Warm-up

Problem 1. Explain why every undirected graph on n vertices has exactly $2^{n-1} - 1$ distinct cuts (not all minimum cuts).

Solution 1. Each cut is identified by a partition of V. To count the number of possible partitions, consider choosing the number of subsets of $\{1, \ldots, n\}$, which is 2^n. So, we can choose a subset $A \subset V$ then let the rest be $B = V \setminus A$ as one partition. Due to the symmetry of partition, $(A_1, B_1) = (A_2, B_2)$ if $A_1 = B_2$ and we need to ensure that it is nonempty: $A_1 = \varnothing$ or $A_1 = \{1, \ldots, n\}$ (B_1 will be empty). We have

$$\frac{2^n - 2}{2} = 2^{n-1} - 1.$$

Problem 2. Solve the two recurrence relations for the Karger–Stein algorithm (time and probability).

Solution 2. For the runtime,

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

We can see the pattern:

$$\begin{aligned}
T(n) &= 4T\left(\frac{n}{2}\right) + 2c \cdot \left(\frac{n}{\sqrt{2}}\right)^2 + cn^2 \\
&= 4T\left(\frac{n}{2}\right) + cn^2 + cn^2 \\
&= 8T\left(\frac{n}{2\sqrt{2}}\right) + 4c\left(\frac{n}{2}\right)^2 + cn^2 + cn^2 \\
&= 8T\left(\frac{n}{2\sqrt{2}}\right) + 3cn^2 \\
&= \sum_{j=0}^{O\left(\log_{\sqrt{2}} n\right)} 2^j c \cdot \left(\frac{n}{\left(\sqrt{2}\right)^j}\right)^2 = O(n^2 \log n).
\end{aligned}$$

For the probability recurrence, see the proof from the lecture notes.

Problem 3. Analyse/describe what would happen to the time complexity and success probability if we only did 1 run (instead of 2) for the Karger–Stein algorithm. What if we did 3 runs instead of 2?

Solution 3. If we only do 1 run, it is the same as first one except you are checking every other step.

If we do 3 runs, we get difference recurrence equations, denote T' the time complexity of the 3-runs version c' denote the time to run 3 Modified Karger:

$$T'(n) = 3T'\left(n/\sqrt{2}\right) + c'n^2.$$

$$\begin{aligned}
T(n) &= 9T\left(\frac{n}{2}\right) + 3c' \cdot \left(\frac{n}{\sqrt{2}}\right)^2 + c'n^2 \\
&= 9T\left(\frac{n}{2}\right) + \frac{3}{2}c'n^2 + c'n^2 \\
&= 27T\left(\frac{n}{2\sqrt{2}}\right) + 9c'\left(\frac{n}{2}\right)^2 + \frac{3}{2}c'n^2 + c'n^2. \\
&= \sum_{k=1}^{t=O(\log_2 n)} \left(\frac{3}{2}\right)^{k-1} \cdot c'n^2 \\
&= c'n^2 \sum_{k=1}^{t=O(\log_2 n)} \left(\frac{3}{2}\right)^{k-1} \\
&= c'n^2 \frac{1\left(1-\left(\frac{3}{2}\right)^t\right)}{1-\frac{3}{2}} \\
&= 2c'n^2\left(\left(\frac{3}{2}\right)^{\log_2(O(n^2))} - 1\right) \\
&= 2c'n^2\left(\left(\frac{3}{2}\right)^{\log_{3/2}(O(n^2))\cdot\log_2(3/2)} - 1\right) \\
&= 2c'n^2((O(n^2))^{\log_2(3/2)} - 1) \\
&= O(n^{2\log_2 3}).
\end{aligned}$$

A slightly larger runtime if we use the same threshold.

Problem solving

Problem 4. Show how to, given as input a (multi)graph $G = (V, E)$ in either the adjacency matrix or adjacency list representation, to sample an edge uniformly at random in time $O(n)$, where $n = |V|$.

Solution 4. Suppose the data structure encodes the degree of any vertex, i.e., query to degree of any vertex V_i takes $O(1)$ operation. Query every vertex's degree and put into an array. We can then do the following:

1. Sample with probability $\Pr[V_i] = \frac{D_i}{\sum_{i=1}^n D_i}$, where D_i denote the degree of V_i.
2. Given sampled V_i, sample one of its neighbouring edges with probability $\frac{1}{D_i}$.

