CS 514: Advanced Algorithms II – Sublinear Algorithms **Rutgers: Fall 2021**

Problem set 1

Due: 11:59PM, October 12, 2021

Problem 1. Let us revisit the balls and bins experiment: We are throwing n balls into n bins by choosing a bin uniformly at random and independently for each bin. In Lecture 1, we considered bounding the maximum number of balls in any bin. We now consider a different question in the same setting: bounding the number of *empty* bins.

(a) Prove that the expected number of bins with no ball in them is at least $n/3$. **(10 points)**

(b) Prove that the expected number of bins with no ball in them is at least $n/10$ with high probability (say, with probability at least $1 - 1/n$). **(10 points)**

Hint: You may be tempted to *directly* apply Chernoff bound on the random variable of part (a). This will not work as that random variable is *not* sum of *independent* random variables. Feel free to do a literature search for finding a concentration result that applies here or prove this using an *indirect* application of Chernoff bound.

Problem 2. We are going to design a sublinear time algorithm for estimating the number of triangles, namely, cliques on 3 vertices, in a given undirected graph. We will work with the general query model plus an additional random edge sample query:

- **Degree queries**: Given a vertex $v \in V$, output $\deg(v)$.
- **Neighbor queries:** Given a vertex $v \in V$ and $i \in [n]$, output the i-th neighbor of v or $\perp$ if $i > \deg(v)$.
- **Pair queries:** Given two vertices $u, v \in V$, output whether (u, v) is an edge in G or not.
- **Edge-sample queries:** Return an edge $e \in E$ uniformly at random and independently.

In the following, let n, m, and T denote the number of vertices, edges, and triangles in G, respectively. Consider the following random variable X:

(i) Sample an edge e from G uniformly at random (using an edge-sample query); let u, v be the endpoints of e such that $\text{ID}(u) < \text{ID}(v)$.

(ii) Sample a vertex w from $N(v)$ uniformly at random (using a neighbor-query); return $X = 0$ if $\text{ID}(v) > \text{ID}(w)$, otherwise go to the next line.

(iii) Check whether (u, w) is an edge in G (using a pair query); if so, let $X = m \cdot \deg(v)$ (using a degree query), otherwise let $X = 0$.

We will use this random variable X to design an algorithm for triangle counting.

(a) Prove that $\mathbb{E}[X] = T$. **(10 points)**

(b) Prove that $\text{Var}[X] \leq m \cdot n \cdot T$. **(10 points)**

(c) Let $k = \frac{10 \cdot mn}{\varepsilon^2 \cdot T}$. Consider an algorithm that computes random variable X independently for k times, denoted by $X_1, \ldots, X_k$, and return $Y = \frac{1}{k} \cdot \sum_{i=1}^{k} X_i$. Prove that, **(10 points)**

$$\Pr\left(|Y - T| \geq \varepsilon \cdot T\right) \leq \frac{1}{10}.$$

The above now implies an algorithm with runtime $O(\frac{mn}{\varepsilon^2 \cdot T})$ for estimating the number of triangles to within a $(1 \pm \varepsilon)$ multiplicative factor.

(d) **Bonus part:** There is a caveat with the algorithm above; one needs to know T to know how many times to repeat the random variable X but T is exactly the quantity we want to estimate!

Show that one can increase the runtime of the algorithm above by an $O(\log n)$ factor and removes the assumption of knowledge of T.

Hint: Assume that T is the maximum value possible, say n^3, and run the above algorithm to get an estimate; if the answer was "very" different with the assumption, repeat the process but this time make the guess equal to $n^3/2$ and then $n^3/4$ and so on. **(+10 points)**

General remark: The algorithm in this question does *not* achieve the optimal bounds for the triangle counting problem, which is known to be $O(\frac{m^{3/2}}{\varepsilon^2 \cdot T})$ (the two bounds match only when $m = \Theta(n^2)$ and otherwise the second bound is always better). However, one can use the ideas from the average degree estimation problem to improve the runtime of the above algorithm to match the optimal bounds (up to logarithmic factors).

