# EE667

# VLSI CAD

***Project Report on***

***Travelling salesman problem using dynamic programming***

**Prashant Suryavanshi**

**10D070051**

# Table of Contents

# Introduction

What is travelling salesman problem? 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? It is an NP-hard problem.

In the theory of computational complexity, the decision version of the TSP (where, given a length *L*, the task is to decide whether the graph has any tour shorter than *L*) belongs to the class of NP-complete problems. Thus, it is likely that the worst-case running time for any algorithm for the TSP increases super polynomially (or perhaps exponentially) with the number of cities.

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. It is a minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once. 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.

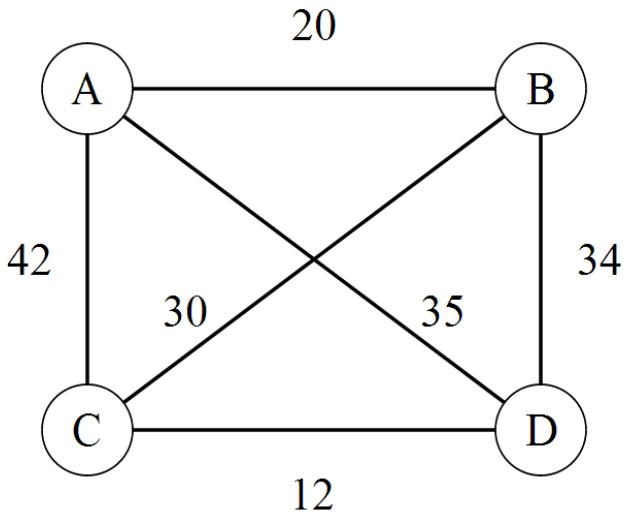



**Figure 1 : Undirected Graph**

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. This asymmetric TSP is the one on which I have worked on, also it is directed graph. Considering below sample problem to solve

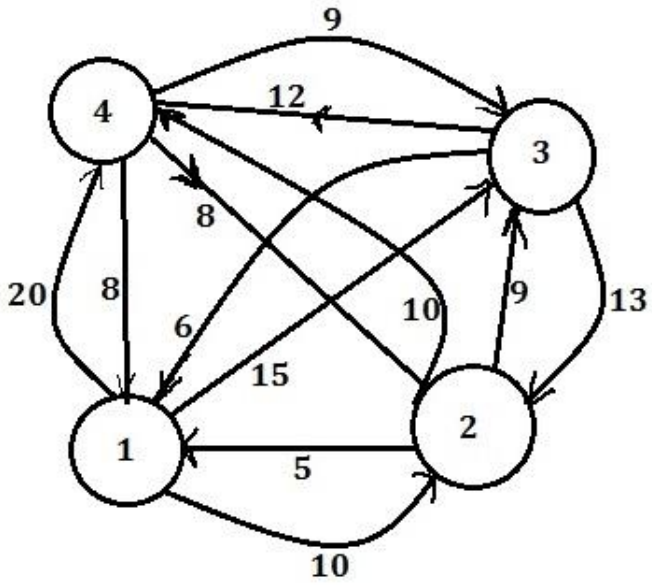



**Figure 2: Directed and asymmetric Graph**

# Different Solutions

TSP problem can be solved by using different kinds of algorithms such as

- Dynamic programming
- Branch and Bound Algorithm
- Linear Programming
- Branch and Cut
- Heuristics and approximation algorithm

# Dynamic Programming

It is one of the optimal methods used for solving TSP problem. Dynamic programming is a method for solving complex problems by breaking them down into simpler sub problems. It is applicable to problems exhibiting the properties of overlapping sub problems and optimal substructure. When applicable, the method takes far less time than naive methods that don't take advantage of the sub problem overlap (like depth-first search).

The idea behind dynamic programming is quite simple. In general, to solve a given problem, we need to solve different parts of the problem (sub problems), then combine the solutions of the sub problems to reach an overall solution. Often when using a more naive method, many of the sub problems are generated and solved many times. The dynamic programming approach seeks to solve each sub problem only once, thus reducing the number of computations: once the solution to a given sub problem has been computed, it is stored or memorized :the next time the same solution is needed, it is simply looked up. This approach is especially useful when the number of repeating sub problems grows exponentially as a function of the size of the input.

A dynamic programming algorithm will examine all possible ways to solve the problem and will pick the best solution. Therefore, we can roughly think of dynamic programming as an intelligent, brute-force method that enables us to go through all possible solutions to pick the best one. If the scope of the problem is such that going through all possible solutions is possible and fast enough, dynamic

programming guarantees finding the optimal solution. The alternatives are many, such as using a greedy algorithm, which picks the best possible choice "at any possible branch in the road". While a greedy algorithm does not guarantee the optimal solution, it is faster. Fortunately, some greedy algorithms (such as minimum spanning trees) are proven to lead to the optimal solution.

