TRAVELLING SALESMAN PROBLEM

Project report submitted in partial fulfillment of the Requirements for the Award of the Degree of

BACHELOR OF TECHNOLOGY

IN

ENGINEERING PHYSICS

By

Aditya Singh, 2K19/EP/005

Arkajyoti Chakraborty, 2K19/EP/022

Under the Guidance of

Ms. Shivani Kumari

DEPARTMENT OF INFORMATION TECHNOLOGY

DELHI TECHNOLOGICAL UNIVERSITY

(Under Delhi Act 6 of 2009, Govt. of NCT of Delhi)

Bawana Rd, Delhi Technological University, Shahbad Daulatpur Village, Rohini, New Delhi, Delhi 110042

2021

DELHI TECHNOLOGICAL UNIVERSITY

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the project report entitled Travelling Salesman Problem being submitted by Aditya Singh 2K19/EP/005, and Arkajyoti Chakraborty 2K19/EP/022

in partial fulfillment for the award of the Degree of Bachelor of Technology in Engineering Physics to the Delhi Technological University, New Delhi is a record of bonafide work carried out under my guidance and supervision.

Ms. Shivani Kumari

DECLARATION

I hereby declare that the dissertation entitled **Travelling Salesman Problem** submitted for the B.Tech Degree is a record of original work carried out by me under the supervision of Ms. Shivani Kumari, and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Place : New Delhi

Date : 15/11/2021

Aditya Singh
2K19/EP/005,
Arkajyoti Chakraborty
2K19/EP/022

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our teacher Ms. Shivani Kumari, who gave us the golden opportunity to do this wonderful project on the topic Travelling Salesman Problem, which also helped us in doing a lot of Research and we came to know about so many new things. We are really thankful to them.

Secondly we would also like to thank some articles and research papers who helped us a lot in finalizing this project within the limited time frame.

Place : New Delhi

Date : 15/11/2021

Aditya Singh
2K19/EP/005,
Arkajyoti Chakraborty
2K19/EP/022

ABSTRACT

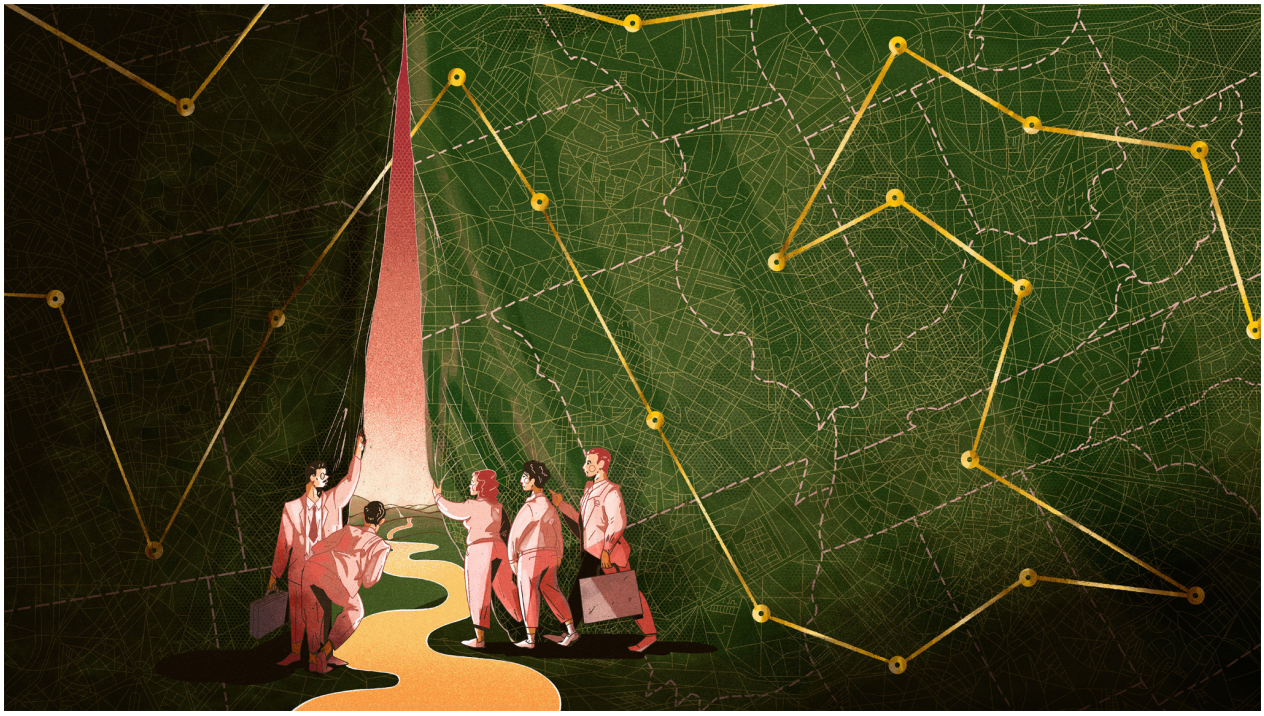
The traveling salesman problem is a permutation problem in which the goal is to find the shortest path between N different cities that the salesman takes is called the TOUR. In other words, the problem deals with finding a route covering all cities so that the total distance traveled is minimal.
No current algorithms are available which can solve these problems in polynomial time, that is, the number of steps grows as a polynomial according to the size of the input. The traveling-salesman problem involves a salesman who must make a tour of a number of cities using the shortest path available. For each number of cities n, the number of paths which must be explored is n!, causing this problem to grow exponentially rather than as a polynomial.

