Introduction to Randomized Algorithms

Arijit Bishnu
(arijit@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

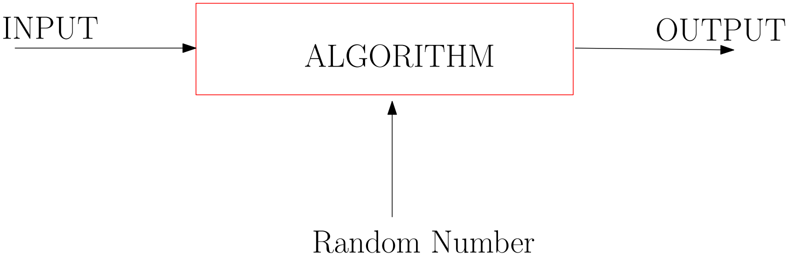
Given a computational problem,

- it may be difficult to design an algorithm with good running time, or
- the running time of an algorithm may be quite high.

Remedies

- Efficient heuristics,
- Approximation algorithms,
- Randomized algorithms

Randomized Algorithm



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 Paradigm

Instead of making a guaranteed good choice, make a random choice and hope that it is good. This helps because guaranteeing a good choice becomes difficult sometimes.

Advantage of Randomized Algorithm

The Paradigm

Instead of making a guaranteed good choice, make a random choice and hope that it is good. This helps because guaranteeing a good choice becomes difficult sometimes.

Randomized Algorithms

make random choices. The expected running time depends on the random choices, not on any input distribution.

Advantage of Randomized Algorithm

The Paradigm

Instead of making a guaranteed good choice, make a random choice and hope that it is good. This helps because guaranteeing a good choice becomes difficult sometimes.

Randomized Algorithms

make random choices. The expected running time depends on the random choices, not on any input distribution.

Average Case Analysis

analyzes the expected running time of deterministic algorithms assuming a suitable random distribution on the input.

Pros and Cons of Randomized Algorithms

Pros

Pros and Cons of Randomized Algorithms

Pros

- Making a random choice is fast.

Pros and Cons of Randomized Algorithms

Pros

- Making a random choice is fast.
- An adversary is powerless; randomized algorithms have no worst case inputs.

Pros and Cons of Randomized Algorithms

Pros

- Making a random choice is fast.
- An adversary is powerless; randomized algorithms have no worst case inputs.
- Randomized algorithms are often simpler and faster than their deterministic counterparts.

Pros and Cons of Randomized Algorithms

Pros

- Making a random choice is fast.
- An adversary is powerless; randomized algorithms have no worst case inputs.
- Randomized algorithms are often simpler and faster than their deterministic counterparts.

Cons

Pros and Cons of Randomized Algorithms

Pros

- Making a random choice is fast.
- An adversary is powerless; randomized algorithms have no worst case inputs.
- Randomized algorithms are often simpler and faster than their deterministic counterparts.

Cons

- In the worst case, a randomized algorithm may be very slow.

Pros and Cons of Randomized Algorithms

Pros

- Making a random choice is fast.
- An adversary is powerless; randomized algorithms have no worst case inputs.
- Randomized algorithms are often simpler and faster than their deterministic counterparts.

Cons

- In the worst case, a randomized algorithm may be very slow.
- 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.

Pros and Cons of Randomized Algorithms

Pros

- Making a random choice is fast.
- An adversary is powerless; randomized algorithms have no worst case inputs.
- Randomized algorithms are often simpler and faster than their deterministic counterparts.

Cons

- In the worst case, a randomized algorithm may be very slow.
- 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.
- Getting true random numbers is almost impossible.

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

Randomized Quick Sort

Deterministic Quick Sort

The Problem:

Given an array $A[1 \dots n]$ 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 $A[r]$ in $A[s+1]$, 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: One execution of 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 $\frac{1}{2}$.

An Useful Result from Probability

- Let p be the probability of success and $1 - p$ be the probability of failure of a random experiment.

An Useful Result from Probability

- Let p be the probability of success and $1 - p$ be the probability of failure of a random experiment.
- If we continue the random experiment till we get success, what is the expected number of experiments we need to perform?

An Useful Result from Probability

- Let p be the probability of success and $1 - p$ be the probability of failure of a random experiment.
- If we continue the random experiment till we get success, what is the expected number of experiments we need to perform?
- Let X: random variable that equals the number of experiments performed.

