2 Greedy Algorithms

In this chapter, we consider optimization problems whose solution can be built iteratively as follows: in each step, there is a natural "greedy" choice, and the algorithm is to repeatedly make that greedy choice. In the analysis, we must show that a solution created this way is indeed an optimal solution to the problem.

Sometimes, there are multiple interpretations for what it means to be greedy, and we will have to figure out which choice is the correct one for the problem. And for many problems, there is no optimal greedy solution. But in this chapter, we will only be concerned with problems for which *some* greedy algorithm works.

2.1 Selecting Compatible Intervals

Problem statement. Let $\{1, \ldots, n\}$ denote a set of n events, where each event i occupies a time interval $[s(i), t(i)]$. A subset of events is *compatible* if no two events overlap in time. Our goal is to select a largest subset of compatible events.

Algorithm. Repeatedly select the event that finishes earliest. More formally, first sort the events by their end time so that $t(1) \leq t(2) \leq \cdots \leq t(n)$. Then add the first event to the solution and traverse the list until we reach a non-conflicting event. Add this event as well, and repeat this process until we've traversed the entire list.

Theorem 2.1. *The algorithm optimally solves the problem of selecting a largest subset of compatible intervals.*

Proof. Suppose our solution returns m events, which we denote by $\mathsf{ALG} = \{i_1, \ldots, i_m\}$. Let $\mathsf{OPT} = \{j_1, \ldots, j_{m^*}\}$ denote an optimal solution, which contains m^* events. Assume that ALG and OPT are both sorted in increasing finishing time, so i_1 is the event that finishes first among all n events. This means $t(i_1) \leq t(j_1)$, so if we only consider i_1 and j_1, ALG is "ahead" of OPT because it finishes at least as early.

In general, we can use induction to show that ALG "stays ahead" of OPT at every step, that is, $t(i_k) \leq t(j_k)$ for every $k \leq m$. We've shown the statement is true for $k = 1$. Assuming it is true for $k - 1$, we have $t(i_{k-1}) \leq t(j_{k-1})$. Since j_k starts after j_{k-1} ends, it also starts after i_{k-1} ends. This means j_k is a candidate after the algorithm picks i_{k-1}, so by the algorithm's greedy choice when picking i_k, we must have $t(i_k) \leq t(j_k)$.

In particular, this means $t(i_m) \leq t(j_m)$. Now, for contradiction, suppose $m^* \geq m + 1$. The event j_{m+1} starts after j_m ends, so it also starts after i_m ends. This means the algorithm could have picked j_{m+1} after picking i_m, but it actually terminated. This contradicts the fact that the algorithm processes the entire list of events. □

2.2 The Fractional Knapsack Problem

Problem statement. Suppose $[n] = \{1, \ldots, n\}$ denote a set of n items, where each item i has weight $w_i > 0$ and value $v_i > 0$. We have a knapsack with capacity B. Our goal is to determine $x_i \in [0, 1]$ for each item i to maximize the total value $\sum_{i=1}^{n} x_i v_i$, subject to the constraint that the total weight $\sum_{i=1}^{n} x_i w_i$ is at most B.

Remark. In the original (i.e., not fractional) knapsack problem, we cannot take a fraction of an item, i.e., we must set $x_i \in \{0, 1\}$ for every item i. For this problem, there is no natural greedy algorithm that always returns an optimal solution. However, it can be solved using a technique known as dynamic programming, as we show in Ch. 3.

Algorithm. Relabel the items in decreasing[1] order of "value per weight," so that

$$\frac{v_1}{w_1} > \frac{v_2}{w_2} > \cdots > \frac{v_n}{w_n}.$$

In this order, take as much of each item as possible (i.e., set each $x_i = 1$) until the total weight of selected items reaches the capacity of the knapsack B.

Theorem 2.2. *The greedy algorithm above returns an optimal solution to the fractional knapsack problem.*

Proof. Let $\mathsf{ALG} = (x_1, \ldots, x_n)$ denote the solution returned by the greedy algorithm, and let $\mathsf{OPT} = (x_1^*, \ldots, x_n^*)$ denote an optimal solution. For contradiction, assume $\mathsf{OPT} > \mathsf{ALG}$; let i denote the smallest index such that $x_i^* \neq x_i$. By design of the algorithm, we must have $x_i^* < x_i$. Furthermore, since $\mathsf{OPT} > \mathsf{ALG}$, there must exist some $j > i$ such that $x_j^* > 0$.

