VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE2003 DATA STRUCTURES AND ALGORITHMS

Embedded Project J Component

Fall Semester 2020-21

Slot: C1 Professor Kauser Ahmed. P

The Travelling Salesman Problem

19BCE2249 19BCE2250

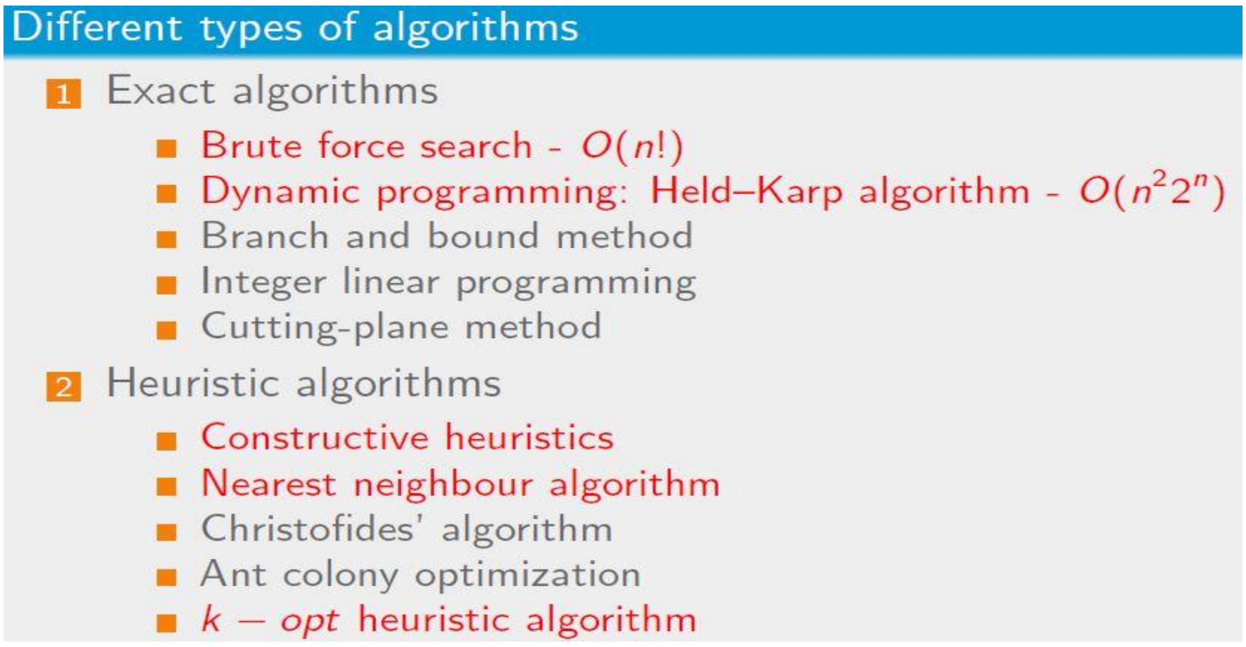
Siddharth Chatterjee Ishan Sagar Jogalekar

Abstract

The world needs a better way to travel; in particular it should be easy to plan an optimal route through multiple destinations. The principal goal of our project is to apply Travelling Salesman Problem algorithms (to be referred to as TSP from now onwards) to solve real world scenarios. In order to do so effectively, we will optimize current TSP algorithms to reduce computation time.

The TSP problem basically asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" In combinatorial optimization, a TSP is categorized as an NP hard problem, important in theoretical computer science and operations research.

There are 2 different types of algorithms with various methods in them to solve the TSP problem.



Different types of algorithms

1. Exact algorithms
   - Brute force search - $O(n!)$
   - Dynamic programming: Held–Karp algorithm - $O(n^2 2^n)$
   - Branch and bound method
   - Integer linear programming
   - Cutting-plane method
2. Heuristic algorithms
   - Constructive heuristics
   - Nearest neighbour algorithm
   - Christofides' algorithm
   - Ant colony optimization
   - $k - opt$ heuristic algorithm

Any TSP consists of determining a minimum distance circuit passing through each vertex once and only once. Such a circuit is known as a tour or Hamiltonian circuit (or cycle).

