COMP 7270/7276: “Advanced Algorithms”

Lecture 20: Randomized Algorithms

Bo Liu

Last Compiled: April 24, 2017

Probability Refresher

- Expectation of random variable:

$$\mathbb{E}[X] = \sum_r r\mathbb{P}[X = r]$$

- Linearity of expectation:

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

- Conditional Probability: For arbitrary events A and B,

$$\mathbb{P}[A|B] = \mathbb{P}[A \cap B] / \mathbb{P}[B]$$

and $\mathbb{P}[\cap_{i=1}^{n} A_i] = \mathbb{P}[A_1]\mathbb{P}[A_2|A_1] \ldots \mathbb{P}\left[A_n | \cap_{i=1}^{n-1} A_i\right]$

Outline

Randomized Algorithms

Quicksort

Karger's Randomized Min-Cut Algorithm

Deterministic Algorithms

- The algorithms we've seen so far have been deterministic.
- We want to aim for properties like
  - Good **worst-case** behavior.
  - Getting **exact** solutions.
- Much of our complexity arises from the fact that there is little flexibility here.
- Often find complex algorithms with nuanced correctness proofs.

Randomized Algorithms

- A **randomized algorithm** is an algorithm that incorporates randomness as part of its operation.
- Often aim for properties like
  - Good **average-case** behavior.
  - Getting exact answers **with high probability**.
  - Getting answers that are **close to the right** answer.
- Often find very simple algorithms with dense but clean analyses.

Randomized Algorithms

Two types of randomized algorithms:

- **Las Vegas** algorithms: the output is deterministic (and correct - it will always return a sorted list) but that the runtime is variable and affected by the randomization.
- **Monte Carlo** algorithms: The runtime of this algorithm is deterministic, but it is not guaranteed to always return the correct answer. Instead, it has a (hopefully small) probability of returning an incorrect answer.

Where We're Going

Motivating examples:

- Quicksort are **Las Vegas** algorithms: they always find the right answer, but might take a while to do so.
- Karger's algorithm is a **Monte Carlo** algorithm: it might not always find the right answer, but has dependable performance.

Outline

Quicksort

Problem: Sort an array of distinct values $X = [x_1, \ldots, x_n]$

Algorithm

1. *Pick a pivot $x \in X$ at random from the array*
2. *Construct new arrays $Y = [y_1, \ldots, y_k]$, $Z = [z_1, \ldots, z_{n-k-1}]$ where*

$$y < x < z \text{ for all } y \in Y, z \in Z$$

3. *Recursively sort Y and Z to get Y' and Z'*
4. *Return the array that concatenates Y', x, and Z'*

What's the expected number of comparisons performed in this algorithm?

Probability two items are compared

Lemma

Let a and b be the i-th and j-th smallest element of X where $i < j$.

$$\Pr[a \text{ is compared to } b] = \frac{2}{j - i + 1}$$

Probability two items are compared

Lemma

Let a and b be the i-th and j-th smallest element of X where $i < j$.

$$\Pr[a \textit{ is compared to } b] = \frac{2}{j - i + 1}$$

Proof.

1. Consider $S = \{x \in X : a \leq x \leq b\}$

$\square$

Probability two items are compared

Lemma

Let a and b be the i-th and j-th smallest element of X where $i < j$.

$$\Pr[a \text{ is compared to } b] = \frac{2}{j - i + 1}$$

Proof.

1. Consider $S = \{x \in X : a \leq x \leq b\}$
2. a and b are compared iff the first pivot chosen from S is either a or b

□

Probability two items are compared

Lemma

Let a and b be the i-th and j-th smallest element of X where $i < j$.

$$\Pr[a \text{ is compared to } b] = \frac{2}{j - i + 1}$$

Proof.

1. Consider $S = \{x \in X : a \leq x \leq b\}$
2. a and b are compared iff the first pivot chosen from S is either a or b
3. Elements of S are equally likely to be chosen as a pivot, so

$$\Pr[a \text{ is compared to } b] = \frac{2}{|S|} = \frac{2}{j - i + 1}$$

□

Expected Number of Comparisons

Lemma

Expected number of comparisons performs is $O(n \log n)$.

Proof.

