Greedy Quals Problems

Sijian Tan

1 Alternative Interval Scheduling (HW1)

1.1 Problem Definition

For the "interval scheduling" problem introduced in the lectures, consider the following algorithm:

- Sort the jobs by ascending order of start time, and obtain jobs $J_1, J_2, \ldots, J_n$, where $s_1 \leq s_2 \leq \cdots \leq s_n$. ($s_i$ is the start time of job J_i).
- Starting with $i = n$ and at every step decreasing i by 1, add the job to the set A (A denotes the set of selected jobs) if job J_i is compatible with all jobs in A.

Prove that this algorithm is optimal.

1.2 Proof

1.2.1 Define Solutions

- Let $G_1, G_2, ..., G_k$ be the starting times of the jobs selected based on the greedy algorithm order, start from the end.
- Let $O_1, O_2, ..., O_m$ start times of the jobs in an optimal solution.

We want to show that $k \geq m$, the number of jobs selected in the greedy solution is at least the number of jobs in the optimal solution.

1.2.2 Greedy Stays Ahead

We will show that each step i, the greedy solution has included at least as many jobs as the optimal solution. In other words, we need to prove that $G_i \geq O_i$ (notice that here it is the starting time) for all i so that to prove the greedy solution could at least many jobs as the optimal solution.

1.2.3 Inductive Argument

For the base case:

- For the first job to be selected, because the greedy solution select the latest starting time (start from the end), so $G_1 \geq O_1$

For the inductive step:

- Assume that for some i, the greedy algorithm has selected the jobs such that $G_j \geq O_j$ for all $j \leq i$.
- Now consider the $(i+1)^{th}$ selection. The greedy algorithm will select the G_{i+1} as the compatible job with latest starting time after G_i, and optimal solution will select the job with some other time.
- Then by the inductive hypothesis, $G_i \geq O_i$, so $G_{i+1} \geq O_{i+1}$

1.2.4 Conclusion

- Since $G_i \geq O_i$ for all i, the greedy algorithm stays ahead of the optimal solution at each step.
- This implies that the greedy algorithm could include the jobs at least as the optimal solution (time processes slower, so could include more compatible jobs)
- Therefore, the total number of jobs selected by the greedy algorithm could not be less than the number used by the optimal solution ($k \geq m$)
- Therefore, greedy is optimal.

2 Weighted Item Addition (HW1)

2.1 Problem Definition

You are given n items and each item i has a value v_i ($v_i \geq 0$) and weight w_i. Given a total weight limit S, you are asked to select a subset of items whose total weight does not exceed S, and whose total value is maximized. Note: you can take a fraction of any item. For example, if you take a fraction k of item i, you gain value $k \cdot v_i$ and weight $k \cdot w_i$ is added.

1. Please design a greedy algorithm that yields the optimal solution.
2. Prove the algorithm yields the optional solution using the exchange argument.

2.2 Algorithm

In this question, the limit condition is weight, and we want to maximize the total value. Therefore, the algorithm is to sort the value per unit weight, which could be expressed as the ratio:

$$r_i = \frac{v_i}{w_i} \tag{1}$$

The larger the ratio is, the more efficient it will be for adding the total value. Therefore, we want to pick the items based on order of ratios using greedy algorithm, then pick a fraction of the last item to reach the weight requirement. The pseudo code is shown below:

The steps of exchange argument proof include:

Algorithm 1 Greedy algorithm

```
Sort items by the value/weight ratio so that r_1 > r_2 > ... > r_n
A ← ∅                                   ▷ Initialize the solution set.
w_t = S                                 ▷ Initialize the total weight for reduction.
for j = 1 to n do
    if w_t > 0 then                     ▷ Not reaching the limit yet
        if w_t > w_j then               ▷ Can still take a full item
            w_t = w_t − w_j
            k_j = 1
            A ← A ∪ (j, k_j)
        else                            ▷ Need to take a fraction of item
            k_j = w_t / w_j
            w_t = 0
            A ← A ∪ (j, k_j)
        end if
    end if
end for
return A
```

1. Assume there is an optimal solution O that is different from greedy solution G.

2. Because G is a greedy solution, we include the higher value-to-weight ratio first, which is the essence of a greedy approach: **it makes the locally optimal choice at each step with the hope of finding a global optimum.**

3. Now we assume j be the first item (in sorted order) that differs in O and G, because we assume O is better than G under the same S limit, so we assume **j is included more in O than in G.** This could be done in two ways: either all the other items are the same in fractions in two solutions, or there are some changes in fractions of items in two solutions.

