CS 6150: HW3 – Greedy algorithms, Graphs basics
Collaborators: Trey Lavery, Shubham Mazumder

Submission date: Monday, Oct 25, 2021 (11:59 PM)

This assignment has 5 questions, for a total of 50 points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Question	Points	Score
Set Cover Analysis	14	
Be lazy or be greedy?	11	
Finding triangles	9	
Road tripping	10	
Santa's tradeoffs	6	
Total:	50	

Instructions. For all problems in which you are asked to develop an algorithm, write down the pseudocode, along with a rough argument for correctness and an analysis of the running time (unless specified otherwise). Failure to do this may result in a penalty. If you are unsure how much detail to provide, please contact the instructors on Piazza.

Question 1: Set Cover Analysis .... **[14]**

In class, we saw the set cover problem (phrased as picking the smallest set of people who cover a given set of skills). Formally, we have n people, and each person has a subset of m skills. Let the set of skill set of the ith person be denoted by S_i, which is a subset of $[m]$ (shorthand for $\{1, 2, \dots, m\}$). We analyzed a greedy algorithm that at every step, selects the person with the largest number of *uncovered* skills.

(a) [**2**] First, let us show that our analysis cannot be improved significantly. Consider the instance given by Figure 1 (with $n = 5$ people and $m = 26$), and write down (i) the optimal solution, and (ii) the solution output by the greedy algorithm.
Optimal : Person 1 and Person 2
Greedy : Person 5, Person 4, Person 3

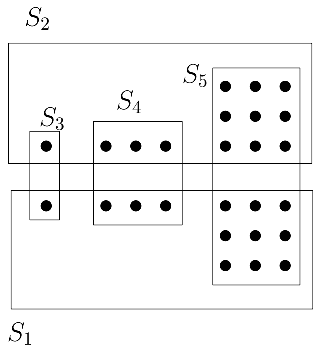

Figure 1: Every dot represents a skill, the rectangles show the skill sets of the people 1, 2, ..., 5.

(b) [**6**] Using Figure 1 as a hint, describe an instance in which the ratio of the size of the solution produced by the greedy algorithm and the size of the optimal solution is $\Omega(\log m)$. An example of an instance when the lower bound would be $\Omega(log(m)$ would be if the number of skills is a power of 2.

(c) [**6**] Next, let us see what happens if we stop the algorithm "mid way". Suppose we are given a parameter k, and we are told that there is a set of k people whose skill sets cover all of $[m]$. Now, suppose we run the greedy algorithm for exactly k iterations. So the algorithm chooses exactly k people, but it could be that some of the skills are still uncovered. Prove, however, that at least 50% of the skills are covered. [*Hint:* Modify the proof we saw in class.]

Question 2: Be lazy or be greedy? .... **[11]**

Consider the following "street surveillance" problem. We have a graph (V, E) with $|V| = n$ nodes and $|E| = m$ edges. We are allowed to place surveillance cameras at the nodes. Once placed, they can monitor all the edges incident to the node. The goal is to place as few cameras as possible, so as to monitor **all the edges** in the graph.

(a) [**4**] Show how to "cast" the street surveillance problem as an instance of Set Cover, and write down the greedy algorithm. (You don't need to provide any analysis.)

In this problem, the n represents the *people* and m represents *skill set* as defined in class. The goal then is to find a minimum subset of vertices v such that all the edges e are covered.

Algorithm

Algorithm 1 Surveillance

```
1: function SURVEILLANCE($G$):
Require: $GraphG$ where G represents a graph
2:     $V$ represents all vertices in the graph G
3:     $E$ represents all edges in the graph G
4:     $S$ is the set that contains all the vertices that store the result. Initialized to none in the
   beginning.
5:     while do$E \neq \emptyset$
6: Pick vertex $v \in V$ that has the highest degree. $S = S \cup v$. Remove edges from $e$ from $E$ that
   are connected to vertex $v$
7:     end while
8: return $S$
9: end function
```

(b) [**7**] Let (V, E) be a graph as above, and now consider the following "lazy" algorithm:

1. initialize $S = \emptyset$
2. while there is an unmonitored edge $\{i, j\}$:
   add both i, j to S and mark all their edges as monitored

Clearly (due to the while loop), the algorithm returns a set S that monitors all the edges. Prove that this set satisfies $|S| \leq 2k$, where k is the size of the optimal solution (i.e., the minimum number of nodes we need to place cameras at in order to monitor all the edges).

MORAL. Even though the algorithm looks "dumber" than the greedy algorithm, it has a better approximation guarantee — 2 versus $\log n$.

Hint. Consider the edges $\{i, j\}$ encountered when we run the algorithm. Could it be that the optimal set chooses *neither* of $\{i, j\}$?

The algorithm defines as an edge as two points $\{i, j\}$ where i and j are vertices. Let us assume that i and j are unique values. In step 2 of the algorithm, when we add vertices i and j to S we also remove all the other edges connected to these individual vertices. Let us say that we have a set SE that stores all the edges that are marked by the algorithm in step 2. Then based on the while loop condition, if every edge in SE is unique, then there are 2 unique vertices in set S that correspond to that edge. We can then say that the $|S| = 2 * |SE|$. Simplifying this, we get $|SE| = \frac{|S|}{2}$. If k represents the optimal solution then the edges in $|SE|$ for the optimal solution are less than k. Thus the statement $|S| \leq 2k$ holds.

Question 3: Finding triangles . . . . [**9**]

We will now see how for some graph problems, "non-standard" approaches can sometimes yield efficient algorithms. Let (V, E) be the set of vertices and edges in a directed graph, respectively.