This paper gives a solution to find an optimum route for traveling salesman problem using Backtracking technique, in which cities are selected randomly as initial population. The new generations are then created repeatedly until the proper path is reached upon reaching the stopping criteria.

TABLE OF CONTENTS

LIST OF FIGURES

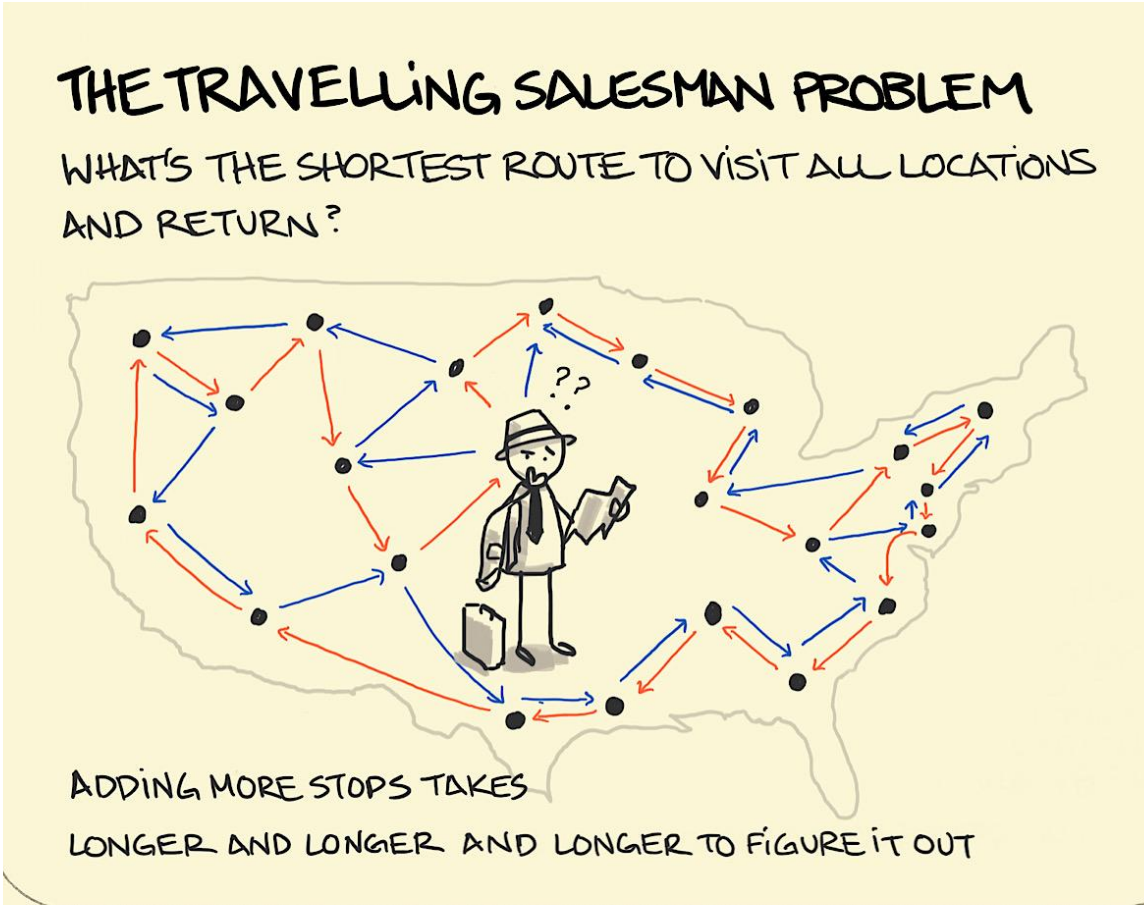
CHAPTER 1



Introduction

1.1 Overview

A traveling salesman wishes to go to a certain number of destinations in order to sell objects. He wants to travel to each destination exactly once and return home taking the shortest total route.

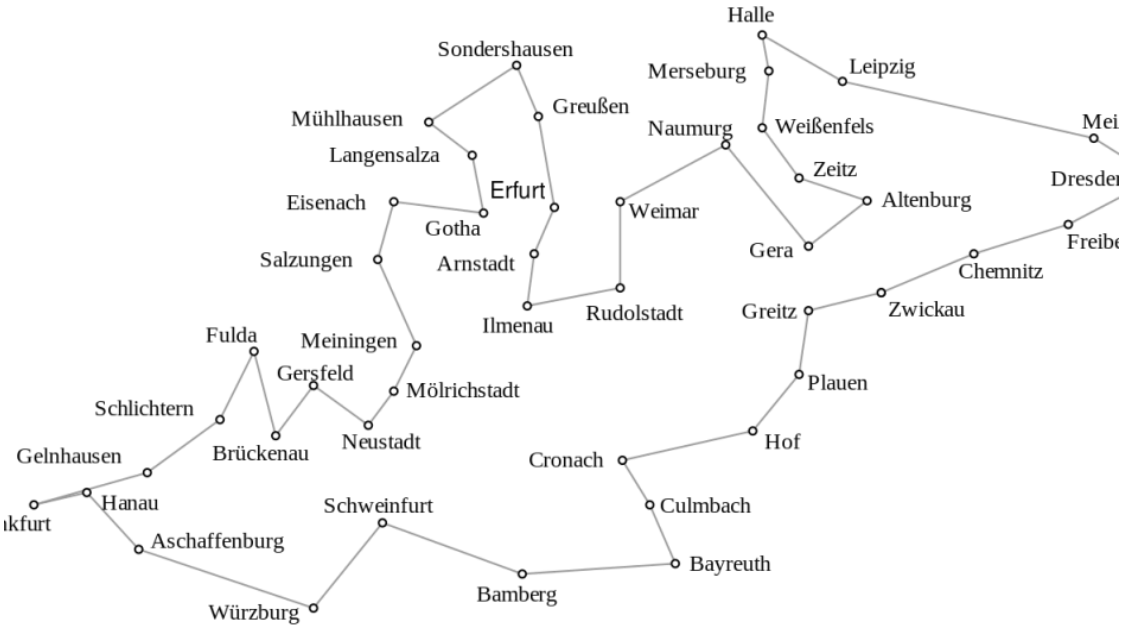


1.2 Problem Statement

Each voyage can be represented as a graph G = (V, E) where each destination, including his home, is a vertex, and if there is a direct route that connects two distinct destinations then there is an edge between those two vertices. The traveling salesman problem is solved if there exists a shortest route that visits each destination once and permits the salesman to return home. The traveling salesman problem can be divided into two types: the problems where there is a path between every pair of distinct vertices (no roadblocks), and the ones where there are not (with road blocks).

Despite being a travelling salesman, this problem interests those who want to optimize their routes, either by considering distance, cost, or time. If one has four people in their car to drop off at their respective homes, then one automatically tries to think about the shortest distance possible. In this case, distance is minimized. If one is traveling to different parts of the city using public transportation, then minimizing distance might not be the goal, but rather minimizing cost. In the first case, each vertex would be a person's home, and each edge would be the distance between homes. In the second case, each vertex would be a destination of the city and each edge would be the cost to get from one part of the city to the next.

