CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2023**

Homework 1 Solutions

September 28, 2023

Problem 1. In this question, we examine a way of speeding up Karger's minimum cut algorithm, due to Karger and Stein (JACM 1996). Basically, the issue with Karger's algorithm is that it "waits too long" before repeating the algorithm. In other words, we had the basic contraction algorithm that succeeds with probability roughly $1/n^2$, and then we repeat it $O(n^2)$ to boost the probability to a constant.

But, the probability that the basic contraction algorithm fails in its first step is very low—only $2/n$—, which is good for us, while the probability that it fails in its last step is very high—already $1/3$—which is undesirable. Karger-Stein algorithm is exploiting this phenomenon cleverly by *interjecting* repetition steps in the middle of basic contraction algorithm. Basically, we are unlikely to destroy the min-cut in the early steps of the contraction algorithm, so why repeat all these steps altogether?

(a) Show that each contraction operation can be implemented in $O(n)$ time. Conclude that the original Karger's algorithm that succeeds with probability at least $2/3$ can be implemented in $O(n^4)$ time.

(5 points)

Solution. Suppose we store the input graph in the adjacency list representation. We can sample an edge uniformly at random from the graph by sampling a vertex v with probability $\deg(v)/2m$ and then sample one of the neighbors of this vertex uniformly at random. Thus, any edge (u, v) will be sampled with probability

$$\frac{\deg(u)}{2m} \cdot \frac{1}{\deg(u)} + \frac{\deg(v)}{2m} \cdot \frac{1}{\deg(v)} = \frac{1}{m},$$

which is uniform as desired. To contract an edge (u, v), we simply need to (1) remove u and v and $N(u)$ and $N(v)$ from the adjacency list representation, (2) place $N(u)$ and $N(v)$ (without u and v) in a new list assigned to a new vertex (corresponding to the contraction of u and v), and (3) compute the degree of this new vertex. All of this can be done in $O(|N(u)| + |N(v)|)$ time trivially which is $O(n)$.

Given Karger's contraction algorithm involves running $n - 2$ contraction step, one run of the basic algorithm takes $O(n^2)$ time and succeeds with probability at least $1/n^2$ as shown in the lectures. Repeating the algorithm $\ln(3) \cdot n^2$ times and returning the smallest cut found fails with probability

$$\leqslant (1 - \frac{1}{n^2})^{\ln(3)\cdot n^2} \leqslant \exp(-\ln(3) \cdot n^2 \cdot \frac{1}{n^2}) = \frac{1}{3};$$

thus, the algorithm succeeds with probability at least $2/3$ and takes $O(n^4)$ time.

(b) Karger's basic contraction algorithm runs $n - 2$ contractions consecutively, and ends with a graph on 2 vertices, which preserves a *fixed* minimum cut $(S, V \setminus S)$ with probability at least $2/n \cdot (n - 1)$.

Instead, consider running only $n - n/\sqrt{2}$ random contraction steps. Prove that the probability that a fixed minimum cut survives this contraction process is at least $1/2$. **(5 points)**

Solution. As pointed out on LEARN, we will only run the contraction steps $n - n/\sqrt{2} - 1$ steps instead (which does not change the subsequent parts of the problem). The probability of success is

$$\begin{aligned}\prod_{i=1}^{n-n/\sqrt{2}-1}\left(\frac{n-i-1}{n-i+1}\right) &= \left(\frac{n-2}{n}\right)\cdot\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{n-(n-n/\sqrt{2}-1)-1}{n-(n-n/\sqrt{2}-1)+1}\right)\\ &= \frac{1}{n}\cdot\frac{1}{n-1}\cdot\Big(n-(n-n/\sqrt{2}-1)\Big)\cdot\Big(n-(n-n/\sqrt{2}-1)-1\Big)\\ &\qquad\text{(by the telescoping canceling)}\\ &= \frac{1}{n\cdot(n-1)}\cdot(n/\sqrt{2}+1)\cdot(n/\sqrt{2})\geqslant\frac{1}{2},\end{aligned}$$

as desired.

(*c*) Consider the following Karger-Stein algorithm: starting from a multi-graph G on n vertices, randomly contract edges until $n/\sqrt{2}$ vertices remain; call the new graph G'. Recursively, run *two* copies of the algorithm *independently* on G' and return the smallest of the two cuts found as the answer.