4. For the first way, if solution O includes more of item j than G, it would be to exceed the weight limit S, because the fraction of j in G is already the available maximum when we keep fill the higher ratio items before j item.

5. Therefore, this is not possible without exceeding the weight limit. Now we assume there must be an item k with a **larger** (if lower, than it will not be efficient, not optimal solution anymore) in O, with **less** fraction included in O than in G. However, it means O does not take full advantage of higher value-to-weight ratio item, which will be less efficient, the solution will also be not optimal.

6. Therefore, because of these contradictions, there could not be any items difference between O and G, which means that G is truly the optimal solution.

3 Supercomputer (Midterm)

Consider the following job scheduling problem:

You have n jobs $J_1, J_2, \ldots, J_n$, one supercomputer and n identical PCs. Each job needs to be first *preprocessed* on the supercomputer, and then it needs to be *finished* on one of the PCs. Job J_i needs p_i seconds of time on the supercomputer, followed by f_i seconds of time on a PC.

Since there are n PCs, the finishing of the jobs can be performed fully in parallel, where each PC can process one job. However, the supercomputer can only work on a single job at a time, so you need to work out an order in which to schedule the jobs to the supercomputer. As soon as a job is done on the supercomputer, it can be handed off to a PC for finishing, and the supercomputer can start to process the next job.

The *completion time* of all jobs is the earliest time at which all jobs will have finished processing on the PCs. Describe a greedy algorithm which produces a schedule that minimizes the completion time, and prove the correctness of your algorithm.

3.1 Algorithm

Arrange the jobs in descending order of f_i, then process all the jobs in this order.

3.2 Proof

We prove this by an exchange argument. Consider any optimal solution, and suppose it does not follow the order in greedy algorithm. Then the optimal solution must contain a pair of i and j such that j is processed directly after i, such that $f_i < f_j$. We call this pair (i, j) as an inversion. Assume the supercomputer has worked T_{before} time before job i. Therefore, before the swap, the completion times after these two jobs are:

$$T_i = T_{before} + p_i + f_i \tag{2}$$

$$T_j = T_{before} + p_i + p_j + f_j \tag{3}$$

So the completion time in this case is T_j. Then, if we swap these two jobs, the completion time after these two jobs are:

$$T'_j = T_{before} + p_j + f_j \tag{4}$$

$$T'_i = T_{before} + p_j + p_i + f_i \tag{5}$$

Even though it is hard to compare T'_j and T'_i, but it is easily to see that:

$$T'_j < T_j, \;\; T'_i < T_j \tag{6}$$

Therefore, this swap of inversion will actually shorten the final complete time, which will not worsen the optimal solution.

Continuing in this way, we can eliminate all inversions without worsening the optimal solution. At the end of this process, the optimal solution will have the same order as greedy algorithm. Thus the greedy algorithm is optimal.

4 Homework Assignment (Final)

At the start of the semester you are given n homework assignments $\{a_1, \ldots, a_n\}$. You can do the assignments in any order, but you must turn in 1 assignment per week over the n weeks of the semester. Assignment a_i has a difficulty d_i (assume the difficulty values are distinct). If you turn in assignment a_i on week j, you get $(n - j) \cdot d_i$ points.

Please give a greedy algorithm which finds the order of the assignments that maximizes your points, and use the exchange argument to prove the optimality of your algorithm.

4.1 Algorithm

Do the homework in descending order of difficulties.