1.3 History



Although the exact origins of the Traveling Salesman Problem are unclear, the first example of such a problem appeared in the German handbook Der Handlungsreisende - Von einem alten Commis - Voyageur for salesmen traveling through Germany and Switzerland in 1832 as explained in. This handbook was merely a guide, since it did not contain any mathematical language. People started to realize that the time one could save from creating optimal paths is not to be overlooked, and thus there is an advantage to figuring out how to create such optimal paths.

CHAPTER 2

Software Design And Analysis

Software analysis and design includes all activities, which help the transformation of requirement specification into implementation. Requirement specifications specify all functional and non-functional expectations from the software. These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing to do.

Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.

2.1 GCC - The GNU Compiler

The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran,, as well as libraries for these languages (libstdc++,...). GCC was originally written as the compiler for the GNU operating system. The GNU system was developed to be 100% free software, free in the sense that it respects the user's freedom.

Compile a CPP file to generate executable target file: g++ file_name command is used to compile and create an executable file a.out (default target name).

This compiles and links tsp.cpp to produce a default target executable file a.out in the present working directory. To run this program, type a.exe where ./ represents the present working directory and a.exe is the executable target file.

2.2 Sublime Text

Sublime Text is a sophisticated text editor for code, markup and prose It is a commercial source code editor. It natively supports many programming languages and markup languages. Users can expand its functionality with plugins, typically community-built and maintained under free-software licenses.

- Provides plenty of customization options via themes.
- Can handle massive files with ease.
- Comes with some of the most powerful search functions in the market.
- Includes a smart code completion feature that supports most languages with autocompletion.

2.3 Graph Theory

Before presenting algorithms and conditions for when a locally optimal solution exists, some basic graph theory definitions are needed.

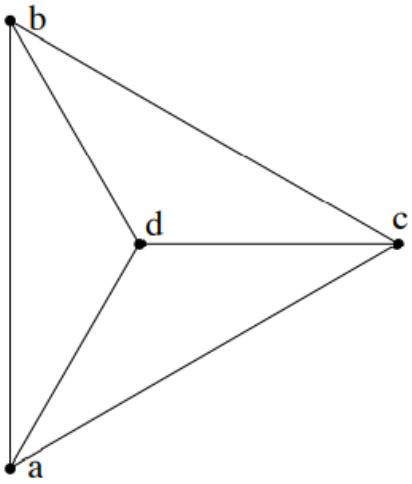


Figure 2.1: Tetrahedral Graph

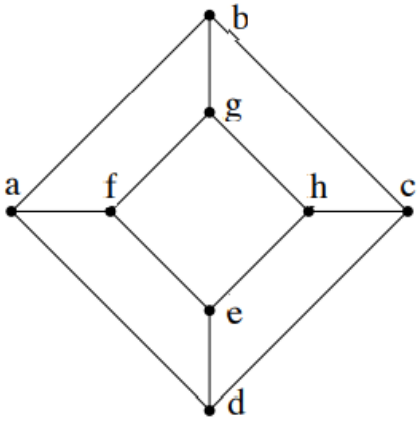


Figure 2.2: Cubical Graph

Simple Graph: A simple graph, G = (V,E), is a finite nonempty set V of objects called vertices (singular vertex) together with a possibly empty set E of 2-element subsets of V called edges.