An Useful Result from Probability

- Let p be the probability of success and $1 - p$ be the probability of failure of a random experiment.
- If we continue the random experiment till we get success, what is the expected number of experiments we need to perform?
- Let X: random variable that equals the number of experiments performed.
- For the process to perform exactly j experiments, the first $j - 1$ experiments should be failures and the j-th one should be a success. So, we have $Pr[X = j] = (1 - p)^{(j-1)} \cdot p$.

An Useful Result from Probability

- Let p be the probability of success and $1 - p$ be the probability of failure of a random experiment.
- If we continue the random experiment till we get success, what is the expected number of experiments we need to perform?
- Let X: random variable that equals the number of experiments performed.
- For the process to perform exactly j experiments, the first $j - 1$ experiments should be failures and the j-th one should be a success. So, we have $Pr[X = j] = (1 - p)^{(j-1)} \cdot p$.
- So, the expectation of X, $E[X] = \sum_{j=0}^{\infty} j \cdot Pr[X = j] = \frac{1}{p}$.

Implication in Our Case

- The expected number of times Step 2 needs to be repeated to get a central splitter (success) is 2 as the corresponding success probability is $\frac{1}{2}$.
- Thus, the expected time complexity of Step 2 is $O(n)$

Analysis of RandQSORT

Time Complexity

- The expected running time for the algorithm on a set A, excluding the time spent on recursive calls, is $O(|A|)$.

Analysis of RandQSORT

Time Complexity

- The expected running time for the algorithm on a set A, excluding the time spent on recursive calls, is $O(|A|)$.
- Worst case size of each partition in j-th level of recursion is $n \cdot (\frac{3}{4})^j$, So, the expected time spent excluding recursive calls is $O(n \cdot (\frac{3}{4})^j)$ for each partition.

Analysis of RandQSORT

Time Complexity

- The expected running time for the algorithm on a set A, excluding the time spent on recursive calls, is $O(|A|)$.
- Worst case size of each partition in j-th level of recursion is $n \cdot (\frac{3}{4})^j$, So, the expected time spent excluding recursive calls is $O(n \cdot (\frac{3}{4})^j)$ for each partition.
- The number of partitions of size $n \cdot (\frac{3}{4})^j$ is $O((\frac{4}{3})^j)$.

Analysis of RandQSORT

Time Complexity

- The expected running time for the algorithm on a set A, excluding the time spent on recursive calls, is $O(|A|)$.
- Worst case size of each partition in j-th level of recursion is $n \cdot (\frac{3}{4})^j$, So, the expected time spent excluding recursive calls is $O(n \cdot (\frac{3}{4})^j)$ for each partition.
- The number of partitions of size $n \cdot (\frac{3}{4})^j$ is $O((\frac{4}{3})^j)$.
- By linearity of expectations, the expected time for all partitions of size $n \cdot (\frac{3}{4})^j$ is $O(n)$.

Analysis of RandQSORT

Time Complexity

- The expected running time for the algorithm on a set A, excluding the time spent on recursive calls, is $O(|A|)$.
- Worst case size of each partition in j-th level of recursion is $n \cdot (\frac{3}{4})^j$, So, the expected time spent excluding recursive calls is $O(n \cdot (\frac{3}{4})^j)$ for each partition.
- The number of partitions of size $n \cdot (\frac{3}{4})^j$ is $O((\frac{4}{3})^j)$.
- By linearity of expectations, the expected time for all partitions of size $n \cdot (\frac{3}{4})^j$ is $O(n)$.
- Number of levels of recursion $= \log_{\frac{4}{3}} n = O(\log n)$.

Analysis of RandQSORT

Time Complexity

- The expected running time for the algorithm on a set A, excluding the time spent on recursive calls, is $O(|A|)$.
- Worst case size of each partition in j-th level of recursion is $n \cdot (\frac{3}{4})^j$, So, the expected time spent excluding recursive calls is $O(n \cdot (\frac{3}{4})^j)$ for each partition.
- The number of partitions of size $n \cdot (\frac{3}{4})^j$ is $O((\frac{4}{3})^j)$.
- By linearity of expectations, the expected time for all partitions of size $n \cdot (\frac{3}{4})^j$ is $O(n)$.
- Number of levels of recursion $= \log_{\frac{4}{3}} n = O(\log n)$.
- Thus, the expected running time is $O(n \log n)$.

