The Chinese University of Hong Kong (Shenzhen)

CSC4120

Design & Analysis of Algorithm

Project Report: Party Together

Authors:
Bryan Delton Tawarikh Sibarani,
Kelvin Ignatius

Student Number:
121040028
121040017

May 19, 2024

1 Metric Travelling Salesman Problem (M-TSP)

The solver is designed to find an optimal tour that visits each node in a given graph exactly once and returns to the starting node, with the constraint that the graph is metric (the distance between two nodes respects the triangle inequality) and undirected. The solver, defined in the function `mtsp_dp(G)`, takes a NetworkX graph `G` as its input. The implementation involves several key steps:

1.1 Graph Representation

- Node Initialization: All nodes in the graph are indexed starting from 0.
- Edge Weights: The graph is expected to have weights assigned to each edge, defining the cost or distance between nodes.

1.2 Dynamic Programming Table

- A dynamic programming table `memo` is initialized to store the minimum cost to reach each subset of nodes ending at each node. The size of the table is $2^N \cdot N$, where N is the number of nodes.
- The base case is set with the starting node (node 0) having zero cost.

1.3 State Transition

- The algorithm iteratively updates the `memo` table by considering all subsets of nodes and all possible last nodes of paths through these subsets.
- For each pair of nodes (u, v), the algorithm checks if moving from node u to node v can provide a cheaper path to the subset ending at node v.

1.4 Optimal Tour Construction

- After filling the dynamic programming table, the algorithm finds the minimum cost to return to the starting node from any node.
- The path or tour is reconstructed in reverse through backtracking, starting from the node that results in the minimum cost to complete the tour

The computational complexity is $\mathcal{O}(N^2 \cdot 2^N)$, due to the need to evaluate all subsets of nodes.

2 Pickup from Home Problem (PTHP)

The Pickup from Home Problem involves visiting a set of specified home nodes and returning to a starting node, minimizing the total travel distance. This problem is transformed into a TSP, where each home node must be visited exactly once, creating a direct analogy to the TSP's requirements.

2.1 Graph Transformation

The function `create_G_prime` takes an original graph `G` and a list of nodes, and constructs a new graph `G_prime` where:

- Every pair of nodes is directly connected.
- The weight between any two nodes is set to the shortest path distance between them in the original graph, calculated using NetworkX's shortest path length functionality.

This transformation ensures that the new graph `G_prime` is complete and metric, adhering to the triangle inequality, which is critical for the application of the TSP dynamic programming solution.

2.2 Dynamic Programming Solver

The function `mtsp_dp` is used to find an optimal tour in the transformed graph `G_prime`. This function applies a dynamic programming approach to solve the TSP efficiently.

2.3 Reconstructing the Tour

After solving the TSP on `G_prime`, the original PTHP is solved by reconstructing the tour in the original graph `G`. This involves:

- Starting at the initial node.
- Traversing through the shortest paths between consecutive nodes in the `mtsp_tour`.

The function `create_php_tour` concatenates these paths to form a complete tour that visits all specified home nodes.

2.4 Result

The result of PTHP minimum path cost from the given inputs are summarized as follows.

Input No.	PTHP Cost
1	4.0
2	69.0
3	142.0
4	130.2
5	271.0
6	252.9
7	886.0
8	1743.0
9	7644.0
10	18192.0

Table 1: PTHP Costs by Input Number

3 Party Together Problem (PTP)

The solution to the Party Together Problem (PTP) involves several steps and utilizes various algorithms to achieve an efficient transportation plan.

3.1 Graph Representation

The city is represented as a graph $G = (V, E)$, where V is the set of locations and E is the set of roads connecting these locations. Each road has a non-negative weight representing the distance between two locations.

3.2 All-Pairs Shortest Path Calculation

The first step in solving the problem is to compute the shortest paths between all pairs of nodes in the graph. This is achieved using the Floyd-Warshall algorithm, which computes the shortest paths between every pair of nodes in $O(|V|^3)$ time.

3.3 Traveling Salesman Problem (TSP)

The core part of the solution involves solving a variation of the Traveling Salesman Problem (TSP). The TSP solver function calculates the shortest path that visits a given number of homes and returns the tour. This is done by creating a subgraph G' consisting of the starting location (home) and the locations of the friends to be visited. The tour is then computed using a dynamic programming approach.

3.4 Random TSP Solver

To introduce variability and explore different potential tours, a random TSP solver function is implemented. This function randomly selects a subset of friends' homes to visit and computes a tour for this subset.

3.5 Cost Function

A cost function is defined to evaluate the efficiency of a tour. The cost consists of two components: driving cost and walking cost. The driving cost is the total distance traveled by the car, multiplied by the constant α. The walking cost is the sum of the shortest walking distances for each friend from their home to the nearest pickup location in the tour.

3.6 PTP Algorithm

The main algorithm iteratively improves upon potential tours using a combination of deterministic and random methods. It starts with a few initial tours generated by the TSP solver and the random TSP solver. In each iteration, it attempts to modify the tour by adding or removing nodes and recalculating the cost. The algorithm maintains a list of the best tours found so far.

