 \\
 & & x_i & = & 0/1 \\
 & & z_j & = & 0/1
\end{array}
$$

Relax ILP to LP

LP:

$$\begin{array}{rrcl}
\max & z = \quad z_1 + z_2 + ...z_k & & \\
s.t. & x_1 + (1 - x_2) + x_3 & \geq & z_1 \\
 & (1 - x_1) + (1 - x_2) + x_4 & \geq & z_2 \\
 & (1 - x_3) + (1 - x_4) + x_5 & \geq & z_3 \\
 & ...... & & \\
 & x_i & \leq & 1 \\
 & z_j & \leq & 1
\end{array}$$

Algo3:

1: Let x^*, z^* denote the optimal solution to LP.
2: Randomly set variable $x_i = TRUE$ with probability x_i^*.

Performance

Theorem

A clause C_j is satisfied with a probability at least $(1-(1-\frac{1}{3})^3)z_j^*$.

Proof.

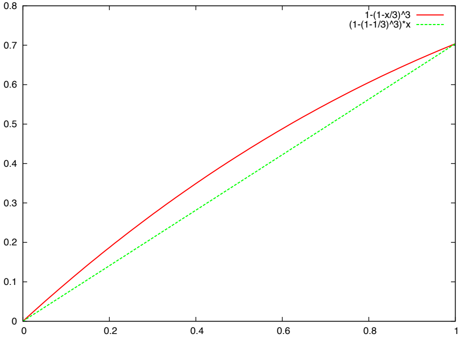
Suppose w.l.o.g $C_j = x_1 \vee x_2 \vee x_3$. We have:

$$
\begin{aligned}
Pr(C_j \text{ is satisfied}) &= 1-(1-x_1^*)(1-x_2^*)(1-x_3^*) \\
&\geq 1-(\tfrac{1}{3}((1-x_1^*)+(1-x_2^*)+(1-x_3^*)))^3 \quad (\text{ by} \\
&\geq 1-(1-\tfrac{1}{3}z_j^*)^3 \quad (\text{ by Fact 2.}) \\
&\geq (1-(1-\tfrac{1}{3})^3)z_j^* \quad (\text{ by Fact 3.})
\end{aligned}
$$

$\square$

Some facts

- Fact 1: The opitmal solution satisfies: $x_1^* + x_2^* + x_3^* \geq z_j^*$.
- Fact 2: $(x_1 x_2 ... x_n)^{\frac{1}{n}} \leq \frac{1}{n}(x_1 + x_2 + ... + x_n)$
- Fact 3: $f(x) = 1 - (1 - \frac{1}{3}x)^3$ is concave, and greater than $g(x) = (1 - (1 - \frac{1}{3})^3)x$ at the two ends of $[0, 1]$. Thus, $f(x) \geq g(x)$ for any $x \in [0, 1]$.



Theorem

(Goemans, W '94) Algo3 is a $(1-\frac{1}{e})$*-approximation algorithm, where* $(1-\frac{1}{e}) = 0.632$*.*

Proof.

Let X be the number of satisfied clauses. Let index variable c_j be 1 when clause C_j is satisfied, and 0 otherwise. Thus, $X = c_1 + c_2 ... + c_k$.

$$
\begin{aligned}
E(X) &= E(c1) + E(c_2) + ... + E(c_k) \\
&= \sum_j Pr(C_j \text{ is satisfied}) \quad \text{(previous theorem)} \\
&\geq \sum_j (1-(1-\tfrac{1}{3})^3) z_j^* \\
&\geq (1-(1-\tfrac{1}{3})^3) \sum_j z_j^* \\
&= (1-(1-\tfrac{1}{3})^3) z_{LP} \\
&\geq (1-(1-\tfrac{1}{3})^3) OPT \quad (\text{by } z_{LP} \geq z_{ILP} = OPT) \\
&\geq (1-\tfrac{1}{e}) OPT \quad (\text{by } (1-\tfrac{1}{n})^n \leq \tfrac{1}{e})
\end{aligned}
$$

$\square$

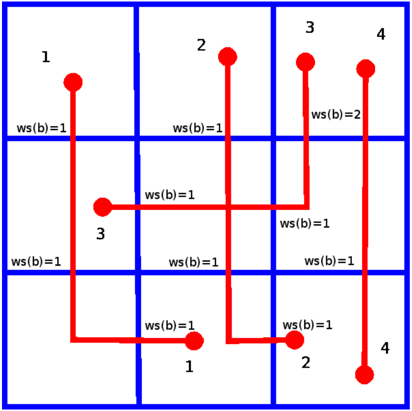
LP+Random rounding paradigm: VLSI DESIGN problem

Problem Statement I

- A *gate-array* is a two-dimensional $\sqrt{n} \times \sqrt{n}$ array of gates abutting each other.
- A *net* is a set of gates to be connected by a wire. In our problem, the number of gates in a set is exactly 2.
- Assume that the wire for each net contains at most one 90^o turn, called "one-bend" route. Thus, in joining the two end-points of a net, the wire will either first traverse the horizontal dimension and then the vertical dimension, or the other way around. In particular, a net , which connects two gates in the same column or the same row, only has one choice.
- Let $w_S(b)$ denote teh number of wires that pass through *boundary* b in a solution S. Here, each of the four edges of a grid is called a *boundary* b.