4.2 Proof

We can prove this algorithm using exchange argument. Suppose there is an optimal solution which does not follow the order in greedy algorithm. Then, there must exist a pair of i and j such that assignment a_j is processed right after a_i ($j = i + 1$) and $d_j > d_i$. We call this pair of (i, j) as an inversion. Therefore, the points we could get from these two assignments are:

$$P = (n - i)d_i + (n - j)d_j \tag{7}$$

Now, if we swap i and j, then the new points we could get will be:

$$P' = (n - i)d_j + (n - j)d_i \tag{8}$$

Therefore we could easily see that $P' > P$, therefore this swap of inversion will not worsen the optimal solution.

Continuing in this way, we can eliminate all inversions without worsening the optimal solution. At the end of this process, the optimal solution will have the same order as the greedy algorithm. Thus the greedy algorithm is optimal.

5 City Road (Spring 2024)

There are n cities in a state. Given any two cities x and y, you know the cost of building a road between x and y. You are required to build a network of roads such that any city can be reached from any other city (through a direct road or via intermediate cities). You need to do this at minimum cost.

Describe a greedy algorithm to solve the task and prove its correctness. What is the running time of your algorithm?

5.1 Proof

Prove Kruskal's Algorithm.

6 Water Supply (Fall 2023)

There are n houses in a village and one large water tank nearby. You are required to supply water to all houses by laying pipes. For each house you can either (i) build a pipe to the water tank or (ii) build a pipe to a house that already has water. You are given array T, where $T[i]$ is the cost of building a pipe between the water tank and house i. You are also given list H, where $H[i]$ is a tuple (j, c) indicating the cost c of building a pipe between houses i and j. The cost c is non-negative.

Your task is to supply water to all the houses at minimum cost. Describe a greedy algorithm to solve the task and prove the correctness of your algorithm. Your proof of correctness can use an exchange argument.

Hint: Model the problem as a graph and devise an algorithm for connecting the vertices.

6.1 Algorithm

Combine $T[i]$ and $H[i]$ as the edge set of the graph, and sort them together in ascending order. Then use Kruskal's algorithm to generate the MST.

6.2 Proof

Use greedy to prove Kruskal's algorithm.

7 Gas Filling (Spring 2023)

Assume you will drive from city A to city B along a highway and you wish to minimize the time you spend on adding gas to your tank. The following quantities are given to you:

- the capacity C of your gas tank in liters
- the rate F of fuel consumption in liters/mile
- the rate r in liters/minute at which you can fill your tank at a gas station. r is the same for all gas stations.
- the distances of the n gas stations from your start point $x_1, x_2, \ldots, x_n$ along the highway. We know $x_1 = 0, x_1 < x_2 < \ldots < x_n$ and $x_i - x_{i-1} \leq C/F$ for $2 \leq i \leq n$.

You will start with an empty tank. For example, if you stop to fill your tank from 2 liters to 8 liters, you would have to stop for $6/r$ minutes. The goal is to minimize the total time you stop for gas.

Consider the following two algorithms:

(a) Stop at every gas station, and fill the tank with just enough gas to make it to the next gas station.

(b) Stop if and only if you don't have enough gas to make it to the next gas station, and if you stop, fill the tank up all the way.

For each algorithm either prove or disprove that this algorithm solves the problem optimally. Your proof of correctness can use an exchange argument.

7.1 Proof for a

We can prove this using an exchange argument. Suppose we have a greedy solution $G_1, G_2, ..., G_n$, where G_i is the filling time at station i, and the optimal solution $O_1, O_2, ..., O_n$. Assume that the optimal solution is different from the greedy solution, that means at some i, $G_i \neq O_i$. Now we change O_i with G_i, because based on the greedy solution, G_i will be the minimum time to the fill the gas so that the car could reach the next station, so $G_i \leq O_i$, therefore this exchange will not increase the total amount of time to fill the gas. Continue this way, finally the optimal solution will be exchanged to greedy solution. Therefore, greedy is optimal.

