THE TRAVELING SALESMAN PROBLEM

Amanur Rahman Saiyed

Indiana State University

November 22, 2011
The Goal of the Traveling Salesman Problem (TSP) is to find the shortest tour of a select number of cities with the following restrictions:

- You must visit each city once and only once.
- You must return to the original starting point.
Problem Statement

Given a list of cities and their pairwise distances, the task is to find a shortest possible tour that visits each city exactly once.

Applications

- Planning
Problem Statement

Given a list of cities and their pair wise distances, the task is to find a shortest possible tour that visits each city exactly once.

Applications
- Planning
- Logistics
Problem Statement

Given a list of cities and their pairwise distances, the task is to find a shortest possible tour that visits each city exactly once.

Applications

- Planning
- Logistics
- Microchips manufacture
Why is the TSP difficult to solve?

- 5 cities:
  \[5! = 5 \times 4 \times 3 \times 2 \times 1 = 120\]
Why is the TSP difficult to solve?

- 5 cities:
  \[ 5! = 5 \times 4 \times 3 \times 2 \times 1 = 120 \]

- 25 cities:
  \[ 25! = 25 \times 24 \times 23 \times 22 \times \cdots \times 2 \times 1 = 15,511,210,043,330,985,984,000,000 \]
Why is the TSP difficult to solve?

- 5 cities:
  \[ 5! = 5 \times 4 \times 3 \times 2 \times 1 = 120 \]

- 25 cities:
  \[ 25! = 25 \times 24 \times 23 \times 22 \times \cdots \times 2 \times 1 \]
  \[ = 15,511,210,043,330,985,984,000,000 \]

- 100 cities:
  \[ 100! = 100 \times 99 \times 98 \times \cdots \times 3 \times 2 \times 1 = \]
  \[ 93,326,215,443,944,152,681,699,238,856,266,700,490,715,968,\]
  \[ 264,381,621,468,592,963,895,217,599,992,229,915,608,941,463,\]
  \[ 976,156,518,286,253,697,920,827,223,758,251,185,210,916,864,\]
  \[ 000,000,000,000,000,000,000,000,000,000,000 \]
Exact Solutions

Brute-force Method

- Find all possible routes and their respective distances. The route with the least distance is selected.
- This method is convenient for relatively small number of nodes.
- Time complexity is $O(n!)$

Figure: A 4 city TSP
Branch and Bound Method

- Branching recursively divides the domain into feasible sub domains.
Branch and Bound Method

- Branching recursively divides the domain into feasible sub domains.
- Bounding determines upper and lower bounds for the optimal solution in a feasible sub domain.
Branch and Bound Method

- Branching recursively divides the domain into feasible sub domains.
- Bounding determines upper and lower bounds for the optimal solution in a feasible sub domain.
- Can be used to process TSPs containing 40-60 cities.
Feasibility of Exact Solutions

- Best for small number of cities.
Feasibility of Exact Solutions

- Best for small number of cities.
- Time Complexity is very high.
Feasibility of Exact Solutions

- Best for small number of cities.
- Time Complexity is very high.
- Not useful for large set of nodes.
Feasibility of Exact Solutions

- Best for small number of cities.
- Time Complexity is very high.
- Not useful for large set of nodes.
- So, for large number of nodes, we use approximation techniques.
Approximate Solutions

Nearest Neighbour
This is perhaps the simplest and most straightforward TSP heuristic. The key to this algorithm is to always visit the nearest city, then return to the starting city when all the other cities are visited.

Nearest Neighbour, $O(n^2)$

- Select a random city.

Figure: Network of 4 city TSP
Approximate Solutions

Nearest Neighbour
This is perhaps the simplest and most straightforward TSP heuristic. The key to this algorithm is to always visit the nearest city, then return to the starting city when all the other cities are visited.

Nearest Neighbour, $O(n^2)$

- Select a random city.
- Find the nearest unvisited city and go there.

Figure: Network of 4 city TSP
Approximate Solutions

Nearest Neighbour
This is perhaps the simplest and most straightforward TSP heuristic. The key to this algorithm is to always visit the nearest city, then return to the starting city when all the other cities are visited.

Nearest Neighbour, $O(n^2)$

- Select a random city.
- Find the nearest unvisited city and go there.
- Are there any unvisited cities left? If yes, repeat step 2.

Figure: Network of 4 city TSP
Approximate Solutions

Nearest Neighbour
This is perhaps the simplest and most straightforward TSP heuristic. The key to this algorithm is to always visit the nearest city, then return to the starting city when all the other cities are visited.

Nearest Neighbour, $O(n^2)$

- Select a random city.
- Find the nearest unvisited city and go there.
- Are there any unvisited cities left? If yes, repeat step 2.
- Return to the first city.

Figure: Network of 4 city TSP
Greedy

Greedy algorithm is the simplest improvement algorithm. It starts with the departure Node 1. Then the algorithm calculates all the distances to other $n - 1$ nodes. Go to the next closest node. Take the current node as the departing node, and select the next nearest node from the remaining $n - 2$ nodes. The process continues until all the nodes are visited once and only once then back to Node 1. When the algorithm is terminated, the sequence is returned as the best tour.

Greedy, $O(n^2 \log_2(n))$
Iterative improvement Methods

k-opt heuristic
- Take a given tour and delete $k$ mutually disjoint edges. Reassemble the remaining fragments into a tour, using exact algorithms which improve the tour.

v-opt heuristic
- The variable-opt methods do not fix the size of the edge set to remove. Instead they grow the set as the search process continues.
2-opt Heuristic

- A special case of k-opt heuristic method.
- Iteratively remove two edges and replace them with two different edges which complete the tour.
- The sum of the sizes of the new edges has to be lesser than that of the existing ones.
- Then only, we can have an optimized tour.
Hybrid approach for TSP

- An approximation technique to find an approximate solution.
- An enhancement technique applied on the approximate solution obtained from previous step.
- This approach gives an approximate solution to TSP which is very close to the optimal solution.
Nearest neighbour with 2-opt improvement

- An approximate solution is found using the Nearest Neighbor method.
- The approximate solution is then improved by using the 2-opt heuristic.
- The resultant improvised approximate solution to the TSP is found.
We generate the nodes randomly, the values of $x$ and $y$, and save them as a linked list.

1. We can input the number of nodes we want to have in the problem.
Nearest Neighbour method

1. We iteratively move through the unvisited nodes in the linked list and save the nearest node.

   - Visited Nodes
   - Current Node
   - Nearest Node
   - Unvisited Nodes

2. When we find the nearest node, we move it to the next node of the current node.
Visisted Nodes

Unvisited Nodes

Move the nearest node as the next visited node

- We then move the current node to the next node, which is nothing but the nearest node to the current node in the previous iteration.
1. The iterations are repeated till the end of the linked list
2. Finally, all the nodes are connected through nearest neighbor method
2-Opt Optimization

The two costly edges are replaced by other two edges which make the tour smaller, without affecting the tour properties.

The final optimized tour is obtained.
Applying 2-opt heuristic in the program

- We iteratively go through the edges of the tour in the linked list and go through all possible disjoint edges.
Applying 2-opt heuristic in the program

- We iteratively go through the edges of the tour in the linked list and go through all possible disjoint edges.
- If there is any optimization possible, that is made by modifying the linked list.
Applying 2-opt heuristic in the program

- We iteratively go through the edges of the tour in the linked list and go through all possible disjoint edges.
- If there is any optimization possible, that is made by modifying the linked list.
- The improved solution of the Traveling Salesman Problem is found.
After generating the nodes randomly

After implementing greedy algorithm
References


http://www.tsp.gatech.edu/