Combined, each edge could be selected if the algorithm selects one of its end points, and then choosing it. Without loss of generality, let $e = (i, j)$:

$$\Pr[T = e] = \frac{D_i}{\sum_{i=1}^n D_i} \cdot \frac{1}{D_i} + \frac{D_j}{\sum_{i=1}^n D_i} \cdot \frac{1}{D_j} = \frac{2}{\sum_{i=1}^n D_i} = \frac{2}{|E|}.$$

Also there is a sampling algorithm on adjacency matrix with expected runtime of $O(n)$, when $m \geqslant n - 1$ via rejection sampling: sample coordinates u.a.r. and if it hits one of the edges in the graph, return that edge; otherwise, repeat. The probability to hit one edge each time is $\frac{m}{n^2}$ and so in expectation, the runtime is $\frac{n^2}{m} = O(n)$ as $m \geq n - 1$ (the graph is connected).

Adjacency list representation: Keep list of each vertex's degree, and total number of edges m. Pick i in $[m]$ u.a.r. Given the degrees, we can find in $O(n)$ time where the i-th edge start from. We then, given that vertex, can find the right (i'-th) neighbor in $O(n)$ time.

Problem 5. Consider the following generalisation of MIN-CUT:

> k-MIN-CUT: Given an (undirected) connected graph $G = (V, E)$ on n vertices and m edges ad an integer $k \geq 2$, output a k-cut $(A_1, \ldots, A_k)$ (partition of V) *minimising* the number $c_k(A_1, \ldots, A_k)$ of edges between the different connected components $A_1, \ldots, A_k$.

a) Adapt (the basic version of) Karger's algorithm to solve this problem.

b) Analyse the success probability and running time.

c) Provide a bound on the maximum number of k-Min-Cuts a graph can have.

Solution 5. Consider k a constant.

a) Contract until $4k$ vertices are left. Brute force (takes $O(k^{4k})$ time – constant time :))

b) Denote $C = (A_1, A_2, \ldots, A_k)$ the minimum cut. Similarly, we look at the iterative process and denote the contracted graph at each step i, (V_i, E_i), for $i = 0, \ldots, n - k$. For each V_i, we can partition V_i into $L = \left\lceil \frac{|V_i|}{k} \right\rceil$ sets of at most size k, denote them $S_1, \ldots, S_L$ (and $S_1, \ldots, S_{L-1}$ have size k; S_L could be less than that). For $j \leqslant L - 1$, suppose sum of S_j's degree is less than $|C|$, then we can cut out every vertex in S_i and get a better cut than C, which is a contradiction.

$$|E_i| = \frac{1}{2} \sum_{v \in V_i} \text{degree}(v) = \frac{1}{2} \sum_{j=1}^{L} \sum_{v \in S_j} \text{degree}(v) \geqslant \frac{1}{2} \left\lfloor \frac{|V_i|}{k} \right\rfloor |C| \geqslant \frac{1}{2} \left(\frac{|V_i|}{k} - 1 \right) |C|.$$

Therefore,

$$\frac{|C|}{|E_i|} \leqslant \frac{|C|}{\frac{1}{2} \left(\frac{|V_i|}{k} - 1 \right) |C|} = \frac{2k}{|V_i| - k}.$$

Therefore, the probability of getting the cut C (the algorithm will choose edges until $n-k-1$),

$$\begin{aligned}
\Pr\left[C \text{ is preserved}\right] &= \prod_{i=0}^{n-4k-1}\left(1-\frac{|C|}{|E_i|}\right) \\
&\geqslant \prod_{i=0}^{n-4k-1}\left(1-\frac{2k}{|V_i|-k}\right) \\
&= \prod_{i=0}^{n-4k-1}\left(1-\frac{2k}{(n-i)-k}\right) \\
&= \prod_{i=0}^{n-4k-1}\left(\frac{n-i-3k}{n-i-k}\right) \\
&= \frac{n-3k}{n-k}\cdot\frac{n-1-3k}{n-1-k}\cdots\frac{n-5k}{n-3k}\cdots\frac{4k+1-2k}{4k+1}\cdots\frac{4k+1-3k}{4k+1-k} \\
&= \frac{3k+2}{n-k}\cdots\frac{k+1}{n-3k+1} \\
&\geqslant \Omega\left(\frac{1}{n^{2k}}\right).
\end{aligned}$$

For large enough $n \gg k$. Repeating this $O(n^{2k}\log(1/\delta))$ times and pick the best one suffices.

c) There are at most $O(n^{2k})$ unique values to return in step 2. Suppose every k-cut in $4k$ vertices are minimum, so we get at most $O(k^{4k}n^{2k}) = O(n^{2k})$:)

We can probably do better for problem c). Consider the same setting, at step i, we have V_i. We know that for any subset $S \subseteq V_i$ of size k, we have that

$$\sum_{v\in S}\text{degree}(v) \geqslant |C|.$$

We can do something silly: make k copies of V_i, i.e., let $Y_j = V_i$ and $Y_1,\ldots,Y_k$. We have in total, $k\cdot|V_i|$ number of vertices' degree to sum over now; and we can rearrange such that all of them form distinct subset with size k.

