CS 630, Fall 2024, Homework 3
Due Wednesday, October 2, 2024, 11:59 pm EST, via Gradescope

Homework Guidelines

Collaboration policy Collaboration on homework problems, with the exception of programming assignments and reading quizzes, is permitted, but not encouraged. If you choose to collaborate on some problems, you are allowed to discuss each problem with at most 5 other students currently enrolled in the class. Before working with others on a problem, you should think about it yourself for at least 45 minutes. Finding answers to problems on the Web or from other outside sources (including generative AI tools or anyone not enrolled in the class) is strictly forbidden.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You must also identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." It is a violation of this policy to submit a problem solution that you cannot orally explain to an instructor or TA.

Typesetting Solutions should be typed and submitted as a PDF file on Gradescope. You may use any program you like to type your solutions. LaTeX, or "Latex", is commonly used for technical writing (`overleaf.com` is a free web-based platform for writing in Latex) since it handles math very well. Word, Google Docs, Markdown or other software are also fine.

Solution guidelines For problems that require you to provide an algorithm, you must provide:

1. pseudocode and, if helpful, a precise description of the algorithm in English. As always, pseudocode should include
   - A clear description of the inputs and outputs
   - Any assumptions you are making about the input (format, for example)
   - Instructions that are clear enough that a classmate who hasn't thought about the problem yet would understand how to turn them into working code. Inputs and outputs of any subroutines should be clear, data structures should be explained, etc.

   If the algorithm is not clear enough for graders to understand easily, it may not be graded.
2. a proof of correctness
3. an analysis of running time and space

You may use algorithms from class as subroutines. You may also use facts that we proved in class.

You should be as clear and concise as possible in your write-up of solutions. A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand.

Problem 1 *Polling (10 points)*

As the elections are coming near new polls are published every day. As part of a research group at BU you are conducting experiments on how belonging to a social group influences ones' vote. For this you will select some known groups and survey every one of its members. To get unbiased results you decide that none of the groups you select can share any members. From the Registrar at BU you get a list of all sanctioned groups within BU along with their member lists. Design an approximation algorithm to select the largest number of groups.

1. Design an approximation algorithm for this problem. (*For proof you have to show that the output of the algorithm indeed yields a valid solution for the problem. Don't forget to analyze the running time.*)

   Solution. We can solve this problem with a reduction to the independent set problem. Each vertex in the graph corresponds to one of the sanctioned groups, and two vertices in the graph have an edge between them if and only if the corresponding groups share any members. A subset of vertices in this graph thus corresponds to a choice of groups. An independent set in this graph corresponds to a choice of groups with no overlapping members. First we need to create the graph. This is done in worst case $O(g^2n)$ if there are g groups and n people. The we can run the greedy approximation algorithm from class for Independent Set.

   Algorithm 1: Independent Groups(M)

   ```
   /* M is a nested list of groups.  M[i] contains the members' list of group i      */
   G ← hash table/* graph representing the input                                      */
   G.keys ← M.keys/* G has the same keys as M.  You can write out this process in a
      for loop or just use this shorter notation                                     */
   for i in M do
       for j in M do
           if M[i] ∩ M[j] == ∅ then
               G[i][j] ← True;
               G[j][i] ← True/* note that 1.  G is undirected, 2.  hash tables don't keep
                  duplicates so that's not a problem                                */
   groups ← GreedyIS(G)/* Greedy Independent Set algorithm from class               */
   return groups
   ```

   correctness: the one-to-one correspondents between groups and Independent Set are clear - groups that share members can't be both selected. Connecting them with an edge ensures that IS only selects at most one of them.

   Running time: Creating the graph takes $O(g^2n)$ time as we have to iterate over every pair of groups (note that in our implementation we check each pair of groups twice. You can do a more clever iteration to avoid this redundancy. However this cuts the running time only in half, which doesn't have a lot of asymptotical benefit. Each group has in worst case $O(n)$ members which makes the set intersection operation $O(n)$. (*if you assume that intersection is constant and clearly state this, it will also be accepted.*)

   The second part of IndependentGroups is the standard greedy algorithm on a graph with g nodes. If the largest degree vertex (= group which intersects the most other groups) hs degree D, then the

graph has size $O(gD)$. We have to sort vertices by degree , this takes an initial $O(g \log g)$. Then we can handle the removal and degree update process in time $O(D)$ for each iteration. As a result the runtime of the greedy algorithm is $O(g \log g)$.

The total running time is then $O(g^2 n) + O(g \log n) = O(g^2 n)$.