We now modify OPT as follows: increase x_i^* by $\epsilon > 0$ and decrease x_j^* by $\epsilon w_i / w_j$. Note that ϵ can be chosen small enough such that x_i^* and x_j^* remain in $[0, 1]$ after the modification. Let OPT' denote this modified solution; notice that OPT and OPT' have the same total weight so OPT' is feasible. Now consider the value of OPT':

$$\mathsf{OPT}' = \mathsf{OPT} + \epsilon \cdot v_i - \frac{\epsilon w_i}{w_j} \cdot v_j = \mathsf{OPT} + \epsilon \cdot \left(v_i - \frac{w_i}{w_j} \cdot v_j \right) > \mathsf{OPT},$$

where the inequality follows from $\epsilon > 0$ and $j > i$. Thus, OPT' is a feasible solution with total value greater than OPT, contradicting the fact that OPT is an optimal solution. □

2.3 Maximum Spacing Clusterings

Problem statement. Let X denote a set of n points. For any two points u, v, we are given their distance $d(u, v) \geq 0$. (Distances are symmetric, and the distance between any point and itself is 0.) The *spacing* of a clustering (i.e., partition) of X is the minimum distance between any pair of points lying in different clusters. We are also given $k \geq 1$, and our goal is to find a clustering of X into k groups with maximum spacing.

Algorithm. Our algorithm is essentially Kruskal's algorithm for Minimum Spanning Tree (Sec. 1.3). In the beginning, each point belongs to its own cluster. We repeatedly add an edge between the closest pair of points in different clusters until we have k clusters. (This is equivalent to computing the MST and removing the $k - 1$ longest edges.)

Theorem 2.3. *The algorithm above returns a clustering with maximum spacing.*

[1] We can assume the v_i / w_i values are unique without loss of generality; we leave the proof as an exercise.

Proof. Let α denote the spacing of the clustering ALG produced by the algorithm, so α is the length of the edge that the algorithm would have added next, but did not. Now consider the optimal clustering OPT. It suffices to show that the spacing of OPT is at most α. If ALG $\neq$ OPT, then there must exist two points $u, v \in X$ such that in ALG, u and v are in the same cluster, but in OPT, u and v are in different clusters.

Now consider the path P between u and v along the edges in ALG. Since the algorithm adds edges in non-decreasing order of length, and the edge corresponding to α was not added, all of the edges in P have length at most α. Furthermore, since u and v lie in different clusters of OPT, there must be some edge on P whose endpoints lie in different clusters of OPT. Thus, the distance between these two clusters is at most α. □

2.4 Stable Matching

Problem statement. Let $G = (X \cup Y, E)$ be a bipartite graph where $|X| = |Y| = n$. The vertices in X represent students, and the vertices in Y represent companies. Each student has a preference list (i.e., an ordering) of the n companies, and each company has a preference list of the n students.

Our goal is to construct a matching function $M \colon Y \to X$ such that each company is matched with exactly one student, and there are no instabilities. An *instability* is defined as a (student, company) pair (x, y) such that y prefers x over $M(y)$, and x prefers y over the company y' such that $M(y') = x$.[2]

Remark. It is not obvious that a stable matching even exists for every possible set of preference lists. In this section, we not only prove this holds, but we do this by presenting an algorithm (known as the Gale-Shapley algorithm) that efficiently returns a stable matching.

Algorithm. If we focus our attention on a particular student x, then the greedy choice to make would be to match that student to their favorite company. So in fact, our algorithm does this in the first round: it assigns every student to their favorite company. If every company receives exactly one applicant, then it's not difficult to show that we'd be done. But if a company receives more than 1 applicant, then it *tentatively* gets matched with that student, and the other applicants return to the pool.

In each subsequent round, an unmatched student applies to their favorite company that hasn't rejected them yet, that company gets tentatively matched with their favorite, and the rejected student (if one exists) returns to the pool. Note that a company never rejects an applicant unless a better one (from the company's perspective) applies.

