Colorado CSCI 5454: Algorithms
Homework 8

Instructor: Bo Waggoner
Due: November 14, 2019 at 11:59pm
Turn in electronically via Gradescope.

Remember to list the people you worked with and any outside sources used (if none, write "none").

Problem 1 (6 points)

Given an unweighted, undirected graph G, let A_G be its adjacency matrix. Using $A_G^3 = A_G \cdot A_G \cdot A_G$, you will give an algorithm to count the number of distinct *triangles* in G.

(A triangle is a triple of vertices (a, b, c) *such that* (a, b), (b, c), *and* (c, a) *are all edges of the graph.*

Part a (4 points) Use A_G^3 to count the number of distinct triangles where (a, b, c) and (b, c, a) are different triangles (in other words, the order of vertices matters).

Solution. The answer is the sum of the diagonal entries of A_G^3, in other words, $\sum_{i=1}^n A_G^3(i,j)$.

Justification: Every length-3 path starting from i and ending at i is a triangle (because it cannot repeat edges or come back to i early and still have length 3). Conversely, every triangle (a, b, c) is a length-3 path $(a, b), (b, c), (c, a)$. So the diagonal entry $A_G^3(i,i)$ counts the number of triangles starting at i and ending at i, and we can just sum all the diagonal entries.

Part b (2 points) Now count the number of distinct triangles where (a, b, c), (b, c, a), (a, c, b), etc are all the same triangle.

Solution. Each triangle now consists of three vertices. There are six orderings of these vertices. All orderings give this same triangle, i.e. six length-three paths. So the answer is $\frac{1}{6}\sum_{i=1}^n A_G^3(i,i)$.

Problem 2 (4 points)

Part a (4 points) You are given access to a list $A_G, A_G^2, A_G^3, \ldots, A_G^n$ of powers of the adjacency matrix an unweighted, undirected graph on n vertices. It takes constant time to query any entry $A_G^t(i,j)$.

Give an $O(\log(n))$ time algorithm to compute the length of the shortest path from vertex s to vertex t using this list.

Hint: first use the list to decide whether there exists any path at all from s *to* t.

Solution. First, there is an important claim: if there is a path from s to t of length k, then there are paths of length $k+2, k+4, k+6, \ldots$. Proof: take the path of length k, then go back-and-forth along the final edge. Each back-and-forth add 2 to the length of the path.

Next claim, from class: If $A_G^c(s,t) > 0$, then there is a path of length exactly c from s to t. Otherwise, there is not.

So we first can check if $A_G^n(s,t) = 0$ and $A_G^{n-1}(s,t) = 0$. If both equal zero, then we return that there is no path at all from s to t in the graph. This follows because if there is such a path, then it has length at most n, and that means there is a path of length either $n-1$ or n.

We can also check if $A_G(s,t) = 1$, i.e. there is an edge from s to t, and if so, we return that the shortest path has length 1.

Now, we use a binary search, always checking for a path of length both c and $c+1$, where c is the midpoint of our interval. Some details follow although these are not needed for points.

To do the binary search, we can initialize bounds $a = 1$ and $b = n$. Set $c = \lfloor (a+b)/2 \rfloor$. Check if $A_G^c(s,t) > 0$, and if so, update the upper bound $b = c$ and recurse. But if not, we also check if $A_G^{c+1}(s,t) > 0$, and if so, we update $b = c+1$ and recurse. If $A_G^c(s,t) = 0$ and $A_G^{c+1}(s,t) = 0$, then we update the lower bound $a = c+1$ and recurse.

We always maintain the invariant that the shortest path from s to t has length strictly greater than a and weakly less than b, and there is a path of length b. And we cut the interval $b - a$ roughly in half each iteration.[1]

Eventually we get to a loop where the bounds are not update, and then we return b. This will be correct because in such a loop, we must have $c+1 = b$ and there is no path of length c, but there is a path of length $c+1 = b$.

Problem 3 (14 points)

(Programming assignment. Attach all code, printed to pdf.)

Part a (2 points) Come up with a nontrivial unweighted, undirected bipartite graph on at least 20 nodes. (No more than 100 is recommended.) This can be completely generated according to some rules or random process; or it can be inspired by some real-life situation or problem you're familiar with. Ensure the graph is connected.

Describe how you came up with the graph, and attach code used to generate it and/or read it into memory.

Solution. There are many possible answers.

Part b (4 points) Let your bipartite graph be $G = (U, V, E)$. Execute 1000 random walks of length 20 with a biased starting distribution, as follows:

[1]There is some slight fuzziness, but one can always argue e.g. that the interval is decreased by 1/3 factor, which still gives $O(\log(n))$ time.

1. With probability $\frac{1}{4}$, start at a uniformly randomly chosen vertex from U. With the remaining probability, start at a uniformly randomly chosen vertex from V.
2. At each step, pick a neighbor of the current vertex uniformly at random and transition to that neighbor.
3. Repeat for 20 vertices.