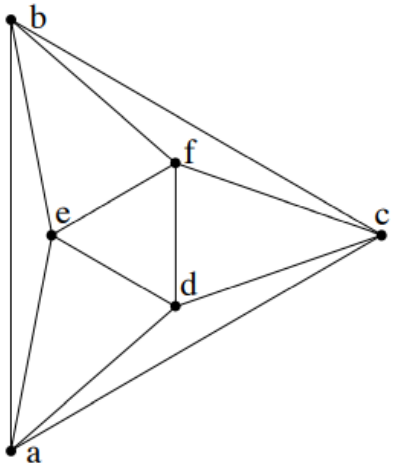


Figure 2.3: Octahedral Graph

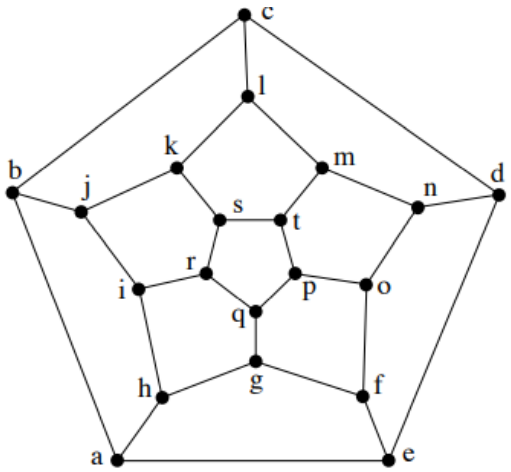


Figure 2.4: Dodecahedral Graph

Adjacent: Two vertices are adjacent if they are connected by an edge.

Incident: Vertex u and the edge uv are said to be incident with each other. Similarly, v and uv are incident.

Degree of a Vertex: The degree of a vertex v (denoted: deg(v)) is the number of vertices in G that are adjacent to v. For example, the vertex a in Figure 2.1 has degree three, while the vertex k in Figure 2.5 has degree 5. In a simple graph the maximum degree that any vertex can have is one less than the total number of vertices.

Minimum and Maximum Degree of a Graph: The maximum degree of a graph, G, is the greatest degree over all of the vertices in G. The minimum degree of a graph is the least degree over all of the vertices in G. The maximum degree is denoted $\Delta(G)$, while the minimum degree is denoted $\delta(G)$. For example, let H1 = Figure 2.5 and H2 = Figure 2.3. Then, $\Delta(H1) = 5$, and $\delta(H1) = 4$, and $\Delta(H2) = \delta(H2) = 4$.

Order and Size of a Graph: The number of vertices in a given graph G, denoted $|V(G)|$, is called the order of G and is denoted Ord(G), or $|G|$. The number of edges in G, denoted $|E(G)|$, is called the size of G.

Applications:

Graph is a data structure which is used extensively in our real-life.

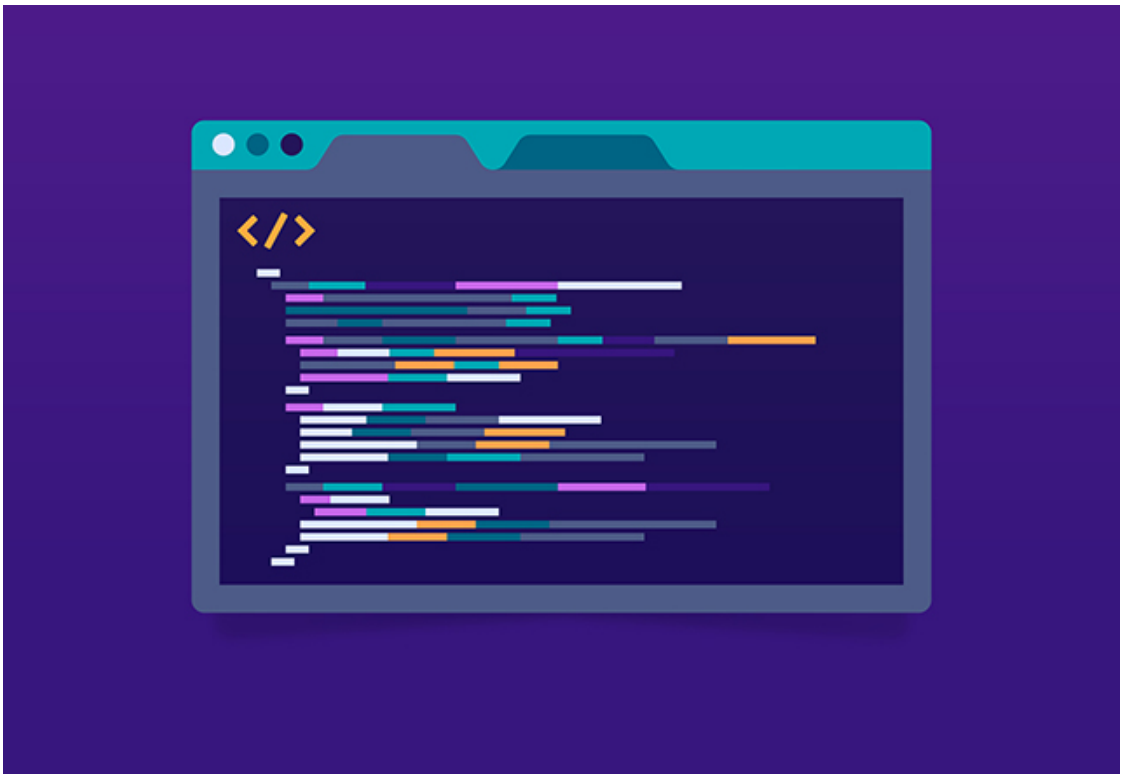
Social Network: Each user is represented as a node and all their activities, suggestions and friend list are represented as an edge between the nodes.