$$|E_i| = \frac{1}{2}\sum_{v\in V_i}\text{degree}(v) = \frac{1}{2k}\sum_{v\in Y_1\cup\cdots\cup Y_k}\text{degree}(v) \geqslant \frac{1}{2k}|V_i||C|.$$

1	2	3	$\cdots$	k	$\cdots$	n
1	2	3	$\cdots$	k	$\cdots$	n
1	2	3	$\cdots$	k	$\cdots$	n
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
1	2	3	$\cdots$	k	$\cdots$	n

Table 1: We can find one such arragement through the diagonals.

Redo the probability calculation, we get

$$\begin{aligned}
\Pr[C \text{ is preserved}] &= \prod_{i=0}^{n-2k-1}\left(1-\frac{2k}{|V_i|}\right)\\
&\geqslant \prod_{i=0}^{n-2k-1}\left(\frac{n-i-2k}{n-i}\right)\\
&= \frac{n-2k}{n}\frac{n-1-2k}{n-1}\cdots\frac{n-4k}{n-2k}\cdots\frac{2k+1}{4k+1}\cdots\frac{1}{2k+1}\\
&= \frac{2k\cdot(2k-1)\cdots 1}{n\cdot(n-1)\cdots(n-2k+1)}\\
&= 1/\binom{n}{2k}\\
&\geqslant \Omega\left(\frac{1}{n^{2k}}\right).
\end{aligned}$$

c) There are at most $O(n^{2k})$ possible unique k-min-cut.

Problem 6. Consider the following algorithm:

1. Draw, independently for every edge $e \in E$, a weight w_e in $[0,1]$ uniformly at random.
2. Build the MST of $G = (V, E, w)$ (according to these weights)
3. Remove the heaviest edge of the MST, and let A, B be the resulting 2 components.
4. Return (A, B) as cut.

Show that this is equivalent to Karger's algorithm (the "basic" version). Deduce how to implement this algorithm in time $O(m \log m)$.

Solution 6. The idea is to run Kruskal's algorithm and it is equivalent to the contraction algorithm: you pick a random ordering of the edges, so when you go through the edges one by one in Kruskal this is like sampling an edge u.a.r.

At each step i, you have all the edges with weights uniformly and independently sampled from $[0,1]$ (conditioned on the edges picked before, but that doesn't matter because of independence) and so this gives you a uniformly random ordering each time.

The step of Kruskal is basically a contraction:

- if that creates a cycle, this means your edge was already in a "supervertex" so you cannot contract it
- otherwise, you contract it When you exclude edges (those edges are picked by MST algo), the rest is not affected (can still be seen as random ordering among themselves).

It is like contraction each time MST picks one edge in. Except it contract all of them and at the end, you cut one to reverse that. Well, it has to be connected at first to run MST. So $|V| - 1 \leq m$. 1. Go through edges, generate random numbers $O(m)$. 2. Run MST $O(m \log |V|) = O(m \log m)$.

See also Exercise 2 of https://www.fundamentalalgorithms.com/f20/notes/min-cut.pdf

Advanced

Problem 7. Prove that (a suitable modification of) Karger's algorithm still works for weighted graphs (with non-negative weights). Do the same for the Karger–Stein algorithm.

Solution 7. Pick e with $\Pr\left[\frac{w_e}{\sum_{e\in E} w_e}\right]$ instead. At each time $i = 0, \dots, n-1$, V_i, E_i. Denote $W = \sum_{u\in A, v\in B} w(u,v)$, the weight of the min-cut. Let $C := \{(u,v) | u \in A, v \in B\}$.

The probability that it gets cut at step i, is

$$\Pr\left[\text{algo picks } e \in C\right] = \Pr\left[\frac{\sum_{e'\in C} w_{e'}}{\sum_{e\in E_i} w_e}\right].$$

The min weights around any one vertex is at least W, otherwise, contradiction. Then

$$\sum_{e\in E_i} w_e \geqslant \frac{1}{2}|V_i| \cdot W \Rightarrow \frac{\sum_{e'\in C} w_{e'}}{\sum_{e\in E_i} w_e} \leqslant \frac{2W}{|V_i| \cdot W} = \frac{2}{|V_i|}.$$

The rest follows the proof from the lecture notes (i.e., is similar to the unweighted case)

$$\Pr\left[C \text{ is returned}\right] \;=\; \prod_{i=0}^{n-2}\left(1 - \Pr\left[\frac{\sum_{e'\in C} w_{e'}}{\sum_{e\in E_i} w_e}\right]\right)$$