Project Description

Let G be a graph with n nodes and m edges. In class, we talked about Karger's randomized algorithm that finds a minimum cut. Here is the original paper [2] and here is the wikipedia entry [1]. Recall that the basic algorithm runs in $O(m)$ time and outputs the minimum cut with very small probability. In order to boost this probability up to $1-1/n^2$, we must execute the algorithm $O(n^2 \log n)$ times, which makes the total time approximately equal to $O(mn^2 \log n)$. Since m can be at most $n(n-1)/2$ (in case of a complete graph), the total time required is $O(n^4 \log n)$, which is very large.
In this project you will design, analyze and implement a much better randomized algorithm for the minimum cut, one that runs in $O(n^2 \log^4 n)$ time and outputs a cut with probability $1-1/n^2$. This will be a huge improvement! Also, to start off, you will get to execute the inefficient $O(n^4 \log n)$ time algorithm that you learned in class. To assist you with the implementation of both algorithms, a skeleton C file will be provided. To assist you with the design and analysis of the new algorithm, several hints will be provided.

Warm-up: Karger's Algorithm as we learned it in class

Recall how Karger's algorithm works. It repeatedly picks an edge at random, contracts it, until there are two nodes v and u left in the graph. Here the pseudocode of the algorithm:

- **KargerCut**$(G=(V,E))$
  - while $|V|>2$
    - choose $e \in E$ uniformly at random
    - $G = G/e$
  - return G

Assume we fix a minimum cut C (there can be more than one minimum cuts). Recall that in its first choice, the probability that the algorithm outputs an edge not belonging to the minimum cut C is at least $1-2/n$ (we proved this in class). Let $p(n)$ be the probability that after the execution of the algorithm on a graph of n nodes, the specific minimum cut C is returned. The probability that the algorithm succeeds can be described by the recursive relation

$$p(n) \geq (1-2/n)p(n-1)\,, \tag{1}$$

where $p(2)=1$. This is because for the algorithm to succeed on a graph of n nodes G_n, it must succeed the first time (and not pick an edge belonging to the minimum cut C) and then succeed on the remaining graph G_{n-1} of $n-1$ nodes.

Task 1: (10%) Solve the recurrence to compute the success probability $p(n)$ of **KargerCut**$(G=(V,E))$. You should derive the same value we derived in class. Note that this is another way to analyze the success probability of the algorithm (in class we did the analysis in a non-recursive manner).

Task 2: (10%) In the discussion section you implemented this algorithm. Run your implementation for a random graph of $n=10,20,30,40$ nodes and $3n$ edges and report (i) the minimum cut; (ii) the number of times you had to run it to compute the minimum cut.

Improving the time complexity of Karger's algorithm

Now you will design a much better algorithm for minimum cut. To begin, please complete the following:

Task 3: (10%) Show that the probability $p(n,t)$ that Karger's algorithm will pick an edge that does not belong to the minimum cut C during its first t edge contractions is

$$p(n,t) \geq \frac{(n-t)(n-t-1)}{n(n-1)}$$

and that

$$p(n,t) \geq \frac{1}{2} - \frac{1}{\sqrt{2}n} \text{ for } t \leq \frac{2-\sqrt{2}}{2}n\,.$$

The above observation tells us, that, in Karger's algorithm, the probability of not picking up a "bad" edge is kept relatively large (always more than $1/2$ for large n) during the first

$$\lfloor \frac{2-\sqrt{2}}{2} n \rfloor \approx \lfloor 0.29n \rfloor$$

edge contractions. After we cross that threshold, this probability deteriorates substantially.
We now use this observation: Instead of removing all edges in one shot, we first remove $t = \lfloor 0.29n \rfloor$ edges one by one, at which point the graph is left with

$$n - t = n - \lfloor \frac{2-\sqrt{2}}{2} n \rfloor = \lceil n - \frac{2-\sqrt{2}}{2} n \rceil = \lceil \frac{n}{\sqrt{2}} \rceil$$

nodes. To amplify the probability of success, we do **two** runs of t contractions, outputting graphs G_1 and G_2, then we call the algorithm recursively on both graphs and then we keep the best cut!
Please see algorithm **FASTmincut**$(G = (V, E))$ below (note that the algorithm below returns just the size of the minimum cut and not the actual graph of the cut):

- **FASTmincut**$(G = (V, E))$
  - if $|V| > 2$
    - $theshold = \lceil |V|/\sqrt{2} \rceil$
    - $G_1 =$ **contract**$(G, threshold)$
    - $G_2 =$ **contract**$(G, threshold)$
    - return $\min\{$**FASTmincut**$(G_1 = (V_1, E_1))$, **FASTmincut**$(G_2 = (V_2, E_2))\}$
  - else
    - return $|E|$;
- **contract**$(G = (V, E)$**,** $threshold$**)**
  - while $|V| \geq threshold$
    - choose $e \in E$ uniformly at random
    - $G = G/e$
  - return G

Task 4: (40%) Compute **a lower bound** on the probability $p(n)$ that **FASTmincut** returns a minimum cut when it is executed on a graph of n nodes. To do that, use the following three hints.
HINT 1: **FASTmincut** outputs a minimum cut during execution on a graph of n nodes when either of the following events are true:

1. **FASTmincut** does not choose some edge belonging in the minimum cut during the call $G_1 =$ **contract**$(G, threshold)$ and **FASTmincut** outputs a minimum cut when it is executed on graph G_1.
2. **FASTmincut** does not choose some edge belonging in the minimum cut during the call $G_2 =$ **contract**$(G, threshold)$ and **FASTmincut** outputs a minimum cut when it is executed on graph G_2.

HINT 2: For three events A, B, C, such that $A = B \cup C$ and B and C are independent, it is

$$\Pr(A) = 1 - (1 - \Pr(B))(1 - \Pr(C)) .$$

HINT 3: Eventually, you should come up with an equation of the form

$$p(n) \geq p(\lceil \frac{n}{\sqrt{2}} \rceil) - \frac{1}{4} p^2(\lceil \frac{n}{\sqrt{2}} \rceil) - \frac{c}{n} ,$$

for some constant c. Then, using the base case $p(2) = 1$, you should prove by induction that $p(n) \geq \frac{1}{\log n}$.

Task 5: (10%) Using the master theorem compute the time complexity $T(n)$ of the **FASTmincut** algorithm. Note that running

$$\textbf{contract}(G, \lceil \frac{n}{\sqrt{2}} \rceil)$$

takes $O(n^2)$ time. How many times we need to run **FASTmincut** in order to find a minimum cut with probability at least $1 - 1/n^2$? Argue that the final time complexity of the algorithm is $O(n^2 \log^4 n)$.

Task 6: (20%) Implement the **FASTmincut** algorithm. Run it for a random graph of $n = 10, 20, 30, 40$ nodes and report (i) the minimum cut; (ii) the number of times you had to run it to compute the minimum cut. Plot the running time of **FASTmincut** and the running time of **KargerCut** at the same plot (with repeated executions so that they output the cut with probability $1 - 1/n^2$).

Academic Integrity

This project is to be done individually. Any implementation or analysis ideas that are not purely your own should be cited, and in particular ideas taken from books or the Internet. Please do not collaborate with other students while developing your programs, and please abide by the student honor code.

References

[1] Karger's algorithm. https://en.wikipedia.org/wiki/Karger%27s_algorithm. Accessed: 2010-09-30.

[2] David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas.*, pages 21–30, 1993.