**CS5234: Combinatorial and Graph Algorithms**

# Problem Set 5 (Solutions)

*Due: September 20th, 5:29pm*

## Exercises (and Review) (*Do not submit.*)

**Exercise 1.** Here we give an alternate method for deciding whether a graph is bipartite. As in class, our goal is to design a streaming algorithm. The stream will consist of a sequence of edges, and your goal is to decide whether or not the graph consisting of these edges is bipartite or not. Your algorithm should be deterministic, and it should always be correct.

a. Imagine you are given a graph $G = (V, E)$. (Forget about streaming, for the moment.) Construct a new graph $D = (V', E')$ as follows:

   - For every node $u \in V$, add two nodes $u_1$ and $u_2$ to $V$;.
   - For every edge $(x, y) \in E$, add two edges $(x_1, y_2)$ and $(x_2, y_1$ to $E'$.

   (Draw a picture!) How does the number of connected components in $D$ relate to whether or not $G$ is bipartite? Give a criterion that you can check on $D$ to determine whether $G$ is bipartite. (Try a few examples, and count the connected components in $D$.)

b. Now give a streaming algorithm that determines whether $G$ is bipartite by simulating $D$ and checking the criterion previously specified.

## Standard Problems (to be submitted)

### Problem 1. Counting Triangles

In this problem, we develop an alternate approach to estimating the number of triangles in a stream based on random sampling. Here is the basic idea: choose a possible triangle and check whether or not it appears in your stream; if it appears, estimate that there are a lot of triangles; if not, estimate that there are very few triangles. More specifically, here is the definition of the basic triangle estimator:

**Algorithm 1:** EstimateTriangles($V$)

```
1 Choose one edge e = (u, v) uniformly at random from the stream.
2 Choose a node z uniformly at random from V \ {u, v}.
3 if edges (u, z) and (v, z) appear in the stream after e then
4     return m(n − 2)
5 else
6     return 0
```

**Problem 1.a.** Describe in more detail how you would implement this estimation algorithm (e.g., how you would choose a random edge, etc.). How much space does the estimator require?

**Solution:** You can use reservoir sampling to choose the edge. Whenever the sampling algorithm chooses a new edge from the stream, then it should choose a new random node $z$ and beginning checking for the missing edges again. The total space required is at most $3 \log n + 2$, since you you need $2 \log n$ bits to store the edge, $\log n$ bits to store the randomly chosen node, and two bits to store whether the extra edges have been seen.

**Problem 1.b.** Let $C$ be the value returned by the estimator. Let $T_3$ be the number of triangles in the graph. Show that $\mathrm{E}[C] = T_3$.

**Solution:** Fix a specific triangle $t = (u, v, w)$ that appears in the graph. The probability that the algorithm finds this triangle can be divided into three cases depending on whether the random edge chosen is $(u, v)$ or $(v, w)$ or $(u, w)$. Each of those edges is chosen with probability $1/m$. There is then an addition $1/(n-2)$ probability that the correct third node is chosen. Finally, there is a $1/3$ chance that the two additional edges come after the chosen edge. (Notice that there are three possible positions for the chosen edge: first, second, or third.) Thus, for each possible choice of first edge, there is a probability of $1/(3m(n-2))$ that triangle $t$ is discovered in this manner. Since there are three possible edges to choose (and since the cases are disjoint,), there is a $3/(3m(n-2)) = 1/m(n-2)$ probability that triangle $t$ is discovered.
Now we consider all $T_3$ triangles. The events of choosing triangle $t_1$ versus $t_2$ are disjoint: the algorithm can choose at most one triangle. Thus the probability of choosing *any* triangle is just the sum of the probabilities, i.e, $T_3/(m(n-2))$.
We can now calculate the expected value of $C$:

$$\mathrm{E}[C] = \frac{T_3}{m(n-2)} \cdot m(n-2) + \left(1 - \frac{T_3}{m(n-2)}\right) \cdot 0 = T_3 \tag{1}$$

Thus we have shown that the expected outcome of the algorithm is exactly $T_3$.

**Problem 1.c.** Show that the $\mathrm{E}\left[C^2\right] = T_3(m)(n-2)$, and thus that $\mathrm{Var}[C] \leq T_3(m)(n-2)$. (Hint: you may want to define random variables that indicate whether or not a given triangle was found, and write $C$ in terms of these random variables.)

