CS 6150: HW3 – Greedy algorithms, Graphs basics

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 $\{1, 2, \ldots, m\}$). 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.

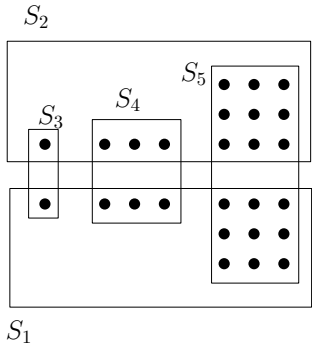

Figure 1: Every dot represents a skill, the rectangles show the skill sets of the people 1, 2, ..., 5.

Solution. The optimal solution is $S_{OPT} = \{S_1, S_2\}$. The greedy solution is $S_G = \{S_3, S_4, S_5\}$.

Rubric:

- 1 point for correct optimal solution
- 1 point for correct greedy solution.

(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 $\Omega(\log m)$.

Solution. Consider an instance similar to the one shown in Figure 1, but extended so that there are $3^0 + 3^1 + 3^2 + \cdots + 3^{k-1}$ dots in both S_1 and S_2, with S_{m+3} containing the $2(3^m)$ dots in S_1 and S_2, as in Figure 1. In this case, the optimal solution will be $S_{OPT} = \{S_1, S_2\}$, but the greedy algorithm will choose k sets, $S_G = \{S_3, S_4, \ldots, S_{k+2}\}$.

The ratio we are looking for is

$$\frac{|S_G|}{|S_{OPT}|} = \frac{k}{2}$$

Now we need to represent k as a function of m, the number of skills. In S_1 and S_2, we have

$$m = 2\left(3^0 + 3^1 + 3^2 + \cdots + 3^{k-1}\right)$$

From the closed formula for a geometric series, this gives

$$m = 2\frac{3^k - 1}{2} = 3^k - 1$$

Solving for m, we have $k = \frac{\log(m+1)}{\log(3)}$, so the ratio is

$$\frac{|S_G|}{|S_{OPT}|} = \frac{k}{2} = \frac{\log(m+1)}{6} \in \Omega(\log m)$$

Rubric:

- 3 points for an instance with ratio of $\Omega(\log m)$
- 3 points for correct justification of the ratio

(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.]

Solution.

To prove that the set of people chosen cover $\geq 50\%$ of the skills, we can simply prove that the number of skills not covered by people chosen is $< 50\%$ of the skills.

To show this we can use $u_t \leq u_0 \left(1 - \frac{1}{k}\right)^t$ (from the class discussion), where u_t is the number of skills uncovered after step t and $u_0 = m$, the number of skills we start with. Letting $t = k$, we get

$$u_k \leq m \left(1 - \frac{1}{k}\right)^k$$

Since for all $x \in \mathbb{R}$, $1 - x \leq e^{-x}$, we have

$$u_k \leq m(e^{-1/k})^k = me^{-1} < 0.37m$$

The number of uncovered skills is $\frac{u_k}{m} < 0.37$, which is less than 0.5. Therefore we have at most the number of skills not covered by people chosen is $< 50\%$ of the skills and thus the set of people chosen cover $\geq 50\%$ of the skills.

Rubric:

- 2 points for discussing the number of uncovered skills after k steps.
- 2 points for stating $u_k \leq m(1 - 1/k)^k$.
- 2 points for stating $u_k \leq me^{-1} < 0.5m$.

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.)

Solution.

We can easily cast this problem as a set cover problem by taking the n nodes to represent "people" and m edges to represent "skills". We say a vertex v covers an edge e if v is incident on e. Formally, $V = \{v_1, \ldots, v_n\}$ be the set of vertices. Then we define $S_{v_i} = \{e \in E | e = v_i y (y \in V)\}$. The set cover problem is to find $U \subseteq V$ such that $\cup_{u \in U} S_u = E$ and $|U|$ is minimized.

Algorithm 1 Street Surveillance Problem

```
1: function streetSurveillanceProblem(G)                ▷ Input graph G = (V, E)
2:     V,E = G
3:     cover = []
4:     monitored = []
5:     while cover is not equal to E do
6:         v = fetch vertex with most unseen edges
7:         cover += v.edges
8:         monitored += v
9:         delete v from unmonitored        ▷ unmonitored = set of unmonitored vertices
10:    return monitored
```

Rubric:

- 2 points for representing problem, as an instance of Set Cover.
- 2 points for stating the Algorithm.

(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 $\{i, j\}$?

Solution.

We can see that since the lazy algorithm picks both endpoints of an edge it is essentially picking an edge. Let F be the set of edges picked by the lazy algorithm. We can see that for any edges $e, f \in F$ the endpoints of e and f are distinct. Without loss of generality we may assume the algorithm picks e before f, and since we mark all the edges incident to endpoints of e as monitored so the unmonitored edges picked after that could not be incident to endpoints of e and thus f is vertex disjoint from e. Thus we have a set of vertex disjoint edges. Since the vertex cover has to cover these edges, then the vertex cover should have at least one endpoint

from each edge in F and thus $|F| \leq k$. We know that $|S| = 2|F|$ since for each edge we have two distinct vertices as we have argued above. Therefore, $|S| = 2|F| \leq 2k$ as desired.