Problem Statement II

The Problem is $\min_S \max_b w_S(b)$. (Intuition: not too many wires pass through any boundary.)



Algorithm I

- This problem can be cast as a $0-1$ linear program (because for each net, there is at most 2 choices.).
- For each net i from left end-point to the right end-point, we define 2 variables x_{i0} and x_{i1} to describe the direction of the wire:

$$
\begin{array}{ll}
x_{i0}=1, x_{i1}=0 & \text{if net } i \text{ goes horizontally first} \\
x_{i0}=0, x_{i1}=1 & \text{if net } i \text{ goes vertically first}
\end{array}
$$

- For each boundary b in the array, let

$$
\begin{aligned}
T_{b0} &= \{i \mid \text{net } i \text{ passes through } b \text{ if } x_{i0}=1\} \\
T_{b1} &= \{i \mid \text{net } i \text{ passes through } b \text{ if } x_{i1}=1\}
\end{aligned}
$$

Algorithm II

- With these definitions, our integer program can be expressed as:

$$\min w$$

$$\begin{aligned} x_{i0} + x_{i1} &= 1 \quad \forall i \\ \sum_{i \in T_{b0}} x_{i0} + \sum_{i \in T_{b1}} x_{i1} &\leq w \quad \forall b \\ x_{i0}, x_{i1} &\in \{0, 1\} \end{aligned}$$

- Let OPT be the objective value of the above ILP.

Algorithm III

- We solve instead the linear program relaxation of ILP by replacing $x_{i0}, x_{i1} \in \{0,1\}$ to $x_{i0}, x_{i1} \in [0,1]$:

$$\min w$$

$$\begin{array}{rcll} x_{i0} + x_{i1} & = & 1 & \forall i \\ \sum_{i \in T_{b0}} x_{i0} + \sum_{i \in T_{b1}} x_{i1} & \leq & w & \forall b \\ x_{i0}, x_{i1} & \in & [0,1] & \end{array}$$

Let $\hat{x_{i0}}$ and $\hat{x_{i1}}$ be the solution, $\hat{w}$ be the objective value, of the above LP. Obviously, $\hat{w} \leq OPT$.

LP +Random Rounding

- Algo: *Randomized Rounding* $\hat{x_{i0}}$ and $\hat{x_{i1}}$ to 0 and 1.
- Indepently for each i, define 2 random variables, $\bar{x_{i0}}$ and $\bar{x_{i1}}$.

$$\begin{aligned} Pr(\bar{x_{i0}} = 1, \bar{x_{i1}} = 0) &= \hat{x_{i0}} \\ Pr(\bar{x_{i1}} = 1, \bar{x_{i0}} = 0) &= \hat{x_{i1}} \end{aligned}$$

- Obviously, $E(\bar{x_{i0}}) = \hat{x_{i0}}$ and $E(\bar{x_{i1}}) = \hat{x_{i1}}$.
- Now we get a solution $S = \{\hat{x_{i0}}, \hat{x_{i1}}, i = 1, 2, ..., n\}$ to the problem, how about its performance?

Analysis I

Theorem

Let ϵ be a real number such that $0 < \epsilon < 1$. Then with probability $1-\epsilon$, the solution S produced by randomized rounding satisfies $w_S \leq (1+\Delta(\hat{w}, \epsilon/2n))\hat{w} \leq (1+\Delta(OPT, \epsilon/2n))OPT$. where $\Delta(\mu, \epsilon)$ is defined as : if let $\left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu} = \epsilon$, then $\delta = \Delta(\mu, \epsilon)$.

