

**DESIGN & ANALYSIS OF ALGORITHMS**

**(BCSDA402)**

|  |  |
| --- | --- |
| **Student Name** | **HS Hariprasad** |
| **USN** | **4VV23CS091** |
| **Branch** | **CSE** |
| **Section/Batch** | **B** |
| **Faculty In-charge** | **Dr. Nithin Kumar** |

**2024-25**

**Comparative Analysis of Greedy and Dynamic Programming Approaches for the Traveling Salesman Problem**

**Problem Statement and Complexity**

**Problem Statement**  
The Traveling Salesman Problem (TSP) is a classic optimization problem where the goal is to find the shortest possible route that visits a set of cities exactly once and returns to the starting city. This problem is NP-hard, meaning that as the number of cities increases, finding the optimal solution becomes computationally infeasible.

**Algorithmic Complexity:**

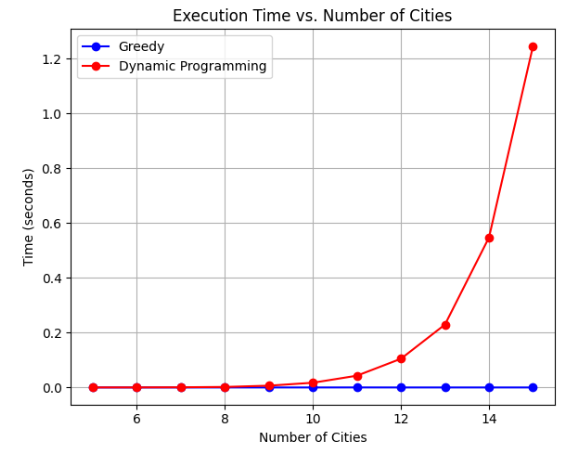
* **Dynamic Programming (Held–Karp Algorithm):**
  + **Time Complexity:** O(n²·2ⁿ)
  + **Space Complexity:** O(n·2ⁿ)
  + *Explanation:* This method guarantees an optimal solution by exploring all subsets of cities, but its exponential growth makes it suitable only for small instances (e.g., less than 15 cities).
* **Greedy (Nearest Neighbour Heuristic):**
  + **Time Complexity:** O(n²)
  + **Space Complexity:** O(n)
  + *Explanation:* The greedy approach builds the tour by always visiting the nearest unvisited city. It is computationally efficient and scales well to larger numbers of cities, but it does not always find the optimal solution.

**Implementation**

**Overview of the Implementation:**

* The program is written in Python and designed to run in a Jupyter Notebook.
* **City Generation & Distance Matrix:**
  + Random (x, y) coordinates are generated to simulate the positions of cities.
  + A distance matrix is computed to store the Euclidean distances between all pairs of cities.
* **Greedy TSP Algorithm:**
  + The algorithm starts at a fixed city (city 0) and iteratively selects the nearest unvisited city until all cities have been visited.
  + The route is then closed by returning to the starting city.
* **Dynamic Programming (DP) TSP Algorithm (Held–Karp):**
  + A DP solution using bitmasking is implemented to guarantee an optimal tour.
  + The DP approach fills a table of costs for visiting subsets of cities and reconstructs the tour path by evaluating transitions.
* **Visualization:**
  + Static plots display the city locations and tour paths (both greedy and DP approximations) side by side.
  + Execution time and total tour length are measured and plotted against the number of cities.
  + An animation shows the step-by-step construction of the greedy tour, using matplotlib’s animation module and interactive Jupyter Notebook widgets.

