

Min-Cut and Randomization

Algorithm Design Kleinberg and Tardos, section 13.2
Notes by Scott Alfeld
17.11.08

1: Min-Cut

We define the problem of *Min-Cut* as follows. Given an undirected graph $G = \{V, E\}$ with $n = |V|$ and $m = |E|$, let a cut of G be defined as a partition (A, B) of G . Because this is a partition, we know:

- $A, B \subset V$
- $A, B \neq \emptyset$
- $A \cap B = \emptyset$
- $A \cup B = V$

We define the cardinality of a cut to be the number of edges between A and B , or more formally:

$$\text{Cardinality}(A, B) = \text{size}(\{(u, v) \in E \mid u \in A, v \in B\})$$

The problem of *Min-Cut* is to find a cut with minimal cardinality.

2: A Max-Flow Approach

Recall that the problem of *Min s-t Cut* is the problem of *Min-Cut* with the added restrictions that $s \in A$ and $t \in B$. Recall the network flow solution to *Min s-t Cut*:

1. Replace every edge (u, v) in G with two directed edges (u, v) and (v, u) .
2. Set the capacity for each edge to 1.
3. Find the max flow from s to t .

In the case of *Min-Cut*, we have no limitations on s and t . For an arbitrary vertex s , however, we know that the minimum cut separates s from **some** t . We therefore have the following algorithm for *Min-Cut*:

1. Replace every edge (u, v) in G with two directed edges (u, v) and (v, u) .
2. Set the capacity for each edge to 1.
3. Pick an arbitrary vertex $s \in V$.
4. Find the minimum of all max flows from s to v , for every $v \in V$.

With a common max-flow algorithm, running in $O(mn)$ time, the above procedure runs in $O(mn^2)$ time. Using a faster max-flow algorithm, we are able to reduce the time to $O(n^3)$.

3: The Contraction Algorithm, A Randomized Approach

We now introduce a new, randomized algorithm for solving *Min-Cut* on a multi-graph $G = \{V, E\}$. First we define what it is to *contract* an edge $e = (u, v)$.

- Replace vertices u and v with a new vertex w .

- Update edges such that edges connecting to u or v now connect to w .
- Keep any parallel edges.
- Delete any self-looping edges.

We do not present an algorithm for implementing a contraction here. However, a contraction of an edge can be done in $O(n)$ time. We now define the full Contraction Algorithm:

1. Choose an edge $e = (u, v)$ drawn uniformly at random from E .
2. Contract e .
3. If the graph has two vertices, return the number of edges between them. Otherwise goto step 1.

Every time the contraction algorithm contracts an edge, the number of vertices is reduced by one. Because every edge contraction takes $O(n)$ time, the running time of a single run of the contraction algorithm is $O(n^2)$. Our claim is that the contraction algorithm returns a correct answer with probability at least $\frac{1}{n^2}$.

Before analyzing the correctness of the Contraction Algorithm, we first discuss some intuition as to why it might work. In G , the min-cut must be relatively small compared to the size of E . Therefore, when contracting an edge e , the probability that e is an edge between A and B is small.

3.1: Analysis of the Contraction Algorithm

We define an oracle-given cut (A^*, B^*) with minimal cardinality. We then define F^* to be the set of edges between A^* and B^* , and let $k = |F^*|$. Because (A^*, B^*) is a min-cut, we know that every vertex v has degree at least k (otherwise the cut $(\{v\}, V \setminus \{v\})$ would have cardinality less than k). Therefore, the total number of edges in G is $|E| \geq \frac{1}{2}kn$. Thus

$$P(\text{the algorithm contracts an edge in } F^*) \leq \frac{k}{|E|} \leq \frac{2}{n}$$

The contraction algorithm, however, fails if it contracts an edge in F^* at any iteration. We let $G' = \{V', E'\}$ be the graph after j iterations. Clearly there are $n' = n - j$ vertices in G' . Suppose that no edge in F^* has been contracted, then the min-cut of G' is still k , and therefore, as before,

$$|E'| \geq \frac{1}{2}kn'$$

We define X_j to be the event that the edge contracted on the j -th iteration is not in F^* . Therefore, assuming that no edges in F^* were contracted before the j -th iteration,

$$P(X_j) \leq \frac{k}{|E|} \leq \frac{2}{n'}$$

We now show that the probability that no edge in F^* is contracted is greater than or equal to $\frac{2}{n^2}$. This is to say

$$P(X_1 \wedge X_2 \wedge \dots \wedge X_{n-3} \wedge X_{n-2}) \geq \frac{2}{n^2}$$

We note that for any events α and β , $P(\alpha \wedge \beta) = P(\alpha) \times P(\beta | \alpha)$. Therefore,

$$\begin{aligned}
P(X_1 \wedge X_2 \wedge \dots \wedge X_{n-3} \wedge X_{n-2}) &= P(X_1) \times P(X_2 \mid X_1) \times \dots \times P(X_{n-2} \mid X_1 \wedge X_2 \wedge \dots \wedge X_{n-3}) \\
&\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \dots \left(1 - \frac{2}{4}\right) \left(1 - \frac{2}{3}\right) \\
&= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \dots \left(\frac{2}{4}\right) \left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)} \\
&\geq \frac{2}{n^2}
\end{aligned}$$

3.2: Amplification of the Contraction Algorithm

Because we want a probability of success greater than $\frac{2}{n^2}$, we now show how the contraction algorithm can be amplified. We do this probability amplification in the following manner

1. Run the contraction algorithm independently ρ times on G .
2. Return the lowest result.

Clearly this algorithm fails only when all of the ρ runs of the contraction algorithm fail. Therefore the probability of this algorithm failing is less than or equal to

$$\left(1 - \frac{2}{n^2}\right)^\rho$$

By letting $\rho = n^2 \log(n)$, we obtain a probability of failure less than or equal to

$$\left(1 - \frac{2}{n^2}\right)^\rho = \left(\left(1 - \frac{2}{n^2}\right)^{\frac{1}{2}n^2}\right)^{2\log(n)}$$

By the fact that for all $x \geq 1$, $(1 - \frac{1}{x})^x \leq \frac{1}{e}$,

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

4: Doing Even Better

The running time for the un-amplified contraction algorithm is $O(n^2)$. Therefore the running time of the amplified contraction algorithm is $O(n^4 \log(n))$. While this is comparable to flow-based algorithms, we would like to do better.

We note that, when contracting edges early in process, the likelihood of contracting a problematic edge is very low. As more edges are contracted, however, the probability of contracting an edge in F^* increases. We therefore define the following, recursive, algorithm

1. Run contraction algorithm until $\frac{n}{\sqrt{2}}$ vertices remain.
2. Recursively run algorithm twice, independently, on resulting graph, returning the lower result.

This recursive algorithm returns the correct answer with probability greater than $\frac{1}{\log(n)}$. The running time, however, is defined by the recurrence relation

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

This is slower than the original contraction algorithm, however the probability of success is exponentially higher. We can amplify our new algorithm by running it $N = c \log^2(n)$ times independently, where c is a constant and obtain a probability of success of

$$1 - \left(\frac{1}{\log(n)}\right)^{c \log^2(n)} \geq 1 - e^{-c \ln(n)} = 1 - \frac{1}{n^c}$$

We now have an algorithm that computes the minimum cut size with high probability with running time

$$O(n^2 \log^3(n))$$

Credits:

In addition to Section 13.2 of *Algorithm Design* by Kleinberg/Tardos, and the corresponding lecture slides, this lecture drew heavily from Jeff Erickson's Min-Cut notes for CS473G - Algorithms, Fall 2005.