Theorem 2.4. *Algorithm 4 returns a stable matching.*

Proof. We first show that the algorithm terminates and the output is a matching. Notice that no student applies to the same company more than once, so the algorithm terminates in most n^2 iterations. Now suppose the algorithm ended with an unmatched student x. Whenever a company y receives an applicant, y is (tentatively) matched for the rest of the

[2] If we view X as men and Y as women, then an instability consist of a man and a woman who both wish they could cheat on their respective partners.

Algorithm 4 The Gale-Shapley algorithm

```
Initialize M(y) = ∅ for all y ∈ Y and S = X.
while S is not empty do
    Remove some student x from S.
    Let y denote x's favorite company that hasn't rejected them yet.
    if M(y) = ∅ or y prefers x to M(y) then
        Add M(y) to S and set M(y) = x.          ▷ x and y are tentatively matched
    else
        Return x to S.                           ▷ x is rejected by y
return the matching function M
```

algorithm. Since x must have applied to every company, in the end, every company must be matched. But $|X| = |Y|$, so it cannot be the case that every company is matched while x remains unmatched.

Now we show at M is stable. For contradiction, suppose (x, y) is an instability, where x is matched to some y' but prefers y, and y is matched to $M(y)$ but prefers x. Since x prefers y to y', x must have applied to y at some point. But, since y ends up matched with $M(y)$, y must have rejected x at some point in favor of someone y prefers more. The tentative match for y only improves over time (according to y's preference list), so y must prefer $M(y)$ to x. This contradicts our assumption that y prefers x to $M(y)$, so we are done. □

2.5 Exercises

1. Let $\{1, \ldots, n\}$ denote a set of n customers, where each customer has a distinct processing time t_i. In any ordering of the customers, the *waiting time* of a customer i is the sum of processing times of customers that appear before i in the order. Give an $O(n \log n)$-time algorithm that returns an ordering such that the sum (or average) of waiting times is minimized.

2. Consider the same setup as the problem in Sec. 2.1. Instead of sorting by end time, suppose we sort by non-decreasing start time, duration (i.e., length of the interval), or number of conflicts with other events. For each of these three algorithms, give an instance where the algorithm does not return an optimal solution.

3. Consider the same setup as the problem in Sec. 2.1. Now our goal is to partition the events into groups such that all of the events in a group are compatible, and the number of groups is minimized. Give an $O(n^2)$-time algorithm for this problem.

4. Consider the same setup as the problem in Sec. 2.1. We say a point x *stabs* an interval $[s, t]$ if $s \leq x \leq t$. Give an $O(n \log n)$-time algorithm that returns a minimum set of points that stabs all intervals.

5. Consider the fractional knapsack problem from Sec. 2.2. Prove that the assumption that all v_i/w_i values are unique can be made without loss of generality.

6. Suppose there are an unlimited number of coins in each of the following denominations: 50, 20, 1. The problem input is an integer n, and our goal is to select the smallest number of coins such that their total value is n. The greedy algorithm is to repeatedly select the denomination that is as large as possible.

   (a) Give an instance where the greedy algorithm does *not* return an optimal solution.

   (b) Now suppose the denominations are 10, 5, 1. Show that the greedy algorithm always returns an optimal solution.

7. Consider the stable matching problem described in Sec. 2.4. For any student x and company y, we say that (x, y) is *valid* if there exists a stable matching that matches x and y. For each student x, let $f(x)$ denote the highest company on x's preference such that $(x, f(x))$ is valid. Similarly, for any company y, let $g(y)$ denote the *lowest* valid student on y's preference list such that $(g(y), y)$ is valid.

   (a) Prove that the Gale-Shapley algorithm always matches x with $f(x)$.

   (b) Prove that the Gale-Shapley algorithm always matches y with $g(y)$.

8. An *independent set* of a graph is a subset of vertices such that no two share an edge. Let $T = (V, E)$ be a tree. Give an $O(n)$-time algorithm that returns a maximum independent set (i.e., one with the most vertices) of T. (In Sec. 3.4, we solve the weighted version of this problem.)

9. A *matching* in a graph is a subset of edges such that no two share an endpoint. Let $T = (V, E)$ be a tree. Give an $O(n)$-time algorithm that returns a maximum matching (i.e., one with the most edges) of T. (In Sec. 3.5 Exercise 13, we solve the weighted version of this problem.)