1. Let $Z_{ij} = 1$ if the i-th smallest element is compared to j-th smallest element and $Z_{ij} = 0$ otherwise.
2. Number of comparisons: $\sum_{1 \le i < j \le n} Z_{ij}$
3. Expected number of comparisons:

$$\mathbb{E}\left[\sum_{1 \le i < j \le n} Z_{ij}\right] = \sum_{1 \le i < j \le n} \mathbb{E}\left[Z_{ij}\right] = \sum_{1 \le i < j \le n} \frac{2}{j - i + 1} = \sum_{j=2}^{n} \sum_{k=2}^{j} \frac{2}{k}$$

4. Because $H_n = 1 + 1/2 + 1/3 + \ldots + 1/n = O(\log n)$,

$$\mathbb{E}\left[\sum_{1 \le i < j \le n} Z_{ij}\right] = O(n \log n)$$

Outline

Min-Cut Problem

Given an unweighted, multi-graph $G = (V, E)$, we want to partition V into V_1 and V_2 such that $|E \cap (V_1 \times V_2)|$ is minimized.

Algorithm

- *Contract a random edge $e = (u, v)$ and remove self-loops but not multi-edges*
- *Repeat until there are only 2 vertices remaining.*
- *Output the number of remaining edges.*

Let $|V| = n$ and $|E| = m$.

Karger's Algorithm

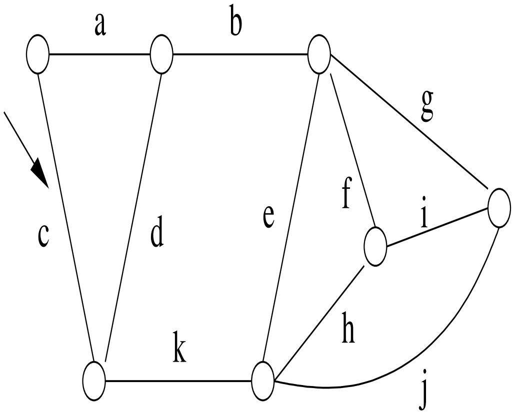
Given an edge (u, v) in a multigraph, we can **contract** u and v as follows:

- Delete all edges between u and v.
- Replace u and v with a new supernode uv.
- Replace all edges incident to u or v with edges incident to the supernode uv.

Karger's algorithm is as follows:

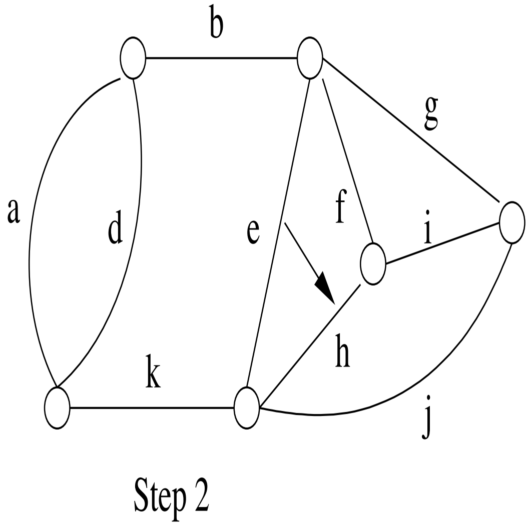
- If there are exactly two nodes left, stop. The edges crossing those nodes form a cut.
- Otherwise, pick a random edge, contract it, then repeat.

