HW 3 – Greedy Algorithms, Local Search

CS 6150

Submission date: Friday, Oct 6, 2023, 11:59 PM

This assignment has **??** questions for a total of **??** points, plus 4 bonus points (earning bonus points will help your grade for points lost here or in other assignments). Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Run LaTeX again to produce the table

Question 1: Set Cover Revisited .... [??]

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 skills of the ith person be denoted by S_i, which is a subset of $[m]$ (shorthand for $\{1, 2, \ldots, m\}$).

(a) [**6**] As before, suppose that there is a set of k people whose skill sets cover all of $[m]$ (i.e., together, they possess all the skills). Now, suppose we run the greedy algorithm discussed in class for $4k$ iterations. Prove that the set of people chosen cover $> 90\%$ of the skills.

Ans)
No. of iterations t=4k
$t = m(1/e)^{\alpha}$
m= number of skills, e=2.71, alpha=4
Therefore t=m(1/2.71)^4
$t = m(0.36)^4$
$t = 0.0167m$
This is t represents people not taken.
So the number people taken would be m-0.0167m
$t = 0.9833m$
This is 98.33%
Which is greater than 90%
Therefore set of people chosen would be greater than 90% of the skills

(b) [**4**] Consider the following "street surveillance" problem. We have a graph (V, E) with n nodes and 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. Show how to cast the street surveillance problem as Set Cover.

Ans)
Street surveillance problem is same as set cover problem
We have n vertices or nodes and m edges.
In the set cover problem say the people and skill example, we have n people and m skills.
We have to choose k optimal people such that all m skills are covered.
In the surveillance problem too we have to choose n nodes or cameras such as all the edges are covered.
In both situation we are choosing some n that can contain all the m.

(c) [**6**] Let (V, E) be a graph as above, and suppose that the optimal solution for the street surveillance problem places k cameras (and is able to monitor all edges). 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 the set also satisfies $|S| \leq 2k$ (recall that k is the number of nodes in the optimal solution).

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 $\{i, j\}$?

Ans)
In the optimal solution, every edge is covered by at least one camera. Say optimal solution doesn't chooses edge i,j i.e. nodes i and j in the solution. That means that there is at least some node that covers the edge represented by edge i,j.
In the lazy algorithm we see that edge i,j are not monitored.
The lazy algorithm will then include edge i,j even if the edge represented by it is covered by some other nodes.
If the optimal solution doesn't choose the nodes i,j this means it must contain come other nodes that represent that edge.

Hence the lazy algorithm adds i and j to S even though its not needed.
Therefore every node gets added almost twice in the lazy algorithm.
In summary, the "lazy" algorithm guarantees that $|S| \leq 2k$, and this is due to the fact that if the optimal solution does not choose nodes i or j for monitoring an edge, the "lazy" algorithm effectively covers those edges by adding i and j to S.

Question 2: Selling Intervals .... **[??]**

Suppose we are given n intervals on the real line — $[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$. Of course, some of the intervals overlap. For each interval, we have a buyer, who is willing to pay a price of p_i for interval i. The goal is to find the best subset of intervals to sell, such that (a) intervals sold are all non-overlapping, and (b) the total price is maximized.

(a) [**4**] It is natural to want to sell intervals that are short but have a high price. Consider a greedy algorithm that first sorts the intervals in decreasing order based of the ratio of price to length, i.e., $p_i/(b_i - a_i)$, and then does the following:

```
1. initialize $S = \emptyset$
2. go over the intervals in sorted order (as above), doing the foll:
        add the interval to $S$ if it does not overlap with any interval already in $S$
```

Give an instance in which the greedy choice is sub-optimal.

Ans)
Let the intervals be
1st $[1, 4]$ with price 10
2nd $[3, 6]$ with price 8
3rd $[5, 8]$ with price 7
Let $p_i/(b_i - b_a)$ be
1st $10/3 = 3.34$
2nd $8/3 = 2.67$
3rd $7/3 = 2.33$
The decreasing order would be
1st then 2nd then 3rd
For greedy 1st would be selected as the first interval. Then 3rd is selected and 2nd is rejected since it is overlapping.
Therefore the sum would be 17.
But for the optimal solution we would select 1st and 2nd even though they are overlapping, their sum is 18 which is greater than 17.
Therefore greedy is sub-optimal.

(b) [**8**] Design an algorithm based on dynamic programming. Your algorithm should run in time polynomial in n. As always, include an analysis of correctness and the running time.