2. Clearly state and prove the approximation ratio of your algorithm. (*Any approximation ratio correlated with one of the relevant algorithms we learned in class will get full credit.*)

   Solution. The approximation ratio of our algorithm is the same as the approximation ratio of the approximate independent set algorithm used, because the student groups and the vertices are one to one, and in particular, the allowed choices of student groups and the independent sets are one to one. That algorithm constructs an independent set within a factor of $(D+1)$ of optimal, so if S is a maximum independent set, then the approximation algorithm returns a set of vertices of size at least $\frac{|S|}{D+1}$. Those vertices are then mapped back to student groups to get a choice of groups that is again within a factor of $(D+1)$ of optimal.

   What is D in this student group problem? D for the approximate independent set algorithm is the maximum degree of vertices. Since edges in our graph represent student groups sharing members, then D is the number of groups overlapping the student group with the most overlapping groups.

Problem 2 *Catering (10 points)*

You are a caterer and you've been asked to pick meal choices. You have a list of entrees and drinks (fancy steak, expensive wine, etc) and a list of valid entree+drink pairings from your sommelier. You also have a list of guest preferences for entrees and drinks (list of acceptable choices), but the preferences are just for entrees and just for drinks, not for the pairs.

(For some reason you don't care about the price of items, nor potential waste of food or drinks...)

1. Design an approximation algorithm to pick a minimal set of pairs of entrees and drinks, such that each pair is valid, and each guest is happy to eat at least one of the pairs. (*Prove that your algorithm indeed yields a valid solution, but you can omit the running time analysis.*)

   Solution. We can reduce this problem to set cover as follows. Each item in the universe will correspond to a guest. Each valid entry+drink pair will correspond to a set that contains all the guests who would be satisfied by both the entree and the drink in the pair. The set of a valid entry+drink pair thus includes all the guests who would be happy to eat that pair, and a set cover corresponds to a set of valid entry+drink pairs where each guest is happy to eat at least one of the pairs. Choices of sets are one-to-one with choices of valid entry+drink pairs, and set covers are one-to-one with sets of valid entry+drink pairs where all the guests are happy. Therefore, a minimum set cover corresponds to minimizing the size of a set of valid entry+drink pairs where all the guests are happy.

   Since we were asked to design an approximation algorithm, we can apply the $\ln n$ approximate set cover algorithm from class to solve the minimum set cover problem, and then map the choice of sets in that algorithm back to a choice of valid entry+drink pairs.

The implementation of this algorithm will create one set for each pair. Then iterate over the guests' preference lists and add the guest to any set that they prefer. The we call the GreedySC algorithm from class on this input.

2. Clearly state and prove the approximation ratio of your algorithm.

   Solution. The approximation ratio of the algorithm presented in class is $\ln n$. What is n in this catering problem? n is the number of items in the universe which is the same as the number of guests with this reduction to set cover.

3. Now suppose that as a caterer you still have to deliver entrees+drinks in pairs. However, guests can mix-and-match; they may take an entree from one pair with a drink from another pair as long as both are on their preference lists. Design an approximation algorithm to cater this scenario with the fewest possible number of pairs selected. (*Prove that your algorithm indeed yields a valid solution, but you can omit the running time analysis.*)

   Solution. We can reduce this version of the problem to set cover as follows. The universe U to be covered includes two items per guest. The first item g_e for each guest g represents their entree choice, and the second item g_d of each guest represents their drink choice. Each set represents an entree+drink pair. An entree+drink pair's set contains the item representing a particular guest's entree choice if the entree in the pair is on the guest's entree preference list. Similarly, an entree+drink pair's set contains the item representing a particular guest's drink choice if the drink in the pair is on the guest's drink preference list. Note that with this, if a guest happens to like both the entree and drink in a pairing, then both g_e and g_d will be added to the set.

   With this graph, a choice of sets covers the universe if and only if each guest's entree and drink preferences can both be satisfied by one of the entree+drink pairs. Picking a set cover exactly corresponds to picking a sufficient set of entree+drink pairs to deliver that will all guests. And picking a minimum set cover minimizes the number of distinct entree+drink pairs to deliver. So, we can apply any approximate set cover algorithm, including the $\ln n$-approximate algorithm covered in lecture. The resulting set cover is then mapped back to the corresponding choice of entree+drink pairs on a 1-1 basis.

4. Clearly state and prove the approximation ratio of your second algorithm.

   Solution. The approximate set cover algorithm from class has an approximate ratio of $\ln n$. What is $\ln n$ from the set cover problem? The number of items to cover in the set cover problem is twice the number of guests in the catering problem. If G is the set of guests, then the approximation ratio is $\ln 2|G| = \ln |G| + \ln 2$.