# Dynamic programming for solving TSP problem

As we have to come back to the same node from which we have been starting therefore if the numbers of nodes in the graph are n
Then number paths possible are (n-1)!  for one of the nodes
As n! =O ($n^n$) => order of n therefore difficult to execute for higher values of n

Now **Algorithm**
If starting from node 1 then
Each tour consists of an Edge<1,k>  for some k∈ V-{1}
And a path from vertex k to vertex 1 going through all vertex ∈ V-{1,k} exactly once. Where V: set of all vertices

Want to make the tour (one cycle) minimal in cost:
Defining a function g(i)(S)
g(i)(S) => Minimal cost of the path starting from ith node and going through all vertices in set S and terminating at node 1

Here i => node

S =>set of vertices to cover in the path

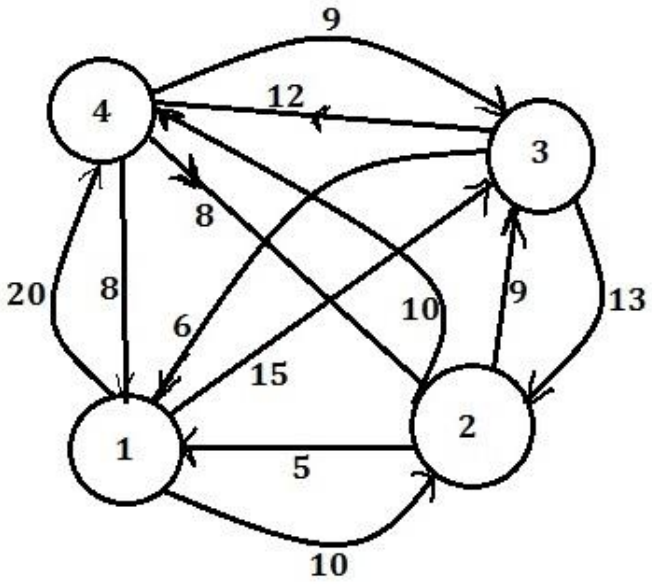
Therefore g(1)(V-{1})=> Minimal cost of the path starting from 1st node and going through all vertices other than 1 exactly once and finally terminating at node 1

=> g(1)(V-{1})=min($C_{1k} + g(k)(k, V - \{1, k\})$) where 1<k<n+1

As **g(i)(S)=min($C_{ij} + g(j)(j, S - \{j\})$)**

Thus minimal cost path can be found from above function g as we have to divide the graph into subparts and later merge the solution to get overall solution. This is how dynamic programming comes into effect.

Consider this sample problem




Matrix corresponding to the above graph

| | | | |
|---|---|---|---|
| 0 | 10 | 15 | 20 |
| 5 | 0 | 9 | 10 |
| 6 | 13 | 0 | 12 |
| 8 | 8 | 9 | 0 |

Suppose want to find minimal path starting from node 1

Stage 1: no vertices to cover i.e. S=Ø

Then g(1)(Ø)=$C_{11}$=0
g(2)(Ø)= $C_{21}$=5
g(3)(Ø)= $C_{31}$=6
g(4)(Ø)= $C_{41}$=8
where $C_{ij}$ is a cost function determining the cost from i to j node
g(2)( Ø) => starting from node 2 and no vertex to cover and ending at 1

Stage 2: Only one vertex to cover i.e. S=1

Then g(2)(3)= $C_{23} + g(3)(Ø)$=$C_{23} + C_{31}$=15
g(2)(4)= $C_{24} + g(4)(Ø)$= $C_{24} + C_{41}$=18
g(3)(2)= $C_{32} + g(2)(Ø)$= $C_{32} + C_{21}$=18
g(3)(4)= $C_{34} + g(4)(Ø)$= $C_{34} + C_{41}$=20
g(4)(2)= $C_{42} + g(2)(Ø)$= $C_{42} + C_{21}$=13
g(4)(3)= $C_{43} + g(3)(Ø)$= $C_{43} + C_{31}$=15

Stage 3: Only two vertices to cover i.e. S=2

Then g(2)(3,4)=min{[$C_{23} + g(3)(4)$], [$C_{24} + g(4)(3)$]} = 25
g(3)(2,4)=min{[$C_{32} + g(2)(4)$], [$C_{34} + g(4)(2)$]} = 25
g(4)(2,3)=min{[$C_{42} + g(2)(3)$], [$C_{43} + g(3)(2)$]} = 23

Stage 4: Final Path

g(1)(2,3,4)=min{ $C_{12} + g(2)(3,4), C_{13} + g(3)(2,4), C_{14} + g(4)(2,3)$} = 35

And the Path direction 1->2->4->3->1

Similarly solve as path can also start from node 2,3 and 4.
Then take the overall minimal solution.

Limitation: Code is written for total number of nodes =4

## Reference

http://en.wikipedia.org/wiki/Travelling_salesman_problem

http://en.wikipedia.org/wiki/Dynamic_programming