Algorithm 1 PTP Algorithm with Insert/Delete Heuristic

$T^1 \leftarrow \{0\}$
$n \leftarrow |V|$ ▷ Number of nodes
for $k = 1, 2, \ldots$ **do**
  for $i = 1, 2, \ldots, n$ **do** ▷ Compute one node change of T^k
    if $i \in T^k$ **then**
      $T_i^k = T^k$.remove(i) ▷ Directly remove
    else
      $T_i^k = T^k$.least_cost_insert(i) ▷ Multiple places to insert. When connecting to a node in the tour use the shortest path to that node. Take the one with minimum $c(T)$
    end if
  end for
  $i^k = \arg\min_i c(T_i^k)$
  if $c(T^k) \leq c(T_{i^k}^k)$ **then**
    Break
  else
    $T^{k+1} = T_{i^k}^k$
  end if
end for

3.7 Final Tour Selection

After a predefined number of iterations, the algorithm selects the tour with the lowest cost. This tour begins and ends at the starting location (home) and includes all the pickup locations. The tour is adjusted to ensure it meets the constraints, such as starting and ending at the home location.

3.8 Pickup Locations Dictionary

Finally, a dictionary of pickup locations is created. This dictionary maps each pickup location in the tour to the list of friends who will be picked up at that location. The pickup location for each friend is chosen to minimize their walking distance. The average cost of the PTP solver for the given 10 test cases is between 2100-2200.

4 Theoretical Questions

Question 5.1 Show that PTP is NP-hard.
Hint: Are there values for α for which PHP = PTP (the solution of PHP is obtained by solving PTP)? Since PHP is NP-hard, then PTP is also NP-hard. In general, we would expect PHP to give a sub-optimal solution for PTP since we don't take the choice of pick-up locations into the optimization. That is to say, in any instance (G, H, α) of the PTP, let C_{php} and C_{ptpopt} be the total cost of the solution obtained from solving PHP (G, H) and the optimal solution of PTP, respectively. Define $\beta = \frac{C_{\text{php}}}{C_{\text{ptpopt}}}$. Clearly, $\beta \geq 1$. We are interested to know how bad β can become if an adversary is free to choose the parameters.
Solution:
To demonstrate the NP-hardness of PTP, we can employ a reduction from PHP to PTP. Firstly, let's recall that a problem X is NP-hard if every problem in NP can be polynomially reduced to X. Given that PHP (Pickup from Home Problem) is NP-hard, our aim is to establish that PTP (Party Together Problem) is at least as challenging as PHP. Consider an instance of PHP. Here, we are given a fixed set of pickup locations (the friends' homes) and need to determine the most efficient route for picking up all friends and returning to the initial point (your home). Let's denote this instance as (G, H, α), where G signifies the graph, H represents the set of friends' homes, and α denotes the relative cost of driving compared to walking.

We can map PHP to PTP by setting $\alpha = 0$. In this context, the driving cost becomes insignificant since α diminishes the driving cost to zero. Consequently, the optimal solution for PTP with $\alpha = 0$ is to directly pick up each friend from their home, mirroring the essence of the PHP problem. This reduction demonstrates that solving PTP with $\alpha = 0$ is tantamount to solving PHP. Given that PHP is NP-hard and we have established that solving PHP can be reduced to solving PTP with $\alpha = 0$, it logically follows that PTP is also NP-hard.

Question 5.2 Show that the cost of PHP is at most twice of that of the optimal solution (which we don't know). That is, $\beta = \frac{C_{\text{php}}}{C_{\text{ptpopt}}} \leq 2$. Also, demonstrate that this bound is tight, i.e., there is an instance where $\beta = 2$ (at least asymptotically). You can assume $\alpha = 1$ for simplicity.
Solution:
To demonstrate that the cost of PHP is at most twice that of the optimal solution of PTP, we aim to prove that $\beta = \frac{C_{\text{php}}}{C_{\text{ptpopt}}} \leq 2$.
Let's define:

- C_{php}: the cost of the solution obtained from solving PHP.
- C_{ptpopt}: the cost of the optimal solution of PTP.

Initially, note that PHP is a constrained version of PTP, where the pickup locations are fixed at the friends' homes. Consequently, the optimal PHP solution cannot surpass (i.e., have a lower cost than) the optimal PTP solution, given PTP's flexibility in selecting pickup locations. Thus, we have:

$$C_{\text{php}} \geq C_{\text{ptpopt}}$$

Now, let's examine an optimal PTP solution, where the choice of pickup locations is optimal. If we confine these pickup locations to those used in PHP (the friends' homes), the resultant solution might be suboptimal but cannot exceed twice the cost of the optimal PTP solution. This can be reasoned as follows:

1. For any pair of pickup and delivery locations, the additional distance required to adjust the pickup location in PHP compared to the optimal pickup location in PTP is limited by the diameter of the graph.
2. In the worst-case scenario, this extra distance can double the cost of the optimal PTP solution, resulting in:

$$C_{\text{php}} \leq 2 \cdot C_{\text{ptpopt}}$$

Therefore, $\beta = \frac{C_{\text{php}}}{C_{\text{ptpopt}}} \leq 2$.
To demonstrate the tightness of this bound, consider an instance where the optimal pickup locations in PTP are positioned at points that minimize the total travel cost. In PHP, if we are compelled to utilize pickup locations that are at a maximum distance from these optimal locations, the travel cost can be nearly doubled. Hence, there exists an instance where $\beta = 2$ asymptotically.