Minimum Enclosing Disk Problem (MED)

Minimum 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).

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 MED of P_i.

A Simple Randomized Algorithm

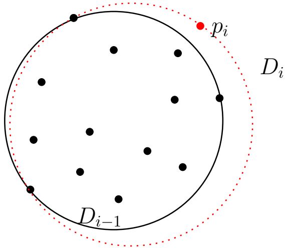
We generate a random permutation of the points in P.

Notations:

- $P_i = \{p_1, p_2, \ldots, p_i\}$. • D_i = the MED of P_i.

Result (Believe this for now) ☹

- 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.



A Simple Randomized Algorithm

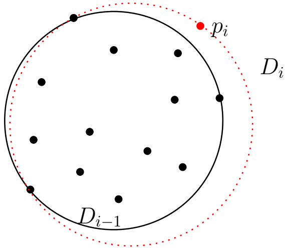
We generate a random permutation of the points in P.

Notations:

- $P_i = \{p_1, p_2, \ldots, p_i\}$. • D_i = the MED of P_i.

Result (Believe this for now) ☹

- 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.



An Implication

The above result implies an incremental algorithm.

The Idea

A Different Recursive Idea

- When we encounter a point $p_i \notin D_{i-1}$, we know that p_i is constrained to lie on D_i,

The Idea

A Different Recursive Idea

- When we encounter a point $p_i \notin D_{i-1}$, we know that p_i is constrained to lie on D_i,
- This leads to a different version of the original problem where we need to find a MED of a set of points with p_i constrained to lie on the boundary.

The Idea

A Different Recursive Idea

- When we encounter a point $p_i \notin D_{i-1}$, we know that p_i is constrained to lie on D_i,
- This leads to a different version of the original problem where we need to find a MED of a set of points with p_i constrained to lie on the boundary.
- In this recursion, if we encounter a point that lies outside the current disk, we recurse on a subproblem where two points are constrained to lie on the boundary.

The Idea

A Different Recursive Idea

- When we encounter a point $p_i \notin D_{i-1}$, we know that p_i is constrained to lie on D_i,
- This leads to a different version of the original problem where we need to find a MED of a set of points with p_i constrained to lie on the boundary.
- In this recursion, if we encounter a point that lies outside the current disk, we recurse on a subproblem where two points are constrained to lie on the boundary.
- How long can this go on?

The Idea

A Different Recursive Idea

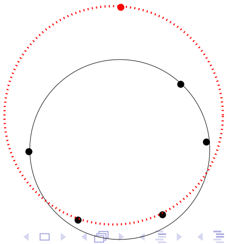
- When we encounter a point $p_i \notin D_{i-1}$, we know that p_i is constrained to lie on D_i,
- This leads to a different version of the original problem where we need to find a MED of a set of points with p_i constrained to lie on the boundary.
- In this recursion, if we encounter a point that lies outside the current disk, we recurse on a subproblem where two points are constrained to lie on the boundary.
- How long can this go on?
- In the next level, we have three points constrained to lie on the boundary and that defines a unique disk.

Algorithm MINIDISC(P)

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

1. Compute a random permutation of $P = \{p_1, p_2, \ldots, p_n\}$.
2. Let D_2 be the MED 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** $D_i =$ MINIDISKWITH1POINT($\{p_1, p_2, \ldots, p_i\}, p_i$)
7. **return** D_n.

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



Algorithm MINIDISCWITH1POINT(P, q)

Idea: Incrementally add points from P one by one and compute the MED 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: MED for P with q on the boundary.

1. Compute a random permutation of $P = \{p_1, p_2, \dots, p_n\}$.
2. Let D_1 be the MED 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$(\{p_1, p_2, \dots, p_j\}, p_j, q)$
7. **return** D_n.

Algorithm MINIDISCWITH2POINTS(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 MED 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.

Time Complexity

Worst Case Time Complexity

With the nested recursion that we have, the worst case time complexity is $O(n^3)$.

Time Complexity

Worst Case Time Complexity

With the nested recursion that we have, the worst case time complexity is $O(n^3)$.

Expected case:

Time Complexity

Worst Case Time Complexity

With the nested recursion that we have, the worst case time complexity is $O(n^3)$.

Expected case:

- MINIDISKWITH2POINTS needs $O(n)$ time.