Let G = (V, A) be a graph where V is a set of n vertices. A is a set of arcs or edges, and let C: (Cij) be a cost or time travel matrix associated with A. Now, a TSP can be modelled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's length.

Often, the model is a complete graph (i.e. each pair of vertices is connected by an edge). If no path exists between two cities, adding an arbitrarily long edge will complete the graph without affecting the optimal tour.

Brief History

The TSP problem was first formulated in 1930s by Merrill M. Flood who was looking to solve a school bus routing problem. The earliest publication using the phrase "traveling salesman problem" was reported by Julia Robinson.

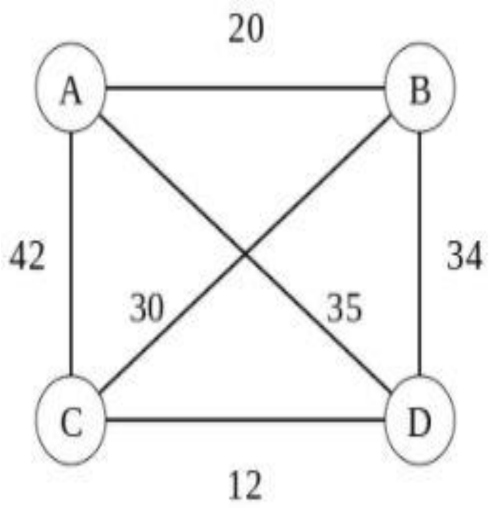
Richard M. Karp showed in 1972 that the Hamiltonian cycle problem was NP-complete, which implies the NP-hardness of TSP. This supplied a mathematical explanation for the apparent computational difficulty of finding optimal tours.

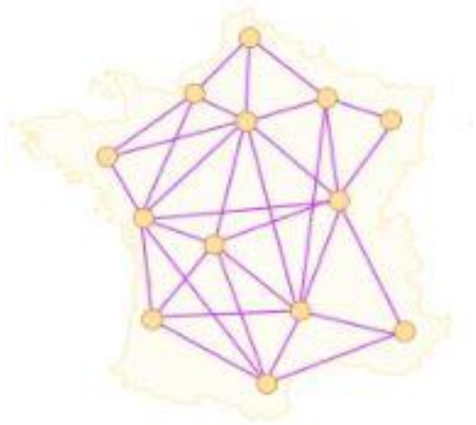
In 1976, Christofides and Serdyukov made a big advance in this direction: the Christofides-Serdyukov algorithm yields a solution that, in the worst case, is at most 1.5 times longer than the optimal solution. As the algorithm was so simple and quick, many hoped it would give way to a near optimal solution method. This remains the method with the best worst-case scenario.

In 2006, William J Cook computed an optimal tour through an 85,900-city instance given by a microchip layout problem, currently the largest solved TSPLIB instance.

Symmetric and Assymmetric

In the symmetric TSP, the distance between two cities is the same in each opposite direction, forming an undirected graph. This symmetry halves the number of possible solutions.





In the asymmetric TSP, paths may not exist in both directions or the distances might be different, forming a directed graph. Traffic collisions, one-way streets, and airfares for cities with different departure and arrival fees are examples of how this symmetry could break down.

Problem Statement

‘Surakshit’ has supplied a large number of RO devices to retailers and distributors over a certain geographical area. According to the service agreements, the company has to attend to service requests and to carry out mandatory service visits to each and every unit that is bought and installed. Due to stiff competition for supply and installation of RO Devices with other companies in the same market, an optimal route is to be constructed to minimize the cost of the services with a well-planned schedule to carry out services efficiently.

We will explore various algorithms which can prove to be optimum for the above situation without having significant disadvantages.

Project Plan

We will be devising ‘exact algorithms’, which work reasonably fast for such problem statements accounting a small geographical area where RO water filters need to be serviced.

