Lecture 2: Randomized Min-Cut

August 29, 2024

1 Randomized Min-Cut

Given an undirected graph $G = (V, E)$, we want to find a global min-cut. That is, a partition of V into S and $V - S$, both non-empty, to minimize the number of edges crossing the partition. This is also called the **edge-connectivity** of a graph and is usually denoted by $\lambda(G)$.

1.1 Alternate View

In this view, we look for a minimum number of edges whose removal creates non-trivial (more than one) components. **Observation:** In a connected undirected graph, a set of edges $E' \subseteq E$ is a min-cut if and only if $G - E'$ has exactly two connected components.

1.2 Examples

- **Cycle Graph (C_n):** For a cycle graph, $\lambda(G) = 2$. There are exactly $\binom{n}{2}$ distinct min-cuts.
- **Complete Graph (K_n):** For a complete graph, the min-cuts are formed by partitioning off a single vertex (singletons). The size of such a cut is $n - 1$.

2 Algorithms for Min-Cut

2.1 Traditional Algorithm

A traditional approach is to use $(n - 1)$ s-t cut computations. We can fix a vertex s and compute the s-t min-cut for all other vertices $t \in V \setminus \{s\}$, and then take the minimum of these cuts. **Recent Development:** Using isolating cuts, it's possible to compute the global min-cut using just $O(\log n)$ s-t cut computations. Undirected graph cuts have more structure than directed graphs, which can be exploited. In the early 90s, Nagamochi and Ibaraki developed an MA-ordering based algorithm that runs in $O(mn)$ time.

2.2 Karger's Randomized Contraction Algorithm

In the mid-90s, David Karger introduced a randomized algorithm based on edge contraction. This algorithm, its analysis, and its implications have been very influential. We will work with multigraphs since they arise naturally from the contraction process, even when starting with a simple graph. Let n be the number of vertices and m be the number of edges.

2.2.1 Motivation

Lemma: The minimum degree δ is at most the average degree.

$$\lambda(G) \leq \delta \leq \frac{1}{n} \sum_{v \in V} \deg(v) = \frac{2m}{n}$$

Proof: The cut formed by isolating the vertex with the minimum degree has size δ. Thus, $\lambda(G) \leq \delta$. The average degree is $\frac{2m}{n}$. Now, fix a specific min-cut C, so $|C| = \lambda$. Suppose we pick a random edge e from the graph.

$$\Pr[e \in C] = \frac{|C|}{|E|} = \frac{\lambda}{m} \leq \frac{2m/n}{m} = \frac{2}{n}$$

A random edge has a very low probability of being in a specific min-cut. This suggests that if we pick an edge $e \notin C$, it is "safe" to contract it without destroying the min-cut C.

2.2.2 Edge Contraction

Given a graph $G = (V, E)$ and an edge $e = (u, v)$, the graph G/e is obtained by merging the two endpoints u and v into a single new vertex. Any self-loops created are removed. This process may create parallel edges. **Observation:** For any edge e, $\lambda(G/e) \geq \lambda(G)$, assuming G is connected.

2.2.3 The Algorithm

```
RandomContraction(G = (V, E)): If n = 2, output the unique cut. Else: Pick an edge e uniformly at rando
```

Theorem: Let C be a fixed min-cut of G. The probability that the algorithm outputs C is at least $\frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}}$. *Proof:* The algorithm outputs C if and only if it never contracts an edge belonging to C. Let $n_i = n - i + 1$ be the number of vertices at step i. The number of edges is $m_i \geq n_i\lambda/2$. The probability of not contracting an edge from C at step i is:

$$1 - \frac{\lambda}{m_i} \geq 1 - \frac{\lambda}{n_i\lambda/2} = 1 - \frac{2}{n_i}$$

The probability of success is the product of these probabilities over all $n - 2$ contractions:

$$\Pr[\text{success}] \geq \prod_{i=0}^{n-3} \left(1 - \frac{2}{n-i}\right) = \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{1}{3}\right) = \frac{2}{n(n-1)}$$

This type of randomized algorithm is called a **Monte Carlo algorithm** because it can fail with some probability but is otherwise efficient.

2.2.4 Improving Success Probability

The RC algorithm can be implemented to run in $O(n^2)$ time, or $O(m)$ with care. To improve the success probability, we can run the algorithm multiple times. If we run it $N = \alpha\binom{n}{2}$ times and take the minimum cut found:

$$\Pr[\text{failure in all N runs}] \leq \left(1 - \frac{1}{\binom{n}{2}}\right)^{\alpha\binom{n}{2}} \approx e^{-\alpha}$$

By choosing $\alpha = \Theta(\log n)$, we can make the failure probability polynomially small ($1/\text{poly}(n)$). However, this increases the total running time to $O(n^4 \log n)$, which is not competitive.

2.3 A Key Structural Result

A major impact of this algorithm was the following structural theorem: **Theorem:** In any graph with n nodes, there are at most $\binom{n}{2}$ distinct min-cuts. *Proof:* Each min-cut is found by the RC algorithm with probability at least $1/\binom{n}{2}$. Since the probabilities for disjoint events (finding different min-cuts) must sum to at most 1, there cannot be more than $\binom{n}{2}$ such cuts. This bound is tight for a cycle graph C_n.