Generate a plot where the horizontal axis is the step ($t = 1, 2, \ldots, 20$) and the vertical axis is the *fraction of the* 1000 *trials in which that step's vertex was in* U. For eaxmple, if there were 400 trials where the last vertex was in U and 600 trials where the last vertex was in V, then the plot would have value 0.4 above $t = 20$.

Briefly describe what you observe from this plot and why.

Solution. We expected an alternating pattern where about $\frac{3}{4}$ of the trials are in U, then about $\frac{1}{4}$, and so on repeating.

Part c (4 points) (*Lazy* random walk.) Repeat the previous experiment, but modify the random walk as follows. At each time step, with probability $\frac{1}{4}$, remain at the same vertex; otherwise, transition to a uniformly randomly chosen neighbor.

Generate the same plot for this experiment, briefly describe what you observe, whether it differs from the previous experiment, and if so, why.

Solution. We expected that the fraction of steps in U would converge to some number (which depends on the graph). This is because the Markov Chain now has a single stationary distribution.

Part d (4 points) (Limit of lazy walk.) Repeat the previous experiment, but now keep track of how many times the final vertex is vertex 1, 2, etc. Generate a histogram where the horizontal axis is the list of vertices and the vertical axis is the fraction of trials in which the final step of the walk was that vertex.

Then repeat this new experiment, but let each random walk have length only 10. Then repeat, but let each random walk have length 100.

Attach the three histograms. Describe your observations. How does the distribution of the final vertex change as we take more steps?

Solution. More steps should allow more convergence to the stationary distribution. So the 20 plot should probably be "in between" the 10 plot and the 100 plot.

Part e (Bonus: 1 point) Compute the exact stationary distribution of the lazy random walk described above. Generate a histogram of this distribution in the same format as the three plots of the previous question, and briefly compare. Describe your approach to computing this stationary distribution and attach your code (you may use linear algebra software packages).

Solution. It should be closest to the 100 plot (hopefully very close!), since that is the longest random walk.

Problem 4 (6 points)

In the Metropolis-Hastings algorithm, we must construct an undirected graph on the state space. There were three important properties for the graph to satisfy. For each one, below, explain why it is important (or in other words, what goes wrong with the algorithm if it fails).

Part a (2 points) The maximum degree of any vertex, r, must be reasonably small.

Solution. In the algorithm as implemented, the running time depends on the degree because we must get a list of all neighbors of u and iterate through it.[2]

Part b (2 points) If vertices u, v are neighbors in the graph, then $\frac{f(u)}{f(v)}$ should not be too small, and similarly for $\frac{f(v)}{f(u)}$.

Solution. If the ratio is very tiny, then we we are at u, we will only have a tiny chance of transitioning to v. In the worst case, we could get stuck at u and take many steps without transitioning to any neighbor.

Part c (2 points) The graph should have good *mixing time*: a random walk should converge quickly to the stationary distribution.

Solution. Otherwise, we have to run the algorithm for a long long time before the samples we draw are good representatives of the stationary distribution.

Problem 5 (10 points)

[Programming Metroplis-Hastings]

Consider a simplified system of n particles each of which is either in state 1 or state 0. Therefore, the state of the system can be described by a vector $x \in \{0, 1\}^n$, i.e. a list of n bits.

Given a state x, let $p_x = \frac{\sum_i x_i}{n}$. In other words, p_x is the fraction of particles in state 1.

[2]TA Charlie points out that a more sophisticated implementation of the algorithm may be able to scale well despite large r, if we have a way to sample a random neighbor of u and count its neighbors without explicitly listing them all out.

A physicist assures you that the probability $\pi(x)$ of any given state x is proportional to the *binary entropy* of p_x. The binary entropy function is

$$H_2(p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}.$$

Part a (2 points) First, the physicist would like to sample configurations x from the distribution π described above. You will help her using Metropolis-Hastings.

First, describe an appropriate undirected, unweighted graph on the possible configurations of the system. How many states are there? Given a state x, how can one compute the list of its neighbors? What is the maximum degree r?

Solution. One good answer is the Boolean hypercube. The vertices are all bit strings $x \in \{0,1\}^n$. There are 2^n vertices ("states"). Given x, the neighbors are all the strings we get by flipping one bit and keeping the rest of x the same. Each vertex has exactly n neighbors, so $r = n$.

Part b (4 points) Implement the Metropolis-Hastings algorithm for the physicist. Your code should work for any n and any number of trials T. Attach the source code to the pdf.

Part c (4 points) For $n = 100$, run your sampling algorithm for at least $T = 10000$ trials.

To collect some statistics about the sampled states, store the fraction of ones, p_x, in each of the T trials Then plot a histogram of all the p_x variables.

Solution. We expected a bell curve that is highly concentrated around 0.5, because most of the vertices of the graph have p_x about 0.5 and also H_2 puts higher weight on vertices with p_x close to 0.5.

Part d (Bonus: 1 point) Write down and plot the correct distribution over p_x when x is drawn from π. *Hint: how many states are there with k ones and $n-k$ zeros?* Compare to your Metropolis-Hastings histogram.