Define the recurrence $T(n)$ as the worst-case runtime of the above algorithm on n-vertex graphs. Prove:

$$T(n)\leqslant O(n^2)+2\cdot T(n/\sqrt{2}).$$

Use this recurrence to bound the runtime of the algorithm with $O(n^2\cdot\log n)$ time. **(7.5 points)**

Solution. The algorithm involves $\leqslant n$ contraction steps and thus by part (a) takes $O(n^2)$ time outside the recursions and then has two recursive calls on a graph with $n/\sqrt{2}$ vertices. Hence, the recurrence is correct. Using the recursion tree method, we have that

$$T(n)\leqslant\underbrace{c\cdot n^2+2\cdot c\cdot(n/\sqrt{2})^2+4\cdot c\cdot(n/\sqrt{2})^4+\dots}_{\text{for depth }\log_{\sqrt{2}}(n)}=\log_{\sqrt{2}}(n)\cdot c\cdot n^2=O(n^2\log n).$$

(*d*) Let $P(n)$ denote the worst-case probability that the above algorithm returns a minimum cut of a given n-vertex graph. Prove:

$$P(n)\geqslant\frac{1-(1-P(n/\sqrt{2}))^2}{2}.$$

Use this recurrence to bound the probability of success of the algorithm with $\Omega(1/\log n)$. **(7.5 points)**

Solution. To get to $n/\sqrt{2}$ vertices, we are running $n-n/\sqrt{2}-1$ contraction steps and hence, by part (b), with probability $1/2$ the minimum cut is preserved. After this, we need at least one of the two recursive calls to preserve the minimum cut. The probability that this does *not* happen is:

$$(1-P(n/\sqrt{2}))^2,$$

which gives us the desired recurrence relation. We now solve this recurrence by assuming inductively that $P(n) \geqslant c/\log n$ for some constant $c < 1$. We have,

$$\begin{aligned}
P(n) &\geqslant \frac{1-(1-P(n/\sqrt{2}))^2}{2} \\
&\geqslant \frac{1-(1-\frac{c}{\log(n/\sqrt{2})})^2}{2} && \text{(by induction step)} \\
&= \frac{1-1-\frac{c^2}{\log^2(n/\sqrt{2})}+\frac{2c}{\log(n/\sqrt{2})}}{2} \\
&= \frac{c}{\log n - 1/2} - \frac{c^2}{2\cdot(\log n - 1/2)^2} && \text{(as } \log(1/\sqrt{2}) = -1/2\text{)} \\
&= \frac{c}{\log n} + \frac{2c\cdot(\log n - 1/2)\log n - c^2\cdot\log n - 2c\cdot(\log n - 1/2)^2}{2\cdot(\log n - 1/2)^2\log n} \\
& && \text{(by adding and subtracting } c/\log n \text{ to get this term separately)} \\
&= \frac{c}{\log n} + \frac{c\log n - c^2\log n - c/2}{2\cdot(\log n - 1/2)^2\log n} \\
&\geqslant \frac{c}{\log n}. && \text{(as the second term is positive for } c<1 \text{ and sufficiently large } n\text{)}
\end{aligned}$$

This proves $P(n) = \Omega(1/\log n)$ as desired (for "small" n, we always have $P(\Theta(1)) = \Theta(1)$).

The conclusion is that we found an algorithm with runtime $O(n^2 \cdot \log n)$ that solves the minimum cut problem with probability $\Omega(1/\log n)$. As in the class, we can repeat this algorithm $O(\log n)$ time and return the best answer to succeed with probability at least $2/3$ in $O(n^2 \cdot \log^2 n)$ time. This is a pretty fast algorithm now! (on dense graphs, this is just tiny bit slower than reading the input graph itself).

Problem 2. A family of functions $\mathcal{H} = \{h \mid h : [n] \mapsto [m]\}$ is called a **pairwise independent hash family** iff for all $x \neq y \in [n]$ and $a, b \in [m]$, for h chosen uniformly at random from $\mathcal{H}$, we have,

$$\Pr(h(x) = a \wedge h(y) = b) = \frac{1}{m^2}.$$

In this problem, we investigate one method to obtain such a hash family (although this is *not* the most efficient method).