**Result Interpretation**

****

**Execution Time vs. Number of Cities:**

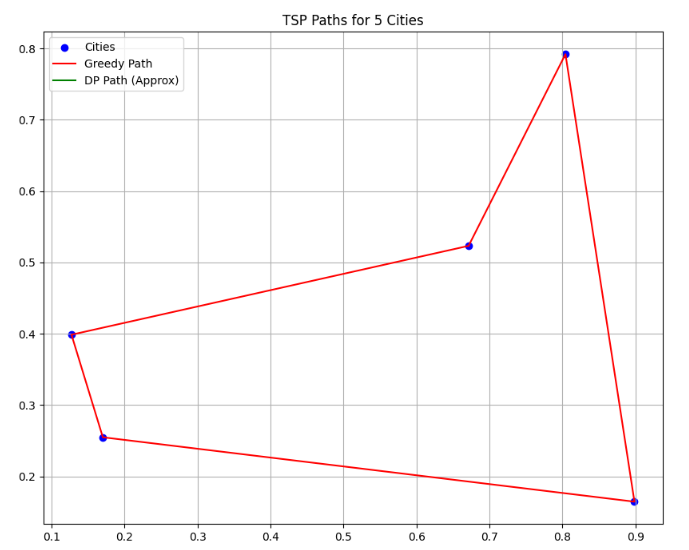
* **Graph Description:**  
  The graph plots the average execution time (in seconds) on the Y-axis against the number of cities (ranging from 5 to 15) on the X-axis. Two curves are displayed:
* A blue line represents the greedy algorithm.
* A red line represents the dynamic Programming (DP) solution.
* **Interpretation:**  
  Explain that the greedy algorithm exhibits a relatively low and almost quadratic growth in runtime, making it efficient for larger instances. In contrast, the DP solution’s runtime increases steeply due to its exponential complexity. This illustrates the trade-off between computational efficiency and optimality.

A graph with a line and a line

Description automatically generated

**Path Length vs. Number of Cities:**

* **Graph Description:**  
  This graph shows the average tour length (total distance) obtained by both methods as the number of cities increases.
* **Interpretation:**  
  Discuss that while the DP method theoretically yields the optimal solution, the greedy method—despite being heuristic—produces tour lengths that are close to the optimal ones for many instances. This visual comparison demonstrates the potential acceptability of the greedy solution in scenarios where speed is more critical than perfect optimality.

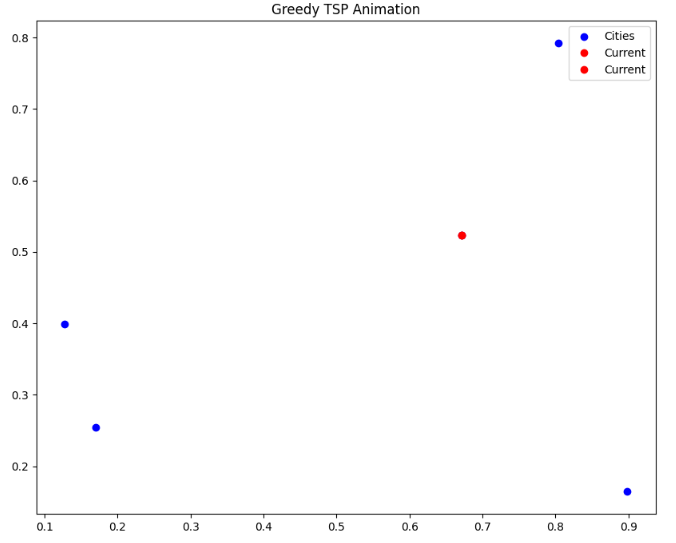


**Static Plot of TSP Paths (for a small instance, e.g., 5 cities):**

* **Graph Description:**  
  A scatter plot displays the randomly generated city positions. Overlaid on this, the tour computed by the greedy algorithm is shown as a red line connecting the cities (with the return to the starting city). A green line (or label) indicates the approximated DP path.
* **Interpretation:**  
  Point out that the graph visually represents how the algorithm connects the cities. Emphasize that while the DP approach would normally yield an optimal route, the approximation used here (which is similar to the greedy path for demonstration) helps the evaluator see the overall structure of the tour.

**Greedy TSP Animation:**

* **Animation Description:**  
  An animation illustrates the step-by-step process of the greedy algorithm. In each frame, a red line is drawn from the current city to the next nearest unvisited city. A red dot highlights the current city.
* **Interpretation:**  
  Explain that the animation helps in understanding how the greedy method incrementally builds the tour. The interactive slider allows the user to pause and inspect each intermediate step, providing insights into the algorithm's local decision-making process and how these decisions eventually form the complete tour.

**SUMMARY**

* The greedy algorithm, while not always optimal, offers significant speed advantages and produces reasonably good results for larger instances.
* The DP method is capable of finding the optimal solution but is limited to small datasets due to its exponential complexity.
* The visualizations and animations serve as powerful tools to bridge the gap between abstract algorithmic theory and tangible, interpretable results, providing clear evidence of the trade-offs between optimality and efficiency.