Google Maps: Various locations are represented as vertices or nodes and the roads are represented as edges and graph theory is used to find the shortest path between two nodes.

Recommendations on e-commerce websites: The “Recommendations for you” section on various e-commerce websites uses graph theory to recommend items of similar type to user’s choice.

Graph theory is also used to study molecules in chemistry and physics.

CHAPTER 3



Implementation in C++

3.1 Simple Approach

- Consider city 1 as the starting and ending point. Since the route is cyclic, we can consider any point as a starting point.
- Now, we will generate all possible permutations of cities which are (n-1)!.
- Find the cost of each permutation and keep track of the minimum cost permutation.
- Return the permutation with minimum cost.

```
int travelling Salesman Problem(int graph[][V], int s) {
  vector < int > vertex;
  for (int i = 0; i < V; i++)
    if (i != s)
      vertex.push_back(i);
  int min_path = INT_MAX;
  do {
    int current_pathweight = 0;
    int k = s;
    for (int i = 0; i < vertex.size(); i++) {
      current_pathweight += graph[k][vertex[i]];
      k = vertex[i];
    }
    current_pathweight += graph[k][s];
    min_path = min(min_path, current_pathweight);

  } while (
    next_permutation(vertex.begin(), vertex.end()));

  return min_path;
}
```

3.2 Dynamic Programming Approach

In this algorithm, we take a subset N of the required cities that need to be visited, the distance among the cities dist, and starting city s as inputs. Each city is identified by a unique city id which we say like 1,2,3,4,5……n
Here we use a dynamic approach to calculate the cost function Cost(). Using recursive calls, we calculate the cost function for each subset of the original problem.
We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on.
There are at most O(n2^n) subproblems, and each one takes linear time to solve. The total running time is, therefore, O(n^22^n). The time complexity is much less than O(n!) but still exponential. The space required is also exponential.

```
void mincost(int city) {
  int i, ncity; completed[city] = 1;
  cout << city + 1 << endl;
  ncity = least(city);
  if (ncity == 999) {
    ncity = 0; cout << city + 1 << endl;
    cost += ary[city][ncity];
    return;
  }
  mincost(ncity);
}

int least(int c) {
  int i, nc = 999, min = 999, kmin;
  for (i = 0; i < n; i++) {
    if ((ary[c][i] != 0) && (completed[i] == 0))
      if (ary[c][i] + ary[i][c] < min) {
        min = ary[i][0] + ary[c][i];
        kmin = ary[c][i];
        nc = i;
      }
  }
  if (min != 999) cost += kmin;
  return nc;
}
```

3.3 Greedy and Backtracking

- First of all, we have to create two primary data holders.
    - First of them is a list that can hold the indices of the cities in terms of the input matrix of distances between cities
    - And the Second one is the array which is our result
- Perform traversal on the given adjacency matrix tsp[][] for all the city and if the cost of reaching any city from the current city is less than the current cost the update the cost.
- Generate the minimum path cycle using the above step and return their minimum cost.

```
#include <bits/stdc++.h>
using namespace std;
#define V 4

void tsp(int graph[][V], vector<bool>& v, int currPos, int n,
                                    int count, int cost, int& ans) {

    cout << currPos << " " << cost << " " << graph[currPos][0] << endl;
    if (count == n && graph[currPos][0]) {
        ans = min(ans, cost + graph[currPos][0]);
        return;
    }

    // BACKTRACKING STEP
    for (int i = 0; i < n; i++) {
        if (!v[i] && graph[currPos][i]) {

            v[i] = true;
            tsp(graph, v, i, n, count + 1, cost + graph[currPos][i], ans);

            v[i] = false;
        }
    }
};
```

