Introduction to Randomized Algorithms

Subhas C. Nandy
(nandysc@isical.ac.in)

Advanced Computing and Microelectronics Unit
Indian Statistical Institute
Kolkata 700108, India.

Organization

Introduction

Goal of a Deterministic Algorithm

- The solution produced by the algorithm is correct, and
- the number of computational steps is same for different runs of the algorithm with the same input.

Problems in Deterministic Algorithm

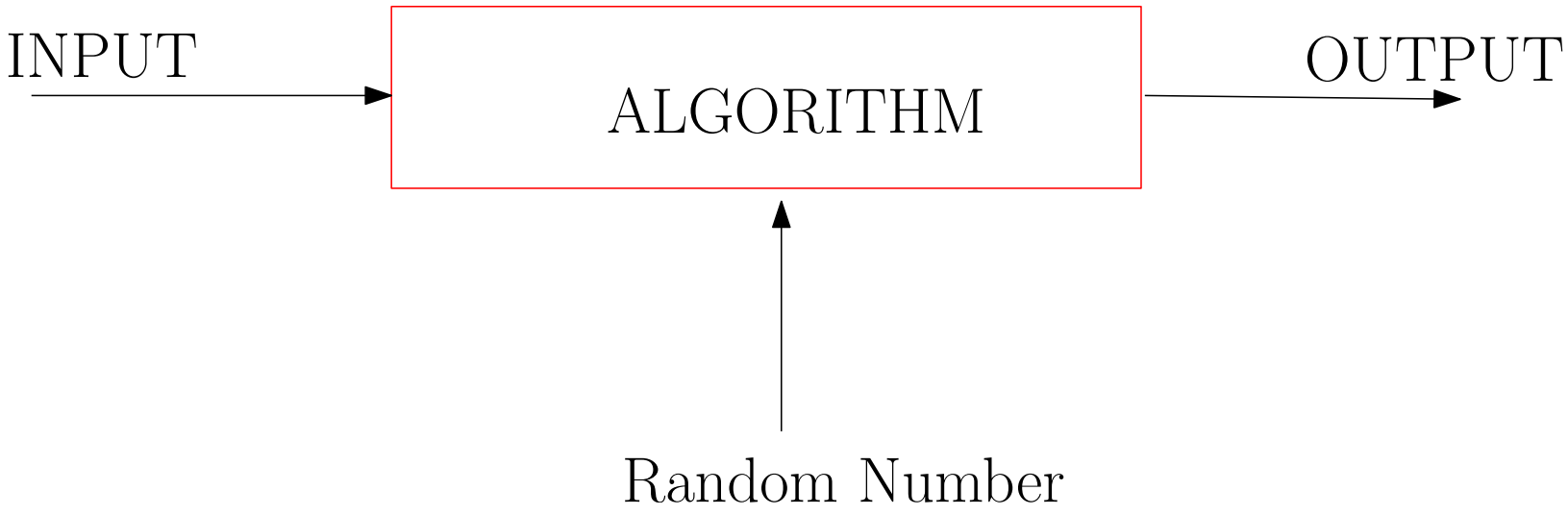
Given a computational problem,

- it may be difficult to formulate an algorithm with good running time, or
- the explotion of running time of an algorithm for that problem with the number of inputs.

Remidies

- Efficient heuristics,
- Approximation algorithms,
- Randomized algorithms

Randomized Algorithm



- In addition to the input, the algorithm uses a source of pseudo random numbers. During execution, it takes random choices depending on those random numbers.
- The behavior (output) can vary if the algorithm is run multiple times on the same input.

Advantage of Randomized Algorithm

- The algorithm is usually simple and easy to implement,
- The algorithm is fast with very high probability, and/or
- It produces optimum output with very high probability.

Difficulties

- There is a finite probability of getting incorrect answer. However, the probability of getting a wrong answer can be made arbitrarily small by the repeated employment of randomness.
- Analysis of running time or probability of getting a correct answer is usually difficult.
- Getting truely random numbers is impossible. One needs to depend on pseudo random numbers. So, the result highly depends on the quality of the random numbers.

An Important Note

Randomized algorithms are not the probabilistic analysis of expected running time of a deterministic algorithm, where

- The inputs are assumed to come from a probability distribution.
- The objective is to compute the expected running time of the algorithm.

Randomized Quick Sort

Deterministic Quick Sort

The Problem:

Given an array $A[1 \ldots n]$ containing n (comparable) elements, sort them in increasing/decreasing order.