Time Complexity

Worst Case Time Complexity

With the nested recursion that we have, the worst case time complexity is $O(n^3)$.

Expected case:

- MINIDISKWITH2POINTS needs $O(n)$ time.
- MINIDISKWITH1POINT needs $O(n)$ time if

Time Complexity

Worst Case Time Complexity

With the nested recursion that we have, the worst case time complexity is $O(n^3)$.

Expected case:

- MINIDISKWITH2POINTS needs $O(n)$ time.
- MINIDISKWITH1POINT needs $O(n)$ time if
  - **we do not consider the time taken in the call of the routine** MINIDISKWITH2POINTS.

Time Complexity

Worst Case Time Complexity

With the nested recursion that we have, the worst case time complexity is $O(n^3)$.

Expected case:

- MINIDISKWITH2POINTS needs $O(n)$ time.
- MINIDISKWITH1POINT 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 MED 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 disk 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 MINIDISKWITH1POINT.

Expected Case Time Complexity

Observation:

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

Expected Running time of MINIDISKWITH1POINT

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

Similarly, we have

Expected Running time of MINIDISK

$O(n)$

The Pending Proof

Result (We are going to prove this now) ☺

- 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.

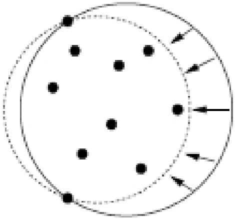
The Pending Proof

Result (We are going to prove this now) ☺

- 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.

Claim 1

For a set of points P in general position, the MED has at least three points on its boundary or it has two points forming the diameter of the disk. If there are three points, then they subdivide the circle bounding the disk into arcs of angle at most π.

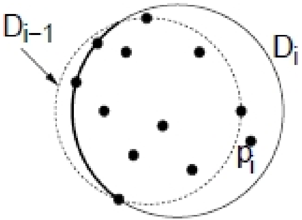


The Proof Continued

The Proof Continued

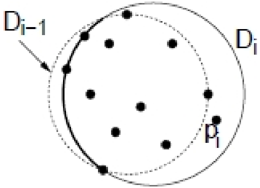
Claim 2

Given a disk of radius r_1 and a circle of radius r_2, with $r_1 < r_2$, the intersection of the disk with the circle is an arc of angle less than π.



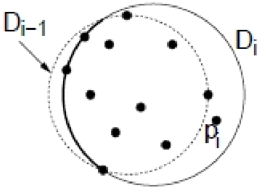
The Proof Continued

- Suppose, for a contradiction, p_i is not on the boundary of D_i.



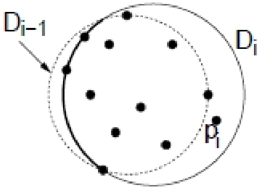
The Proof Continued

- Suppose, for a contradiction, p_i is not on the boundary of D_i.
- Let r_1 = radius of D_{i-1} and r_2 = radius of D_i.



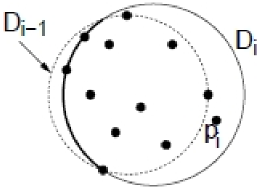
The Proof Continued

- Suppose, for a contradiction, p_i is not on the boundary of D_i.
- Let r_1 = radius of D_{i-1} and r_2 = radius of D_i.
- Using Claim 2, D_i intersects D_{i-1} in an arc of angle less than π.



The Proof Continued

- Suppose, for a contradiction, p_i is not on the boundary of D_i.
- Let r_1 = radius of D_{i-1} and r_2 = radius of D_i.
- Using Claim 2, D_i intersects D_{i-1} in an arc of angle less than π.
- Since $p_i \notin D_i$, points defining D_i should lie in the said arc implying an angle more than π. We get a contradiction.



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.

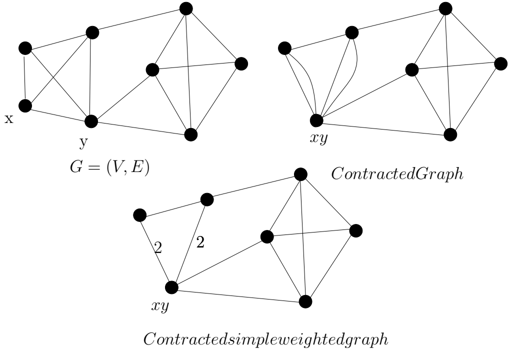
Applications:

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

A Simple Randomized Algorithm

Contraction 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.