**Solution:** Let $t_j$ be the indicator random variable that equals 1 if the estimator discovers triangle $j$ and equal to 0 otherwise. (We only consider triangles that actually exist in the graph. We already know (from the previous part) that $\Pr[t_j] = 1/m(n-2)$. We can define $C = \sum_j t_j \cdot m(n-2) = m(n-2)\sum_j t_j$. To calculate the expected value of $C^2$, we first calculate the expected value of the sum squared:

$$\begin{aligned}
\mathrm{E}\left[(\sum_j t_j)^2\right] &= \mathrm{E}\left[\sum_i \sum_j t_i t_j\right] \\
&= \mathrm{E}\left[\sum_i t_i^2 + \sum_i \sum_{j \neq i} t_i t_j\right] \\
&= \sum_i \mathrm{E}\left[t_i^2\right] + \sum_i \sum_{j \neq i} \mathrm{E}\left[t_i t_j\right] \\
&= \sum_i \frac{1}{m(n-2)} + 0 \\
&= \frac{T_3}{m(n-2)}
\end{aligned}$$

Notice that $\mathrm{E}[t_i t_j]$ is always zero because the events are disjoint: the probability of both of them being (simultaneously) 1 is zero, since the algorithm finds at most one triangle. All the summations above are over all the triangles in the graph.
We now observe that $\mathrm{E}\left[C^2\right] = m^2(n-2)^2 \cdot \mathrm{E}\left[(\sum_j t_j)^2\right] = m(n-2)T_3$. The $\mathrm{Var}[C] \leq \mathrm{E}\left[C^2\right]$, and so the claim follows.

**Problem 1.d.** Now consider the following algorithm: run $s$ instances of the triangle estimator, and return the average value. What is the expected value and variance of the resulting output? (An upper bound on variance, as in the previous part, is expected.)

**Solution:** The expected value is unchanged: $T_3$. The variance is reduce by $s$, i.e., $\leq \frac{T_3(m)(n-2)}{s}$.

**Problem 1.e.** Use Chebychev's Inequality to determine a value for $s$ as a function of $n, m$, and $T_3$ such that the algorithm returns a $(1 \pm \epsilon)$ approximation with probability at least $1 - \delta$. (Your answer may include a factor that looks like $\frac{m(n-2)}{T_3}$, even though $T_3$ is unknown in advance.)

*Recall:* Chebychev's Inequality says that for any random variable $X$, $\Pr\left[|X - \mathrm{E}\left[X\right]| \geq t\right] \leq \frac{\mathrm{Var}[X]}{t^2}$.

**Solution:** Let $\bar{C}$ be the output of the averaging algorithm. Chebychev's Inequality says that:

$$\begin{aligned}
\Pr\left[|\bar{C} - T_3| \geq \epsilon T_3\right] &\leq \frac{\mathrm{Var}\left[\bar{C}\right]}{\epsilon^2 T_3^2} \\
&\leq \frac{T_3(m)(n-2)}{s} \frac{1}{\epsilon^2 T_3^2} \\
&\leq \frac{m(n-2)}{s\epsilon^2 T_3}
\end{aligned}$$

Thus, if we choose $s = \frac{m(n-2)}{T_3} \frac{1}{\delta\epsilon^2}$, we will get the final probability at $\leq \delta$.

**Problem 1.f.** Given that $T_3$ is unknown in advance, how might you choose a reasonable sample size? For what sort of graphs does this yield a reasonable estimator? (For what sort of graphs does this yield an inefficient estimator?) How does the computational cost of processing the stream compare to the triangle estimator we saw in class?

**Solution:** This estimator only works for graphs where there are a lot of triangles! Probably the best way to choose the sample size is based on the amount of space that is available. This choice, along with the number of triangles in the graph, then determines the accuracy of your sampler. Of course, unless you know there are going to be a lot of triangles (and hence can get a good lower bound on the needed sample size), you won't know the accuracy of your estimator!
The computational cost is much lower than the estimator from class, in that it only has to look at each edge once for each of the $s$ estimators. The estimator from class, by contrast, required generator a virtual stream of $n - 2$ triangles per edge, and so increased the cost significantly.

**Problem 1.g.** (Optional.) You can reduce the dependence on $\delta$ by running many copies of the averaging-estimator and taking the median. Design an algorithm that runs $s'$ instances of the averaging-estimator. In order to achieve a $(1 \pm \epsilon)$ estimator that is correct with probability at least $1 - \delta$, what are good values for $s$ and $s'$?