Proof

- The second inequality is obvious.
- In order to prove the first inequality, we just need to prove that : for any boundary b, the probability that $w_S(b) > \hat{w}(1+\Delta(\hat{w}, \epsilon/2n))$ is at most $\epsilon/2n$. (Why?)
- Consider a boundary b, $w_S(b) = \sum_{i \in T_{b0}} \bar{x_{i0}} + \sum_{i \in T_{b1}} \bar{x_{i1}}$ then $E(w_S(b)) = \sum_{i \in T_{b0}} E(\bar{x_{i0}}) + \sum_{i \in T_{b1}} E(\bar{x_{i1}}) = \sum_{i \in T_{b0}} \hat{x_{i0}} + \sum_{i \in T_{b1}} \hat{x_{i1}} \leq \hat{w}$

Analysis II

- According to the Definition of Δ and Chenoff Bound, we have $Pr(w_S(b) > \hat{w}(1 + \Delta(\hat{w}, \epsilon/2n))) \leq \epsilon/2n$ and the theorem follows.

Randomized divide-and-conquorer

Randomized divide-and-conquorer: SELECTION problem I

INPUT:
Given a set of number $S = \{a_1, a_2, ..., a_n\}$, and a number $k \leq n$;
OUTPUT:
the median in S, or the k-th smallest item.

Note: known deterministic linear algorithms, say Blum '73 ($16n$ comparisons), and D. Zuick '95 ($2.95n$ comparisons).
Randomized algorithm:

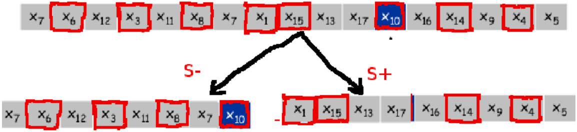
Randomized divide-and-conquorer: SELECTION problem II

```
Select( n, k)

Choose an element ai in S uniformly at random;

for j = 1 to n
do
    if aj > ai
        add aj onto S+;
    else
        add aj onto S-;
done
if |S-| = k-1
    return ai;
else
     if |S-| >=k
         Select( S-, k);
     else
          Select(S+, k-1-l);
          //Here, l = |S-|
```

Randomized divide-and-conquorer: SELECTION problem III



(Intuition: sloving an extension of the original problem, i.e., to find the k-th median. At first, an element a_i is chosen to split S into two parts $S^+ = \{a_j : a_j \geq a_i\}$, and $S^- = \{a_j : a_j < a_i\}$. We can determine whether the k-th median is in S^+ or S^-. Thus, we perform iteration on ONLY one subset.)

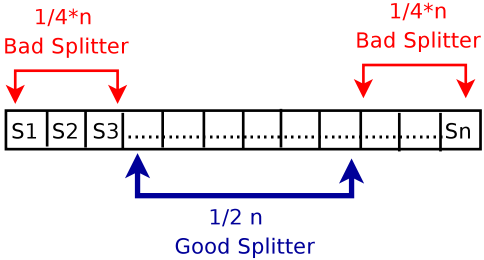
Difficulty: how to choose the splitter?

- Bad choice: select the smallest element at each iteration. $T(n) = T(n-1) + O(n) = O(n^2)$

Randomized divide-and-conquorer: SELECTION problem IV

- Ideal choice: select the median at each iteration.
  $T(n) = T(\frac{n}{2}) + O(n) = O(n)$
- Good choice: select a "centered" element a_i, i.e., $|S^+| \geq \epsilon n$, and $|S^-| \geq \epsilon n$ for a fixed $\epsilon > 0$.
  $T(n) \leq T((1-\epsilon)n) + O(n) \leq$
  $cn + c(1-\epsilon)n + c(1-\epsilon)^2 n + ... = O(n).$

e.g.: $\epsilon = \frac{1}{4}$:



Randomized divide-and-conquorer: SELECTION problem V

Key observation: if we choose a splitter $a_i \in S$ uniformly at random, it is easy to get a good splitter since a fairly large fraction of the elements are "centered".

Theorem

The expected running time of Select(n,k) is $O(n)$.

Randomized divide-and-conquer: SELECTION problem VI

Proof.

- Let $\epsilon = \frac{1}{4}$. We'll say that the algorithm is in phase j when the size of set under consideration is in $[n(\frac{3}{4})^{j-1}, n(\frac{3}{4})^{j}]$.
- Let X be the number of steps. And X_j be the number of steps in phase j. Thus, $X = X_0 + X_1 + ...$
- Consider the j-th phase. The probability to find a centered splitter is $\geq \frac{1}{2}$ since at least half elements are centered. Thus, the expected number of iterations to find a centered splitter is: 2.
- Each iteration costs $cn(\frac{3}{4})^{j}$ steps since there are at most $n(\frac{3}{4})^{j}$ elements in phase j. Thus, $E(X_j) \leq 2cn(\frac{3}{4})^{j}$.
- $E(X) = E(X_0 + X_1 + ...) \leq \sum_j 2cn(\frac{3}{4})^{j} \leq 8cn$.

$\square$

Hashing

(See extra slides)