Results on Contraction 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.

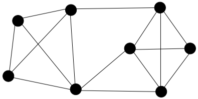
Theorem

Time Complexity: $O(n^2)$

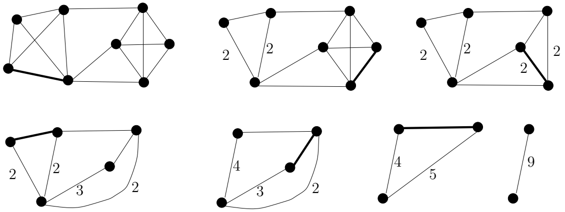
A Trivial Observation: The algorithm outputs a cut whose size is *no smaller than the mincut*.

Demonstration of the Algorithm

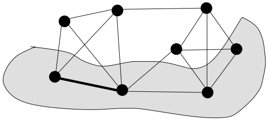
The given graph:



Stages of Contraction:



The corresponding output:



Quality Analysis: How good is the solution?

Result 3: Lower bounding $|E|$

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

Quality Analysis: How good is the solution?

Result 3: Lower bounding $|E|$

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

Proof

Quality Analysis: How good is the solution?

Result 3: Lower bounding $|E|$

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

Proof

- If any node v has degree less than k, then the $\mathrm{cut}(\{v\}, V - \{v\})$ will have size less than k.

Quality Analysis: How good is the solution?

Result 3: Lower bounding $|E|$

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

Proof

- If any node v has degree less than k, then the $\text{cut}(\{v\}, V - \{v\})$ will have size less than k.
- This contradicts the fact that (A, B) is a global min-cut.

Quality Analysis: How good is the solution?

Result 3: Lower bounding $|E|$

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

Proof

- If any node v has degree less than k, then the $\text{cut}(\{v\}, V - \{v\})$ will have size less than k.
- This contradicts the fact that (A, B) is a global min-cut.
- Thus, every node in G has degree at least k. So, $|E| \geq \frac{1}{2}kn$.

Quality Analysis: How good is the solution?

Result 3: Lower bounding $|E|$

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

Proof

- If any node v has degree less than k, then the $\mathrm{cut}(\{v\}, V - \{v\})$ will have size less than k.
- This contradicts the fact that (A, B) is a global min-cut.
- Thus, every node in G has degree at least k. So, $|E| \geq \frac{1}{2}kn$.

So, the probability that an edge in F is contracted is at most $\frac{k}{(kn)/2} = \frac{2}{n}$

But, we don't know the min cut.

Summing up: Result 4

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

Summing up: Result 4

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

Continuing Contraction

Summing up: Result 4

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

Continuing Contraction

- After i iterations, there are $n - i$ supernodes in the current graph G' and suppose no edge in the cut F has been contracted.

Summing up: Result 4

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

Continuing Contraction

- After i iterations, there are $n - i$ supernodes in the current graph G' and suppose no edge in the cut F has been contracted.
- Every cut of G' is a cut of G. So, there are at least k edges incident on every supernode of G'.

Summing up: Result 4

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

Continuing Contraction

- After i iterations, there are $n - i$ supernodes in the current graph G' and suppose no edge in the cut F has been contracted.
- Every cut of G' is a cut of G. So, there are at least k edges incident on every supernode of G'.
- Thus, G' has at least $\frac{1}{2}k(n - i)$ edges.

Summing up: Result 4

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

Continuing Contraction

- After i iterations, there are $n - i$ supernodes in the current graph G' and suppose no edge in the cut F has been contracted.
- Every cut of G' is a cut of G. So, there are at least k edges incident on every supernode of G'.
- Thus, G' has at least $\frac{1}{2}k(n-i)$ edges.
- So, the probability that an edge in F is contracted in iteration $i + 1$ is at most $\frac{k}{\frac{1}{2}k(n-i)} = \frac{2}{n-i}$.