(*a*) Let $X_1, X_2, \ldots, X_k$ be independent and uniform binary random variables, i.e.,

$$\Pr(X_i = 0) = \Pr(X_i = 1) = 1/2.$$

Given $S \subseteq \{1, \ldots, k\}$, $S \neq \emptyset$, define a random variable $Y_S = \oplus_{i\in S} X_i$, i.e., the XOR of X_i's for $i \in S$. Prove that the set

$$\{Y_S \mid S \subseteq \{1, \ldots, k\}, S \neq \emptyset\}$$

is a collection of **pairwise independent random variables**, meaning that for all non empty sets $S \neq T \subseteq \{1, \ldots, k\}$ and pairs $a, b \in \{0, 1\}$:

$$\Pr[Y_S = a \wedge Y_T = b] = \frac{1}{4}.$$

(12.5 points)

Solution. Fix a non-empty set $S \subseteq \{1, \ldots, k\}$. For any index $s \in S$, we have,

$$Y_S = \oplus_{i\in S} X_i = (\oplus_{i\in S\setminus\{s\}} X_i) \oplus X_s.$$

If the values of all X_i for $i \in S \setminus \{s\}$ are determined already, then the value of Y_S will be equal to $(\oplus_{i \in S\setminus\{s\}} X_i)$ or differ from it based on each value of X_s. Since $\Pr[X_s = 1] = \Pr[X_s = 0] = 1/2$ and all the X_i's are independent, Y_S is uniformly distributed in $\{0, 1\}$.

Now consider two non-empty sets S and T, $S \neq T \subseteq \{1, \ldots, k\}$. We have:

$$Y_S = Y_{S\cap T} \oplus Y_{S\setminus T}$$
$$Y_T = Y_{S\cap T} \oplus Y_{T\setminus S}$$

Moreover, $Y_{S\cap T}$, $Y_{S\setminus T}$ and $Y_{T\setminus S}$ are independent of each other because they rely on disjoint subsets of X_i's. Since $S \neq T$, at least two of these subsets are non-empty and therefore the pair (Y_S, Y_T) is uniformly distributed in $\{0, 1\} \times \{0, 1\}$.

(b) Use the construction in part (a) to design a family of pairwise independent hash functions

$$\mathcal{H} = \{h \mid h : [n] \mapsto [m]\}$$

constructed using $O(\log n \log m)$ random bits and $\text{poly}(\log n, \log m)$ time to compute. **(12.5 points)**

Solution. For simplicity, we assume m is a power of two in this question. Create $M := \log m$ independent collections of pairwise independent random variables each of size n:

$$\mathcal{Y} := \left\{(Y_1^1, \ldots, Y_n^1), (Y_1^2, \ldots, Y_n^2), \ldots, (Y_1^M, \ldots, Y_n^M)\right\}.$$

We define a family of functions $\mathcal{H}$ where each instance h of $\mathcal{H}$ is created as follows. We first sample a valuation for random variables in $\mathcal{Y}$. For any number $x \in [n]$, we define the bit-representation of $h(x)$ as $(Y_x^1, Y_x^2, \ldots, Y_x^M)$. To prove that $\mathcal{H}$ is indeed a family of pairwise independent hash functions, we argue for all $i \neq j \in [n]$ and $k, \ell \in [m]$:

$$\Pr_{h\in\mathcal{H}}(h(i) = k \wedge h(j) = \ell) = \frac{1}{m^2}.$$

Assume bit representation of k and ℓ are $(a_1, a_2, \ldots, a_M)$ and $(b_1, \ldots, b_M)$. By definition of $\mathcal{H}$, we can write the above probability as:

$$= \Pr[(Y_i^1, Y_i^2, \ldots, Y_i^M) = (a_1, \ldots, a_M) \wedge (Y_j^1, Y_j^2, \ldots, Y_j^M) = (b_1, \ldots, b_M)]$$
$$= \Pr[(Y_i^1 = a_1 \wedge Y_j^1 = b_1) \wedge (Y_i^2 = a_2 \wedge Y_j^2 = b_2) \wedge \ldots \wedge (Y_i^M = a_M \wedge Y_j^M = b_M)]$$

By independence of collections in $\mathcal{Y}$, the above probability is

$$= \Pr[Y_i^1 = a_1 \wedge Y_j^1 = b_1] \cdots \Pr[Y_i^M = a_M \wedge Y_j^M = b_M]$$

And finally by pairwise independence, each term is $1/4$ and thus,

$$= (\frac{1}{4})^M = \frac{1}{m^2}.$$

Now observe that we can create $\mathcal{Y}$ implicitly by using the idea in part (a) and create each set of pairwise independent random variables using $N = \lceil \log n \rceil + 1$ truly independent random variables $X_1, \ldots, X_N$. Given any x, we can define $S_x = \{i \mid i\text{'th bit of } x \text{ is } 1\}$ and then evaluate Y_{S_x} as described in part (a). This way the representation size of each function in $h \in \mathcal{H}$ will be $O(\log n \log m)$ bits and $h(x)$ can be computed in $\text{poly}(\log n, \log m)$ time.