QSORT(A, p, q)

- If $p \geq q$, EXIT.
- Compute $s \leftarrow$ correct position of $A[p]$ in the sorted order of the elements of A from p-th location to q-th location.
- Move the pivot $A[p]$ into position $A[s]$.
- Move the remaining elements of $A[p-q]$ into appropriate sides.
- QSORT(A, p, $s-1$);
- QSORT(A, $s+1$, q).

Complexity Results of QSORT

- An INPLACE algorithm
- The worst case time complexity is $O(n^2)$.
- The average case time complexity is $O(n \log n)$.

Randomized Quick Sort

An Useful Concept - The Central Splitter

It is an index s such that the number of elements less (resp. greater) than $A[s]$ is at least $\frac{n}{4}$.

- The algorithm randomly chooses a key, and checks whether it is a central splitter or not.
- If it is a central splitter, then the array is split with that key as was done in the QSORT algorithm.
- It can be shown that the expected number of trials needed to get a central splitter is constant.

Randomized Quick Sort

RandQSORT(A, p, q)

1: If $p \geq q$, then EXIT.

2: While no central splitter has been found, execute the following steps:

2.1: Choose uniformly at random a number $r \in \{p, p+1, \ldots, q\}$.

2.2: Compute s = number of elements in A that are less than $A[r]$, and
t = number of elements in A that are greater than $A[r]$.

2.3: If $s \geq \frac{q-p}{4}$ and $t \geq \frac{q-p}{4}$, then $A[r]$ is a central splitter.

3: Position $A[r]$ is $A[s+1]$, put the members in A that are smaller than the central splitter in $A[p \ldots s]$ and the members in A that are larger than the central splitter in $A[s+2 \ldots q]$.

4: RandQSORT(A, p, s);

5: RandQSORT(A, $s+2$, q).

Analysis of RandQSORT

Fact: Step 2 needs $O(q - p)$ time.

Question: How many times Step 2 is executed for finding a central splitter ?

Result:

The probability that the randomly chosen element is a central splitter is $\frac{1}{2}$.

Implication

- The expected number of times the Step 2 needs to be repeated to get a central splitter is 2.
- Thus, the expected time complexity of Step 2 is $O(n)$

Analysis of RandQSORT

Time Complexity

- Worst case size of each partition in j-th level of recursion is $n \times (\frac{3}{4})^j$.
- Number of levels of recursion $= \log_{\frac{4}{3}} n = O(\log n)$.
- Recurrence Relation of the time complexity: $T(n) = 2T(\frac{3n}{4}) + O(n) = O(n \log n)$

Smallest Enclosing Disk Problem

Smallest Enclosing Disk

The Problem:

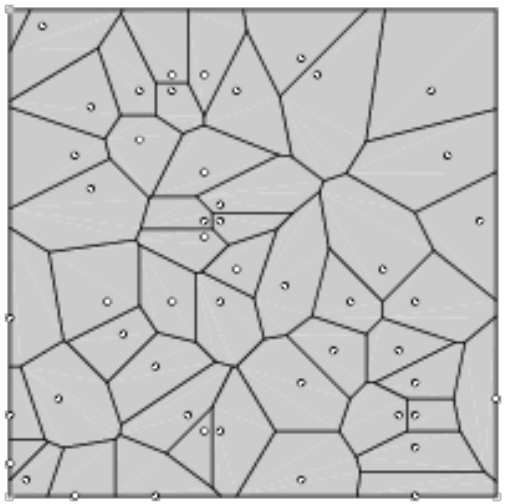
GIven a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in 2D, compute a disk of minimum radius that contains all the points in P.

Trivial Solution: Consider each triple of points $p_i, p_j, p_k \in P$, and check whether every other point in P lies inside the circle defined by p_i, p_j, p_k. Time complexity: $O(n^4)$