We can try all permutations (ordered combinations) and see which one is cheapest (using brute-force search algorithm). The running time for this approach lies within a polynomial factor of O(n!), the factorial of the number of cities, so this solution becomes impractical even for only 20 cities. Therefore, this is something we will not go forward with.

Now, instead of brute-force algorithm; using dynamic programming (DP) approach the solution can be obtained in lesser time, though there is no polynomial time algorithm. One of the earliest applications of dynamic programming is the Held–Karp algorithm that solves the problem in time $O(n^2 2^n)$.

The project plan will also include some other approaches to compare the time complexity and determine the efficiency of each one of them:

- Various branch-and-bound algorithms, which can be used to process TSPs containing 40–60 cities.
- Progressive improvement algorithms which use techniques reminiscent of linear programming. Works well for up to 200 cities.
- Implementations of branch-and-bound and problem-specific cut generation (branch-and-cut);

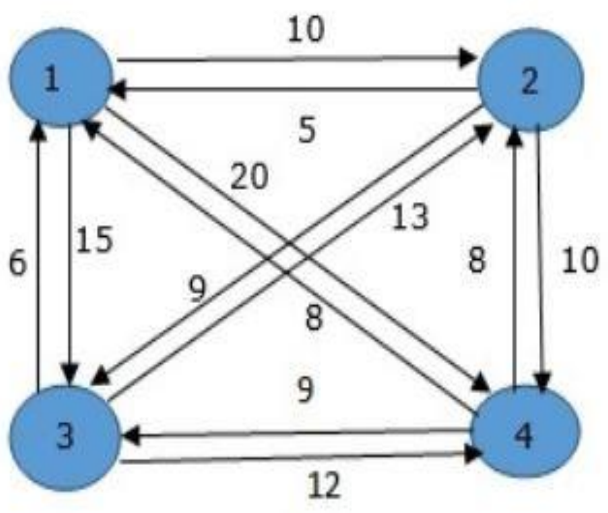
The prime mathematical deduction for our project plant will consider a subset of cities S Є {1, 2, 3, ... , n} that includes 1, and j Є S. Now let C(S, j) be the length of the shortest path visiting each city in S exactly once, starting at 1 and ending at j. When |S| > 1, we define C(S, 1) = ∝ since the path cannot start and end at 1.

Now, let us express **C(S, j)** in terms of smaller sub-problems. We need to start at ***1*** and end at **j**. We should select the next city in such a way that:

$$C(S,j) = min\, C(S-\{j\},i) + d(i,j)\ where\ i \in S\ and\ i \neq jc(S,j) = minC(s-\{j\},i) + d(i,j)\ where\ i \in S\ and\ i \neq j$$

Implementation Details

We will use a simple example in the beginning to illustrate the steps to solve the travelling salesman problem.



From the above graph, the following table is prepared:

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

For S = Φ:

Cost(3,Φ,1)=d(3,1)=6Cost(3,Φ,1)=d(3,1)=6

Similarly for 2 and 4 it will be 5 and 8 respectively.

For S = 1:

$$Cost(2,\{3\},1) = d[2,3] + Cost(3,\Phi,1) = 9 + 6 = 15 cost(2,\{3\},1) = d[2,3] + cost(3,\Phi,1) = 9 + 6 = 15$$

$$Cost(2,\{4\},1) = d[2,4] + Cost(4,\Phi,1) = 10 + 8 = 18 cost(2,\{4\},1) = d[2,4] + cost(4,\Phi,1) = 10 + 8 = 18$$

Similarly, there are others like **(3,{2},1) =18; (3,{4},1) = 20; (4,{3},1)=15; (4,{2},1)=13**