CHAPTER 4

Input/Output

4.1 Main Function

```
int main() {
    int n = 4;

    int graph[n][V];

    cout << "Enter the cost of Nodes :\n";

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < V; j++) {
            cin >> graph[i][j];
        }
    }
    cout << "\nThe adjacency Matrix will look like :\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < V; j++) {
            cout << graph[i][j] << " ";
        }
        cout << endl;
    }
    cout << "\n";
    vector<bool> v(n);
    for (int i = 0; i < n; i++)
        v[i] = false;

    v[0] = true;
    int ans = INT_MAX;

    tsp(graph, v, 0, n, 1, 0, ans);

    cout << ans << endl;

    return 0;
}
```

4.2 Results

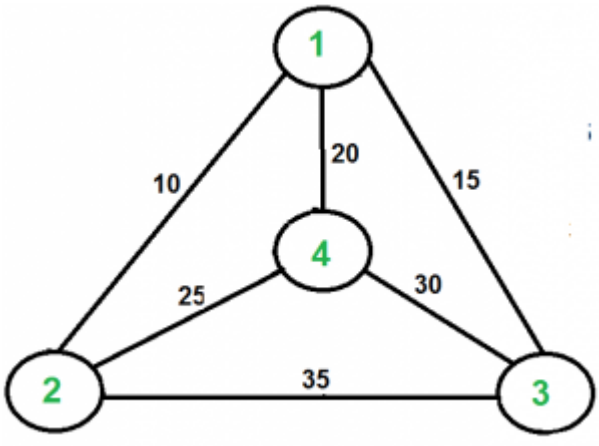
Input -

```
{ 0, 10, 15, 20 },
{ 10, 0, 35, 25 },
{ 15, 35, 0, 30 },
{ 20, 25, 30, 0 }
```

```
0 0 0
1 10 10
2 45 15
3 75 20
3 35 20
2 65 15
2 15 15
1 50 10
3 75 20
3 45 20
1 70 10
3 20 20
1 45 10
2 80 15
2 50 15
1 85 10
80
```

Output

```
 The adjacency Matrix will look like :
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0
```



A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80.

4.3 Complexity Analysis

The problem is a famous NP-hard problem. There is no polynomial-time known solution for this problem.

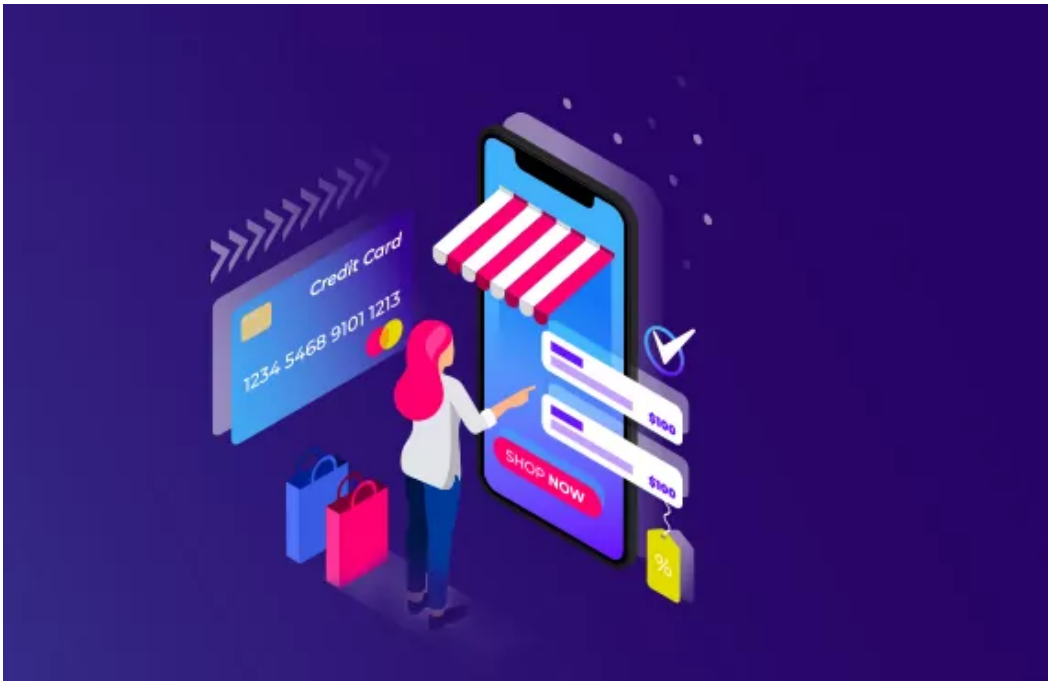
One would like to believe that the The Traveling Salesman Problem can always be solved by a computer. The maximum number of Hamiltonian Cycles in a given graph with three or more vertices is (n−1)! 2 . Hence, in theory a computer could consider each of these possible tours. The problem here is that as n gets large, (n−1)! 2 gets extremely large, as we can see in the table in Figure