Ans)
We will sort in non-decreasing order for $n \log(n)$
Theory
We will start with a_1
There are two choices either we choose a_1 or not
If we choose a_1 then we will go to next $a_i > b_1$(left)
If we don't choose a_1 then we a_2(right)
Steps are gonna repeat for both parts
Say for left if we choose yes it will be next $a_i > b_2$ then again yes and no
Say for the right if we choose no it will be a_2
therefore the dp would be $f(a_1) = max(f(a_1) + f(a_i > b_1), f(a_2))$
For some reason algorithm is below bonus question.
Since there are 2 for loops, the $O(n^2)$
Sorting time would be $n + \log(n)$

Algorithm 1 DP

```
Let the array be sorted in non-decreasing order.
Initialize DP of size n to store all the intervals.
Initialize DP[0] with $a_1$
for $i = 1$ to n-1 do
  Set DP[i] to the price of interval i
  for $j = i - 1$ to 0 do
    if $a_i > b_1$ then
      DP[0]=max(f(a$_1$) $+ f(a_i > b_1), f(a_2)) -$
    Return DP[0] =0
```

Proof of correctness

This approach is correct as it considers all the combinations of non-overlapping solutions and selects that interval that has the maximum price. Intervals are sorted by the end points. This guarantees that we select non-overlapping and interval with maximum price.

(c) [] **[4] Bonus points**: You are given the same set of intervals with associated prices, and you still want to maximize the total price. However, you are allowed to sell overlapping intervals, but for each overlapping interval, you incur a penalty cost of "penalty" dollars. Your goal is to find the subset of intervals to maximize the total price while minimizing the penalty cost.

Provide a dynamic programming solution for this extended problem. Analyze the time complexity of your solution.

Question 3: Forming pairs . . . . [**??**]

Consider n people, each having a *skill level* ($s_1, s_2, \ldots, s_n$, which are integers). We are also given a threshold T. Two people form a *successful pair* if the sum of their skill levels is $\geq T$.

The goal is to form the maximum number of successful pairs. For convenience, suppose that s_i are already sorted in increasing order. Now, consider the following greedy algorithm: given a set of people, consider pairing the least skilled person and the most skilled person. If the sum of the skill levels is $\geq T$, form the pair and recurse; else, delete the least skilled person and recurse on the rest.

(a) [**3**] Show how to implement this algorithm in $O(n)$ time (by writing pseudo-code and performing a runtime analysis).

Ans)
Two pointer method can be used to get $O(n)$

Algorithm 2 Two pointer technique

```
Require: An array skills containing the skill levels of n people, already sorted in increasing order, and an
  integer T representing the threshold for a successful pair.
  Initialize variables:
  $left \leftarrow 0$
  $right \leftarrow n - 1$
  $pairs \leftarrow 0$
  while $left < right$ do
    Calculate the sum of skill levels at indices $left$ and $right$.
    if The sum is $\geq T$ then
      Increment $pairs$ by 1
      Increment $left$ by 1
      Decrement $right$ by 1
    else
      Increment only $left$ by 1.
    end if
  end while
  return The value of $pairs$ as the maximum number of successful pairs. =0
```

Since the algorithm is iterating only once through the array, therefore the run-time complexity is O(n).

(b) [**7**] Does the algorithm always output the pairing with the largest possible number of (successful) pairs? Either provide a counter-example, or give a formal proof of correctness.

Ans)
Claim: The greedy algorithm correctly identifies the maximum number of successful pairs.
Proof by induction
Base case(n=1) there is only one person in the list.
We assume the algorithm performs correctly for k number of pairs.
We need to show that this algorithm performs correctly for the k+1 steps.
We have two cases:
a) If the total value is less than T then we check the next least skilled member, this means we discard the least skilled member, so the total number of members would be k.
b) If total value is greater than equal to T then we
form a successful pair and the number would be k.
By inductive hypothesis we say that algorithm performs correctly for k people.
Therefore it performs correctly for k+1 people too.

Question 4: Finding diverse elements .... [??]

A common problem in returning search results is to display results that are *diverse*. A simplified formulation of the problem is as follows. We have n points in Euclidean space of d-dimensions, and suppose that by distance, we mean the standard Euclidean distance. The goal is to pick a subset of k (out of the n) points, so as to maximize the sum of the pairwise distances between the chosen points. I.e., if the points are denoted $P = \{p_1, p_2, \dots, p_n\}$, then we wish to choose an $S \subseteq P$, such that $|S| = k$, and $\sum_{p_i, p_j \in S} d(p_i, p_j)$ is maximized.

