CS:3330 Spring 2017: Homework 4

Due at the start of class on Tue, Feb 21

Vertex Cover Algorithms

In this assignment you will implement the different algorithms we discussed in class for the *minimum vertex cover (MVC)* problem and analyze their performance both in terms of running time and the cost (size) of solution that they return. You are expected to implement the following three algorithms. Each of these algorithms should be implemented as a function take as an input parameter the *adjacency list* representation of a graph; each algorithm should return the vertex cover (which a set of vertices) that the algorithm computed.

- GreedyDegreeBased algorithm: this is the algorithm that repeatedly picks a vertex with highest degree in currently active graph; once a vertex is picked, it is added to the solution and all active edges incident on this vertex become inactive.
- MaximalMatchingBased algorithm: this is the 2-approximation algorithm that starts by computing a maximal matching of the graph and then adds to the solution both endpoints of each edge in the maximal matching.
- BruteForce algorithm: this is the algorithm that generates all vertex-subsets and checks each generated vertex-subset for being a vertex cover; eventually (after a long, long time) this algorithm returns a smallest vertex cover.

Programming Problems

1. Write a function `InputGraph` which creates and returns an adjacency list of a graph by reading an input file of the following format–

   The first line of the input contains two positive integers n and m separated by a space denoting the number of nodes and edges respectively in the graph. The subsequent m lines will each contain two integers u and v (in the range 1 through n) separated by a space. Each such line denotes an edge between vertices u and v. For example, a particular input, representing a graph with 4 vertices and 5 edges, could look as follows:

   ```
   4 5
   1 2
   1 3
   1 4
   2 4
   3 4
   ```

2. Write a function named `RandomGraph` that takes 5 input parameters, n_1, n_2, p_1, p_2 and p_3 and generates the random graph denoted $G(n_1, n_2, p_1, p_2, p_3)$ which has n_1+n_2 vertices partitioned into two sets V_1, V_2 such that $|V_1| = n_1$ and $|V_2| = n_2$. Here, n_1 and n_2 are nonnegative integers and p_1, p_2 and p_3 are probabilities and therefore in the range 0 through 1. For every pair of vertices $u, v \in V_1$, add an edge connecting u and v with probability p_1. For every pair

of vertices $u, v \in V_2$, add an edge connecting u and v with probability p_2. For every pair of vertices $u \in V_1$ and $v \in V_2$, add an edge connecting u and v with probability p_3. This function should return an adjacency list representing the generated graph.

3. Implement the GreedyDegreeBased algorithm so that it runs in $O((m+n)\log n)$ time on a graph with n vertices and m edges. Problem 4 in HW3 tells you how to do this. Consult the posted solution for an explanation.

4. Implement the MaximalMatchingBased algorithm so that it runs in $O(m+n)$ time on a graph with n vertices and m edges. For this you need to use a graph traversal algorithm such a *breadth first search* or *depth first search* that systematically explores the graph in linear time (i.e., $O(m+n)$ time). Whenever an edge is "visited" your algorithm should check if both endpoints are available (i.e., have not been matched) and if so, it should add that edge to the matching and mark the two endpoints as being unavailable. Note that the input graph may consist of several connected components and so any graph traversal algorithm you use needs to be able to move from one component to the next. If you need to review graph traversal algorithms, you should carefully review the readings on this posted on the course page.

5. Implement the BruteForce algorithm. You will have to think about how to systematically and efficiently generate vertex-subsets. Also, you should implement some basic pruning rules as part of your algorithm. For example, if you have found a vertex cover of size c, there is no need to generate any vertex-subset of size larger than c. Similarly, if you have found that a set C is not a vertex cover, then no subset of C can be a vertex cover and therefore there is no reason to generate any subsets of C.

Experiments

1. **How slow is BruteForce?** To understand the growth of the running time of BruteForce, we want you to run this algorithm on $G(n, 0, 0.5, 0, 0)$ for different values of n. Use values of $n = 20, 22, 24, \ldots, 40$ and for each n generate 10 instances of $G(n, 0, 0.5, 0, 0)$, run BruteForce on all 10 instances, and report the mean running time for each n. Then plot a graph that shows these mean running times as a function of n. In 2-3 sentences, discuss and explain the trend you see in this plot.