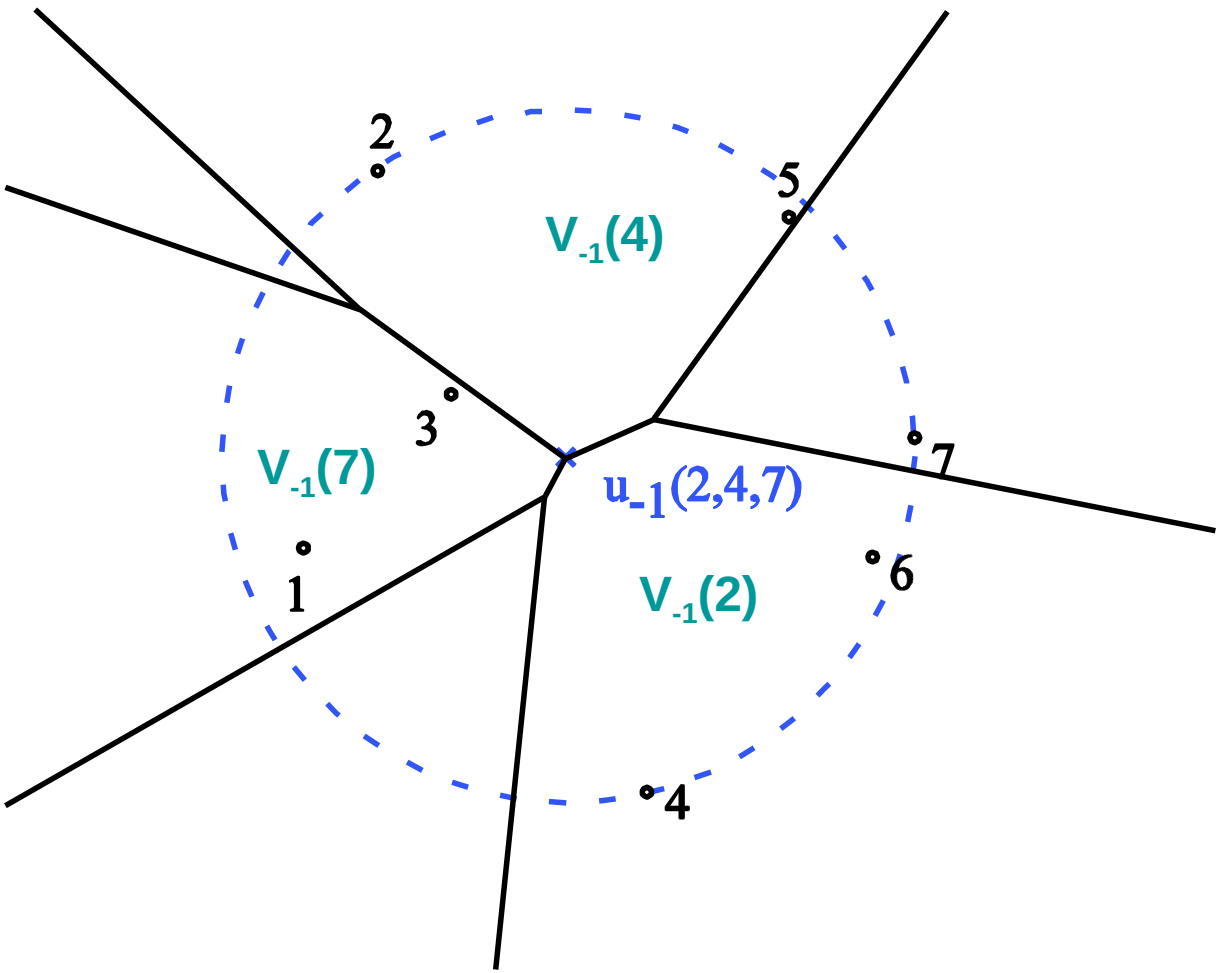
An Easy Implementable Efficient Solution: Consider furthest point Voronoi diagram. Its each vertex represents a circle containing all the points in P. Choose the one with minimum radius. Time complexity: $O(n \log n)$

Best Known Result: A complicated $O(n)$ time algorithm (using linear programming).

Smallest Enclosing Disk



Smallest Enclosing Disk



A Simple Randomized Algorithm

We generate a random permutation of the points in P.

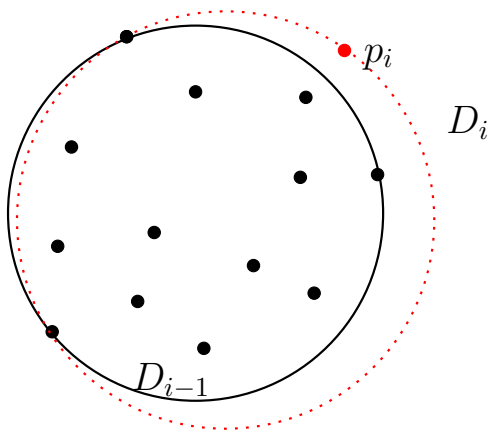
Notations:

- $P_i = \{p_1, p_2, \ldots, p_i\}$.
- D_i = the smallest enclosing disk of P_i.

An incremental procedure

Result:

- If $p_i \in D_{i-1}$ then $D_i = D_{i-1}$.
- If $p_i \notin D_{i-1}$ then p_i lies on the boundary of D_i.