Problem 3. Consider a complete tree of height h, wherein the root, as well as any internal node has exactly 3 child-nodes; thus, the tree has $n = 3^h$ nodes. Suppose each leaf of the tree is assigned a Boolean value. We define the value of each internal node as the *majority* of the value of its child-nodes. The goal in this problem is to determine the value of the root.

An algorithm for this problem is provided with the structure of the tree (not the valuation of the leaves) and at each step it can *query* a leaf and read its value.

(a) Show that for any deterministic algorithm, there is an instance (a set of Boolean values for the leaves) that forces the algorithm to query all the $n = 3^h$ leaves.

(10 points)

Solution. We claim that for a tree of height h, an adversary can choose the values of leaves adaptively as the deterministic algorithm queries them in such a way that until the last leaf is queried, the value of the root is indeterminate. We prove the claim by induction.

The base case is a tree of one node and the claim is trivial. For the inductive step on a tree of height $h+1$, the adversary works as follows. Let T_1, T_2, T_3 be the subtree of the root. By induction, we assume there exists a sub-adversary for each of these subtrees that makes the value of the root indeterminate until all the leaves are queried. We say a subtree is queried completely iff all the leaves in the subtree is queried. Therefore, until at least one of the three subtrees is queried completely, value of no child of root is determined. Now assume one of the subtrees is queried completely, then the sub-adversary for that subtree commits to a value, say 0. The other two sub-adversaries are still uncommitted. Eventually a second subtree will be queried completely. This time the sub-adversary commits to value of 1 for this subtree. Thus the overall value of the original tree is determined by the value of last sub-tree, which is only determined after all its leaves are queried. This completes the inductive step.

This way, for any deterministic algorithm, we can use this adversary to find an input that forces the algorithm to query all leaves, concluding the proof.

(b) Consider the recursive randomized algorithm that evaluates two subtrees of the root chosen at random. If the values returned disagree, it proceeds to evaluate the third subtree. Show that the expected number of the leaves queried by the algorithm on any instance is at most $n^{0.9}$. **(15 points)**

Solution. Consider the three subtrees of the root. If their values are equal, regardless of the order, the algorithm will only evaluate two of them. Otherwise, if two subtrees have the same value and the value of the third one is different. Then, the probability that the algorithm needs to recurse on all three subtrees is the same as the probability of picking two different values out of the three, or equivalently, leave out the one odd number odd. This happens with probability 1/3.

Let $T(h)$ be the expected number of the leaves the algorithm have to query for a tree of height h. Then we have:

$$T(h) \leqslant 2 \cdot T(h-1) + \frac{2}{3} \cdot T(h-1)$$

(we always need to query two subtrees, and with probability at most 2/3 we need to query the third one also)

$$= \frac{8}{3} \cdot T(h-1)$$

$$= (\frac{8}{3})^h \leqslant n^{\log_3 (\frac{8}{3})} < n^{0.9}.$$

Problem 4. Given a set of numbers S and a number $x \in S$, the **rank** of x is defined to be the number of elements in S that have value at most x:

$$rank(x, S) = |\{y \in S : y \leqslant x\}|$$

Given a parameter $\varepsilon \in (0, 1/2]$, we say that an element $x \in S$ is an ε**-approximate element of rank** r if

$$(1 - \varepsilon) \cdot r \leqslant rank(x, S) \leqslant (1 + \varepsilon) \cdot r$$

Recall the streaming model of computation discussed in the class. Suppose we are given a stream of numbers $S = (s_1, s_2, \ldots, s_n)$, where $s_i \in [m]$ for $i \in [n]$, and assume that all s_i's are distinct. Our goal is to design an $O(\varepsilon^{-2} \log m \log n)$ space streaming algorithm for retrieving an ε-approximate element for any given rank value.

(*a*) Recall that the median of a set S of n (distinct) elements is the element of rank $r = \lfloor n/2 \rfloor$ in S.

Consider this algorithm for computing an ε-approximate median: sample $O(\varepsilon^{-2} \log n)$ numbers from the stream uniformly at random (with repetition) and then return the median of the sampled numbers. Prove that this algorithm returns an ε-approximate median with probability at least $1 - 1/\text{poly}(n)$.