2. **How fast are GreedyDegreeBased and MaximalMatchingBased?** To understand the growth of the running times of GreedyDegreeBased and MaximalMatchingBased algorithms, we want you to run these two algorithms on $G(n, 0, 10/n, 0, 0)$ for different values of n. Use values of $n = 10000, 11000, 12000, \ldots, 20000$ and for each n generate 10 instances of $G(n, 0, 10/n, 0, 0)$, run the two algorithms on all 10 instances, and report the mean running times for each n for both algorithms. Then plot a single graph that shows the mean running times of both algorithms as functions on n. In 2-3 sentences, discuss and explain the trends you see in this plot.

   If you have implemented the algorithms GreedyDegreeBased and MaximalMatchingBased correctly, then the bottleneck for this experiment is likely to be the graph generation algorithm and not the vertex cover algorithms. This is because for a graph with n vertices, the natural graph generation algorithm will examine (roughly) $n^2/2$ pairs, which is 50 million pairs for $n = 10,000$. So our suggestion is to use the following idea for graph generation that

approximately generates $G(n, 0, 10/n, 0, 0)$. Note that a graph with n vertices has $n(n-1)/2$ vertex pairs and since we connect each vertex pair with an edge with probability $10/n$, the *expected* number of edges is $5(n-1)$. Let us take this expected number to be the *exact* number of edges. So our goal is to generate $5(n-1)$ edges and we do this by generating each of the $5(n-1)$ edges by selecting the two endpoints of each edge uniformly at random from among the n vertices.

3. **What are the approximation ratios?** Now we focus on the cost of the solution (i.e., size of the vertex cover) produced by the three algorithms. Fix $n = 40$ and $p = 1/4$. Generate 100 instances of $G(n, 0, p, 0, 0)$ and for each graph, compute vertex covers using the three algorithms. Report the average and maximum (over the 100 instances) ratio of the size of vertex cover computed by GreedyDegreeBased to the size of the vertex cover computed by BruteForce. Similarly, report the average and maximum (over the 100 instances) ratio of the size of vertex cover computed by MaximalMatchingBased to the size of the vertex cover computed by BruteForce. For what fraction of the instances did (i) GreedyDegreeBased perform better than MaximalMatchingBased, (ii) GreedyDegreeBased perform worse than MaximalMatchingBased, and (iii) GreedyDegreeBased and MaximalMatchingBased produce vertex covers of the same size?

4. **How large are the computed vertex covers?** Fix $n_1 = 300$ and $n_2 = 700$. Now we vary the values of p in the range $[0, 1]$, starting with $p = 0$ and ending with $p = 1$, with step size $\epsilon = 0.1$. For each of these 11 values of p, generate 10 instances of the graph $G(n_1, n_2, p, p, 1-p)$. Run GreedyDegreeBased and MaximalMatchingBased on these graphs and record the sizes of vertex covers produced. Finally plot a graph that shows the mean vertex cover size (the mean is taken over 10 instances) produced by the two algorithms as a function of p. In 2-3 sentences, discuss and explain the trends you see in this plot.

What to Turn in

You are expected to turn in two items for this homework.

1. A single file named `hw4.ext` containing all your code. The extension "`.ext`" will of course depend on the programming language you use. Your code should be extensively documented. For example, functions should be preceded by comment blocks that tell the reader what the purpose of the function is, variable names should be suggestive of their purpose, etc. The file should also start with a listing of any bugs or issues with your code, sources you used, etc. You are not allowed to share code with classmates, but it is okay to use code available on the web, with appropriate attribution. For example, you might find it convenient to use a class that implements the *max-heap* data structure. This is fine, provide you carefully acknowledge your source. We will set up a dropbox on ICON for you to upload your code file. This needs to be done before class time on Tue, Feb 21st.

2. A single pdf file named `hw4.pdf` containing the results of all your experiments. You should upload this file onto the ICON dropbox *and* bring a printout to class on Tue, Feb 21st. This document should be well-formatted, should contain clear and concise text, and clearly and completely labeled plots. This document should be at most 3 pages long.