Digression into Probability

Conditional Probability

The conditional probability of X given Y is

$$Pr[X = x \mid Y = y] = \frac{Pr[(X = x) \cap (Y = y)]}{Pr[Y = y]}$$

Digression into Probability

Conditional Probability

The conditional probability of X given Y is

$$Pr[X = x \mid Y = y] = \frac{Pr[(X = x) \cap (Y = y)]}{Pr[Y = y]}$$

An Equivalent Statement

$$Pr[(X = x) \cap (Y = y)] = Pr[X = x \mid Y = y] \cdot Pr[Y = y]$$

Digression into Probability

Conditional Probability

The conditional probability of X given Y is

$$Pr[X = x \mid Y = y] = \frac{Pr[(X = x) \cap (Y = y)]}{Pr[Y = y]}$$

An Equivalent Statement

$$Pr[(X = x) \cap (Y = y)] = Pr[X = x \mid Y = y] \cdot Pr[Y = y]$$

Independent Events

Two events X and Y are independent, if $Pr[(X = x) \cap (Y = y)] = Pr[X = x] \cdot Pr[Y = y]$. In particular, if X and Y are independent, then

$$Pr[X = x \mid Y = y] = Pr[X = x]$$

A Result that we need on Intersection of events

Let $\eta_1, \eta_2, \ldots, \eta_n$ be n events not necessarily independent. Then,

$$Pr[\cap_{i=1}^{n} \eta_i] = Pr[\eta_1] \cdot Pr[\eta_2 \mid \eta_1] \cdot Pr[\eta_3 \mid \eta_1 \cap \eta_2] \cdots Pr[\eta_n \mid \eta_1 \cap \ldots \cap \eta_{n-1}].$$

The proof is by induction on n.

Correctness

Theorem

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

Proof:

The correct cut(A, B) will be returned by MINCUT if no edge of F is contracted in any of the iterations $1, 2, \ldots, n-2$.

Let $\eta_i \Rightarrow$ the event that an edge of F is not contracted in the ith iteration.

We have already shown that

- $Pr[\eta_1] \geq 1 - \frac{2}{n}$.
- $Pr[\eta_{i+1} \mid \eta_1 \cap \eta_2 \cap \cdots \cap \eta_i] \geq 1 - \frac{2}{n-i}$

Lower Bounding the Intersection of Events

We want to lower bound $Pr[\eta_1 \cap \cdots \cap \eta_{n-2}]$.
We use the earlier result

$$Pr[\cap_{i=1}^{n} \eta_i] = Pr[\eta_1] \cdot Pr[\eta_2 \mid \eta_1] \cdot Pr[\eta_3 \mid \eta_1 \cap \eta_2] \cdots Pr[\eta_n \mid \eta_1 \cap \ldots \cap \eta_{n-1}].$$

So, we have $Pr[\eta_1] \cdot Pr[\eta_1 \mid \eta_2] \cdots Pr[\eta_{n-2} \mid \eta_1 \cap \eta_2 \cdots \cap \eta_{n-3}]$
$\geq (1 - \frac{2}{n}) \left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{n-i}\right) \cdots (1 - \frac{2}{3})$
$= \binom{n}{2}^{-1}$

Bounding the Error Probability

- We know that a single run of the contraction algorithm fails to find a global min-cut with probability at most $1 - \frac{1}{\binom{n}{2}} \approx 1$.

Bounding the Error Probability

- We know that a single run of the contraction algorithm fails to find a global min-cut with probability at most $1 - \frac{1}{\binom{n}{2}} \approx 1$.
- We can amplify our success probability by repeatedly running the algorithm with independent random choices and taking the best cut.

Bounding the Error Probability

- We know that a single run of the contraction algorithm fails to find a global min-cut with probability at most $1 - \frac{1}{\binom{n}{2}} \approx 1$.
- We can amplify our success probability by repeatedly running the algorithm with independent random choices and taking the best cut.
- If we run the algorithm $\binom{n}{2}$ times, then the probability that we fail to find a global min-cut in any run is at most

$$\left(1 - \frac{1}{\binom{n}{2}}\right)^{\binom{n}{2}} \leq \frac{1}{e}.$$

Bounding the Error Probability

- We know that a single run of the contraction algorithm fails to find a global min-cut with probability at most $1 - \frac{1}{\binom{n}{2}} \approx 1$.
- We can amplify our success probability by repeatedly running the algorithm with independent random choices and taking the best cut.
- If we run the algorithm $\binom{n}{2}$ times, then the probability that we fail to find a global min-cut in any run is at most

$$\left(1 - \frac{1}{\binom{n}{2}}\right)^{\binom{n}{2}} \leq \frac{1}{e}.$$

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 $\frac{1}{e}$.

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.
- There are ways to reduce the randomness from several algorithms while maintaining the efficiency nearly the same.