(12.5 points)

Solution. Let L be the set of numbers in stream S with rank less than $(1 - \varepsilon)n/2$ and G be the set of numbers with rank greater than $(1 + \varepsilon)n/2$. Note that any element of $S \setminus (L \cup G)$ is a valid answer for the problem.

Assume we are sampling $t = \lceil 24\varepsilon^{-2} \log n \rceil$ numbers from the stream. Therefore, each number is appearing in the sample with probability t/n. There are two bad events that may result in the failure; either, at least $t/2$ numbers are sampled from L, or at least $t/2$ numbers are sampled from G. Since size of G and L are equal, the probability of these events are equal and thus we only bound the probability of the first bad event.

Let X_i for $i \in [n]$ be a random variable indicating if s_i is sampled. Define X as the random variable counting the numbers sampled from L. We have:

$$\mathbb{E}[X] = \sum_{i \in L} X_i = (1 - \varepsilon)\frac{n}{2} \cdot \frac{t}{n} = (1 - \varepsilon)\frac{t}{2}.$$

Variable X is a sum of independent random variables and we can use Chernoff bound,

$$\begin{aligned} \Pr[X \geqslant \frac{t}{2}] &= \Pr[X \geqslant \mathbb{E}[X]/(1 - \varepsilon)] \\ &\leqslant \Pr[X \geqslant (1 + \varepsilon)\mathbb{E}[X]] \\ &\leqslant \exp\left(\frac{-\varepsilon^2 \cdot \mathbb{E}[X]}{3}\right) \leqslant \frac{1}{n^2}. \end{aligned}$$

Therefore, using union bound with probability at least $1 - 2/n^2$, neither of the two bad events happen. This completes the proof.

(*b*) We now extend the previous algorithm to compute an ε-approximate element of rank r for any $r \in [n]$.

Consider this algorithm: Let $t = \lceil 24\varepsilon^{-2} \log m \rceil$. If $r \leqslant t$, then simply maintain a list T of r smallest elements seen in the stream, and output the largest element in T at the end of the stream. Otherwise, choose each element in the stream with probability t/r, and maintain the t smallest sampled values in a list T. At the end of the stream, output the largest number in T. Prove that this algorithm outputs an ε-approximate element of rank r with probability at least $1 - 1/\text{poly}(n)$. **(12.5 points)**

Solution. This time define L as the set of numbers in the stream S with rank smaller than $(1-\varepsilon)r$ and define G as the set of numbers in S with rank smaller than $(1+\varepsilon)r$. Note the difference between the definition of set G in this part compare to part (a). For the algorithm to answer correctly, it should return a number in set $G \setminus L$.

There are two bad events that may contribute to the failure of the algorithm; either at least t numbers are sampled from L, or less than t numbers are sampled from G. If neither of the two bad events happens, then the t_{th} smallest number is guaranteed to be in the set $G \setminus L$. We bound the probability of each of these two bad events separately.

Define X_L and X_G as the random variables counting the numbers sampled from L and G, respectively. The probability that each number s_i is chosen in the sample is t/r and considering the size of L and G we have $\mathbb{E}[X_L] = (1-\varepsilon)\cdot t$ and $\mathbb{E}[X_G] = (1+\varepsilon)t$.

Variables X_L and X_G are sum of independent random variables and we can use Chernoff bound,

$$\begin{aligned}\Pr[X_L \geqslant t] &= \Pr[X_L \geqslant \mathbb{E}[X_L]/(1-\varepsilon)] \\ &\leqslant \Pr[X_L \geqslant (1+\varepsilon)\,\mathbb{E}[X_L]] \\ &\leqslant \exp\left(-\frac{\varepsilon^2\cdot\mathbb{E}[X_L]}{3}\right) \leqslant \frac{1}{m^2}.\end{aligned}$$

And,

$$\begin{aligned}\Pr[X_G < t] &= \Pr[X_G < \mathbb{E}[X_G]/(1+\varepsilon)] \\ &\leqslant \Pr[X_G < (1-\frac{\varepsilon}{2})\cdot\mathbb{E}[X_G]] \\ &\leqslant \exp -\frac{\varepsilon^2\cdot\mathbb{E}[X_G]}{8} \leqslant \frac{1}{m^2}.\end{aligned}$$

Therefore with probability at least $1-2/m^2$ neither of the two bad events happen, and the algorithm returns a correct answer. Finally, note that $m \geqslant n$ as we are picking n *distinct* elements from $[m]$ and so this probability bound is also $1-1/\text{poly}(n)$.