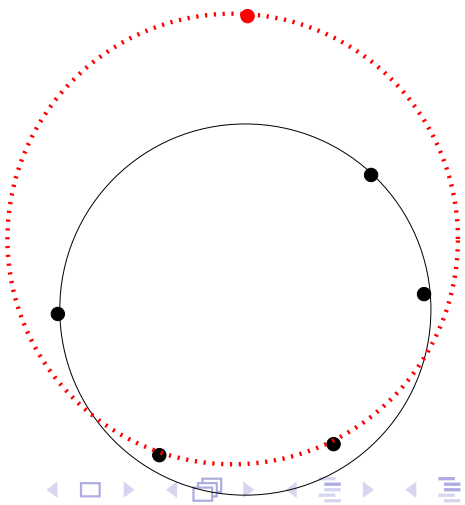


Algorithm MINIDISC(P)

Input: A set P of n points in the plane.
Output: The smallest enclosing disk for P.

1. Compute a random permutation of $P = \{p_1, p_2, \ldots, p_n\}$.
2. Let D_2 be the smallest enclosing disk for $\{p_1, p_2\}$.
3. **for** $i = 3$ to n **do**
4. **if** $p_i \in D_{i-1}$
5. **then** $D_i = D_{i-1}$
6. **else** $D_i =$ MINIDISKWITHPOINT$(\{p_1, p_2, \ldots, p_i\}, p_i)$
7. **return** D_n.

Critical Step: If $p_i \notin D_{i-1}$.



Algorithm MINIDISCWITHPOINT(P, q)

Idea: Incrementally add points from P one by one and compute the smallest enclosing circle under the assumption that the point q (the 2nd parameter) is on the boundary.

Input: A set of points P, and another point q.
Output: Smallest enclosing disk for P with q on the boundary.

1. Compute a random permutation of $P = \{p_1, p_2, \ldots, p_n\}$.
2. Let D_1 be the smallest enclosing disk for $\{p_1, q\}$.
3. **for** $j = 2$ to n **do**
4. **if** $p_j \in D_{j-1}$
5. **then** $D_j = D_{j-1}$
6. **else** $D_j =$ MINIDISKWITH2POINTS$(\{p_1, p_2, \ldots, p_j\}, p_j, q)$
7. **return** D_n.

Algorithm MINIDISCWITH2POINT(P, q_1, q_2)

Idea: Thus we have two fixed points; so we need to choose another point among $P \setminus \{q_1, q_2\}$ to have the smallest enclosing disk containing P.

1. Let D_0 be the smallest disk with q_1 and q_2 on its boundary.
3. **for** $k = 1$ to n **do**
4. **if** $p_k \in D_{k-1}$
5. **then** $D_k = D_{k-1}$
6. **else** $D_k =$ the disk with q_1, q_2 and p_k on its boundary
7. **return** D_n.

Correctness

Result:

Let P be a set of points in the plane, and R be a set of points with $P \cap R = \phi$. Then

- If there is a disk that encloses P, and has all points of R on its boundary, then the smallest enclosing disk is unique. We shall denote this disk by $md(P, R)$.
- For a point $p \in P$,
  - if $p \in md(P \setminus \{p\}, R)$, then $md(P, R) = md(P \setminus \{p\}, R)$
  - if $p \notin md(P \setminus \{p\}, R)$, then $md(P, R) = md(P \setminus \{p\}, R \cup \{p\})$

Time Complexity

Worst case: $O(n^3)$

Expected case:

- MINIDISKWITH2POINTS needs $O(n)$ time.
- MINIDISKWITHPOINTS needs $O(n)$ time if
  - **we do not consider the time taken in the call of the routine** MINIDISKWITH2POINTS.

Question: How many times the routine MINIDISKWITH2POINTS is called ?

Expected Case Time Complexity

Backward Analysis

- Fix a subset $P_i = \{p_1, p_2, \ldots, p_i\}$, and D_i is the smallest enclosing circle of P_i.
- If a point $p \in P_i$ is removed, and if p is in the proper interior of D_i, then the enclosing circle does not change.
- However, if p is on the boundary of D_i, then the circle gets changed.
- One of the boundary points is q. So, only for 2 other points, MINIDISKWITH2POINTS will be called from MINIDISKWITHPOINTS.

Expected Case Time Complexity

Observation:

The probability of calling MINIDISKWITH2POINTS is $\frac{2}{i}$.

