Design and Analysis of Algorithms (24CS2203)

BRANCH & BOUND TSP FOR SALES ROUTE PLANNING

Karishma Suthar : 2420030384

Diksha Yadav : 2420030001

Course Instructor

Dr. J Sirisha Devi

Professor

Department of Computer Science and Engineering

Case study - statement

Problem: Optimizing Sales Route Planning
The goal of sales route planning is to find a single, continuous tour that visits a set of customer locations (cities) exactly once and returns to the starting point, while adhering to various constraints.
It is a classic combinatorial optimization problem faced by logistics companies, delivery services, and traveling sales teams.
Objective:
To create a valid sales route (a complete tour) that **minimizes total travel distance or cost** (e.g., fuel, time).
Key Constraints:
Completeness: The final route must visit **every single customer location** on the list.
Cycle (Tour): The route must **start and end** at the same location (e.g., the sales office or home base).
Uniqueness: Each customer location must be visited **exactly once**.
Cost Matrix: The costs (distances, time) between every pair of locations are known and fixed.
Time Windows (Optional): (If added to the problem) The salesperson must arrive at certain locations within pre-defined time slots.

Algorithm /Pseudo code

What is the Branch and Bound (B&B) Strategy?

•**Branch and Bound** is an algorithm design paradigm for solving discrete and combinatorial optimization problems[1]. It systematically explores a tree of all possible solutions, called the **state-space tree**.

Core Concepts:

1. **Branching:** The process of dividing a problem into smaller, more manageable subproblems.In our case (TSP), a "branch" represents a decision, such as **choosing the next unvisited customer location** to add to the current partial route.
2. **Bounding:** For each subproblem (or node in the tree), we calculate a **lower bound** (LB) on the cost.This bound is an **optimistic estimate** of the best possible total route cost that can be achieved from that point onwards.
3. **Pruning:** This is the key to B&B's efficiency. If the **lower bound** (LB) of a subproblem is already greater than the cost of a known complete solution (the "upper bound"), we can **discard the entire branch** of the tree stemming from that subproblem.We know it cannot lead to a better solution than what we've already found.

Algorithm /Pseudo code

```
function BranchAndBound_TSP(CostMatrix,
Cities): // Priority queue stores nodes (partial
routes), prioritized by lower bound cost
[cite_start]PQ = new PriorityQueue() [cite: 35] //
Create root node with the starting city path and
add to queue rootNode = create_node(path:
[StartCity], cost: 0) rootNode.lower_bound =
calculate_lower_bound(rootNode)
[cite_start]PQ.add(rootNode) [cite: 38]
[cite_start]best_cost = infinity // Stores the
Upper Bound (UB) [cite: 38]
[cite_start]best_route = null [cite: 39]
[cite_start]while PQ is not empty: [cite: 40]
[cite_start]currentNode = PQ.pop() // Get the
most promising node (lowest LB) [cite: 41, 42] //
Pruning Step 1: If this node's LB is already worse
than our best complete route (UB), discard it.
[cite_start]if currentNode.lower_bound >=
best_cost: [cite: 44] [cite_start]continue [cite: 45]
```

// If the route is complete (visited all cities and is ready to return to start) [cite_start]if is_complete(currentNode.path): [cite: 47] // Calculate the total cost including the return trip to StartCity [cite_start]current_cost = calculate_total_cost_with_return(currentNode.path) [cite: 48] [cite_start]if current_cost < best_cost: [cite: 49] [cite_start]best_cost = current_cost [cite: 50] [cite_start]best_route = currentNode.path [cite: 51] [cite_start]continue [cite: 52] // Branching Step: Create new solutions by visiting an unvisited city current_city = get_last_city(currentNode.path) [cite_start]for each unvisited_city in Cities: [cite: 55] // Create a new node by adding this city to the path [cite_start]newNode = create_node_from(currentNode, unvisited_city) [cite: 58] // Pruning Step 2: Only explore branches that could beat the best solution [cite_start]newNode.lower_bound = calculate_lower_bound(newNode) [cite: 60] [cite_start]if newNode.lower_bound < best_cost: [cite: 61] [cite_start]PQ.add(newNode) [cite: 62] [cite_start]return best_route [cite: 63]

Time Complexity

The time complexity of the Branch and Bound algorithm is highly dependent on the problem instance and the **quality of the bounding function**[1].

Worst-Case Complexity: Exponential

•In the worst-case scenario, the algorithm may fail to prune any significant branches and ends up exploring the entire state-space tree[2].

•The complexity is given by $\text{O}(b^d)$[3], where:

- •b is the **branching factor** (the number of choices at each decision point, e.g., the number of unvisited cities available from the current city).
- •d is the **depth of the tree** (the number of decisions to be made, e.g., the total number of cities to visit, N).

•*Note: For TSP, the worst-case complexity can approach $\text{O}(N!)$ (N-factorial).*

Practical Performance:

•The true power of Branch and Bound lies in its ability to **prune the search space**[4].

•A **tight lower bound** is crucial[5]. A good bounding function that provides a close estimate of the final cost will lead to more aggressive pruning, drastically reducing the actual runtime[6].

•For many real-world sales route planning problems, B&B significantly outperforms brute-force enumeration and finds optimal solutions in a reasonable amount of time[7].

Space Complexity

The space complexity is determined by the maximum number of nodes (partial routes) that need to be stored in memory at any given time, which typically corresponds to the nodes in the **priority queue**.

Worst-Case Complexity: Exponential

- Similar to time complexity, the space required can be **exponential** in the worst case.
- The complexity is also $\text{O}(b^d)$. This occurs when a large number of nodes across the width of the search tree have promising lower bounds and must be kept in the queue for future exploration.

Factors Influencing Space Usage:

- **Bounding Function:** A better bounding function prunes more nodes, which not only saves time but also **reduces the number of nodes stored** in the priority queue.
- **Search Strategy:** The choice of which node to explore next (e.g., Best-First Search as used in the pseudocode) influences which nodes are kept in memory.