7.2 Counterexample for b

This algorithm is not optimal. A simple counterexample is that if the car has some gas left before it finishes the last route to the destination, based on this algorithm it needs to be filled fully, which may exceed the amount of gas it needs and it is not optimal.

8 Disk (Spring 2022)

You are given n distinct points and one line l on the plane and some constant $r > 0$. Each of the n points is within distance at most r of line l (as measured along the perpendicular). You are to place disks of radius r centered along line l such that every one of the n points lies within at least one disk. Devise a greedy algorithm that runs in $O(n \log n)$ time and uses a minimum number of disks to cover all n points; prove its optimality. You can use the "greedy stays ahead" argument.

8.1 Algorithm

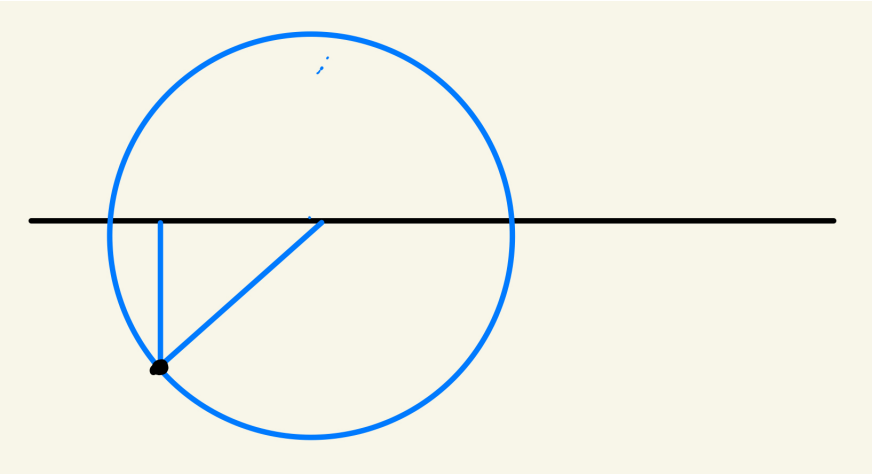

Figure 1: Disk Solution

Assume the line l is in horizontal direction, then we sort n distinct points based on their horizontal locations (from left to right, assume they do not have the same locations). Then when we first meet a uncovered point, we place a disk (with radius as r) at the right most location that could cover the point (shown in the graph). Continue to do this until all the points are covered.

8.2 Proof

8.2.1 Define Solutions

- Let $G_1, G_2, ..., G_k$ be the positions of the disk center selected by the greedy algorithm.
- Let $O_1, O_2, ..., O_m$ be the positions of the disk center selected by the optimal solution.

We want to show that $k \leq m$, or in the other words, greedy algorithm uses at most number of disks as the optimal solution.

8.2.2 Greedy Stays Ahead

We will show that $G_i \geq O_i$ for all i, meaning that the greedy algorithm places the i^{th} center of disk at a position at least as the i^{th} position in the optimal solution

8.2.3 Inductive Argument

For the base case:

- For the first uncover point, because the greedy algorithm always places the center of the disk at the right most position, so $G_1 \geq O_1$

For the inductive step:

- Assume for some i, the greedy algorithm has selected the positions such that $G_j \geq O_j$ for $j \leq i$
- Now consider the $(i+1)^{th}$ center of disk. The greedy algorithm places G_{i+1} at the right most position that could cover the first remaining uncovered point.
- Then by inductive hypothesis, $G_i \geq O_i$. Since the greedy algorithm places the center at the right most point, it ensures that $G_{i+1} \geq O_{i+1}$

8.2.4 Conclusion

- Since $G_i \geq O_i$ for all i, the greedy algorithm stays ahead of the optimal solution at each step
- This implies that the greedy algorithm could cover at least number of points as the optimal solution with same number of disks (the covered range is larger)
- Therefore, the total number of disks used by the greedy algorithm could not be more than the number used by the optimal solution ($k \leq m$)
- Therefore, greedy is optimal.