3 Faster Min-Cut: The Karger-Stein Algorithm

The analysis of the RC algorithm shows that the probability of error is small at the beginning but grows as n decreases. **Observation:** The probability that a fixed min-cut C survives the contraction process down to k vertices is $\geq \frac{k(k-1)}{n(n-1)}$. If we contract down to $n/\sqrt{2}$ vertices, the success probability is approximately $1/2$. This suggests that we can be more careful in the later stages.

3.1 The Karger-Stein (KS) Algorithm

The algorithm is recursive.

`KargerStein(G):`

If $n \leq 6$, find min-cut by brute force in $O(1)$ time.

Else:
1. Set $t = \lceil 1 + n/\sqrt{2} \rceil$.
2. Run the basic Random Contraction algorithm on G until t vertices remain. Let the resulting graph be G_1.
3. Independently, run Random Contraction again on the original G until t vertices remain. Let the result be G_2.
4. Recursively call $C_1 = \texttt{KargerStein}(G_1)$ and $C_2 = \texttt{KargerStein}(G_2)$.
5. Return $\min(C_1, C_2)$.

3.2 Analysis

Running Time: The contraction phases take $O(n^2)$ time. The recurrence relation for the running time is $T(n) = O(n^2) + 2T(n/\sqrt{2})$. This solves to $T(n) = O(n^2 \log n)$. **Success Probability:** Let $P(n)$ be the probability of success for a graph of size n. The probability of a fixed min-cut surviving one contraction phase to t vertices is $\approx 1/2$. A recursive call succeeds if one of the two subproblems succeeds. A subproblem succeeds if its contraction phase preserves the cut AND the subsequent recursive call finds it.

$$P(n) \geq 1 - \left(1 - \frac{1}{2} P\left(\lceil 1 + n/\sqrt{2} \rceil\right)\right)^2$$

This recurrence solves to $P(n) = \Omega\left(\frac{1}{\log n}\right)$. We can boost this to high probability by running the entire KS algorithm $O(\log^2 n)$ times, for a total runtime of $O(n^2 \log^3 n)$.

3.3 Connection to Galton-Watson Processes

A Galton-Watson process is a branching stochastic process that models population growth. Imagine a species where each individual, in each generation, produces a random number of offspring according to a fixed probability distribution. Let μ be the expected number of offspring for any individual.

- If $\mu < 1$, the population goes extinct with probability 1.
- If $\mu > 1$, the population may survive with positive probability.
- If $\mu = 1$ (the critical case), the population almost surely goes extinct, but the probability of surviving for d generations is $\Theta(1/d)$.

The Karger-Stein algorithm can be mapped to this process. The recursion forms a binary tree. Each node represents a call to the algorithm. A call "survives" if the initial contraction phase preserves the min-cut (with probability $\approx 1/2$). So each node has two children, and the expected number of surviving children is $2 \times (1/2) = 1$. This is the critical case $\mu = 1$. The algorithm succeeds if any path of recursive calls survives to the base case. The depth of the recursion is $d = O(\log n)$. The survival probability is therefore $\Omega(1/d) = \Omega(1/\log n)$.

3.4 Direct Proof of Success Probability

Let p_i be the probability of success at recursion depth i, where the number of vertices is $n_i = n/(\sqrt{2})^i$. $p_0 = 1$ (for a sufficiently small graph solved by brute force). The probability of success at depth $i+1$, p_{i+1}, depends on the success at depth i, p_i. A call at depth i succeeds if at least one of its two children succeeds. A child succeeds if its contraction phase is successful (prob $\geq 1/2$) and its recursive call is successful (prob p_i). The probability of one branch failing is $(1 - \frac{1}{2}p_i)$.

$$p_{i+1} = 1 - \left(1 - \frac{1}{2}p_i\right)^2 = p_i - \frac{p_i^2}{4}$$

We claim by induction that $p_i \geq \frac{c}{i+c'}$ for some constants c, c'. For simplicity, let's prove $p_i \geq \frac{2}{i+2}$.

- **Base Case ($i = 1$):** $p_1 = p_0 - p_0^2/4 = 1 - 1/4 = 3/4$. We need $3/4 \geq 2/3$, which is true.
- **Inductive Step:** Assume $p_i \geq \frac{2}{i+2}$.

$$p_{i+1} = p_i\left(1 - \frac{p_i}{4}\right) \geq \frac{2}{i+2}\left(1 - \frac{1}{4}\cdot\frac{2}{i+2}\right) = \frac{2}{i+2}\left(1 - \frac{1}{2(i+2)}\right) = \frac{2}{i+2}\frac{2i+3}{2(i+2)} = \frac{2i+3}{(i+2)^2}$$

  We need to show $\frac{2i+3}{(i+2)^2} \geq \frac{2}{i+3}$. This is equivalent to $(2i+3)(i+3) \geq 2(i+2)^2$, which simplifies to $2i^2 + 9i + 9 \geq 2i^2 + 8i + 8$, or $i + 1 \geq 0$, which is true for $i \geq 0$.

The depth of recursion is $d \approx \log_{\sqrt{2}}(n) = 2\log_2(n)$. The success probability is $\Omega(1/d) = \Omega(1/\log n)$.