Design and Analysis of Algorithms Project Research Paper

Presented to Dr. Ashraf AbdelRaouf

Faculty of Engineering Ain Shams University

Project Title : Maximum Interval Scheduling Problem

I. Team Members

Name	ID
Ahmed Mohamed Elmahdy	2100723
Abdelrhman Mohamed Ghazy	19P6651
Hassan Adel Hassan	18P1726

II. PROJECT OVERVIEW

This project aims to explore and compare two algorithmic approaches—Dynamic Programming and the Greedy Approach—by solving the ”Maximum Interval Scheduling Problem.” This problem involves selecting the maximum number of non-overlapping intervals from a given set of intervals. By implementing both approaches, the project will evaluate their efficiency and suitability for solving this problem, offering insights into their computational trade-offs.

III. PROBLEM STATEMENT

A. *Problem Description:*

Given n intervals, where each interval i is represented as [starti,endj], determine the maximum number of non-overlapping intervals that can be included in a schedule. Inputs: An array of intervals: [[start1, end1], [start2, end2], ..., [startn, endn]] Output: The maximum number of non-overlapping intervals.

IV. GOALS AND OBJECTIVES

1. Design and implement solutions using both the Greedy and Dynamic Programming approaches.

2. Analyze and compare the approaches in terms of computational efficiency, time complexity, and implementation complexity.

V. DETAILED COMPARISON BETWEEN GREEDY AND DYNAMIC PROGRAMMING APPROACHES FOR MAXIMUM INTERVAL SCHEDULING PROBLEM

The ”Maximum Interval Scheduling Problem” focuses on selecting the maximum number of non-overlapping intervals from a given set. This problem can be addressed using both Greedy and Dynamic Programming (DP) approaches. While both methods aim to achieve the same result, their methodologies, efficiency, and computational trade-offs differ significantly. Below is a comprehensive comparison between the two:

A. *Approach and Strategy:*

Greedy Approach: The greedy algorithm operates on the principle of making the most optimal choice at each step without considering the global impact of decisions. Specifically, for the interval scheduling problem, the algorithm sorts the intervals by their ending times (or start times in case of ties) and iteratively selects intervals that do not overlap with the previously chosen ones. The key idea is to maximize the number of intervals by always selecting the one that ends the earliest among the remaining candidates.

Sorting Step: The intervals are sorted based on their ending times, ensuring that the smallest end time is always considered first.
Selection Step: Intervals are traversed, and a new interval is added to the solution if and only if its start time is greater than or equal to the end time of the last selected interval.

Dynamic Programming Approach: The dynamic programming algorithm takes a global optimization perspective, systematically solving subproblems and combining their results to address the overall problem. In this case, the DP approach involves creating a table (dp) where each entry dp[i] represents the maximum number of non-overlapping intervals that

can be selected from the first i intervals, assuming the intervals are sorted by their end times .
Sorting Step: Like the greedy approach, intervals are sorted by their ending times to simplify comparisons.
State Transition: For each interval, the algorithm evaluates whether to include the interval in the solution. It checks previous intervals to find the last compatible one (i.e., one that ends before the current interval starts) and adds its dp value to the current interval's contribution.
Recursive Relation: dp[i]=max(dp[i-1],dp[j]+1), where j is the index of the last compatible interval with i

B. Time Complexity:

Greedy Approach:

Sorting: The algorithm requires sorting the intervals based on their end times, which takes O(nlogn), where n is the number of intervals.

Selection: Iterating through the intervals to determine non-overlapping intervals takes O(n).

Overall Complexity: O(nlogn). This makes the greedy approach highly efficient and suitable for large datasets.

Steps in Code: Sorting (customSort function):
A nested loop is used to sort the intervals by their end times, implementing a basic bubble sort. Bubble sort has a time complexity of O(n^2) is the number of intervals.
Selection:
The main greedy logic involves a single pass through the sorted intervals, which takes O(n)
Overall Time Complexity:
Sorting: O(n^2) (due to bubble sort)

Selection: O(n)

Total Complexity:O(n^2)

Note: The complexity here is higher than the O(nlogn) discussed earlier because the code uses bubble sort instead of an efficient sorting algorithm like Merge Sort or Quick Sort.