Our goal is to find out if the graph has a *triangle*, i.e., if there exist $i, j, k \in V$ such that $(i, j), (j, k)$ and (k, i) all belong to E.

(a) [**3**] Show how to do this in time $O\left(\sum_{i \in V} d_{in}(i) d_{out}(i)\right)$, where d_{in} and d_{out} refer to the in-degree and out-degree of a vertex respectively.
Algorithm

Algorithm 2 Find the Triangle

```
1: N is the number of vertices in the graph
2: for $i = 0$ to $N - 1$ do
3:     (j, k) is an edge in the graph
4:     if $(i, j), (j, k), (k, i) \in E$ then return true;
5:     end if
6: end for
```

Correctness
There are two cases to prove :
1) If Find Triangle finds a triangle in the graph, then the result will be true.
2) If Find Triangle returns true, then there is a triangle in the graph.
Let us assume the first statement is true. Then to achieve the run time we have to loop through all vertices and loop through all the in-degree vertices and the out-degree vertices and check to see if there is an edge between an in-degree vertex and out-degree vertex. This takes $O(\sum_i n = i \in V d_{in}(i) * d_{out}(i))$. As we demonstrated in the algorithm above, the converse is also true.
Runtime $O(\sum_i n = i \in V d_{in}(i) * d_{out}(i))$ as we have shown in the algorithm and proof of correctness.

(b) [**6**] Of course, the above reduces to $O(n^3)$ if the graph is dense and d_{in} and d_{out} are $\Theta(n)$ for many vertices. However, assuming that two $n \times n$ matrices can be multiplied in time $O(n^{2.4})$ (which is known to be true), show how to solve the problem in time $O(n^{2.4})$. [*Hint:* Consider the adjacency matrix of the graph A. What can you say about the entries of $A \cdot A$? Can you use this matrix to then check the existence of a triangle?]
We are given in the problem that the runtime for matrix multiplication is $O(n^2.4)$. We first multiply the adjacency matrix representation of the Graph A from the problem with itself which takes $O(n^2.4)$. Next we loop through all the entries in the matrix which takes $O(v^2)$ time checking to see if every entry in the squared matrix A^2 have a non-zero entry in the same location. If it does, then we have found a triangle. The overall runtime of this approach is $O(n^2.4)$ because the matrix multiplication operation takes the longest time.

Question 4: Road tripping.. **[10]**

Let us roll back to the days when a road trip meant writing all your favorite music onto CDs and carrying them along.

Suppose we have n songs, of durations $d_1, d_2, \ldots, d_n$, where each d_i is less than one hour. Suppose that every CD can hold one hour of music. The goal is to write all the songs to a set of CDs, so as to minimize the number of CDs required.

Consider the natural greedy algorithm for the problem: at every step, we have a list of CDs containing the music written so far (initially empty). Now consider the songs in the given order, and for song i, write it to the <u>first CD so far</u> that has sufficient space for the song. If there is no such CD, then we buy a new CD, and write song i to it (and add it to the CD list; we are assuming that we can keep adding songs to a CD as long as there is space).

(a) [**3**] Give an example in which the greedy algorithm is not optimal. I.e., give a set of song durations for which the above procedure uses more CDs than an optimal "packing".
Example : $59, 40, 1, 20$. This is a non-optimal ordering because we use 1 CD for 59, 1 CD for $40, 1$, 1 CD for 20 which results in 3 CD's used. If the ordering were $59, 1, 40, 20$ then utilizing the greedy approach we use 2 CD's.

(b) [**7**] Suppose you are told that all the song durations are at most 20 minutes. Now prove that the greedy algorithm is within a factor 3/2 of the "best possible" number of CDs. In other words, prove that if OPT denotes the number of CDs in the best-possible solution, then the algorithm uses $\leq (1.5 \cdot \text{OPT})$ CDs. You will get full credit even if your bound is off by an additive constant, e.g., you prove $\leq (1.5 \cdot \text{OPT} + 2)$. [*Hint:* You may want to compare the number of CDs used by the algorithm with the "total duration" of the songs. Also, play around with small examples to see how the algorithm is doing.]
Say that the greedy algorithm in the problem uses c CDs. Then at least $c - 1$ CDs are more than $\frac{2}{3}$ full. If a CD has space that is < 40 minutes we can pack in more songs. Then say CD represents the optimal solution for how many CDs we need for the songs. Then $CD \geq \sum_{n=1}^{k} a_n > \frac{2(c-1)}{3}$ where the summation represents the sum of all the songs on a particular CD. Simplifying this statement we get the following: $\frac{3CD}{2} + 1 \geq c$.

Question 5: Santa's tradeoffs .... [**6**]

Recall the matching problem we saw in class: there are n gifts, and n children, and each child has a non-negative valuation for each gift. Formally, the value of gift j to child i is given by $H_{i,j}$. We assume that all $H_{i,j} \geq 0$. Santa's goal is to give one gift to each child, so as to maximize the *total value* (of course, a gift cannot be given to more than one child).

Suppose we now perform a more elaborate local search, this time picking every *triple* of edges in the current solution, and seeing if there is a reassignment of gifts between the end points of these edges that can improve the total value. Prove that a locally optimal solution produced this way has a value that is at least (2/3) the optimum value. [This kind of a trade-off is typical in local search – each iteration is now more expensive $O(n^3)$ instead of $O(n^2)$, but the approximation ratio is better.]

Note. We will see in the coming lectures that this problem can indeed be solved optimally in polynomial time, using a rather different algorithm.