S = 2

$$Cost(2,\{3,4\},1) = \begin{cases} d[2,3] + Cost(3,\{4\},1) = 9 + 20 = 29 \\ d[2,4] + Cost(4,\{3\},1) = 10 + 15 = 25 = 25Cost(2,\{3,4\},1) \\ \{d[2,3] + cost(3,\{4\},1) = 9 + 20 = 29d[2,4] + Cost(4,\{3\},1) = 10 + 15 = 25 \end{cases}$$
$$= 25$$

$$Cost(3,\{2,4\},1) = \begin{cases} d[3,2] + Cost(2,\{4\},1) = 13 + 18 = 31 \\ d[3,4] + Cost(4,\{2\},1) = 12 + 13 = 25 = 25Cost(3,\{2,4\},1) \\ \{d[3,2] + cost(2,\{4\},1) = 13 + 18 = 31d[3,4] + Cost(4,\{2\},1) = 12 + 13 = 25 \end{cases}$$
$$= 25$$

$$Cost(4,\{2,3\},1) = \begin{cases} d[4,2] + Cost(2,\{3\},1) = 8 + 15 = 23 \\ d[4,3] + Cost(3,\{2\},1) = 9 + 18 = 27 = 23Cost(4,\{2,3\},1) \\ \{d[4,2] + cost(2,\{3\},1) = 8 + 15 = 23d[4,3] + Cost(3,\{2\},1) = 9 + 18 = 27 \end{cases}$$
$$= 23$$

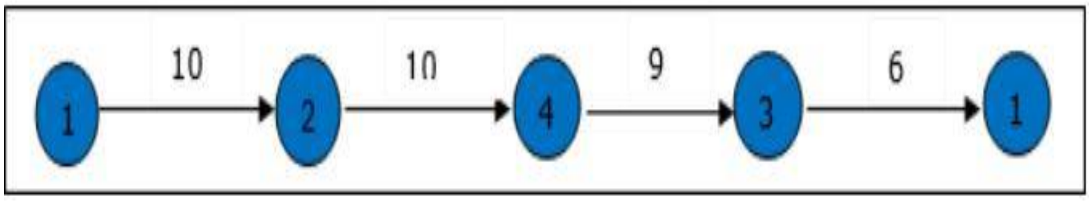
S = 3

$$Cost(1,\{2,3,4\},1) = \begin{cases} d[1,2] + Cost(2,\{3,4\},1) = 10 + 25 = 35 \\ d[1,3] + Cost(3,\{2,4\},1) = 15 + 25 = 40 \\ d[1,4] + Cost(4,\{2,3\},1) = 20 + 23 = 43 = 35cost(1,\{2,3,4\}),1) \\ d[1,2] + cost(2,\{3,4\},1) = 10 + 25 = 35 \\ d[1,3] + cost(3,\{2,4\},1) = 15 + 25 = 40 \\ d[1,4] + cost(4,\{2,3\},1) = 20 + 23 = 43 = 35 \end{cases}$$

Now, after implementing the above mathematical statements, we can deduce that the minimum cost path is 35.

Start from cost **{1, {2, 3, 4}, 1}**, we get the minimum value for **d [1, 2]**. When **s = 3**, select the path from 1 to 2 (cost is 10) then go backwards. When **s = 2**, we get the minimum value for **d [4, 2]**. Select the path from 2 to 4 (cost is 10) then go backwards.

When **s = 1**, we get the minimum value for **d [4, 3]**. Selecting path 4 to 3 (cost is 9), then we shall go to then go to **s = Φ** step. We get the minimum value for **d [3, 1]** (cost is 6).



References

1. https://sites.cs.ucsb.edu/~cappello/290b-2007-Spring/grades/project/brian/FinalProject/documents/FinalPaper.pdf
2. http://www.cs.rpi.edu/courses/spring00/dsa/projdir/project4/proj4.html
3. https://www.researchgate.net/publication/323750246_Dynamic_Simulated_Annealing_for_solving_the_Traveling_Salesman_Problem_with_Cooling_Enhancer_and_Modified_Acceptance_Probability
4. https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_travelling_salesman_problem.htm
5. Also special thanks to this article, which gave us an amazing insight into the infamous and evergrowing TSP problem https://www.wired.com/2013/01/traveling-salesman-problem/