Expected Running time of MINIDISKWITHPOINTS

$$O(n) + \sum_{i=2}^{n} O(i) \times \frac{2}{i} = O(n)$$

Similarly, we have

Expected Running time of MINIDISK

$O(n)$

Global Mincut Problem for an Undirected Graph

Global Mincut Problem

Problem Statement

Given a connected undirected graph $G = (V, E)$, find a cut (A, B) of minimum cardinality.

Applications:

- Partitioning items in a database,
- Identify clusters of related documents,
- Network reliability,
- Network design,
- Circuit design, etc.

Network Flow Solution

- Replace every edge (u, v) with two directed edges (u, v) and (v, u).
- Choose a pseudo vertex s, and connect it with all each vertex in $v \in V$ by a directed edge (s, v).
- For each vertex $v \in V$, compute the minimum s-v cut.

Result

Time complexity: $O(nM)$, where M is the time complexity of the best known algorithm for the Network Flow problem.

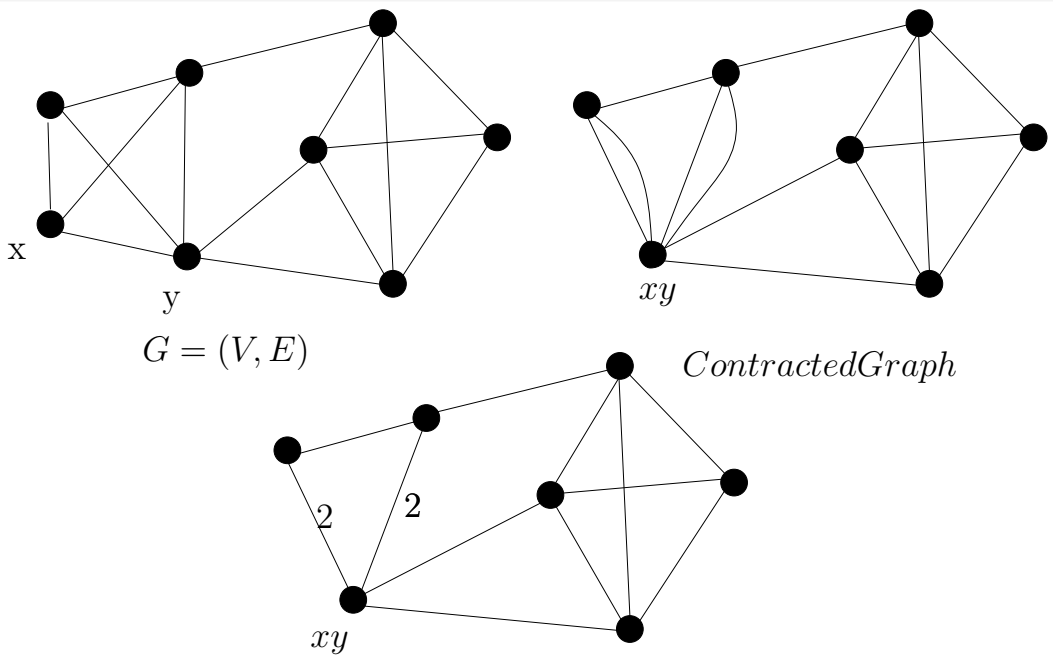
Best known result on Network Flow Problem:
Goldberg & Tarjan (1985) – $O(|E||V| \log(|V|^2/|E|))$

Relatively Easy algorithm for Network Flow Problem:
Malhotra, Pramod Kumar and Maheswari (1978) – $O(|V|^3)$

A Simple Randomized Algorithm

Contration of an Edge

Contraction of an edge $e = (x, y)$ implies merging the two vertices $x, y \in V$ into a single vertex, and remove the self loop. The contracted graph is denoted by G/xy.



$G = (V, E)$

$ContractedGraph$

$Contractedsimpleweightedgraph$

Results on Contration of Edges

Result - 1

As long as G/xy has at least one edge,

- The size of the minimum cut in the weighted graph G/xy is at least as large as the size of the minimum cut in G .

Result - 2

Let $e_1, e_2, \ldots, e_{n-2}$ be a sequence of edges in G, such that

- none of them is in the minimum cut of G, and
- $G' = G/\{e_1, e_2, \ldots, e_{n-2}\}$ is a single multiedge.

Then this multiedge corresponds to the minimum cut in G.

Problem: Which edge sequence is to be chosen for contraction?

Analysis

Algorithm **MINCUT**(G)

$G_0 \leftarrow G$; $i = 0$

while G_i has more than two vertices **do**

Pick randomly an edge e_i from the edges in G_i

$G_{i+1} \leftarrow G_i / e_i$

$i \leftarrow i + 1$

$(S, V - S)$ is the cut in the original graph corresponding to the single edge in G_i.