A common heuristic for this problem is local search. Start with some subset of the points, call them $S = \{q_1, q_2, \dots, q_k\} \subseteq P$. At each step, we check if replacing $\underline{\text{one of the } q_i}$ with a point in $P \setminus S$ improves the objective value. If so, we perform the swap, and continue doing so as long as the objective improves. The procedure stops when no improvement (of this form) is possible. Suppose the algorithm ends with $S = \{q_1, \dots, q_k\}$. We wish to compare the objective value of this solution with the optimum one. Let $\{x_1, x_2, \dots, x_k\}$ be the optimum subset.

(a) [**3**] Use local optimality to argue that:

$$d(x_1, q_2) + d(x_1, q_3) + \cdots + d(x_1, q_k) \le d(q_1, q_2) + d(q_1, q_3) + \cdots + d(q_1, q_k).$$

Ans)
$d(q_1, q_2) + d(q_1, q_3) + \cdots + d(q_1, q_k)$ is local optimal as given in the question.
Therefore it means if we swap elements the result should not improve.
$d(x_1, q_2) + d(x_1, q_3) + \cdots + d(x_1, q_k)$ in this solution we see that q_1 is replaced by x_1.
This value should not be more than $d(q_1, q_2) + d(q_1, q_3) + \cdots + d(q_1, q_k)$ because that would mean it is a better solution.
Which can't be true as $d(q_1, q_2) + d(q_1, q_3) + \cdots + d(q_1, q_k)$ is local optimal solution.
Therefore

$$d(x_1, q_2) + d(x_1, q_3) + \cdots + d(x_1, q_k) \le d(q_1, q_2) + d(q_1, q_3) + \cdots + d(q_1, q_k).$$

(b) [**4**] Deduce that: [*Hint:* Use two inequalities of the form above.]

$$(k-1) \cdot d(x_1, x_2) \le 2\left[d(q_1, q_2) + d(q_1, q_3) + \cdots + d(q_1, q_k)\right].$$

Ans)
We know that

$$d(x_1, q_2) + d(x_1, q_3) + \cdots + d(x_1, q_k) \le d(q_1, q_2) + d(q_1, q_3) + \cdots + d(q_1, q_k).$$

Now similarly

$$d(x_2, q_2) + d(x_2, q_3) + \cdots + d(x_2, q_k) \le d(q_1, q_2) + d(q_1, q_3) + \cdots + d(q_1, q_k).$$

When we add these two, we get

$$d(x_1, q_2)+d(x_1, q_3)+\cdots+d(x_1, q_k)+d(x_2, q_2)+d(x_2, q_3)+\cdots+d(x_2, q_k) \le 2[d(q_1, q_2)+d(q_1, q_3)+\cdots+d(q_1, q_k).]$$

Using triangle inequality we can say that

$$d(x_1, q_2) + d(x_1, q_3) + \cdots + d(x_1, q_k) + d(x_2, q_2) + d(x_2, q_3) + \cdots + d(x_2, q_k) \ge (k-1)d(x_1, x_2)$$

There k-1 terms as it starts from q_2 and lasts till q_k

We can see replace $d(x_1, q_2)$ and $d(x_2, q_2)$ by $d(x_1, x_2)$ because of triangle inequality.
Therefore we can say that

$$(k-1) \cdot d(x_1, x_2) \leq 2\left[d(q_1, q_2) + d(q_1, q_3) + \cdots + d(q_1, q_k)\right].$$

(c) **[5]** Use this expression to argue that

$$\sum_{i,j} d(x_i, x_j) \leq 2 \sum_{i,j} d(q_i, q_j).$$

Note that this shows that the locally optimum solution has objective value at least 1/2 the optimum.
Ans)
Adding all the variables we get.

$$(k-1) \cdot d(x_1, x_2) \leq 2\left[d(q_1, q_2) + d(q_1, q_3) + \cdots + d(q_1, q_k)\right].$$

We can convert this in the sigma form and we will get.

$$\sum_{i,j} d(x_i, x_j) \leq 2 \sum_{i,j} d(q_i, q_j).$$

This is the same expression as the b) part just with sigma.