Rubric:

- 4 points for discussing the optimal solution.
- 3 points to provide proof for $|S| \leq 2k$

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.

Solution. Consider the following algorithm.

Algorithm 2 Finding triangles

Input Directed graph $G = (V, E)$
Output True if G has a triangle; False otherwise

```
1: function FindTriangles1(G)
2:     for v ∈ V do
3:         for u ∈ N⁻(v) do                ▷ N⁻(v) denotes the in-neighbors of v.
4:             for w ∈ N⁺(v) do            ▷ N⁺(v) denotes the out-neighbors of v.
5:                 if (w, u) ∈ E then
6:                     return True
7:     return False
```

Proof. **Correctness.** When Algorithm 2 returns True in line 6, it is guaranteed that $\{(u, v), (v, w), (w, u)\} \subseteq E$ and thus (u, v, w) forms a triangle. Conversely, if G has a triangle (i, j, k), then Algorithm 2 must return True when $v = j, u = i, w = k$ in line 6.

Running time. It is clear to see the inner-most loop (lines 5-6) iterates $\sum_{i \in V} d_{in}(i) d_{out}(i)$ times. And we can check for the existence of an edge in constant time by using an adjacency matrix. The total running time is $O\left(\sum_{i \in V} d_{in}(i) d_{out}(i)\right)$. □

Rubric:

- 2 points for algorithm description.
- 1 point for running time discussion.

(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?]

Solution. Given a simple directed graph[1] $G = (V, E)$ with $V = \{1, \ldots, n\}$, let $A \in$

[1] Graphs with no self-loops or multiple edges. They may have symmetric edges (e.g. (v, u) and (u, v)).

$\{0,1\}^{n\times n}$ be the adjacency matrix of G, that is, $(A)_{ij} = 1$ if and only if $(i,j) \in E$ for every i, j. Now, consider the following algorithm.

Algorithm 3 Finding triangles with matrix operations

Input Directed graph $G = (V, E)$
Output True if G has a triangle; False otherwise

```
1: function FindTriangles2(G)
2:     let A ← adjacency matrix of G
3:     let B ← A · A                    ▷ matrix multiplication (equivalently, A^2)
4:     let C ← B ∘ A^T                  ▷ "∘" denotes element-wise multiplication
5:     if C ≠ 0_{n,n} then
6:         return True
7:     return False
```

Here $(A \circ B)_{ij} := (A)_{ij} \cdot (B)_{ij}$ denotes element-wise multiplication, $(A^T)_{ij} := (A)_{ji}$ denotes transposition and $0_{n,n}$ is the zero matrix in $\mathbb{Z}^{n\times n}$, whose elements are all zeros.

Proof. **Correctness.** First, observe that each element of matrix B corresponds to the number of length-2 trails[2] in G. By definition, $(B)_{ij} = \sum_{k=1}^{n} (A)_{ik} \cdot (A)_{kj}$, and then $(B)_{ij} \geq 1$ if and only if there exists k such that $(A)_{ik} = (A)_{kj} = 1$. Therefore (i,k)-(k,j) is a trail in G.

Next, assume that Algorithm 3 returns True. Then, there must exist i, j such that $(C)_{ij} \geq 1$. From $(C)_{ij} = (B)_{ij} \cdot (A^T)_{ij} = (B)_{ij} \cdot (A)_{ji} \geq 1$, we get $(A)_{ji} = 1$, equivalently, $(j,i) \in E$ such that $i \neq j$ because G is simple. Also knowing $(B)_{ij} \geq 1$, we conclude that there exists k such that (i,k,j) forms a circuit[3] in G. Again, since G is simple, $i \neq k$ and $k \neq j$, which determines that (i,k,j) is in fact a triangle.

Conversely, if G admits a triangle (u,v,w), then $(A)_{uv} = (A)_{vw} = (A)_{wu} = 1$. Then, we get $(B)_{uw} = \sum_{k=1}^{n} (A)_{uk} \cdot (A)_{kw} \geq (A)_{uv} \cdot (A)_{vw} = 1$ and $(C)_{uw} = (B)_{uw} \cdot (A^T)_{uw} = (B)_{uw} \cdot (A)_{wu} \geq 1$. C is not a zero matrix, and Algorithm 3 should return True in line 6[4].

Running time. By assumption, the matrix multiplication in line 3 takes $O(n^{2.4})$ time. Other matrix operations (element-wise multiplication, transposition, etc.) can be done in $O(n^2)$ time. Hence, the total running time is $O(n^{2.4})$, as desired. □

Rubric:

- 2 points for arguing the property of A^2.
- 1 point for algorithm description.
- 2 points for correctness proof.
- 1 point for running time discussion.