Problem 3. The *arboricity* of an undirected graph $G = (V, E)$ is a measure of "uniform sparsity" of G. There are multiple equivalent definitions of arboricity[1]. Here, we mention one key definition for our purpose:

- A graph G is said to have arboricity $\alpha(G) = \alpha$ if

$$\alpha = \max_{S \subseteq V,\ |S|>1} \left\lceil \frac{|E(S)|}{|S| - 1} \right\rceil,$$

where $E(S)$ denotes the set of edges with both endpoints in S.[2]

In this problem, we design a sublinear *query* algorithm for this problem in the same query model as that of Problem 2. Namely, we obtain an algorithm that given $G = (V, E)$ via the general query model plus random edge samples (as specified in Problem 2), and a parameter $\varepsilon \in (0, 1)$, outputs an estimate $\widetilde{\alpha}$ such that

$$\Pr\left(|\widetilde{\alpha} - \alpha(G)| > \varepsilon \cdot \alpha(G)\right) < 1/10.$$

(a) Suppose we are told that G has n vertices and m edges. Sample each edge of G independently with probability

$$p := \frac{100}{\varepsilon^2} \cdot \log n \cdot \frac{n}{m},$$

to get a subgraph H of G. Prove that

$$\Pr\left(|p^{-1} \cdot \alpha(H) - \alpha(G)| > \varepsilon \cdot \alpha(G)\right) < 1/10,$$

where the probability is over the choice of H. **(15 points)**

(b) Use the above result to obtain an algorithm with $O(\varepsilon^{-2} \cdot n \log n)$ *queries* to the graph for approximating the arboricity problem. (You do *not* need to bound the *runtime* of the algorithm) **(15 points)**

[1] These equivalences require proof which are not provided here.

[2] This definition perhaps gives an intuition why arboricity is a measure of uniform sparsity as opposed to just sparsity. Think of a graph on n vertices that consists of a clique on $\sqrt{n}$ vertices and no edges on the other vertices. This graph has $O(n)$ edges and thus is generally considered sparse. However, it is clear that arboricity of G is $\Theta(\sqrt{n})$, which is large, as we expected (because the graph we have should not be considered uniformly sparse or "everywhere sparse").

Problem 4. Prove that any *deterministic* algorithm for estimating the number of connected components to within an $\varepsilon \cdot n$ additive factor in the adjacency list model requires $\Omega(n)$ time for some *constant* $\varepsilon \in (0, 1)$.

(20 points)

Bonus part: Prove that any deterministic algorithm for the above problem requires $\Omega(n/\varepsilon^2)$ time when parameter $\varepsilon \in (0, 1)$ can be *sub-constant* (or rule this out by showing that there is a better deterministic algorithm for *every* $\varepsilon > 0$). **(+20 points)**

Problem 5 (Bonus Problem). Recall that $\mathrm{OR}_n : \{0,1\}^n \to \{0,1\}$ denotes the OR function on n variables and $R(\mathrm{OR}_n)$ is the randomized query complexity of OR_n function as defined in Lecture 3.

Prove that $R(\mathrm{OR}_n) = n/2$. **(+20 points)**

Hint: For the upper bound, notice that the algorithm in Lecture 3 that shows $R(\mathrm{OR}_n) \leq 2n/3$ does not make any error when the correct answer is 0 (so informally speaking, it "wastes its error budget"). Change this by simply outputting 1 a certain constant[3] fraction of time randomly and otherwise run the previous algorithm.

For the lower bound, again notice that the distribution we worked with results in $\mathrm{OR}_n(x)$ being uniformly at random over $\{0, 1\}$ when x is chosen from the distribution. See if biasing the input distribution toward one side allows for proving the stronger lower bound.

[3] You need to compute this constant explicitly.