Randomized Algorithms, Lecture No. 4

Russell Impagliazzo*

January 20, 1998

1 Karger's Min-Cut Algorithm

Randomization can also be applied to problems that have known polynomial time solutions in hopes of obtaining faster and/or simpler algorithms. Karger's Min-Cut algorithm is an example of a randomized algorithm that achieves near-linear speedup and greatly simplifies the standard deterministic approach.

The Problem: Given a graph, find a (non-trivial) division that minimizes the number of cross-edges.

This problem can be solved deterministically:

1. fix one vertex, s, to be in the "left" division;
2. for each $t \neq s$ apply Ford-Fulkerson to find the max-flow from s to t, and compute the residual graph to find the min-cut s to t;
3. select the t that minimizes this cut;

Each of the n runs of Ford-Fulkerson works in time $O(cutsizem)$. As the minimum cut cannot be larger than the minimum degree in the graph (otherwise select that vertex to comprise one partition), the algorithm is $O(m^2) = O(n^4)$

Karger's Basic Algorithm: Define a *contraction* of two nodes, u and v, in a (multi-)graph as a new node u' with edges that are the (multi-set) union of the edges incident on either u or v, except for any edge(s) connecting u and v.

The algorithm is simply:

1. pick a random edge in the graph, $(u, v) \epsilon E$;
2. replace u and v with the contraction u';
3. repeat until only 2 nodes remain, which will be compound-contractions of the two divisions.

*As interpreted by Ted Carson

Lemma 1: A division (cut) returned by this algorithm is valid.

Proof: Edges are deleted from the graph only when they connect two contracting verticies, and these verticies necessarily end up in the same partition. Therefore no edges between the two partitions are removed and exist in the final stage. $\diamondsuit$

However, these valid partitions are not guaranteed to be minimal, so how likely are we to find the minimum cut?

Lemma 2: Let k be the actual min-cut size, and let $C = \{e_1, e_2, \ldots, e_k\}$ be a k-cut. Unless some $e \,\epsilon\, C$ is selected for contraction, C is output.

Proof: Let S and T be the partitions of G, and C be all edges $\{(s,t) : s \,\epsilon\, S, t \,\epsilon\, T\}$. If we don't contract an element of C, then the compound nodes stay entirely within S or T. At the end of the algorithm, we have two compound nodes covering the entire graph, each contained entirely in S or T. One must be S and the other T. $\diamondsuit$

Given that some k-cut exists, how likely are we to find it? We bound this probability by showing that many other edges exist and are likely to be choosen at each stage of the algorithm.

Lemma 3: Every node in the contracted graph has degree $\geq k$.

Proof: If a node has degree $\leq k$, then selecting it as one partition gives a cut $\leq k$. But, k is the min-cut size, contradicting the assumption. $\diamondsuit$

At each stage we have $n - t$ nodes, and at least $(n-t)k/2$ edges to choose from. The probability of selecting an element of C to contract is

$$Pr[x_t \,\epsilon\, C] \leq \frac{k}{(n-t)\,k/2} \leq \frac{2}{n-t}$$

or, for the whole algorithm, the probability of returning C,

$$\begin{aligned}
Pr[C] &\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(\frac{1}{3}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{3}{5}\right)\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{2}{n\,(n-1)}
\end{aligned}$$

Each run of the algorithm we have at least $O(1/n^2)$ chance of finding cut C. Running the algorithm $O(n^2)$ times, we expect to find C, giving a total running time of $O(n^4)$ when we would first expect to find the minimum-cut, (as bad as, or even worse for sparse graphs, the deterministic approach).

This approach is in some sense unbalanced, as the entire algorithm is rerun at each trial, while the chance of failure increases toward the end of the run. It might benefit us to rerun the different stages of the algorithm based on their likelihood of failing (as determined by the above analysis).

If we run for only t steps we have the probability that C remains uncut

$$\begin{aligned} Pr[C_t] &\geq \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{n-t-1}{n-t+1}\right)\left(\frac{n-t-2}{n-t}\right) \\ &= \frac{(n-t-1)(n-t-2)}{n(n-1)} \\ &\epsilon \; O\left(\frac{(n-t)^2}{n^2}\right) \end{aligned}$$

Take the following modified algorithm: run the basic algorithm to $t = n/2$ four times; recurse on each with the modified algorithm. The running time for this will be:[1]

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + O\left(n^2\right) \\ &\epsilon \; O\left(n^2 \lg n\right) \end{aligned}$$

Finally, the probability of success is the probability that not all the recursions fail:

$$\begin{aligned} P_n &= 1 - \left(1 - \frac{1}{4}P_{n/2}\right) \\ &= 1 - \left(1 - 4\left(\frac{1}{4}P_{n/2} + 6\left(\frac{1}{4}P_{n/2}\right)^2 - 4\left(\frac{1}{4}P_{n/2}\right)^3 + \left(\frac{1}{4}P_{n/2}\right)^4\right)\right) \\ &\geq P_{n/2} - \frac{3}{8}P_{n/2}^2 \\ &= P_{n/2}\left(1 - \frac{3}{8}P_{n/2}\right) \end{aligned}$$

Intuitively, each time n doubles, we have one more recursion where we might fail (but expect to succeed with constant probability). So we are harmonically decreasing the probability of success. Therefore, try $P_n = 1/\lg n$. Since we have,

[1] note that this is a "balanced" steady state recursion

$$P_n \geq P_{n/2}\left(1 - \frac{3}{8}P_{n/2}\right)$$

we need,

$$\begin{aligned}
\frac{1}{\lg(n-1)}\left(1 - \frac{3}{8}\frac{1}{\lg(n-1)}\right) &\overset{?}{>} \frac{1}{\lg n} \\
1 - \frac{3}{8}\frac{1}{\lg(n-1)} &\overset{?}{>} 1 - \frac{1}{\lg(n-1)} \\
\frac{3}{8}\frac{1}{\lg(n-1)} &\overset{?}{<} \frac{1}{\lg(n-1)} \\
\frac{3}{8} &\overset{\surd}{<} 1
\end{aligned}$$

The modified algorithm runs in time $T(n)\,\epsilon\,O(n^2 \lg n)$, with probability of success, $P(n)\,\epsilon\,O(1/\lg n)$. Repeating the modified algorithm $O(\lg n)$ times we expect to succeed with constant probability. To achieve a small probability of failure, ϵ, we repeat $O(\lg n \lg \epsilon^{-1})$ times, for a total running time:

$$\overline{T}(n)\;\epsilon\;O\left(n^2 \lg^2 n \lg \epsilon^{-1}\right)$$