n	(n-1)!/2
3	1
4	3
5	12
6	60
7	360
8	2,520
9	20,160
10	181,440
20	6.08226E+16
30	4.42088E+30
40	1.01989E+46

Brute Force

As these numbers grow larger, it becomes clear that a computer will not be able to solve such problems in a reasonable amount of time. Solving the Traveling Salesman Problem means searching for an algorithm that will make the number of computations acceptable. The Traveling Salesman Problem can be split into different, perhaps easier to solve, problems. In this chapter, we will explain these different problems, their complexity classes, and their relation to the full Traveling Salesman Problem.

CHAPTER 5



Future Scope

5.1 Conclusion

The Idea of Travel salesman problem has much application in different fields .To

Find best routes of travelling salesmen, we used three algorithms: simple, dynamic and backtracking. Hence we conclude that the Backtracking algorithm is more efficient than the other two. The time complexity of the other two is more compared to the Greedy and Backtracking approach, and it takes less memory space. The Simple Approach algorithm is more suitable for finding the minimum tour for a limited number of cities, because if we take 50 to 100 cites depth first search expands each node of a tree to reach the goal which is time consuming and memory waste.

5.2 Applications of the Traveling Salesman Problem

The TSP still finds applications in all verticals. Efficient solutions found through the TSP are being used in astronomy, computer science, and even vehicle routing.

The traveling salesman problem can also be used to find out how a laser should move when boring points into a printed circuit board.

Astronomers use the concepts of TSP to determine the movement of a telescope for the shortest distance between different stars in a constellation.

Other uses for TSP include internet planning, agriculture, microchip production, and computer-generated art.

BIBLIOGRAPHY

[1] Gerard Reinelt. The Traveling Salesman: Computational Solutions for TSP Applications. Springer-Verlag, 1994.

[2] D. B. Fogel, "An Evolutionary Approach to the Traveling Salesman Problem'', Biol. Cybern. 60,139- 144 (1988)

[3] Kylie Bryant ,Arthur Benjamin, Advisor, "Genetic Algorithms and the Traveling Salesman Problem'', Department of Mathematics, December 2000.
[4] Kovarik V,: Heuristic Algorithms for decision Support, Dissertation, Prague: VŠCHT, 2008

[5] Gros, I.: Quantitative Methods for managerial decision making, Prague: Grada publishing a.s., 2003, ISBN 80-247-0421-8 (In Czech)

[6] L. Guerra, T. Murino, E. Romano, The Location - Routing Problem: an innovative approach, 6th WSEAS International Conference on System Science and Simulation in Engineering, Venice, Italy, November 21-23, 2007

[7] Wiśniewski, M.: Methods for Managerial Decision Making, Prague: Grada publishing, 1996, ISBN 80- 7169-089-9 (In Czech)

[8] S. Dasgupta, C.H. Papadimitriou, and U.V. Vaziran: Chapter 6 - Dynamic Programming, Berkeley Quantum Computation Center (BQIC), University of California at Berkeley, Berkeley, CA 94720, U.S.A.

END