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.

[2] Not necessarily a path because these may include (v,u)-(u,v).
[3] A closed trail.
[4] One may also design an algorithm using A^3. (e.g. Return $\mathrm{tr}(A^3) > 0$.)

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 first CD so far 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".

Solution. Example = [d1 = 20, d2 = 35, d3 = 25, d4 = 40]
Greedy Solution:
CD1: [20, 35]
CD2: [25]
CD3: [40]
Optimal Solution:
CD1: [20, 40]
CD2: [35, 25]

Rubric:

- 3 points for correct example

(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.]

Solution.

d_i: length of a song in mins; CD_i: space occupied by songs in mins
Let's say we have filled the first k-1 songs in m-1 CDs.
When will we consider putting the song d_k in a new CD_m?

From our greedy algorithm, when all the m-1 CDs have been filled more than $60 - d_k$ mins, i.e d_k doesn't fit in any of the m-1 CDs.
$\therefore 60 - d_k < CD_i \ \forall i \in [1, 2, ..m-1]$
We know that $d_i \leq 20$, for any i
$\therefore 40 \leq 60 - d_k$
$\therefore 40 \leq 60 - d_k < CD_i \ \forall i \in [1, 2, ..m-1]$
Which implies that all the m-1 CDs occupy more than 40 mins.
Let's say that by using the greedy algorithm, we require G CDs to fill all the n songs.
Then this implies that all the **1st *G*-1 CDs occupy more than 40 mins**.

Let OPT be the optimal number of CDs that will be able to contain all the songs, then $\sum_{i=1}^{n} d_i \leq 60 \cdot \text{OPT}$.

$$\sum_{i=1}^{G} CD_i = \sum_{i=1}^{n} d_i$$

$$\sum_{i=1}^{G-1} CD_i + CD_G = \sum_{i=1}^{n} d_i$$

$$40 * (G-1) + CD_G \leq \sum_{i=1}^{n} d_i$$

$$40 * (G-1) + CD_G \leq 60 \cdot \textsc{Opt}$$

$$G \leq 1.5 \cdot \textsc{Opt} + 1 - \frac{CD_G}{40}$$

$$G \leq 1.5 \cdot \textsc{Opt} + 1 - \epsilon \qquad \textit{where } 0 < \epsilon \leq 1.5$$

$$G \leq 1.5 \cdot \textsc{Opt} + 2$$

Note: Research "bin packing problem" to find more information.

Rubric:

- 2 points for recognising CDs will occupy at least 40/60 mins.
- 2 points for correctly reasoning the Optimal Solution
- 3 points for getting the final ans with all the steps

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.

Solution. Similar to the proof in class, suppose that the optimal solution assigns child i the gift $\pi(i)$. Thus, the total happiness of our optimal assignment is $\sum_i H_{i,\pi(i)}$. Consider the new local search algorithm, and let σ be a locally optimal solution. I.e., in this solution, $\sigma(i)$ is the gift assigned to child i.

Due to local optimality, we know that for any i, j, k, the happiness of these current assignments, $H_{i,\sigma(i)} + H_{j,\sigma(j)} + H_{k,\sigma(k)}$, is greater than the happiness if any of their gifts were to be swapped.

Now consider changing the locally opt solution as follows: give gift $\pi(i)$ to child i (this means the child $\sigma^{-1}(\pi(i))$ —call this j— loses his/her gift), give the gift $\pi(j)$ to j (which leaves the child $\sigma^{-1}(\pi(j))$ —call this k— without a gift), give the gift $\sigma(i)$ to child k.

From local optimality along with the non-negativity of the H values, we have

$$H_{i,\sigma(i)} + H_{j,\sigma(j)} + H_{k,\sigma(k)} \geq H_{i,\pi(i)} + H_{j,\pi(j)} + H_{k,\sigma(i)} \geq H_{i,\pi(i)} + H_{j,\pi(j)}.$$

Note that j and k are determined by i, and further, as i ranges over $1, 2, \ldots, n$, so do j and k (in different orders[5]). Thus, if we sum the LHS and RHS over the index i, we get

$$\begin{aligned}
\sum_i H_{i,\sigma(i)} + \sum_i H_{j,\sigma(j)} + \sum_i H_{k,\sigma(k)} &\geq \sum_i H_{i,\pi(i)} + \sum_i H_{j,\pi(j)} \\
3\sum_i H_{i,\sigma(i)} &\geq 2\sum_i H_{i,\pi(i)} \\
\sum_i H_{i,\sigma(i)} &\geq \frac{2}{3}\sum_i H_{i,\pi(i)}
\end{aligned}$$

So our triple-local search gives a 2/3 approximation.

Rubric:

- 2 points for recognizing local optimality
- 2 points for recognizing that j and k range with i
- 2 points for a cogent explanation of the 2/3 approximation

[5]This is the crucial part — make sure you understand why this is true!