Dynamic Programming Approach:

Sorting: Sorting: As with the greedy approach, sorting the intervals takes O(nlogn).

.

State Transition: For each interval, the algorithm potentially examines all previous intervals to find the last compatible one, leading to a worst-case complexity of O(n^2)

.

Optimization with Binary Search: By using binary search to find the last compatible interval, the complexity can be improved to O(nlogn), but this requires additional effort in implementation.

.

Overall Complexity: Without optimization O(n^2), with optimization O(nlogn).

.

Steps in Code:
Sorting (customSort function):

Like the greedy approach, this uses bubble sort with a complexity of O(n^2)

Dynamic Programming Table (solveDynamicProgramming function): For each interval i, it looks back at all previous intervals j to find the last compatible one. This nested loop results in a complexity of O(n^2).
Overall Time Complexity:

Sorting: O(n^2) (due to bubble sort)
DP computation Table:O(n^2)
Total Complexity: O(n^2)

Note: With optimized sorting (e.g., O(nlogn) sorting) and binary search for finding compatible intervals, the DP complexity could be reduced to O(nlogn), as discussed earlier.

While both can achieve O(nlogn) in their best implementations, the greedy approach is inherently simpler and more direct.

C. *Space Complexity*

Greedy Approach: The greedy algorithm does not require additional storage apart from sorting and a few variables to track the last selected interval.

Space Complexity:
O(1) additional space, making it space-efficient.

Dynamic Programming Approach: The DP algorithm requires a dp table to store intermediate results for all intervals.

Space Complexity:
O(n), due to the storage of the dp array. While this is not prohibitive, it is greater than the space requirement of the greedy approach.

D. *Simplicity and Implementation*

Greedy Approach: The greedy algorithm is straightforward and easy to implement. Its logic is intuitive and involves fewer computational steps, making it suitable for quick solutions where performance is critical.

Dynamic Programming Approach: The DP algorithm is more complex and involves careful handling of state transitions and indexing. It requires a deeper understanding of recursion and subproblem optimization, making it less intuitive than the greedy method.

E. *Suitability for Different Scenarios*

Greedy Approach:
Best suited for scenarios where the problem is naturally "greedy-solvable," meaning locally optimal decisions lead to globally optimal solutions. Ideal for large datasets due to its simplicity and lower overhead.

Dynamic Programming Approach:
More versatile and powerful, capable of handling problems where greedy solutions fail or are suboptimal. Useful when the problem requires exploring multiple potential solutions and combining results

F. *Computational Trade-offs*

Greedy Approach:
Strengths: Efficiency, simplicity, and minimal space usage.
Weaknesses: Limited applicability; it works only if the greedy choice property and optimal substructure hold true for the problem.

Dynamic Programming Approach:
Strengths: General applicability and ability to handle more complex variations of the problem.
Weaknesses: Higher computational and space overhead, making it less efficient for simpler problems.

VI. CONCLUSION

Both the greedy and dynamic programming approaches provide solutions to the maximum interval scheduling problem, but their efficiency and practicality differ. The greedy algorithm is the preferred choice when its properties are satisfied, due to its simplicity and efficiency. On the other hand, the dynamic programming approach is more powerful and adaptable, offering solutions in cases where greedy methods fall short. For this specific problem, the greedy approach is typically sufficient and offers a clear advantage in terms of both time and space complexity. The provided code for both approaches is less efficient than the theoretically discussed complexities due to the use of bubble sort for sorting intervals and, in the case of the DP approach, the absence of binary search to optimize the compatibility checks. To achieve the O(nlogn) complexity for the greedy approach or the optimized DP approach, replacing bubble sort with a faster sorting algorithm (like Merge Sort or Quick Sort) and incorporating binary search for compatibility checks in the DP implementation would be necessary.

VII. REFERENCES

- GeeksforGeeks: Greedy Approach vs Dynamic Programming
- Fiveable
- Medium: Greedy Approach vs Dynamic Programming