18.433 Combinatorial Optimization

Minimum Cuts

October 2 *Lecturer: Santosh Vempala*

Finding minimum cuts in graphs is an interesting problem and has many applications in some areas such as network design. This problem has two variants: (1) finding a minimum set of edges whose removal disconnects a particular vertex s from a particular vertex t (we call this version the *minimum s-t cut problem*), (2) finding a minimum set of edges whose removal disconnects the graph (we call this version the *minimum cut problem*). Both versions apply to directed or undirected graphs.

The minimum s-t cut problem for directed graphs can be solved by using the Max flow-Min cut theorem which says the maximum flow in the graph is equal to the minimum cut in the graph. In fact, a minimum cut $(S, \overline{S})$ can be obtained by choosing those vertices of G to which there exist directed paths from s in the residual graph as S and the rest of the vertices as $\overline{S}$. We can also solve the minimum s-t cut problem in undirected graphs using the algorithm for directed ones (the solution is left as an exercise).

One naive way to solve the minimum cut problem is to choose every pair of vertices as s and t and then use the algorithm for the minimum s-t cut problem. The running time of this algorithm is $\binom{n}{2}$ times of that of the maximum flow algorithm. We can reduce the factor $\binom{n}{2}$ to $n-1$ by considering the fact that each minimum cut separates a fixed vertex s from at least one other vertex t (we have $n-1$ ways to choose t).

Below, we give a randomized approach for finding a minimum cut in undirected graphs. We also analyze the running time of the algorithm carefully.

Randomized minimum cut algorithm

1. **While** there exist more than two vertices in the graph
   (a) Pick an edge e at random.
   (b) Contract e to a single vertex to get a multigraph (we keep multiple edges).
2. Report the edges between the two remaining super-vertices as a minimum cut.

One can observe that the running time of contracting an edge is in $O(n)$ and the number of iterations of the main loop is $n-2$. Thus the overall running time is in $O(n^2)$. Later in

this lecture, we use some tricks to reduce the running time from $O(n^2)$ time to $O(m)$ time (m is the number of edges).

We now compute the chance of reporting a minimum cut. One can see that the number of cuts in a graph is exponential (2^n) and thus the chance of getting a minimum cut from a random process might be very low, i.e. $\frac{1}{2^n}$. However, we show in Lemma 1 that this chance in not small in our case.

Lemma 1. *Any particular minimum cut will be the output of the algorithm with probability at least* $\frac{1}{\binom{n}{2}} \approx \frac{2}{n^2}$.

Proof. Suppose a minimum cut $C = (S, \overline{S})$ has c edges. Thus the degree of each vertex in the graph is at least c and the number of edges is at least $\frac{nc}{2}$. If the algorithm does not pick any edge from C, then our final cut will be C.

$$Prob(picking\ an\ edge\ from\ C) \leq \frac{c}{\frac{nc}{2}} = \frac{2}{n}$$

and thus

$$Prob(not\ picking\ any\ edge\ from\ C) \geq 1 - \frac{2}{n}$$

Using the same reasoning, we can observe that the chance of not picking any edge from C after contracting the first edge is $1 - \frac{2}{n-1}$ and so on. Therefore

$$Prob(finding\ cut\ C) \geq (1 - \frac{2}{n}) \cdot (1 - \frac{2}{n-1}) \cdots (1 - \frac{2}{3}) = \frac{1}{\binom{n}{2}}.$$

□

Thus the chance of succeed in our randomized algorithm is at least $\frac{1}{\binom{n}{2}}$. We can improve our chance by iterating this algorithm more than one time and choosing the best cut as our final result. We have

$$\begin{aligned} Prob(succeed\ in\ k\ attempts) &= 1 - Prob(fail\ in\ all\ attempts) \\ &= 1 - Prob(F_1)Prob(F_2) \cdots Prob(F_k) \\ &= 1 - Prob(F_1)^k \\ &= 1 - (1 - \frac{1}{\binom{n}{2}})^k \end{aligned}$$

For example, after $k = \binom{n}{2}$ iterations of the algorithm, we have at least $1 - \frac{1}{e}$ chance and by $k = 2\binom{n}{2} \ln n$ iterations, we have at least $1 - \frac{1}{n^2}$ chance of success.

It is worth mentioning that using Lemma 1, we can obtain the upper bound $\binom{n}{2}$ for the maximum number of minimum cuts in a graph. In fact, each minimum cut can be obtained uniquely by the above algorithm with probability at least $\frac{1}{\binom{n}{2}}$ and thus the number of minimum cuts can not be more that the aforementioned upper bound.

As stated earlier, the running time of the algorithm is $O(n^2)$. We now improve the running time to $O(m)$. Suppose we pick a random permutation $e_1, e_2, \cdots, e_m$ of all edges and contract each edge in this order that we can until we have two vertices left in the graph. We can observe that this algorithm has the same output as our first algorithm. Now, consider this binary search approach: first contract edges $e_1, e_2, \cdots, e_{\lceil \frac{m}{2} \rceil}$ in $O(m)$. If the number of vertices after contracting is two, then we report the edges between these vertices as a minimum cut, if the number is one we recurse on the first half of edges, otherwise we continue the algorithm for the second half of edges. The running time of this algorithm is in $O(m \log m)$. However still there is some room for improvement. If the number of remaining vertices is more than two, the new obtained graph G' has at most $\lfloor \frac{m}{2} \rfloor$ edges (we contracted the first half of edges before) and if the number of remaining vertices is 1 then we can discard the second half of edges. In both cases, we have at most $\frac{m}{2}$ edges left. Thus, the second iteration takes time proportional to $m/2$. Similarly, the third iteration takes time propotional to $m/4$ and so on. Thus the overall running time is in $O(m)$.