Karger's Algorithm

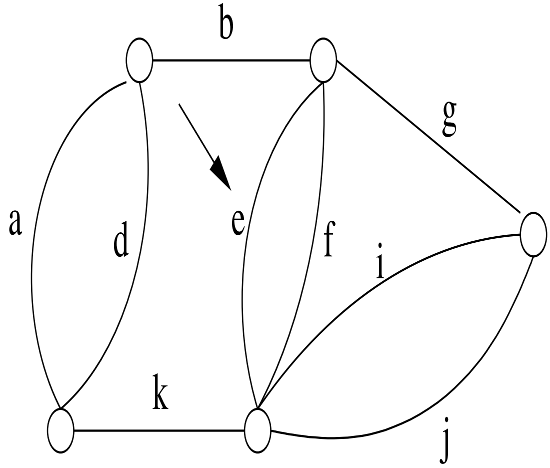


Step 1

Karger's Algorithm

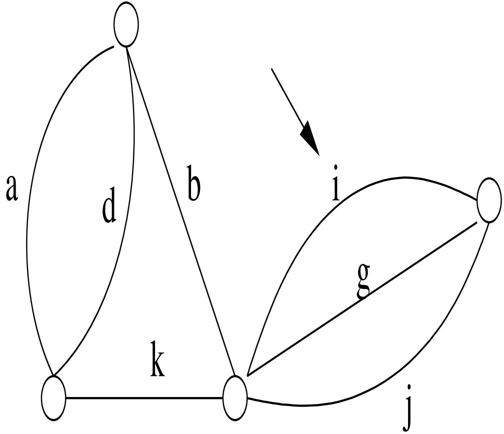


Karger's Algorithm



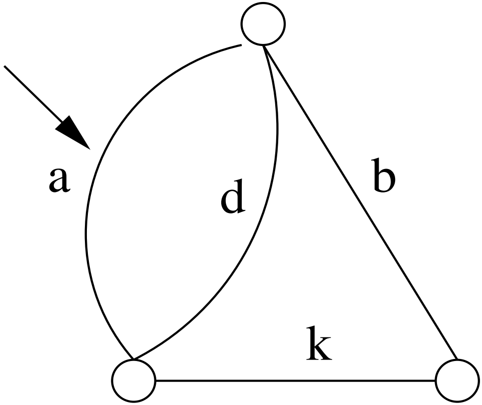
Step 3

Karger's Algorithm



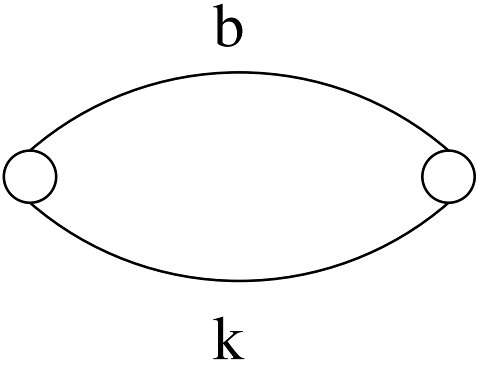
Step 4

Karger's Algorithm



Step 5

Karger's Algorithm



Step 6

Karger's Algorithm

- Consider any cut $C = (S, V - S)$.
- If we never contract any edges crossing C, then Karger's algorithm will produce the cut C.
  - Initially, all nodes are in their own cluster.
  - Contracting an edge that does not cross the cut can only connect nodes that both belong to the same side of the cut.
  - Stops when two supernodes remain, which must be the sets S and $V - S$.

The Story So Far

- We now have the following: **Karger's algorithm produces cut C iff it never contracts an edge crossing C.**
- How does this relate to min cuts?
- Across all cuts, min cuts have **the lowest probability** of having an edge contracted.
- Fewer edges than all non-min cuts.
- Intuitively, we should be more likely to get a min cut than a non-min cut.
- What is the probability that we do get a min cut?

Correctness with low probability

Theorem

The algorithm is correct with probability at least

$$\mathbf{2/n^2}$$

and never an underestimate.

Preliminaries

- Let $C = (V_1, V_2)$ be a specific minimum cut with $|C| = k$.
- What can you infer?

Preliminaries

- Let $C = (V_1, V_2)$ be a specific minimum cut with $|C| = k$.
- What can you infer?
  - Every vertex in G must have degree at least k.
  - Since there are n vertices in G, there are at least $nk/2$ edges in G.
- $\Pr[\text{first edge chosen randomly} \in \text{C}] \leqslant \frac{k}{nk/2} = \frac{2}{n}$.

Preliminaries

- Suppose at some step of the algorithm, there are Graph G_ℓ with ℓ vertices left.
- What can you infer?

Preliminaries

- Suppose at some step of the algorithm, there are Graph G_ℓ with ℓ vertices left.
- What can you infer?
  - the size of the minimum cut in G_ℓ is at least k.
  - there are at least $\ell k/2$ edges in G_ℓ.
- Pr[C is hit when there are ℓ vertices left $\mid$ C is not hit before] $\leqslant \frac{k}{\ell k/2} = \frac{2}{\ell}$

Correctness with low probability

Proof.