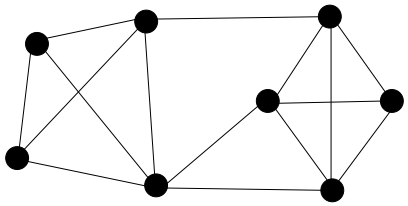
Theorem

Time Complexity: $O(n^2)$

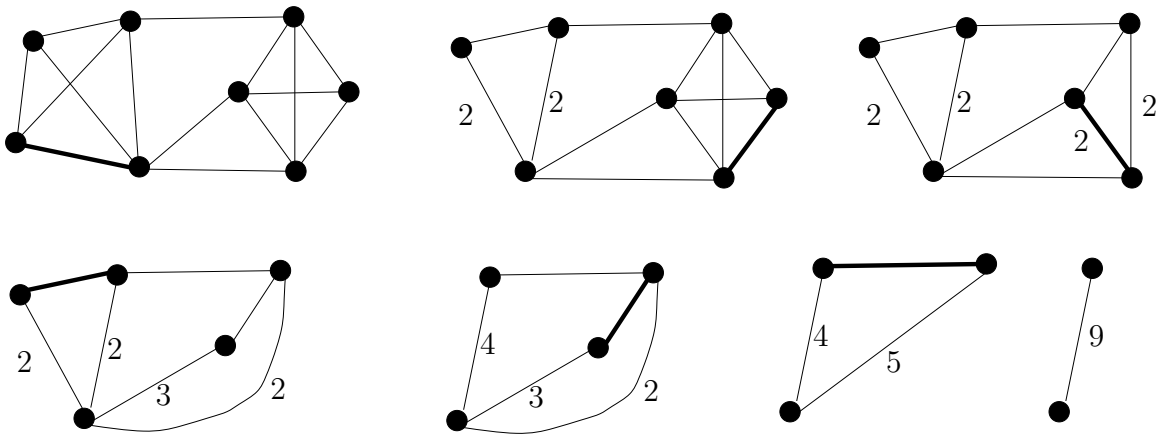
A Trivial Observation: The algorithm outputs a cut whose size is *no smaller than the mincut*.

Demonstration of the Algorithm

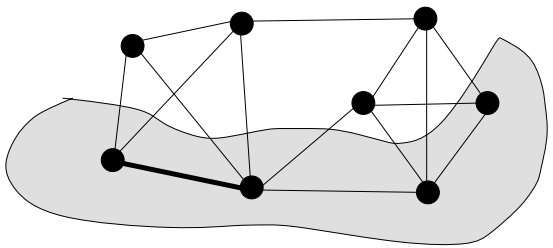
The given graph:



Stages of Contraction:



The corresponding output:



Quality Analysis

A Big Question: How good is the solution?

Result 1:

If a graph $G = (V, E)$ has a minimum cut of size k, and it has n vertices, then $|E| \geq \frac{kn}{2}$.

But, we don't know the min cut

Result 2:

If we pick a random edge e from the graph G, then the probability of belonging it in the mincut is at most $\frac{2}{n}$.

Correctness

Theorem

The procedure MINCUT outputs the mincut with probability $\geq \frac{2}{n(n-1)}$.

Proof:

Let $\eta_i \Rightarrow$ the event that e_i picked in the ith step is not in the mincut.

- Probability that e_1 in $C = \frac{k}{nk/2} = \frac{2}{n}$. $\Rightarrow Pr[\eta_1] = 1 - \frac{2}{n}$.
- Assuming η_1 occurs, during the second step, the number of edges in G_1 is $\frac{k(n-1)}{2}$.
  Thus, Probability that e_2 in $C = \frac{2}{n-1}$, and
  $Pr[\eta_2|\eta_1] = 1 - \frac{2}{n-1}$.
- Proceeding similarly, we have

$$Pr[\eta_i| \cap_{j=1}^{i-1} \eta_j] = 1 - \frac{2}{n-i+1} = \frac{n-i-1}{n-i+1}$$

.

Thus, in the worst case,

Probability that no edge in C is picked =

$$Pr[\cap_{i=1}^{n-2}\eta_i] = \Pi_{i=1}^{n-2}(\frac{n-i-1}{n-i+1}) = \frac{2}{n(n-1)}$$

Thus, we have

Probability of discovering a particular mincut is larger than $\frac{2}{n^2}$.

