Greedy Algorithms

Mayank Pratap Singh

August 21, 2024

1 Introduction

Greedy algorithms are a class of algorithms used for optimization problems where the goal is to find the minimum or maximum value. These algorithms make a series of choices, each of which looks the best at the moment, aiming to find the global optimum. They are often used in scenarios where finding an exact solution is less important than finding a solution quickly.

2 Optimization Problems

Optimization problems involve finding the best solution among a set of feasible solutions. Examples include minimizing travel cost, maximizing profit, or minimizing time.

2.1 Problem Example: Bengaluru to Delhi

Consider a problem where we need to travel from Bengaluru to Delhi. The available modes of transport include:

- On Road (Driving)
- Flight
- Train
- Bus
- Bike
- Cycling

All these options form the **solution space**. The goal is to minimize both the cost and time of travel.

2.1.1 Feasible Solutions

Feasible solutions are those that meet the problem's constraints. For instance:

- If the constraint is **minimum cost**, then options like Flight and Car would not be feasible due to their higher costs.
- If the constraint is **minimum time**, then options like Bike and Cycling would not be feasible due to the longer time required.

2.1.2 Optimized Solution

An optimized solution is the best solution among all feasible ones. For example:

- If the goal is to minimize both cost and time, taking the Train might be the optimized solution as it balances both constraints.

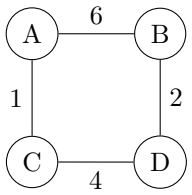
The key difference between a feasible solution and an optimized solution is that there can be multiple feasible solutions, but only one optimized solution.

3 Examples of Greedy Algorithms

3.1 Shortest Path Problem

The shortest path problem involves finding the path between two points that minimizes the distance or time. Dijkstra's algorithm is a common example of a greedy algorithm that solves the shortest path problem.

- **Application**: Find the shortest path from a starting node to a destination node in a weighted graph.
- **Constraint**: Minimize the total distance or time.



3.2 Job Scheduling Problem

In the job scheduling problem, the goal is to maximize profit by completing jobs within their deadlines. The steps involve sorting jobs by profit and scheduling them to maximize the total profit while adhering to deadlines.

Job	Profit	Deadline
J_1	55	5
J_2	65	2
J_3	75	7
J_4	60	3
J_5	70	2
J_6	50	1
J_7	85	4
J_8	68	5
J_9	45	3

Table 1: Job Scheduling Example

Steps:

1. Sort the jobs in descending order of profit.
2. Schedule each job if it can be done within its deadline.
3. Maximize the total profit while ensuring jobs are completed within their deadlines.

Solution: After sorting by profit, we get the sequence: $J_7, J_3, J_5, J_8, J_2, J_4, J_6, J_9$. We then pick jobs to maximize profit while adhering to deadlines.

3.3 Fractional Knapsack Problem

In the fractional knapsack problem, the goal is to maximize the profit of items that can be carried within a weight limit.

Object	Profit	Weight	Profit/Weight
n_1	25	5	5
n_2	75	10	7.5
n_3	100	12	8.3
n_4	50	4	12.5
n_5	45	7	6.4
n_6	90	9	10
n_7	30	3	10

Table 2: Fractional Knapsack Problem Example

Steps:

1. Calculate the Profit/Weight ratio for each item.
2. Sort the items in descending order based on the Profit/Weight ratio.
3. Add items to the knapsack in order of the sorted list until the weight limit is reached.

4. If the next item cannot be added fully, add a fraction of it to fill the knapsack.

Solution: Sorting by Profit/Weight ratio gives the sequence: $n_4, n_6, n_7, n_3, n_2, n_5, n_1$. Then, by following the steps, the maximum profit is calculated to be 337.5 with the given weight constraint.

4 Greedy vs Dynamic Programming

Greedy algorithms provide a simple approach but may not always yield the optimal solution. In contrast, dynamic programming explores all possible solutions and saves intermediate results, ensuring an optimal solution at the expense of increased space complexity.

4.1 Greedy Algorithm

- **Approach**: Make a locally optimal choice at each stage with the hope of finding the global optimum.
- **Space Complexity**: Usually lower as it doesn't store intermediate results.
- **Time Complexity**: Can vary, often depends on sorting or traversal.
- **Limitation**: May not always provide the globally optimal solution.

4.2 Dynamic Programming

- **Approach**: Explore all possible solutions by breaking down problems into simpler sub-problems, storing the results of these sub-problems.
- **Space Complexity**: Higher due to storage of intermediate results.
- **Time Complexity**: Often better than Greedy, especially for overlapping subproblems.
- **Advantage**: Always provides the globally optimal solution.

Trade-offs: Greedy algorithms are faster and use less memory but may not always give the best solution. Dynamic programming is slower and uses more memory but guarantees the best solution.