- Minimum cut of the graph doesn't decrease.
- Let $C = (V_1, V_2)$ be a specific minimum cut with $|C| = k$.
- Let A_i be event that we don't contract edge across C at step i.

$$\mathbb{P}\left[\cap_{1\le i\le n-2} A_i\right] = \mathbb{P}\left[A_1\right]\mathbb{P}\left[A_2|A_1\right]\ldots\mathbb{P}\left[A_{n-2}|\cap_{1\le i\le n-3} A_i\right]$$

- Number of edges before i-th step if no edges across C have been contracted so far is at least $(n-i+1)k/2$
- $\mathbb{P}\left[A_i|A_1\cap A_2\cap\ldots\cap A_{i-1}\right] \ge 1 - 2/(n-i+1)$ and so

$$\begin{aligned}
\mathbb{P}\left[\cap_{1\le i\le n-2} A_i\right] &\ge (1-\frac{2}{n})(1-\frac{2}{n-1})(1-\frac{2}{n-2})\ldots(1-\frac{2}{3}) \\
&= \frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdot\frac{n-4}{n-2}\cdot\ldots\cdot\frac{1}{3} = \frac{2}{n(n-1)}
\end{aligned}$$

$\square$

Content

We will see

- Boosting The Probability
- Speeding Things Up

Boosting the probability

Theorem

Repeating $\alpha n^2/2$ times (with new random coin flips) and returning smallest cut is correct with probability at least $1 - e^{-\alpha}$.

Proof.

$\square$

Boosting the probability

Theorem

Repeating $\alpha n^2/2$ *times (with new random coin flips) and returning smallest cut is correct with probability at least* $1 - e^{-\alpha}$.

Proof.

- Because each repeat is independent,

$$\mathbb{P}\left[\text{always fails}\right] = \prod_{1 \le i \le \alpha n^2/2} \mathbb{P}\left[i\text{-th try fails}\right] \le (1 - 2/n^2)^{\alpha n^2/2}$$

- Use fact $1 - x \le e^{-x}$ for $x \ge 0$ and simplify.

$\square$

Speeding Things Up: The Karger-Stein Algorithm

Some Quick History

- David Karger developed the contraction algorithm in 1993. Its runtime was $O(n^4 logn)$.
- In 1996, David Karger and Clifford Stein (the S in CLRS) published an improved version of the algorithm that is *dramatically* faster.
- **The Good News**: The algorithm makes intuitive sense.
- **The Bad News**: Some of the math is really, really hard.

Some Observations

- Karger's algorithm only fails if it contracts an edge in the min cut.
- The probability of contracting the wrong edge increases as the number of supernodes decreases.
- Since failures are more likely later in the algorithm, repeat just the later stages of the algorithm when the algorithm fails.

Intelligent Restarts

- Interesting fact: If we contract from n nodes down to $n/\sqrt{2}$ nodes, the probability that we don't contract an edge in the min cut is about 50%.
  - Can work out the math yourself if you'd like.
- What happens if we do the following?
  - Contract down to $n/\sqrt{2}$ nodes.
  - Run two passes of the contraction algorithm from this point.
  - Take the better of the two cuts.

The Success Probability

- This algorithm finds a min cut iff
  - The partial contraction step doesn't contract an edge in the min cut, and
  - At least one of the two remaining contractions does find a min cut.
- The first step succeeds with probability around 50%.
- Each remaining call succeeds with probability at least $4/n(n-1)$. Thinking assignment to calculate why.

A Success Story

- This new algorithm has roughly twice the success probability as the original algorithm!
- Key Insight: Keep repeating this process!
  - Base case: When size is some small constant, just brute-force the answer.
  - Otherwise, contract down to $n/\sqrt{2}$ nodes, then recursively apply this algorithm twice to the remaining graph and take the better of the two results.
- This is the **Karger-Stein** algorithm.

Two Questions

- What is the success probability of this new algorithm?
  - This is extremely difficult to determine.
  - We'll talk about it later.
- What is the runtime of this new algorithm.
  - Let's use the Master Theorem.

The Runtime

- We have the following recurrence relation:

$$\boxed{\begin{array}{ll} T(n) = c & \text{if } n \leq n_0 \\ T(n) = 2T(n/\sqrt{2}) + O(n^2) & \text{otherwise} \end{array}}$$

- What does the Master Theorem say about it?

$$T(n) = O(n^2 \log n)$$

The Accuracy

- By solving a very tricky recurrence relation, we can show that this algorithm returns a min cut with probability $\Omega(1/\log n)$.
- If we run the algorithm roughly $ln^2 n$ times, the probability that *all* runs fail is roughly

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

- **Theorem**: The Karger-Stein algorithm is an $O(n^2 \log^3 n)$-time algorithm for finding a min cut with high probability.

Major Ideas

- You can increase the success rate of a Monte Carlo algorithm by iterating it multiple times and taking the best option found.
- If you're more intelligent about how you iterate the algorithm, you can often do much better than this.