Probabilistic Conclusion

Now, if we repeat the attempt $\frac{n^2}{2}$ times, then the probability of not getting a mincut is

$$(1-\frac{2}{n^2})^{\frac{n^2}{2}} = \frac{1}{e}.$$

Result

By spending $O(n^4)$ time, we can reduce the failure probability from $1-\frac{2}{n^2}$ to a reasonably small constant value $\frac{1}{e}$.

Improving the Algorithm

Why MINCUT has high time complexity

The probability of getting a mincut is $\frac{2}{n^2}$.
So, in order to get a correct solution with reasonable probability, we need to repeat the process at least $\Omega(n^2)$ times.

Remedy

Revise the contract process to increase the probability of getting a mincut.

Important Observations and Suggestions

Consider a mincut $\mathcal{C}$.

Observation:

- Initial contractions are very unlikely to involve edges from $\mathcal{C}$.
- At the end, when the number of edges apart from $\mathcal{C}$ is less.
- Thus, the probability of choosing an edge from $\mathcal{C}$ increases.

Suggestions:

- Repeat the earlier fast algorithm till the number of edges is not very less.
- After that use some slower algorithm that guarantees high probability of not choosing member of $\mathcal{C}$.
- The first stage also guarantees that the second stage will not take much time.

Algorithm FASTCUT

FASTCUT($G = (V, E)$)
Input: $G \rightarrow$ A multigraph
begin
  $n \leftarrow |V(G)|$
  if $n \leq 6$ **then**
    Compute (by brute force), the
    mincut of G and return the cut.
  else
    $t \leftarrow \lceil 1 + \frac{n}{\sqrt{2}} \rceil$
    $H_1 \leftarrow$ CONTRACT(G, t)
    $H_2 \leftarrow$ CONTRACT(G, t)
    $X_1 \leftarrow$ FASTCUT(H_1)
    $X_2 \leftarrow$ FASTCUT(H_2)
    return minimum cut out of X_1 and X_2
end.

CONTRACT($G = (V, E), t$)
begin
  while$|V| > t$ **do**
    Pick a random edge $e \in E$.
    $G \leftarrow G/e$
  return G
end.

Theme of the Algorithm

- We considered two contracted instances, each of size t; computed their mincuts, and have chosen the one having smaller size.
- The recursion stopped hen $n \leq 6$ since at that point of time $t = 1 + \frac{n}{\sqrt{2}}$ becomes greater than n.
- The computation can be viewed as a binary computation tree. The height of this tree is $\log_{\sqrt{2}} n = 2 \log n$.
  The number of leaves is $2^{2 \log n} = n^2$.
- Thus, we have generated n^2 different min-cuts, as we did in the earlier method.

Difference with the Earlier Method

The difference with the earlier method is that,

- In the earlier method, we created a tree of height 1 with $O(n^2)$ leaves. Each leave correspond to a mincut.
- In this method, we have created a tree of height $2 \log n$, and with $O(n^2)$ leaves.
- But, here we have shared the computation of generating several mincuts.

Time and Space Complexities

Theorem

FASTCUT runs in $O(n^2 \log n)$ time and $O(n^2)$ space.

Proof: Time Complexity

- H_1 and H_2 can be obtained in $O(n^2)$ time, since
  - Each contraction step needs $O(n)$ time.
  - Contraction process continues $O(n)$ times.
- Thus, we have the recursion

$$T(n) = 2T(\frac{n}{\sqrt{2}}) + O(n^2) = O(n^2 \log n).$$

Space:

At d-th level of recursion, the graph has $O(\frac{n}{2^{d/2}})$ vertices.
Thus, the total space is

$$O(\sum_{d=0}^{\infty} \frac{n^2}{2^d}) = O(n^2).$$

Correctness Probability

Let $\mathcal{C}$ be a mincut.

$\eta_i \rightarrow$ In the i-th stage of contraction, e_i is not in $\mathcal{C}$.

Result:

$\mathcal{P} = Pr(H_1 \text{ does not contain any edge of } \mathcal{C}) \geq \frac{1}{2}$.

