Yılmaz Efe Şenkal, Neris Lara Karasulu

343-Project Report

**TSP:**

* The Nearest Neighbor algorithm is a heuristic method for approximating the shortest possible route that visits each city exactly once and returns to the origin city.

A screenshot of a computer

Description automatically generated



In this code Nearest Neighbourhood algorithm algorithm sorts unvisited n-1 from smaller to larger with complexity O(n-1) for each node with time complexity O(n). Since loops are nested (n-1)\*n yields n^2-n and when we eliminate less affecting terms algorithm complexity gives O(n^2).

* Than we adapt 1-opt improvement heuristic to improve solution by choosing an edge and reconnecting again in a different way to check tour.

A close-up of a computer screen

Description automatically generated



In this code we apply one\_opt heuristic, first we iterate over len(tour)-1 for edges with complexity O(n) than with nested loop we iterate over (2, len(tour)) for remain edges with complexity O(n-2). Since these are nested loops we multiply n\*(n-2) which gives n^2-2n with most affecting term n^2 than complexity of this algorithm is O(n^2).

We first apply NN with complexity O(n^2) then 1-opt with complexity O(n^2). To perform both algorithm consecutively we need to sum complexities n^2+n^2 gives 2n^2. When we eliminate coefficients iterative complexity also gives O(n^2).

**Knapsack(value/weight):**

The Knapsack Algorithm addresses the problem of selecting items with given weights and values to maximize the total value within a limited capacity, like a backpack. There are two main types: 0/1 Knapsack, where items are taken entirely or not at all, typically solved using dynamic programming; and Fractional Knapsack, where items can be fractionally included, often tackled with greedy algorithms. We developed a greedy approach that prioritizes based on value/weight in below code. In this case value is popularity, weight is duration. Our aim is to maximize popularity by allocating best songs by providing total duration capacity.



A screenshot of a computer

Description automatically generated



Our code has a time complexity of O(n log n). Let's break down the complexity analysis:

The primary operation in this code is the sorting of the songs list based on the popularity/duration ratio. The sort operation has a time complexity of O(n log n) in the average and worst-case scenarios, where "n" is the number of elements in the list. Sorting dominates the overall time complexity.

The subsequent operations, such as the for loop and conditional statements, have a linear time complexity, O(n), as they iterate through the sorted list and perform constant-time operations.

Therefore, considering the dominant sorting operation, the overall time complexity of the knapsack method is O(n log n). This complexity indicates that the execution time grows logarithmically with the number of songs in the input list.