Proof: The probability $\mathcal{P} = Pr(\eta_0 \cap \eta_1 \cap \ldots \cap \eta_{n-t})$

$$
\begin{aligned}
&= \Pi_{i=0}^{n-t-1}(1 - \frac{2}{n-i}) = \Pi_{i=0}^{n-t-1}\frac{n-i-2}{n-i} \\
&= \frac{n-2}{n} \times \frac{n-3}{n-1} \times \frac{n-4}{n-2} \times \ldots \frac{t}{t+2} \times \frac{t-1}{t+1} \\
&= \frac{t.(t-1)}{n.(n-1)} \\
&= \frac{\lceil 1 + n/\sqrt{2} \rceil (\lceil 1 + n/\sqrt{2} \rceil - 1)}{n(n-1)} \\
&\geq \frac{1}{2}
\end{aligned}
$$

Correctness Probability

Theorem

FASTCUT finds a mincut with probability $\Omega(\frac{1}{\log n})$

Proof:

$$\begin{aligned} P(n) \rightarrow & \; Pr(\text{algorithm succeeds on a graph of } n \text{ vertices}) \\ = & \; \Pi_{i=1}^{2} Pr(H_i \text{ does not contract any edge of } \mathcal{C}) \\ = & \; Pr(H_1 \text{ does not contract any edge of } \mathcal{C}) \\ = & \; \frac{1}{2} \times P(\frac{n}{\sqrt{2}}) \end{aligned}$$

Thus, Probability to fail on $H_1 \leq 1 - \frac{1}{2}P(\frac{n}{\sqrt{2}})$

Probability to fail on both H_1 and $H_2 \leq (1 - \frac{1}{2}P(\frac{n}{\sqrt{2}}))^2$

Correctness Probability

$$P(n) \geq 1 - (1 - \frac{1}{2}P(\frac{n}{\sqrt{2}}))^2 = P(\frac{n}{\sqrt{2}}) - \frac{1}{4}(P(\frac{n}{\sqrt{2}}))^2$$

This, on simplification gives

$$P(n) \geq \frac{2 \log 2}{\log n}$$

[See Motwani and Raghavan, *Randomized Algorithms*, pages 292-295.]

Types of Randomized Algorithms

Definition

Las Vegas: a randomized algorithm that always returns a correct result. But the running time may vary between executions.

Example: Randomized QUICKSORT Algorithm

Definition

Monte Carlo: a randomized algorithm that terminates in polynomial time, but might produce erroneous result.

Example: Randomized MINCUT Algorithm

Complexity Classes

RP

The class Randomized Polynomial Time (RP) consists of all languages $\mathcal{L}$ that have randomized algorithm A with worst case polynomial running time, and if for any input $x \in \Sigma^*$

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

RP algorithm is Monte Carlo, but the mistake can only be if $x \in L$

Co-RP

A class of Monte Carlo algorithms that can make mistake if $x \notin L$.

A problem in $RP \cap Co - RP$
$\Rightarrow$ It has an algorithm that does not make any mistake.
In other words, it has a *Las Vegas* Algorithm.

Complexity Classes

Zero Error Probabilistic Polynomial Time Algorithm (ZPP)

The class of languages that have Las Vegas algorithm in expected polynmial time.

Example: QUICKSORT or Minimum Enclosing Circle Problem

Probabilistic Polynomial Time Algorithm (PP)

The class of languages that have randomized algorithm A with worst case polynomial execution time, and for any input $x \in \Sigma^*$,

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

An algorithm in PP is not very useful

Las Vegas vs Monte-Carlo

- Las Vegas $\longrightarrow$ Monte-Carlo
- A - Las Vegas algo with $E[T_A(x)] \leq poly(n)$ for every $x \in \Sigma^*$.
- By incorporating a counter which counts every elementary step into A and stopping after, say, $4poly(n)$ steps, one gets a polynomial time Monte-Carlo algorithm B with a guaranteed confidence of at least $\frac{3}{4}$.

Las Vegas vs Monte-Carlo

- Monte-Carlo $\rightarrow$ Las Vegas
- A - Monte-Carlo algorithm with $\theta_1 = poly_1(n)$ execution time, success probability $\frac{1}{\theta_2}$, where $\theta_2 = poly_2(n)$.
  Suppose correctness of output can be verified in $\theta_3 = poly_3(n)$ time.
- Run the algorithm A repeatedly until it gives a correct solution.
  Let it need k trials. k follows geometric distribution, with $E(k) = \theta_2$.
  Thus, we get a Las Vegas algo with expected time $(\theta_1 \times \theta_3) \times \theta_2$, which is a polynomal in n.

Conclusions

- Employing randomness leads to improved simplicity and improved efficiency in solving the problem.
- It assumes the availability of a perfect source of independent and unbiased random bits.
- Access to truly unbiased and independent sequence of random bits is expensive.
  So, it should be considered as an expensive resource like time and space.
  Thus, one should aim to minimize the use of randomness to the extent possible.
- Assumes efficient realizability of any rational bias. However, this assumption introduces error and increases the work and the required number of random bits.
- There are ways to reduce the randomness from several algorithms while maintaining the efficiency nearly the same.